UNIVERZITA PARDUBICE
FAKULTA ELEKTROTECHNIKY
A INFORMATIKY

# BAKALÁŘSKÁ PRÁCE

2009 Jiří Engliš

Univerzita Pardubice
**Fakulta elektrotechniky a informatiky**

# Výpočet kořenů polynomu n–tého stupně

**Jiří Engliš**

**Bakalářská práce**
**2009**

**Univerzita Pardubice**
**Fakulta elektrotechniky a informatiky**
**Katedra informačních technologií**
Akademický rok: **2008/2009**

# ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení: **Jiří ENGLIŠ**

Studijní program: **B2646 Informační technologie**

Studijní obor: **Informační technologie**

Název tématu: **Výpočet kořenů polynomu n-tého stupně**

Zásady pro vypracování:

Rovnici n-tého stupně an*$x^n$ + a(n-1)*$x^{(n-1)}$ + ... + a0 = 0 lze přesně vyřešit jen pro přirozené n<5. V případě většího n se používají přibližné metody. Cílem teoretické části práce bude studium a popis algoritmů na výpočet kořenů polynomů a výběr vhodného algoritmu pro co nejrychlejší nalezení všech komplexních řešení polynomiální rovnice n-tého řádu. Cílem aplikační části bude vytvoření programu pro výpočet všech přibližných řešení (v komplexním oboru). Uživatel zadá rovnici a požadovanou přesnost. Program zobrazí výsledek, popřípadě chybovou hlášku.

Rozsah grafických prací:

Rozsah pracovní zprávy:

Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam odborné literatury:

**TOPFER, P.: Algoritmy a programovací techniky, Prometheus 1995**
**ŠISLER, M. - ANDRYS, J.: O řešení algebraických rovnic. Mladá fronta, Praha 1966**

Vedoucí bakalářské práce: **RNDr. Josef Rak**
Katedra informačních technologií

Datum zadání bakalářské práce: **15. ledna 2009**

Termín odevzdání bakalářské práce: **15. května 2009**

doc. Ing. Simeon Karamazov, Dr.
děkan

L.S.

Ing. Lukáš Čegan
vedoucí katedry

V Pardubicích dne 31. března 2009

Prohlašuji:

Tuto práci jsem vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Souhlasím s prezenčním zpřístupněním své práce v Univerzitní knihovně.

V Pardubicích dne 30. 4. 2009

Jiří Engliš

## SOUHRN

Práce se zabývá polynomy, zvláště způsoby hledání jejich kořenů. Uvádí metody přímého výpočtu pro polynomy stupně menšího než pátého a některé metody přibližné pro polynomy obecné. Nakonec provádí porovnání některých přibližných metod.

## KLÍČOVÁ SLOVA

Polynom, řešení polynomu, kořeny polynomu, přibližné metody, přímé metody

## TITLE

Computing the roots of n–th degree polynomials

## SUMMARY

The thesis deals with polynomes, especially with methods for finding their roots. It lists direct copmuting methods for polynomes of degree less than five and some approximate methods for common polynomials. In the end it presents a comparison of some of the approximate methods.

## KEYWORDS

Polynome, Polynome solving, Polynome roots, Approximate methods, Direct methods

## OBSAH

## SEZNAM OBRÁZKŮ



## SEZNAM TABULEK

# 1 ÚVOD

Problém polynomů a jejich řešení je velmi starý. Přestože jsou to jednoduché funkce, hledání bodů, pro které je hodnota rovna nule, je velmi složité, zvláště se zvyšujícím se stupněm polynomu. V minulosti se tímto problémem zabývala spousta lidí a vymyslela několik postupů, jak jej řešit.

V této práci se pokusím rozebrat některé metody pro hledání kořenů polynomu, vyzkoušet je a porovnat jejich efektivitu a přesnost.

# 2 POLYNOM

Polynom je matematická funkce, kterou můžeme zapsat ve tvaru:

$$y = P(x) = a_n x^n + a_{n-1} x^{n-1} + a_{n-2} x^{n-2} + \ldots + a_1 x + a_0 \tag{1}$$

Funkci (1) také můžeme upravit na tvar

$$y = P(x) = x^n + \frac{a_{n-1}}{a_n} x^{n-1} + \frac{a_{n-2}}{a_n} x^{n-2} + \ldots + \frac{a_1}{a_n} x + \frac{a_0}{a_n} \tag{2}$$

Polynom můžeme vyjádřit i jinými způsoby, a sice pomocí jeho lineárních faktorů:

$$y = P(x) = (x - x_n)(x - x_{n-1}) \ldots (x - x_1)(x - x_0) \tag{3}$$

nebo zvláštním zápisem, který umožňuje jednodušší výpočet hodnoty, využívaným např. v Hornerově schématu

$$y = P(x) = ((\ldots(((a_n)x + a_{n-1})x + a_{n-2})x + \ldots + a_1)x + a_0) \tag{4}$$

U polynomů nás obvykle zajímají jejich hodnoty v bodě *x*, derivace, integrály a hodnotu *x*, pro kterou platí $P(x) = 0$ nebo $P(x) = y$.

Řešení prvních tří problémů je velmi jednoduché. Jde pouze o dosazení do vzorce, respektive o velmi jednoduchou derivaci a integraci součtu jednočlenů. Rovnost

hodnotě $y$ můžeme převést na problém $P(x) = 0$, pokud k polynomu přičteme konstantu $C = -y$. Avšak hledání kořenů, tedy hodnot $x$, pro které je $P(x) = 0$ je již záležitostí velmi složitou.

## 2.1 KOŘENY POLYNOMU

Kořen polynomu je hodnota $x$, pro kterou platí $P(x) = 0$. Obecně můžeme říct, že polynom n–tého stupně má n komplexních kořenů, počítáme-li všechny násobné kořeny zvlášť. Jsou-li všechny koeficienty polynomu $P(x)$ reálné, pak také platí, že ke každému komplexnímu kořenu polynomu $P(x)$, který má nenulovou imaginární část, existuje kořen komplexně sdružený. Ze všech komplexních kořenů můžeme určit počet reálných pomocí Sturmovy metody (viz [11]).

Všechny Komplexní kořeny lze snadno vyčíst ze zápisu (3), kde je polynom rozdělen na jednoduše řešitelné lineární části. Pokud polynom není zapsaný tímto způsobem, musíme kořeny nalézt některou z přesných nebo přibližných metod výpočtu (viz kapitoly 4 a 5 ).

Existují i zvláštní případy kořenů polynomu, na které si při řešení musíme dát pozor, a sice kořeny násobné a shluky kořenů.

Násobný kořen je takový, který se v polynomu vyskytuje vícekrát. To znamená, že z jedné hodnoty, ve které je polynom nulový, můžeme sestrojit lineární faktor $x - x_0$, kterým můžeme polynom dělit beze zbytku vícekrát. Např. polynom

$$P(x) = x^3 - 3x^2 + 3x - 1 \tag{5}$$

můžeme rozložit na součin

$$P(x) = (x-1)(x-1)(x-1) \tag{6}$$

a vidíme, že má trojnásobný kořen $x = 1$.

Násobné kořeny mohou působit u některých numerických metod problémy. Derivace v násobném kořenu je rovna 0, v blízkém okolí se nule blíží. Proto v takových případech selhává například metoda Newtonova, která ve svém algoritmu

zpřesňuje odhad polohy kořene pomocí derivace v bodě. Z hodnoty derivace je také vidět, že polynom v okolí násobného kořene mění svou hodnotu velmi málo, tudíž při hledání kořene musíme dávat pozor i na přesnost výpočtu, pokud bychom používali metody, které s hodnotou polynomu pracují. Např. pro bod $x = 1,1$ má polynom (5) hodnotu $P(x) = 0,001$.

Shluky kořenů popisují situaci, kdy jsou kořeny sice různé, ale leží blízko sebe. Polynom, který takový shluk kořenů má, může být například polynom

$$P(x)=(x-1,1509)(x-1,14)(x-1,145)(x-1,155) \tag{7}$$

Shluk kořenů může metodám působit potíže stejné jako násobný kořen. Jiný problém může spočívat v tom, že jednotlivé iterace metod mohou střídavě konvergovat k prvnímu, druhému až n–tému kořenu. Pokud se tak stane, není možné nalézt ani jeden z kořenů. Tento problém se dá obejít dostatečně přesným počátečním odhadem.

# 3 POMOCNÉ OPERACE PŘI PRÁCI S POLYNOMY

## 3.1 HORNEROVO SCHÉMA

Hornerovo schéma je způsob, jakým zjišťovat hodnotu polynomu v daném bodě. Vedle tohoto použití má i další užitečná využití.

Mějme polynom (1). Pro využití Hornerova schématu jej zapíšeme ve tvaru (4). Počítání je v tomto tvaru jednodušší a rychlejší. Také jej můžeme přehledně zarovnat do tabulky 1.

*Tabulka 1: Znázornění Hornerova schématu*

| $x$ | $a_n$ | $a_{n-1}$ | $a_{n-2}$ | ... | $a_1$ | $a_0$ |
|---|---|---|---|---|---|---|
| | | $x*b_n$ | $x*b_{n-1}$ | ... | $x*b_2$ | $x*b_1$ |
| | $b_n=$ $a_n$ | $b_{n-1}=$ $a_{n-1}+x*b_n$ | $b_{n-2}=$ $a_{n-2}+x*b_{n-1}$ | ... | $b_1=$ $a_1+x*b_2$ | $b_0=$ $a_0+x*b_1$ |

Hodnota $b_0$ pak je hodnota $P(x)$.

Koeficienty $b_n$ až $b_1$ můžeme využít pro výpočet hodnoty derivace polynomu v bodě $x$. Následujícím způsobem vypočítáme pouze hodnotu $P'(x)$ ve stejném $x$, jako jsme počítali hodnotu $P(x)$, nikoliv derivaci obecně.

Z koeficientů $b_n$ až $b_1$ sestrojíme polynom

$$Q(x)=(\ldots((b_n)x+b_{n-1})x+\ldots+b_2)x+b_1 \tag{8}$$

a počítáme opět pomocí Hornerova schématu, jak je uvedeno v tabulce 2. Hodnota $c_1$ pak je hodnota $P'(x)$. Tento postup lze opakovat i pro vyšší derivace.

