TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

Aplikace pro numerické řešení matematických úloh

Diplomová práce

Studijní program: N2612 – Elektrotechnika a informatika
Studijní obor: 1802T007 – Informační technologie

Autor práce: **Bc. Zdeněk Kybl**
Vedoucí práce: RNDr. Dana Černá, Ph.D.

Liberec 2015

TECHNICAL UNIVERSITY OF LIBEREC
Faculty of Mechatronics, Informatics
and Interdisciplinary Studies

Applications for the numerical solution of mathematical problems

Diploma thesis

Study programme: N2612 – Electrotechnology and informatics
Study branch: 1802T007 – Information technology

Author: **Bc. Zdeněk Kybl**
Supervisor: RNDr. Dana Černá, Ph.D.

Liberec 2015

TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky a mezioborových studií
Akademický rok: **2014/2015**

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. Zdeněk Kybl**
Osobní číslo:	**M12000207**
Studijní program:	**N2612 Elektrotechnika a informatika**
Studijní obor:	**Informační technologie**
Název tématu:	**Aplikace pro numerické řešení matematických úloh**
Zadávající katedra:	**Ústav nových technologií a aplikované informatiky**

Zásady pro vypracování:

1. Nastudujte vybrané numerické metody z oblasti aproximace funkce, metody pro řešení nelineárních rovnic, soustavy lineárních rovnic, metody pro výpočet vlastních čísel a vlastních vektorů reálných symetrických matic a metody pro nemurický výpočet určitého integrálu a derivace.
2. Navrhněte a implementujte aplikaci v MS VS C# pro numerické řešení uživatelem zadaných matematických úloh včetně možnosti zobrazování mezivýsledků, vykreslování grafů a exportu do vhodných formátů.
3. Vytvořte sadu cvičných úloh využívajících aplikaci.
4. Vytvořte dokumentaci vytvořené aplikace a použitých numerických metod.
5. Vytvořte webovou stránku pro distribuci aplikace.

Rozsah grafických prací:	**dle potřeby**
Rozsah pracovní zprávy:	**70 stran**
Forma zpracování diplomové práce:	**tištěná/elektronická**

Seznam odborné literatury:

[1] RALSTON, Anthony. Základy numerické matematiky. Praha: Academia, 1978, 635 s.
[2] DĚMIDOVIČ, B. P. a I. A. MARON. Základy numerické matematiky. Praha: SNTL, 1966, 721 s.
[3] VITÁSEK, Emil. Numerické metody. Praha: SNTL, 1987, 512 s.
[4] PRESS, W. Numerical recipes in C the art of scientific computing. Cambridge: University Press, 1996, 990 s. ISBN 05-214-3108-5.
[5] GRIFFITHS, Ian; ADAMS, Matthew; LIBERTY, Jesse. Programming C# 4.0. USA: O, 2010. 856 s. ISBN 978-0-596-15983-2.
[6] SHARP, John. Microsoft Visual C# 2010: krok za krokem. Brno: Computer Press, 2010, 696 s. ISBN 978-80-251-3147-3.

Vedoucí diplomové práce:	**RNDr. Dana Černá, Ph.D.** Katedra matematiky a didaktiky matematiky
Konzultant diplomové práce:	**Ing. Igor Kopetschke** Ústav nových technologií a aplikované informatiky
Datum zadání diplomové práce:	**20. října 2014**
Termín odevzdání diplomové práce:	**15. května 2015**

prof. Ing. Václav Kopecký, CSc.
děkan

Technická univerzita v Liberci - 14 -
L.S.

prof. Dr. Ing. Jiří Maryška, CSc.
vedoucí ústavu

V Liberci dne 20. října 2014

Prohlášení

Byl jsem seznámen s tím, že na mou diplomovou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci (TUL) nezasahuje do mých autorských práv užitím mé diplomové práce pro vnitřní potřebu TUL.

Užiji-li diplomovou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti TUL; v tomto případě má TUL právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Diplomovou práci jsem vypracoval samostatně s použitím uvedené literatury a na základě konzultací s vedoucím mé diplomové práce a konzultantem.

Současně čestně prohlašuji, že tištěná verze práce se shoduje s elektronickou verzí, vloženou do IS STAG.

Datum: 15.5.2015

Podpis:

Abstrakt

Diplomová práce si klade za cíl vytvoření rešerše vybraných numerických metod a zhotovení aplikace, jež slouží zejména jako didaktická pomůcka studentům při studiu problematiky numerické matematiky. Teoretická část je rozdělena do šesti kapitol, přičemž v každé kapitole jsou charakterizovány hlavní principy numerických metod jednoho odvětví numerické matematiky. Práce postupně seznamuje s algoritmy zabývajícími se aproximací a interpolací funkce, numerickou integrací a derivací, řešením nelineárních rovnic, metodami pro řešení soustav lineárních rovnic a s algoritmy sloužícími pro výpočet vlastních čísel a vlastních vektorů reálných symetrických matic. V praktické části je nejprve představena aplikace implementovaná v jazyce *C#* z pohledu jejího návrhu, kdy jsou blíže představeny všechny vrstvy aplikace. Později je ilustrována interakce jednotlivých vrstev aplikace a dění v pozadí aplikace při jejím užívání uživatelem. Pro zajištění dostatečné didaktické úrovně využívá aplikace nástrojů, pomocí nichž dochází k zobrazení nejen získaného řešení, ale i postupu, který vedl k jeho dosažení. V případě aproximace a interpolace funkce, řešení nelineárních rovnic a numerické integrace je didaktická úroveň umocněna grafickou interpretaci úlohy a jejího řešení. Aplikace dále obsahuje sadu cvičných úloh a podporuje exporty do dalších formátů. Pro distribuci aplikace byly zhotoveny webové stránky.

Klíčová slova

Numerická matematika, aproximace funkcí, numerická integrace, numerická derivace, nelineární rovnice, soustavy lineárních rovnic, vlastní čísla a vektory symetrických matic, didaktická aplikace, postup výpočtu, cvičné úlohy, export

Abstract

The aim of the thesis is a review of numerical methods and manufacturing applications, which are mainly used as a didactic aid for students studying the problems of numerical mathematics. The theoretical part is divided into six chapters, where each chapter outlines the main principles of numerical methods, one branch of numerical mathematics. Work gradually introduces algorithms dealing with approximations and interpolation functions, numerical integration and differentiation, solution of nonlinear equations, methods for solving systems of linear equations and algorithms serving for calculating eigenvalues and eigenvectors of real symmetric matrices. The practical part is firstly introduced by application implemented in language $C\#$ in terms of its design, which introduces each application layer in more details. Then the interaction of the layers of the application during its use is illustrated. To ensure sufficient levels of educational uses the application uses tools for viewing not only the results, but also the process leading to their achievement. In the case of approximation and interpolation functions, solving nonlinear equations and numerical integration a didactic level is enhanced by graphical interpretation of the examples and its solutions. The application also includes a set of training tasks and supports exports to other formats. The websites were made for distribution of the application.

Keywords

Numerical methods, approximation of function, numerical integration, numerical differentiation, nonlinear equations, linear equations, eigenvalues and eigenvectors of symmetric matrices, didactic applications, calculation procedure, practice tasks, export

Poděkování

Rád bych poděkoval vedoucí mé práce RNDr. Daně Černé, Ph.D. za odborné vedení, cenné rady, trpělivost a ochotu, kterou mi v průběhu zpracování diplomové práce věnovala.

Obsah

Seznam zkratek

$\mathbb{N}$	Množina přirozených čísel
$\mathbb{R}$	Množina reálných čísel
$\mathbb{C}$	Množina komplexních čísel
$f(x)$	Hodnota funkce f v bodě x
$f'(x)$	Hodnota první derivace funkce f v bodě x
$\int_a^b f(x)dx$	Integrál funkce f na intervalu $\langle a, b\rangle$
$\lim_{x\to 0} f$	Limita funkce f pro x jdoucí k nule
δ_{ij}	Kroneckerovo delta
$\forall$	Pro všechna
$\exists$	Existuje
$\in$	Je prvkem
$C^{n+1}(I)$	Prostor $(n+1)$ spojitých derivací na intervalu I
Π_n	Prostor polynomů stupně nejvýše n
$\tilde{\Pi}_n$	Prostor normovaných polynomů stupně nejvýše n
$\mathbf{A}^T$	Transponovaná matice k matici $\mathbf{A}$
$\mathbf{A}^{-1}$	Inverzní matice k matici $\mathbf{A}$
$\mathbf{A}^+$	Pseudoinverzní matice k matici $\mathbf{A}$
$h(\mathbf{A})$	hodnost matice $\mathbf{A}$
$\mathbf{I}$	Jednotková matice
$det(\mathbf{A})$	Determinant matice $\mathbf{A}$
λ	Vlastní číslo matice
$\langle a, b\rangle$	Uzavřený interval
(a, b)	Otevřený interval
$\rho(\mathbf{A})$	Spektrální poloměr matice $\mathbf{A}$
σ_i	i-té singulární číslo
$\|\mathbf{x}\|$	Euklidovská norma vektoru $\mathbf{x}$
$\|a\|$	Absolutní hodnota a
AJAX	Asynchronous JavaScript and XML
BMP	Bit Mapped Picture
GIF	Graphic Interchange Format
GUI	Graphic User Interface
HTML	HyperText Markup Language
JPG	Joint Photographic Experts Group
LIFO	Last In First Out
MathML	Mathematical Markup Language
MSIE	Microsoft Internet Explorer
PDF	Portable Document Format
PNG	Portable Network Graphics
RPN	Reverzní polská notace
TIFF	Tag Image File Format
XML	Extensible Markup Language
XSD	XML Schema

Seznam obrázků

Seznam tabulek

Úvod

Skoro každý člověk moderního světa denně využívá technických pomůcek či vymožeností, aniž by si uvědomoval, že za jejich objevením stojí velice často různá odvětví matematiky. Při vývoji těchto moderních výdobytků často narážíme na velice složité matematické modely a metody. V předpočítačové éře bylo naprosto nepředstavitelné provádět velké množství početních operací, a proto se odborníci aplikované matematiky snažili nalézt řešení analytickým způsobem, pomocí něhož došlo k redukci počtu prováděných operací. Ovšem získání tohoto přesného řešení je mnohdy nemožné, či velice složité a pracné. Naštěstí se ukazuje, že pro reálné využití nám ve většině případů postačí pouze řešení přibližné. A v tuto chvíli přichází na řadu numerická matematika.

Numerická matematika zaznamenala prudkého rozmachu až s rozvojem počítačů, kdy došlo k výraznému zlevnění početní operace. Proto mohly přejít do popředí výpočty často cyklického charakteru, v nichž můžeme uvažovat dříve nemyslitelné počty operací.

I přes nesporné využití numerické matematiky a matematiky obecně při řešení reálných problémů (jmenujme například odvětví strojírenství, teorie obvodů či stavitelství) není o studium této problematiku příliš velký zájem. Z tohoto důvodu si předkládaná diplomová práce klade nejprve za úkol seznámit čtenáře s vybranými numerickými metodami z oblasti aproximace funkce, numerického výpočtu určitého integrálu a derivace, s postupy pro řešení nelineárních rovnic či metodami pro řešení soustav lineárních rovnic a v neposlední řadě také s algoritmy pro výpočet vlastních čísel a vlastních vektorů reálných symetrických matic.

Na základě této teoretické stati si diplomová práce klade za cíl vytvoření didaktické aplikace, která na rozdíl od již existujících matematických programů, bude kromě samotného řešení uživateli interpretovat i postup výpočtu, pomocí něhož bylo řešení dosaženo. Implementovaná aplikace by měla dále obsahovat soubor cvičných úloh včetně podpory pro vykreslování grafů a exportů dat. Pro snazší distribuci aplikace případnému čtenáři bude v rámci řešení diplomové práce zhotovena webová stránka, která bude obsahovat zhotovenou aplikaci.

Předkládaná diplomová práce by tedy měla sloužit zejména jako studijní opora a současně jako výuková pomůcka pro zájemce zabývající se problematikou numerické matematiky.

1 Aproximace funkce

V první kapitole se budeme zabývat metodami pro interpolaci a aproximaci funkce. Princip aproximace funkce spočívá v nahrazení jisté funkce f jinou funkcí ϕ, která je v jistém smyslu původní funkci podobná.

Aproximující funkce ϕ by měla být co možná nejjednodušší, a zároveň by měla být zadaným bodům $f(x_i)$ pro $i = 0, \ldots, n$ co nejblíže.

Využití aproximací funkce je poměrně různorodé. Pokud například chceme na počítači vypočítat funkční hodnotu jisté funkce, tak výpočet této hodnoty se často děje právě pomocí aproximací funkcí, kdy je vstupní funkce f nahrazena jistým polynomem P, a to zejména z důvodu snadné a hlavně rychlé práce s mnohočleny. Polynomy jsou totiž poměrně snadno vyčíslitelné, jejich derivace i integrály jsme schopni snadno a rychle vypočítat.

Další oblastí, které se budeme věnovat později a kde můžeme vidět využití aproximací funkce, je výpočet integrálu, kdy opět nahrazujeme vstupní funkci f jistým polynomem P.

Nyní vyvstává otázka, jak nalézt funkci, která bude aproximovat původní funkci. Existuje několik typů metod pro její určení. Prvním typem, se kterým se budeme blíže seznamovat, je aproximace interpolačním polynomem.

1.1 Aproximace interpolačním polynomem

O interpolaci mluvíme tehdy, je-li úkolem stanovit hodnotu funkce f v bodech ležících mezi dvěma tabulkovými body.

Snažíme se nalézt funkci, která v bodech $x_0, x_1, x_2, \ldots, x_n$ nabývá hodnoty $f(x_0)$, $f(x_1), f(x_2), \ldots, f(x_n)$. Body $x_0, x_1, x_2, \ldots, x_n$ nazýváme uzlové body.

1.1.1 Lagrangeův interpolační polynom

Lagrangeova interpolace je aproximace polynomem L_n , pro který platí, že splňuje *základní úlohu interpolace.*

Definice 1.1.1. *Uvažujme $n + 1$ bodů, které jsou navzájem různé. Hledáme polynom L_n stupně nejvýše n takový, že platí $f(x_i) = L_n(x_i)$, $i = 0, \ldots, n$. Tento problém budeme označovat jako základní úlohu interpolace. Polynom L_n se nazývá Lagrangeův interpolační polynom.*

Věta 1.1.1. *Mějme dány body* $[x_i, f(x_i)]$, $i = 0, \ldots, n$. *Pak existuje právě jeden interpolační polynom* L_n *stupně nejvýše* n *takový, že* $L_n(x_i) = f(x_i)$, $i = 0, \ldots, n$.

Důkaz. Předpokládejme, že existují dva polynomy L_n a R_n stupně nejvýše n takové, že $L_n(x_i) = R_n(x_i) = f(x_i)$ pro $i = 0, \ldots, n$. Ukážeme, že jsou tyto dva polynomy shodné.

Položme $Q_n = L_n - R_n$. Potom $Q_n(x_i) = L_n(x_i) - R_n(x_i) = 0$. Polynom Q_n je polynom stupně n, který má $n + 1$ nulových bodů. Podle základní věty algebry je tento polynom identicky roven nule, a tedy $L_n \equiv R_n$. □

Hledáme polynom stupně nejvýše n , který splňuje podmínku interpolace. Máme tedy $n + 1$ bodů, kterými musí graf hledaného polynomu procházet. Tuto úlohu můžeme řešit jako soustavu lineárních rovnic, kdy bychom zjistili hodnoty koeficientů hledaného polynomu (pomocí tzv. *Vandermondovy matice*). Nicméně tento způsob řešení se příliš nevyužívá a lze se mu vyhnout pomocí Lagrangeova interpolačního polynomu.

Věta 1.1.2. *Lagrangeův interpolační polynom lze vyjádřit ve tvaru*

$$L_n(x) = \sum_{i=0}^{n} f(x_i) g_i(x), \tag{1.1}$$

kde $g_i(x) = \prod_{j=0, j\neq i} \frac{x - x_j}{x_i - x_j}$.

Věta 1.1.3. *Nechť* $f \in C^{n+1}(I)$, *kde* I *je nejmenší interval obsahující* $x_0, x_1, x_2, \ldots, x_{n-1}, x_n, x^*$ *a* $x_0, x_1, x_2, \ldots, x_n$ *jsou navzájem různé uzly.*

Pak $\forall\, x^* \in I\ \exists\, \xi \in I$, *pro které platí:*

$$f(x^*) - L_n(x^*) = f^{(n+1)}(\xi) \frac{\omega_{n+1}(x^*)}{(n+1)!}, \tag{1.2}$$

kde $\omega_{n+1}(x) = (x - x_0)(x - x_1) \ldots (x - x_n)$.

Důkaz. Předpokládejme, že $x_i = x^*$. Potom

$$f(x_i) - L_n(x_i) = f^{n+1}(\xi) \frac{\omega_{n+1}(x_i)}{(n+1)!}. \tag{1.3}$$

Z vlastnosti interpolace plyne

$$f(x_i) - L_n(x_i) = 0. \tag{1.4}$$

Pokud $x_i \neq x^*$, potom definujme funkci

$$F(x) = f(x) - L_n(x) - t\omega_{n+1}(x), \tag{1.5}$$

kde $x \in I, t \in \mathbb{R}$.

Funkce $F(x)$ má $n+1$ nulových bodů (uzlové body x_i). Hledáme takové t, aby byla splněna rovnost

$$F(x^*) = 0. \tag{1.6}$$

To splňuje

$$t = \frac{f(x^*) - L_n(x^*)}{\omega_{n+1}(x^*)}. \tag{1.7}$$

Funkce F má tedy $n+2$ nulových bodů. Z Rolleovy věty plyne, že F' má alespoň $n+1$ nulových bodů. F'' má alespoň n nulových bodů. $F^{(n+1)}$ má alespoň jeden nulový bod ξ,

$$F^{(n+1)}(\xi) = 0. \tag{1.8}$$

Protože $L_n^{(n+1)} \equiv 0$, dostaneme

$$F^{(n+1)}(\xi) = f^{(n+1)}(\xi) - 0 - t(n+1)!. \tag{1.9}$$

Dosadíme za proměnnou t

$$0 = F^{(n+1)}(\xi) = f^{(n+1)}(\xi) - \frac{f(x^*) - L_n(x^*)}{\omega_{n+1}(x^*)}(n+1)! \tag{1.10}$$

a vyjádříme výslednou chybu

$$f(x^*) - L_n(x^*) = f^{(n+1)}(\xi)\frac{\omega_{n+1}(x^*)}{(n+1)!}. \tag{1.11}$$

□

Více informací nalezneme například v [1] nebo [2]

1.1.2 Newtonův interpolační polynom

Tento polynom je pro $n+1$ uzlových bodů interpolačním polynomem stupně nejvýše n. Bude se tedy jednat o nový způsob zápisu Lagrangeova interpolačního polynomu. Tento polynom budeme hledat ve tvaru

$$\begin{aligned} N_n(x) &= a_0 + a_1(x - x_0) + a_2(x - x_0)(x - x_1) + \dots \\ &+ a_n(x - x_0)(x - x_1)\dots(x - x_{n-1}). \end{aligned} \tag{1.12}$$

Newtonův interpolační polynom tedy musí vyhovovat podmínce interpolace

$$N_n(x_i) = f(x_i), i = 0, 1, \dots, n. \tag{1.13}$$

Koeficienty $a_i, i = 0, \dots, n$, vyjádříme pomocí poměrných diferencí.

Definice 1.1.2. *Poměrná diference nultého řádu je definována:*

$$f[x_i] = f(x_i), i = 0, \dots, n. \tag{1.14}$$

Poměrná diference prvního řádu je definována:

$$f[x_i, x_{i+1}] = \frac{f[x_{i+1}] - f[x_i]}{x_{i+1} - x_i}, i = 0, \ldots, n-1. \tag{1.15}$$

Poměrná diference k-tého řádu je definována rekurentně:

$$f[x_i, x_{i+1}, \ldots, x_{i+k}] = \frac{f[x_{i+1}, \ldots, x_{i+k}] - f[x_i, x_{i+1}, \ldots, x_{i+k-1}]}{x_{i+k} - x_i}. \tag{1.16}$$

Newtonův interpolační polynom můžeme zapsat pomocí poměrných diferencí:

$$\begin{aligned} N_n(x) = f[x_0] + f[x_0, x_1](x - x_0) + f[x_0, x_1, x_2](x - x_0)(x - x_1) + \ldots \\ + \ldots + f[x_0, x_1, \ldots, x_n](x - x_0)(x - x_1)\ldots(x - x_{n-1}). \end{aligned} \tag{1.17}$$

Newtonova interpolace má oproti Lagrangeově interpolaci podstatnou výhodu, která spočívá v tom, že je výpočetně méně náročné přidat další bod s jeho funkční hodnotou, protože některé výpočty zůstanou beze změny (například předchozí koeficienty a_k se nezmění). Bližší informace jsou k nalezení například v [1].

1.2 Interpolace spline funkcemi

Pro intervaly větší délky je použití interpolačního polynomu nízkého stupně mnohdy nepřesné a použití interpolačního polynomu vyšších stupňů nevhodné vzhledem k vlastnosti, kdy interpolační polynom na krajích intervalů nepříjemně osciluje.

Vhodnější je využití interpolačních spline funkcí, které jsou po částech polynomy. Princip spline interpolace spočívá v rozdělení intervalu na podintervaly. Na každém z těchto podintervalů poté budeme konstruovat obecně jiný polynom.

Mezi nejčastěji využívané spliny patří lineární a kubický [3].

1.2.1 Lineární interpolační spline

Definice 1.2.1. *(Lineární interpolační spline) Lineárním splinem nazýváme funkci $\phi(x)$, která je spojitá na intervalu $\langle x_0, x_n \rangle$ a na každém podintervalu $\langle x_i, x_{i+1} \rangle$, $i = 0, \ldots, n-1$ je polynomem prvního stupně.*

Lineární interpolační spline, který prochází uzlovými body, tj. $\phi(x_i) = f(x_i)$, na podintervalu $\langle x_i, x_{i+1} \rangle, i = 0, \ldots, n-1$ můžeme zkonstruovat následujícím způsobem

$$\phi_i(x) = f(x_i) + \frac{f(x_{i+1}) - f(x_i)}{h_i}(x - x_i) \ , \tag{1.18}$$

kde $h_i = x_{i+1} - x_i$.

Grafem tohoto spline je lomená čára. Více informací lze získat například v [4].

1.2.2 Kvadratický interpolační spline

Definice 1.2.2. *(Kvadratický interpolační spline) Kvadratickým splinem nazýváme funkci $\phi(x)$, která je spojitá na intervalu $\langle x_0,x_n\rangle$, přičemž na každém podintervalu $\langle x_i, x_{i+1}\rangle, i = 0,\dots,n-1$, je polynomem druhého stupně. Tyto polynomy na sebe v uzlových bodech hladce navazují – mají spojitou první derivaci.*

Kvadratický interpolační spline procházející uzlovými body, tj. $\phi(x_i) = f(x_i)$, na podintervalu $\langle x_i, x_{i+1}\rangle$, $i = 0,\dots,n-1$, můžeme zkonstruovat následujícím způsobem

$$\phi_i(x) = \frac{d_{i+1} - d_i}{2(x_{i+1} - x_i)}(x - x_i)^2 + d_i(x - x_i) + f(x_i), \tag{1.19}$$

kde $d_{i+1} = 2\frac{f(x_{i+1})-f(x_i)}{x_{i+1}-x_i}, i = 0, 1, \dots, n-1$.

Hodnoty d_i vycházejí z podmínky první spojité derivace ve vnitřních bodech kvadratického splinu. Jednotlivé spline na intervalech tedy ve výsledku tvoří hladkou křivku.

Jelikož se budeme zabývat přirozeným kvadratickým splinem, klademe $d_1 = 0$. Více informací týkající se konstrukce kvadratického interpolačního splinu můžeme nalézt v [5].

1.2.3 Kubický interpolační spline

Interpolace pomocí kubických splinů je nejčastěji využívanou spline interpolací, protože podle [3] se ukazuje, že právě tato po částech polynomiální interpolace dosahuje nejlepších výsledků.

Definice 1.2.3. *Kubickým splinem nazveme funkci $\phi(x)$, která má na intervalu $\langle x_0,x_n\rangle$ dvě spojité derivace a na každém podintervalu $\langle x_i, x_{i+1}\rangle, i = 0,\dots,n-1$, je polynomem třetího stupně.*

Konstrukce přirozeného kubického spline

Mějme v uzlových bodech x_i dány funkční hodnoty $f(x_i), i = 0,\dots,n$, funkce f. Naším úkolem je sestrojit kubický interpolační polynom, který splňuje základní úlohu interpolace.

Z definice kubického polynomu $\phi(x) = ax^3 + bx^2 + cx + d$ plyne, že kubický polynom je určen čtyřmi koeficienty. Máme-li $n+1$ bodů, budeme mít n intervalů. Z toho vyplývá, že $\phi(x)$ je na intervalu $\langle x_0, x_n\rangle$ určena $4n$ parametry. Podmínky spojitosti $\phi(x), \phi'(x)$ *a* $\phi''(x)$ ve vnitřních bodech intervalu $\langle x_0, x_n\rangle$ dávají dalších $3n-3$ podmínek. Interpolační podmínky nám dají $n+1$ podmínek. Celkem tedy máme $4n-2$ podmínek pro $4n$ neznámých. Protože konstruujeme přirozený kubický spline, zbylé dvě podmínky doplníme tak, že položíme druhé derivace v krajních bodech intervalu $\langle x_0, x_n\rangle$ rovny nule.

Pokud budeme vycházet z předpokladu, že $\phi(x)$ je kubický polynom, pak $\phi'(x)$ je kvadratický polynom a $\phi''(x)$ je lineární polynom, který prochází body $[x_i, M_i]$

a $[x_{i+1}, M_{i+1}]$, kde $M_i = f''(x_i)$ a $M_{i+1} = f''(x_{i+1})$ jsou tzv. *momenty splinu*. Jelikož je přímka jednoznačně určena dvěma body, můžeme položit $\phi''(x) = L_1(x)$ s nulovou chybou, a zároveň můžeme odvodit vztahy pro výpočet kubické interpolační spline funkce:

$$\begin{aligned} \phi_i''(x) &= L_1(x) = f''(x_i)\frac{x - x_{i+1}}{x_i - x_{i+1}} + f''(x_{i+1})\frac{x - x_i}{x_{i+1} - x_i} && (1.20)\\ &= M_i\frac{x - x_{i+1}}{-h_i} + M_{i+1}\frac{x - x_i}{h_i} = -M_i\frac{x - x_{i+1}}{h_i} + M_{i+1}\frac{x - x_i}{h_i}, && (1.21) \end{aligned}$$

$$\phi_i'(x) = \frac{-M_i}{2h_i}(x - x_{i+1})^2 + \frac{M_{i+1}}{2h_i}(x - x_i)^2 + A_i, \qquad (1.22)$$

$$\phi_i(x) = \frac{-M_i}{6h_i}(x - x_{i+1})^3 + \frac{M_{i+1}}{6h_i}(x - x_i)^3 + A_i(x - x_i) + B_i, \qquad (1.23)$$

kde $h_i = x_{i+1} - x_i$ a A_i a B_i jsou integrační konstanty.

Nyní určíme integrační konstanty, přičemž využijeme skutečnosti, že funkce $\phi_i(x)$ musí na intervalu $\langle x_i, x_{i+1}\rangle$ splňovat podmínky interpolace:

$$\phi_i(x_i) = f(x_i) \text{ a } \phi_i(x_{i+1}) = f(x_{i+1}).$$

Dostaneme

$$\begin{aligned} \phi_i(x_i) &= \frac{-M_i}{6h_i}(x_i - x_{i+1})^3 + \frac{M_{i+1}}{6h_i}(x_i - x_i)^3 + A_i(x_i - x_i) + B_i && (1.24)\\ &= \frac{-M_i}{6h_i}(-h_i)^3 + B_i && (1.25) \end{aligned}$$

a můžeme vyjádřit

$$B_i = f(x_i) - \frac{M_i}{6}{h_i}^2. \qquad (1.26)$$

Analogicky dostaneme

$$\begin{aligned} \phi_i(x_{i+1}) &= -\frac{M_i}{6h_i}(x_{i+1} - x_{i+1})^3 + \frac{M_{i+1}}{6h_i}(x_{i+1} - x_i)^3 + A_i(x_{i+1} - x_i) + B_i && (1.27)\\ &= \frac{M_{i+1}}{6h_i}h_i^3 + A_ih_i + B_i && (1.28)\\ &= \frac{M_{i+1}}{6}h_i^2 + A_ih_i + f(x_i) - \frac{M_i}{6}{h_i}^2. && (1.29) \end{aligned}$$

Z těchto vztahů vyjádříme

$$A_i = \frac{f(x_{i+1}) - f(x_i)}{h_i} + \frac{M_i - M_{i+1}}{6}h_i. \qquad (1.30)$$

Nyní určíme momenty splinu M_i. Protože konstruujeme přirozený kubický spline, tak $M_0 = M_n = 0$. Momenty splinu určíme pomocí další podmínky, kterou musí

funkce $\phi_i(x)$ splňovat. Požadujeme, aby derivace funkce $\phi_i(x)$ byla zleva i zprava spojitá, tj. $\phi_{i-1}^{'}(x_i) = \phi_i^{'}(x_i)$.

Jelikož víme, že

$$\phi_i'(x) = -\frac{M_i}{2h_i}(x - x_{i+1})^2 + \frac{M_{i+1}}{2h_i}(x - x_i)^2 + A_i. \tag{1.31}$$

Potom

$$\begin{aligned}
\phi_{i-1}'(x_i) &= -\frac{M_{i-1}}{2h_{i-1}}(x_i - x_i)^2 + \frac{M_i}{2h_{i-1}}(x_i - x_{i-1})^2 + A_{i-1} && (1.32)\\
&= \frac{M_i}{2h_{i-1}}(h_{i-1})^2 + A_{i-1} && (1.33)\\
&= \frac{M_i}{2h_{i-1}}(h_{i-1})^2 + \frac{f(x_i) - f(x_{i-1})}{h_{i-1}} + \frac{M_{i-1} - M_i}{6}h_{i-1} && (1.34)\\
&= \frac{M_{i-1} + 2M_i}{6}h_{i-1} + \frac{f(x_i) - f(x_{i-1})}{h_{i-1}}. && (1.35)
\end{aligned}$$

Analogicky dostaneme

$$\begin{aligned}
\phi_i'(x_i) &= -\frac{M_i}{2h_i}(x_i - x_{i+1})^2 + \frac{M_{i+1}}{2h_i}(x_i - x_i)^2 + A_i && (1.36)\\
&= -\frac{M_i}{2h_i}(-h_i)^2 + A_i && (1.37)\\
&= -\frac{M_i}{2}h_i + \frac{f(x_{i+1}) - f(x_i)}{h_i} + \frac{M_i - M_{i+1}}{6}h_i && (1.38)\\
&= -\frac{M_{i+1} + 2M_i}{6}h_i + \frac{f(x_{i+1}) - f(x_i)}{h_i}. && (1.39)
\end{aligned}$$

Nyní můžeme napsat následující rovnost:

$$\frac{M_{i-1} + 2M_i}{6}h_{i-1} + \frac{f(x_i) - f(x_{i-1})}{h_{i-1}} = -\frac{M_{i+1} + 2M_i}{6}h_i + \frac{f(x_{i+1}) - f(x_i)}{h_i}, \tag{1.40}$$

$$\frac{M_{i-1} + 2M_i}{6}h_{i-1} + \frac{M_{i+1} + 2M_i}{6}h_i = \frac{f(x_{i+1}) - f(x_i)}{h_i} - \frac{f(x_i) - f(x_{i-1})}{h_{i-1}}. \tag{1.41}$$

Při předpokladu ekvidistantního dělení intervalu platí: $h_{i-1} = h_i$.

$$\begin{aligned}
\frac{M_{i-1} + 2M_i}{6}h_i + \frac{M_{i+1} + 2M_i}{6}h_i &= \frac{f(x_{i+1}) - f(x_i)}{h_i} - \frac{f(x_i) - f(x_{i-1})}{h_i}, && (1.42)\\
\frac{M_{i-1} + 4M_i + M_{i+1}}{6}h_i &= \frac{f(x_{i+1}) - 2f(x_i) + f(x_{i-1})}{h_i}. && (1.43)
\end{aligned}$$

Při praktickém výpočtu interpolace pomocí kubické spline funkce postupujeme v níže naznačených krocích.

Nejdříve vypočteme pomocí rovnic (1.41) momenty splinu $M_i, i = 1, \ldots, n-1$. V praxi se při výpočtu momentů splinu při předpokladu konstrukce přirozeného

kubického splinu využívá Gaussova eliminace aplikovaná na třídiagonální matici pro řešení $n-1$ rovnic o $n-1$ neznámých.

Soustava má tvar

$$\begin{pmatrix} \frac{h_0+h_1}{3} & \frac{h_1}{6} & & & & & & \\ \frac{h_1}{6} & \ddots & \ddots & & & & & \\ & \ddots & \ddots & \ddots & & & & \\ & & \frac{h_{i-1}}{6} & \frac{h_{i-1}+h_i}{3} & \frac{h_{i+1}}{6} & & & \\ & & & \ddots & \ddots & \ddots & & \\ & & & & & \frac{h_{n-2}}{6} & \frac{h_{n-2}+h_{n-1}}{3} \end{pmatrix} \begin{pmatrix} M_1 \\ \vdots \\ M_{i-1} \\ M_i \\ M_{i+1} \\ \vdots \\ M_{n-1} \end{pmatrix} = \begin{pmatrix} g_1 \\ \vdots \\ g_{i-1} \\ g_i \\ g_{i+1} \\ \vdots \\ g_{n-1} \end{pmatrix}, \quad (1.44)$$

kde $g_i = \frac{f(x_{i+1})-f(x_i)}{h_i} - \frac{f(x_i)-f(x_{i-1})}{h_{i-1}}$.

Po výpočtu momentů splinu musíme určit integrační konstanty $A_i, i = 1, \ldots, n$, a $B_i, i = 1, \ldots, n$, podle vztahů (1.26) a (1.30).

V tuto chvíli již známe všechny neznámé a můžeme vypočítat kubické spliny na jednotlivých intervalech dosazením do vztahu (1.23). Bližší informace lze nalézt ve [2] a [6].

1.3 Metoda nejmenších čtverců

Metoda nejmenších čtverců již není interpolační metoda. Její princip můžeme popsat tak, že zadanými body $[x_i, y_i]$ pro $i = 1, \ldots, n$ prokládáme funkci ϕ tak, aby součet druhých mocnin rozdílu hodnot y_i a funkčních hodnot $\phi(x_i)$ byl minimální.

Odvození metody nejmenších čtverců pro lineární regresi

Uvažujme množinu n bodů $[x_i, y_i]$ pro $i = 1, \ldots, n$. Vzdálenost bodů od přímky $y = ax + b$ můžeme vyjádřit jednoduchým způsobem $s_i = |ax_i + b - y_i|$, kde $i = 1, \ldots, n$.

Smyslem této metody je minimalizovat funkci $S(a, b) = \sum_{i=1}^{n}(ax_i + b - y_i)^2$. Protože je známo, že lokální extrém diferencovatelné funkce může nastat pouze ve stacionárním bodě, využijeme pro minimalizaci funkce S parciální derivace

$$\frac{\partial S}{\partial a} = 2\sum_{i=1}^{n}(ax_i + b - y_i)x_i = 2\left(a\sum_{i=1}^{n}x_i^2 + b\sum_{i=1}^{n}x_i - \sum_{i=1}^{n}x_iy_i\right), \quad (1.45)$$

$$\frac{\partial S}{\partial b} = 2\sum_{i=1}^{n}(ax_i + b - y_i) = 2\left(a\sum_{i=1}^{n}x_i + b\sum_{i=1}^{n}1 - \sum_{i=1}^{n}y_i\right). \quad (1.46)$$

Proto nyní položíme tyto parciální derivace rovny nule:

$$\begin{aligned} 2\left(a\sum_{i=1}^{n}x_i^2 + b\sum_{i=1}^{n}x_i - \sum_{i=1}^{n}x_iy_i\right) &= 0, \quad (1.47) \\ 2\left(a\sum_{i=1}^{n}x_i + b\sum_{i=1}^{n}1 - \sum_{i=1}^{n}y_i\right) &= 0. \end{aligned}$$

Tyto rovnice upravíme jednoduchými úpravami na následující tvar,

$$\begin{aligned} a\sum_{i=1}^{n} x_i^2 + b\sum_{i=1}^{n} x_i &= \sum_{i=1}^{n} x_i y_i, \\ a\sum_{i=1}^{n} x_i + bn &= \sum_{i=1}^{n} y_i. \end{aligned} \tag{1.48}$$

čímž získáme soustavu rovnic s neznámými a, b.

Pokud tedy chceme najít aproximační polynom prvního řádu $\phi(x) = ax + b$, musíme zjistit hodnotu koeficientů a, b. Tyto koeficienty zjistíme vyřešením soustavy (1.48).

Protože všechny hlavní subdeterminanty matice jsou kladné, je kvadratická forma pozitivně definitní. Podle Sylvestrova kritéria je tedy nalezený stacionární bodem minima.

Pro aproximaci obecným polynomem $\phi(x) = \sum_{i=0}^{m} b_i x^i$ řešíme následující soustavu rovnic:

$$\begin{aligned} nb_0 + \sum_{i=1}^{n} x_i b_1 + \sum_{i=1}^{n} x_i^2 b_2 + \ldots + \sum_{i=1}^{n} x_i^m b_m &= \sum_{i=1}^{n} y_i, \\ \sum_{i=1}^{n} x_i b_0 + \sum_{i=1}^{n} x_i^2 b_1 + \ldots + \sum_{i=1}^{n} x_i^{m+1} b_m &= \sum_{i=1}^{n} x_i y_i, \\ &\vdots \\ \sum_{i=1}^{n} x_i^m b_0 + \sum_{i=1}^{n} x_i^{m+1} b_1 + \ldots + \sum_{i=1}^{n} x_i^{2m} b_m &= \sum_{i=1}^{n} x_i^m y_i. \end{aligned} \tag{1.49}$$

Další informace lze nalézt v [7].

2 Numerická derivace

Ve druhé kapitole této práce se budeme zabývat metodami pro určení numerické derivace funkce f v bodě.

Definice 2.0.1. *Funkce f má v bodě x_0 derivaci, je-li definována v okolí bodu x_0 a existuje limita*

$$\lim_{h \to 0} \frac{f(x_0 + h) - f(x_0)}{h}. \tag{2.1}$$

Tuto limitu nazýváme derivací funkce f v bodě x_0 a značíme ji $f'(x_0)$.

Metody pro výpočet numerické derivace v bodě vycházejí přímo z definice nebo například z myšlenky nahrazení funkce f v okolí bodu x_0 interpolačním polynomem (funkci lze nahradit například i aproximací získanou metodou nejmenších čtverců, nebo Čebyševovými polynomy) [8].

2.1 Derivace pomocí interpolace

V následujících odstavcích ukážeme, jak můžeme odvodit vztahy pro výpočet derivace funkce v bodě pomocí interpolace.

Funkci f můžeme aproximovat Lagrangeovým interpolačním polynomem

$$f(x) = L_n(x) + \frac{1}{(n+1)!} f^{(n+1)}(\xi)\omega_{n+1}(x), \tag{2.2}$$

$$f(x) = \sum_{i=0}^{n} f(x_i) l_i(x) + \frac{1}{(n+1)!} f^{(n+1)}(\xi)\omega_{n+1}(x). \tag{2.3}$$

Pro chybu Lagrangeovy interpolace platí

$$f(x) - L_n(x) = \frac{1}{(n+1)!} f^{(n+1)}(\xi)\omega(x), \tag{2.4}$$

kde $\omega_{n+1}(x) = \prod_{i=0}^{n} (x - x_i)$.

Zderivujeme-li $\omega_{n+1}(x)$ podle proměnné x, dostaneme

$$\omega'_{n+1}(x) = \sum_{i=0}^{n} \prod_{j=0, j\neq i}^{n} (x - x_j). \tag{2.5}$$

Nyní provedeme derivaci funkce f

$$f'(x) = \sum_{i=0}^{n} \left(f(x_i) l_i'(x) + \frac{1}{(n+1)!} f^{(n+1)}(\xi) \prod_{j=0, j\neq i}^{n} (x - x_j) \right) \tag{2.6}$$

a pro derivaci funkce f v bodě x_0 tedy platí

$$f'(x_0) = \sum_{i=0}^{n} \left(f(x_i) l_i'(x_0) + \frac{1}{(n+1)!} f^{(n+1)}(\xi) \prod_{j=1}^{n} (x_0 - x_j) \right). \tag{2.7}$$

Podle vztahu (2.7) můžeme tedy konstruovat vztahy pro výpočet derivace funkce $f(x)$ v bodě x_0. Nyní odvodíme vztah pro derivaci v bodě pomocí interpolačního polynomu prvního stupně.