*Tabulka 2: Výpočet P'(x) pomocí Hornerova schématu*

| $x$ | $b_n$ | $b_{n-1}$ | $b_{n-2}$ | ... | $b_2$ | $b_1$ |
|---|---|---|---|---|---|---|
| | | $x*c_n$ | $x*c_{n-1}$ | ... | $x*c_3$ | $x*c_2$ |
| | $c_n=$ $b_n$ | $c_{n-1}=$ $b_{n-1}+x*c_n$ | $b_{n-2}=$ $b_{n-2}+x*c_{n-1}$ | ... | $c_2=$ $_{b2}+x*c_3$ | $c_1=$ $b_1+x*c_2$ |

Pomocí Hornerova schématu lze také jednoduše dělit polynom polynomem prvního stupně. Opět mějme polynom (4) a polynom prvního stupně

$$Q(x)=q_1 x+q_0 \tag{9}$$

Celý výpočet můžeme uspořádat do tabulky 3. Algoritmus výpočtu je podobný, ale na výsledek se nahlíží jinak a také se musí pracovat s více hodnotami.

*Tabulka 3: Dělení polynomu P(x) polynomem prvního stupně pomocí Hornerova schématu*

| $q_0$ | $a_n$ | $a_{n-1}$ | $a_{n-2}$ | ... | $a_1$ | $a_0$ |
|---|---|---|---|---|---|---|
| $q_1$ | | $q_0*b_n$ | $q_0*b_{n-1}$ | ... | $q_0*b_2$ | $q_0*b_1$ |
| | $b_n=$ $a_n/q_1$ | $b_{n-1}=$ $(a_{n-1}+q_0*c_n)/q_1$ | $b_{n-2}=$ $(a_{n-1}+q_0*b_{n-1})/q_1$ | ... | $b_1=$ $(a_1+q_0*b_2)/q_1$ | $b_0=$ $(a_0+q_0*b_1)/q_1$ |

Z posledního řádku jako výsledek vyčteme výsledný polynom

$$R(x)=b_n x^{n-1}+b_{n-1}x^{n-2}+b_{n-2}x^{n-3}+\ldots+b_2 x+b_1+\frac{b_0}{Q(x)} \tag{10}$$

### 3.2 ODSTRAŇOVÁNÍ NÁSOBNÝCH KOŘENŮ

Zajímavou vlastností polynomu je, že derivace polynomu $P'(x)$ má stejný kořen jako původní polynom $P(x)$, pokud tento kořen je v $P(x)$ kořenem násobným. Pokud $P(x)$ nemá násobný kořen, jsou všechny kořeny $P'(x)$ od kořenů $P(x)$ různé.

Z této skutečnosti vyplývá, že $P(x)$ i $P'(x)$ mají stejný lineární faktor $x\text{-}x_0$. Toho můžeme využít tak, že najdeme společného dělitele $P(x)$ a $P'(x)$ pomocí Euklidova algoritmu a tímto vydělíme $P(x)$. Detaily jsou uvedeny např. v [5].

Vlivem případného zaokrouhlování můžeme ztratit požadovanou přesnost následujících výpočtů, ale některým metodám tímto způsobem usnadníme konvergenci ke kořenům.

## 4 PŘESNÉ METODY PRO ŘEŠENÍ POLYNOMU

Tato kapitola se zabývá metodami pro přesné nalezení kořenů polynomu.

Přesné metody jsou známy pouze pro polynomy stupně menšího než 5. Je dokázáno, že pro polynomy vyšších stupňů obdobné metody neexistují.

### 4.1 PRO LINEÁRNÍ POLYNOM

Pro lineární polynom (polynom prvního stupně $P(x) = a_1x + a_0,\ a_1 \neq 0$) platí jednoduchý vztah

$$x_0 = -\frac{a_0}{a_1} \tag{11}$$

### 4.2 PRO KVADRATICKÝ POLYNOM

Pro řešení kvadratického polynomu (polynom druhého stupně $P(x) = a_2x^2 + a_1x + a_0,\ a_2 \neq 0$) se využívá diskriminantu.

$$x_0 = \frac{-a_1 - \sqrt{D}}{2a_2} \tag{12}$$

$$x_1 = \frac{-a_1 + \sqrt{D}}{2a_2} \tag{13}$$

kde

$$D = a_1^2 - 4\,a_2\,a_0 \tag{14}$$

### 4.3 PRO KUBICKÝ POLYNOM

Pro kubický polynom (polynom třetího stupně $P(x) = a_3x^3 + a_2x^2 + a_1x + a_0$, $a_3 \neq 0$) musíme před řešením zaručit, že $a_3 = 1$, tedy upravit jej na tvar (2). Pak můžeme kořeny najít pomocí následujících rovnic.

$$x_0 = -\frac{a_2}{3} + B \tag{15}$$

$$x_1 = -\frac{a_2}{3} - \frac{B}{2} + A * \frac{i\sqrt{3}}{2} \tag{16}$$

$$x_2 = -\frac{a_2}{3} - \frac{B}{2} - A * \frac{i\sqrt{3}}{2} \tag{17}$$

Kde

$$B = \sqrt[3]{q + \sqrt{p^3 + q^2}} + \sqrt[3]{q + \sqrt{p^3 - q^2}} \tag{18}$$

$$A = \sqrt[3]{q + \sqrt{p^3 + q^2}} - \sqrt[3]{q + \sqrt{p^3 - q^2}} \tag{19}$$

$$p = \frac{3\,a_1 - a_2^2}{9} \tag{20}$$

$$q = \frac{9\,a_1 a_2 - 27\,a_0 - 2\,a_2^3}{54} \tag{21}$$

Odvození těchto rovnic je uvedeno např. v [3].

### 4.4 PRO KVARTICKÝ POLYNOM

Pro kvartický polynom (také známý jako bikvadratický, polynom čtvrtého stupně $P(x) = a_4x^4 + a_3x^3 + a_2x^2 + a_1x + a_0$, $a_4 \neq 0$) musíme před řešením zaručit, že $a_4 = 1$, tedy upravit jej na tvar (2). Pak můžeme kořeny najít pomocí následujících rovnic.

$$x_0 = -\frac{a_3}{4} + \frac{R + D}{2} \tag{22}$$

$$x_1 = -\frac{a_3}{4} + \frac{R - D}{2} \tag{23}$$

$$x_2 = -\frac{a_3}{4} - \frac{R - E}{2} \tag{24}$$

$$x_0 = -\frac{a_3}{4} - \frac{R + E}{2} \tag{25}$$

Kde

$$R = \sqrt{\frac{1}{4}a_3^2 - a_2 + y_1} \tag{26}$$

$$D = \sqrt{\frac{3}{4}a_3^2 - R^2 - 2a_2 + \frac{1}{4R}\left(4a_3a_2 - 8a_1 - a_3^3\right)} \quad pro\ R \neq 0 \tag{27}$$

$$D = \sqrt{\frac{3}{4}a_3^2 - 2a_2 + 2\sqrt{y_1^2 - 4a_0}} \qquad pro\ R = 0$$

$$E = \sqrt{\frac{3}{4}a_3^2 - R^2 - 2a_2 - \frac{1}{4R}\left(4a_3a_2 - 8a_1 - a_3^3\right)} \quad pro\ R \neq 0 \tag{28}$$

$$E = \sqrt{\frac{3}{4}a_3^2 - 2a_2 - 2\sqrt{y_1^2 - 4a_0}} \qquad pro\ R = 0$$

přičemž y1 je reálný kořen pomocného kubického polynomu

$$y^3 - a_2y^2 + (a_3a_1 - 4a_0)y + (4a_2a_0 - a_1^2 - a_3^2a_0) = 0 \tag{29}$$

Odvození rovnic je uvedeno např. v [4] a [6].

## 5 PŘIBLIŽNÉ METODY PRO ŘEŠENÍ POLYNOMU

Tato kapitola se zabývá přibližnými metodami hledání kořenů polynomu.

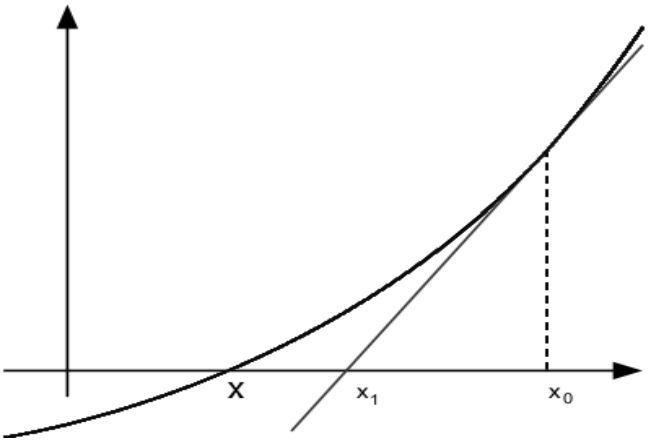
Polynomy stupně většího než 4 nelze řešit jinak než těmito metodami. Neudávají hodnotu kořenů přesně, ale přibližně.

Společný princip těchto metod spočívá v opakovaném přepočítávání odhadu podle nějakých vztahů, proto se jim říká iterační. Při tomto opakovaném výpočtu se snažíme, aby se tento odhad postupně měnil čím dál méně, tedy konvergoval k jedinému bodu, a sice kořenu.

### 5.1 NEWTONOVA METODA

Newtonova metoda je nejznámější metoda pro hledání kořenů polynomu, kterou lze použít pro hledání reálných i komplexních kořenů. Z této metody vychází hodně dalších metod.

Princip Newtonovy metody spočívá v přibližování se kořenu pomocí lineární aproximace polynomu. Mějme polynom (1) s kořenem $X$ a počáteční odhad kořenu $x_0$. Když k polynomu sestrojíme v bodě $x_0$ tečnu, pak pro její průsečík s osou x označený $x_1$ bude platit $|x_0-X| > |x_1-X|$. Je jasné, že pokud $x_0 = X$, pak i $x_1 = X$. Situaci popisuje obrázek 1.




*Obrázek 1: Ilustrace principu Newtonovy metody*

*Zdroj: [9], upravil Jiří Engliš*

Vztah pro Newtonovu metodu je

$$x_{i+1} = x_i - \frac{P(x_i)}{P'(x_i)} \tag{30}$$

kde $x_{i-1}$ je bližší aproximace $X$ a $x_i$ původní aproximace $X$. Rychlost konvergence je kvadratická.

Při vysokých hodnotách $P'(x)$ může být konvergence k $X$ pomalá, a proto se pro zrychlení občas využívá tzv. Dvojitá Newtonova metoda. Tato oproti klasické Newtonově metodě místo tečny pracuje se sečnou a má mírně upravený vztah

$$x_{i+1} = x_i - 2\frac{P(x_i)}{P'(x_i)} \tag{31}$$

Pro konvergenci Newtonovy metody existuje postačující podmínka. Můžeme ji vyjádřit tak, že pokud máme interval $I$, který obsahuje kořen $x_k$ a pro všechna $x$ z tohoto intervalu platí

$$\left|\frac{P(x)P''(x)}{(P'(x))^2}\right|<1 \tag{32}$$

pak pokud počáteční odhad $x_0$ spadá do tohoto intervalu, Newtonova metoda bude konvergovat ke kořenu $x_k$.

Pokud počáteční odhad $x_0$ do tohoto intervalu nespadá, konvergence jistá není. Problémy přináší jednak okolí lokálních maxim, tedy bodů, ve kterých je $P'(x) = 0$, které mohou způsobit „odskočení“ odhadu nebo zacyklení algoritmu, dále kořeny v inflexních bodech, které mohou způsobit zacyklení algoritmu. Pokud se těmto potížím vyhneme, může se také stát, že algoritmus konverguje k úplně jinému kořenu, než který je k původnímu odhadu nejblíže.

### 5.2 BAIRSTOWOVA METODA

Bairstowova metoda je další metoda založena na metodě Newtonově. Její princip spočívá v tom, že se snaží najít polynom druhého stupně, který je dělitelem polynomu $P(x)$. Z toho můžeme vidět, že zatímco pomocí metody Newtonovy hledáme kořeny po jednom, pomocí této metody je hledáme po dvou.

Na počátku máme polynom (1). Tento polynom můžeme vyjádřit jako

$$P(x)=(x^2+ux+v)Q(x)+c(z+u)+d \tag{33}$$

kde

$$Q(x)=b_{n-2}x^{n-2}+b_{n-3}x^{n-3}+\ldots+b_1x+b_0 \tag{34}$$

V této metodě hledáme kvadratický faktor $x^2 + ux + v$ polynomu $P(x)$. Pro něj platí, že $c = d = 0$. Tyto proměnné jsou funkcemi $u$ a $v$.

Postup je takový, že si zvolíme počáteční $u$ a $v$. Tyto můžeme zvolit jakékoliv. Pomocí těchto proměnných pak vypočítáme koeficienty polynomu $Q(x)$ podle následujících vztahů:

$$b_i = a_{i+2} - ub_{i+1} - vb_{i+2} \qquad i = n-2, n-3, \ldots, 0 \tag{35}$$

$$c = a_1 - ub_0 - vb_1 \tag{36}$$

$$d = a_0 - vb_0 \tag{37}$$

Přičemž koeficienty $b_n = b_{n-1} = 0$. Jakmile tyto hodnoty vypočítáme, použijeme dvourozměrnou Newtonovu metodu pro zpřesňování $u$ a $v$. Tuto můžeme zapsat pomocí matice

$$\begin{bmatrix} u_{i+1} \\ v_{i+1} \end{bmatrix} = \begin{bmatrix} u_i \\ v_i \end{bmatrix} - \begin{bmatrix} \frac{\delta c}{\delta u} & \frac{\delta d}{\delta u} \\ \frac{\delta c}{\delta v} & \frac{\delta d}{\delta v} \end{bmatrix}^{-1} \begin{bmatrix} c \\ d \end{bmatrix} \tag{38}$$

přičemž po úpravách dostaneme vztahy

$$u_{i+1} = u_i + \frac{c}{b_0} - d\frac{b_1}{b_0^2} \tag{39}$$

$$v_{i+1} = v_i + \frac{d}{b_0} \tag{40}$$

Celý proces počítání koeficientů opakujeme, dokud $c \neq 0$ nebo $d \neq 0$ nebo dokud se dostatečně nule nepřiblíží. Pokud se tak stane, vypočítáme kořeny kvadratického faktoru $x^2 + ux + v$, které jsou také kořeny polynomu $P(x)$. Po vypočítání těchto kořenů můžeme začít hledat stejnou metodou kořeny vzniklého polynomu $Q(x)$, pokud je jeho stupeň větší než 2. Pokud je stupeň menší, můžeme použít přímou metodu.

Rychlost konvergence Bairstowovy metody je kvadratická, ale při přibližování se k násobným kořenům je rychlost konvergence menší.

### 5.3 LAGUERROVA METODA

Laguerrova metoda je spolehlivá a jednoduše implementovatelná metoda hledání kořenů polynomu. Není však příliš používaná, jelikož její teorie není v dostatečné míře vysvětlena, zvláště pro komplexní kořeny.

Pro tento výpočet mějme polynom (3). Celý polynom logaritmujeme a dostaneme vztah

$$\ln|P(x)| = \ln|x - x_n| + \ln|x - x_{n-1}| + \ldots + \ln|x - x_0| \tag{41}$$

ze kterého vypočítáme první a druhou derivaci a položíme je rovné polynomům G(x) a H(x) dle následujících vztahů:

$$G(x) = \frac{d \ln|P(x)|}{d x} = \frac{P'(x)}{P(x)} = \frac{1}{x - x_n} + \frac{1}{x - x_{n-1}} + \ldots + \frac{1}{x - x_0} \tag{42}$$

$$H(x) = \frac{-d^2 \ln|P(x)|}{d x^2} = (G(x))^2 - \frac{P''(x)}{P(x)} = \frac{1}{(x - x_n)^2} + \ldots + \frac{1}{(x - x_0)^2} \tag{43}$$

Pro další výpočet pak zvolíme hodnotu $x_0$ a předpokládáme, že vzdálenost od jednoho kořene $(x - x_k)$ je rovna $a$ a vzdálenost ke všem ostatním kořenům je rovna $b$. Tím se vztahy (42) a (43) upraví na vztahy

$$G(x) = \frac{1}{a} + \frac{n-1}{b} \tag{44}$$

$$H(x) = \frac{1}{a^2} + \frac{n-1}{b^2} \tag{45}$$

ze kterých můžeme vyjádřit vzdálenost $a$

$$a = \frac{n}{G(x_0) \pm \sqrt{(n-1)(nH(x_0) - G(x_0)^2)}} \tag{46}$$

Znaménko ve jmenovateli vybereme takové, aby byla velikost jmenovatele největší. Nový odhad spočítáme jako $x_0 - a$. Pokud velikost $a$ neodpovídá požadované přesnosti, opakujeme výpočet pro (42), (43) a (46) pro novou hodnotu $x_0$. Jakmile kořen s požadovanou přesností nalezneme, snížíme pomocí něj stupeň polynomu a proces opakujeme.

Pomocí Laguerrovy metody můžeme najít jakýkoliv kořen, nezávisle na zvoleném počátečním $x_0$. Existují sice případy, kdy metoda konvergovat pro dané $x_0$ nebude, ty se však vyskytují zřídka. Rychlost konvergence je kubická.

### 5.4 LEHMER-SCHUROVA METODA

Lehmer-Schurova metoda je založena na skutečnosti, že můžeme zjistit, zda se od daného bodu v dané vzdálenosti nachází kořen daného polynomu.

Pro polynom *P(x)* můžeme zjistit, zda se v jednotkovém kruhu se středem v počátku Gaussovy komplexní roviny nachází jeho kořen. Obecně, pokud za *x* dosadíme dvojčlen *Rx + C*, dostaneme polynom

$$Q(x)=P(Rx+C) \tag{47}$$

Pak totéž, co zjistíme o poloze kořenu polynomu *Q(x)* v jednotkovém kruhu se středem v počátku Gaussovy komplexní roviny, je stejné, jako co bychom zjistili o kořenu polynomu *P(Rx + C)* v kruhu o poloměru *R* se středem v bodě *C*.

Pro výpočet polohy kořene mějme polynom (1). K němu definujeme doplňkový polynom

$$P_d(x)=x^n\left(P\left(\frac{1}{x^*}\right)\right)^* \tag{48}$$

kde hvězdičkou je označeno číslo komplexně sdružené. Tímto dostaneme

$$P_d(x)=a_0^*x^n+a_1^*x^{n-1}+\ldots+a_{n-1}^*x+a_n^* \tag{49}$$

Pomocí doplňkového polynomu spočítáme operátor T

$$T^0[P]=a_0^*P-a_nP_d \tag{50}$$

Tento operátor je také polynom, takže operátor T lze počítat rekurentně

$$T^j[P]=T[T^{j-1}[P]] \tag{51}$$

Tímto výpočtem můžeme najít nejmenší *l*, pro které platí

$$T^l[P](0)=0 \tag{52}$$

Výsledek výpočtu T je polynom o minimálně jeden stupeň nižší než původní polynom. Proto pro polynom n–tého stupně vyjde *l* maximálně *n*.

Pro jednotlivé výpočty T pak zkoušíme, jestli $T^k(0)<0$ pro $0<k<l$. Pokud takové *k* existuje, nachází se kořen v prohledávané oblasti. Pokud takové číslo neexistuje, kořen se v této oblasti nenachází.

Postup, jakým oblasti tímto způsobem prohledávat, je začít v počátku s kruhem o poloměru 1, tzn. podle vztahu (47) je $R = 1$ a $C = 0 + 0i$. Pokud se v tomto kruhu kořen nenachází, zvýšíme poloměr na dvojnásobek a opakujeme, dokud nenajdeme poloměr, ve kterém se kořen vyskytuje. Jakmile daný poloměr najdeme, začneme aproximovat kořen. To provádíme tím způsobem, že mezikruží mezi aktuálním poloměrem a poloměrem před poslední úpravou pokryjeme menšími kruhy, celkem osmi, s poloměrem 0,8R a se středy podle vztahu

$$C_{j+1}=C_j+\frac{3\mathrm{R}}{2\cos\left(\frac{\pi}{8}\right)}\mathrm{e}^{\frac{k\pi}{4}i} \qquad k=0,1,\ldots,7 \tag{53}$$

Pro těchto osm kruhů zkoušíme, do kterého kořen padne. Jakmile ho určíme, začneme zmenšovat poloměr na polovinu, dokud se v něm kořen vyskytuje. Poslední dva kroky opakujeme, dokud je poloměr větší než požadovaná přesnost.