Funkci f nahradíme v okolí bodu x_0 interpolačním polynomem prvního stupně. Tento polynom poté zderivujeme podle proměnné x.

Sestrojíme interpolační polynom prvního stupně pro body $[x_i, f(x_i)]$ a $[x_i + h, f(x_i + h)]$,

$$L_1(x) = f(x_i) \frac{x - (x_i + h)}{x_i - (x_i + h)} + f(x_i + h) \frac{x - x_i}{(x_i + h) - x_i} \tag{2.8}$$

$$= f(x_i) \frac{x - (x_i + h)}{-h} + f(x_i + h) \frac{x - x_i}{h}. \tag{2.9}$$

Polynom $L_1(x)$ zderivujeme podle proměnné x. Dostaneme

$$L_1'(x) = \frac{1}{h}(f(x_i + h) - f(x_i)) \tag{2.10}$$

$$f'(x_0) = \frac{f(x_0 + h) - f(x_0)}{h} - \frac{1}{2} f^{(2)}(\xi) h. \tag{2.11}$$

Pokud stejným způsobem využijeme pro určení první derivace funkce $f(x)$ v bodě x_0 interpolační polynom druhého řádu dostáváme

$$L_2'(x) = \frac{1}{2h}(f(x_i + h) - f(x_i - h)), \tag{2.12}$$

$$f'(x_0) = \frac{f(x_0 + h) - f(x_0 - h)}{2h} - \frac{1}{6} f^{(3)}(\xi) h^2. \tag{2.13}$$

Jestliže L_2 zderivujeme ještě jednou dostáváme předpis pro aproximaci druhé derivace funkce f v bodě x_0

$$f''(x_0) = \frac{f(x_0 + h) - 2f(x_0) + f(x_0 - h)}{h^2}. \tag{2.14}$$

Při praktickém výpočtu numerické derivace funkce $f(x)$ v bodě x_0, musíme dbát na vliv zaokrouhlovacích chyb, které mohou mít podstatnou měrou vliv na výsledek. Jmenovatelé uvedených vzorců obsahují krok h, který musí být jakýmsi kompromisem mezi dostatečně přesnou aproximací, která vyžaduje dostatečně malý krok h, a zaokrouhlovací chybou, která se naopak při malém h zvyšuje. Bližší informace v [3] nebo [9].

2.2 Richardsonova extrapolace

Richardsonova extrapolace je technika, která je v praxi hojně využívaná. Setkáme se s ní například v numerické integraci v Rombergově kvadratuře nebo například v Bulirsch-Stoerovu algoritmu, jenž se zabývá řešením obyčejných diferenciálních rovnic. Její podstata vychází z předpokladu, že ze dvou přibližných výsledků můžeme pomocí lineární kombinace vypočítat třetí, který bude přesnější, přičemž tato nová hodnota se nachází mimo interval, ohraničený předcházejícími dvěma hodnotami (proto hovoříme o extrapolaci).

Odvození Richardsonovy extrapolace

Mějme funkci R a krok h. Předpokládejme, že funkci R je možné vyjádřit mocninnou řadou

$$R(h) = a_0 + a_1 h + a_2 h^2 + a_3 h^3 + a_4 h^4 + \dots . \tag{2.15}$$

Jestliže $h < 1$, pak je $R(h)$ dobrou aproximací členu a_0. Lepší aproximaci ovšem získáme, když určíme hodnotu funkce R s krokem $\frac{h}{2}$

$$R\left(\frac{h}{2}\right) = a_0 + a_1\frac{h}{2} + \underbrace{a_2\left(\frac{h}{2}\right)^2 + a_3\left(\frac{h}{2}\right)^3 + \dots .}_{O(h^2)} \tag{2.16}$$

Pomocí vhodné lineární kombinace můžeme vyjádřit člen a_0 s chybou $O(h^2)$. Ve vyjádření (2.15) a (2.16) zanedbáme další členy rozvoje

$$R(h) - a_0 = a_1 h + O\left(h^2\right), \tag{2.17}$$

$$R\left(\frac{h}{2}\right) - a_0 = a_1\frac{h}{2} + O\left(h^2\right). \tag{2.18}$$

Rovnici (2.17) odečteme od dvojnásobku rovnice (2.18)

$$R_2(h) = 2R\left(\frac{h}{2}\right) - R(h) = a_0 + \underbrace{a_2^{(2)}(h)^2 + a_3^{(2)}(h)^3 + \dots .}_{O(h^2)} \tag{2.19}$$

Pomocí vhodné lineární kombinace vztahů, z nichž oba aproximují hodnotu funkce R s chybou $O(h)$, jsme vyjádřili vztah pro a_0 s chybou $O(h^2)$.

Podle [13] platí, že $|a_i^{(2)}| < |a_i|$. $R_2(h)$ je lepší aproximace než $R(h)$ a pro malé h je i lepší aproximací než $R(\frac{h}{2})$.

Obecný vztah pro Richardsonovu extrapolaci můžeme zapsat pomocí následující iterační formule:

$$R_{j+1}(h) = R_j\left(\frac{h}{2}\right) + \frac{R_j\left(\frac{h}{2}\right) - R_j(h)}{2^j - 1}, \text{ kde } j = 1, 2, \dots . \tag{2.20}$$

Výpočet můžeme uspořádat do následující tabulky:

$R(h)$					
$R(\frac{h}{2})$	$R_2(h)$				
$R(\frac{h}{4})$	$R_2(\frac{h}{2})$	$R_3(h)$			
$R(\frac{h}{8})$	$R_2(\frac{h}{4})$	$R_3(\frac{h}{2})$	$R_4(h)$		
$R(\frac{h}{16})$	$R_2(\frac{h}{8})$	$R_3(\frac{h}{4})$	$R_4(\frac{h}{2})$	$R_5(h)$	
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$
$\uparrow$	$\uparrow$	$\uparrow$	$\uparrow$	$\uparrow$	
$O(h)$	$O(h^2)$	$O(h^3)$	$O(h^4)$	$O(h^5)$	

Tabulka 2.1: Richardsonova extrapolace

Při výpočtu postupujeme po řádcích, přičemž prvek v prvním sloupci vypočteme pomocí základní metody. Ostatní prvky v řádku vypočteme pomocí vztahu (2.20). Výpočet ukončíme ve chvíli, kdy je rozdíl dvou diagonálních prvků tabulky menší než požadovaná přesnost. Více například v [10], [11], [12] nebo [13].

3 Numerická integrace funkcí

Ve další kapitole se budeme zabývat numerickou integrací.

Je-li funkce $f(x)$ na intervalu $\langle a, b\rangle$ spojitá a známe její primitivní funkci $F(x)$, můžeme hodnotu určitého integrálu $\int_a^b f(x)dx$ určit pomocí Newton - Leibnizova vztahu

$$\int_a^b f(x)dx = F(b) - F(a). \tag{3.1}$$

V praxi ovšem často nemůžeme primitivní funkci $F(x)$ najít analyticky nebo je analytický výpočet integrálu příliš složitý a pracný. V tuto chvíli využijeme metody numerické kvadratury, kdy funkci $f(x)$ obvykle nahrazujeme jednodušší aproximující funkcí $\phi(x)$(nejčastěji polynomem), a hodnotu integrálu $f(x)$ položíme přibližně hodnotě integrálu $\phi(x)$ [14].

Definice 3.0.1. *Vyjádřeme integrál ve tvaru*

$$I(f) = \int_a^b f(x)dx \approx I_h(f) = \int_a^b \phi(x)dx = \sum_{i=0}^{n} \alpha_i f(x_i) \tag{3.2}$$

kde $I_h(f) = \sum_{i=0}^{n} \alpha_i f(x_i)$ *se nazývá kvadraturní formule,* α_i *jsou koeficienty kvadraturní formule, které nezávisí na funkci* $f(x)$, x_i *jsou uzly kvadraturní formule, přičemž* $x_i \in \langle a, b\rangle$. *Chybu kvadraturní formule* $R_n(f)$ *vyjádříme* $R_n(f) = I(f) - I_h(f)$.

U kvadraturních formulí nás zajímá, jakou přesnost daná kvadraturní formule při aproximaci určitého integrálu má. Řekneme, že kvadraturní formule je řádu n, když integruje polynomy stupně n s nulovou chybou a polynomy stupně $n + 1$ integruje s nenulovou chybou [2]. To nás vede k následující definici.

Definice 3.0.2. *Řád kvadraturní formule* $I_h(f) = \sum_{i=0}^{n} \alpha_i f(x_i)$ *je maximální číslo* $m \in \mathbb{N} \cup \{0\}$ *takové, že* $\int_a^b p(x)dx = \sum_{i=0}^{n} \alpha_i p(x_i)$, *kde* $p(x)$ *je polynom stupně* m.

3.1 Newton-Cotesovy vzorce

Velkou skupinou v oblasti numerické integrace funkcí představují Newton-Cotesovy vzorce. Tyto kvadraturní vzorce předpokládají ekvidistantní dělení intervalu integrace.

Newton-Cotesovy vzorce lze rozdělit do dvou podskupin. První skupinu označujeme jako otevřené a druhou jako uzavřené. Rozdíl mezi těmito skupinami spočívá v přístupu ke krajním bodům intervalu integrace. Uzavřené vzorce tyto body považují za uzly kvadratury, otevřené nikoli. Uzly kvadratury otevřených vzorců jsou určeny symetricky podle středu intervalu.

Newton-Cotesovy kvadraturní vzorce aproximují hodnotu integrálu pomocí nahrazení funkce $f(x)$ Lagrangeovým interpolačním polynomem $L_n(x)$. Více například v [2], [15].

Odvození Newton-Cotesových vzorců

Nyní odvodíme obecný tvar Newton - Cotesových vzorců. Hodnotu integrálu funkce f na intervalu $\langle a, b\rangle$ aproximujeme pomocí Lagrangeova interpolačního polynomu $L_n(x)$.

$$\int_a^b f(x)dx \approx \int_a^b L_n(x)dx =$$
$$= \int_a^b \sum_{i=0}^{n} f(x_i) \underbrace{\prod_{j=0, j\neq i}^{n} \left(\frac{x - x_j}{x_i - x_j}\right)}_{l_i(x)} dx = \sum_{i=0}^{n} f(x_i) \underbrace{\int_a^b l_i(x)dx}_{\alpha_i}, \tag{3.3}$$

kde $a = x_0 < x_1 < x_2 < \ldots < x_n = b$. Určíme koeficienty α_i. Položme $h = \frac{b-a}{n}$ a

$$x_0 = a + 0h, x_1 = a + h, \ldots, x_n = a + nh. \tag{3.4}$$

Nyní provedeme substituci

$$x = a + th, dx = hdt. \tag{3.5}$$

Použitím substituce došlo ke změně integračních mezí

$$t = \frac{x - a}{h}. \tag{3.6}$$

Pokud za proměnnou x dosadíme dolní mez intervalu integrace dostáváme jako novou dolní mez nulovou hodnotu. Dosadíme-li do vztahu (3.6) za proměnnou x výraz $b = a + nh$, dostáváme pro novou horní mez integrálu hodnotu n:

$$\alpha_i = h \int_0^n \prod_{j=0, j\neq i}^{n} \left(\frac{t - j}{i - j}\right) dt. \tag{3.7}$$

Z Newton - Cotesových vzorců lze poměrně jednoduše odvodit pravidla a jejich složené varianty pro výpočet určitého integrálu.

Princip aproximace určitého integrálu pomocí složených vzorců spočívá v rozdělení intervalu na jednotlivé podintervaly. Přičemž na každý z podintervalů použijeme jednoduché Newton - Cotesovy vzorce. Hodnotu integrálu poté aproximujeme hodnotou, kterou vypočteme jako součet obsahu ploch na jednotlivých intervalech, čímž dostáváme přesnější aproximaci daného integrálu.

Šířku intervalu h určíme jako $h = \frac{b-a}{m}$, kde m je počet podintervalů. Další informace můžeme nalézt v [2] nebo [16].

Věta 3.1.1. *Kvadraturní formule získaná integrací interpolačního polynomu pro uzlové body $[x_i, f(x_i)]$ pro $i = 0, \ldots, n$ má stupeň přesnosti alespoň n.*

Důkaz. Uvažujme integrál $I(f)$, který aproximujeme Lagrangeovým interpolačním polynomem L_n stupně n s chybou $E_n(f)$

$$I(f) = \int_a^b L_n(x) + E_n(f)dx. \tag{3.8}$$

Upravíme pravou stranu rovnosti

$$\begin{aligned} I(f) &= \int_a^b \sum_{i=0}^{n} f(x_i)l_i(x) + E_n(f)dx \qquad (3.9)\\ &= \int_a^b \sum_{i=0}^{n} f(x_i)l_i(x)dx + \int_a^b E_n(f)dx. \\ &= \sum_{i=0}^{n} f(x_i) \int_a^b l_i(x)dx + \int_a^b E_n(f)dx, \text{ kde } \int_a^b l_i(x)dx = \alpha_i. \end{aligned}$$

Nyní vyjádříme chybu $E_n(f)$

$$I(f) - \sum_{i=0}^{n} f(x_i)\alpha_i = \int_a^b E_n(f)dx. \tag{3.10}$$

Přičemž pro chybu Lagrangeovy interpolace platí

$$E_n(f) = f^{(n+1)}(\xi)\frac{\omega_{n+1}(x)}{(n+1)!}. \tag{3.11}$$

Nyní můžeme vidět, že vyjádření chyby $E_n(f)$ obsahuje $(n+1)$-ní derivací funkce f. Z toho přímo plyne, že chyba interpolační kvadraturní formule bude nulová pro všechny polynomy stupně nejvýše n a řád přesnosti bude tedy alespoň n. □

Později ukážeme, že interpolační kvadraturní formule má řád přesnosti pro $n+1$ bodů nejvýše $2n+1$. Další informace v [17].

3.1.1 (Složené) obdélníkové pravidlo

Je nejjednodušším vzorcem pro kvadraturu a řadíme jej mezi otevřené Newton - Cotesovy vzorce. Jeho řád přesnosti je jedna, ovšem je vhodné jej využít v případě,

kdy nemůžeme vypočíst integrál složitějšími metodami z důvodu nemožnosti určení funkční hodnoty na krajích integrálu.

Princip tohoto pravidla spočívá v určení funkční hodnoty středu intervalu. Hodnota určitého integrálu je tak aproximována jako obsah obdélníku, kdy jedna strana je určena šířkou intervalu a druhá funkční hodnotou.Dostaneme

$$\int_a^b f(x)dx \approx hf\left(\frac{a+b}{2}\right). \tag{3.12}$$

Pokud jsme ovšem schopni určit funkční hodnoty i na krajích integračních mezí, je vhodnější použít dále popsané metody.

Předpis pro aproximaci určitého integrálu pomocí složeného obdélníkového pravidla:

$$\int_a^b f(x)dx \approx h\sum_{i=1}^{m} f\left(\frac{x_{i-1}+x_i}{2}\right), \tag{3.13}$$

kde m je počet podintervalů. Další informace lze nalézt v [3].

3.1.2 (Složené) lichoběžníkové pravidlo

Jedná se o jednoduchou metodu s řádem přesnosti jedna, která je snadno implementovatelná pomocí počítače. Hodnota integrálu je vypočtena jako obsah lichoběžníku, kdy funkci f nahrazujeme Lagrangeovým interpolačním polynomem prvního řádu. Pokud využijeme složenou variantu tohoto pravidla je interval rozdělen na části a na každé této části je zkonstruován lichoběžník. Hodnota integrálu je určena jako součet obsahu jednotlivých lichoběžníků.

Použitím vztahu (3.3) odvodíme vztah pro lichoběžníkové pravidlo. Určíme šířku intervalu $h = x_1 - x_0$, kde $a = x_0 < x_1 = b$. Integrál funkce f na intervalu $\langle a, b\rangle$ aproximujeme pomocí Lagrangeova interpolačního polynomu prvního stupně

$$\int_a^b f(x)dx \approx \sum_{i=0}^{1} \alpha_i f(x_i). \tag{3.14}$$

Pro koeficienty α_i dle vztahu (3.7) platí

$$\alpha_i = h\int_0^1 \prod_{j=0, j\neq i}^{n}\left(\frac{t-j}{i-j}\right)dt, \tag{3.15}$$

$$\alpha_0 = h\int_0^1 \frac{t-1}{0-1}\,dt = h\int_0^1 1-t\,dt = h\left[t-\frac{t^2}{2}\right]_0^1 = \frac{h}{2}, \tag{3.16}$$

$$\alpha_1 = h\int_0^1 \frac{t-0}{1-0}\,dt = h\int_0^1 t\,dt = h\left[\frac{t^2}{2}\right]_0^1 = \frac{h}{2}. \tag{3.17}$$

Tímto dostáváme vztah pro lichoběžníkové pravidlo

$$\int_a^b f(x)\,dx \approx \frac{h}{2}\left[f(a)+f(b)\right]. \tag{3.18}$$

Předpis pro aproximaci určitého integrálu pomocí složeného lichoběžníkového pravidla:

$$\int_a^b f(x)dx \approx \frac{h}{2}\left[f(x_0) + 2\sum_{i=1}^{m-1} f(x_i) + f(x_m)\right], \tag{3.19}$$

kde m je zvolený počet podintervalů. Více například v [3], [18].

3.1.3 (Složené) Simpsonovo pravidlo

Další metodou vycházející z Newton - Cotesových vzorců je Simpsonovo pravidlo. Ze zadaného intervalu vybereme body na obou krajích intervalu a bod uprostřed tohoto intervalu. Tyto body proložíme křivkou, která je grafem polynomu druhého řádu-parabolou. Hodnota integrálu je určena jako obsah plochy pod parabolou.

$$\int_a^b f(x)dx \approx \frac{h}{3}\left[f(a) + 4f\left(\frac{a+b}{2}\right) + f(b)\right], \tag{3.20}$$

kde $h = \frac{b-a}{2}$.

U složené varianty tohoto pravidla rozdělujeme daný interval na sudý počet podintervalů a v každém intervalu $\langle x_{2i}, x_{2i+2}\rangle$, $i = 0, \ldots, \frac{m}{2} - 1$, se provede náhrada původní funkce polynomem druhého řádu. Hodnota integrálu se určí jako součet ploch pod těmito parabolami.

Předpis pro aproximaci určitého integrálu pomocí složeného Simpsonova pravidla (hodnota h opět představuje šířku podintervalu a m počet podintervalů):

$$\int_a^b f(x)dx \approx \frac{h}{3}\left[f(x_0) + 4\sum_{i=1}^{\frac{m}{2}} f(x_{2i-1}) + 2\sum_{i=1}^{\frac{m}{2}-1} f(x_{2i}) + f(x_m)\right], \tag{3.21}$$

kde $h = \frac{b-a}{m}$. Bližší informace jsou k nalezení v [19].

3.1.4 (Složené) tříosminové pravidlo

K určení hodnoty integrálu využívá tato metoda plochy pod grafem kubického polynomu, který je určen čtyřmi body, které se nacházejí na obou krajích intervalu a současně v jedné a druhé třetině daného intervalu. Pokud zvolíme složenou variantu tohoto pravidla, je nutné, aby počet požadovaných podintervalů byl dělitelný třemi.

$$\int_a^b f(x)dx \approx \frac{3h}{8}\left[f(a) + 3f\left(\frac{2a+b}{3}\right) + 3f\left(\frac{a+2b}{3}\right) + f(b)\right], \tag{3.22}$$

kde $h = \frac{b-a}{3}$.

Předpis pro aproximaci určitého integrálu pomocí složeného tříosminového pravidla:

$$\begin{aligned}\int_a^b f(x)dx &\approx \frac{3h}{8}\left[f(x_0)+3\sum_{i=1}^{\frac{m}{3}}(f(x_{3i-1})+f(x_{3i-2}))\right. \\ &+ \left. 2\sum_{i=1}^{\frac{m}{3}-1} f(x_{3i})+f(x_m)\right],\end{aligned} \tag{3.23}$$

kde $h=\frac{b-a}{m}$. Podrobnější informace získáme v [20].

3.1.5 (Složené) Boolevo pravidlo

Booleovo pravidlo je dalším z uzavřených Newton - Cotesových vzorců a má z uvedených vzorců nejvyšší řád přesnosti, protože je přesný až pro polynomy pátého stupně. K výpočtu hodnoty integrálu využívá kromě bodů na krajích intervalů a prostředního bodu také další dva, které jsou umístěny v jedné a třetí čtvrtině intervalu. Pokud zvolíme složenou variantu tohoto pravidla, je nutné, aby počet požadovaných podintervalů byl dělitelný čtyřmi. Vzorec má tvar:

$$\begin{aligned}\int_a^b f(x)dx &\approx \frac{h}{90}\left[7f(a)+32f\left(\frac{3a+b}{4}\right)+12f\left(\frac{a+b}{2}\right)\right. \\ &+ \left. 32f\left(\frac{a+3b}{4}\right)+7f(b)\right],\end{aligned} \tag{3.24}$$

kde $h=\frac{b-a}{4}$.

Předpis pro aproximaci určitého integrálu pomocí složeného Booleova pravidla:

$$\begin{aligned}\int_a^b f(x)dx &\approx \frac{h}{90}\left[7f(x_0)+32\sum_{i=1}^{\frac{m}{4}}(f(x_{4i-1})+f(x_{4i-3}))\right. \\ &+ \left. 12\sum_{i=1}^{\frac{m}{4}} f(x_{4i-2})\ 14\sum_{i=1}^{\frac{m}{4}-1} f(x_{4i})+7f(x_m)\right],\end{aligned} \tag{3.25}$$

kde $h=\frac{b-a}{m}$, více například v [16].

3.1.6 Chyba Newton - Cotesových vzorců

Z věty 3.1.1 plyne, že pro chybu Newton - Cotesových vzorců platí:

$$R_n(f)=\frac{f^{(n+1)}(\xi)}{(n+1)!}\int_a^b \omega_{n+1}(x), \tag{3.26}$$

kde $\omega_{n+1}(x)=(x-x_0)(x-x_1)\dots(x-x_n)$.

V praxi můžeme hodnotu $|f^{n+1}(\xi)|$ omezit $\max_{x\in\langle a,b\rangle}|f^{(n+1)}(x)|$.

Pokud interval integrace rozdělíme na m podintervalů a na každém z nich aproximujeme funkci $f(x)$ Lagrangeovým interpolačním polynomem stupně n, budeme chybu $\overline{R_n}(f)$ odhadovat jako součet chyb na dílčích intervalech. Tedy

$$\int_a^b f(x)dx = \sum_{i=1}^{n}\int_{x_i-1}^{x_i} f(x)dx \approx \sum_{i=1}^{n}\int_{x_i-1}^{x_i} L_{n,i}(x)dx, \tag{3.27}$$

$$\overline{R_n}(f) = \sum_{i=1}^{n} R_{n,i}(f), \tag{3.28}$$

kde $R_{n,i}(f)$ označuje chybu na intervalu $\langle x_{i-1}, x_i\rangle$.

V následujících odstavcích odvodíme chybu pro lichoběžníkové pravidlo. Označíme

$$\begin{aligned} h = b-a, a &= x_0 < x_1 = b, \\ x_0 &= a + 0h, \\ x_1 &= a + h, \end{aligned} \tag{3.29}$$

a použijeme substituci

$$x = a + th, dx = hdt. \tag{3.30}$$

Pro chybu platí

$$\begin{aligned} R_1(f) &= \frac{f''(\xi)}{2}\int_a^b (x-a)(x-b)dx \qquad (3.31)\\ &= \frac{f''(\xi)}{2}h\int_0^1 (a+th-a)(a+th-(a+h))dt \\ &= \frac{f''(\xi)}{2}h^3\int_0^1 t(t-1)dt \\ &= \frac{-f''(\xi)}{12}h^3 \\ &= \frac{-f''(\xi)}{12}(b-a)^3. \end{aligned}$$

Je-li funkce f'' na intervalu $[a,b]$ spojitá, existuje bod $\xi \in \langle a,b\rangle$ takový, že platí

$$f''(\xi_1) + f''(\xi_2) + \ldots + f''(\xi_m) = mf''(\xi). \tag{3.32}$$

Pro odhad chyby složeného lichoběžníkového pravidla, kde $h = \frac{b-a}{m}$, pak dostáváme

$$\overline{R_1}(f) = -m\frac{f''(\xi)}{12}h^3 \tag{3.33}$$

$$= -\frac{(b-a)h^2}{12}f''(\xi) \tag{3.34}$$

Analogicky bychom mohli odvodit vztahy pro chyby dalších pravidel. Chyba Simpsonova pravidla je dána vztahem

$$R_2(f) = \frac{-f^{(4)}(\xi)}{90} h^5. \tag{3.35}$$

Pro chybu jeho složené varianty platí

$$\overline{R_2}(f) = -\frac{(b-a)h^4}{180} f^{(4)}(\xi). \tag{3.36}$$

Více například v [1] nebo [18].

3.2 Metoda polovičního kroku

Chybu numerické kvadratury lze odhadnout pomocí vztahů (3.26). V těchto vztazích se vyskytuje maximální hodnota derivace na intervalu (a, b), kterou není jednoduché zjistit. Odhad chyby je také hodně pesimistický, protože ve skutečnosti je chyba mnohem menší.

Z toho důvodu se pro odhad chyby využívá nejčastěji metoda poloviční kroku.

Metoda poloviční kroku vychází z myšlenky, že chybu lze vyjádřit v v závislosti na kroku h následující řadou.

$$R(h) = a_k h^k + a_{k+1} h^{k+1} + \ldots, \text{ kde } k = 0, 1, \ldots, \tag{3.37}$$

pokud pro chybu metody platí, že je řádu $O(h^k)$. Pro obdélníkovou a lichoběžníkovou metodu je $k = 2$, pro Simpsonovu metodu a tříosminové pravidlo je $k = 4$ a $k = 6$ v případě Booleova pravidla.

Odvození metody poloviční kroku

Spočteme integrál $I(f)$ s dvěma různými kroky h_1 a h_2 a dostaneme

$$I(f) = I(h_1) + a_k h_1^k + a_{k+1} h_1^{k+1}, \tag{3.38}$$
$$I(f) = I(h_2) + a_k h_2^k + a_{k+1} h_2^{k+1}. \tag{3.39}$$

Označme

$$E(h) = I(f) - I(h). \tag{3.40}$$

Zanedbáme vyšší členy v rozvoji chyby

$$I(f) \approx I(h_1) + a_k h_1^k, \tag{3.41}$$
$$I(f) \approx I(h_2) + a_k h_2^k. \tag{3.42}$$

Od rovnice (3.42) odečteme rovnici (3.41). Tím dostaneme

$$\begin{aligned} 0 &\approx [I(h_2) - I(h_1)] + a_k h_2^k - a_k h_1^k, \\ 0 &\approx [I(h_2) - I(h_1)] + a_k (h_2^k - h_1^k), \\ a_k &\approx \frac{[I(h_2) - I(h_1)]}{h_1^k - h_2^k}, \end{aligned} \tag{3.43}$$

a proto

$$E(h_1) \approx a_k h_1^k \approx \frac{[I(h_2) - I(h_1)]}{h_2^k - h_1^k} h_1^k. \tag{3.44}$$

Pokud zvolíme $h_1 = h$ a $h_2 = \frac{h}{2}$ (odtud název metoda poloviční kroku) dostáváme po jednoduché úpravě pro chybu s krokem h

$$E(h) = \frac{2^k}{2^k - 1}\left[I\left(\frac{h}{2}\right) - I(h)\right], \tag{3.45}$$

kde $k = 1, \ldots$. Metoda poloviční kroku je tedy založena na stejné myšlence jako jeden krok Richardsonovy extrapolace. Více v [21].

3.3 Rombergova kvadratura

Rombergova kvadratura vychází z myšlenky Euler - Maclaurinova vzorce, kdy můžeme chybu složeného lichoběžníkového ($CT_h(f)$) pravidla rozvinout do řady sudých mocnin integračního kroku h

$$CT_h(f) = I(f) + \sum_{i=1}^{n} a_i h^{2i}. \tag{3.46}$$

Tím zajistíme vyšší přesnosti kvadraturních vzorců. Tuto myšlenku vyjádříme pomocí vztahu. K tomu Rombergova kvadratura využívá tzv. Richardsonovy extrapolace (2.2), která je využívána i při výpočtech numerické derivace.

V praxi bývá Rombergova kvadratura znázorňována pomocí stejné tabulky jako u Richardsonovy extrapolace (2.1), kde hodnoty na diagonále představují aproximaci integrálu s chybami řádu $O(h^4), O(h^6), \ldots$. První sloupec (při znázornění Rombergovy kvadratury pomocí tabulky) odpovídá složenému lichoběžníkovému pravidlu s kroky $h/2, h/4, \ldots$, druhý složenému Simpsonovu pravidlo a třetí složenému Booleovu pravidlu. Platí, že n - tý sloupec je kvadraturní formule řádu $2n+1$ s krokem h_n.

Tento způsob výpočtu je vhodný pro počítače, protože pro výpočet lepší aproximace určitého integrálu využíváme hodnoty, které jsou již vypočtené. Další informace jsou k nalezení v [13] a [15].

Odvození Rombergovy kvadratury pro chybu $O(h^6)$

Vyjádřeme postupně chybu integrálu $O(h^6)$ pro kroky $h, \frac{h}{2}, \frac{h}{4}$. Ze vztahu (3.46) plyne

$$I(f) - CT_h(f) = a_1 h^2 + a_2 h^4 + O(h^6) \tag{3.47}$$

$$I(f) - CT_{\frac{h}{2}}(f) = a_1 \left(\frac{h}{2}\right)^2 + a_2 \left(\frac{h}{2}\right)^4 + O(h^6) \tag{3.48}$$

$$I(f) - CT_{\frac{h}{4}}(f) = a_1 \left(\frac{h}{4}\right)^2 + a_2 \left(\frac{h}{4}\right)^4 + O(h^6). \tag{3.49}$$

Nyní pomocí vhodné lineární kombinace vyjádříme $I(f)$ s chybou $O(h^6)$. Rovnici (3.47) vynásobíme neznámou x_1. Rovnici (3.48) vynásobíme neznámou x_2. Rovnici (3.49) vynásobíme neznámou x_3. Následně tyto rovnice sečteme. Celkově tedy dostáváme

$$\begin{aligned}(x_1+x_2+x_3)I(f) &= x_1 CT_h(f) + x_2 CT_{\frac{h}{2}}(f) + x_3 CT_{\frac{h}{4}}(f) + \\ &+ x_1 a_1 h^2 + x_2 a_1 \frac{h^2}{4} + x_3 a_1 \frac{h^2}{16} + \\ &+ x_1 a_2 h^2 + x_2 a_2 \frac{h^2}{16} + x_3 a_2 \frac{h^2}{256} + O(h^6).\end{aligned} \tag{3.50}$$

Nyní sestavíme soustavu rovnic tak, abychom u $I(f)$ dostali hodnotu 1 a eliminovali členy s koeficienty a_1, a_2 a a_3. Dostáváme soustavu

$$\begin{aligned}x_1 + x_2 + x_3 &= 1, \\ x_1 + \frac{x_2}{4} + \frac{x_3}{16} &= 0, \\ x_1 + \frac{x_2}{16} + \frac{x_3}{256} &= 0.\end{aligned} \tag{3.51}$$

Vyřešením této soustavy tří rovnic se třemi neznámými (např. Gaussovou eliminací) získáme $x_1 = \frac{64}{45}, x_2 = \frac{-20}{45}$ a $x_3 = \frac{1}{45}$. Integrál $I(f)$ s chybou $O(h^6)$ vypočteme tedy

$$I(f) \approx \frac{64CT_h(f) - 20CT_h(f) + CT_h(f)}{45}. \tag{3.52}$$

Obecný vztah pro Rombergovu metodu můžeme zapsat pomocí následující iterační formule:

$$I_{2h}^{(k)} = \frac{4^k I_{2h-1}^{(k)} - I_h^{(k-1)}}{4^k - 1}, \text{ kde } k = 1, 2, \ldots. \tag{3.53}$$

3.4 Gaussova kvadratura

V případě Newton-Cotesových vzorců jsme předpokládali ekvidistantní dělení intervalu, na kterém jsme chtěli aproximovat hodnotu integrálu. Pokud upustíme od požadavku ekvidistantního dělení intervalu, jsme schopni dosáhnout vyšší algebraické přesnosti.

Věta 3.4.1. *Řád kvadraturní formule pro $n+1$ bodů je nejvýše $2n+1$.*

Důkaz. Nechť polynom $p(x) = \prod_{i=0}^{n}(x - x_i)^2 \in \Pi_{2n+2}$, kde x_i jsou uzly kvadraturní formule. Polynom $p(x)$ je nezáporná funkce na intervalu $\langle a, b\rangle$, pro který platí

$$\int_a^b p(x) > 0.$$

Kvadraturní formule (3.2) je ovšem rovna nule, což je spor. Pro tento polynom tedy není kvadraturní formule přesná a řád přesnosti je tedy nejvýše $2n+1$. □

Kvadraturní formule, konstruovaná tak, že její řád je $2n+1$, se nazývá Gaussova. Předtím, než budeme definovat Gaussovy kvadraturní vzorce, je nezbytné, abychom si objasnili pojem ortogonální polymony.

3.4.1 Ortogonální polynomy

Zaveďme množinu Π_n pro množinu všech polynomů stupně nejvýše n. Symbolem $\tilde{\Pi}_n$ budeme značit množinu všech normovaných polynomů stupně nejvýše n (jejich koeficient u nejvyšší mocniny je roven jedné).

Definice 3.4.1. *Nechť ω je funkce, o které předpokládáme, že je integrovatelná a nezáporná na intervalu $\langle a, b\rangle$ a $\omega(x) > 0$ skoro všude na $[a, b]$. Takovou funkci budeme nazývat váhovou funkcí.*

Použitím váhové funkce $\omega(x)$ se můžeme podle [17] vyhnout řadě problémů. Jedním z nich je například singularita integrandu v bodech, které jsou uzly kvadraturní formule.

V další části se proto budeme zabývat výpočtem integrálu

$$\int_a^b \omega(x) f(x) dx. \tag{3.54}$$

Dále definujeme skalární součin funkcí.

Definice 3.4.2. *Skalárním součinem na prostoru spojitých reálných funkcí na uzavřeném intervalu $\langle a,b\rangle$ budeme rozumět*

$$\langle f, g\rangle = \int_a^b \omega(x) f(x) g(x) dx$$

Definice 3.4.3. *Jestliže $\langle f, g\rangle = 0$, říkáme, že funkce f, g jsou ortogonální na intervalu $\langle a, b\rangle$.*

Věta 3.4.2. *Pro váhovou funkci $\omega(x)$ na intervalu $\langle a, b\rangle$ existují polynomy $p_j \in \tilde{\Pi}_j$ $j = 0, 1, 2, \ldots$ takové, že*

$$\langle p_i, p_k\rangle = 0 \text{ pro } i \neq k.$$

Tyto polynomy lze sestrojit pomocí Gram - Schmidtova ortogonalizačního procesu.

Věta 3.4.3.

1. *Každý polynom $p \in \Pi_k$ lze vyjádřit jako lineární kombinaci ortogonálních polynomů $p_i \in \tilde{\Pi}_i$, $i \leqq k$.*
2. *Polynom $p_n \in \tilde{\Pi}_n$ je ortogonální ke všem polynomům $p \in \Pi_{n-1}$.*
3. *Kořeny každého polynomu z posloupnosti ortogonálních polynomů jsou reálné, jednoduché a leží v (a, b).*

Nyní již můžeme přistoupit k samotné charakterizaci Gaussova kvadraturního vzorce. Bližší informace jsou k nalezení v [2], [17] a [22].

3.4.2 Gaussův Kvadraturní vzorec

Věta 3.4.4. *Nechť kvadraturní formule ve tvaru (3.2) má stupeň přesnosti alespoň n. Předpokládejme, že $p_n \in \tilde{\Pi}_n$, $n = 0, 1, \ldots$, jsou ortogonální polynomy na intervalu $\langle a, b\rangle$ vzhledem k váhové funkci ω. Pak tato formule má stupeň přesnosti $2n + 1$, právě tehdy, když uzly x_i pro $i = 0, 1, \ldots, n$ této kvadraturní formule jsou kořeny polynomu $p_{n+1} \in \tilde{\Pi}_{n+1}$.*

Důkaz. Uvažujme kvadraturní formuli (3.2), která má stupeň přesnosti alespoň n. Tuto formuli můžeme podle věty 3.1.1 získat například integrací interpolačního polynomu. Dále uvažujme polynom $q(x) \in \Pi_{2n+1}$. Zřejmě platí, že

$$q(x) = p_{n+1}(x)s_n(x) + r_n(x), \tag{3.55}$$

kde s_n je podíl po dělení polynomu $q(x)$ polynomem $p_{n+1}(x)$ a $r_n(x)$ je zbytek po tomto dělení. Nyní vypočteme chybu $R(q)$.

$$\begin{aligned} R(q) &= \int_a^b \omega(x)q(x)dx - \sum_{i=0}^{n} \alpha_i q(x_i) \qquad (3.56)\\ &= \int_a^b \omega(x)\left[p_{n+1}(x)s_n(x) + r_n(x)\right]dx - \sum_{i=0}^{n} \alpha_i \left[p_{n+1}(x_i)s_n(x_i) + r_n(x_i)\right]\\ &= \int_a^b \omega(x)\left[p_{n+1}(x)s_n(x)\right]dx - \sum_{i=0}^{n} \alpha_i p_{n+1}(x_i)s_n(x_i)\\ &+ \int_a^b \omega(x)r_n(x)dx - \sum_{i=0}^{n} \alpha_i r_n(x_i) \end{aligned}$$

Polynom $p_{n+1}(x)$ je ortogonální s vahou $\omega(x)$ k polynomu $s_n(x)$. Zároveň platí, že uzly x_i jsou kořeny polynomu $p_{n+1}(x)$. První sčítanec je tedy roven nule. Z předpokladu plyne, že stupeň přesnosti kvadraturní formule je alespoň n. Proto je i druhý sčítanec roven nule. Pro chybu $R(q)$ platí, že je rovna nule pro libovolný polynom $q(x) \in \Pi_{2n+1}$. Stupeň přesnosti kvadraturní formule je tedy $2n + 1$. □

Připomeňme, že taková formule se nazývá Gaussova. Gaussovy interpolační kvadraturní vzorce volí tedy koeficienty a uzly kvadratury tak, aby jejich řád přesnosti byl maximální. Více například v [3] nebo [17].

Odvození Gaussova kvadraturního vzorce

Podle [2] určíme koeficienty α_i Gaussovy kvadratury, tak aby platilo

$$\int_a^b \omega(x)q(x)dx = \sum_{i=0}^{n} \alpha_i q(x_i), \tag{3.57}$$

kde $q(x) \in \Pi_{2n+1}$. Uvažujme ortogonální systém polynomů $p_0, p_1, \ldots, p_{n+1}$ a kořeny x_i polynomu p_{n+1}.

Polynom $q(x)$ můžeme vyjádřit následujícím způsobem

$$q(x) = p_{n+1}(x)s(x) + r(x), \tag{3.58}$$

kde $s \in \Pi_n$ je podíl po dělení polynomu $q(x)$ polynomem $p_{n+1}(x)$ a $r(x) \in \Pi_n$ je zbytek po tomto dělení.