Pomocí nalezeného kořene pak sestrojíme lineární faktor, kterým polynom vydělíme, a s tímto polynomem nižšího stupně pak metodu opakujeme do nalezení všech kořenů.

Tato metoda k danému kořenu vždy konverguje.

### *5.5 JENKINS-TRAUBŮV ALGORITMUS*

Jenkins-Traubův algoritmus je prověřený algoritmus pro hledání kořenů polynomu, často implementovaný v matematických programech (např. Mathematica).

Algoritmus Jenkins-Traubova algoritmu vychází opět z metody Newtonovy (30). V tomto vztahu nahrazuje derivaci *P'(x)* hodnotou vlastního polynomu *H(x)* sestaveného ve třech stupních.

První stupeň je tzv. proces bez posunů. Tento stupeň teoreticky není potřeba, ale v praxi je užitečný. Jeho smyslem je zvýraznit malé kořeny.

V tomto stupni vypočítáme *H(x)* pomocí těchto vztahů:

$$H^0(x) = P'(x) \tag{54}$$

$$H^{j+1}(x) = \frac{1}{x}\left[H^j(x) - \frac{H^j(0)}{P(0)}P(x)\right] \quad pro\ j = 0,1,\ldots,M-1 \tag{55}$$

Hodnota M se volí podle přesnosti aritmetiky, obvykle M = 5.

Druhý stupeň je tzv. proces s pevnými posuny. Jeho smysl je separovat kořeny se stejnou či skoro stejnou absolutní hodnotou.

V tomto stupni pokračujeme s výpočtem *H(x)*. Nejdříve zvolíme číslo $\beta$, takové, že

$$\beta \leqslant \min_{i=1,\ldots,k} |z_i| \tag{56}$$

kde $z_i$ jsou předchozí nalezené kořeny. Pokud žádné nejsou, zvolíme $\beta = 0$. K $\beta$ zvolíme číslo $|s| = \beta$. Dále počítáme

$$H^{j+1}(x) = \frac{1}{x-s}\left[H^j(x) - \frac{H^j(s)}{P(s)}P(x)\right] \quad pro\ j = M, M+1,\ldots,L-1 \tag{57}$$

přičemž číslo L opět volíme.

Třetí stupeň je tzv. proces s proměnnými posuny. V tomto stupni nacházíme aproximaci kořene.

Začneme vypočítáním odhadu

$$x^L = s - \frac{P(s)}{H^L(s)} \tag{58}$$

Kde $H^L(S)$ je zapsán ve tvaru (2), a pokračujeme vztahy

$$H^{j+1}(x)=\frac{1}{x-x^j}\left[H^j(x)-\frac{H^j(z^j)}{P(z^j)}P(x)\right] \tag{59}$$

$$x^{j+1}=x^j-\frac{P(x^j)}{H^{j+1}(x^j)} \qquad kde\ j=L, L+1,\ldots$$

dokud nedosáhneme požadované přesnosti.

Po nalezení kořene jím snížíme stupeň polynomu a metodu opakujeme, dokud nenalezneme všechny kořeny.

Tento algoritmus konverguje pro jakékoliv počáteční $x_0$ a to Rychleji než metoda Newtonova.

### *5.6 WEIERSTRASSŮV ALGORITMUS*

Weierstrassův algoritmus (také udáván jako Durand-Kernerova metoda) je jednoduchý a rychlý algoritmus, který na rozdíl od předchozích algoritmů nepočítá kořeny postupně, ale najednou.

Tento algoritmus využívá toho, že polynom (2) můžeme vyjádřit vztahem (3). Pomocí tohoto vyjádření pak také můžeme vyjádřit kořen polynomu pro jakoukoliv hodnotu $x$ jako

$$x_k=x-\frac{P(x)}{\prod_{i=0,i\neq k}^{n}(x-x_i)} \tag{60}$$

Z tohoto vztahu lze odvodit iterativní metoda, která postupně počítá rovnice:

$$x_k^{(j+1)}=x_k^{(j)}-\frac{P(x_k^{(j)})}{\prod_{i=0,i\neq k}^{n}(x_k^{(j)}-x_i)} \qquad k=1,2,\ldots,n \tag{61}$$

Kde za $x_i$ dosazujeme poslední vypočítané odhady ostatních kořenů. Zvláštností metody je, že zpočátku se hodnoty $x_i$ mění zdánlivě náhodně, ale po několika iteracích se začínají ustalovat a konvergovat ke kořenům *P(x)*.

Jako počáteční hodnoty je možné zvolit jakákoliv čísla. Existuje však několik podmínek:

- Alespoň jedno musí být komplexní. Tato podmínka neplatí, pokud jsou všechny kořeny polynomu reálné. Pokud je však nějaký kořen komplexní a zároveň jsou všechny odhady i koeficienty polynomu *P(x)* reálné, pak není možné získat v iteracích imaginární jednotku.
- Žádné dva odhady se sobě nesmí rovnat. Pokud by se nějaké dva odhady rovnaly, vznikla by ve jmenovateli nula. Doporučení je, aby také nebyly hodnotou blízko sebe.

Rozumné je vybrat hodnoty na jednotkové kružnici v komplexní rovině se stejnými rozestupy, případně pro polynomy vyšších stupňů také na kružnicích vyšších poloměrů.

Tato metoda konverguje ke kořenům kvadraticky.

Problémy metodě mohou působit násobné kořeny a shluky kořenů, které mohou způsobit dělení nulou.

### *5.7 GRAEFFEOVA METODA*

Graeffeova metoda je další metoda, která dokáže najednou najít všechny kořeny zadaného polynomu.

Pro výpočet kořenů touto metodou mějme polynom (3). K němu můžeme sestrojit opačný polynom

$$P(-x)=(-1)^n(x+x_n)(x+x_{n-1})\ldots(x+x_0) \tag{62}$$

A po jejich vzájemném vynásobení dostáváme

$$P(x)P(-x)=(-1)^n(x^2-x_n^2)(x^2-x_{n-1}^2)\ldots(x^2-x_0^2) \tag{63}$$

Násobení dále opakujeme v-krát podle požadované přesnosti. Ve výsledku násobení provedeme substituci

$$y = x^{2v} \tag{64}$$

a dostaneme

$$Q(y) = y^n + b_1 y^n - 1 + \ldots + b_{n-1} y + b_n \tag{65}$$

Kde koeficienty $b_i$ splňují Vietovy vztahy. V důsledku opakovaného násobení jsou hodnoty prvních sčítanců Vietových vztahů mnohem větší než ostatní. Proto si můžeme dovolit zaokrouhlení

$$\begin{aligned} b_1 &= -y_1 \\ b_2 &= y_1 y_2 \\ b_n &= (-1)^n y_1 y_2 \ldots y_n \end{aligned} \tag{66}$$

Z těchto můžeme postupně vyjádřit kořeny *Q(y)*

$$\begin{aligned} y_1 &= -b_1 \\ y_2 &= -\frac{b_2}{b_1} \\ y_n &= -\frac{b_n}{b_{n-1}} \end{aligned} \tag{67}$$

a z těchto odvozením z (64) můžeme dopočítat kořeny původního polynomu

$$x_i = \sqrt[2v]{y_i} \tag{68}$$

Touto metodou je možné najít všechny kořeny polynomu. Při implementaci v programu je však nutné dát pozor na to, že hodnoty, kterých nabývají $y_i$, potažmo $b_i$, mohou být tak veliké, že je obyčejné datové typy používané pro výpočty s pohyblivou řádkovou čárkou nemusejí obsáhnout.

### 5.8 DALŠÍ METODY

Existuje mnoho dalších metod, které zde neuvádím, protože jsou neefektivní nebo protože jsou to pouze variace na některou z metod již uvedených, popřípadě jejich kombinace. Příklady takových metod jsou např. metoda půlení intervalů, metoda sečen, Müllerova metoda, Halleyova metoda a další.

## 6 POROVNÁNÍ PŘIBLIŽNÝCH METOD HLEDÁNÍ KOŘENŮ

Ve svém programu jsem implementoval a vyzkoušel Weierstrassův algoritmus, Bairstowovu metodu, Laguerreovu metodu a Newtonovu metodu, pro níž jsem počáteční odhad počítal pomocí Lehmer-Schurova algoritmu. Výpočty jsem zkoušel s přesností na 3 a 8 desetinných míst pro celkem 10 polynomů, které jsou vypsány níže:

1. $(x+1)(x+2)(x-1)^2$
2. $(x-0,4+0,7i)(x+100)(x+678+312i)(x-500+600i)$
3. $(x-0,4-0,8i)(x+1,6i)(x+4-i)(x-5+i)$
4. $(x+1)(x-2,5)(x+10)(x+2)^2(x-1)^2$
5. $(x-0,4+0,7i)(x+100)(x+678+312i)(x-500+600i)(x-0,5-4,1i)$
   $*(x+8000+8000i)(x-5+5i)$
6. $(x-0,4-0,8i)(x+1,6i)(x+4-i)(x-5+i)(x-8i)(x+8)(x+5-7i)$
7. $(x-5)(x+1)(x-2,5)(x+10)(x+2)^2(x-1)^2(x+0,2)^2$
8. $(x-0.4+0.7i)(x+0.8+0.6i)(x-0.5-4.1i)(x-5+5i)(x+100)(x+678+312i)$
   $*(x-500+600i)(x+5500i)(x+7000+i)(x+8000+8000i)$
9. $(x-0.4+0.7i)(x+1+5i)(x+6+3,1i)(x-0,5-4i)(x+10)(x+6i)(x+7-i)(x-5-5i)$
   $*(x-5+6i)(x+8+8i)$
10. $(x-20+10i)^{32}$

Tímto výběrem jsem se pokusil vybrat takové typy polynomů, které vhodně reprezentují situace, které bude muset program řešit.

Údaje o výpočtech jsem pro každou metodu urovnal do tabulky, kde sleduji potřebný počet iterací[1], celkový čas hledání[2], zda výpočet splňuje požadovanou přesnost a nakonec největší odchylku odhadu od odpovídajícího kořene (tzn. $max \mid x - x_k \mid$)[3].