Podle (3.4.3) můžeme polynomy $s(x)$ a $r(x)$ vyjádřit jako lineární kombinaci ortogonálních polynomů. Integrál $\int_a^b \omega(x)dx$ můžeme vyjádřit takto

$$\int_a^b \omega(x)q(x)dx = \int_a^b \omega(x)s(x)p_{n+1}(x)dx + \int_a^b \omega(x)r(x)dx. \tag{3.59}$$

Z vlastnosti ortogonálních polynomů uvedených ve větě 3.4.3 plyne, že polynom $p_{n+1}(x)$ je ortogonální ke všem polynomům nižšího stupně. Proto

$$\int_a^b \omega(x)q(x)dx = \underbrace{\int_a^b \omega(x)s(x)p_{n+1}(x)dx}_{=0} + \int_a^b \omega(x)r(x)dx, \tag{3.60}$$

a tedy

$$\int_a^b \omega(x)q(x)dx = \int_a^b \omega(x)r(x)dx. \tag{3.61}$$

Polynom $r(x)$ můžeme opět podle věty 3.4.3 vyjádřit jako lineární kombinaci ortogonálních polynomů. Položme

$$r(x) = \sum_{j=0}^{n} \beta_j p_j(x), \text{ kde } p_0(x) = 1. \tag{3.62}$$

Nyní vyjádříme druhý sčítanec

$$\begin{aligned} \int_a^b \omega(x)q(x)dx &= \int_a^b \omega(x) \sum_{j=0}^{n} \beta_j p_j(x)dx \\ &= \int_a^b \omega(x) \sum_{j=0}^{n} \beta_j p_j(x)p_0(x)dx \\ &= \sum_{j=0}^{n} \beta_j \int_a^b \omega(x)p_j(x)p_0(x)dx. \end{aligned} \tag{3.63}$$

Pokud využijeme větu 3.4.2, je druhý sčítanec, a tedy levá strana rovnosti (3.57) rovna

$$\beta_0 \int_a^b \omega(x)dx. \tag{3.64}$$

Zbývá vyjádřit pravou stranu rovnosti (3.57), která je prozatím vyjádřena ve tvaru

$$\sum_{i=0}^{n} \alpha_i q(x_i). \tag{3.65}$$

Jestliže využijeme vyjádření (3.58), pak pro pravou stranu rovnosti platí

$$\sum_{i=0}^{n} \alpha_i \left[p_{n+1}(x_i)s(x_i) + r(x_i) \right]. \tag{3.66}$$

Protože body x_i jsou kořeny polynomu $p_{n+1}(x)$, platí

$$\sum_{i=0}^{n} \alpha_i \left[\underbrace{p_{n+1}(x_i)s(x_i)}_{=0} + r(x_i) \right]. \tag{3.67}$$

S využitím (3.62) dostaneme rovnici

$$\beta_0 \int_a^b \omega(x)dx = \sum_{i=0}^{n} \alpha_i \sum_{j=0}^{n} \beta_j p_j(x_i), \tag{3.68}$$

kterou můžeme vyjádřit v maticovém tvaru:

$$\begin{pmatrix} p_0(x_0) & p_0(x_1) & \dots & p_0(x_n) \\ p_1(x_0) & p_1(x_1) & \dots & p_1(x_n) \\ \vdots & \vdots & & \vdots \\ p_n(x_0) & p_n(x_1) & \dots & p_n(x_n) \end{pmatrix} \begin{pmatrix} \alpha_0 \\ \alpha_1 \\ \vdots \\ \alpha_n \end{pmatrix} = \begin{pmatrix} \int_a^b \omega(x)dx \\ 0 \\ \vdots \\ 0 \end{pmatrix}. \tag{3.69}$$

Uzlové body x_i budeme proto nyní volit podle tvaru tzv. váhové funkce $\omega(x)$. Jedná se o kořeny polynomů, které jsou ortogonální s váhovou funkcí $\omega(x)$.

Vlastností, která je pro Gaussovu kvadraturu nejpodstatnější je, že pro stejný počet uzlových bodů je přesnější než výsledky získané metodami založenými na Newton - Cotesových vzorcích.

V rámci práce byl implementován Gaussův-Legendrův kvadraturní vzorec s váhovou funkcí $\omega(x) = 1$ na intervalu $\langle -1, 1 \rangle$.

Odvoďme Gauss-Legendrovu kvadraturu řádu 5 na intervalu $\langle -1, 1 \rangle$, přičemž budeme uvažovat první čtyři Legendrovy ortogonální polynomy $p_0 = 1, p_1 = x, p_2 = x^2 - \frac{1}{2}, p_3 = x^3 - \frac{3}{5}x$.

Podle věty 3.4.4 je kvadraturní formule Gaussova právě tehdy, když jako uzly kvadratury x_i pro $i = 0, 1, \dots, n$ volíme kořeny polynomu $p_{n+1}(x) \in \tilde{\Pi}_{n+1}$. Zjistíme proto kořeny polynomu $p_3 = x^3 - \frac{3}{5}x$. Tyto kořeny jsou $x_0 = 0, x_1 = -\sqrt{\frac{3}{5}}, x_2 = \sqrt{\frac{3}{5}}$. Integrál $\int_{-1}^{1} \omega(x)dx = 2$.

Nyní již můžeme sestavit maticovou rovnici tří rovnic o třech neznámých, jejímž řešením jsou koeficienty α_i kvadraturní formule.

$$\begin{pmatrix} 1 & 1 & 1 \\ 0 & -\sqrt{\frac{3}{5}} & \sqrt{\frac{3}{5}} \\ -\frac{1}{3} & \frac{4}{15} & \frac{4}{15} \end{pmatrix} \begin{pmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_1 \end{pmatrix} = \begin{pmatrix} 2 \\ 0 \\ 0 \end{pmatrix}.$$

Řešením této soustavy jsou koeficienty $\alpha_0 = \frac{8}{9}, \alpha_1 = \frac{5}{9}, \alpha_2 = \frac{5}{9}$. Dostáváme tak tříbodovou kvadraturní formuli

$$\int_{-1}^{1} f(x)dx \approx \frac{5}{9} f\left(-\sqrt{\frac{3}{5}}\right) + \frac{8}{9} f\left(0\right) + \frac{5}{9} f\left(\sqrt{\frac{3}{5}}\right), \tag{3.70}$$

která je na intervalu $\langle -1, 1\rangle$ přesná pro všechny polynomy stupně nejvýše pět.

Nicméně hodnoty uzlových bodů a koeficientů kvadraturního vzorce nemusíme počítat, protože jsou již zaznamenány v literatuře.

Hodnotu integrálu na intervalu $\langle a, b\rangle$ můžeme aproximovat podle následujícího složeného vztahu [6]

$$\int_{a}^{b} f(x)dx \approx \frac{h}{2} \sum_{j=0}^{m-1} \left(\sum_{i=0}^{n} \omega_i f\left(\frac{1}{2} x_i h + a + h\left(j + \frac{1}{2}\right)\right)\right), \tag{3.71}$$

kde ω_i jsou koeficienty Gaussova-Legendrova kvadraturního vzorce a x_i jsou kořeny tzv. *Legendrových polynomů*.

Počet uzlů n	*Uzly* x_i	*Váhy* ω_i
1	0	2
2	$\pm\frac{\sqrt{3}}{3}$	1
3	0	$\frac{8}{9}$
	$\pm\sqrt{\frac{3}{5}}$	$\frac{5}{9}$
4	$\pm\sqrt{\frac{3-2\frac{\sqrt{6}}{5}}{7}}$	$\frac{18+\sqrt{30}}{36}$
	$\pm\sqrt{\frac{3+2\frac{\sqrt{6}}{5}}{7}}$	$\frac{18-\sqrt{30}}{36}$
5	0	$\frac{128}{225}$
	$\frac{\pm 1}{3}\sqrt{5-2\sqrt{\frac{10}{7}}}$	$\frac{323+13\sqrt{70}}{900}$
	$\frac{\pm 1}{3}\sqrt{5+2\sqrt{\frac{10}{7}}}$	$\frac{323-13\sqrt{70}}{900}$

Tabulka 3.1: Uzly a váhy Gaussovy kvadratury

4 Nelineární rovnice

Ve čtvrté kapitole se budeme zabývat řešením nelineárních rovnic $f(x) = 0$, kde $x \in \mathbb{R}$. Budeme tedy hledat body, které jsou kořeny rovnice.

Numerické metody zabývající se touto problematikou jsou zpravidla iterační. Tyto metody generují posloupnost $\{x^k\}_{k=0}^{\infty}$. Proto jsou pro nás důležité informace o samotné konvergenci metody.

Definice 4.0.4. *Řekneme, že metoda konverguje k řešení rovnice $f(x) = 0$, jestliže platí*

$$\lim_{k\to\infty} x^k = \alpha, \tag{4.1}$$

kde $f(\alpha) = 0$.

Jestliže metoda konverguje k danému řešení, bude nás zpravidla zajímat, jak rychle tato metoda konverguje. V případě, že metoda diverguje nebo konverguje velice pomalu, musíme výpočet nějakým způsobem zastavit a nenechat jej pokračovat.

Věta 4.0.5. *(Bolzanova)*
Nechť funkce $f \in C[a, b]$ a nechť f nabývá v koncových bodech intervalu hodnot s opačnými znaménky, tj. $f(a)f(b) < 0$. Potom uvnitř tohoto intervalu existuje alespoň jeden bod α takový, že $f(\alpha) = 0$. V případě, že první derivace funkce f nemění na tomto intervalu znaménko, pak se zde nachází právě jeden takový bod.

4.1 Metoda půlení intervalu

Metoda půlení intervalů neboli bisekce je metoda, která je vždy konvergentní a do jisté míry i univerzální algoritmus, který se dá použít ve značné míře praktických případů. K užití této metody je zapotřebí splnit dvě podmínky. První podmínkou, kterou musí tato metoda splňovat je ta, že funkce f musí být spojitá na zvoleném počátečním intervalu $\langle x_0, x_1\rangle$. Druhým požadavkem je, aby funkční hodnoty v krajních bodech zvoleného intervalu měly opačná znaménka, tj. musí splňovat předpoklady Bolzanovy věty. Pokud jsou tyto podmínky splněny, pak tato metoda konverguje.

Půlením intervalu $\langle x_0, x_1\rangle$ zjistíme hodnotu x_2. Nyní musíme zjistit, ve kterém z nové vzniklých intervalů kořen leží. Je-li $f(x_2) = 0$, našli jsme kořen. V případě, že $f(x_0)f(x_2) < 0$, pak uvažujeme v dalším postupu výpočtu interval $\langle x_0, x_2\rangle$. Je-li $f(x_0)f(x_2) > 0$, pak nově uvažovaným intervalem je $\langle x_2, x_1\rangle$. Následně budeme celý postup znovu opakovat. Z poslední iterace tohoto postupu, dostaneme bod (střed

intervalu), který považujeme za aproximaci kořene zadané rovnice. Čím více iterací tedy provedeme, tím je výsledek přesnější. Střed intervalu určíme jako:

$$x^{k+1} = \frac{a^k + b^k}{2}, \tag{4.2}$$

kde a^k a b^k jsou krajní body intervalu uvažovaného v k-tém kroku. Je-li $f(x^{k+1}) = 0$, pak jsme nalezli kořen a výpočet ukončíme. Jinak určíme nový interval ve smyslu Bolzanova kritéria, tj. je-li $f(x^k)f(x^{k+1}) < 0$, budeme v další iteraci pracovat s intervalem $\langle a^{k+1}, b^{k+1}\rangle = \langle x^k, x^{k+1}\rangle$. V případě, že $f(x^k)f(x^{k+1}) > 0$, budeme dále uvažovat interval $\langle a^{k+1}, b^{k+1}\rangle = \langle x^{k+1}, x^{k-1}\rangle$. Ukončit metodu můžeme například ve chvíli, kdy šířka intervalu je menší než námi zadaná přesnost nebo když dvě po sobě jdoucí iterace jsou dostatečně shodné, tj. $|x^k - x^{k-1}| < \epsilon$. Více například v [29].

4.2 Metoda regula falsi

Regula falsi zvaná též metoda tětiv, je velice podobná metodě půlení intervalu. Stejně jako předchozí metoda vždy konverguje. Nicméně jako dělící bod intervalu neuvažuje střed intervalu, ale průsečík přímky spojující oba krajní body aktuálního intervalu s osou x (proto metoda tětiv). Tento průsečík vypočteme podle vztahu:

$$x^k = \frac{f(b_k)a_k - f(a_k)b_k}{f(b_k) - f(a_k)}. \tag{4.3}$$

Z nově vzniklých intervalů $\langle a^k, x^k\rangle$, $\langle x^k, b^k\rangle$ vybereme ten, který má v krajních bodech intervalu opačná znaménka a označíme ho $\langle a^{k+1}, b^{k+1}\rangle$ a podle Bolzanovy věty musí tedy obsahovat kořen rovnice. Výpočet zastavíme ve chvíli, kdy nalezneme kořen rovnice, nebo opět v případě dostatečné shodnosti dvou po sobě následujících iterací, tj. kdy $|x^k - x^{k-1}| < \epsilon$. Více informací lze nalézt například v [19].

4.3 Metoda sečen

Metoda sečen je podobná metodě regula falsi. Krajní body intervalu spojíme přímkou a nalezneme průsečík s osou x. Nyní ovšem nepostupujeme ve smyslu Bolzanovy věty, kdy bychom vybrali nový interval obsahující kořen, ale vedeme sečnu z bodu $[x_{k-1}, f(x_{k-1})]$ do bodu $[x_k, f(x_k)]$. Nyní opět nalezneme průsečík s osou x. Tímto způsobem pokračujeme do doby, než nalezneme kořen nebo $|x^k - x^{k-1}| < \epsilon$. Tento algoritmus lze charakterizovat vztahem:

$$x^{k+1} = x^k - \frac{x^k - x^{k-1}}{f(x^k) - f(x^{k-1})} f(x^k). \tag{4.4}$$

Více informací lze nalézt v [10].

4.4 Newtonova metoda

Newtonova metoda je metoda, která využívá k nalezení kořene rovnice tečny. Pro výpočet musíme zvolit počáteční aproximaci x^0. Novou aproximaci kořene x^{i+1} nalezneme jako průsečík tečny ke grafu funkce f v bodě $[x^i, f(x^i)]$ s osou x a opět sestrojíme tečnu. Využijeme faktu, že hodnoty první derivace funkce určují směrnici tečny. Metoda konverguje ve většině případů rychleji než metoda bisekce. Pro výpočet pomocí Newtonovy metody tečen se užívá vzorec vycházející z Taylorova rozvoje:

$$x^{k+1} = x^k - \frac{f(x^k)}{f'(x^k)} \ , \tag{4.5}$$

kde x^k je k-tá aproximace kořene.

Newtonova metoda je za určitých předpokladů kvadraticky konvergentní, a tudíž s každou novou iterací dojde ke zdvojnásobení počtu platných číslic.

Nevýhodou Newtonovy metody je ovšem skutečnost, že ne vždy konverguje. Musíme si uvědomit, že volba počáteční aproximace je velmi důležitým (ne však postačujícím) faktorem při hledání kořene rovnice. Pokud máme za úkol hledat kořen konvexní funkce, musíme počáteční aproximaci x^0 volit tak, aby $f(x_0) > 0$, pokud vyšetřujeme kořen rovnice pro konkávní funkce, musíme počáteční aproximaci x^0 volit tak, aby $f(x_0) < 0$. Další informace lze získat v [23], [24].

4.5 Steffensenova metoda

Tato metoda je velmi podobná Newtonově metodě tečen. Steffensenova metoda dosahuje kvadratické konvergence, ale na rozdíl od Newtonovy metody nevyužívá pro svůj výpočet derivace v bodě.

Metoda je definována následujícím předpisem:

$$x^{k+1} = x^k - \frac{f(x^k)}{d^k} \ , \tag{4.6}$$

kde $d^k = \frac{f(x^k + f(x^k)) - f(x^k)}{f(x^k)}$ je speciálně spočtená aproximace $f^{'}(x_k)$.

V každé iteraci se funkce $f(x)$ vyhodnocuje hned dvakrát a to konkrétně $f(x^k)$ a $f(x^k + f(x^k))$. Je tak vidět, že vyhnutí se výpočtu derivací přináší o jedno vyhodnocení navíc. Bližší informace lze nalézt v [25].

4.6 Halleyova metoda

Tato metoda je založena na aproximaci funkce Taylorovým polynomem druhého řádu, je tedy přesnější než Newtonova metoda, která je založena na aproximaci Taylorovým polynomem prvního řádu, a proto je její konvergence rychlejší. Ovšem za cenu nutnosti vypočtení i druhé derivace funkce v bodě.

Stejně jako u Newtonovy či Steffensenovy metody musíme vhodně zvolit počáteční aproximaci. Pro nalezení kořene rovnice využívá Halleyova metoda následujícího vztahu:

$$x^{k+1} = x^k - \frac{f(x^k)}{f'(x^k) - \frac{f''(x^k)f(x^k)}{2f'(x^k)}}. \tag{4.7}$$

Další informace jsou k nalezení například v [26].

4.7 Sturmova posloupnost

Sturmova posloupnost slouží pro stanovení počtu a lokalizaci reálných kořenů polynomu

$$P(x) = \sum_{i=0}^{n} a_i x^i, kde\ a_i \in \mathbb{R}\ ,\ x \in \mathbb{C}. \tag{4.8}$$

Definice 4.7.1. *Sturmova posloupnost pro polynom P na intervalu (a, b) je posloupnost polynomů $P_0(x) = P(x)$, $P_1(x) = -P'(x), \ldots, P_k(x) = konstanta, P_{i+1}(x) = -rem(P_{i-1}(x), P_i(x))$, kde rem je zbytek po dělení polynomů.*

Znakem $W(x)$ budeme značit počet znaménkových změn ve Sturmově posloupnosti $\{P_i(x)\}_{i=1}^{k}$ v bodě x.

Věta 4.7.1. *(Sturmova)*
Je-li $P(a)$ a $P(b)$ různé od nuly, je počet různých reálných kořenů polynomu $P(x)$ v intervalu $\langle a, b\rangle$ roven číslu $I_a^b = W(b) - W(a)$.

Konstruujeme posloupnost polynomů zmenšujících se stupňů, přičemž poslední člen posloupnosti $P_k(x)$ není roven nule a polynom $P(x)$ má pouze jednoduché kořeny. V opačném případě má polynom $P(x)$ vícenásobné kořeny. Pokud v tomto případě vydělíme polynom $P(x)$ předposledním členem posloupnosti $P_{k-1}(x)$, který je největším společným dělitelem polynomu $P_0(x)$ a $P_1(x)$, získáme polynom, jež má stejné kořeny jako polynom $P_0(x)$, ale všechny jednoduché. Toto plyne z Euklidova algoritmu. Následně znovu aplikujeme Sturmovu větu. Více například v [20] nebo [27].

5 Lineární rovnice

Soustavy lineárních rovnic jsou velice častou úlohou numerické matematiky vyskytující se například v teorii obvodů či statistice.

Předtím než charakterizujeme numerické metody pro výpočet řešení soustav lineárních rovnic, je nutné definovat některé pojmy, které budeme ve zbytku kapitoly využívat.

5.1 Základní pojmy

Definice 5.1.1. *Soustavu n lineárních rovnic o n neznámých budeme definovat*

$$\begin{array}{ccccccccc} a_{11}x_1 & + & a_{12}x_2 & + & \dots & + & a_{1n}x_n & = & b_1, \\ a_{21}x_1 & + & a_{22}x_2 & + & \dots & + & a_{2n}x_n & = & b_2, \\ \vdots & & & & & & \vdots & & \vdots, \\ a_{n1}x_1 & + & a_{n2}x_2 & + & \dots & + & a_{nn}x_n & = & b_n, \end{array} \tag{5.1}$$

kde $a_{ij}, b_i \in \mathbb{R}$ *pro* $i = 1, \dots, n$, $j = 1, \dots, n$.

Vektorově můžeme tuto soustavu zapsat:

$$\mathbf{A}\mathbf{x} = \mathbf{b}, \tag{5.2}$$

kde

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix}, \mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}, \mathbf{b} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}.$$

Matice $\mathbf{A}$ se nazývá matice soustavy, vektor x se nazývá vektor neznámých, vektor b se nazývá vektor pravé strany.

Definice 5.1.2. *Rozšířenou soustavu n lineárních rovnic o n neznámých budeme definovat*

$$\overline{\mathbf{A}} = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} & b_1 \\ a_{21} & a_{22} & \dots & a_{2n} & b_2 \\ \vdots & \vdots & & \vdots & \\ a_{n1} & a_{n2} & \dots & a_{nn} & b_n \end{pmatrix}. \tag{5.3}$$

Definice 5.1.3. *Hodnost matice* $\mathbf{A}$ *je rovna maximálnímu počtu nezávislých řádků. Hodnost matice značíme* $h(\mathbf{A})$.

Definice 5.1.4. *Nechť* $\mathbf{A} \in \mathbb{R}^{n,n}$. *Matici* $\mathbf{A}$ *nazveme dolní trojúhelníkovou maticí, pokud platí* $a_{ij} = 0$ *pro všechna* $i < j$.

Definice 5.1.5. *Nechť* $\mathbf{A} \in \mathbb{R}^{n,n}$. *Matici* $\mathbf{A}$ *nazveme horní trojúhelníkovou maticí, pokud platí* $a_{ij} = 0$ *pro všechna* $i > j$.

Definice 5.1.6. *Nechť* $\mathbf{A} \in \mathbb{R}^{n,n}$. *Pak* A *se nazývá regulární, pokud* $h(\mathbf{A}) = n$. *V opačném případě se nazývá singulární.*

Věta 5.1.1. *(Frobeniova)*
Je-li $h(\mathbf{A}) < h(\overline{\mathbf{A}})$, *pak soustava (5.1) nemá řešení.*
Je-li $h(\mathbf{A}) = h(\overline{\mathbf{A}}) = n$, *pak soustava (5.1) má právě jedno řešení.*
Je-li $h(\mathbf{A}) = h(\overline{\mathbf{A}}) = r < n$, *pak soustava (5.1) má nekonečné mnoho řešení závislých na* $n - r$ *parametrech.*

Definice 5.1.7. *Matice* $\mathbf{A}$ *se nazývá řádkově ostře diagonálně dominantní právě tehdy, když*

$$|a_{ii}| > \sum_{j=1, j\neq i}^{n} |a_{ij}|,\ i = 1, \ldots, n.$$

Definice 5.1.8. *Matice* $\mathbf{A}$ *se nazývá sloupcově ostře diagonálně dominantní právě tehdy, když*

$$|a_{jj}| > \sum_{i=1, i\neq j}^{n} |a_{ij}|,\ j = 1, \ldots, n.$$

Definice 5.1.9. *Je-li matice* $\mathbf{A} = (a_{ij})$ *typu* (n, m), *pak transponovaná matice k matici* $\mathbf{A}$ *je matice* $\mathbf{A}^T = (b_{ji})$ *typu* (m, n), *kde* $b_{ji} = a_{ij}$, $i = 1, \ldots, n,\ j = 1, \ldots, m$.

Definice 5.1.10. *Nechť* $\mathbf{A} \in \mathbb{R}^{n,n}$. *Matice* $\mathbf{A}$ *se nazývá symetrická právě tehdy, když platí* $\mathbf{A}^T = \mathbf{A}$.

Definice 5.1.11. *Matice* $\mathbf{A}$ *je pozitivně definitní, pokud pro libovolný nenulový vektor* $\mathbf{x}$ *platí*

$$\mathbf{x}^T \mathbf{A} \mathbf{x} > 0.$$

Věta 5.1.2. *Nechť* $\mathbf{A} \in \mathbb{R}^{n,n}$ *je symetrická pozitivně definitní matice. Potom následující vlastnosti jsou ekvivalentní.*

1. *Matice* $\mathbf{A}$ *je pozitivně definitní.*
2. *Všechna vlastní čísla všech submatic matice* $\mathbf{A}$ *jsou kladná.*
3. *Všechny hlavní minory (determinanty hlavních submatic) matice* $\mathbf{A}$ *jsou kladné.*
4. *Všechna vlastní čísla matice* $\mathbf{A}$ *jsou kladná.*

5. *Existuje regulární dolní trojúhelníková matice* $\mathbf{L}$ *tak, že* $\mathbf{A} = \mathbf{L}\mathbf{L}^{\mathbf{T}}$.

Definice 5.1.12. *Spektrem* $\sigma(\mathbf{A})$ *čtvercové matice* $\mathbf{A}$ *nazveme množinu všech vlastních čísel.*

Definice 5.1.13. *Spektrální poloměr matice* $\mathbf{A} \in \mathbb{R}^{n,n}$ *je číslo* $\rho(\mathbf{A}) = \max_{i=1,2,\ldots,n} |\lambda_i|$, *kde* λ_i, $i = 1, 2, \ldots, n$ *jsou vlastní čísla matice* $\mathbf{A}$.

Definice 5.1.14. *Nechť* $\mathbf{A} \in \mathbb{R}^{n,n}$. *Pak* $\mathbf{A}$ *se nazývá ortonormální, pokud* $\mathbf{A}\mathbf{A}^T = I$, *kde* $\mathbf{I}$ *je jednotková matice.*

Jestliže je matice soustavy regulární, tj. determinant matice je různý od nuly, má tato soustava pouze jediné řešení $\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$. Nicméně výpočet inverzní matice $\mathbf{A}^{-1}$ k matici $\mathbf{A}$ je velice časově a paměťově náročná operace, zejména pro rozsáhlé systémy rovnic. Z tohoto důvodu využíváme pro řešení soustav lineárních rovnic numerické metody, které lze rozdělit do dvou skupin. Více například v [18], [28] nebo [29].

5.2 Přímé metody

Princip přímých metod spočívá v převedení soustavy rovnic na ekvivalentní soustavu s trojúhelníkovou maticí. Použitím těchto metod získáme po konečném počtu elementárních algebraických úprav teoreticky přesné řešení. Toto řešení ovšem podléhá vlivu zaokrouhlovacích chyb, a proto nevypočteme obvykle přesné, ale pouze přibližné řešení. Chyba řešení je závislá na vlastnostech matice $\mathbf{A}$. Tyto vlastnosti souvisí s tzv. podmíněností matic. Podle [2] můžeme říci, že matice je dobře podmíněná, jestliže relativně malé změny prvků matice nebo vektoru pravé strany způsobí relativně malé změny v řešení. Naopak matice je označována jako špatně podmíněná, jestliže relativně malé změny prvků matice nebo vektoru pravé strany způsobí velké změny řešení.

5.2.1 Gaussova eliminace

Realizaci Gaussovy eliminace lze rozdělit do dvou na sebe navazujících kroků.

Přímý chod

Prvním krokem je přímý chod. Hlavní myšlenka přímého chodu spočívá v postupném odečítání rovnic, které jsou násobeny různými koeficienty až do stavu, kdy převedeme původní soustavu na ekvivalentní soustavu s trojúhelníkovou maticí:

$$a_{ij} = a_{ij} - \frac{a_{ik}}{a_{kk}} a_{kj},$$
$$b_i = b_i - \frac{a_{ik}}{a_{kk}} b_k,$$

kde $k = 1, \ldots, n-1, i = k+1, \ldots, n, j = k+1, \ldots, n$.

Zpětný chod

Druhým krokem je zpětný chod, který slouží k určování hodnot neznámých zadané soustavy lineárních rovnic. Z trojúhelníkového tvaru určíme hodnotu neznámé x_n přímo a zpětným dosazováním vypočtených neznámých jsme schopni určit hodnoty všech zbylých neznámých [30]:

$$x_i = \frac{1}{a_{ii}} \left(b_i - \sum_{j=i+1}^{n} x_j a_{ij} \right), \; i = n, \ldots, 1. \tag{5.4}$$

Realizovatelnost Gaussovy eliminace

Algoritmus Gaussovy eliminace je realizovatelný, jestliže matice $\mathbf{A}$ je sloupcově ostře diagonálně dominantní, tj. je-li absolutní hodnota diagonálního prvku větší než součet absolutních hodnot ostatních prvků ve sloupci. Dále můžeme říci, že Gaussova eliminace je realizovatelná tehdy, je-li matice $\mathbf{A}$ symetrická a pozitivně definitní, tj. všechna vlastní čísla matice $\mathbf{A}$ jsou kladná. Pro určení pozitivní definitnosti se v praxi často využívá Choleského rozklad, který popíšeme dále. Pokud matice $\mathbf{A}$ nesplňuje některý předpoklad z předchozího odstavce nelze obecně říci, že soustavu lineárních rovnic nelze řešit Gaussovou eliminací. K výpočtu řešení soustavy lineárních rovnic s libovolnou regulární maticí pomocí Gaussovy eliminace můžeme využít tzv. pivotaci.

Pokud se v průběhu přímého chodu vyskytne nulový diagonální prvek, nelze výpočet z důvodu dělení nulou provést. Druhý důvod použití pivotace plyne z numerické nestability, která je způsobena kumulací zaokrouhlovací chyby. Použitím pivotace se snažíme tuto chybu zmenšit.

Pivotaci lze realizovat v zásadě dvěma způsoby. Prvním způsobem je částečná pivotace, jejíž podstatu přiblížíme v následujícím odstavci.

V i-tém kroku Gaussovy eliminace nalezneme mezi i-tou až n-tou rovnicí rovnici, která má u neznámé x_i největší koeficient v absolutní hodnotě a následně provedeme záměnu pořadí této rovnice s i-tou rovnicí. Analogicky lze provádět záměnu pořadí sloupců.

Druhou variantou pivotace je úplná pivotace, která spočívá v tom, že v k-tém kroku volíme za hlavní prvek ten, který je největší v absolutní hodnotě v submatici vytvořené vynecháním prvních $k-l$ řádků a sloupců v upravené matici. Provádíme záměnu pořadí řádků a sloupců k-té rovnice s rovnicí, která má u neznámé x_k největší koeficient v absolutní hodnotě.

Je důležité si uvědomit, že nutnost sledovat největší prvek v celé submatici a následná záměna pořadí řádků a sloupců způsobuje časovou a paměťovou náročnost. Z tohoto důvodu je v aplikaci využita částečné pivotace, více viz [31].

Závěrem této podkapitoly upozorníme na náročnost Gaussovy eliminační metody, a to jak z časového, tak i paměťového hlediska. Gaussova eliminace se hodí nejlépe pro nepříliš rozsáhlé soustavy s plnou maticí nebo pro určité typy velkých řídkých matic (např. třídiagonální matice). Bližší informace můžeme nalézt například v [18], [32].

5.2.2 LU rozklad

Metoda **LU** dekompozice je vlastně variací Gaussovy eliminační metody. Matici **A** vyjádříme pomocí součinu dvou matic **L** a **U**, kde matice **L** je dolní trojúhelníková matice a matice **U** horní trojúhelníková matice. Místo soustavy $\mathbf{Ax} = \mathbf{b}$ budeme řešit tedy soustavy dvě:

$$\mathbf{Ly} = \mathbf{b}, \tag{5.5}$$

$$\mathbf{Ux} = \mathbf{y}. \tag{5.6}$$

Ovšem řešení těchto dvou soustav je vzhledem ke tvaru matic **L** a **U** jednodušší a rychlejší. Tento rozklad je zvláště výhodný pro řešení soustav se stejnou maticí, ale různými vektory pravé strany [32]. Matice **L** a **U** konstruujeme například podle následujících vztahů [10]:

$$l_{rr} = 1, \tag{5.7}$$

$$u_{11} = a_{11}, \tag{5.8}$$

$$l_{i1} = \frac{a_{i1}}{u_{11}}, \tag{5.9}$$

$$u_{1r} = a_{1r}. \tag{5.10}$$

Zbývající prvky určíme podle těchto vztahů:

$$u_{ir} = a_{ir} - \sum_{j=1}^{r-1} l_{ij}u_{jr},\ i = 2, \ldots, r, \tag{5.11}$$

$$l_{ir} = \frac{1}{u_{rr}} - \sum_{j=1}^{r-1} l_{ij}u_{jr},\ i = r+1, \ldots, n,\ r = 2, \ldots, n. \tag{5.12}$$

Poté můžeme řešit soustavy $\mathbf{Ly} = \mathbf{b}$ a $\mathbf{Ux} = \mathbf{y}$ tímto způsobem:

$$y_i = b_i - \sum_{k=1}^{i-1} l_{ik}y_k,\ i = 1, \ldots, n, \tag{5.13}$$

$$x_i = \frac{1}{u_{ii}}\left(y_i - \sum_{k=i+1}^{n} u_{ik}x_k\right),\ i = n, \ldots, 1.$$

Uvedený algoritmus lze podle [33] provést, jestliže se během výpočtu neobjeví nulový diagonální prvek. Pokud se tento prvek objeví, je nutné stejně jako u Gaussovy eliminace provést pivotaci. V opačném případě se součin **LU** nerovná matici **A**, ale matici **A** s permutovanými řádky, tj.

$$\mathbf{PA} = \mathbf{LU}. \tag{5.14}$$

Informace o pivotaci (záměně řádků) je zaznamenávaná do permutační matice **P**, jejíž základní tvar odpovídá jednotkové matici. Nyní budeme řešit soustavy:

$$\mathbf{Ly} = \mathbf{Pb}, \tag{5.15}$$

$$\mathbf{Ux} = \mathbf{y}.$$

5.2.3 Choleského rozklad

Jestliže je zadaná matice $\mathbf{A}$ symetrická a pozitivně definitní, můžeme matici $\mathbf{A}$ rozložit na součin dolní trojúhelníkové matice $\mathbf{L}$ a její transponované matice $\mathbf{L}^T$, tj.

$$\mathbf{A} = \mathbf{L}\mathbf{L}^{\mathbf{T}}. \tag{5.16}$$

Tím docílíme ušetření počtu aritmetických operací. Tento způsob výpočtu se nazývá metoda odmocnin, nebo také Choleského dekompozice. Matici $\mathbf{L}$ vypočteme podle následujících vztahů [10]:

$$l_{kk} = \sqrt{a_{kk} - \sum_{i=1}^{k-1} l_{ki}^2}, \tag{5.17}$$

$$l_{ij} = \frac{1}{l_{jj}} \left(a_{ij} - \sum_{k=1}^{j-1} l_{ik} l_{jk} \right). \tag{5.18}$$

Po vypočtení matice $\mathbf{L}$ budeme opět řešit dvě soustavy rovnic

$$\begin{aligned} \mathbf{L}^{\mathbf{T}}\mathbf{x} &= \mathbf{y}, \\ \mathbf{L}\mathbf{y} &= \mathbf{b}. \end{aligned} \tag{5.19}$$

5.3 Iterační metody

V případě iteračních metod konstruujeme posloupnost vektorů $\{\mathbf{x}^k\}_{k=0}^{\infty}$ takovou, že $\lim_{k\to\infty} \mathbf{x}^k = \mathbf{x}^*$, kde $\mathbf{x}^*$ je řešením $\mathbf{A}\mathbf{x} = \mathbf{b}$ a $\mathbf{x}^0$ je počáteční aproximace řešení. Vektory $\mathbf{x}^k$ konstruujeme pomocí předpisu

$$\mathbf{x}^{k+1} = \mathbf{T}\mathbf{x}^k + \mathbf{g}, \tag{5.20}$$

kde $\mathbf{T}$ je iterační matice a $\mathbf{g}$ vektor.

Pokud je $\mathbf{x}^* = \mathbf{A}^{-1}\mathbf{b}$ řešením soustavy $\mathbf{A}\mathbf{x} = \mathbf{b}$, pak požadujeme, aby platilo $\mathbf{x}^* = \mathbf{T}\mathbf{x}^* + \mathbf{g}$.

Jestliže tedy pro řešení soustavy lineárních rovnic $\mathbf{A}\mathbf{x} = \mathbf{b}$ využijeme iterační metody a splníme-li podmínky věty 5.3.1 a věty 5.3.2, dostáváme teoreticky přesné řešení po nekonečně mnoha iteracích. Důležité je tedy stanovení zastavovacího kritéria, kterým může být počet iterací nebo například $||\mathbf{x}^{k+1} - \mathbf{x}^k|| < \epsilon$, kde ϵ je požadovaná přesnost.

Při použití těchto metod si klademe v zásadě dvě otázky. Zda iterační proces konverguje a jak rychlá je jeho případná konvergence.

Věta 5.3.1. *(Nutná a postačující podmínka konvergence) Posloupnost* $\{\mathbf{x}^k\}_{k=0}^{\infty}$ *generovaná vztahem* $\mathbf{x}^{k+1} = \mathbf{T}\mathbf{x}^k + \mathbf{g}$ *konverguje k přesnému řešení* $\mathbf{x}^*$ *soustavy* $\mathbf{A}\mathbf{x} = \mathbf{b}$ *pro každou volbu počáteční aproximace* $\mathbf{x}^0$*, právě když spektrální poloměr iterační matice* $\mathbf{T}$ *je menší než 1, tj.* $\rho(\mathbf{T}) < 1$.

Věta 5.3.2. *(Postačující podmínka konvergence) Posloupnost* $\{\mathbf{x}^k\}_{k=0}^{\infty}$ *generovaná vztahem* $\mathbf{x}^{k+1} = \mathbf{T}\mathbf{x}^k + \mathbf{g}$ *konverguje k přesnému řešení* $\mathbf{x}^*$ *soustavy* $\mathbf{A}\mathbf{x} = \mathbf{b}$ *pro každou volbu počáteční aproximace* $\mathbf{x}^0$*, pokud* $||\mathbf{T}|| < 1$ *pro některou z maticových norem.*

Více informací lze nalézt v [32].

5.3.1 Jacobiho metoda

Princip této metody spočívá v rozložení matice $\mathbf{A}$ na součet

$$\mathbf{A} = \mathbf{E} + \mathbf{D} + \mathbf{F}, \tag{5.21}$$

kde $\mathbf{E}$ je ostře dolní trojúhelníková, $\mathbf{D}$ diagonální a $\mathbf{F}$ je ostře horní trojúhelníková matice. Jacobiho metodu můžeme vypočítat podle maticových nebo složkových vztahů:

$$\begin{aligned} \mathbf{A}\mathbf{x} &= \mathbf{b}, \\ (\mathbf{E}+\mathbf{D}+\mathbf{F})\mathbf{x} &= \mathbf{b}, \\ \mathbf{D}\mathbf{x} &= -(\mathbf{E}+\mathbf{F})\mathbf{x} + \mathbf{b}, \\ \mathbf{x} &= -\mathbf{D}^{-1}(\mathbf{E}+\mathbf{F})\mathbf{x} + \mathbf{D}^{-1}\mathbf{b}. \end{aligned} \tag{5.22}$$

Z poslední rovnosti získáme iterační předpis Jacobiho metody:

$$\mathbf{x}^{k+1} = -\mathbf{D}^{-1}(\mathbf{E}+\mathbf{F})\mathbf{x}^k + \mathbf{D}^{-1}\mathbf{b},\ k = 0, 1, \ldots. \tag{5.23}$$

Jacobiho metodu můžeme rozepsat po složkách:

$$x_i^{k+1} = \frac{1}{a_{ii}}\left(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^k - \sum_{j=i+1}^{n} a_{ij}x_j^k\right),\ i = 1, \ldots, n,\ k = 0, 1, \ldots. \tag{5.24}$$

Další podrobnosti lze nalézt v [2] nebo [19].

5.3.2 Gauss-Seidelova metoda

Princip Gauss-Seidelovy metody spočívá v transformaci soustavy lineárních rovnic $\mathbf{A}\mathbf{x} = \mathbf{b}$ na soustavu $(\mathbf{D}+\mathbf{E})\mathbf{x} + \mathbf{F}\mathbf{x} = \mathbf{b}$. Jednoduchých odvozením můžeme získat předpis Gauss-Seidelovy metody

$$\mathbf{x}^{k+1} = -(\mathbf{D}+\mathbf{E})^{-1}\mathbf{F}\mathbf{x}^k + (\mathbf{D}+\mathbf{E})^{-1}\mathbf{b},\ k = 0, 1, \ldots. \tag{5.25}$$

V maticovém zápisu můžeme vidět nutnost potřeby výpočtu inverzní matice. Tomu se můžeme vyhnout tímto způsobem:

$$\begin{aligned} (\mathbf{D}+\mathbf{E})\mathbf{x}^{k+1} &= -\mathbf{F}\mathbf{x}^k + \mathbf{b}, \\ \mathbf{D}\mathbf{x}^{k+1} &= -\mathbf{E}\mathbf{x}^{k+1} - \mathbf{F}\mathbf{x}^k + \mathbf{b}, \\ \mathbf{x}^{k+1} &= -\mathbf{D}^{-1}\mathbf{E}\mathbf{x}^{k+1} - \mathbf{D}^{-1}\mathbf{F}\mathbf{x}^k + \mathbf{D}^{-1}\mathbf{b}. \end{aligned} \tag{5.26}$$

Předpis po složkách má tvar:

$$x_i^{k+1} = \frac{1}{a_{ii}} \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{k+1} - \sum_{j=i+1}^{n} a_{ij} x_j^k \right), \; i = 1, \ldots, n, \; k = 0, 1, \ldots . \tag{5.27}$$

Z rovnic uvedených u Jacobiho a Gauss-Seidelovy metody můžeme vypozorovat, že v případě Jacobiho metody je nutné si pro výpočet další iterace pamatovat celý vektor $\mathbf{x}^k$, zatímco Gauss-Seidelova metoda využívá při výpočtu k-té aproximace neznámé x_i^k již vypočtené neznámé $x_j^k, j = 1, \ldots, i-1$.

U Jacobiho a Gauss–Seidelovy metody dochází ke konvergenci, pokud je matice soustavy ostře řádkové nebo sloupcově diagonálně dominantní. V případě Gauss-Seidelovy metody je konvergence zaručena i pro případ symetrických pozitivně definitních matic soustav.

Gauss – Seidelova metoda ve většině případů konverguje rychleji než Jacobiho metoda. Nicméně podle [18] existují případy, kdy Jacobiho metoda konverguje, zatímco Gauss – Seidlova metoda diverguje. Podrobnější informace nalezneme v [27].

5.3.3 Superrelaxační metoda

Superrelaxační metoda je variací Gauss-Seidelovy metody, která konverguje pro matice soustav, které jsou ostře řádkově nebo sloupcově diagonálně dominantní nebo pro symetrické pozitivně definitní matice. Obsahuje superrelaxační faktor ω, který ovlivňuje rychlost konvergence. Superrelaxační faktor klademe obvykle jako hodnotu náležící intervalu $\langle 1, 2)$. Ze vztahů uvedených níže můžeme vidět, že pro $\omega = 1$ je superrelaxační metoda shodná s Gauss-Seidelovou metodou. Důležitým faktorem konvergence této metody je správná volba ω, která může při vhodné volbě vést ke zrychlení konvergence (ke zrychlení nejčastěji volíme $\omega > 1$). V některých případech je ovšem vhodná volba $\omega < 1$, jež sice zpomalí konvergenci, ale zlepší stabilitu úlohy. Tato metoda je dána předpisem

$$\mathbf{x}^{k+1} = (\mathbf{D} + \omega\mathbf{E})^{-1}[(1-\omega)\mathbf{D} - \omega\mathbf{F}]\mathbf{x}^k + (\mathbf{D} + \omega\mathbf{E})^{-1}\omega\mathbf{b}. \tag{5.28}$$

Rozepíšeme-li předpis po složkách dostáváme

$$x_i{}^{k+1} = x_i^k + \omega(\tilde{x_i}{}^{k+1} - x_i^k), \tag{5.29}$$

kde

$$\tilde{x_i}{}^{k+1} = \frac{1}{a_{ii}} \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{k+1} - \sum_{j=i+1}^{n} a_{ij} x_j^k \right), \; i = 1, \ldots, n, \; k = 0, 1, \ldots$$

Více například v [29], [32], [34] .

5.4 Soustavy s obdélníkovou maticí

V následující části budeme uvažovat soustavy s obdélníkovou maticí.

Definice 5.4.1. *Soustava m lineárních rovnic o n neznámých je definována*

$$\begin{array}{ccccccccc} a_{11}x_1 & + & a_{12}x_2 & + & \dots & + & a_{1n}x_n & = & b_1, \\ a_{21}x_1 & + & a_{22}x_2 & + & \dots & + & a_{2n}x_n & = & b_2, \\ \vdots & & & & & & \vdots & & \vdots, \\ a_{m1}x_1 & + & a_{n2}x_2 & + & \dots & + & a_{mn}x_n & = & b_m, \end{array} \tag{5.30}$$

kde $a_{ij}, b_i \in \mathbb{R}$ pro $i = 1, \dots, m$, $j = 1, \dots, n$.

Z Frobeniovy věty 5.1.1 vyplývá, že soustava rovnic (5.30) má jediné řešení, jestliže hodnost matice $\mathbf{A}$ a hodnost rozšířené soustavy rovnic je rovna $min(m, n)$.

Zjistíme-li, že hodnost matice $\mathbf{A}$ a hodnost rozšířené soustavy rovnic je rozdílná, pak soustava (5.30) nemá řešení.

Soustava (5.30) má nekonečně mnoho řešení, jestliže hodnost matice $\mathbf{A}$ a hodnost rozšířené soustavy rovnic je menší než $min(m, n)$.

Definice 5.4.2. *Matice $\mathbf{A}^+$ se nazývá pseudoinverzní (Mooreova - Penroseova), jestliže splňuje:*

1. $\mathbf{A}^+\mathbf{A}\mathbf{A}^+ = \mathbf{A}^+$,
2. $\mathbf{A}\mathbf{A}^+\mathbf{A} = \mathbf{A}$,
3. $(\mathbf{A}^+\mathbf{A})^T = \mathbf{A}^+\mathbf{A}$,
4. $(\mathbf{A}\mathbf{A}^+)^T = \mathbf{A}\mathbf{A}^+$.

Jestliže je matice $\mathbf{A}$ regulární, pak je pseudoinverzní matice rovna matici inverzní.

5.4.1 Singulární rozklad

Věta 5.4.1. *Každou matici $\mathbf{A} \in \mathbb{R}^{m\times n}$ lze rozložit na součin*

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T, \tag{5.31}$$

kde $\mathbf{U} \in \mathbb{R}^{m\times m}$, $\mathbf{V} \in \mathbb{R}^{n\times n}$ jsou ortonormální matice a matice $\mathbf{\Sigma} \in \mathbb{R}^{m\times n}$ je diagonální,

$$\mathbf{\Sigma} = \operatorname{diag}(\sigma_1, \sigma_2, \dots, \sigma_k), \quad k = \min(m, n), \tag{5.32}$$

jejíž diagonální prvky splňují

$$\sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_r > \sigma_{r+1} = \dots = \sigma_k = 0, \tag{5.33}$$

kde $r = h(\mathbf{A})$.

Diagonální prvky $\sigma_1, \sigma_2, \ldots, \sigma_k$ matice Σ se nazývají singulární čísla matice $\mathbf{A}$. Sloupce matice $\mathbf{U}$, resp. matice $\mathbf{V}$ jsou ortonormální vlastní vektory matice $\mathbf{A}\mathbf{A}^T$ typu $[m, m]$, resp. matice $\mathbf{A}^T\mathbf{A}$ typu $[n, n]$.

Dále platí, že $\mathbf{A}\mathbf{A}^T$ a $\mathbf{A}^T\mathbf{A}$ jsou reálné čtvercové pozitivně semidefinitní matice, která mají stejná nenulová vlastní čísla. Tato vlastní čísla jsou proto nezáporná. Nenulová singulární čísla matice $\mathbf{A}$ jsou odmocniny z těchto vlastních čísel.

Nalezneme-li singulární rozklad matice, můžeme snadno vypočítat matici pseudoinverzní, neboť platí

$$\mathbf{A}^+ = \mathbf{V}\mathbf{\Sigma}^+\mathbf{U}^T, \tag{5.34}$$

kde $\mathbf{\Sigma}^+ \in \mathbb{R}^{n\times m}$ je diagonální matice

$$\mathbf{\Sigma}^+ = \operatorname{diag}\left(\frac{1}{\sigma_1}, \frac{1}{\sigma_2}, \ldots, \frac{1}{\sigma_r}, 0, \ldots, 0\right). \tag{5.35}$$

Řešení maticové rovnice $\mathbf{A}\mathbf{x} = \mathbf{b}$ poté nalezneme jako $\mathbf{x} = \mathbf{A}^+\mathbf{b}$, přičemž platí, že pokud soustava $\mathbf{A}\mathbf{x} = \mathbf{b}$ má nekonečně mnoho řešení, pak $\mathbf{x} = \mathbf{A}^+\mathbf{b}$ je řešení s nejmenší Euklidovou normou. V případě, že daná soustava řešení nemá, pak $\mathbf{x} = \mathbf{A}^+\mathbf{b}$ je nejlepší řešení ve smyslu nejmenších čtverců, tj. $\|\mathbf{b} - \mathbf{A}\mathbf{x}\| \leqq \|\mathbf{b} - \mathbf{A}\mathbf{y}\|$, kde $\mathbf{y} \in \mathbb{R}^n$. Má-li rovnice $\mathbf{A}\mathbf{x} = \mathbf{b}$ jediné řešení, pak $\mathbf{x} = \mathbf{A}^+\mathbf{b}$ je tímto řešením. Podrobnější informace lze nalézt v [19], [28], [32] nebo [35].

V rámci diplomové práci je singulární rozklad matice realizován pomocí QR rozkladu s Householderovou transformací.

6 Vlastní čísla a vlastní vektory matic

Numerické metody pro výpočet vlastních čísel a vlastních vektorů matice $\mathbf{A}$ můžeme rozdělit do dvou kategorií. Metody první kategorie se soustředí na tzv. částečný problém vlastních čísel matice $\mathbf{A}$, kdy se snažíme nalézt pouze určitou podmnožinu vlastních čísel vstupní matice (například vlastní číslo o největší resp. nejmenší absolutní hodnotě). Druhou kategorii tvoří metody, které se zabývají tzv. úplným problémem vlastních čísel. Použitím těchto metod se snažíme nalézt všechna vlastní čísla matice $\mathbf{A}$.

Předtím než se budeme zabývat numerickými metodami pro výpočet vlastních čísel a vlastních vektorů matice, shrneme některé základní pojmy, které budeme ve zbytku kapitoly využívat.

6.1 Základní pojmy

Definice 6.1.1. *Nechť $\mathbf{A} \in \mathbb{R}^{n,n}$. Číslo λ, pro které má homogenní soustava*

$$(\mathbf{A} - \lambda\mathbf{I})\,\mathbf{x} = \mathbf{0} \tag{6.1}$$

nenulové řešení, se nazývá vlastní číslo matice $\mathbf{A}$. Toto nenulové řešení $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ označujeme jako (pravý) vlastní vektor matice $\mathbf{A}$.

Z věty 5.1.1 plyne, že homogenní soustava má vždy alespoň jedno řešení (triviální). Jestliže je daná matice navíc singulární, má soustava nenulové řešení. Toto řešení je určeno $n - h(\mathbf{A})$ parametry, kde n je počet rovnic dané soustavy. Pro singulární matici soustavy platí, že její determinant je roven nule, což nás vede k následující definici.

Definice 6.1.2. *Levým vlastním vektorem $\mathbf{y}$ označujeme vektor, který je řešením maticové rovnice*

$$\mathbf{y}^T\mathbf{A} = \lambda\mathbf{y}^T. \tag{6.2}$$

Věta 6.1.1. *Levý vlastní vektor $\mathbf{y}$ je (pravým) vlastním vektorem transponované matice $\mathbf{A}^T$.*

Důkaz. Ze vztahu (6.2) plyne, že

$$\mathbf{y}^T\mathbf{A} = \lambda\mathbf{y}^T. \tag{6.3}$$

Transponujeme-li maticovou rovnici, pak dostáváme

$$(\mathbf{y}^T\mathbf{A})^T = (\lambda\mathbf{y}^T)^T, \tag{6.4}$$
$$\mathbf{A}^T\mathbf{y} = \lambda\mathbf{y} \tag{6.5}$$

a levý vlastní vektor je tedy zároveň pravým vlastním vektorem matice $\mathbf{A}^T$. □

Věta 6.1.2. *Levé a pravé vlastní vektory, které odpovídají různým vlastním číslům, jsou ortogonální.*

Důkaz. Uvažujme dvě vlastní čísla λ_1 a λ_2 a dva vlastní vektory $\mathbf{x}_1$, $\mathbf{x}_2$ příslušné k vlastnímu číslu λ_1, resp. λ_2. Předpokládejme, že $\lambda_1 \neq \lambda_2$.

Platí

$$\mathbf{A}\mathbf{x}_1 = \lambda_1\mathbf{x}_1 \tag{6.6}$$

a

$$\mathbf{A}\mathbf{x}_2 = \lambda_1\mathbf{x}_2. \tag{6.7}$$

Maticovou rovnici transponujeme

$$(\mathbf{A}\mathbf{x}_1)^T = (\lambda_1\mathbf{x}_1)^T \tag{6.8}$$
$$\mathbf{x}_1^T\mathbf{A} = (\lambda_1\mathbf{x}_1)^T. \tag{6.9}$$

Rovnici vynásobíme zprava vektorem $\mathbf{x_2}$

$$\mathbf{x}_1^T\mathbf{A}\mathbf{x}_2 = (\lambda_1\mathbf{x}_1)^T\mathbf{x}_2, \tag{6.10}$$
$$\mathbf{x}_1^T\lambda_2\mathbf{x}_2 = \lambda_1\mathbf{x}_1^T\mathbf{x}_2, \tag{6.11}$$
$$\lambda_2\mathbf{x}_1^T\mathbf{x}_2 = \lambda_1\mathbf{x}_1^T\mathbf{x}_2. \tag{6.12}$$

Kdyby $\mathbf{x}_1^T\mathbf{x}_2 \neq 0$, tak můžeme tímto číslem rovnici vydělit. Potom by ovšem $\lambda_1 = \lambda_2$, což je spor s předpokladem. Proto $\mathbf{x}_1^T\mathbf{x}_2 = 0$ a vlastní vektory příslušné různým vlastním číslům jsou ortogonální. □

Definice 6.1.3. *Vlastními čísly matice* $\mathbf{A}$ *jsou právě ta čísla* λ*, která jsou kořenem charakteristického polynomu*

$$p(\lambda) = det(\mathbf{A} - \lambda\mathbf{I}) = (-1)^n\lambda^n + b_1\lambda^{n-1} + \ldots + b_{n-1}\lambda + b_n. \tag{6.13}$$

Každá matice řádu n má tedy n vlastních čísel $\lambda_1, \lambda_2, \ldots, \lambda_n$, jestliže každé vlastní číslo počítáme tolikrát, jaká je jeho násobnost. Metody, které však počítají vlastní čísla jako kořeny charakteristického polynomu, se příliš nevyužívají. Hlavním důvodem je podle [32] výpočetní náročnost (zejména pro velké matice) a numerická nestabilita. Metody, které jsou v praxi využívány častěji, jsou založeny na podobnostní transformaci (připomeňme, že podobnostní transformace nemění vlastní čísla, viz věta 6.1.5), kdy se snažíme transformovat původní matici na matici podobnou, která je trojúhelníková, neboť platí následující věta.

Věta 6.1.3. *Vlastními čísly trojúhelníkové matice jsou prvky na její diagonále.*

Věta 6.1.4. *Vlastní vektory libovolné matice odpovídající různým vlastním číslům jsou lineárně nezávislé.*

Definice 6.1.4. *Čtvercové matice* $\mathbf{A}$ *a* $\mathbf{B}$ *nazýváme podobné, jestliže existuje regulární matice* $\mathbf{P}$ *taková, že platí*

$$\mathbf{P}^{-1}\mathbf{A}\mathbf{P} = \mathbf{B}. \tag{6.14}$$

Věta 6.1.5. *Podobné matice mají stejná vlastní čísla včetně jejich násobnosti.*

Důkaz. Jestliže λ je vlastní číslo matice $\mathbf{A}$ a $\mathbf{x}$ příslušný vlastní vektor, pak platí vztah (6.1). Uvažujme regulární matici $\mathbf{P}$. Ze vztahu (6.1) plyne

$$\mathbf{P}^{-1}\mathbf{A}\mathbf{x} = \lambda\mathbf{P}^{-1}\mathbf{x}. \tag{6.15}$$

Nechť $\mathbf{y} = \mathbf{P}^{-1}\mathbf{x}$. Z toho plyne, že $\mathbf{x} = \mathbf{P}\mathbf{y}$. Pokud dosadíme do vztahu (6.15), dostáváme

$$\mathbf{P}^{-1}\mathbf{A}\mathbf{P}\mathbf{y} = \lambda\mathbf{y}. \tag{6.16}$$

Číslo λ je tedy zároveň vlastním číslem $\mathbf{P}^{-1}\mathbf{A}\mathbf{P}$ a $\mathbf{y}$ je příslušný vlastní vektor. □

Obrácené tvrzení ale neplatí. Z rovnosti spekter matic tedy nevyplývá jejich podobnost. Uvažujme nyní matici $\mathbf{X}$, jejíž i-tý sloupec je vlastní vektor příslušný k vlastnímu číslu λ_i, a diagonální matici $\mathbf{\Lambda}$ s diagonálními prvky λ_1,$\lambda_2 \ldots$,λ_n.

Pak platí

$$\mathbf{A}\mathbf{X} = \mathbf{X}\mathbf{\Lambda} \tag{6.17}$$

$$\mathbf{X}^{-1}\mathbf{A}\mathbf{X} = \mathbf{\Lambda}. \tag{6.18}$$

Budou-li vlastní vektory matice $\mathbf{X}$ lineárně nezávislé, pak bude matice $\mathbf{X}$ regulární a matice $\mathbf{A}$ bude tedy podobná diagonální matici.

Tento případ podle [3] nastává, když má matice $\mathbf{A}$ všechna vlastní čísla různá nebo když je symetrická.

V diplomové práci jsme se soustředili na výpočet vlastních čísel a vlastních vektorů reálné symetrické matice, uvedeme proto některé vlastnosti těchto matic.

Věta 6.1.6. *Všechna vlastní čísla a k nim příslušné vlastní vektory symetrické matice jsou reálné.*

Věta 6.1.7. *Symetrická matice řádu n má právě n lineárně nezávislých vlastních vektorů.*

Věta 6.1.8. *Levé a pravé vlastní vektory symetrické matice příslušné stejným vlastním číslům si jsou rovny.*

Důkaz. Z definice 6.1.1 plyne, že levý vlastní vektor je pravým vlastním vektorem matice $\mathbf{A}^T$. Z definice symetrické matice $\mathbf{A}^T = \mathbf{A}$ plyne, že levý a pravý vlastní vektor si jsou rovny. □

Definice 6.1.5. *Dva vektory nazveme ortogonální, jestliže je jejich skalární součin roven nule.*

Věta 6.1.9. *Vlastní vektory symetrické matice odpovídající různým vlastním číslům jsou ortogonální.*

Důkaz. Plyne z věty 6.1.2 a z věty 6.1.8. □

Z předcházejících odstavců plyne, že pro symetrické matice můžeme nalézt podobnostní transformaci v následujícím tvaru

$$\mathbf{A} = \mathbf{X}\mathbf{\Lambda}\mathbf{X}^T, \tag{6.19}$$

kde sloupce matice $\mathbf{X}$ jsou tvořeny ortonormálními vlastními vektory matice $\mathbf{A}$ a $\mathbf{\Lambda}$ je diagonální matice s vlastními čísly matice $\mathbf{A}$. Více například v [15], [28], [36], [37], [38] nebo [39].

6.2 Částečný problém vlastních čísel

6.2.1 Mocninná metoda

Mocninná metoda slouží k určení vlastního čísla matice $\mathbf{A}$ o největší absolutní hodnotě (dominantní vlastní číslo).

Konstrukce mocninné metody

Předpokládejme, že matice $\mathbf{A}$ řádu n má n lineárně nezávislých vlastních vektorů a jediné dominantní vlastní číslo λ_1. Uspořádejme vlastní čísla sestupně

$$|\lambda_1| > |\lambda_2| \geqq |\lambda_3| \geqq \ldots \geqq |\lambda_n|. \tag{6.20}$$

Z předpokladu plyne, že libovolný vektor $\mathbf{x}^0 \in \mathbb{R}^n$ můžeme vyjádřit jako lineární kombinaci n lineárně nezávislých vlastních vektorů $\mathbf{v}_i, i = 1, \ldots, n$

$$\mathbf{x}^0 = \sum_{i=1}^{n} \gamma_i \mathbf{v}_i. \tag{6.21}$$

Nyní budeme konstruovat posloupnost

$$\mathbf{x}^{i+1} = \mathbf{A}\mathbf{x}^i, \ i = 0, 1, \ldots. \tag{6.22}$$

Pro $\mathbf{x}^{i+1}$ dostáváme

$$\mathbf{x}^{i+1} = \mathbf{A}\mathbf{x}^i = \mathbf{A}^2\mathbf{x}^{i-1} = \mathbf{A}^3\mathbf{x}^{i-2} = \ldots = \mathbf{A}^{i+1}\mathbf{x}^0, \ i = 0, 1, \ldots. \tag{6.23}$$

V dalším postupu využijeme následující větu.

Věta 6.2.1. *Jestliže λ je vlastním číslem matice $\mathbf{A}$ a $\mathbf{x}$ je příslušným vlastním vektorem, pak λ^2 je vlastním číslem matice $\mathbf{A}^2$ a $\mathbf{x}$ příslušným vlastním vektorem.*

Z předchozích vztahů a z rovnosti $\mathbf{A}\mathbf{x}_i = \lambda_i \mathbf{x}_i$ vyplývá, že

$$\mathbf{x}^k = \mathbf{A}^k \mathbf{x}^0 = \sum_{i=1}^{n} \gamma_i \mathbf{A}^k \mathbf{v}_i = \sum_{i=1}^{n} \gamma_i \lambda_i^k \mathbf{v}_i = \lambda_1^k \sum_{i=1}^{n} \gamma_i \left(\frac{\lambda_i}{\lambda_1}\right)^k \mathbf{v}_i. \tag{6.24}$$

Vyjdeme-li z předpokladu (6.20), pak platí

$$\lim_{k\to\infty} \left(\frac{\lambda_i}{\lambda_1}\right)^k = 0, \ i = 2, 3, \ldots, n \tag{6.25}$$

a

$$\lim_{k\to\infty} \mathbf{x}^k = \lambda_1^k \gamma_1 \mathbf{v}_1. \tag{6.26}$$

Ze vztahu (6.26) plyne, že vlastní vektor určený jako $\lim_{k\to\infty} \mathbf{x}^k$ se liší od vlastního vektoru $\mathbf{v}_1$ pouze multiplikativní konstantou. Připomeňme, že libovolný nenulový násobek vlastního vektoru je také vlastním vektorem dané matice.

Nyní vyvstává otázka, jak určíme aproximaci vlastního čísla λ_1. Pro j-tou složku vektoru $\mathbf{x}^k$ platí

$$x_j^k \approx \lambda_1^k \gamma_1 \mathbf{v}_{1_j}, \tag{6.27}$$

kde $\mathbf{v}_{1_j}$ je j-tá složka vektoru $\mathbf{v}_1$. Analogicky pro j-tou složku vektoru $\mathbf{x}^{k+1}$ platí

$$x_j^{k+1} \approx \lambda_1^{k+1} \gamma_1 \mathbf{v}_{1_j}. \tag{6.28}$$

Ze vztahu (6.27) vyjádříme λ_1^k

$$\lambda_1^k \approx \frac{x_j^k}{\gamma_1 \mathbf{v}_{1_j}}. \tag{6.29}$$

Vztah (6.28) rozepíšeme do následujícího tvaru

$$x_j^{k+1} \approx \lambda_1 \lambda_1^k \gamma_1 \mathbf{v}_{1_j}, \tag{6.30}$$

a pro aproximaci vlastního čísla platí

$$\lambda_1 \approx \frac{x_j^{k+1}}{x_j^k}. \tag{6.31}$$

Toto odvození můžeme zobecnit.

Věta 6.2.2. *Je-li* $\mathbf{y}$ *libovolný vektor, který není ortogonální k vlastnímu vektoru* $\mathbf{v}$*, pak pro aproximaci vlastního čísla* λ_1 *platí*

$$\lambda_1 = \frac{\mathbf{y}^T \mathbf{x}^{k+1}}{\mathbf{y}^T \mathbf{x}^k}. \tag{6.32}$$

V praxi často volíme vektor $\mathbf{y}$ tak, aby měl na pozici odpovídající největší složce vektoru $\mathbf{x}^{k+1}$ složku rovnu jedné a ostatní složky nulové. Aproximaci vlastního čísla λ_1 určíme tedy následujícím způsobem

$$\lambda_1 \approx \frac{\max\limits_{1\leq j\leq n} x_j^{k+1}}{\max\limits_{1\leq j\leq n} x_j^{k}}. \tag{6.33}$$

Při implementaci algoritmu mocninné metody je každá aproximace vlastního vektoru z důvodu omezené paměti (překročení datového rozsahu datového typu) počítače obvykle normalizována (například tak, aby byla největší složka vektoru rovna jedné). Jako aproximaci vlastního čísla λ_1 poté volíme největší složku vlastního vektoru nebo normu vlastního vektoru.

Z předcházejících odstavců plyne, že mocninná metoda je iterační metoda, jejíž rychlost konvergence závisí zejména na podílu $|\frac{\lambda_2}{\lambda_1}|$. Jestliže je tento podíl roven přibližně jedné, je konvergence mocninné metody pomalá. Je-li ovšem $|\lambda_1|$ o hodně větší než $|\lambda_2|$, pak je konvergence rychlá. Samotnou konvergenci může výrazně ovlivnit nevhodná volba počáteční aproximace $\mathbf{x}^0$. Volíme-li počáteční vektor například tak, že $\gamma_1 \approx 0$ a platí, že $|\lambda_2| > |\lambda_3|$, pak metoda konverguje k vlastnímu číslu λ_2.

Metodu ukončíme ve chvíli, je-li rozdíl dvou po sobě jdoucích aproximací vlastního čísla menší než zadaná přesnost nebo po m krocích algoritmu. Více v [15], [28] nebo [37].

6.2.2 Inverzní mocninná metoda

Jestliže chceme nalézt v absolutní hodnotě nejmenší vlastní číslo vstupní matice $\mathbf{A}$ a k němu příslušný vlastní vektor, můžeme použít inverzní mocninnou metodu. Opět vyjdeme z předpokladu, že matice $\mathbf{A}$ je řádu n, má n lineárně nezávislých vlastních vektorů a jediné dominantní vlastní číslo λ_1.

Dále předpokládejme, že pro vlastní čísla λ_i, $i = 0, 1, \ldots, n$, platí

$$|\lambda_1| \geqq |\lambda_2| \geqq |\lambda_3| \geqq \ldots > |\lambda_n| > 0, \tag{6.34}$$

tj. pro vlastní čísla inverzní matice $\mathbf{A}^{-1}$ platí

$$\left|\frac{1}{\lambda_n}\right| > \left|\frac{1}{\lambda_{n-1}}\right| \geqq \ldots > \left|\frac{1}{\lambda_1}\right| \geqq 0. \tag{6.35}$$

Věta 6.2.3. *Je-li λ nenulovým vlastním číslem matice $\mathbf{A}$ a vektor $\mathbf{x}$ je vlastním vektorem příslušný k tomuto vlastnímu číslu, pak $\frac{1}{\lambda}$ je vlastním číslem inverzní matice $\mathbf{A}^{-1}$ a vektor $\mathbf{x}$ je vlastním vektorem příslušným k tomuto vlastnímu číslu.*

Pomocí inverzní mocninné metody můžeme tedy určit vlastní číslo o největší absolutní hodnotě matice $\mathbf{A}^{-1}$, a tím zároveň v absolutní hodnotě nejmenší vlastní číslo matice $\mathbf{A}$. Další informace například v [32] nebo [40].

6.2.3 Rayleighův podíl

Metoda Rayleighova podílu je modifikací mocninné metody, která se využívá pro výpočet dominantního vlastního čísla symetrické matice a příslušného vlastního vektoru. Z věty 6.1.6 plyne, že vlastní čísla reálné symetrické matice jsou reálná. Pro normované vlastní vektory příslušné k různým vlastním číslům platí

$$\mathbf{v}_i^T \mathbf{v}_j = 0,\ i \neq j,\ \mathbf{v}_i^T \mathbf{v}_i = 1. \tag{6.36}$$

Aproximaci vlastního čísla určíme pomocí podílu

$$\lambda_1 = \frac{(\mathbf{x}^{k+1})^T \mathbf{x}^{k+1}}{(\mathbf{x}^k)^T \mathbf{x}^k}. \tag{6.37}$$

Konvergence této metody k řešení bude tedy zhruba dvakrát rychlejší než v případě mocninné metody. Další informace jsou uvedeny například v [32] nebo [40].

6.2.4 Hottelingova redukce

Často potřebujeme určit i další vlastní čísla a příslušné vektory. Tato vlastní čísla a vektory můžeme zjistit pomocí tzv. deflace.

Idea deflace spočívá v tom, že pro zjištění dalšího vlastního čísla a příslušného vlastního vektoru nahradíme původní matici redukovanou maticí, která má stejná vlastní čísla jako původní matice, kromě již spočteného vlastního čísla. Na tuto matici poté opět aplikujeme mocninnou metodu a zjistíme další dominantní vlastní číslo a příslušný vlastní vektor matice.

Věta 6.2.4. *(Maticová redukce) Nechť je λ_1 vlastní číslo matice $\mathbf{A}$, $\mathbf{x}_1$ jemu odpovídající vlastní vektor a $\mathbf{v}$ libovolný vektor, pro který platí $\mathbf{v}^T \mathbf{x}_1 = 1$. Pak redukovaná matice*

$$\mathbf{W} = \mathbf{A} - \lambda_1 \mathbf{x}_1 \mathbf{v}^T \tag{6.38}$$

má stejná vlastní čísla jako matice $\mathbf{A}$ kromě vlastního čísla λ_1, které je nahrazeno nulou.

Hotellingova redukce je vhodná pro reálné symetrické matice. Položme $\mathbf{v} = \mathbf{y_1}$, kde $\mathbf{y}_1$ je levý vlastní vektor příslušný vlastnímu číslu λ_1. Levý vlastní vektor normalizujeme tak, aby platilo $\mathbf{y}_1^T \mathbf{x}_1 = 1$. Z věty 6.1.8 vyplývá, že levé a pravé vlastní vektory příslušné stejným vlastním číslům symetrické matice jsou si rovny, tj.

$$\mathbf{y}_i = \mathbf{x}_i. \tag{6.39}$$

Ukážeme, že tato volba zajistí, že vlastní vektory $\mathbf{W}$ a $\mathbf{A}$ budou totožné

$$\mathbf{W} = \mathbf{A} - \lambda_1 \mathbf{x}_1 \mathbf{x}_1^T. \tag{6.40}$$

Vynásobme rovnici zprava vlastním vektorem $\mathbf{x}_i$

$$\mathbf{W}\mathbf{x}_i = \mathbf{A}\mathbf{x}_i - \lambda_1 \mathbf{x}_1 \mathbf{x}_1^T \mathbf{x}_i. \tag{6.41}$$

Z věty 6.1.9 plyne, že

$$\mathbf{x}_i^T \mathbf{x}_j = \begin{cases} 1 & i = j \\ 0 & i \neq j. \end{cases} \tag{6.42}$$

Položme $i = 1$. Potom

$$\begin{aligned} \mathbf{W}\mathbf{x}_1 &= \mathbf{A}\mathbf{x}_1 - \lambda_1 \mathbf{x}_1 \underbrace{\mathbf{x}_1^T \mathbf{x}_1}_{=1} \\ &= \lambda_1 \mathbf{x}_1 - \lambda_1 \mathbf{x}_1 \\ &= \mathbf{0}. \end{aligned} \tag{6.43}$$

Z toho plyne, že $\lambda_1 = 0$ a $\mathbf{x}_1$ je vlastním vektorem matice $\mathbf{A}$ a zároveň matice $\mathbf{W}$. Uvažujme $i \neq 1$, potom

$$\mathbf{W}\mathbf{x}_i = \mathbf{A}\mathbf{x}_i - \lambda_1 \mathbf{x}_1 \mathbf{x}_1^T \mathbf{x}_i = \mathbf{A}\mathbf{x}_i = \lambda_i \mathbf{x}_i. \tag{6.44}$$

Z poslední rovnosti vyplývá, že λ_i je vlastním číslem a $\mathbf{x}_i$ vlastním vektorem matice $\mathbf{A}$ i matice $\mathbf{W}$.

Pro reálnou symetrickou matici pak konstruujeme redukované matice následovně

$$\mathbf{W}_i = \mathbf{W}_{i-1} - \lambda_i \mathbf{x}_i \mathbf{x}_i^T, i = 1, \ldots, n-1, \mathbf{W}_0 = \mathbf{A}. \tag{6.45}$$

Další informace lze nalézt v [15] nebo [22].

6.3 Úplný problém vlastních čísel

Metody, které se zabývají úplným problémem vlastních čísel můžeme rozdělit do dvou kategorií. První kategorie je tvořena metodami, která určují vlastní čísla matice pomocí charakteristického polynomu. Nicméně je známo, že analyticky lze najít kořeny algebraické rovnice nejvýše čtvrtého stupně. Metody patřící do této skupiny (např. Krylovova nebo Le Verrierova metoda) určují kořeny charakteristického polynomu iteračně. Velkou nevýhodou těchto metod je výpočetní náročnost, a proto se v praxi příliš nevyužívají [3].

Metody druhé kategorie využívají podobnostní transformaci, která nemění vlastní čísla matice. Algoritmy této skupiny konstruují posloupnost navzájem podobných matic, která konverguje k matici (např. trojúhelníkové), jejíž vlastní čísla lze jednoduchým způsobem určit. V této podkapitole se budeme soustředit na problém nalezení všech vlastní čísel a příslušných vlastních vektorů reálné symetrické matice.

6.3.1 QR algoritmus

QR algoritmus je typickým představitelem metody založené na podobnostní transformaci. Tento algoritmus využívá *QR rozkladu*, pomocí kterého rozložíme matici na součin ortonormální matice $\mathbf{Q}$ a matice $\mathbf{R}$, která je horní trojúhelníková.

Věta 6.3.1. *(QR rozklad)*
Libovolnou matici $\mathbf{A}$ *typu* (m, n), $m \geq n$. *lze rozložit na součin* $\mathbf{A} = \mathbf{QR}$, *kde matice* $\mathbf{Q}$ *je ortonormální matice typu* (m, n) *a* $\mathbf{R}$ *je horní trojúhelníková matice typu* (n, n).

Ukážeme konstrukci QR algoritmu. Na počátku algoritmu položíme $\mathbf{A} = \mathbf{A_0}$ a provedeme QR rozklad

$$\mathbf{A}_0 = \mathbf{Q}_1\mathbf{R}_1. \tag{6.46}$$

Využijeme-li podobnostní transformaci, definujeme

$$\begin{aligned} \mathbf{A}_1 &= \mathbf{Q}_1^T\mathbf{A}_0\mathbf{Q}_1 \\ &= \mathbf{Q}_1^T\mathbf{Q}_1\mathbf{R}_1\mathbf{Q}_1. \end{aligned} \tag{6.47}$$

Protože matice $\mathbf{Q}_1$ je ortonormální, platí $\mathbf{Q}_1^T\mathbf{Q}_1 = \mathbf{I}$ a

$$\mathbf{A}_1 = \mathbf{R}_1\mathbf{Q}_1. \tag{6.48}$$

Matici $\mathbf{A}_1$ rozložíme opět pomocí QR rozkladu

$$\mathbf{A}_1 = \mathbf{Q}_2\mathbf{R}_2 \tag{6.49}$$

a postup opakujeme

$$\mathbf{A}_2 = \mathbf{Q}_2^T\mathbf{A}_1\mathbf{Q}_2 = \mathbf{Q}_2^T\mathbf{Q}_2\mathbf{R}_2\mathbf{Q}_2 = \mathbf{R}_2\mathbf{Q}_2. \tag{6.50}$$

Pro matici $\mathbf{A}_{k-1}$ tedy definujeme

$$\mathbf{A}_{k-1} = \mathbf{Q}_k\mathbf{R}_k. \tag{6.51}$$

Matici $\mathbf{A}_k$ můžeme rozepsat do následujícího vztahu

$$\begin{aligned} \mathbf{A}_k &= \mathbf{R}_k\mathbf{Q}_k \\ &= \mathbf{Q}_k^T\mathbf{A}_{k-1}\mathbf{Q}_k \\ &= \mathbf{Q}_k^T\mathbf{Q}_{k-1}^T\mathbf{A}_{k-2}\mathbf{Q}_{k-1}\mathbf{Q}_k \\ &= \vdots \\ &= \mathbf{Q}_k^T\mathbf{Q}_{k-1}^T\ldots\mathbf{Q}_1^T\mathbf{A}_0\mathbf{Q}_1\ldots.\mathbf{Q}_{k-1}\mathbf{Q}_k \\ &= (\mathbf{Q}_k\mathbf{Q}_{k-1}\ldots\mathbf{Q}_1)^T\mathbf{A}_0\mathbf{Q}_1\ldots.\mathbf{Q}_{k-1}\mathbf{Q}_k \\ &= (\mathbf{Q}_1\ldots\mathbf{Q}_{k-1}\mathbf{Q}_k)^T\mathbf{A}_0\mathbf{Q}_1\ldots.\mathbf{Q}_{k-1}\mathbf{Q}_k, \end{aligned} \tag{6.52}$$

kde matice součinu ortonormálních matic $\mathbf{Q}_1\ldots\mathbf{Q}_{k-1}\mathbf{Q}_k$ je opět ortonormální maticí.

Matice $\mathbf{A}_k$ konvergují k trojúhelníkové matici (v případě symetrické dokonce k diagonální). Zároveň můžeme pozorovat, že matice $\mathbf{A}_k$ je podobná matici $\mathbf{A}$.

Ze vztahu (6.19) vyplývá, že sloupce matice $\mathbf{Q}_1\ldots\mathbf{Q}_{k-1}\mathbf{Q}_k$ jsou ortonormálními vlastními vektory matice $\mathbf{A}_0$, více viz [15] nebo [32]. Nyní ukážeme dva způsoby, jak lze získat QR rozklad matice.

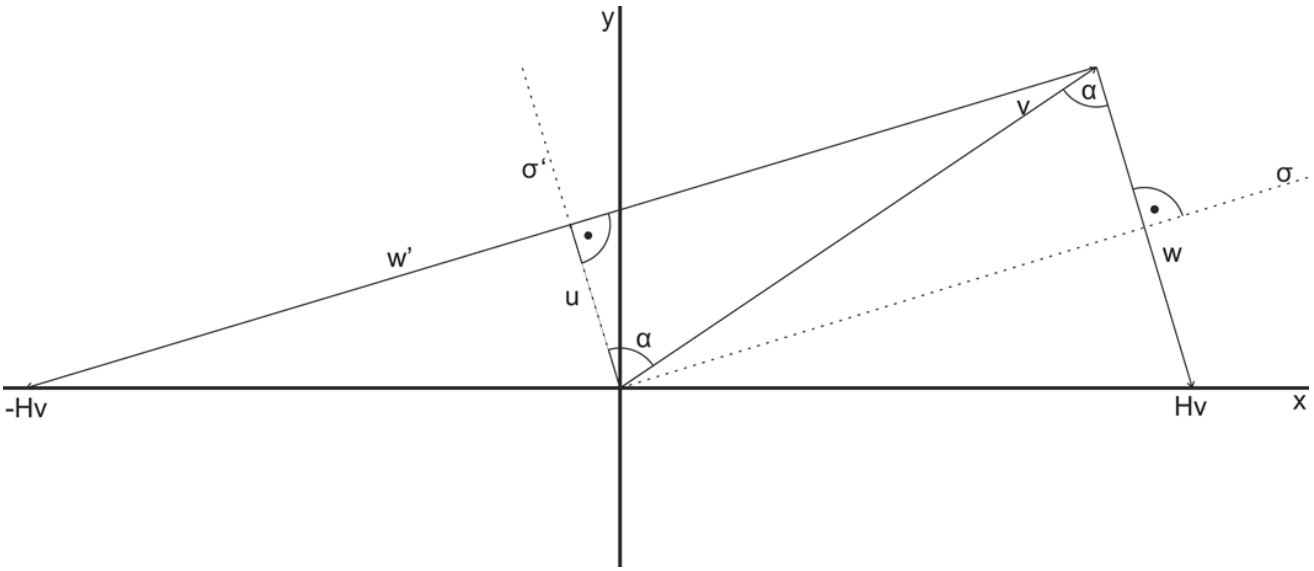
6.3.2 QR rozklad pomocí Householderovy transformace

Householderova transformace je postup pro získání horní trojúhelníkové matice, skládající se z $n-1$ transformačních kroků, kde n je počet sloupců matice. Princip tohoto algoritmu spočívá v postupném nulování prvků pod diagonálou, přičemž v k-tém transformačním kroku eliminujeme poddiagonální prvky v k-tém sloupci matice. K tomu využíváme Householderovu matici a Householderův vektor.

Odvození Householderovy matice zrcadlení

Cílem tohoto odvození je nalézt transformační matici, pomocí které zobrazíme vektor $\mathbf{v}$ na vektor $\mathbf{Hv}$, jenž je nenulovým násobkem vektoru $\mathbf{e}_1 = (1, 0, \ldots, 0)^T$. Přičemž požadavkem bude, aby délka vektoru $\mathbf{Hv}$ byla stejná jako délka vektoru $\mathbf{v}$, tj. $\|\mathbf{Hv}\| = \|\mathbf{v}\|$. Pro $\mathbf{Hv}$ tedy platí

$$\mathbf{Hv} = \pm\|\mathbf{v}\|\mathbf{e}_1. \tag{6.53}$$



Obrázek 6.1: Odvození Householderovy matice

Z obrázku 6.1 můžeme vypozorovat, že pro vektor $\mathbf{w}$ platí

$$\mathbf{w} = \mathbf{Hv} - \mathbf{v} = \pm\|\mathbf{v}\|\mathbf{e}_1 - \mathbf{v}. \tag{6.54}$$

Z ilustrace zároveň vyplývá, že

$$\mathbf{Hv} = \mathbf{v} - 2\mathbf{u}, \tag{6.55}$$

kde $\mathbf{w} = -2\mathbf{u}$.