Výsledky Bairstowovovy metody zde nerozebírám, jelikož se kvůli nevhodné implementaci při každé iteraci zvyšuje čas potřebný pro zjištění nových odhadů (roli v tom hrála s největší pravděpodobností rostoucí počet desetinných míst při dělení). Z tohoto důvodu i řešení polynomu č. 1. s přesností na 3 desetinná místa trvalo příliš dlouho než aby bylo toto řešení zajímavé. Je však nutno podotknout, že v jednotlivých iteracích při pouštění programu "krok po kroku" docházelo ke konvergenci.

Metodu Newtonovu + Lehmer-Schurovu popisuje tabulka 4. Při této metodě probíhá výpočet tak, že se metoda Lehmer-Schurova ke kořenu přiblíží s přesností na tři desetinná místa a tento odhad pak předá metodě Newtonově jako odhad prvotní. Tato úvodní přesnost byla zvolena zkoušením, kdy pro přesnosti nižší, zvláště kvůli nepřesnostem při snižování stupně, vedl odhad ke konvergenci k úplně jinému kořenu. Pokud Metoda Newtonova nekonverguje (nenajde kořen do 150 iterací), počítá se odhad metodou Lehmer-Schurovou znovu s přesností o další dvě desetinná místa vyšší. Po nalezení kořene se sníží stupeň polynomu. V polynomu se sníženým stupněm pak metoda Lehmer-Schurova hledá další kořen. Metoda Newtonova kvůli přesnosti vždy pracuje s původním polynomem. Jako iteraci Lehmer-Schurovy metody považuji změnu $C$, jak je udávaná ve vztahu (53).

Je patrné, že tato metoda je vhodná spíše pro polynomy nižšího stupně. Z poměru celkového počtu iterací a celkového času vyplývá, že se čas nezvedá lineárně se stupněm polynomu, ale se složitostí mnohem vyšší. Domnívám se, že toto zapříčiňuje opakované počítání operátoru $T^4$, protože pro veliké stupně polynomu vznikají v koeficientech veliké hodnoty. Proto je právě výhodné nepočítat pomocí této metody s velikou přesností, pokud to není nutné. Dále si můžeme všimnout z počtu iterací

1 Udává průměrný počet pro jeden průchod metodou, tedy pro nalezení tolika kořenů, kolik jich danou metodou můžeme nalézt

2 Udává celkovou dobu hledání kořenů na počítači s jednojádrovým procesorem Intel s frekvencí 1,7 GHz, 2 GB RAM, OS MS Windows XP professional

3 Program automaticky vynechává číslice na místech, které jsou mimo požadovanou přesnost. Pro účel sběru těchto dat byl upraven tak, aby vypisoval výsledky s třemi desetinnými místy navíc.

4 Vztahy (50) a (51)

Newtonovy metody, že při výpočtu polynomů s násobnými kořeny konverguje pomaleji. Zajímavé také je, že celková doba výpočtu je pro polynomy s násobnými kořeny kratší než pro jiné polynomy stejného stupně, přestože jsem pro tuto možnost výpočet neoptimalizoval.

Metodu Laguerreovu popisuje tabulka 5. Při této metodě probíhá výpočet tak, že se kořeny polynomu hledají postupně. Po nalezení kořenu je jím stupeň polynomu snížen a další hledání se provádí na tomto polynomu.

Je vidět, že i při snižování stupně polynomu si metoda zachovává svou přesnost a je velmi rychlá. Rychlost konvergence mírně snižují násobné kořeny. Velmi mne také překvapil fakt, že pro polynom č. 8. s přesností na 8 desetinných míst našel kořen přesně (dokud jsem neopravil podmínku konce iterací, docházelo v algoritmu[5] k dělení nulou).

Výsledky pro metodu Weierstrassovu popisuje tabulka 6. Je vidět, že i tato metoda je rychlá, pokud však polynom nemá násobné kořeny. Při takové situaci nastává problém dělení nulou. V programu jsem tuto situaci nijak zvlášť neošetřil. Předpokládám totiž, že počítané hodnoty jsou pouze aproximace a při použití Javovské třídy BigDecimal místo datového typu double je zaručena přesnost výpočtu a nějaký minimální odstup. Přesto se nepodařilo vypočítat kořeny polynomu 10. právě kvůli vznikajícím malým číslům ve jmenovateli.

5 Vztah (42)

*Tabulka 4: Výsledky pro Newtonovu + Lehmer-Schurovu metodu*

| Číslo Polynomu | Přesnost ($10^{-n}$) | Počet iterací[6] | Celková doba (s) | Odpovídá Přesnosti | max $\lvert x-x_k \rvert$ |
|---|---|---|---|---|---|
| 1. | 3 | 6,5<br>6,5 | 0,19 | Ano | $(0,78+1,41i)*10^{-4}$ |
| | 8 | 6,5<br>23 | 0,26 | Ano | $(0,59+1,08i)*10^{-9}$ |
| 2. | 3 | 13,5<br>1,5 | 0,36 | Ano | $(0+3i)*10^{-6}$ |
| | 8 | 13,5<br>3 | 0,38 | Ano | $(1+i)*10^{-11}$ |
| 3. | 3 | 10<br>2 | 0,39 | Ano | $(1+i)*10^{-6}$ |
| | 8 | 10<br>3 | 0,41 | Ano | $(1+i)*10^{-11}$ |
| 4. | 3 | 8<br>5,4 | 10,92 | Ano | $(0,94+2,02i)*10^{-4}$ |
| | 8 | 6<br>19,2 | 10,64 | Ano | $(0,72+1,54i)*10^{-9}$ |
| 5. | 3 | 15,8<br>1,8 | 63,88 | Ano | $8*10^{-6}$ |
| | 8 | 15,8<br>3,2 | 63,16 | Ano | $(1+i)*10^{-11}$ |
| 6. | 3 | 10,2<br>2,2 | 77,05 | Ano | $1*10^{-6}$ |
| | 8 | 10,2<br>3,2 | 77,09 | Ano | $(1+i)*10^{-11}$ |
| 7. | 3 | 7,13<br>5,75 | 235,77 | Ano | $(0,94+7,97i)*10^{-4}$ |
| | 8 | 6,25<br>18,5 | 168,66 | Ano | $(0,7+1,56i)*10^{-9}$ |
| 8. | 3 | - | > 900 | - | - |
| | 8 | - | > 900 | - | - |
| 9. | 3 | - | > 900 | - | - |
| | 8 | - | > 900 | - | - |
| 10. | 3 | - | > 900 | - | - |
| | 8 | - | > 900 | - | - |

6 První číslo je počet iterací Lehmer-Schurovy metody, druhé číslo je počet iterací Newtonovy metody.

*Tabulka 5: Výsledky pro Laguerreovu metodu*

| Číslo Polynomu | Přesnost ($10^{-n}$) | Počet iterací | Celková doba (s) | Odpovídá přesnosti | Max $\lvert x-x_k \rvert$ |
|---|---|---|---|---|---|
| 1. | 3 | 6,5 | <0.001 | Ano | $1,3*10^{-5}$ |
| | 8 | 11 | <0.001 | Ano | $3,5*10^{-10}$ |
| 2. | 3 | 2,5 | 0,02 | Ano | $(1+i)*10^{-6}$ |
| | 8 | 3,5 | 0,02 | Ano | $(1+i)*10^{-11}$ |
| 3. | 3 | 3,5 | 0,02 | Ano | $(1+i)*10^{-6}$ |
| | 8 | 4 | 0,03 | Ano | $(1+i)*10^{-11}$ |
| 4. | 3 | 6 | 0,16 | Ano | $2,2*10^{-5}$ |
| | 8 | 10 | 0,17 | Ano | $2,4*10^{-10}$ |
| 5. | 3 | 3,4 | 0,08 | Ano | $(1+i)*10^{-6}$ |
| | 8 | 3,8 | 0,17 | Ano | $(1+i)*10^{-11}$ |
| 6. | 3 | 4,4 | 0,09 | Ano | $(1+i)*10^{-6}$ |
| | 8 | 4,8 | 0,13 | Ano | $(1+i)*10^{-11}$ |
| 7. | 3 | 5,38 | 0,13 | Ano | $3,5*10^{-5}$ |
| | 8 | 9,5 | 0,41 | Ano | $2,8*10^{-10}$ |
| 8. | 3 | 3,5 | 0,2 | Ano | $(1+i)*10^{-6}$ |
| | 8 | 4,13 | 0,41 | Ano | $(1+i)*10^{-11}$ |
| 9. | 3 | 4,5 | 0,23 | Ano | $(1+i)*10^{-6}$ |
| | 8 | 5,13 | 0,69 | Ano | $(1+i)*10^{-11}$ |
| 10. | 3 | 1 | 0,49 | Ano | 0 |
| | 8 | 1 | 0,52 | Ano | 0 |

*Tabulka 6: Výsledky pro Weierstrassovu metodu*

| Číslo Polynomu | Přesnost ($10^{-n}$) | Počet iterací | Celková doba (s) | Odpovídá přesnosti | Max $\|x-x_k\|$ |
|---|---|---|---|---|---|
| 1. | 3 | 6 | 0,35 | Ano | $1*10^{-6}$ |
| | 8 | 7 | 0,16 | Ano | $1*10^{-11}$ |
| 2. | 3 | 8 | 0,06 | Ano | $(1+i)*10^{-6}$ |
| | 8 | 9 | 0,06 | Ano | $(1+i)*10^{-11}$ |
| 3. | 3 | 5 | 0,04 | Ano | $(1+i)*10^{-6}$ |
| | 8 | 6 | 0,04 | Ano | $(1+i)*10^{-11}$ |
| 4. | 3 | 19 | 0,41 | Ano | $3,5*10^{-5}$ |
| | 8 | 31 | 1,69 | Ano | $3,4*10^{-10}$ |
| 5. | 3 | 15 | 0,37 | Ano | $(1+i)*10^{-6}$ |
| | 8 | 16 | 0,21 | Ano | $(1+i)*10^{-11}$ |
| 6. | 3 | 10 | 0,11 | Ano | $(1+i)*10^{-6}$ |
| | 8 | 11 | 0,11 | Ano | $(1+i)*10^{-11}$ |
| 7. | 3 | 14 | 0,8 | Ano | $(2,3+6,1i)*10^{-5}$ |
| | 8 | 26 | 3,1 | Ano | $(2,2+5,9i)*10^{-11}$ |
| 8. | 3 | 15 | 0,47 | Ano | $(1+i)*10^{-6}$ |
| | 8 | 15 | 0,53 | Ano | $(1+i)*10^{-11}$ |
| 9. | 3 | 15 | 0,4 | Ano | $(1+i)*10^{-6}$ |
| | 8 | 16 | 0,42 | Ano | $(1+i)*10^{-11}$ |
| 10. | 3 | - | - | - | - |
| | 8 | - | - | - | - |