Pomocí goniometrických funkcí vyjádříme $\mathbf{u}$ a dostaneme

$$\cos\alpha = \frac{\|\mathbf{u}\|}{\|\mathbf{v}\|}, \tag{6.56}$$

$$\frac{\|\mathbf{v}\|\cos\alpha}{\|\mathbf{u}\|} = 1. \tag{6.57}$$

Získaný vztah vydělíme nejprve Euklidovskou normou $\|\mathbf{u}\|$ a poté vynásobíme vektorem $\mathbf{u}$.

$$\frac{\mathbf{u}\|\mathbf{u}\|\|\mathbf{v}\|\cos\alpha}{\|\mathbf{u}\|^2} = \mathbf{u}. \tag{6.58}$$

Definice 6.3.1. *(Skalární součin vektorů)*
Skalárním součinem dvou vektorů délky n budeme rozumět vztah

$$\mathbf{u}^T\mathbf{v} = \sum_{i=1}^{n} u_i v_i = \|\mathbf{u}\|\|\mathbf{v}\|\cos\alpha, \tag{6.59}$$

kde úhel α je úhel sevřený vektory $\mathbf{u}$ a $\mathbf{v}$.

Využijeme-li definice skalárního součinu, můžeme psát

$$\frac{\mathbf{u}\mathbf{u}^T\mathbf{v}}{\|\mathbf{u}\|^2} = \mathbf{u}. \tag{6.60}$$

Protože platí $\|\mathbf{u}\|^2 = \mathbf{u}^T\mathbf{u}$, dostáváme

$$\frac{\mathbf{u}\mathbf{u}^T\mathbf{v}}{\mathbf{u}^T\mathbf{u}} = \mathbf{u}. \tag{6.61}$$

Zpětným dosazením za $\mathbf{u}$ do vztahu (6.55) a jednoduchou úpravou dostaneme výsledný tvar Householderovy matice zrcadlení $\mathbf{H}$. Připomeňme, že $\mathbf{u} = -\frac{\mathbf{w}}{2}$.

$$\mathbf{H}\mathbf{v} = \mathbf{v} - 2\mathbf{u} = \mathbf{v} - 2\mathbf{u}\frac{\mathbf{u}^T\mathbf{v}}{\|\mathbf{u}\|^2} \tag{6.62}$$

$$= \mathbf{v} - 2\frac{\mathbf{w}\mathbf{w}^T\mathbf{v}}{\|\mathbf{w}\|^2} = \mathbf{v} - 2\frac{\mathbf{w}\mathbf{w}^T\mathbf{v}}{\mathbf{w}^T\mathbf{w}} \tag{6.63}$$

$$= \left(\mathbf{I} - 2\frac{\mathbf{w}\mathbf{w}^T}{\mathbf{w}^T\mathbf{w}}\right)\mathbf{v}. \tag{6.64}$$

Aplikujeme-li Householderovu matici na vektor $\mathbf{v}$, získáme vektor $\mathbf{H}\mathbf{v}$, který má pouze první složku nenulovou.

Podle [41] je vhodné volit $\mathbf{w} = -sign(v_1)\|\mathbf{v}\|\mathbf{e}_1 - \mathbf{v}$. Touto volbou předejdeme zaokrouhlovacím chybám.

Definice 6.3.2. *(Householderova matice) Householderovou maticí budeme rozumět matici ve tvaru*

$$\mathbf{H}_k = \mathbf{I} - 2\frac{\boldsymbol{\omega}^k(\boldsymbol{\omega}^k)^T}{(\boldsymbol{\omega}^k)^T\boldsymbol{\omega}^k},\ k = 1, \ldots, n-1, \tag{6.65}$$

kde $(\boldsymbol{\omega}^k)^T\boldsymbol{\omega}^k = 1$. Vektor $\boldsymbol{\omega}^k$ budeme nazývat Householderovým vektorem.

Householderův vektor určíme následujícím způsobem. Definujme nejprve vektor

$$\boldsymbol{\chi}^k = -sign(v_{k_k})\|\mathbf{v}\|\mathbf{e}_k - \mathbf{v}_k, \tag{6.66}$$

kde $\mathbf{v}_k$ je k-tý sloupec matice $\mathbf{A}_k$ s prvními $k-1$ prvky nulovými. Vektor $\mathbf{e}_k$ je jednotkový vektor, pro který platí, že jeho k-tý prvek $e_{k_k} = 1$. Vektor $\boldsymbol{\omega}^k$ určíme jako

$$\boldsymbol{\omega}^k = \frac{\boldsymbol{\chi}^k}{\|\boldsymbol{\chi}^k\|_2}. \tag{6.67}$$

Věta 6.3.2. *Matice $\mathbf{H}_k$ je ortonormální a symetrická.*

Důkaz. Jestliže je matice $\mathbf{H}_k$ symetrická, pak $\mathbf{H}_k = \mathbf{H}_k^T$. Úpravou (6.65) dostaneme

$$\begin{aligned} \mathbf{H}_k^T &= \left(\mathbf{I} - 2\boldsymbol{\omega}^k(\boldsymbol{\omega}^k)^T\right)^T = \mathbf{I}^T - 2\left[\boldsymbol{\omega}^k(\boldsymbol{\omega}^k)^T\right]^T \\ &= \mathbf{I} - 2\left[\left(\boldsymbol{\omega}^k\right)^T\right]^T(\boldsymbol{\omega}^k)^T = \mathbf{I} - 2\boldsymbol{\omega}^k(\boldsymbol{\omega}^k)^T = \mathbf{H}_k. \end{aligned} \tag{6.68}$$

Nyní ukážeme, že matice $\mathbf{H}_k$ je ortonormální, tj. $\mathbf{H}_k^T\mathbf{H}_k = 1$. Platí

$$\begin{aligned} \mathbf{H}_k^T\mathbf{H}_k &= \left(\mathbf{I} - 2\boldsymbol{\omega}^k(\boldsymbol{\omega}^k)^T\right)^T\left(\mathbf{I} - 2\boldsymbol{\omega}^k(\boldsymbol{\omega}^k)^T\right) \\ &= \left(\mathbf{I} - 2\boldsymbol{\omega}^k(\boldsymbol{\omega}^k)^T\right)\left(\mathbf{I} - 2\boldsymbol{\omega}^k(\boldsymbol{\omega}^k)^T\right) \\ &= \mathbf{I} - 2\boldsymbol{\omega}^k(\boldsymbol{\omega}^k)^T - 2\boldsymbol{\omega}^k(\boldsymbol{\omega}^k)^T + 4\boldsymbol{\omega}^k(\boldsymbol{\omega}^k)^T\boldsymbol{\omega}^k(\boldsymbol{\omega}^k)^T \\ &= \mathbf{I} - 4\boldsymbol{\omega}^k(\boldsymbol{\omega}^k)^T + 4\boldsymbol{\omega}^k(\boldsymbol{\omega}^k)^T\boldsymbol{\omega}^k(\boldsymbol{\omega}^k)^T. \end{aligned} \tag{6.69}$$

Připomeňme, že $(\boldsymbol{\omega}^k)^T\boldsymbol{\omega}^k = 1$, a proto

$$\mathbf{H}_k^T\mathbf{H}_k = \mathbf{I} - 4\boldsymbol{\omega}^k(\boldsymbol{\omega}^k)^T + 4\boldsymbol{\omega}^k(\boldsymbol{\omega}^k)^T = \mathbf{I}. \tag{6.70}$$

□

Díky větě 6.3.2, resp. ortogonálnosti Householderovy matice, můžeme tuto ortogonální transformaci využít pro konstrukci QR rozkladu.

Budeme-li konstruovat posloupnost ve tvaru

$$\mathbf{A}_k = \mathbf{H}_k\mathbf{A}_{k-1}, \tag{6.71}$$

přičemž $k = 1, \ldots, n-1$ a $\mathbf{A}_0 = \mathbf{A}$, získáme po $n-1$ krocích horní trojúhelníkovou matici.

Platí tedy

$$\begin{aligned} \mathbf{A}_1 &= \mathbf{H}_1\mathbf{A}_0, \\ \mathbf{A}_2 &= \mathbf{H}_2\mathbf{A}_1 = \mathbf{H}_2\mathbf{H}_1\mathbf{A}_0, \\ \mathbf{A}_3 &= \mathbf{H}_3\mathbf{A}_3 = \mathbf{H}_3\mathbf{H}_2\mathbf{H}_1\mathbf{A}_0, \\ &= \vdots \\ \mathbf{A}_{n-1} &= \mathbf{H}_{n-1}\mathbf{A}_{n-2} = \mathbf{H}_{n-1}\mathbf{H}_{n-2}\ldots\mathbf{H}_2\mathbf{H}_1\mathbf{A}_0, \end{aligned} \tag{6.72}$$

kde $\mathbf{A}_{n-1}$ je horní trojúhelníková matice.

Položme $\mathbf{R} = \mathbf{A}_{n-1}$ a $\mathbf{Q}^T = \mathbf{H}_{n-1}\mathbf{H}_{n-2}\dots\mathbf{H}_1$. Matice $\mathbf{Q}^T$ je ortonormální, neboť je součinem ortonormálních matic $\mathbf{H}_i, i = 1,\dots,n-1$. Můžeme tedy psát následující rovnost

$$\mathbf{R} = \mathbf{Q}^T\mathbf{A}, \tag{6.73}$$

$$\mathbf{QR} = \mathbf{A}, \tag{6.74}$$

kde $\mathbf{Q} = \mathbf{H}_1\mathbf{H}_2\dots\mathbf{H}_{n-2}\mathbf{H}_{n-1}$. Vice například v [19], [44], [41], [42], [43], nebo [45].

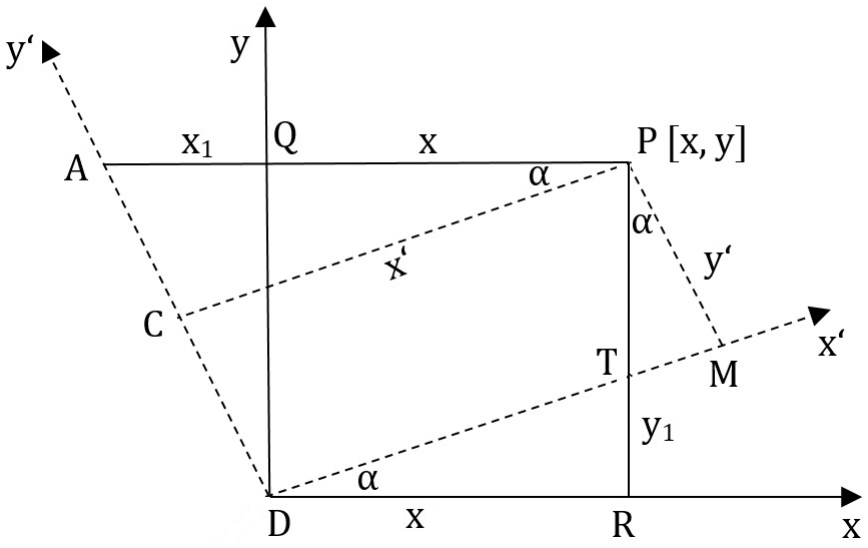
6.3.3 QR rozklad pomocí Givensovy transformace

Givensova transformace je další možnou volbou pro získání horní trojúhelníkové matice. Na rozdíl od Householderovy transformace ovšem eliminuje v každém transformačním kroku pouze jeden prvek pod diagonálou. To znamená, že k převodu matice na horní trojúhelníkovou matici je zapotřebí nejvýše $\frac{n(n-1)}{2}$ transformačních kroků (transformaci nebudeme aplikovat na již nulový prvek), kde n je počet sloupců matice. Givensovu transformaci pro převod matice do horního trojúhelníkového tvaru je tedy vhodné použít tehdy, má-li matice mnoho prvků pod diagonálou nulových. Obecně je ale tento postup výpočetně náročnější než předchozí metoda ortogonální transformace.

Givensova transformace využívá pro převod matice na horní trojúhelníkovou matici rovinné rotace.

Odvození Givensovy matice rovinné rotace

Odvodíme transformační matici pro nové souřadnice (x', y') bodu $P[x, y]$, který budeme rotovat o úhel α. Tento problém můžeme interpretovat podle [46] tak, že budeme hledat souřadnice bodu P s otočením soustavy souřadnic o stejný úhel.



Obrázek 6.2: Odvození matice rovinné rotace

Z trojúhelníku ACP na obrázku 6.2 můžeme vyjádřit

$$x' = (x_1 + x)\cos\alpha. \tag{6.75}$$

Využitím trojúhelníku DQA pro x' dostáváme

$$x_1 = y \operatorname{tg} \alpha. \tag{6.76}$$

Připomeňme vztah $\operatorname{tg}\alpha = \frac{\sin\alpha}{\cos\alpha}$. Dosazením do (6.75) získáme

$$x' = x\cos\alpha + y\sin\alpha. \tag{6.77}$$

Obdobným způsobem vyjádříme y'. Z trojúhelníku TMP vyjádříme y'

$$y' = (y - y_1)\cos\alpha. \tag{6.78}$$

Hodnotu y_1 získáme z trojúhelníku DRT

$$y_1 = x\tan\alpha = x\frac{\sin\alpha}{\cos\alpha}. \tag{6.79}$$

Dosazením do vztahu (6.78) získáme novou souřadnici y'

$$y' = \left(y - x\frac{\sin\alpha}{\cos\alpha}\right)\cos\alpha = -x\sin\alpha + y\cos\alpha. \tag{6.80}$$

Získali jsme tak soustavu

$$\begin{pmatrix} \cos\alpha & \sin\alpha \\ -\sin\alpha & \cos\alpha \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x' \\ y' \end{pmatrix}, \tag{6.81}$$

kde matici

$$\mathbf{G} = \begin{pmatrix} \cos\alpha & \sin\alpha \\ -\sin\alpha & \cos\alpha \end{pmatrix} \tag{6.82}$$

označujeme maticí rovinné rotace v $\mathbb{R}^2$.

Využitím Givensovy matice můžeme eliminovat prvky ve vektoru. Chceme-li například eliminovat druhou složku vektoru $\mathbf{x} = (x_1, x_2)$ pro $x_1 > 0$ a $x_2 > 0$, budeme postupovat tímto způsobem.

$$\mathbf{Gx} = \begin{pmatrix} \cos\alpha & \sin\alpha \\ -\sin\alpha & \cos\alpha \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} x_1\cos\alpha + x_2\sin\alpha \\ -x_1\sin\alpha + x_2\cos\alpha \end{pmatrix}, \tag{6.83}$$

přičemž požadujeme, aby

$$-x_1\sin\alpha + x_2\cos\alpha = 0. \tag{6.84}$$

Využijeme-li vztahu $\cos^2\alpha + \sin^2\alpha = 1$, dostáváme rovnost $\cos\alpha = \sqrt{1 - \sin^2\alpha}$. Dosazením do vztahu (6.84) a následným umocněním dostaneme

$$\begin{aligned} -x_1\sin\alpha + x_2\sqrt{1-\sin^2\alpha} &= 0 \\ x_2^2(1-\sin^2\alpha) &= x_1^2\sin^2\alpha \\ x_2^2 &= (x_1^2 + x_2^2)\sin^2\alpha \\ \frac{x_2^2}{(x_1^2+x_2^2)} &= \sin^2\alpha \\ \sin\alpha &= \frac{|x_2|}{\sqrt{x_1^2+x_2^2}}. \end{aligned} \tag{6.85}$$

Protože jsme předpokládali že $x_1 > 0$ a $x_2 > 0$, tak platí

$$\sin\alpha = \frac{x_2}{\sqrt{x_1^2 + x_2^2}}. \tag{6.86}$$

Hodnotu $\cos\alpha$ určíme dosazením do vztahu $\cos^2\alpha = 1 - \sin^2\alpha$

$$\cos\alpha = \frac{|x_1|}{\sqrt{x_1^2 + x_2^2}} = \frac{x_1}{\sqrt{x_1^2 + x_2^2}}. \tag{6.87}$$

Dosadíme-li do vztahu za $\cos\alpha$ a $\sin\alpha$, získáme pro vektor $\mathbf{Gx}$ tuto rovnost

$$\mathbf{Gx} = \begin{pmatrix} x_1\cos\alpha + x_2\sin\alpha \\ -x_1\sin\alpha + x_2\cos\alpha \end{pmatrix} = \begin{pmatrix} \frac{x_1^2}{\sqrt{x_1^2+x_2^2}} + \frac{x_2^2}{\sqrt{x_1^2+x_2^2}} \\ \frac{-x_1x_2}{\sqrt{x_1^2+x_2^2}} + \frac{x_2x_1}{\sqrt{x_1^2+x_2^2}} \end{pmatrix} = \begin{pmatrix} \|\mathbf{x}\| \\ 0 \end{pmatrix}. \tag{6.88}$$

Nyní uvažujme prostor $\mathbb{R}^n$. Zavedeme-li matici rotace v rovině ij, můžeme posloupností vhodně zvolených Givensových transformací eliminovat poddiagonální prvky vstupní matice a transformovat ji tak na horní trojúhelníkový tvar.

Definice 6.3.3. *Matice* $\mathbf{G}_{i,j,\alpha} \in \mathbb{R}^{n,n}$, $i < j$, *tvaru*

$$\mathbf{G}_{i,j,\alpha} = \begin{pmatrix} 1 & & & & & & & & \\ & \ddots & & & & & & & \\ & & 1 & & & & & & \\ & & & \cos\alpha & & & & \sin\alpha & \\ & & & & 1 & & & & \\ & & & & & \ddots & & & \\ & & & & & & 1 & & \\ & & & -\sin\alpha & & & & \cos\alpha & \\ & & & & & & & & \ddots \\ & & & & & & & & & 1 \end{pmatrix} \tag{6.89}$$

se nazývá Givensova matice rovinné rotace v rovině ij. Prvek $g_{ii} = g_{jj} = \cos\alpha$, prvek $g_{ij} = \sin\alpha$ a prvek $g_{ji} = -\sin\alpha$.

Podobně jakou u (6.86) a (6.87) položíme

$$\cos\alpha = \frac{a_{jj}}{\sqrt{(a_{jj})^2 + (a_{ij})^2}}, \quad \sin\alpha = \frac{a_{ij}}{\sqrt{(a_{jj})^2 + (a_{ij})^2}}, \tag{6.90}$$

$j = 1, \ldots, n-1$, $i = j + 1, \ldots, n$, a_{ij} i a_{jj} jsou prvky matice $\mathbf{A}$.

Pokud takto sestrojenou matici vynásobíme zleva maticí $\mathbf{A}$, vynulujeme prvek na pozici i, j, přičemž se změní pouze i-tý a j-tý řádek matice $\mathbf{A}$.

Věta 6.3.3. *Givensova matice rovinné rotace je ortonormální, tj.* $\mathbf{G}_{i,j,\alpha}^T \mathbf{G}_{i,j,\alpha} = \mathbf{I}$.

Důkaz. Vynásobíme-li matici $\mathbf{G}_{i,j,\alpha}^T$ maticí $\mathbf{G}_{i,j,\alpha}$, zjistíme, že
$g_{ii} = g_{jj} = \sin^2\alpha + \cos^2\alpha = 1$ (zbylé diagonální prvky jsou také rovny 1)
a $g_{ij} = g_{ij} = -\sin\alpha\cos\alpha + \sin\alpha\cos\alpha = 0$. Ostatní nediagonální prvky jsou také rovny nule. □

Této věty využijeme pro konstrukci QR rozkladu.

Při transformování matice $\mathbf{A}$ na horní trojúhelníkový tvar budeme postupovat tak, že nejprve eliminujeme prvky pod diagonálou prvního sloupce. Poté poddiagonální prvky druhého sloupce, až nakonec eliminujeme poddiagonální prvek $(n-1)$-ho sloupce. Touto volbou zajistíme, že se již eliminované prvky nestanou opět nenulovými. Budeme tedy konstruovat posloupnost v tomto tvaru

$$\mathbf{A}_k = \mathbf{G}_k\mathbf{A}_{k-1}, \quad \mathbf{A}_0 = \mathbf{A},\ k = 1,\ldots,m, \tag{6.91}$$

kde $\mathbf{G}_{k-1} = \mathbf{G}_{i,j,\alpha}$, přičemž nejvýše po $m = \frac{n(n-1)}{2}$ krocích získáme matici v horním trojúhelníkovou tvaru.

Položme $\mathbf{R} = \mathbf{G}_m\mathbf{G}_{m-1}\ldots\mathbf{G}_1\mathbf{A}$ a $\mathbf{Q}^T = \mathbf{G}_m\mathbf{G}_{m-1}\ldots\mathbf{G}_1$. Matice $\mathbf{Q}^T$ je ortonormální, neboť je součinem ortonormálních matic $\mathbf{G}_i, i = 1,\ldots,m$. Můžeme tedy psát následující rovnost

$$\mathbf{R} = \mathbf{Q}^T\mathbf{A} \tag{6.92}$$

$$\mathbf{QR} = \mathbf{A}, \tag{6.93}$$

kde $\mathbf{Q} = \mathbf{G}_1\mathbf{G}_2\ldots\mathbf{G}_{m-1}\mathbf{G}_m$. Podrobnější informace můžeme nalézt ve [41], [45], [47], nebo [48].

7 Implementace aplikace

V této části diplomové práce budeme popisovat samotnou naprogramovanou aplikaci, kde se nejprve seznámíme s jednotlivými vrstvami programu. Poté popíšeme jejich vzájemnou kooperaci při užívání aplikace. Na obrázku 7.1 můžeme vidět zjednodušený diagram znázorňující hierarchii jednotlivých vrstev aplikace a stěžejní třídy obsažené v těchto vrstvách, na nichž je založena celková funkcionalita aplikace. Dále diagram znázorňuje jednotlivé vztahy mezi třídami na úrovni vrstev a také popisuje jejich vzájemnou komunikaci.

7.1 Implementační vrstva aplikace

Implementační vrstva je nejnižší vrstvou aplikace, která obsahuje implementaci všech numerických metod popsaných v první části diplomové práce. Tyto algoritmy jsou rozděleny do tříd, přičemž každá třída obsahuje metody z jedné numerické oblasti.

Vzhledem k tomu, že hlavním cílem této práce bylo vytvoření didaktické aplikace, nebylo časově výhodné využít nějakou již existující numerickou knihovnu, protože tyto knihovny mají primárně za úkol předat uživateli pouze výsledek dané matematické úlohy. Vyvíjená aplikace bude nabízet uživateli i samotný postup výpočtu včetně mezivýsledků úlohy. Všechny implementované metody předávají na svém výstupu struktury dat, které jsou ve vyšších vrstvách za tímto účelem dále zpracovávány.

V této podkapitole si nebudeme klást za cíl popsat samotnou algoritmizaci příslušných numerických metod, ale uvedeme zde seznam všech implementovaných numerických metod obsažených v aplikaci a zaměříme se na množiny dat, které metody přijímají a na datové struktury, které metody předávají vyšším vrstvám, kde jsou nakonec uživateli interpretovány ve formě HTML stránky.

Nejnižší vrstva aplikace je navržena tak, aby došlo ke striktnímu oddělení vlastního výpočtu od jeho prezentace. Tato vrstva nespolupracuje pouze s nejvyšší vrstvou ale i s výpočetní mezivrstvou. Pokud je třeba zjistit hodnotu jakéhokoli výrazu nebo funkce, dochází k odeslání výrazu do mezivrstvy, kde dojde k vyčíslení daného výrazu. Více v podkapitole 7.2.

Implementaci metod zabývající se numerickým výpočtem určitého integrálu nalezneme ve třídě `Integration_alg`. Tato třída obsahuje:

1. Newton - Cotesovy vzorce

a) Jednoduché
   i. Obdélníkové pravidlo
   ii. Lichoběžníkové pravidlo
   iii. Simpsonovo pravidlo
   iv. Tříosminové pravidlo
   v. Booleovo pravidlo

b) Složené
   i. Obdélníkové pravidlo
   ii. Lichoběžníkové pravidlo
   iii. Simpsonovo pravidlo
   iv. Tříosminové pravidlo
   v. Booleovo pravidlo

2. Rombegova kvadratura

3. Gauss-Legendrova kvadratura

Metody numerické integrace přijímají jako argument řetězec obsahující funkci a integrační meze. Složené varianty Newton - Cotesových vzorců přijímají dále celočíselnou hodnotu, která určuje počet podintervalů, na který bude interval rozdělen a požadovanou přesnost. Struktura reprezentující výsledek získaný pomocí Newton - Cotesových vzorců a Gauss - Legendrovy kvadratury je jednotná. Metody předávají do nejvyšší vrstvy objekt třídy `Integration_ret`, který obsahuje informace o šířce uvažovaného integrálu, uzlových bodech, funkčních hodnotách a vypočtenou hodnotu integrálu. Jestliže uživatel zvolí v úvodní obrazovce aplikace složenou variantu Newton - Cotesových vzorců, pak je výpočet opakován metodou poloviční kroku do té doby, než je splněna požadovaná přesnost.

Výsledkem úlohy spočtené pomocí Rombergovy kvadratury je dvourozměrné pole hodnot obsahující hodnotu integrálu vypočteného pomocí složeného lichoběžníkového pravidla pro zjemňující se krok integrace. Výsledná data jsou předávána prezentační vrstvě, a to konkrétně formuláři `Integration_win`.

Druhá skupina algoritmů se zabývá metodami numerické derivace a nalezneme ji ve třídě `Derivation_alg`. Tato skupina metod je v zásadě pouze podpůrného charakteru pro další kolekci metod týkajících se řešení nelineárních rovnic. Nachází se zde proto základní metody, z nichž tři jsou určeny pro výpočet první derivace a zbylá pro výpočet druhé derivace funkce v bodě.

1. První derivace

   a) Centrální diference

   b) Přímá diference

   c) Richardsonova extrapolace

2. Druhá derivace

Numerické algoritmy přijímají z GUI vrstvy řetězec obsahující uživatelem zadanou funkci a dvě číselné hodnoty.

První hodnota předává metodě informaci o požadovaném bodu a druhá hodnotu kroku. Metoda Richardsonovy extrapolace přebírá navíc údaj s požadovanou přesností. Metody založené na diferencích odesílají formuláři `Derivation_win` objekt `Derivation_ret`, který s sebou nese číselnou informaci o příslušných uzlových bodech, jejich funkčních hodnotách, šíři kroku a výsledku nesoucí hodnotu první či druhé derivace funkce v bodě. Výjimku představuje metoda Richardsonovy extrapolace, která posílá dvourozměrné pole obsahující hodnoty aproximace derivace funkce v bodě.

Další skupinou jsou metody pro výpočet kořene nelineárních rovnic, které nalezneme v `Nelinear_solutions_alg`. Implementované algoritmy jsou uvedeny ve výčtu:

1. Metoda půlení intervalu
2. Metoda sečen
3. Regula falsi
4. Newtonova metoda
5. Steffensenova metoda
6. Halleyova metoda
7. Sturmova posloupnost

Všechny metody vyžadují na vstupu informace o funkci, přesnosti a počtu iterací. První tři metody z uvedeného seznamu dále požadují data týkající se dolní a horní meze uvažovaného intervalu. Zbylé tři na místo toho očekávají předání informace s počáteční aproximací. Metody jsou schopny přijmout předpis funkce první (v případě Halleyovy metody i druhé) derivace. Tato pole ale nejsou bezpodmínečně vyžadována. Pokud je ovšem uživatel vyplní, nedochází k numerické aproximaci derivace funkce v bodě pomocí metody centrální diference resp. druhé derivace z předchozí numerické kategorie, ale je vyčíslena předaná funkce v daném bodě. V případě Sturmovy posloupnosti jsou nepovinnými parametry informace o uvažovaném intervalu. Pokud je uživatel zadá, dochází k hledání všech reálných kořenů na tomto intervalu. V opačném případě jsou reálné kořeny hledány na intervalu $\langle -10^{10}, 10^{10} \rangle$, který simuluje množinu reálných čísel.

I při implementaci metod z této numerické oblasti byl ve výstupní části kladen důraz na jednotnost, a proto metody posílají formuláři `Nelinear_solutions_win` instanci třídy `Nelinear_ret`, která přenáší do vyšší vrstvy dva seznamy obsahující uzlové body a jejich funkční hodnoty. V případě Newtonovy metody a Steffensenovy metody je využit další číselný seznam pro předání hodnot první derivace funkce v bodě. Halleyova metoda využívá navíc i seznam pro přenos hodnot druhé derivace funkce v bodě.

Odlišná situace nastává v případě Sturmovy posloupnosti, která předává své výsledky formuláři ve formě instance třídy `SeparationOfRoots_ret`. Tato třída

udržuje informace o vygenerované posloupnosti polynomů, počtu znaménkových změn na daných intervalech. Dále udržuje informace o tvaru polynomu, který je největším společným dělitelem polynomu a jeho první derivace. Jestliže tento polynom není nulový, pak je nejvyšší vrstvě předána další Sturmova posloupnost nového polynomu, který má ale již jednoduché kořeny.

Čtvrtá kolekce metod slouží pro výpočet aproximace funkce. Jejich zdrojový kód je v `Aproximation_alg`.

Všechny metody přebírají na svém vstupu dva číselné seznamy, které obsahují uzlové body a jejich funkční hodnoty. Metoda nejmenších čtverců dále přijímá informaci o požadovaném stupni aproximačního polynomu. Na svém výstupu jednotlivé metody již formuláři `Aproximation_win` nepředávají stejnou kolekci dat, což je dáno zejména odlišností způsobu nalezení aproximující funkce. Z toho důvodu předává každá metoda nejvyšší vrstvě různou datovou strukturu ve formě *Tuple*, který obsahuje data nutná pro popis postupu výpočtu. V aplikaci jsou implementovány tyto metody:

1. Interpolace funkce
   a) Lagrangeův interpolační polynom
   b) Newtonův interpolační polynom
   c) Lineární spline
   d) Interpolace přirozeným kvadratickým spline
   e) Interpolace přirozeným kubickým spline
2. Aproximace funkce
   a) Metoda nejmenších čtverců

Všechny metody numerické aproximace funkce využívají při řešení zadané úlohy algoritmy pro práci s polynomy, které nalezneme ve třídě `Polynoms`. Jestliže uživatel požaduje výpočet úlohy metodou nejmenších čtverců nebo využitím metody implementující interpolaci přirozeným kubickým splinem, dochází při výpočtu k zavolání metody Gaussovy eliminace pro výpočet vzniklé soustavy lineárních rovnic.

Předposlední skupinu metod tvoří algoritmy týkající se řešení soustav lineárních rovnic, které můžeme rozdělit do dvou hlavních podkategorií.

1. Přímé
   a) Gaussova eliminace s částečnou pivotací
   b) LU rozklad
   c) Choleského dekompozice
2. Iterační
   a) Maticový tvar
      i. Jacobiho metoda

ii. Gauss-Seidelova metoda
iii. Superrelaxační metoda

b) Složkový tvar

i. Jacobiho metoda
ii. Gauss-Seidelova metoda
iii. Superrelaxační metoda

3. Řešení soustav s obdélníkovými maticemi

a) Singulární rozklad

Metody nalezneme ve třídě `Linear_solutions_alg`. Všechny algoritmy přijímají vstupní matici a vektor pravé strany. Iterační algoritmy dále vyžadují informace týkající se vektoru počáteční aproximace, maximálního počtu iterací v případě pomalé konvergence, resp. divergence metody a požadované přesnosti. Metoda určená pro výpočet LU rozkladu a metoda Choleského dekompozice poskytuje formuláři objekt `LinearEq_ret`. Tato instance obsahuje několik dvourozměrných polí - vstupní matici, matici $\mathbf{L}$ a $\mathbf{U}$, permutační matici $\mathbf{P}$ (v případě Choleského rozkladu není tato matice užita), vektor pravé strany $\mathbf{b}$, vektor $\mathbf{x}$ a $\mathbf{y}$. Metoda pro výpočet soustavy lineárních rovnic pomocí Gaussovy eliminace vrací datovou strukturu *Tuple*, přičemž pro zvýšení numerické stability této metody je v aplikaci implementována částečná pivotace.

V případech, kdy vstupní matice má stejnou hodnost jako rozšířená matice soustavy, ale menší než je řád matice, dochází u Gaussovy eliminace k výpočtu parametrického řešení.

Druhá podkategorie počítá řešení soustavy lineárních rovnic iteračně. Tyto algoritmy byly implementovány dvěma způsoby. První způsob počítá řešení soustavy lineárních rovnic pomocí maticového zápisu a druhý pomocí složkového zápisu.

Iterační metody implementované pomocí složkového zápisu odesílají formuláři `Linear_solutions_win` jednoduchý objekt `LinearniEqFolder_ret`, který obsahuje dvourozměrná pole (vstupní matici, vektor $\mathbf{b}$ a vektor $\mathbf{x}$), seznam vektorů, kde každý vektor je výsledkem jedné iterace, a hodnot, které nesou informaci o konvergenci metody pro danou úlohu (Euklidovská norma rozdílu dvou po sobě jdoucích vektorů).

Iterační metody hledající řešení soustavy lineárních rovnic maticovým zápisem posílají do nejvyšší vrstvy datovou strukturu *Tuple*, která obsahuje seznam matic nutných pro zobrazení mezivýsledků, seznam vektorů získaných každou iterací a seznam hodnot, které budou uživatele informovat o konvergenci, respektive divergenci metody pro danou úlohu.

Pro výpočet soustav rovnic s obdélníkovou maticí je v aplikaci implementován singulární rozklad. Metoda implementující postup singulárního rozkladu přijímá na svém vstupu vstupní matici a vektor pravé strany. Cílem singulárního rozkladu je rozložit matici na součin tří matic $\mathbf{U}$, $\mathbf{\Sigma}$, $\mathbf{V}^T$. Během samotného výpočtu singulárního rozkladu je využíván QR algoritmus s tisíci iteracemi, kde QR rozkladu matice je dosaženo pomocí Housholderovy transformace s jehož pomocí získáme potřebné matice $\mathbf{U}$ a $\mathbf{V}$. Diagonální matice $\mathbf{\Sigma}$ bude obsahovat odmocniny ze zjištěných

nenulových vlastních čísel. Po nalezení singulárního rozkladu dojde nejprve k výpočtu pseudoinverzní matice a poté k nalezení příslušného řešení. Tento algoritmus předává prezentační vrstvě objekt třídy `SVD_ret`, který obsahuje všechny matice a vektory získané během výpočtu (matice $\mathbf{A}$, $\mathbf{A}\mathbf{A}^T$, $\mathbf{A}^T\mathbf{A}$, $\mathbf{U}$, $\mathbf{V}$, $\mathbf{\Sigma}$, $\mathbf{\Sigma}^+$, $\mathbf{A}^+$, vektor $\mathbf{x}$ a vektor pravé strany) a které jsou uživateli interpretovány v prezentační vrstvě ve formě HTML stránky.

Poslední skupina metod se věnuje výpočtu vlastních čísel a vlastních vektorů reálné symetrické matice. Implementované algoritmy můžeme rozdělit následujícím způsobem:

1. Částečný problém vlastních čísel
   a) Mocninná metoda
   b) Rayleighův podíl
   c) Inverzní metoda
   d) Hottelingova redukce (Mocninná metoda)
   e) Hottelingova redukce (Rayleighův podíl)
2. Úplný problém vlastních čísel
   a) QR algoritmus
      i. Householderova transformace
      ii. Givensova transformace

Metody věnující se částečnému problému vlastních čísel a vlastních vektorů matice přebírají na vstupu dvourozměrné pole obsahující vstupní matici, počáteční vektor, maximální počet iterací a požadovanou přesnost. První tři metody předávají posloupnost vypočtených vlastních čísel, vlastních vektorů a normovaných vlastních vektorů. Hottelingova redukce odesílá k zobrazení do prezentační vrstvy (formuláři `Eigen_values_vectors_win`) posloupnost redukovaných matic a kolekci dat z mocninné metody, respektive Rayleighova podílu.

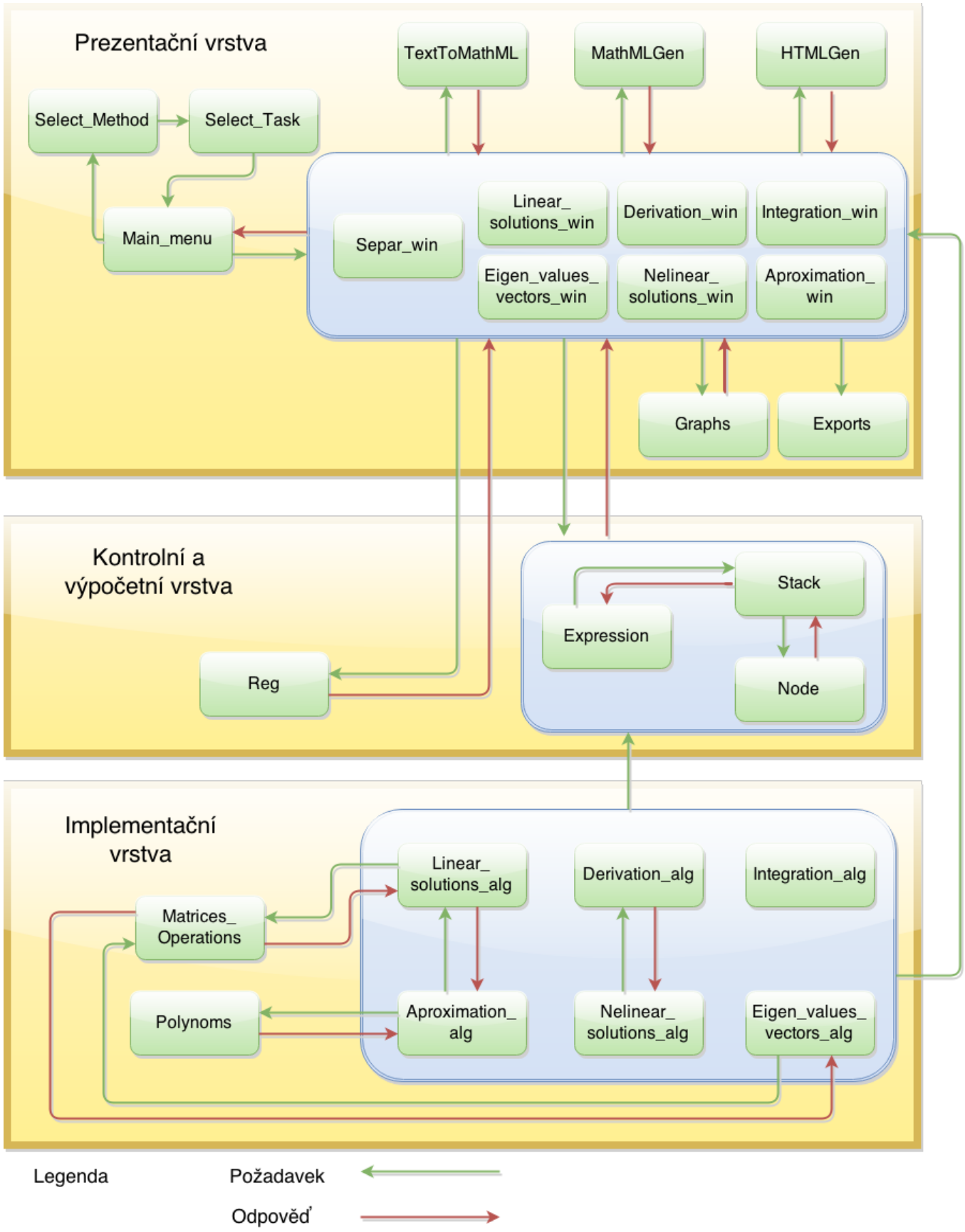
Ač řadíme Hottelingovu redukci mezi první skupinu metod, které naleznou největší či nejmenší dominantní vlastní číslo matice, je tato metoda v aplikaci naprogramována tak, že opakováním algoritmu redukce matice nalezneme všechna vlastní čísla a vlastní vektory původní matice.

Metody druhé kategorie jsou založeny na podobnosti matic. K nalezení podobné matice je v aplikaci naprogramován QR algoritmus. Příslušný QR rozklad matice je nalezen Householderovou nebo Givensovou transformací, který je odeslán metodě *QRAlgorithmHouseHolder* resp. *QRAlgorithmGivens*. Pokud budeme celý tento postup opakovat v rámci QR algoritmu, bude celý proces konvergovat k matici podobné původní matici s vlastními čísly na diagonále.

Metody *QRAlgorithmHouseHolder* resp. *QRAlgorithmGivens* předávají nejvyšší vrstvě instanci třídy `QRRozklad_RetHouseHolder` respektive `QRRozklad_RetGiven`, které obsahují seznamy matic $\mathbf{Q_k}, \mathbf{R_k}$ a $\mathbf{A_k}$, dále posloupnost transformačních matic využitých v každé iteraci a nakonec seznam vlastních čísel a vlastních vektorů.

Během výpočtu může dojít k různým chybám, například z důvodu dělení nulou, k porušení Bolzanova kritéria v případě metod, které se zabývají řešením nelineárních rovnic,nebo například ke zjištění, že uživatelem zadaná vstupní matice je singulární (při výpočtu inverzní matice). Z toho důvodu jsou v každé třídě nejnižší vrstvy aplikace implementovány delegáty, které při zjištění tohoto stavu, okamžitě informují prezentační vrstvu. GUI vrstva přeruší výpočet běžící na samostatném vlákně a pomocí dialogového okna informuje o tomto stavu uživatele.

Zbylé třídy implementační vrstvy (`Polynoms` a `Operation_of_Matrices`) obsahují metody pro počítání s polynomy (násobení, sčítání, odčítání, derivace polynomu, Hornerovo schéma pro vyčíslení hodnoty polynomu v bodě nebo například Euklidův algoritmus pro zjištění největšího společného dělitele polynomů) resp. metody pro operace s maticemi (kupříkladu násobení, sčítání, odčítání matic, hodnost a defekt matice, algoritmus pro výpočet inverzní matice či algoritmus pro zjištění Euklidovské normy vektorů a matic nebo hodnoty determinantu matice).



Obrázek 7.1: Zjednodušený diagram návrhu aplikace

7.2 Kontrolní a výpočetní vrstva

Kontrolní a výpočetní vrstva je zavedena do modelu aplikace z několika důvodů. Prvním důvodem je, že kromě základní kontroly uživatelem zadaných dat na úrovni GUI vrstvy dochází pomocí metod prostřední vrstvy ke kontrole vstupních dat ze sémantického hlediska (je-li například vstupní funkce zadána korektně nebo zda uživatel do vstupního pole umístil správně závorky) a navíc je zde také implementován *Shunting-Yard* algoritmus pro převedení výrazu z infixové notace (klasický způsob matematického zápisu výrazu, ve kterém jsou operátory napsány mezi operandy) do reverzní polské notace (RPN). Reverzní polská notace je způsob zápisu matematického výrazu, ve kterém jsou operátory umístěny za operandy (postfixový zápis). Výhoda tohoto zápisu spočívá v odstranění nutnosti využití závorek při zápisu výrazu. Dále je zde implementována metoda pro vyčíslení takového postfixového výrazu. Tyto dvě metody jsou využívány vždy, pokud je pro výpočet vyžadována funkční hodnota funkce v určitém bodě nebo například hodnota určitého algebraického výrazu. Ke kooperaci s touto vrstvou dochází i směrem od prezentační vrstvy, protože tyto metody jsou využity kupříkladu pro převod uživatelského vstupu (zadání funkcí, mezí intervalu, přesností, počátečních hodnot, analyticky zadaných prvních nebo druhých derivací, prvků matic a vektorů, uzlových bodů a dalších vstupních dat) do formátu, se kterým jsou metody implementační vrstvy schopny pracovat. Provázanost vrstev se dále využívá pro převod vstupních dat zadaných ve formě neformátovaného textu do MathML kódu nebo pro vykreslování grafů.

Metody *RPN* a *ComputeRPN* jsou nejčastěji vykonávanými metodami v celé aplikaci.

7.2.1 Parser

Kontrolní a výpočetní vrstva nacházející se uprostřed modelu aplikace obsahuje metody sloužící mimo jiné i pro kontrolu správnosti zadaných dat uživatelem. Tyto metoda jsou implementovány pomocí regulárních výrazů ve třídě `Reg`.

Parser pro ověření správnosti uživatelského vstupu pracuje ve čtyřech krocích. Prvním z nich je příprava pro ověřování, která spočívá v naplnění dvou seznamů regulárními výrazy. První seznam ověřuje správnou posloupnost znaků (například, že za znaménkem pro násobení následuje závorka, číslo, funkce nebo neznámá). Do druhého seznamu se postupně přidávají regulární výrazy, které také ověřují, zda je vstup korektní či nikoliv. Rozdíl těchto seznamů spočívá v tom, že první seznam pouze ověřuje, zdali nějaká část vstupního řetězce odpovídá regulárnímu výrazu, naproti tomu druhý seznam ze vstupního řetězce vyjímá řetězce odpovídající danému regulárnímu výrazu. Po naplnění obou seznamů se vstupní řetězec nejprve porovná s výrazy prvního seznamu. Jestliže je nalezena shoda, tak algoritmus ověřování končí, protože uživatelský vstup není validní. Dále následuje ověření uživatelského vstupu pomocí regulárních výrazů druhého seznamu. Jestliže je nalezena shoda, tak je nalezený řetězec odstraněn ze vstupního řetězce. V dalších krocích se tedy pracuje s novým zkráceným řetězcem. Ve všech regulárních výrazech druhého seznamu se

nevyskytují závorky, protože jejich správný počet a umístění se testují až v závěrečné fázi. Po odstranění řetězců pomocí regulárních výrazů druhého seznamu vznikne nový řetězec, který by měl obsahovat pouze závorky, jestliže obsahuje i nějaké znaky navíc, pak je uživatelský vstup nesprávný. V případě, že obsahuje pouze závorky, tak se musí ověřit jejich správnost ve smyslu počtu a umístění těchto závorek.

Při kontrole závorek se nabízely dvě možnosti. První z nich bylo ověření počtu levých a pravých závorek. Tato možnost ovšem neověřovala jejich správné umístění, a proto bylo nutné vymyslet jiné řešení. Toto řešení spočívá v použití cyklu, který končí pouze v případě, že vstupní řetězec je po úpravě stejný jako v minulém cyklu nebo že vstupní řetězec má nulovou délku. Jádrem tohoto cyklu je regulární výraz, který z daného vstupního řetězce vymaže levou a pravou závorku, které následují v řetězci za sebou, čímž se tyto závorky postupně zevnitř odstraňují. Jestliže byl uživatelský vstup v pořádku, zůstane na konci algoritmu řetězec nulové délky. Pokud je zadaný řetězec prohlášen parserem za chybný, je uživatel informován o tom, že musí vstupní data opravit.

Touto kontrolou předejdeme problémům, které by byly způsobeny vyšetřováním neplatného výrazu pomocí Reverzní polské notace, či jejího vyčíslení, kdy by došlo k selhání programu. V opačném případě by byl výraz vyhodnocen špatně. Použití regulárních výrazů je pro tento způsob ověřování vstupů výhodné, protože je jednoduše modifikovatelný a nevyžaduje zásah do logiky programu.

7.2.2 Reverzní polská notace

Reverzní polská notace je způsob zápisu aritmetických, algebraických, či analytických výrazů. Tato postfixová notace snižuje nároky na paměť počítače a zejména v minulosti byla často využívána v kalkulátorech. Dále je využívána například při tvorbě překladače nebo interpretů pro programovací jazyky. Její princip spočívá v umístění operandů před operátor. Mluvíme tedy o postfixové notaci, kdy pro jednoznačnost zápisu nepotřebujeme využívat závorky, jako v klasické notaci infixové [49].

Jedním ze způsobů, jak převést matematické zápisy z infixové do postfixové notace je *Shunting - Yard algoritmus* vyvinutý Edgarem Dijsktrou. Tento algoritmus je implementován v metodě `RPN`. V aplikaci je základní postfixová notace rozšířena o možnost zadávání goniometrických funkcí, dekadického a přirozeného logaritmu, Eulerova čísla, Ludolfova čísla, nebo o parametr funkcí. Před vlastním algoritmem jsou volány metody sloužící k úpravě vstupního výrazu. Nejdříve je volána metoda *EditSigned*, kde pomocí regulárních výrazů dochází k nahrazení vedle sebe stojících znamének plus nebo minus (a jejich kombinace) jedním znaménkem. V metodě *ConvertDecNumToFraction* dochází k převedení desetinného čísla na zlomek. Dále je zde zlomek blíže analyzován a případně zkrácen do základního tvaru. Touto metodu navíc odstraníme z výrazu desetinnou tečku, kterou pro převod do postfixové notace nepotřebujeme. Následně pomocí metody *AddMultiplyOperator* doplníme znaky násobení mezi dvě vedle sebe stojící závorky, číslo a závorku nebo číslo a proměnnou. Tím umožníme uživateli zadávat výrazy bez toho, aniž by musel explicitně zadávat znak násobení.

Pro zajištění větší stability převodu výrazu do postfixové notace bylo třeba

ošetřit případy, kdy vstupní výraz obsahuje binární operátor s pouze jedním operandem (například výraz -2). V tomto případě by převod selhal, protože by nebyl nalezen druhý operand, respektive vstupní výraz by obsahoval znaménko navíc. Z toho důvodu jsou v metodě *DetectUnaryOperator* ve vstupním výrazu vyhledány všechny tyto případy, přičemž po jejich nalezení dochází k následnému zabalení operátoru do výrazu $(0 + operátor + 1).operand$. Tím dojde ke změně unárního operátoru na binární.

Po vykonání metod, které slouží k úpravě výrazu, je aplikován vlastní RPN algoritmus. V rámci tohoto algoritmu nejdříve vytvoříme instanci třídy `Stack`. Poté pomocí cyklu procházíme vstupní výraz. V tomto cyklu nejdříve zjistíme, jestli je i-tý znak vstupního řetězce číslem, proměnnou, Eulerovým číslem nebo číslem π. Pokud ano, přidáme jej do proměnné *output*. Jestliže je i-tý znak písmenem, přidáme jej do proměnné *function*. Pokud tento znak nevyhovuje těmto dvěma podmínkám, jedná se s největší pravděpodobností o znak mezery, závorky nebo operátoru.

Detekujeme-li levou závorku, je vložena na vrchol zásobníku s nulovou prioritou. Je-li naopak detekována pravá závorka, přidáme do proměnné *output* znak z vrcholu zásobníku, který z něj následně odstraníme. Tento postup je třeba opakovat do té doby, než narazíme na levou závorku. V případě detekce operátoru dochází k rozpoznání konkrétního operátoru, načež je nastavena proměnná *priority*. Tato priorita je porovnávána s prioritou prvku na vrcholu zásobníku. V případě, že je priorita větší, je uzel na vrcholu předán do proměnné *output* a odstraněn ze zásobníku. Priorita operátoru je porovnávána s novým vrcholem. Toto je opakováno, dokud není priorita operátoru větší než priorita vrcholu zásobníku. Následně je operátor přidán na vrchol. Stejný postup je aplikován v případě detekce funkce. Na konci celého procesu je infixový zápis převeden na postfixový.

Operátor	*Priorita*
levá závorka	0
sčítání, odčítání	1
násobení, dělení	2
umocnění, funkce	3

Tabulka 7.1: Priorita operátorů v metodě `RPN`

Princip algoritmu osvětlíme na příkladu. Mějme dán výraz $sin(2 + pi) - 6/10$. Před vlastním vlastním zahájením *Shunting - Yard* algoritmu dojde k úpravě výrazu na výraz

$$sin(2 + pi) - 3/5. \tag{7.1}$$

Do proměnné *function* procházením vstupního řetězce přidáme výraz *sin*. Následně je uložena funkce sinus na vrchol prázdného zásobníku s prioritou tři. Během příštího průchodu vstupního řetězce detekujeme levou závorku, kterou přidáme na vrchol

zásobníku s nulovou prioritou. Do proměnné *output* přidáme číslo dvě a detekujeme operátor plus. Nastavíme prioritu jedna, kterou porovnáme s vrcholem zásobníku. Na vrcholu zásobníku je levá závorka s nulovou prioritou, a proto nic nebrání ve vložení operátoru plus na vrchol zásobníku. Do *output* uložíme číslo *pi* a detekujeme pravou závorku. Následně přeřadíme všechny prvky ze zásobníku do proměnné *output*, dokud nenarazíme na levou závorku v zásobníku. V našem případě odstraníme ze zásobníku pouze operátor plus (přidán do *output*) a poté levou závorku. V dalším kroku detekuje algoritmus operátor minus. Tento operátor přidáme na vrchol zásobníku, ovšem před tímto vlastním přidáním musíme z vrcholu zásobníku odstranit funkci sinus a předat ji do *output* (má prioritu tři). Do *output* dále přidáme číslo tři a opět detekujeme operátor dělení. Jelikož mu přiřadíme prioritu dvě, může být okamžitě vložen do zásobníku před operátor minus. Nyní přidáme do výstupu číslo pět. V zásobníku zbyly dva operátory. Jeden pro dělení a druhý pro odečítání. Tyto operátory tedy přidáme do *output* a vyprázdníme zásobník. Výstup `RPN` metody našeho příkladu vypadá tedy takto:

$$2\ pi\ +\ sin\ 3\ 5\ /\ -\ . \tag{7.2}$$

Posledním krokem je vyčíslení výrazu. Vyčíslení daného výrazu v aplikaci zajišťuje metoda *ComputeRPN*. V *ComputeRPN* nejdříve vytvoříme instanci třídy `Stack`. Poté rozdělíme podle mezer příchozí postfixový výraz do pole *expressions*. V cyklu testujeme nejprve s jakým prvkem pole *expressions* v dané iteraci pracujeme. Jestliže detekujeme číslo, přiřadíme tuto hodnotu do zásobníku. Dále můžeme detekovat textový řetězec. Pokud tento řetězec odpovídá některé z povolených funkcí, je z vrcholu zásobníku vyjmut operand, který se předá k vypočtení dané matematické funkce v tomto bodě. Pokud by nedošlo k zachycení chybného uživatelského vstupu ve třídě `Reg` a řetězec by neodpovídal žádné funkci z uvažovaného seznamu funkcí, dojde k vyslání řídícího signálu přes objekt delegátu, načež GUI vrstva v obslužné metodě delegáta kontaktuje uživatele o chybném vstupu.

V případě detekování operátoru je situace obdobná. Ze zásobníku ovšem vybíráme operandy dva, přičemž je na ně nutné aplikovat matematickou operaci s obráceným pořadím operandů, což je dáno LIFO charakteristikou zásobníku.

Po dokončení této operace testujeme počet zbylých položek v zásobníku. Jestliže je počet zbylých položek větší než jedna, můžeme předpokládat chybu v zadání a opět dochází k upozornění uživatele skrze GUI vrstvu. V opačném případě vracíme položku z vrcholu zásobníku, kde je uložen výsledek. Pokusíme se nyní vyčíslit výraz $2\ pi\ +\ sin\ 3\ 5\ /\ -$.

Nejprve dochází k detekci čísla dvě, které se uloží do prázdného zásobníku. V další iteraci je detekována konstanta *pi*. kterou vložíme na vrchol zásobníku. Nyní rozpoznáme operátor plus. Z vrcholu zásobníku vezmeme dva po sobě jdoucí operandy a provedeme jejich součet. Tento součet opět vložíme do zásobníku. V další iteraci budeme pracovat s řetězcem *sin*. Vyjmeme hodnotu z vrcholu zásobníku a zavoláme matematickou operaci pro výpočet hodnoty funkce sinus v bodě $2 + pi$. Výsledek opět vložíme na vrchol zásobníku. Číslo tři přidáme na vrchol zásobníku, stejnou operací provedeme i s číslem pět. V dalším kroku rozpoznáme operátor dělení. Nyní, stejně jako v případě operátoru plus, vyjmeme ze zásobníku dva nejvýše

umístěné operandy a provede jejich podíl, který vložíme zpět do zásobníku. V instanci zásobníku se nyní nachází dvě hodnoty, a to $3/5$ a $sin(2+pi)$, které v poslední iteraci odečteme a tento výsledek uložíme do zásobníku. V zásobníku zbyl pouze jeden prvek, přičemž tento prvek je výsledek výrazu $sin(2 + pi) - 3/5$.

7.3 Prezentační vrstva aplikace

V další části charakterizujeme nejvyšší vrstvu aplikace a popíšeme vlastnosti společné pro všechny její části. Prezentační vrstva aplikace je tvořena úvodním formulářem a dalšími formuláři, které slouží pro zadávání dat a zobrazování výsledků uživateli. Každý formulář je propojen jednak s kontrolní a výpočetní vrstvou, a jednak s vrstvou implementační.

7.3.1 Úvodní formulář

Implementaci úvodního formuláře nalezneme ve třídě `MainMenu`. Pomocí tohoto formuláře si uživatel z top-level menu vybere příslušnou numerickou metodu. Jakmile tuto metodu zvolí, dojde v této třídě k vytvoření příslušného objektu formuláře dané numerické kategorie a k jeho zobrazení. Pokud si uživatel vybere cvičnou úlohu, vytvoří se objekt třídy `SelectMethod` obsahující seznam implementovaných metod dané numerické kategorie. Po vybrání příslušné metody, dojde k vytvoření formuláře `SelectTask`, jež obsahuje seznam připravených cvičných úloh pro danou metodu. Každá cvičná úloha je uložena ve formě XML souboru. Při vytváření tohoto seznamu dojde k vytvoření instance třídy `XMLParser`, která zkontroluje pomocí metody *CheckByXSD* strukturu XML souboru pomocí předpřipraveného XSD souboru. Cvičné úlohy, které nejsou validní vzhledem k danému XSD, jsou v seznamu úloh podbarveny červeně a obsahují chybovou zprávu z `XMLParser`. Takto označené úlohy není možné zvolit, protože nejsou vzhledem k aplikaci v pořádku. Jakmile si uživatel vybere cvičnou úlohu, dojde k vyextrahování zadání z XML dokumentu cvičné úlohy a vytvoří se formulář příslušné numerické metody, kterému předáme všechna data nutná pro výpočet dané úlohy včetně případné poznámky k úloze. Tyto údaje jsou automaticky předvyplněny ve vstupní části formulářů numerických metod.

Sadu cvičných úloh lze editovat. Každý formulář pro výpočet matematické úlohy obsahuje tlačítko pro přidání úlohy a tlačítko pro napsání poznámky k dané úloze. Smazání úlohy je možné zvolením *Smazat úlohu* v kontextovém menu každé úlohy.

Programového návrhu úvodní obrazovky aplikace je využito i v tom případě, kdy uživatel chce stejnou úlohu vypočítat pomocí jiné metody bez toho, aniž by musel stejné zadání přepisovat do nového formuláře. V tomto případě si může vybrat zvolenou metodu ze seznamu umístěného ve formuláři dané numerické metody.

7.3.2 Formuláře numerických metod

Formuláře numerických metod jsou rozděleny do dvou částí. První část slouží pro zadávání vstupních dat uživatelem. Druhá část je určena hlavně pro zobrazení informací s řešením dané úlohy ve formě HTML stránky. V aplikaci je implementováno sedm těchto formulářů. Každý formulář se přizpůsobuje konkrétní numerické

metodě pomocí funkcí *DisableComponents* a *AlignComponents*. Jejich provedením dojde k vypnutí všech v danou chvíli nepotřebných prvků formuláře a zarovnání zbylých komponent. V každém formuláři je kromě povinných vstupních polí vytvořen *ListBox* obsahující seznam všech metod implementovaných v dané kategorii. Vybráním jedné z nabízených metod dojde interně k přenesení vstupních dat do třídy `MainMenu`, kde se vytvoří nový formulář úplně stejným způsobem jako v případě cvičných úloh, tj. do konstruktoru nového formuláře vložíme přenesená vstupní data původní metody a vyplníme jimi vstupní pole tohoto nového formuláře, čímž uživateli významně usnadníme a urychlíme práci s programem, jelikož je osvobozen od zadávání stejných vstupních dat v situacích, kdy potřebuje stejný příklad vyřešit pomocí více metod.

Dalším společným prvkem všech formulářů je *Trackbar* s jehož pomocí může uživatel měnit velikost písma bez toho, aniž by muselo dojít k novému překreslení výstupní části formuláře. Absence nutnosti překreslení je dána použitou knihovnou pro zobrazování matematických výrazů, která využívá technologii *AJAX*. Pomocí této technologie můžeme měnit obsah webové stránky právě bez nutnosti jejího plného znovunačtení.

V aplikaci je myšleno i na situaci, kdy by uživatel mohl mít zájem pouze o výsledek dané úlohy. K tomuto nastavení stačí odškrtnout *Checkbox* týkající se zobrazení mezivýsledku. Po této změně dojde automaticky k vygenerování a zobrazení nové HTML stránky, která již neobsahuje postup výpočtu, ale pouze řešení dané úlohy.

Každým formulář je dále vybaven *numericUpDown* komponentou, pomocí které lze nastavit maximální počet desetinných míst zobrazených u všech čísel obsažených ve výsledné webové stránce.

Pro cvičné úlohy jsou určeny dvě tlačítka *Přidat úlohu* a *Přidat poznámku*. Stisknutím tlačítka *Přidat úlohu* dochází v pozadí aplikace k vytvoření XML souboru dané úlohy, který obsahuje zadání úloh a případnou poznámku k tomuto příkladu. Tento soubor je dále porovnáván s již existujícími XML soubory dané metody. Jestliže jsou dané dokumenty shodné, je uživatel informován o tom, že daný příklad se mezi cvičnými úlohami vyskytuje a není přidán k ostatním cvičným úlohám. Jinak je vytvořený XML soubor uložen mezi ostatní. V případě, že uživatel načte do aplikace cvičnou úlohu s poznámkou, je její obsah automaticky vložen do formuláře *Note*, kde je připravena k případné editaci.

Posledním tlačítkem ve vstupní části formuláře je tlačítko *Vypočítej*, které je aktivní jen tehdy, když jsou vyplněna všechna povinná vstupní pole. Po stisknutí tohoto tlačítka dojde k zavolání metody *CheckInput*, pomocí které bude provedena kontrola validnosti uživatelem zadaných vstupních dat (dochází k interakci s kontrolní a výpočetní mezivrstvou). Pokud uživatelský vstup není validní, dojde k informování uživatele pomocí dialogového okna s příslušnou chybou. V opačné situaci dojde k předání požadavku výpočtu *BackGroundWorkeru*, který získá řešení dané úlohy na dalším vlákně aplikace. Po skončení výpočtu si v nejvyšší vrstvě převezmeme předávanou strukturu dat z implementační vrstvy a zavoláme obslužnou metodu, která zajistí vytvoření webové stránky, jež obsahuje kromě jiného i MathML kód.

MathML je matematický značkovací jazyk založený na bázi XML dokumentů

určený pro zápis matematických výrazů.

Pro usnadnění tvorby MathML kódu byla vytvořena statická třída `MathmlGen`, která obsahuje struktury pro tvorbu MathML tagů. Například výraz

$$cos(x^3)/(x-2) \tag{7.3}$$

zapsaný ve formě MathML kódu vypadá takto:

```
<math xmlns='http://www.w3.org/1998/Math/MathML'>
 <mfrac>
  <mrow>
   <mi>cos</mi>
   <mo>(</mo>
   <msup>
    <mi>x</mi>
    <mn>3</mn>
   </msup>
   <mo>)</mo>
  </mrow>
  <mrow>
   <mi>x</mi>
   <mo>-</mo>
   <mn>2</mn>
  </mrow>
 </mfrac>
</math>
```

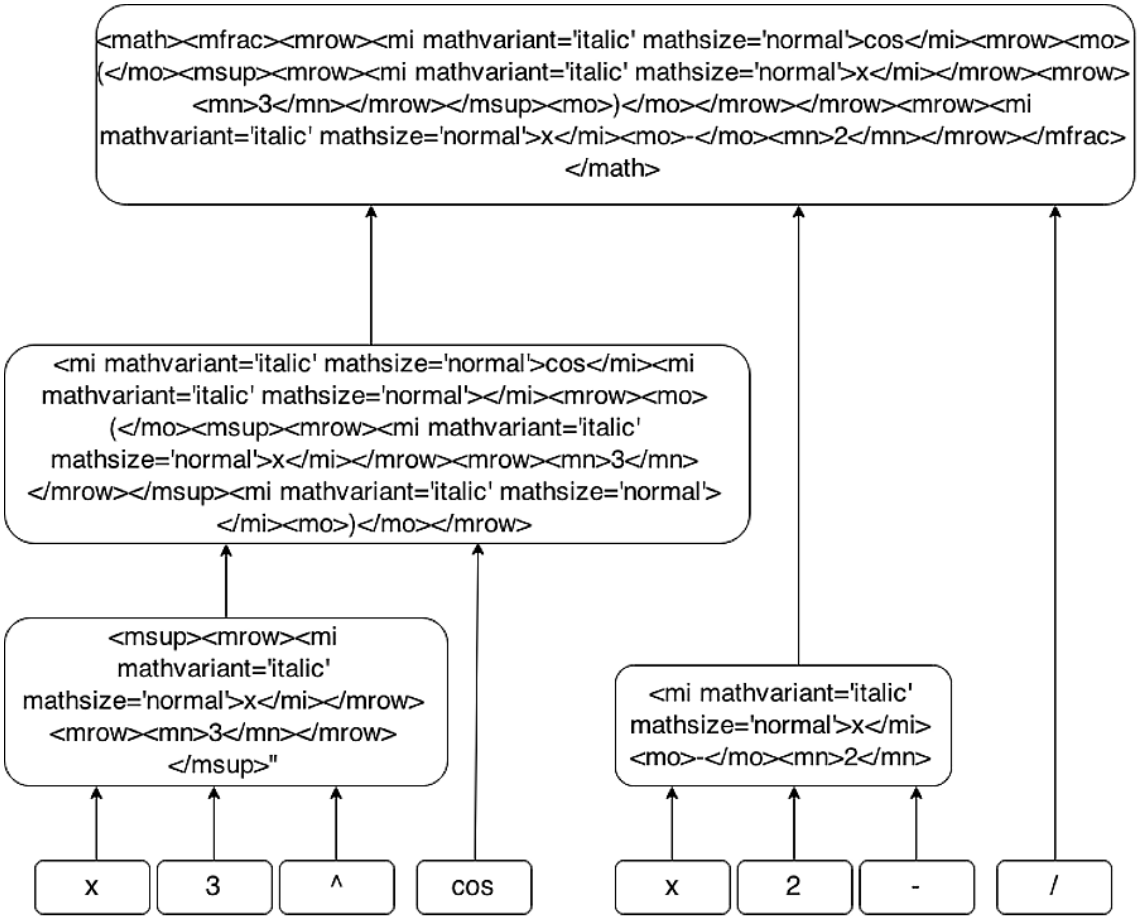
Pomocí aplikace vytvoříme ručně tento výraz tímto způsobem:

```
Mathml.HeaderMath(Mathml.Display.inline)
        + Mathml.Frac(Mathml.Literal("cos")
                + Mathml.LZ + Mathml.Sup(
                        Mathml.Literal("x"),
                        Mathml.Constant(3))
                + Mathml.PZ,
                Mathml.LZ
                        + Mathml.Literal("x") + Mathml.Minus
                        + Mathml.Constant(2)
                + Mathml.PZ)
+ Mathml.EndMath;
```

Naznačený způsob generování matematických výrazů je využít jednak pro všechny předpisy a vzorce numerických metod v aplikaci, a jednak při každém generování postupu výpočtů a výsledků spočtené matematické úlohy. Pro výstupy metod bylo třeba navíc vymyslet algoritmus, který by automaticky převedl v podstatě libovolná uživatelská a veškerá proměnná data ve výpisech do MathML podoby. Tento algoritmus nalezneme ve třídě `TextToMathml` v metodě *GenTextToMathml*.

Základní myšlenka algoritmu je založena na následující úvaze. Nejprve převedeme příchozí matematický výraz do postfixové notace pomocí metody `RPN` a rozdělíme

podle mezer do seznamu. Nyní budeme pomocí smyčky procházet tento seznam a vždy, když narazíme na binární operátor, převedeme dva předchozí prvky do kódu MathML (v případě, že narazíme na funkci, je situace obdobná pouze s tím rozdílem, že nebereme v úvahu dva předchozí prvky, ale pouze jeden). Nalezené položky spojíme v jeden výsledný prvek, přičemž mezi ně vložíme MathML kód příslušného operátoru (plus, minus, násobení nebo mocnění). V ostatních případech (funkce, odmocnina či dělení) jsou operandy umístěny do připravených MathML konstrukcí. Operandy včetně operátoru vymažeme ze seznamu a na jejich místo vložíme získanou položku v MathML podobě. Tento postup budeme opakovat do doby, než získáme jeden výsledný prvek. Obrázek 7.2 ilustruje postup převodu výrazu $cos(x^3)/(x-2)$ do MathML.



Obrázek 7.2: Ukázka převodu výrazu do MathML

Nejprve detekujeme operátor mocnění, který spojíme s operandy x a 3 v jednu MathML konstrukci. Operátor včetně operandů vymažeme ze seznamu a na jejich místo vložíme získanou položku. V dalším průchodu nalezneme goniometrickou funkci *cosinus*, jejíž konstrukci předáme jako parametr předchozí prvek. Opět zmenšíme seznam. Dalším detekovaným operátorem je operátor minus a vytvoříme MathML prvek spojením se dvěma předchozími operandy. V tuto chvíli se v seznamu nachází dvě dílčí MathML kódy a operátor dělení. Pro tento operátor je

připravena příslušná struktura, které jako parametr předáme poslední dva prvky. Touto stromovou konstrukcí získáme výsledný MathML kód uvažovaného výrazu.

Vzhledem k tomu, že výrazy zapsané pomocí RPN notace nevyžadují závorky, bylo nutné implementovat metodu, pomocí které rozpoznáme jejich původní umístění, abychom je byly schopni zpětně vykreslit ve výstupní části formuláře.

Řešením bylo porovnání počtu vnořených výrazů (které jsou obaleny závorkami) a operátorů v daném výrazu. Pokud je operátorů více než vnořených výrazů, bude výraz umístěn mezi závorky.

Pro vyrenderování MathML je v aplikaci využita javascriptová knihovna MathJax, která je určena pro zobrazování matematických výrazů ve webových prohlížečích sázených za pomoci LaTeX nebo MathML. Pro snazší vygenerování webové stránky byla vytvořena statická třída `HTMLGen`. Tato třída je svým návrhem velice podobná třídě `MathMLGen`, protože obsahuje metody, které usnadňují zápis HTML tagů. Vygenerovaná HTML stránka bude následně zobrazena uživateli ve výstupní části formuláře.

Výstupní část každého formuláře obsahuje *TabControl* s dvěma stránkami. První stránka obsahuje komponentu *Awesomium*, což je webový prohlížeč určený pro *.NET* aplikace nebo programy psané v jazyce *C++*. Tato komponenta byla upřednostněna před ostatními možnostmi z několika důvodů. Nejdůležitějším aspektem byla úroveň zpracování MathML jazyka. *Awesomium* je například schopno si poradit mnohem lépe s odřádkováním použitým v Mathml než další komponenta podobného zaměření, kterou je *Gecko*. Tato komponenta nepodporuje odřádkování vůbec. Další výraznou výhodou této komponenty v porovnání s MSIE prohlížečem či již zmíněným *Geckem* je možnost využití metod pro export do obrázkového formátu PNG či PDF. Samotní vývojáři *Awesomia* si navíc položili za jako jeden z hlavních cílů přidání podpory tzv. *thread-safe*, která by přinesla možnost *multithreadingu* při manipulaci s touto komponentou.

První stránka *TabControl* obsahuje také nabídku, jejíž první položka je určena pro konverzi HTML výstupů metod do dalších formátů. V aplikaci jsou podporovány převody do XML, HTML, PDF, JPG, GIF, PNG, BMP a TIFF. Druhá položka nabídky slouží k výběru cvičné úlohy pro danou numerickou metodu.

Druhá stránka *TabControl* obsahuje komponentu *AcroPDF*, ve které jsou zobrazovány PDF soubory obsahující charakteristiku implementovaných numerických metod.

Kromě samotného výpočtu jsou HTML stránky generované metodami z oblasti aproximace funkce, Newton - Cotesovy vzorce a řešení nelineárních rovnic doplněny o obrázky grafů. Jejich implementaci nalezneme ve třídě `Graph`. Příslušné grafy vykreslíme v prezentační vrstvě tak, že využijeme získané struktury přijaté z nejnižší vrstvy. Z těchto struktur zjistíme předpisy funkcí, které budeme v soustavě souřadnic zobrazovat. Zjištěné předpisy odešleme do kontrolní a výpočetní vrstvy aplikace, kde vypočteme funkční hodnotu na požadovaném intervalu v celkem čtyř stech bodech. Body následně proložíme Beziérovou křivkou. Získaný graf je vložen do bitmapy a uložen jako obrázek ve formátu PNG.

V případě metod pro řešení nelineárních rovnic a metod pro aproximaci a interpolaci funkce (vyjma interpolačních splinů) je v soustavě souřadnic zobrazen

graf funkce. Metody interpolačních spline funkcí zobrazují v soustavě souřadnic na každém podintervalu graf interpolačního polynomu. U Newton - Cotesových vzorců je kromě grafu funkce barevně ilustrována spočtená plocha integrálu. Jestliže uživatel zvolí složenou variantu Newton - Cotesových vzorců je zobrazen graf funkce s barevně vyznačenou plochou, jejíž obsah vyhovuje požadavku zadané přesnosti. Vybarvená plocha se skládá z dílčích částí odpovídajících spočtenému integrálu funkce na každém z uvažovaných podintervalů.

V další části popíšeme postup převodu HTML stránky do dalších formátů. Tyto algoritmy jsou k nalezení ve třídě `Exports`.

Princip exportu do XML spočívá v odstranění všech HTML tagů a atributů z vygenerované webové stránky pomocí připraveného regulárního výrazu. Výjímku tvoří HTML tag *span* obsahující text, který je nahrazen MathML tagem *mtext*, čímž neztratíme při této konverzi žádný text z dané webové stránky. Po tomto převodu je na začátek vzniklého souboru umístěna XML hlavička a soubor je uložen.

Uživatel může z aplikace extrahovat přímo vygenerovanou webovou stránku. Při tomto exportu dochází k vytvoření nového HTML souboru, který obsahuje zdrojový kód vygenerované webové stránky. Je-li součástí webové stránky i graf, pak je obrázek grafu zkopírován do stejné složky jako příslušný HTML dokument.

Při tvorbě exportu do některého grafického formátu nebo do formátu PDF byla situace složitější než v předchozích případech. Největším problémem bylo získat celý obrázek v případě výpočtu úloh, pro které byla vygenerována HTML stránka, k jejímuž plnému zobrazení v aplikaci bylo nutné využít posuvníku. Pokud bychom v tomto okamžiku chtěli vytvořit příslušný export, tak by obrázek obsahoval pouze viditelnou část výpočtu. Z tohoto důvodu bylo využito potenciálu *Awesomia*, kdy bylo nutné nejprve zjistit šířku a výšku vygenerované HTML stránky. Toho bylo dosaženo pomocí Javascriptu, kdy každá vygenerovaná HTML stránka v aplikaci obsahuje javascriptové funkce pro získání těchto informací. Dále jsme vytvořili instanci třídy *WebView*, které bylo zapotřebí předat zjištěné rozměry a cestu k HTML dokumentu. Výhodou této třídy je schopnost načtení a vykreslení HTML stránky bez toho, aniž by daná plocha musela být zobrazena. Po vykreslení celé stránky dojde ke zjištění rozměrů webové stránky a podle nich k úpravě velikosti *WebView*. Tímto postupem dosáhneme stavu, kdy *WebView* pojme celou HTML stránku bez nutnosti použití posuvníku, a tudíž při uložení plochy *WebView* jako obrázku bude viditelná celá úloha. Během exportu se zároveň snažíme o to, aby ani jeden z rozměrů nebyl o mnoho větší než druhý. Pokud se tak stane, zmenšíme velikost písma v HTML stránce (včetně velikosti písma použitého v rámci MathML kódu) a opětovně načteme stránku, čímž dojde ke zmenšení rozdílu mezi šířkou a výškou stránky. Toto opakujeme však nejvýše třikrát, přičemž minimální velikost písma je šestnáct. Je-li v případě konverze do formátu PDF šířka větší než maximální povolená šířka, dochází k vytvoření PDF souboru orientovaného na šířku.

Během exportu je uživateli zobrazeno dialogové okno nesoucí informaci o právě probíhající konverzi. Jakmile je export dokončen, dialogové okno zmizí a ve stavové liště v levém dolním rohu formuláře se objeví zpráva s informací o dokončeném exportu. Uživatel může vytvořený soubor otevřít z aplikace kliknutím na ikonu složky vedle této stavové informace. Soubor je otevřen ve stejném programu, s jakým má

uživatel nastavenou asociaci v rámci jeho operačního systému.

Slabinou tohoto řešení je, že *Awesomium* v současné době nepodporuje možnost provedení vykreslení na separovaném vlákně, což u úloh s rozsáhlým postupem přináší nemožnost práce s formulářem po dobu provádění exportu.

Další nevýhodou zejména v případě exportu do formátu PDF je doba konverze, která je dána především rychlostí metody *PrintToFile*. Tato metoda je součástí *Awesomia* a slouží pro samotné vytvoření a následné uložení PDF souboru z přiložené HTML stránky.

7.4 Ilustrace interakce vrstev aplikace

V poslední části diplomové práce nastíníme interakci vrstev při výpočtech matematických úloh pomocí vybraných numerických metod a z různých pohledů přiblížíme dění v pozadí aplikace.

7.4.1 Numerická integrace

První kategorií, kterou budeme blíže charakterizovat, je numerická integrace. V okamžiku kdy uživatel zvolí v hlavním menu programu metodu numerické integrace, je ve třídě `MainMenu` vyvolána událost *SelectIntegrationMethod_Click*, která získá celočíselný identifikátor (ID) metody a zavolá metodu *SelectIntegrationMethod*. Tato metoda na základě identifikátoru vytvoří instanci třídy `Integration_win`, které předá získané ID metody. V prezentační vrstvě dochází k lokalizaci a vypnutí aktuálně nepotřebných komponent formuláře a připojení obslužných metod pro delegáty a inicializaci třídy *BackGroundWorker*, která slouží pro obsluhu aplikace na úrovni vláken. Uživatel nyní vyplní všechna vstupní pole v horní části zobrazeného formuláře, provede nastavení týkající se počtu požadovaných desetinných míst či požadavku o zobrazení mezivýsledku a stiskne tlačítko *Vypočítej*.

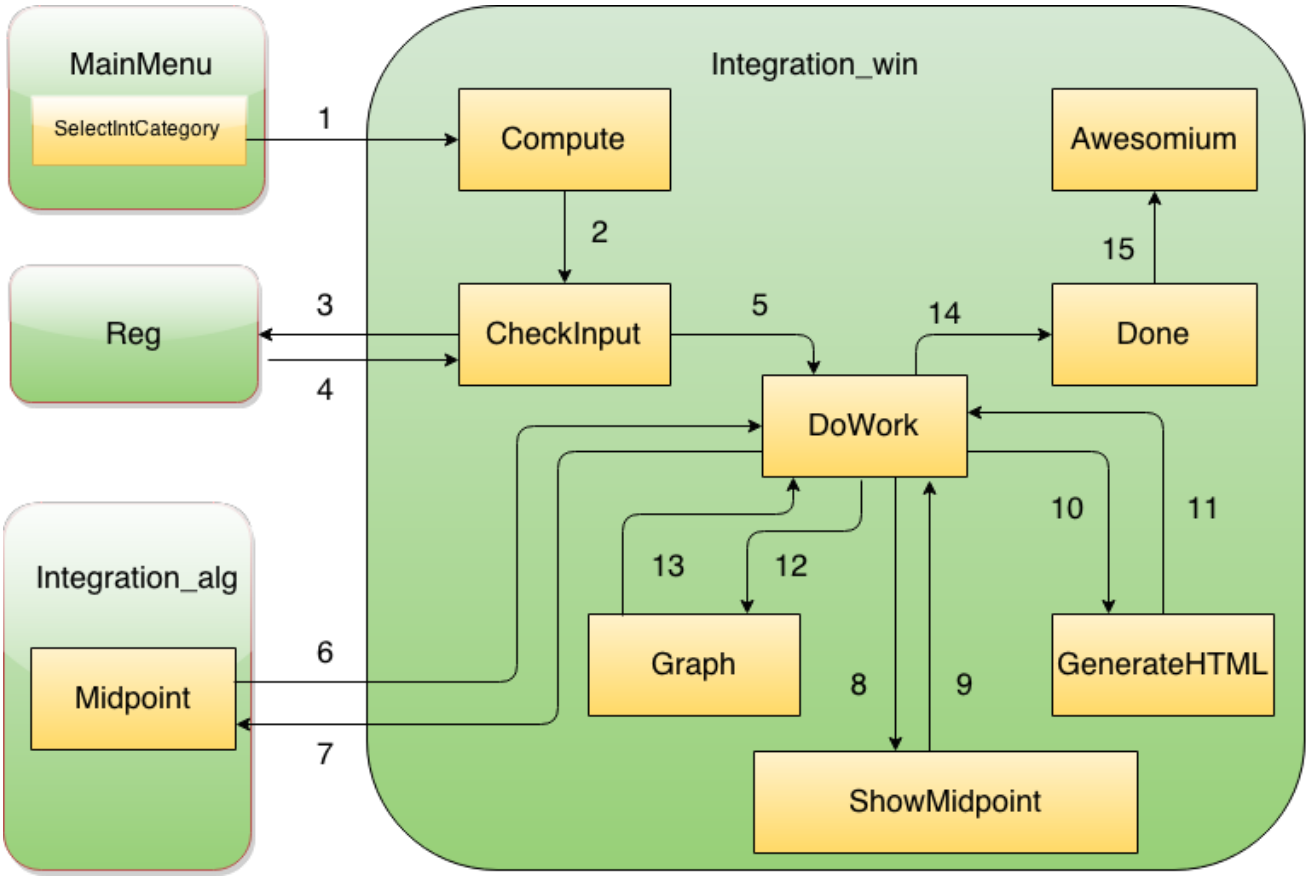
V tuto chvíli dochází v prezentační a kontrolní vrstvě k validaci vstupních údajů a testu jejich smysluplnosti. Díky provázanosti vrstev mohou všechny vstupní položky v celé aplikaci obsahovat funkce a algebraické výrazy. V případě, že vstupní data nejsou validní, je uživateli zobrazeno dialogové okno sdělující konkrétní příčinu. Pokud žádný z validátorů vstupních dat nezastaví proces vyvoláním chyby, je nad daty zavolána anonymní metoda fungující na separovaném vlákně, která na základě číselného identifikátoru spustí konkrétní metodu implementační vrstvy, jež předá požadované parametry v souladu s kapitolou 7.1. V implementační vrstvě nyní dochází k výpočtu zadané úlohy, k čemuž využívá kontrolní a výpočetní vrstvu (konkrétně metodu *RPN* a *ComputeRPN* třídy `Expression`) pro vyčíslení funkční hodnoty funkce v daném bodě.