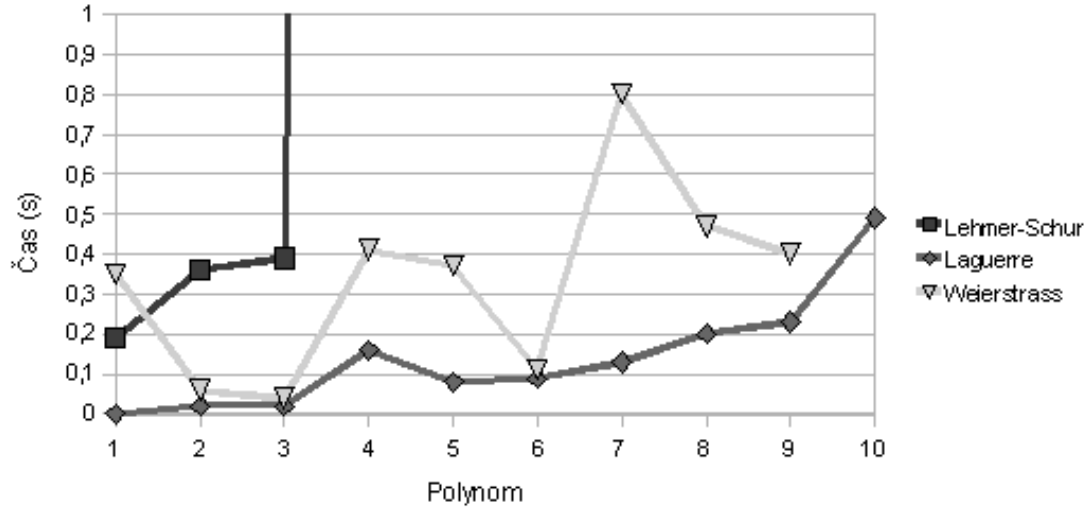
Pokud porovnám výsledky těchto metod, je vidět, že všechny metody lze naprogramovat tak, aby zaručily požadovanou přesnost výpočtu kořenu, liší se pouze počtem iterací, dobou výpočtu a tím, jak zvládají "extrémní" polynomy.

Z pohledu doby výpočtu si nejhůře vede kombinace metod Lehmer-Schurova s Newtonovou. Přestože se jedná o algoritmus, který celkově nevykazoval žádné problémy při konvergenci, pro polynomy vyššího stupně trvají výpočty příliš dlouhou dobu.

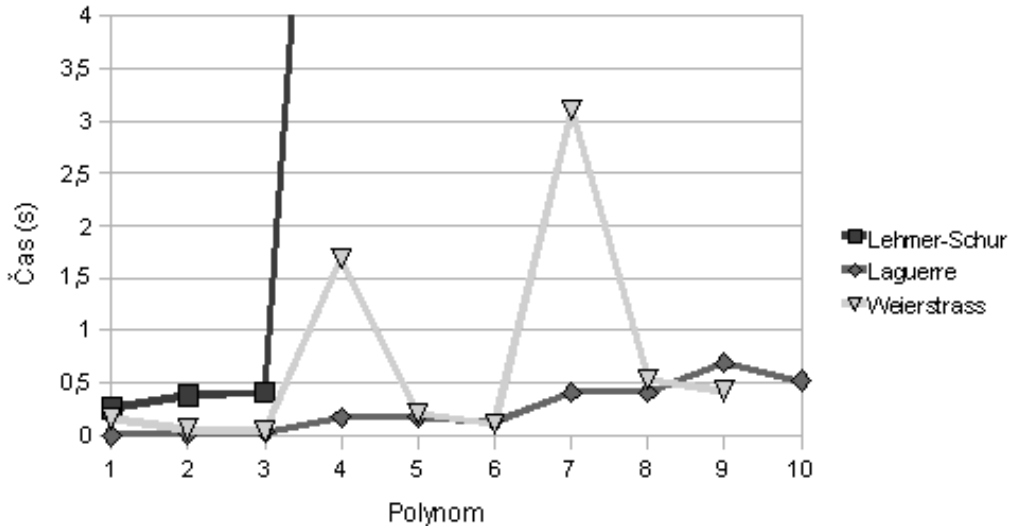
Mezi metodami Laguerrovou a Weierstrassovou není veliký rozdíl. Hlavní odlišnost, které jsem si všimnul, je, že Laguerreova metoda se lépe vypořádává s násobnými kořeny a čas výpočtu udržuje stále přibližně stejný jako pro ostatní poly-

nomy stejného stupně, na rozdíl od metody Weierstrassovy, u které se v takových případech doba výpočtu od průměru liší výrazněji nebo celkově selhává.

Grafické porovnání počtu iterací a rychlosti výpočtu můžeme vidět na obrázcích 2 a 3.




*Obrázek 2: Grafické porovnání doby výpočtů jednotlivých metod pro přesnost na 3 desetinná místa*




*Obrázek 3: Grafické porovnání doby výpočtů jednotlivých metod pro přesnost na 8 desetinných míst*

Podstatná je však otázka, nakolik jsou výsledky ovlivněny výše zmíněným užitím třídy BigDecimal pro výpočty namísto datového typu double. Tato třída je navržena tak, aby umožňovala oproti typu double vyšší přesnost při práci s desetinnými čísly. Některé její metody však nejsou dostatečně dobře implementovány a proto se na mnoha místech (zvláště při dělení) stejně musí používat typ double. Přesnost výpočtu v této třídě má také dopad na dobu výpočtu. Obvykle je mezi přesností a dobou výpočtu přímá uměra. Je tedy možné, že veliké zpomalení metody Bairstowovy a Lehmer-Schurovy by bylo zmírněno použitím méně přesných datových typů.

Vzhledem k této skutečnosti a výsledkům je pravděpodobné, že implementace pomocí třídy BigDecimal je nevhodná.

# 7 ZÁVĚR

V programu jsem implementoval a vyzkoušel některé algoritmy, o kterých v této práci teoreticky pojednávám.

Z mého porovnání vyplývá, že ze zkoušených metod (Lehmer-Schurova + Newtonova, Weierstrassova, Laguerreova a Bairstowova) je nejvhodnější pro implementaci metoda Laguerreova, protože pomocí ní můžeme najít kořeny polynomu s požadovanou přesností, minimálním nebezpečím selhání a příznivou rychlostí výpočtu.

V budoucnu bych rád do porovnání zahrnul také metodu Jenkins-Traubovu, popřípadě i Graeffeovu. Také bych rád prozkoumal otázku rozdílů implementací metod s pomocí třídy BigDecimal a datového typu double.

## LITERATURA


[1] ŠÍPEK, Jan, ZÍTKO, Jan. Algoritmy na výpočet kořenů polynomu. *Pokroky matematiky, fyziky a astronomie*. 2001, roč. 46, č. 1, s. 33-42.

[2] HOROVÁ, Ivana, ZELINKA, Jiří. *Numerické metody*. 2. rozš. vyd. [s.l.]: [s.n.], 2004. 294 s. ISBN 80-210-3317-7.

[3] WEISSTEIN, Eric W.. *MathWorld: Cubic formula* [online]. c1999-2009 [cit. 2009-04-22]. Dostupný z WWW: <http://mathworld.wolfram.com/CubicFormula.html>.

[4] WEISSTEIN, Eric W.. *MathWorld: Quartic Equation* [online]. c1999-2009 [cit. 2009-04-22]. Dostupný z WWW: <http://mathworld.wolfram.com/QuarticEquation.html>.

[5] BOND, C. *A robust strategy for finding all real and complex roots of real polynomials* [online]. 2004 [cit. 2009-04-22]. Dostupný z WWW: <http://www.crbond.com/papers/bproots.pdf>.

[6] *Roots of poylnomials* [online]. 2008 [cit. 2009-04-22]. Dostupný z WWW: <http://www.cs.iastate.edu/~cs577/handouts/polyroots.pdf>.

[7] *Horner scheme* [online]. 2001-, 31 March 2009 [cit. 2009-04-22]. Dostupný z WWW: <http://en.wikipedia.org/wiki/Horner_scheme>.

[8] CHOW, S.-Sum. *Example of horner's scheme* [online]. 2000 [cit. 2009-04-22]. Dostupný z WWW: <http://www.math.byu.edu/~schow/work/horner.htm>.

[9] *Newton's method* [online]. 2002, 16 April 2009 [cit. 2009-04-22]. Dostupný z WWW: <http://en.wikipedia.org/wiki/Newton_method>.

[10] *Bairstow's method* [online]. 2006, 11 January 2009 [cit. 2009-04-22]. Dostupný z WWW: <http://en.wikipedia.org/wiki/Bairstow%27s_method>.

[11] *Věda a technika: Počet reálných kořenů polynomu* [online]. 2007 [cit. 2009-04-27]. Dostupný z WWW: <http://veda-technika.blogspot.com/2007/10/pocet-realnych-korenu-polynomu.html>.

[12] *Jenkins-Traub algorithm* [online]. 2007, 29 July 2008 [cit. 2009-04-28]. Dostupný z WWW: <http://en.wikipedia.org/wiki/Jenkins-Traub_algorithm>.

[13] BINNER, David. *Math~Blog: Polynomial Root-finding with the Jenkins-Traub Algorithm* [online]. 2008 [cit. 2009-04-28]. Dostupný z WWW: <http://math-blog.com/2008/03/06/polynomial-root-finding-with-the-jenkins-traub-algorithm>.

[14] *Durand-Kerner method* [online]. 2006, 20 February 2009 [cit. 2009-04-28]. Dostupný z WWW: <http://en.wikipedia.org/wiki/Durand-Kerner_method>.

[15] *Laguerre's method* [online]. 2004, 9 June 2008 [cit. 2009-04-28]. Dostupný z WWW: <http://en.wikipedia.org/wiki/Laguerre%27s_method>.

[16] WEISSTEIN, Eric W.. *MathWorld: Laguerre's method* [online]. c1999-2009 [cit. 2009-04-22]. Dostupný z WWW: <http://mathworld.wolfram.com/LaguerresMethod.html>.

[17] *Graeffe's method* [online]. 2006, 3 March 2009 [cit. 2009-04-28]. Dostupný z WWW: <http://en.wikipedia.org/wiki/Graeffe's_method>.

[18] WEISSTEIN, Eric W.. *MathWorld: Graeffe's method* [online]. c1999-2009 [cit. 2009-04-22]. Dostupný z WWW: <http://mathworld.wolfram.com/GraeffesMethod.html>.

[19] SMITH, David A.. *Library of math: Newton's method* [online]. c2009 [cit. 2009-05-05]. Dostupný z WWW: <http://www.libraryofmath.com/the-newton-method.html>.

# PŘÍLOHA A: PROGRAMÁTORSKÁ DOKUMENTACE PROGRAMU

Toto je programátorská dokumentace k programu PolyŘešitel, který jsem naprogramoval k bakalářské práci Výpočet kořenů polynomu n–tého stupně. Program je určen pro hledání kořenů polynomu, který mu je zadán. Jde o GUI aplikaci napsanou v jazyce Java.

## TŘÍDY

Zdrojový kód je rozdělen do jedenácti tříd (každou v odpovídajícím *.java souboru) s tímto významem:

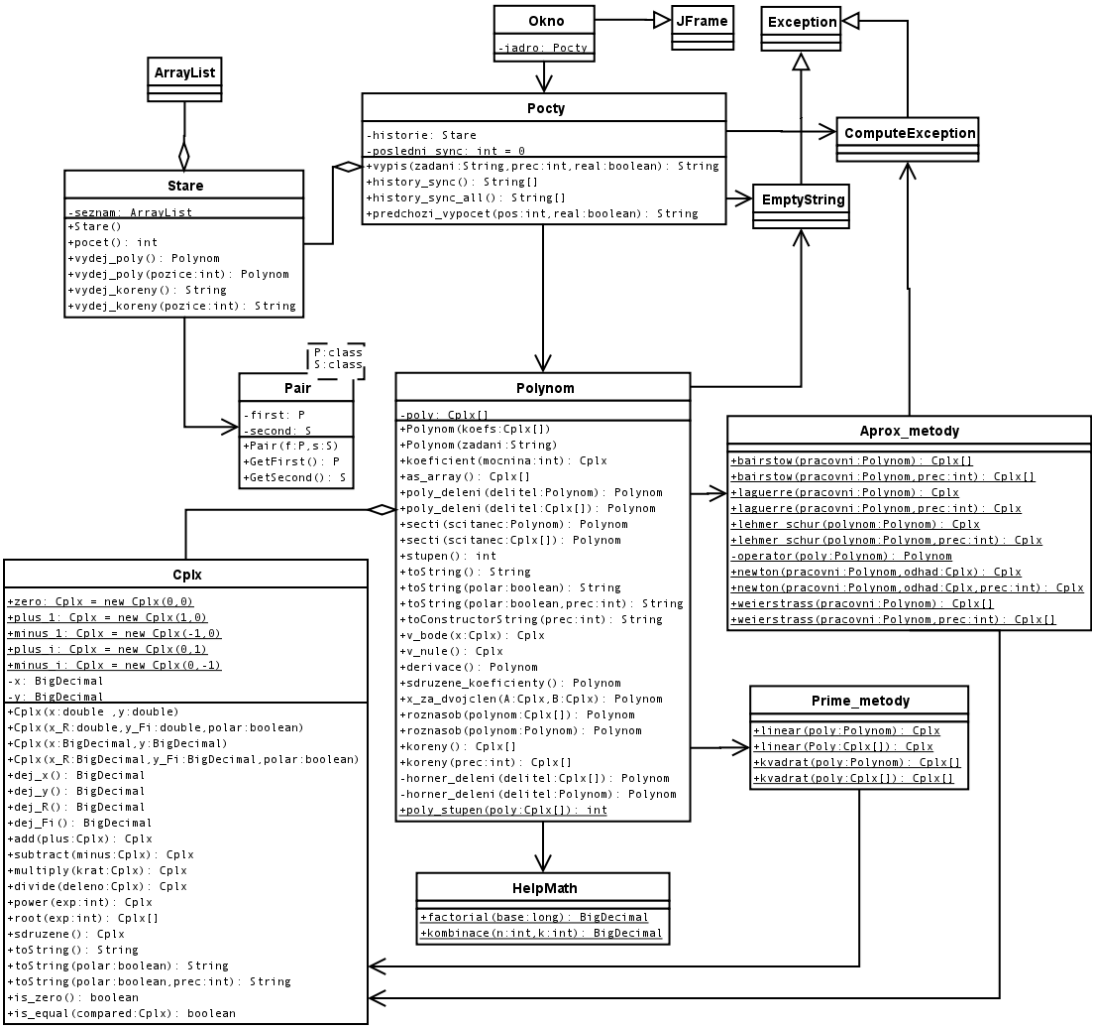
1. Okno popisuje chování hlavního okna programu.
2. Pair je obecná třída pro uchovávání dvojice datových typů
3. Pocty je jádro programu, které se stará o propojení ovládání, výpočtů a historie
4. Stare je třída pro uchovávání předchozích výpočtů
5. Cplx je třída pro komplexní čísla
6. Polynom je třída polynomů
7. Aprox_metody je třída pro abstraktní metody implementující aproximační algoritmy hledání kořenů polynomů
8. Prime_metody je třída pro abstraktní metody implementující přesné výpočty kořenů polynomů prvního a druhého stupně
9. HelpMath je třída implementující statické pomocné matematické metody, které nejsou zahrnuty ve třídě Math
10. ComputeException je třída pro výjimku, která nastane, když aproximační algoritmus nekonverguje
11. EmptyString je třída pro výjimku, která nastane, když se programu řekne, aby spočítal nezadaný polynom

Tyto třídy jsou pro přehlednost rozděleny do čtyř balíčků:

1. bc pro třídy 1 až 4

2. number pro třídy 5 a 6
3. metody pro třídy 7 a 8
4. helpers pro třídy 9 až 11

Vztahy mezi třídami popisuje diagram na obrázku 4.




*Obrázek 4: Třídní diagram programu*

## POPIS ATRIBUTŮ

Třída Okno:

- Atributy této třídy jsou prvka hlavního okna. Tyto atributy byly automaticky vygenerovány při návrhu v prostředí NetBeans.

Třída Pair:

- Atributy first a second jsou prvky datových typů určených šablonou třídy, které tvoří dvojici.

Třída Pocty:

- Atribut historie je statická instance třídy Stare, který slouží k uchovávání předchozích výpočtů.
- Atribut posledni_sync je statická celočíselná hodnota používaná při synchronizaci, která udává, kolik prvků bylo v atributu historie při poslední synchronizaci historie s vnějšími třídami.

Třída Stare:

- Atribut seznam je instance třídy ArrayList, která uchovává instance Třídy Pair. V tomto atributu jsou uchovávány samostatné informace o předchozích výpočtech.

Třída Cplx:

- Atributy x a y jsou instance třídy BigDecimal, které reprezentují reálnou, resp. imaginární část komplexního čísla.
- Atributy plus_1, plus_i, minus_1, minus_i a zero jsou statické a jedná se o instance užitečných komplexních čísel (1, i, -1, -i, resp. 0).

Třída Polynom:

- Atribut poly je pole instancí třídy Cplx, které reprezentuje koeficienty polynomu. Tyto koeficienty jsou udávány tak, že instance na n–tém místě udává koeficient u n–té mocniny (např. poly[4] je koeficient pro $x^4$).
- Atribut stupen je celočíselná hodnota udávající stupeň polynomu. Je udávána pro urychlení výpočtu, jelikož kontrolovat stupeň z koeficientů je zdlouhavé a zbytečné, jelikož se tyto koeficienty nemění (všechny výpočty se projevují do nové instance třídy Polynom).

Ostatní třídy atributy nemají, kromě tříd CopmuteException a EmptyString, které mají atributy zděděné ze třídy Exception.

### POPIS METOD

Třída Okno:

- Metody této třídy jsou určené pro inicializaci okna, běh okna a události ovládacích prvků. Byly automaticky vygenerovány při návrhu v prostředí NetBeans.

Třída Pair:

- Má konstruktor Pair, jehož parametry jsou prvky, které mají být vloženy.
- Další metody (GetFirst, GetSecond) tyto prvky zpřístupňují pro čtení.

Třída pocty: Obsahuje pouze statické metody.

- Metoda vypis slouží k výpočtu kořenů polynomu. Její návratová hodnota je textová reprezentace zadaného polynomu a seznam kořenů. Její parametry jsou zadaný polynom v textové reprezentaci dle konstruktoru třídy Polynom, počet platných desetinných míst, na které se má počítat a příznak, zda má navrátit ve výpisu pouze reálné nebo všechny kořeny.
- Metoda history_sync slouží k vybrání polynomů zapsaných jako řetězec, které v historii přibyly od posledního volání této metody. Pokud metoda volána ještě nebyla, chová se stejně jako metoda history_sync_all, která vrací všechny předchozí vypočítávané polynomy zapsané jako textový řetězec uložené v historii.
- Metoda predchozi_vypocet vrací právě jeden předchozí výpočet podle celočíselného parametru, zformátovaný stejně, jak jej formátuje metoda vypis.

Třída Stare:

- Konstruktor Stare inicializuje vnitřní ArrayList.

- Metoda pridej na konec seznamu přidá zadaný polynom a textový řetězec, jež reprezentuje všechny jeho kořeny.
- Metody vydej_poly a vydej_koreny slouží pro přístup k obsahu vnitřního ArrayListu. Navrátí polynom nebo kořeny na zadaném místě. Pokud místo zadáno není, navrátí poslední přidané.
- Metoda pocet vrací počet prvků ve vnitřním ArrayListu.