Po ukončení výpočtu je zpět nejvyšší vrstvě odeslána příslušná datová struktura nesoucí informace o průběhu výpočtu společně s výsledkem zadané úlohy. V rámci metody *DoWork* dojde k předání získané datové množiny obslužné rutině, jejímž úkolem je na základě přijatých dat vygenerovat webovou stránku se zadáním a předpisem metody (tyto předpisy nalezneme ve třídě `Formulas`). Pokud uživatel vyžaduje postup výpočtu, dojde nejprve k dosazení do obecných vztahů metod.

Až poté je zobrazen konkrétní postup při řešení dané úlohy. Na konec webové stránky umístíme do rámečku barevně odlišený výsledek zadané úlohy.

Patří-li zvolená metoda mezi Newton - Cotesovy vzorce, je součástí HTML stránky i obrázek s grafem funkce s barevně vyznačenou plochou vypočteného integrálu. Výsledná HTML stránka je nakonec uživateli zobrazena v komponentě *Awesomium*.

V případě, že během výpočtu dojde k chybě, výpočet se přeruší a uživatel je o tomto stavu a příčině informován pomocí obslužných metod delegátů.



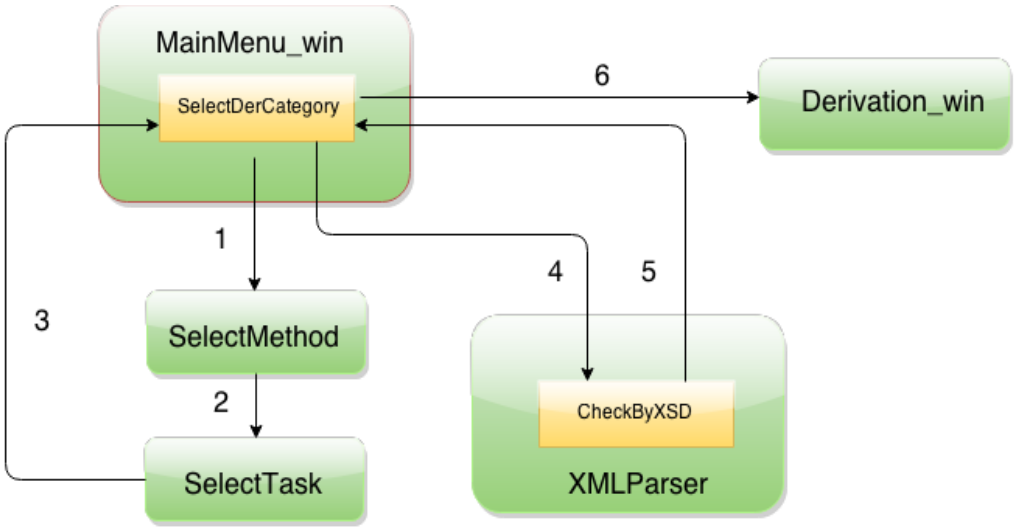
Obrázek 7.3: Zjednodušený postup výpočtu úlohy obdélníkovým pravidlem s následným zobrazení pomocí Awesomia

7.4.2 Numerická derivace

Postup výpočtu úloh týkající se určení derivace funkce v bodě je velice podobný jako v případě numerické integrace. Proto se nyní podíváme na interakci vrstev a dění v pozadí aplikace z pohledu cvičných úloh.

V hlavním okně programu vyberme cvičné úlohy. V rámci třídy `MainMenu` se skrze metodu *numerickáDerivaceCvicneUlohy_Click* vytvoří formulář `SelectMethod` obsahující příslušné numerické metody z vybrané numerické oblasti. Po výběru některé z metod se vytvoří nový formulář `SelectTask`, který obsahuje seznam cvičných úloh. Při plnění tohoto seznamu je v rámci metody *SelectedDerCategory* provedena

kontrola XML souboru s cvičnou úlohou vůči XSD dokumentu (pomocí metody *CheckByXSD* třídy `XMLParser`). Vybereme-li konkrétní úlohu dojde dále v metodě *SelectedDerCategory* třídy `MainMenu` nejprve k extrahování obsahu elementů XML dokumentu a poté k vytvoření příslušného formuláře, kterému ovšem nepředáme pouze číselný identifikátor metody, ale i všechna extrahovaná data. Těmito daty jsou ve formuláři `Derivation_win` předvyplněna všechna vstupní pole. Další postup je již analogický numerické integraci. Po nastavení požadovaného počtu desetinných míst a zobrazení mezivýsledků stiskneme tlačítko *Vypočítej*. Ve spolupráci s kontrolní a výpočetní vrstvou proběhne kontrola vstupních dat a v případě, že není zjištěn žádný problém, je spuštěn *BackGroundWorker*, v jehož režii proběhne kooperace s implementační vrstvou. V rámci implementační vrstvy je získáno řešení, které je odesláno GUI vrstvě. Zde dojde k vygenerování webové stránky a jejímu zobrazení pomocí *Awesomia*.



Obrázek 7.4: Diagram znázorňující načtení cvičné úlohy

7.4.3 Řešení nelineárních rovnic

Jakmile vybereme konkrétní metodu z této numerické kategorie je vytvořena instance třídy `Nelinear_solutions_win` (respektive `SeparationOfRoots_win` v případě Sturmovy posloupnosti). Po inicializaci podpůrných metod a nastavení nezbytných proměných je uživateli zobrazen formulář, ve kterém nejprve vyplní požadované údaje. V případě Newtonovy či Halleyovy metody může uživatel vyplnit navíc vstupní pole týkající se analytického tvaru první a druhé derivace. Pokud jsou tato pole vyplněna, jsou použita přednostně. V opačném případě je první i druhá derivace počítána numericky (první derivace funkce v bodě je určena pomocí centrální diference). Na úrovni implementační vrstvy tedy dochází ke spolupráci mezi třídou `Nelinear_Solutions_alg` a `Derivation_alg` . V okamžiku vyplnění požadovaných informací, dochází k jejich ověření kontrolní a prezentační vrstvou. V případě validního vstupu spustíme *BackGroundWorker*, který předá řízení příslušné metodě

implementační vrstvy. Tím vyhradíme samotnému výpočtu vlastní vlákno. Výstupem těchto metod je objekt `Nelin_ret`, který je předán GUI vrstvě. Na základě informací, které nese, je vytvořena webová stránka, kde je postup výpočtu uspořádán ve formě tabulky. Na konec tohoto HTML výpisu je připojen graf zadané funkce získaný jejím vyhodnocením ve čtyř stech bodech.

7.4.4 Aproximace funkce

Čtvrtou oblastí, kterou budeme blíže popisovat je aproximace a interpolace funkce. Tyto metody blíže popíšeme z hlediska jejich programové implementace. Metody této skupiny lze rozdělit do tří podmnožin.

První podskupina je tvořena interpolačními polynomy, ve které jsou implementovány dvě metody realizující Newtonův respektive Lagrangeův interpolační polynom. Po vyplnění uzlových bodů a jejich funkční hodnot proběhne kontrola validity vstupních dat. Kromě správnosti zadání z pohledu správného zápisu bodů a funkčních hodnot je například kontrolováno, zda nemá určitý uzlový bod přiřazeny dvě funkční hodnoty (a tudíž by se nejednalo o funkci). *BackGroundWorker* po kontrole předá řízení požadované metodě implementační vrstvy.

Metoda *LagrangePolynomial* nejprve vytvoří jmenovatele a čitatele všech fundamentálních polynomů. Tito čitatelé jsou v rámci konkrétního polynomu roznásobeni. Současně je vypočtena hodnota jeho jmenovatele. Poté dochází k vytvoření polynomu vynásobením čitatele a převrácené hodnoty jmenovatele. Získaný polynom je vynásoben funkční hodnotou uzlu x_i. Naznačeným způsobem vypočteme všechny polynomy, které mezi sebou sečteme, čímž získáme výsledný Lagrangeův interpolační polynom.

V případě výpočtu Newtonova interpolačního polynomu dochází nejdříve k určení poměrných diferencí pomocí dvou vnořených cyklů, které jsou následně vynásobeny polynomy ve tvaru $x-x_i$. Zde využíváme myšlenky, kterou můžeme vyčíst z předpisu Newtonova interpolačního polynomu (1.12), kdy v i-té iteraci cyklu stačí aktuální součin polynomů vynásobit polynomem $x-x_i$, čímž ušetříme mnoho početních operací. Tímto způsobem získáme soubor polynomů, ze kterých jejich sečtením získáme výsledný Newtonův interpolační polynom.

Druhá skupina je tvořena metodami, které jsou založeny na interpolaci funkce pomocí interpolačních splinů. Princip interpolačních spline funkcí spočívá ve vytvoření polynomů na dílčích intervalech, které jsou vytvořeny mezi všemi uzlovými body. V případě interpolace přirozeným kubickým splinem sestavíme nejprve v souladu se vztahem (1.44) soustavu rovnic. Na úrovni implementační vrstvy následně dojde ke spolupráci se třídou `Linear_solutions_alg`, kdy na vzniklou soustavu s třídiagonální maticí aplikujeme přímý a zpětný chod Gaussovy eliminace, čímž vypočteme tzv. momenty spline (pro zachování univerzálnosti předpokládáme neekvidistantní dělení intervalu). V další fázi výpočtu zjistíme podle vztahů (1.26) a (1.30) hodnoty integračních konstant A_i a B_i. Poté využijeme operace s polynomy a vypočteme kubické polynomy na intervalech podle vztahu (1.23).

K vypočtení interpolačních polynomů bylo nutné v aplikaci implementovat metody umožňující například sčítat, odečítat nebo násobit polynomy. Tyto algoritmy

nalezneme ve třídě `Polynoms`, jejíž metody přijímají seznamy číselných hodnot, obsahující hodnoty koeficientů daného polynomu.

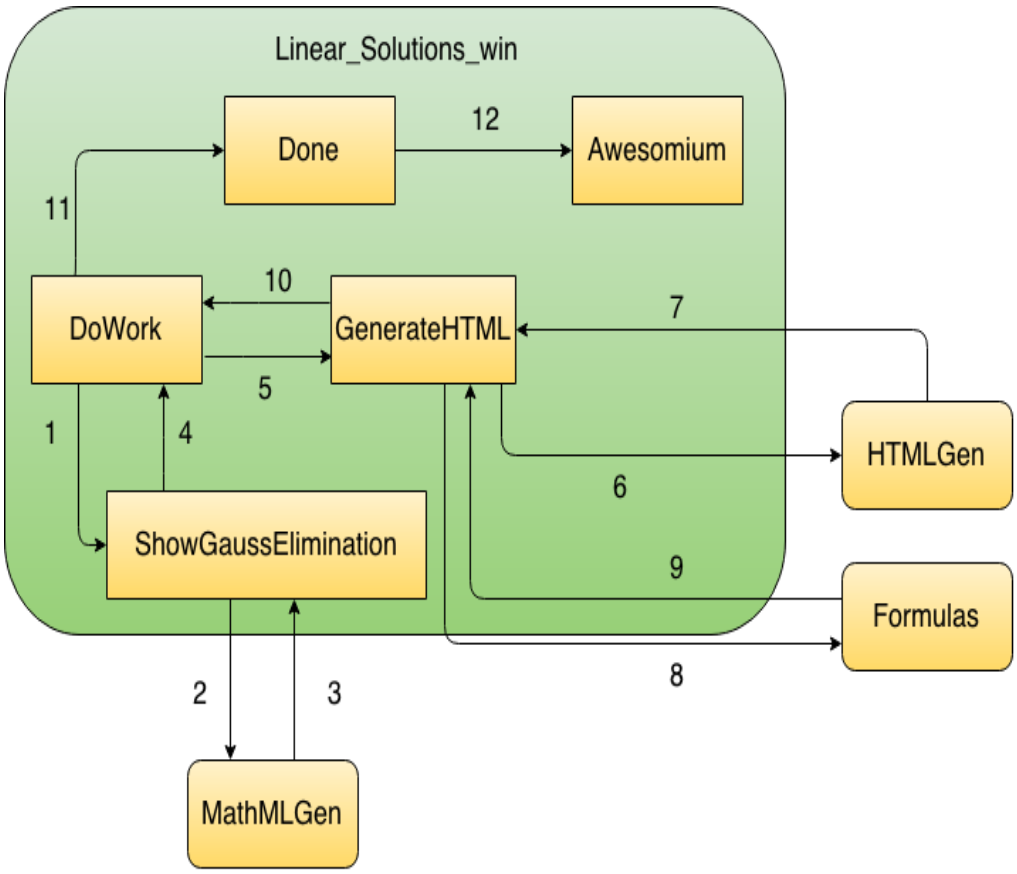
Do poslední skupiny byl zařazen algoritmus metody nejmenších čtverců, který realizuje aproximaci funkce pro obecný stupeň polynomu. Tato metoda skládající se z několika cyklů je implementována pomocí funkce *LeastSquarePolynom*. Využitím smyček jsou vypočteny součty uvedené ve vztahu (1.49) a které využijeme pro získání řešení výsledné soustavy lineárních rovnic. Pro získání koeficientů hledaného polynomu aplikujeme na vzniklou soustavu lineárních rovnic přímý a zpětný chod Gaussovy eliminace.

Po skončení výpočtu úlohy jakoukoli z uvedených metod jsou veškeré mezivýpočty společně s výsledkem odeslány prezentační vrstvě, kde její metody tuto datovou strukturu použijí pro vygenerování a interpretaci výsledné webové stránky uživateli. Ve výpisu je kladen důraz na didaktičnost, a proto u každé z metod nalezneme dosazení do obecného vztahu a až poté postup výpočtu konkrétní úlohy s výsledkem. Všechny vygenerované webové stránky obsahují na svém konci graf vypočteného polynomu se znázorněnými uzlovými body.

7.4.5 Řešení soustav lineárních rovnic

Předposlední skupinu týkající se řešení soustav lineárních rovnic metod budeme blíže charakterizovat zejména z pohledu interpretace výsledků směrem k uživateli.

Po vytvoření formuláře, jeho následné přípravě konkrétní numerické metodě, kontrole korektnosti vstupních dat a vypočtení příslušné úlohy metodou, kterou nalezneme ve třídě `Linear_solutions_alg`, nastává fáze vygenerování výsledného HTML dokumentu. Ke každé numerické metodě určené pro řešení soustav lineárních rovnic byla naprogramována obslužná metoda, pomocí níž vytvoříme MathML kód obsahující obecný postup výpočtu, za nímž následuje konkrétní výpočet a barevně odlišený výsledek úlohy. K tomu využíváme připravených struktur statické třídy `MathmlGen`. Získaný MathML kód je opět odeslán *BackGroundWorkeru* (konkrétně metodě *DoWork*), který poté zavolá metodu *GenerateHtml*. Tato metoda je již společná pro všechny metody z dané numerické kategorie. Pro snazší zápis HTML tagů dokumentu využíváme připravené konstrukce statické třídy `HTMLGen`. Každý vygenerovaný soubor začíná HTML hlavičkou. V této hlavičce je uvedena cesta ke knihovně *MathJax*. Za touto cestou nalezneme konfiguraci této knihovny a javascriptové funkce využívané při provádění exportu do některého z obrázkových formátů či PDF. Po hlavičce následuje shrnutí zadání úlohy a formule zvolené metody. Pro možnost vykreslení předpisů numerických algoritmů byla vytvořena statická třída `Formulas`, která obsahuje MathML kódy těchto předpisů. Za předpisem metody následuje HTML tag *div*, do něhož umístíme vygenerovaný MathML kód úlohy. Jakmile tento dočasný dokument vytvoříme pošleme cestu k němu komponentě *Awesomium*, která ve spolupráci s knihovnou *MathJax* zajistí interpretaci výsledků uživateli včetně vykreslení matematických výrazů.



Obrázek 7.5: Ukázka postupu generování HTML pro Gaussovu eliminaci a jeho zobrazení pomocí Awesomia

7.4.6 Vlastní čísla a vlastní vektory

V této podkapitole se budeme soustředit nikoli na interakci mezi vrstvami modelu aplikace, ale na spolupráci mezi třídami nejvyšší vrstvy při požadavku exportu úlohy. Jako ilustrační metodu zvolme QR algoritmus s QR rozkladem získaným pomocí Householderovy transformace a demonstrujme situaci, kdy uživatel chce provést export postupu a výsledku konkrétní cvičné úlohy zvolené metody.

Z hlavního okna aplikace zvolíme *Cvičné úlohy* a pomocí formulářů `SelectMethod` a `SelectTask` si vybereme jednu úlohu ze seznamu připravených zadání. Po extrahování XML dokumentu a kontrole vůči XSD dokumentu, který v tomto případě vyžaduje, aby XML soubor povinně obsahoval vstupní vstupní matici, vstupní vektor a počet iterací a nepovinně poznámku k úloze, je zadání příkladu přeneseno do formuláře `EigenValues_EigenVectors_win` připraveného pro metodu QR algoritmu. Uživatel nyní může provést nastavení týkající se zobrazení postupu výpočtu či počtu zobrazovaných desetinných míst a stiskne tlačítko *Vypočítej*.

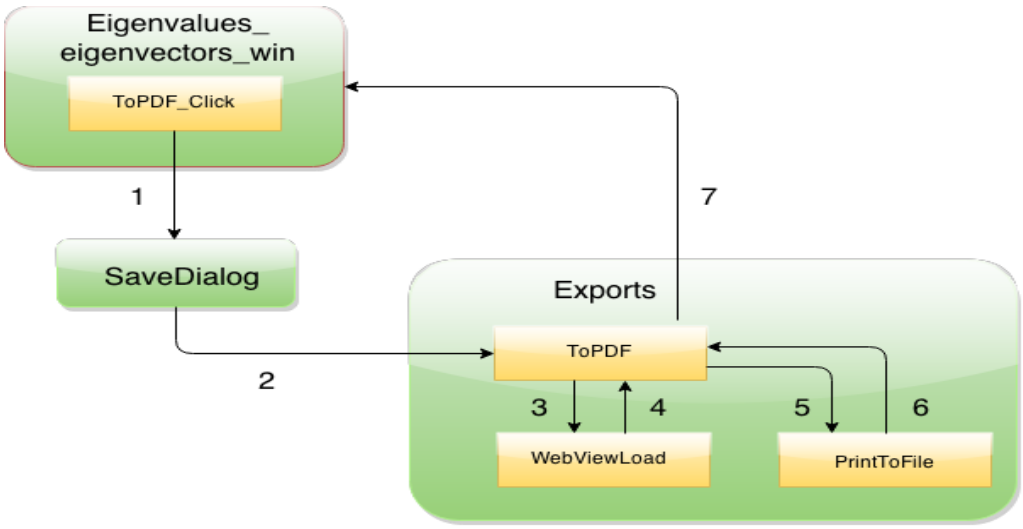
Jakmile dojde k ověření vstupních dat, je zahájen proces výpočtu úlohy (kooperace *BackGroundWorkeru* s konkrétní metodou implementační vrstvy). Po skončení výpočtu dojde k přenosu získané množiny dat do funkce *ShowHouseHolder* určené

k vygenerování MathML kódu pro QR algoritmus. Vykonáním metody *GenerateHTML* vytvoříme soubor, který je zobrazen uživateli.

Nyní si uživatel ve výstupní části formuláře zvolí z hlavní nabídky záložku *Exporty* a vybere formát souboru, do kterého chce provést konverzi.

V ilustračním příkladě zvolíme konverzi do formátu PDF. Uživateli je zobrazen *SaveDialog*, ve kterém zvolí požadované umístění PDF dokumentu. Metodou *ExecuteJavasctiptWithResult* zavoláme funkce umístěné v HTML dokumentu vypočtené úlohy a zjistíme rozměry HTML stránky. Tyto údaje společně s velikostí písma a cestou k dokumentu přepošleme metodě *ToPDF* třídy `Export`. V rámci této metody zavoláme metodu *WebViewLoad* a vytvoříme instanci třídy *WebView*, které předáme rozměry stránky s vypočtenou úlohou QR algoritmu. Následně načteme webovou stránku a zkontrolujeme jestli se výstup vejde na šířku stránky o rozměrech A4. Jestliže nikoli, zmenšíme písmo a stránku opětovně načteme. Toto opakujeme do doby, dokud je písmo větší než šestnáct. Avšak nejvýše třikrát. Pokud je i poté šířka HTML dokumentu větší, než šířka strany A4, je uložen daný PDF soubor na šířku pomocí dodatečného nastavení v metodě *PrintToFile*.

Během těchto operací je uživateli zobrazeno dialogové okno informující o právě probíhajícím exportu. Po dokončení exportu je dialogové okno uzavřeno a pomocí stavové lišty umístěné v levé dolní části formuláře je uživatel informován o dokončeném exportu, přičemž kliknutím na ikonu složky může získaný PDF soubor otevřít k nahlédnutí.



Obrázek 7.6: Ukázka postupu při exportu do PDF

7.5 Distribuce a instalace aplikace

Pro distribuci aplikace byly vytvořeny webové stránky a zakoupena doména `www.numerickemetody.cz`. Na těchto stránkách uživatel nalezne pokyny k instalaci aplikace, a zároveň si zde může stáhnout do svého počítače text diplomové práce ve formátu *PDF* a archiv obsahující vytvořenou aplikaci. Archiv obsahuje složku soubory, v níž jsou umístěny cvičné úlohy, *PDF* soubory popisující každou

numerickou kategorii, složku *mathjax2.3*, která obsahuje potřebné skripty pro vykreslování matematických výrazů, adresář font z něhož jsou instalovány potřebná matematická písma, cvičné úlohy a instalační soubory *Adobe Readeru* a *Awesomia*. Samotnou aplikaci nainstalujeme poměrně jednoduchým způsobem:

1. Stáhneme aplikaci ve formátu *.zip*.
2. Archiv extrahujeme do libovolné složky počítače.
3. Otevřeme složku s aplikací.
4. Spustíme soubor *Diplomova_prace.exe* s oprávněním správce.

Při prvním spuštění, respektive vždy, když není detekována přítomnost potřebných součástí v počítači uživatele, aplikace nainstaluje do počítače (z důvodu instalace potřebných součástí je vyžadováno oprávnění správce) *Awesomium SDK*, *Adobe Reader* a matematická písma. Po této instalaci je aplikace připravena k užívání.

Závěr

Cílem této diplomové práce bylo vytvoření rešerše věnující se vybraným metodám z šesti odvětví numerické matematiky a na jejím základě vytvořit aplikaci, jež kromě výsledku řešené úlohy prezentuje uživateli podrobný postup výpočtu úlohy.

V první části této práce byla pozornost věnována vytvoření teoretické stati, ve které byly charakterizovány principy a shrnuty vlastnosti metod z oblasti aproximace funkce, numerické derivace a integrace, řešení nelineárních rovnic, řešení soustav lineárních rovnic a oblasti pro numerický výpočet vlastních čísel a vlastních vektorů reálných symetrických matic.

Praktická část se zabývá programovou implementací aplikace v jazyce *C#*. Model aplikace je rozdělen do tří vrstev. Nejnižší vrstva obsahuje programovou realizaci všech numerických metod charakterizovaných v první části diplomové práce. V současné době aplikace obsahuje čtyřicet šest numerických metod. Tyto metody jsou vykonávány vždy pomocí vyhrazeného vlákna. Dále tato vrstva obsahuje pomocné třídy, jež jsou využity například při počítání s polynomy či maticemi. Další vrstvou je kontrolní a výpočetní vrstva, která implementuje *Shunting-Yard* algoritmus pro převod matematických výrazů do postfixové notace a obsahuje metodu pro výpočet hodnoty výrazů v této notaci. Dalším důležitým úkolem této vrstvy je provedení kontroly uživatelem zadaných vstupních dat. Díky tomu mohou všechny vstupní položky v aplikaci obsahovat matematické výrazy.

V rámci celé aplikace byl kladen důraz na didaktickou stránku aplikace, kdy je každá řešená úloha doplněna o výstup obsahující informace o postupu výpočtu. Prezentační vrstva proto vytváří na základě přijatých dat z nejnižší vrstvy HTML soubor s automaticky generovaným MathML kódem, který kromě samotného výsledku obsahuje i postup výpočtu. V případě aproximace funkcí, numerické integrace a metod pro řešení nelineárních rovnic je k HTML souboru připojen obrázek s grafickým řešením dané úlohy. Vzhledem ke skutečnosti, že uživatel může do vstupních polí zadat libovolné výrazy, musel být napsán algoritmus pro automatický převod těchto výrazů do MathML podoby. Výsledná webová stránka je v aplikaci zobrazena pomocí *Awesomia*. Pro vykreslení všech matematických výrazů zapsaných pomocí MathML kódu je využita javascriptová knihovna *MathJax*. Aplikace dále umožňuje exportovat webovou stránku do formátu XML, HTML, JPG, BMP, PNG, TIFF, GIF nebo PDF. Při exportu do obrázkového formátu nebo formátu PDF je využito možností *Awesomia*, které ovšem v poslední verzi nepodporovalo *multithreading*. Nicméně tento nedostatek by měl být vývojáři *Awesomia* v nejbližší době odstraněn.

Součástí programu je dále sada cvičných úloh, k nimž může uživatel jednoduchým způsobem přidávat (respektive mazat) své úlohy včetně případné poznámky k dané

úloze. Jednotlivé úlohy každé metody jsou uloženy jako XML dokumenty.

Pro distribuci aplikace byla naprogramována webová stránka a zakoupena doména www.numerickemetody.cz. Na této adrese je přístupný text diplomové práce a samotná aplikace.

Během testování aplikace a po vydání její prvotní verze jsem se setkal ve většině případů s pozitivními reakcemi. Přičemž byl oceňován způsob zadání vlastních příkladů a výstupy metod s vykreslováním matematických výrazů včetně podpory exportu. Největšího ohlasu se ovšem dočkala možnost zobrazování mezivýsledků. Z těchto důvodů je aplikace studenty využívána nejen pro kontrolu při ručním počítání matematických úloh, ale i při ladění jejich programů.

Osobně jsem si výběrem této diplomové práce rozšířil znalosti studiem numerických algoritmů, a zároveň prohloubil své programátorské dovednosti z hlediska návrhu aplikace, kooperace většího počtu použitých technologií (v rámci aplikace jsou využity jazyky *C#*, HTML, MathML, XML, Javascript) algoritmizace vyčíslování výrazů pomocí počítače či práce s regulárními výrazy. Navíc jsem si psaním této diplomové práce osvojil základy LaTeX.

V budoucnu bych chtěl tuto aplikaci nadále rozvíjet a přidat metody například z oblasti numerické optimalizace či obyčejných a parciálních diferenciálních rovnic.

Literatura

[1] FAJMON, Břetislav a RŮŽIČKOVÁ Irena. *Matematika 3: Numerické metody* [online]. Brno, 2013 [cit. 2013-05-11]. Dostupné z: http://www.umat.feec.vutbr.cz/~novakm/matematika3.pdf. Vysokoškolská skripta. Vysoké učení technické v Brně.

[2] FELCMAN, Jiří. *Numerická matematika* [online].Praha, 2013 [cit. 2013-05-11]. Dostupné z: http://www.karlin.mff.cuni.cz/~felcman/nm.pdf. Vysokoškolská skripta. Univerzita Karlova v Praze.

[3] PŘIKRYL, Petr. *Numerické metody matematické analýzy.* 2. vyd. Praha: SNTL, 1988, 192 s.

[4] KUČERA, Radek. *Numerické metody* [online]. 2010 [cit. 2015-04-28]. Dostupné z: http://homel.vsb.cz/~kuc14/textyNM/FINALNI_VERZE_CD.pdf. Vysokoškolská skripta. Vysoká škola báňská - Technická univerzita Ostrava.

[5] SALAMON, Nicholas. *Chapter 4. Common Splines. [online].* [cit. 2013-05-15]. Dostupné z: http://www.esm.psu.edu/courses/emch407/njs/notes02/ch4_3.doc. PennState University.

[6] MOŠOVÁ, Vratislava. *Numerické metody* [online]. Ústí nad Labem, 2009 [cit. 2013-05-11]. Dostupné z:http://physics.ujep.cz/~jskvor/NME/DalsiSkripta/numerickemetody.pdf. Vysokoškolská skripta. Univerzita Jana Evangelisty Purkyně v Ústí nad Labem.

[7] MAŘÍK, Robert. *Metoda nejmenších čtverců.* [online]. [cit. 2013-05-15]. Dostupné z: http://user.mendelu.cz/marik/prez/mnc-cz.pdf. Mendelova univerzita v Brně.

[8] *Definice derivace* [online]. Praha, 2013 [cit. 2013-05-11]. Dostupné z: http://math.feld.cvut.cz/others/pavlu/teorie2.htm. České vysoké učení technické v Praze.

[9] LEVY, Doron. *Numerical Differentiation.* [online]. 2011 [cit. 2015-04-28]. Dostupné z: http://www2.math.umd.edu/~dlevy/classes/amsc466/lecture-notes/differentiation-chap.pdf.University of Maryland.

[10] VITÁSEK, Emil. *Numerické metody.* 1. vyd. Praha: SNTL, 1987, 516 s.

[11] SHAPIRO, Bruece. *Richardson Extrapolation* [online]. California, 2008 [cit. 2013-05-11]. Dostupné z: http://bruce-shapiro.com/math481A/notes/24-Richardson.pdf. Vysokoškolská skripta. California State University Northridge.

[12] DANĚK, Josef. *Derivace funkce* [online]. Plzeň, 2003 [cit. 2013-05-11]. Dostupné z: http://www.cam.zcu.cz/~danek/Students/2003_ZS/Materialy/derivace_integrace.pdf. Západočeská univerzita v Plzni.

[13] ČERMÁK, Libor a HLAVIČKA Rudolf. *Numerické metody: Numerický výpočet derivace a integrálu* [online]. Brno, 2006 [cit. 2013-05-11]. Dostupné z: http://mathonline.fme.vutbr.cz/UploadedFiles/243.pdf. Vysoké učení technické v Brně.

[14] DĚMIDOVIČ, Boris Pavlovič a MARON Isaak Abramovič. *Základy numerické matematiky*. 1. vyd. Praha: SNTL, 1966, 724 s.

[15] RALSTON, Anthony. *Základy numerické matematiky*. 2. vyd. Praha: Academia, 1978, 636 s.

[16] *Numerical Integration* [online]. Japonsko, 2009 [cit. 2013-05-11]. Dostupné z: http://netedu.xauat.edu.cn/jpkc/netedu/jpkc2009/jsff/content/syjx/3/Chapter4.pdf

[17] ZELINKA, Jiří a HOROVÁ Ivana. *Numerika* [online]. Brno, 2008 [cit. 2015-04-28]. Dostupné z: https://www.math.muni.cz/~zelinka/dokumenty/numerika.pdf. Masarykova univerzita v Brně.

[18] RIEČANOVÁ, Zdena et al. *Numerické metódy a matematická štatistika*. 3. vyd. Bratislava: Alfa, 1987, 496 s.

[19] QUARTERONI, Alfio, Riccardo SACCO a Fausto SALERI. *Numerical mathematics*. New York: Springer, c2000, xx, 654 p. ISBN 0-387-98959-5.

[20] LIMPOUCH, Jiří. *Numerická integrace*. [online]. [cit. 2013-05-15]. Dostupné z: http://kfe.fjfi.cvut.cz/~limpouch/numet/numint1.pdf. České Vysoké učení technické v Praze.

[21] VICHER, Miroslav. *Numerická matematika* [online]. 2003 [cit. 2015-04-28]. Dostupné z: http://physics.ujep.cz/~mlisal/nm_2/vicher_nm1.pdf. Vysokoškolská skripta. Univerzita Karlova v Praze.

[22] LEGRAS, Jean. *Metódy a použitie numerickej matematiky*. 1. vyd. Překlad Kamil Hrubina. Bratislava: Alfa, vydavateľstvo technickej a ekonomickej literatúry, 1978, 383 s. Edícia teoretickej literatúry.

[23] KOCUR, Pavel. *Numerické metody: Metoda tečen (Newtonova metoda)*[online]. Plzeň, 2000 [cit. 2013-05-11]. Dostupné z: http://www.kvd.zcu.cz/cz/materialy/numet/_numet.html#_Toc501178905. Západočeská univerzita v Plzni.

[24] WEN-CHIEH, Lin. *Solving Nonlinear Equation* [online]. Hsinchu, Taiwan, 2005 [cit. 2013-05-11]. Dostupné z: http://caig.cs.nctu.edu.tw/course/NM/chap1.pdf. National Chiao-Tung University.

[25] *Steffensen's method: Simple description*. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2013-05-11]. Dostupné z: http://en.wikipedia.org/wiki/Steffensen's_method.

[26] ALBEANU, Grigore. *On the generalized Halley method for solving nonlinear equations* [online]. Bucharest, 2008 [cit.2013-05-11]. Dostupné z: http://profs.info.uaic.ro/~jromai/romaijournal/arhiva/2008/2/RJv4n2-Albeanu.pdf. Spiru Haret University.

[27] HASÍK, Karel. *Numerické metody* [online]. Opava, 2006 [cit. 2013-05-11]. Dostupné z: http://www.slu.cz/math/cz/knihovna/ucebni-texty/Numericke-metody/Numericke-metody.pdf. Vysokoškolská skripta. Slezská univerzita v Opavě.

[28] MÍKA, Stanislav. *Numerické metody algebry*. 2., nezm. vyd. Praha: SNTL, 1985, 169 s. Matematika pro vysoké školy technické.

[29] ČERNÁ, Růžena. *Základy numerické matematiky a programování: celostátní vysokoškolská učebnice pro strojní, elektrotechnické a stavební fakulty vysokých škol technických*. 1. vyd. Praha: Státní nakladatelství technické literatury, 1987, 445 s.

[30] LÍSAL, Martin. *Přímé metody řešení soustav lineárních rovnic* [online]. Chomutov, 2009 [cit. 2013-05-11]. Dostupné z: http://physics.ujep.cz/~mlisal/nm_1-chomutov/jkrejci/texty/Prime%20metody.pdf. Univerzita Jana Evangelisty Purkyně v Ústí nad Labem.

[31] NATARAJ, Neela. *Gauss Elimination method with partial pivoting* [online]. Bombay, 2008 [cit. 2013-05-11]. Dostupné z: http://www.math.iitb.ac.in/~neela/partialpivot.pdf. Indian Institute of Technology Bombay.

[32] ČERNÁ, Dana. *Numerické metody lineární algebry* [online]. Liberec, 2013 [cit. 2013-05-11]. Dostupné z: https://kmd.fp.tul.cz/images/stories/vyuka/cerna-matematika3-fs/num_lin_alg.pdf. Vysokoškolská skripta. Technická univerzita v Liberci.

[33] OLŠÁK, Petr. *LU rozklad* [online]. Praha, 2010 [cit. 2013-05-11]. Dostupné z: http://petr.olsak.net/bilin/lurozklad.pdf.

[34] ČERNÝ, Pavel. *Metoda SOR (Successive OverRelaxation)* [online]. Ústí nad Labem [cit. 2013-05-11]. Dostupné z: http://physics.ujep.cz/~mlisal/nm_2-chomutov/pcerny/files/sor_m.html. Univerzita Jana Evangelisty Purkyně v Ústí nad Labem.

[35] HLAVÁČ, Zdeněk. *Řešení soustav lineárních algebraických rovnic*. [online]. [cit. 2015-04-27]. Dostupné z: http://www.kme.zcu.cz/download/predmety/399-eseni-soustav-linearnich-algebraickych-rovnic.pdf. Západočeská univerzita v Plzni.

[36] OLŠÁK, Petr. *Vlastní číslo, vektor*. [online]. 2012 [cit. 2015-05-02]. Dostupné z: http://petr.olsak.net/bilin/vlcisla-v2.pdf

[37] BAŠTINEC, Jaromír a Michal NOVÁK. *Moderní numerické metody* [online]. Brno [cit. 2015-04-28]. Dostupné z: http://matika.umat.feec.vutbr.cz/inovace/materialy/skripta/mmnm.pdf. Vysoké učení technické v Brně.

[38] *Symetrické matice*. [online]. [cit. 2015-04-28]. Dostupné z: https://math.feld.cvut.cz/ftp/vyuka/MA5/SymetrMatice.pdf. České vysoké učení technické v Praze.

[39] TEKNOMO, Kardi. *Similarity Transformation and Matrix Diagonalization* [online]. 2011 [cit. 2015-04-28]. Dostupné z: http://people.revoledu.com/kardi/tutorial/LinearAlgebra/MatrixDiagonalization.html.

[40] OKŠA, Gabriel. *Úvod do numerických metód lineárnej algebry*. 1. vyd. Bratislava: Nakladateľstvo STU, 2009, 106 s. Edícia skrípt. ISBN 978-80-227-3055-6.

[41] VONDRÁK, Vít a POSPÍŠIL Lukáš. *NUMERICKÉ METODY I* [online]. Ostrava, 2011 [cit. 2015-04-28]. Dostupné z: http://mi21.vsb.cz/sites/mi21.vsb.cz/files/unit/numericke_metody.pdf. Západočeská univerzita v Plzni a Vysoká škola báňská – Technická univerzita Ostrava.

[42] HÖSCHL, Cyril. *VYUŽITÍ MALÝCH POČÍTAČŮ PRO PRÁCI KONSTRUKTÉRA* [online]. Praha: Dům techniky ČSVTS Praha, 1981 [cit. 2015-04-28]. Dostupné z: http://www.it.cas.cz/files/skripta/11_VYUZ_MALYCH_POCITACU_PRO_PRACI_KONSTRUKTERA-ocr.pdf. Ústav termomechaniky AV ČR.

[43] GAO, Tangan. *Householder's method*. [online]. 2000 [cit. 2015-04-28]. Dostupné z: http://web.csulb.edu/~tgao/math423/s93.pdf.California State University.

[44] SOJKA, Radim. *Paralelní implementace ortogonalizace matice* [online]. Ostrava, 2013 [cit. 2015-04-28]. Dostupné z: http://www.fei.vsb.cz/export/sites/fei/k470/cs/theses/bakalari/2013/soj0018.pdf. Bakalářská práce. Technická univerzita Ostrava, Fakulta elektrotechniky a informatiky, Katedra informatiky.

[45] DOMPIERRE, Julien. *Householder Reflections and Givens Rotations Matrix Computations — CPSC 5006 E* [online]. 2010 [cit. 2015-04-28]. Dostupné z: http://www.cs.laurentian.ca/jdompierre/html/CPSC5006E_F2010/cours/ch05_Householder_Givens.pdf. Laurentian University.

[46] *Odvození transformačních matic pro různé typy rotace*. [online]. [cit. 2015-05-02]. Dostupné z: http://mineralogie.sci.muni.cz/kap_1_3_symetrie/rotace_priklad.htm. Masarykova univerzita v Brně.