Třída Cplx:

- Konstruktor Cplx vytvoří instanci komplexního čísla. Parametry jsou buď dvě čísla, a to složky komplexního čísla (kde první je reálná, druhá imaginární), nebo dvě čísla a příznak. Je-li příznak nepravda, jsou čísla brána jako složky komplexního čísla, pokud je pravda, jsou čísla brána polární souřadnice.
- Metody add, subtract, multiply a divide jsou základní početní operace (sčítání, odčítání, násobení, resp. dělení). Pro zachování přesnosti čísla se počítá pomocí reálné a imaginární složky, nikoliv pomocí polárních souřadnic.
- Metoda power umocňuje komplexní číslo na zadaný parametr. Opět pracuje pouze s reálnou a imaginární složkou a je napsána tak, aby výsledná složitost byla maximálně logaritmická. Funguje i pro záporné exponenty.
- Metoda root vrací všechny odmocniny komplexního čísla. V této metodě už je třeba pracovat s polárními souřadnicemi. Funguje i pro záporné exponenty.
- Metoda sdruzene vrací číslo komplexně sdružené k volající instanci.
- Metoda toString slouží k textovému vyjádření čísla na daný počet desetinných míst v daném souřadnicovém systému (systém je určen z příznaku podobně jako pro konstruktor). Pokud je zavolána bez parametrů, vrací číslo v kartézských souřadnicích na tři desetinná místa. Metoda umí úsporný zápis (tzn. např. číslo 0+1i umí zapsat pouze jako i apod.).
- Metody is_equal a is_zero slouží k porovnávání s jiným komplexním číslem, resp. s nulou. Vrací true pokud jsou si čísla rovny, jinak vrací false.
- Ke čtení složek čísla (reálné, imaginární, absolutní hodnotě a úhlu od kladného směru osy x) slouží metody dej_x, dej_y, dej_R, resp. dej_Fi.

Třída Polynom:

- Konstruktor Polynom sestrojí polynom ze zadaných koeficientů nebo z textové reprezentace "stupeň: koeficient_nejvyšší koeficient_nižší … koeficient_poslední". Pro textové zadání platí, že jsou-li koeficienty vynechány, je polynom doplněn nulovými a překračuje-li počet koeficientů daný stupeň, jsou poslední zahozeny. V metodě se používá regulárních výrazů pro mírnou úpravu chybných vstupů.
- Metoda koeficient vrací koeficient u mocniny zadané v parametru.
- Metoda as_array vrací všechny koeficienty jako pole komplexních čísel.
- Metoda secti sečte dva polynomy.
- Metoda poly_deleni vydělí polynom polynomem s tím, že zbytek po dělení zahodí. Pokud je dělitel lineární, dělení provádí privátní metoda horner_dělení, která dělí podle Hornerova schématu.
- Metoda stupen navrátí stupeň polynomu.
- Metoda toString navrátí textové vyjádření polynomu s tím, že koeficienty vypíše v daném souřadnicovém systému na daný počet desetinných míst stejně jako v metodě toString ve třídě Cplx.
- Metoda toConstructorString navrátí textové vyjádření polynomu v takovém formátu, v jakém jej přijímá konstruktor.
- Metody v_bode a v_nule slouží k vyjádření hodnoty polynomu v zadaném bodě a v počátku komplexní roviny.
- Metoda derivace navrátí polynom, který je derivací polynomu původního.
- Metoda sdruzene_koeficienty navrátí polynom, jehož koeficienty jsou čísla sdružená ke koeficientům původního polynomu.
- Metoda x_za_dvojclen navrátí polynom po roznásobení původního polynomu substitucí x=Ay+B.
- Metoda roznasob navrátí výsledek násobení dvou polynomů.
- Metoda koreny navrátí kořeny polynomu jako pole komplexních čísel. Tato metoda se odkazuje do balíčku metody. Pokud je stupeň polynomu menší než 3, použije se přímá metoda, jinak se použije metoda aproximační. V těle metody jsou předepsány všechny algoritmy, ale pouze jeden je odkomentován. Výchozí je algoritmus Laguerreův.

- Statická metoda poly_stupen slouží k určení stupně polynomu vyjádřeného jako pole komplexních čísel.

Třída Prime_metody:

- Obsahuje statické metody linear a kvadrat, které slouží k výpočtu kořenů lineárního, resp. kvadratického členu.

Třída Aprox_metody:

- Obsahuje statické metody, které se jmenují podle používaného algoritmu pro výpočet. Obsahuje také jednu privátní metodu operator, která slouží pro výpočet zvláštního operátoru pro Lehmer-Schurovu metodu.

Třída HelpMath:

- Statická metoda factorial spočítá faktoriál daného celého čísla
- Statická metoda kombinace spočítá kombinační číslo „n nad k“.

Třídy ComputeException a EmptyString mají pouze metody zděděné ze třídy Exception.

Všechny metody mají ve zdrojovém kódu napsán anglický Javadoc. Tento javadoc je také přiložen k dokumentaci ve složce javadoc.

## SOUBORY

U této dokumentace jsou následující soubory:

- src/*.java obsahují zdrojové kódy programu
- Poly.jar je přeložený Java archív, který lze spustit pomocí JRE
- Poly.exe je spustitelný soubor programu pro MS Windows vytvořený pomocí skriptu Jelude (http://www.geocities.com/java_intro/)
- Poly.sh je spustitelný soubor pro systém GNU/Linux
- javadoc/*.html jsou soubory pro Javadoc

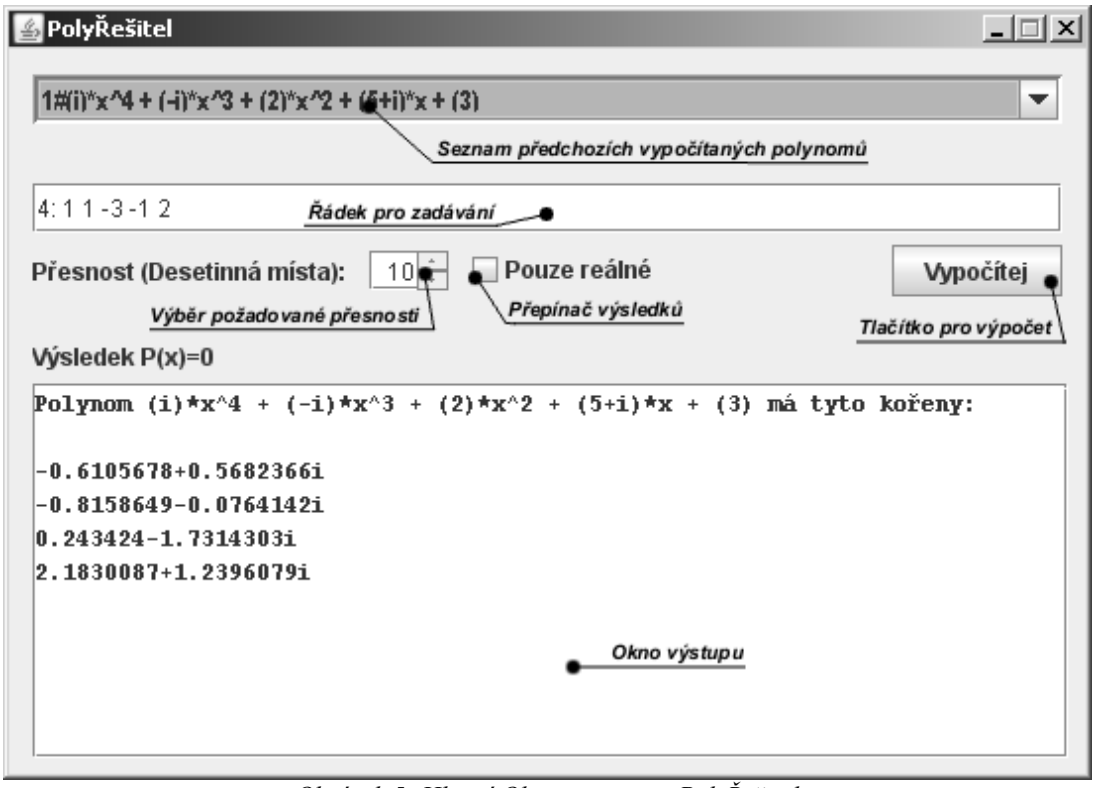
# PŘÍLOHA B: UŽIVATELSKÁ DOKUMENTACE PROGRAMU

## OBECNÝ POPIS

Program PolyŘešitel je multiplatformní program napsaný v jazyce Java. Pro jeho spuštění je potřeba mít nainstalováno JRE verze alespoň 1.5.

Spouštět program můžete pomocí souboru Poly.exe na systémech MS Windows, pro systémy GNU/Linux je vytvořen spouštěcí skript Poly.sh. Program je vždy možno spustit z příkazové řádky příkazem `java -jar Poly.jar`.

## POPIS OKNA PROGRAMU




*Obrázek 5: Hlavní Okno programu PolyŘešitel*

Seznam předchozích vypočítaných polynomů uchovává polynomy, které program spočítal. Při ukončení programu se tento seznam bez uložení vymaže. Polynomy jsou seřazeny sestupně podle pořadí výpočtu.

Řádek pro zadávání slouží k zadání polynomu, který má být spočítán.

Výběr požadované přesnosti udává, na kolik desetinných míst má být přesnost spočítaných kořenů.

Přepínač výsledků přepíná, zda jsou zobrazeny všechny kořeny polynomu nebo pouze reálné. Výpočet však vždy najde kořeny všechny.

Tlačítko pro výpočet zahajuje výpočet polynomu.

Okno výstupu slouží pro zobrazení výsledků nebo chybové zprávy.

## FORMÁT ZADÁVÁNÍ POLYNOMU

Polynom je zadáván v následujícím formátu: "$n: a_n\ a_{n-1}\ ...\ a_0$"

První číslo je stupeň polynomu a od ostatních je odděleno dvojtečkou. Následují koeficienty od koeficientu nejvyšší mocniny po nejnižší, navzájem oddělené mezerami. Pokud je počet koeficientů menší než kolik jich polynom daného stupně má, je zbytek doplněn nulami. Pokud je počet koeficientů naopak větší, jsou ty nejvíce vpravo ignorovány.

Komplexní koeficienty se zadávají jako součet reálné a imaginární části. Mezi složky komplexního čísla se mezera psát nesmí. Jako imaginární jednotku je možné použít znaky i nebo j. Jako desetinný oddělovač je možné použít tečku nebo čárku.

Program po zadání automaticky odfiltruje znaky, které v zadání nemají význam a mírné překlepy. Tato funkčnost ale není stoprocentní. Pro kontrolu je vždy zadaný polynom po úspěšném výpočtu i po chybě (kromě chyby špatného formátu) vypsán do okna výstupu.