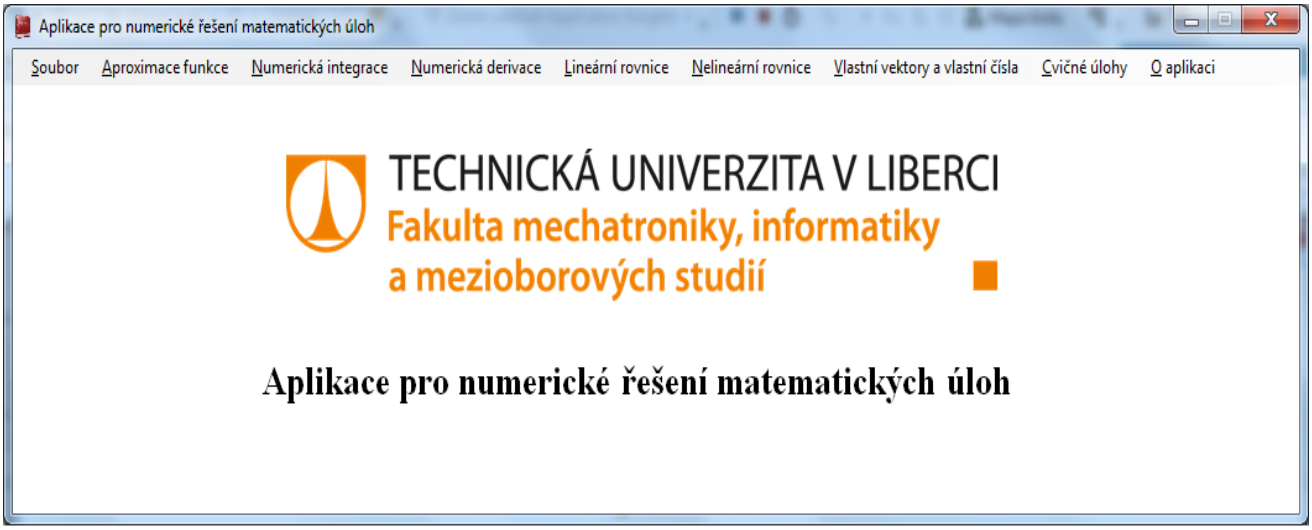
[47] KUBÍČEK, Milan, DUBCOVÁ Miroslava a JANOVSKÁ Drahoslava. *Numerické metody a algoritmy: Givensovy matice rovinné rotace* [online]. 2. vydání. Praha: VŠCHT, 2005, s. 167 [cit. 2015-04-28]. ISBN 80-7080-558-7. Dostupné z: http://vydavatelstvi.vscht.cz/knihy/uid_isbn-80-7080-558-7/pages-img/167.html.

[48] JANOVSKÁ, Drahoslava. *Matematika pro chemické inženýry*. [online]. 2011 [cit. 2015-04-28]. Dostupné z: https://vscht.cz/mat/MCHI/1_prednaska.pdf. Vysoká škola chemicko-technologická v Praze.

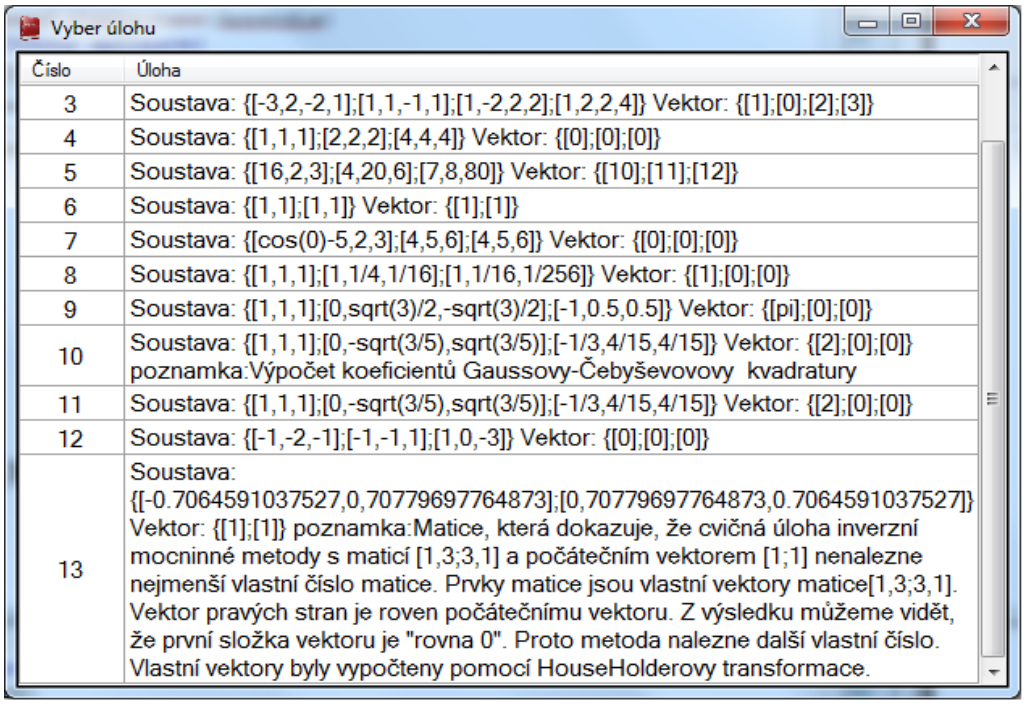
[49] BAYER. *Výpočet hodnot aritmetického výrazu: Infixová notace, Prefixová notace, Postfixová notace* [online]. Praha, 2013 [cit. 2013-05-11]. Dostupné z: http://web.natur.cuni.cz/~bayertom/Prog2/prog2_3.pdf. Vysokoškolská skripta. Univerzita Karlova v Praze.

[50] *Shunting-yard algorithm*. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2015-04-27]. Dostupné z: http://en.wikipedia.org/wiki/Shunting-yard_algorithm.

A Ukázky aplikace

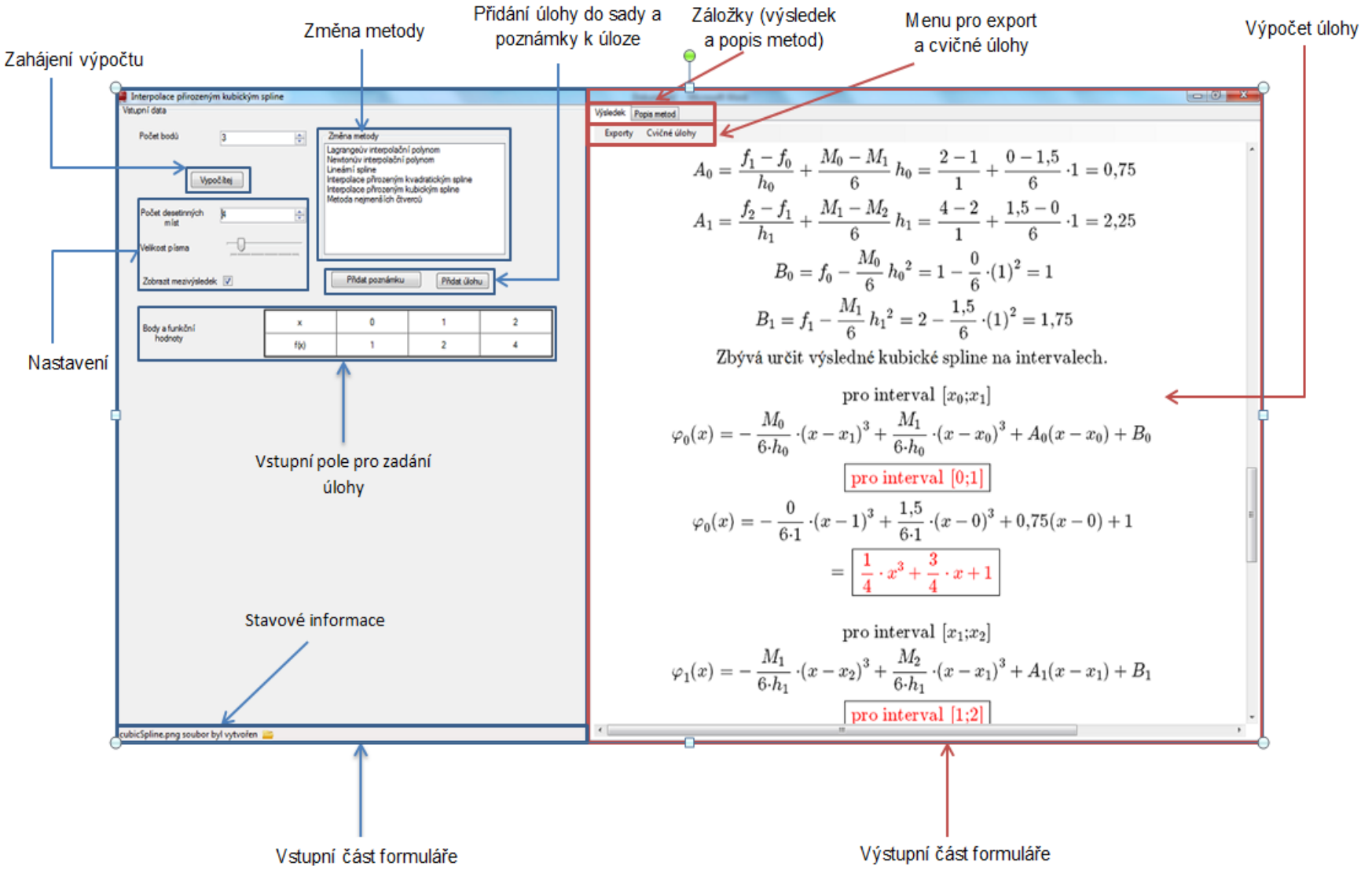


Obrázek A.1: Úvodní obrazovka aplikace



Číslo	Úloha
3	Soustava: {[-3,2,-2,1];[1,1,-1,1];[1,-2,2,2];[1,2,2,4]} Vektor: {[1];[0];[2];[3]}
4	Soustava: {[1,1,1];[2,2,2];[4,4,4]} Vektor: {[0];[0];[0]}
5	Soustava: {[16,2,3];[4,20,6];[7,8,80]} Vektor: {[10];[11];[12]}
6	Soustava: {[1,1];[1,1]} Vektor: {[1];[1]}
7	Soustava: {[cos(0)-5,2,3];[4,5,6];[4,5,6]} Vektor: {[0];[0];[0]}
8	Soustava: {[1,1,1];[1,1/4,1/16];[1,1/16,1/256]} Vektor: {[1];[0];[0]}
9	Soustava: {[1,1,1];[0,sqrt(3)/2,-sqrt(3)/2];[-1,0.5,0.5]} Vektor: {[pi];[0];[0]}
10	Soustava: {[1,1,1];[0,-sqrt(3/5),sqrt(3/5)];[-1/3,4/15,4/15]} Vektor: {[2];[0];[0]} poznamka:Výpočet koeficientů Gaussovy-Čebyševovovy kvadratury
11	Soustava: {[1,1,1];[0,-sqrt(3/5),sqrt(3/5)];[-1/3,4/15,4/15]} Vektor: {[2];[0];[0]}
12	Soustava: {[-1,-2,-1];[-1,-1,1];[1,0,-3]} Vektor: {[0];[0];[0]}
13	Soustava: {[-0.7064591037527,0,70779697764873];[0,70779697764873,0.7064591037527]} Vektor: {[1];[1]} poznamka:Matice, která dokazuje, že cvičná úloha inverzní mocninné metody s maticí [1,3;3,1] a počátečním vektorem [1;1] nenalezne nejmenší vlastní číslo matice. Prvky matice jsou vlastní vektory matice[1,3;3,1]. Vektor pravých stran je roven počátečnímu vektoru. Z výsledku můžeme vidět, že první složka vektoru je "rovna 0". Proto metoda nalezne další vlastní číslo. Vlastní vektory byly vypočteny pomocí HouseHolderovy transformace.

Obrázek A.2: Cvičné úlohy metody Gaussovy eliminace



Obrázek A.3: Ukázka návrhu formuláře pro výpočet úloh pomocí interpolace přirozeným kubickým splinem

Předpis zvolené metody:

$$L_n(x) = P_0(x) + P_1(x) + \ldots + P_n(x) = f(x_0)l_0(x) + f(x_1)l_1(x) + \ldots + f(x_n)l_n(x) = \sum_{i=0}^{n} f(x_i)l_i(x), \text{ kde } P_i(x) \text{ jsou pomocné polynomy.}$$

$$l_i(x_j) = \begin{cases} 1 \text{ pro i = } j \\ 0 \text{ pro i} \neq j \end{cases}$$

tyto podmínky splňuje polynom:

$$l_i(x) = \prod_{\substack{0 \le m \le n \\ m \neq i}} \frac{x - x_m}{x_i - x_m} = \frac{x - x_0}{x_i - x_0} \ldots \frac{x - x_{i-1}}{x_i - x_{i-1}} \frac{x - x_{i+1}}{x_i - x_{i+1}} \ldots \frac{x - x_n}{x_i - x_n}$$

Výsledek

Byly zadány body a jejich funkční hodnoty:

x	-4	3	5	7
$f(x)$	17	10	26	50

Nejprve určíme polynomy $l_i(x_j)$.

$$l_0(x) = \frac{x - x_1}{x_0 - x_1} \cdot \frac{x - x_2}{x_0 - x_2} \cdot \frac{x - x_3}{x_0 - x_3} = \frac{x - 3}{-4 - 3} \cdot \frac{x - 5}{-4 - 5} \cdot \frac{x - 7}{-4 - 7} = \frac{x^3 - 15 \cdot x^2 + 71 \cdot x - 105}{-693} = (-0{,}00144) \cdot x^3 + 0{,}02165 \cdot x^2 - 0{,}10245 \cdot x + 0{,}15152$$

$$l_1(x) = \frac{x - x_0}{x_1 - x_0} \cdot \frac{x - x_2}{x_1 - x_2} \cdot \frac{x - x_3}{x_1 - x_3} = \frac{x + 4}{3 + 4} \cdot \frac{x - 5}{3 - 5} \cdot \frac{x - 7}{3 - 7} = \frac{x^3 - 8 \cdot x^2 - 13 \cdot x + 140}{56} = 0{,}01786 \cdot x^3 - 0{,}14286 \cdot x^2 - 0{,}23214 \cdot x + \frac{5}{2}$$

$$l_2(x) = \frac{x - x_0}{x_2 - x_0} \cdot \frac{x - x_1}{x_2 - x_1} \cdot \frac{x - x_3}{x_2 - x_3} = \frac{x + 4}{5 + 4} \cdot \frac{x - 3}{5 - 3} \cdot \frac{x - 7}{5 - 7} = \frac{x^3 - 6 \cdot x^2 - 19 \cdot x + 84}{-36} = (-0{,}02778) \cdot x^3 + \frac{1}{6} \cdot x^2 + 0{,}52778 \cdot x - \frac{7}{3}$$

$$l_3(x) = \frac{x - x_0}{x_3 - x_0} \cdot \frac{x - x_1}{x_3 - x_1} \cdot \frac{x - x_2}{x_3 - x_2} \cdot = \frac{x + 4}{7 + 4} \cdot \frac{x - 3}{7 - 3} \cdot \frac{x - 5}{7 - 5} = \frac{x^3 - 4 \cdot x^2 - 17 \cdot x + 60}{88} = 0{,}01136 \cdot x^3 - 0{,}04545 \cdot x^2 - 0{,}19318 \cdot x + 0{,}68182$$

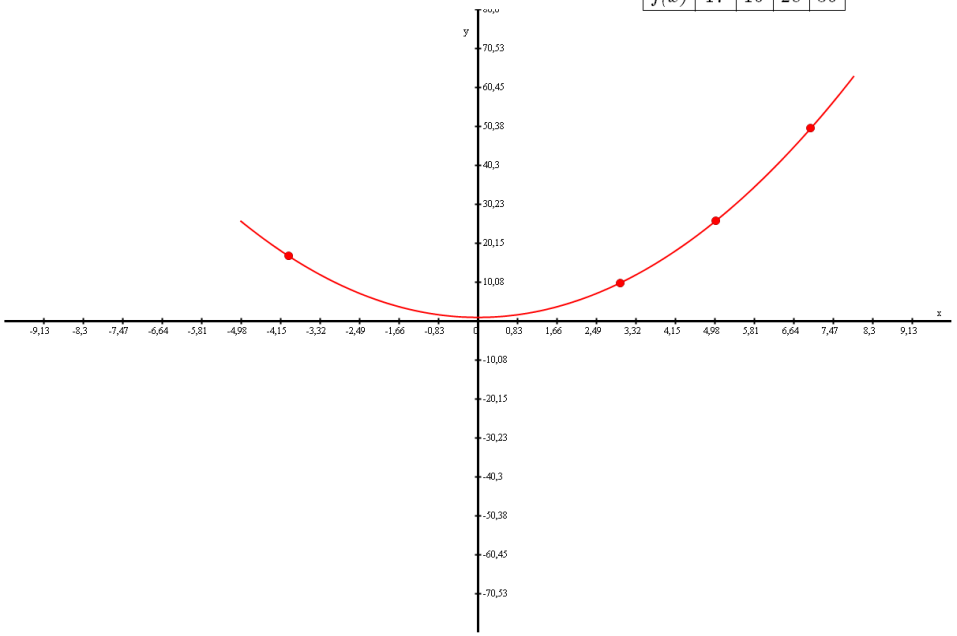
Nyní určíme výsledný interpolační polynom.

$$L_3(x) = f(x_0)l_0(x) + f(x_1)l_1(x) + f(x_2)l_2(x) + f(x_3)l_3(x) = 17 \cdot \left((-0{,}00144) \cdot x^3 + 0{,}02165 \cdot x^2 - 0{,}10245 \cdot x + 0{,}15152\right) + 10 \cdot \left(0{,}01786 \cdot x^3 - 0{,}14286 \cdot x^2 - 0{,}23214 \cdot x + \frac{5}{2}\right)$$

$$+ 26 \cdot \left((-0{,}02778) \cdot x^3 + \frac{1}{6} \cdot x^2 + 0{,}52778 \cdot x - \frac{7}{3}\right) + 50 \cdot \left(0{,}01136 \cdot x^3 - 0{,}04545 \cdot x^2 - 0{,}19318 \cdot x + 0{,}68182\right) = \boxed{x^2 + 1}$$

Grafické znázornění aproximace funkce pro uzlové body

x	-4	3	5	7
$f(x)$	17	10	26	50



Zvolená metoda: Metoda nejmenších čtverců

Předpis zvolené metody:

$$P(x) = \sum_{i=0}^{n} b_i x^i$$

$$nb_0 + \sum_{i=1}^{n} x_i b_1 + \sum_{i=1}^{n} {x_i}^2 b_2 + \ldots\ldots\ldots + \sum_{i=1}^{n} {x_i}^m b_m = \sum_{i=1}^{n} y_i$$

$$\sum_{i=1}^{n} x_i b_0 + \sum_{i=1}^{n} {x_i}^2 b_1 + \ldots\ldots\ldots + \sum_{i=1}^{n} {x_i}^{m+1} b_m = \sum_{i=1}^{n} x_i y_i$$

$$\sum_{i=1}^{n} {x_i}^2 b_0 + \sum_{i=1}^{n} {x_i}^3 b_1 + \ldots\ldots\ldots + \sum_{i=1}^{n} {x_i}^{m+2} b_m = \sum_{i=1}^{n} {x_i}^2 y_i$$

$$\vdots$$

$$\sum_{i=1}^{n} {x_i}^m b_0 + \sum_{i=1}^{n} {x_i}^{m+1} b_1 + \ldots\ldots\ldots + \sum_{i=1}^{n} {x_i}^{2m} b_m = \sum_{i=1}^{n} {x_i}^m y_i.$$

Výsledek

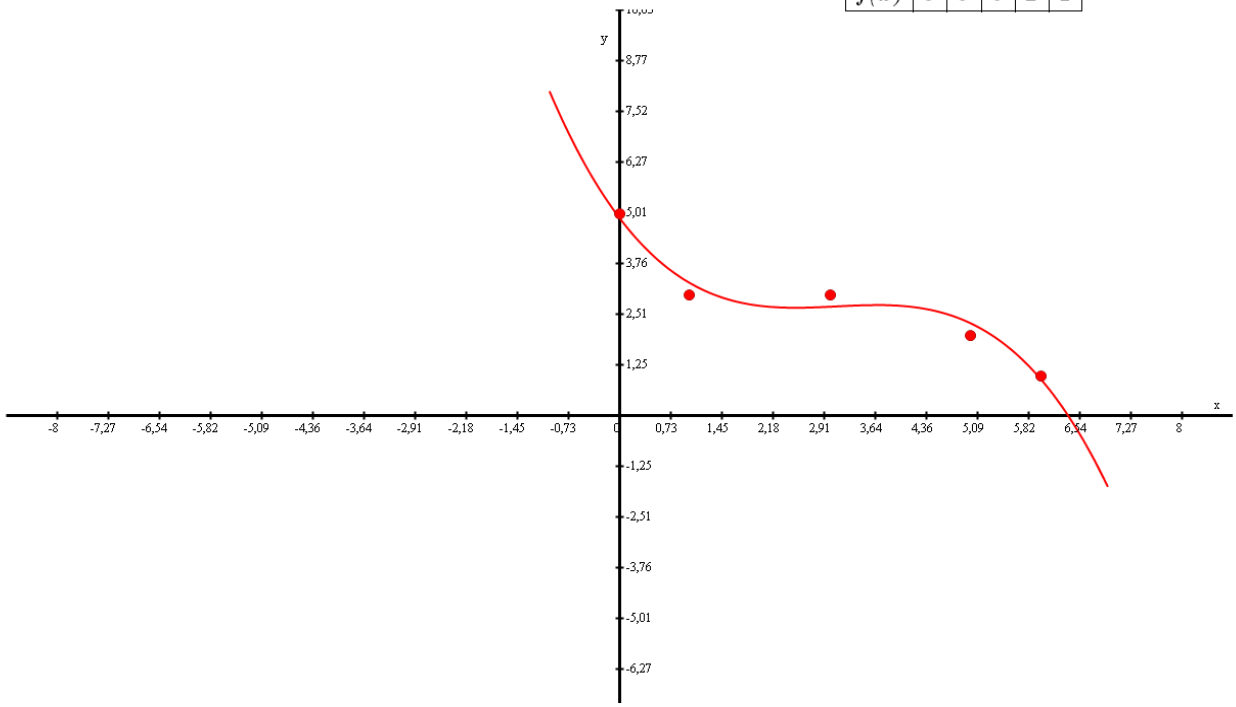
Byly zadány body a jejich funkční hodnoty:

x	0	1	3	5	6
$f(x)$	5	3	3	2	1

$$P(x) = \boxed{(-0{,}08333) \cdot x^3 + 0{,}77041 \cdot x^2 - 2{,}28912 \cdot x + 4{,}87755}$$

Grafické znázornění aproximace funkce pro uzlové body

x	0	1	3	5	6
$f(x)$	5	3	3	2	1



Obrázek A.5: Řešení aproximační úlohy pomocí metody nejmenších čtverců (bez postupu výpočtu)

Byl zadán polynom $P(x) = x^6 + 4 \cdot x^5 + 4 \cdot x^4 - x^2 - 4 \cdot x - 4$

Úkolem je určit počet a lokalizovat reálné kořeny polynomu na intervalu:$(-\infty; +\infty)$

S přesností: $\frac{1}{100}$

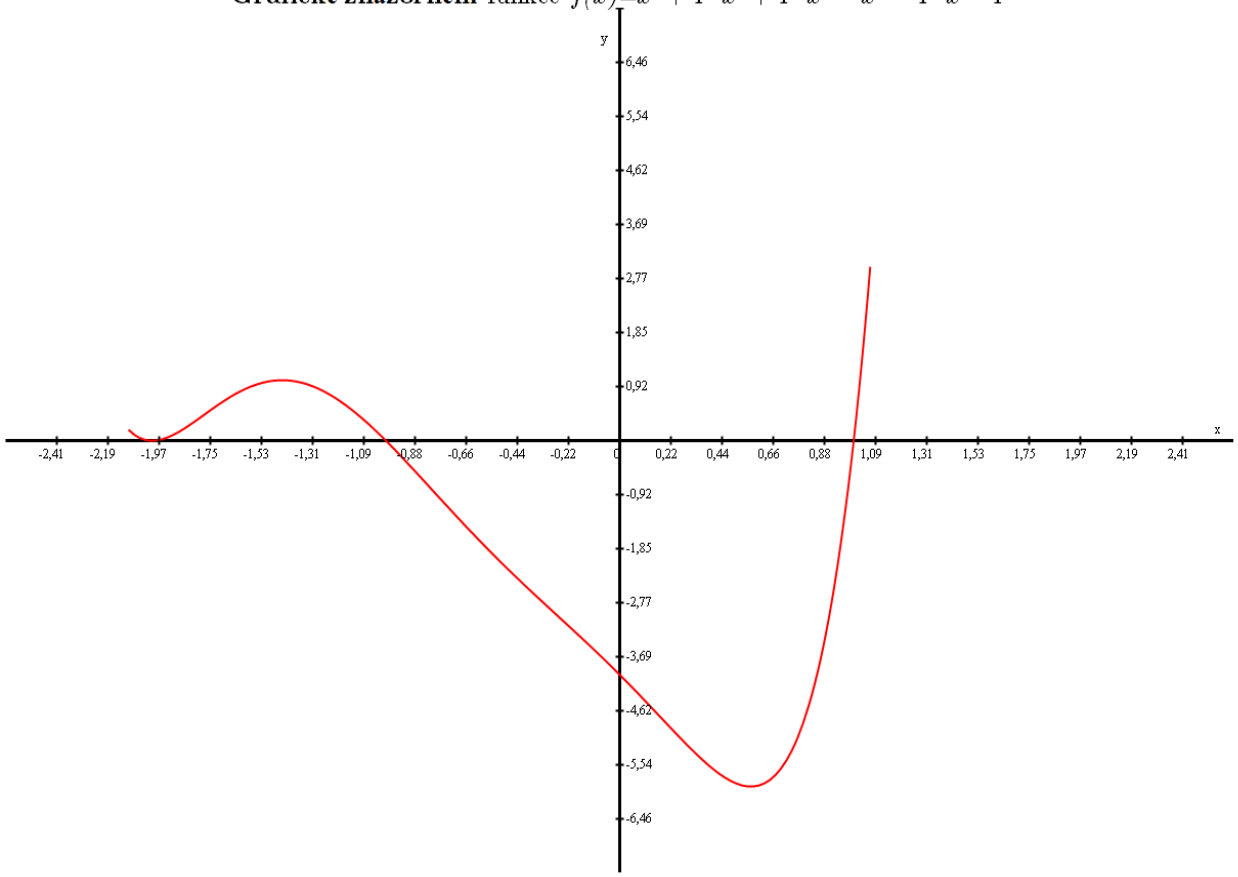
Kořeny leží v intervalech:

1. Kořen leží v intervalu [-2,00888834390044;-1,99520400201436) .

2. Kořen leží v intervalu [-1,00044417195022;-0,99475983006414) .

3. Kořen leží v intervalu [0,99475983006414;1,00044417195022) .

Grafické znázornění funkce $f(x)=x^6 + 4 \cdot x^5 + 4 \cdot x^4 - x^2 - 4 \cdot x - 4$



Obrázek A.6: Nalezení reálných kořenů polynomu (bez postupu výpočtu)

Byla zadána funkce: $f(x) = x^3 - x^2$.

Úkolem je určit hodnotu integrálu funkce f na intervalu: $[-5;6]$.

Zvolená metoda: Lichoběžníkové pravidlo

Předpis zvolené metody:

$$\int_a^b f(x)\mathrm{dx} \approx \frac{b-a}{2}\left[f(a) + f(b)\right]$$

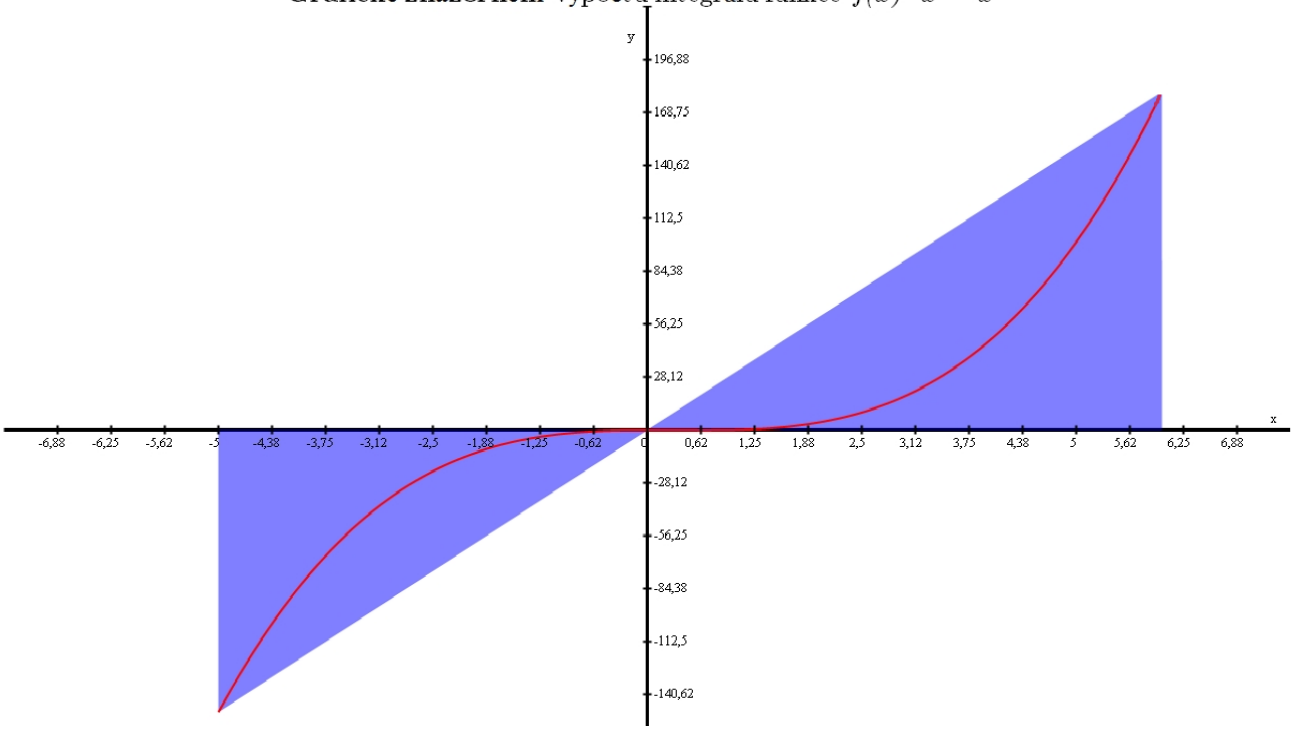
Výsledek

$$\left(\frac{6+5}{2}\right) \cdot (f(-5) + f(6)) =$$

$$= \frac{11}{2} \cdot (-150 + 180) = \boxed{165}$$

$$\int_{-5}^{6} x^3 - x^2\, dx \approx \boxed{165}$$

Grafické znázornění výpočtu integrálu funkce $f(x) = x^3 - x^2$



Obrázek A.7: Výpočet určitého integrálu lichoběžníkovým pravidlem

Byla zadána funkce: $f(x) = x^3 - x^2$.

Úkolem je určit hodnotu integrálu funkce f na intervalu: $[-5;6]$.

Zvolená metoda: Složené lichoběžníkové pravidlo

Předpis zvolené metody:

$$\int_a^b f(x)dx \approx \frac{h}{2}\left[f(x_0) + 2\sum_{i=1}^{m-1} f(x_i) + f(x_m)\right], \text{ kde } h = \frac{b-a}{m} \text{ a } x_i = a + ih$$

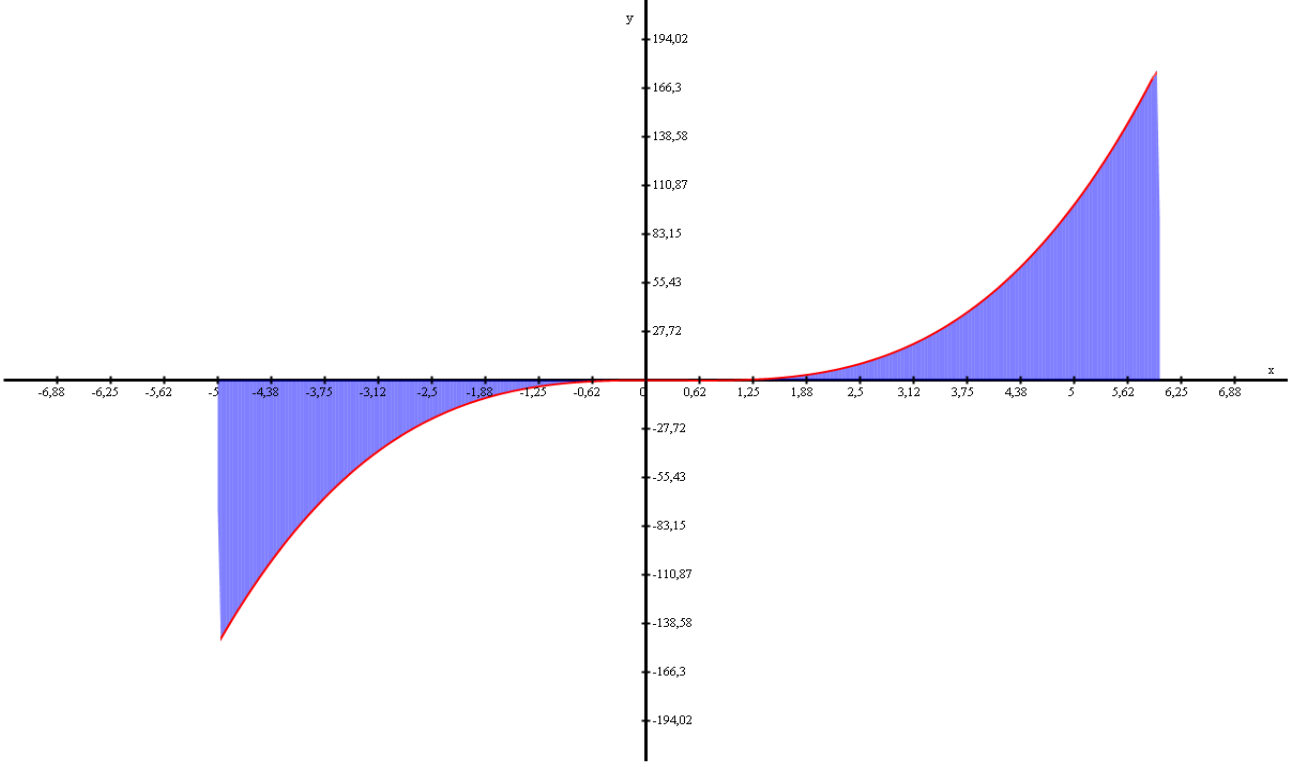
Počet dílů, na který bude interval rozdělen: 160

Krok h metody : $\frac{6+5}{160} = \frac{11}{160}$

Výsledek

Výsledná hodnota integrálu $\int_{-5}^{6} x^3 - x^2\,dx \approx$ 54,08442 + 0,00434

Grafické znázornění výpočtu integrálu funkce $f(x)=x^3 - x^2$



Obrázek A.8: Výpočet určitého integrálu složeným lichoběžníkovým pravidlem pomocí 160 podintervalů (bez postupu výpočtu)

Byla zadána funkce: $f(x) = sin\left(\frac{x+\sqrt{2}}{2^3}.\right)$

Úkolem je určit hodnotu první derivace v bodě: 5 **s krokem:** $\frac{1}{1000}$ **a přesností:** $\frac{1}{100}$.

Zvolená metoda: Richardsonova extrapolace

Předpis zvolené metody:

$$R_{i,j} = R_{i,j-1} + \frac{R_{i,j-1} - R_{i-1,j-1}}{4^j - 1}$$

Výsledek

$h[s]$	$T[s,0]$	$T[s,1]$	$T[s,2]$
0,001	0,086928885673299	0	0
0,0005	0,086928885842941	0,086928885899488	0
0,00025	0,086928885885573	0,086928885899784	0,086928885899804

$$f'(5) = \boxed{0{,}086928885899804}$$

Obrázek A.9: Výpočet první derivace funkce v bodě pomocí Richardsonovy extrapolace

Zvolená metoda: LU rozklad

Předpis zvolené metody:

$$\boldsymbol{Ax} = \boldsymbol{b}, \text{kde } \boldsymbol{A} = \boldsymbol{LU}$$
$$\boldsymbol{LUx} = \boldsymbol{b}$$
$$\boldsymbol{Ly} = \boldsymbol{b}$$
$$\boldsymbol{Ux} = \boldsymbol{y}$$

Vztahy pro výpočet matice $\boldsymbol{L}$ a $\boldsymbol{U}$

$$l_{rr} = 1$$
$$u_{11} = a_{11}$$
$$l_{i1} = \frac{a_{i1}}{u_{11}}$$
$$u_{1r} = a_{1r}$$

$$u_{ir} = a_{ir} - \sum_{j=1}^{i-1} l_{ij} u_{jr}\ , \text{kde } \ i = 2, \ldots, r$$

$$l_{ir} = \frac{1}{u_{rr}} \left(\sum_{j=1}^{r-1} l_{ij} u_{jr} \right)\ , \text{kde } \ i = r+1, \ldots, n.$$

Poté můžeme řešit soustavy $\boldsymbol{Ly} = \boldsymbol{Pb}$

a $\boldsymbol{Ux} = \boldsymbol{y}$

tímto způsobem:

$$y_i = b_i - \sum_{k=1}^{i-1} l_{ik} y_k\ , \text{kde } \ i = 1, \ldots, n$$

$$x_i = \frac{1}{u_{ii}} \left(y_i - \sum_{k=i+1}^{n} u_{ik} x_k \right)\ , \text{kde } \ i = n, \ldots, 1.$$

Byla zadána soustava: $\left(\begin{array}{ccc|c} 4 & -1 & 2 & -7 \\ 1 & cos(2 \cdot \pi) & 2 & sin(0) \\ 2+3 & -1 & -3 & 9 \end{array} \right)$

Výsledek

$$\boldsymbol{x} = \boxed{\begin{pmatrix} 1 \\ 5 \\ -3 \end{pmatrix}}$$

Obrázek A.10: Výpočet soustavy lineárních rovnic pomocí LU rozkladu (bez postupu výpočtu)

Zvolená metoda: QR rozklad - Householderova transformace

Byla zadána matice: $\begin{pmatrix} 6 & 4 & 4 & 1 \\ 4 & 6 & 1 & 4 \\ 4 & 1 & 6 & 4 \\ 1 & 4 & 4 & 6 \end{pmatrix}$

Počet iterací : 20
Úkolem je nalézt vlastní čísla a k nim příslušné vlastní vektory vstupní matice.

Předpis zvolené metody:

QR rozklad matice $\boldsymbol{A} = \boldsymbol{QR}$, kde $\boldsymbol{Q}$ je ortogonální matice a R je horní trojúhelníková matice

$$\boldsymbol{A_k} = \boldsymbol{R_k Q_k}.$$

Pro reálné symetrické matice konverguje matice $\boldsymbol{A_k}$ k diagonální matici, která je podobná původní matici $\boldsymbol{A}$.
Diagonální matice má stejná vlastní čísla jako původní matice.
Příslušné vlastní vektory nalezneme ve sloupcích matice $\boldsymbol{Q}$.
Matice $\boldsymbol{Q} = \boldsymbol{Q_1 Q_2 \ldots . Q_k}$
QR rozklad příslušné matice určíme pomocí Householderovy transformace.

$$\boldsymbol{H_k} = \boldsymbol{I} - 2\frac{\omega_k \omega_k^T}{\omega_k^T \omega_k}$$

$$\chi = -\text{sign}(v_i)\|v\|e_1 - v$$

$$\omega = \frac{\chi}{\|\chi\|^2}$$

$$\boldsymbol{A^{(n)}} = \boldsymbol{H_n H_{n-1} \ldots H_1} = \boldsymbol{R}$$

$$\boldsymbol{Q} = \boldsymbol{H_1 H_2 \ldots H_n}$$

Výsledek

K vlastnímu číslu $\lambda_0 = \boxed{15}$ přísluší vlastní vektor

$$\boxed{\begin{pmatrix} \frac{1}{2} \\ \frac{1}{2} \\ \frac{1}{2} \\ \frac{1}{2} \end{pmatrix}}.$$

K vlastnímu číslu $\lambda_1 = \boxed{5}$ přísluší vlastní vektor

$$\boxed{\begin{pmatrix} -\frac{1}{2} \\ \frac{1}{2} \\ -\frac{1}{2} \\ \frac{1}{2} \end{pmatrix}}.$$

K vlastnímu číslu $\lambda_2 = \boxed{5}$ přísluší vlastní vektor

$$\boxed{\begin{pmatrix} -\frac{1}{2} \\ -\frac{1}{2} \\ \frac{1}{2} \\ \frac{1}{2} \end{pmatrix}}.$$

K vlastnímu číslu $\lambda_3 = \boxed{-1}$ přísluší vlastní vektor

$$\boxed{\begin{pmatrix} \frac{1}{2} \\ -\frac{1}{2} \\ -\frac{1}{2} \\ \frac{1}{2} \end{pmatrix}}.$$

Obrázek A.11: Výpočet vlastních čísel a vlastních vektorů symetrické matice pomocí QR algoritmu (Householderovy transformace) s 20 iteracemi (bez postupu výpočtu)

Obrázek A.12: Webová stránka pro distribuci aplikace

B Obsah DVD

Na přiloženém DVD nalezneme:

1. Text diplomové práce ve formátu PDF
2. Aplikaci pro numerické řešení matematických úloh
3. Aplikaci pro numerické řešení matematických úloh ve formátu *.zip*
4. Zdrojové kódy Aplikace pro numerické řešení matematických úloh