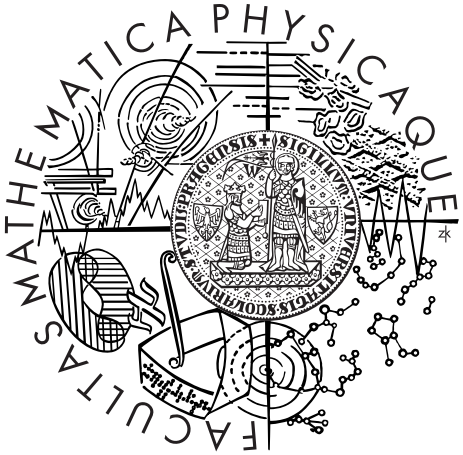
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Eva Havelková

Metody pro řešení nelineárních rovnic

Katedra numerické matematiky

Vedoucí bakalářské práce: RNDr. Václav Kučera, Ph.D.
Studijní program: Matematika
Studijní obor: Obecná matematika

Praha 2015

Děkuji vedoucímu práce panu RNDr. Václavu Kučerovi, Ph.D. za cenné rady a připomínky ke vznikající práci a za čas, který mi věnoval. Velké poděkování patří také mým rodičům za podporu ve studiu a zázemí, které mi poskytují. V neposlední řadě bych chtěla poděkovat i svému příteli za psychickou podporu a pomoc při práci s programem LaTeX.

Prohlašuji, že jsem tuto bakalářskou práci vypracovala samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle §60 odst. 1 autorského zákona.

V ..... dne ..... Podpis autora

Název práce: Metody pro řešení nelineárních rovnic

Autor: Eva Havelková

Katedra numerické matematiky

Vedoucí bakalářské práce: RNDr. Václav Kučera, Ph.D., Katedra numerické matematiky

Abstrakt: Cílem této bakalářské práce je shrnout základní numerické metody pro řešení nelineárních algebraických rovnic jedné proměnné. Nejprve jsou zavedeny nezbytné pojmy z numerické matematiky a matematické analýzy. Dále jsou podrobně popsány vybrané numerické iterační metody, konkrétně metoda půlení intervalu, metoda fixed-point iterace, metoda regula falsi, Newtonova metoda, metoda sečen a metody založené na kvadratické interpolaci, a to včetně důkazů řádů jejich konvergence. Praktická část práce se skládá z numerických experimentů provedených na různých typech nelineárních rovnic pomocí softwaru Matlab a porovnání výsledků s teoretickými poznatky z předchozích částí. Přínosem práce je především vytvoření uceleného přehledu teoretických vlastností základních metod řešení nelineárních rovnic čerpajícího z množství odborných zdrojů.

Klíčová slova: nelineární rovnice, iterační metody, numerické řešení rovnic, metody Newtonova typu

Title: Methods for the solution of nonlinear equations

Author: Eva Havelková

Department of Numerical Mathematics

Supervisor: RNDr. Václav Kučera, Ph.D., Department of Numerical Mathematics

Abstract: The aim of this bachelor thesis is to present an overview of elementary numerical methods for solving nonlinear algebraic equations in one variable. Firstly, related concepts from numerical mathematics and mathematical analysis are explained. The main part of the thesis provides a detailed description of chosen iterative methods as well as the proofs of their orders of convergence. The methods covered are namely the bisection method, fixed-point iteration, regula falsi method, Newton's method, secant method and methods that are based on quadratic interpolation. The practical part of the thesis presents results of numerical experiments that were carried out with Matlab software on various types of nonlinear equations. These results are compared with the theory introduced in the preceding parts. The contribution of this thesis is to provide a comprehensive overview and comparison of the characteristics of basic methods for solving nonlinear equations based on a variety of literature.

Keywords: nonlinear equations, iterative methods, numerical solution of equations, Newton-type methods

Obsah

1 Základní teorie **4**
- 1.1 Specifikace úlohy . . . 4
  - 1.1.1 Počet řešení . . . 4
- 1.2 Aproximace řešení . . . 5
  - 1.2.1 Podmíněnost úlohy . . . 5
  - 1.2.2 Iterační metody . . . 6
  - 1.2.3 Řád konvergence . . . 6
  - 1.2.4 Zastavovací kritéria . . . 7
- 1.3 Věty z matematické analýzy . . . 8

2 Základní metody **9**
- 2.1 Půlení intervalu (bisekce) . . . 9
  - 2.1.1 Algoritmus . . . 9
  - 2.1.2 Implementace . . . 10
  - 2.1.3 Řád konvegrence . . . 10
  - 2.1.4 Vlastnosti . . . 11
- 2.2 Fixed-point iterace . . . 11
  - 2.2.1 Existence řešení . . . 12
  - 2.2.2 Algoritmus a implementace . . . 13
  - 2.2.3 Konvergence algoritmu . . . 13
  - 2.2.4 Vlastnosti . . . 15
- 2.3 Regula falsi . . . 15
  - 2.3.1 Algoritmus . . . 15
  - 2.3.2 Implementace . . . 16
  - 2.3.3 Řád kovergence . . . 17

3 Newtonova metoda a její modifikace **19**
- 3.1 Newtonova metoda . . . 19
  - 3.1.1 Algoritmus . . . 19
  - 3.1.2 Implementace . . . 20
  - 3.1.3 Konvergence . . . 21
  - 3.1.4 Vlastnosti . . . 22
- 3.2 Newtonova metoda pro vícenásobné kořeny . . . 22
- 3.3 Newtonova metoda s konečnými diferencemi . . . 24
- 3.4 Backtracking pro Newtonovu metodu . . . 25
- 3.5 Metoda sečen . . . 26
  - 3.5.1 Algoritmus a implementace . . . 26
  - 3.5.2 Řád konvergence . . . 27

3.5.3 Vlastnosti 28
3.6 Metody založené na kvadratické interpolaci 29
3.6.1 Mullerova metoda 29
3.6.2 Inverzní kvadratická interpolace 30
3.6.3 Konvergence 31
3.6.4 Vlastnosti 32

4 Numerické experimenty **33**

Literatura **45**

A Zdrojové kódy **47**
A.1 Bisekce 47
A.2 Fixed-point iterace 48
A.3 Regula falsi 49
A.4 Newtonova metoda 50
A.5 Newtonova metoda pro vícenásobné kořeny 51
A.6 Newtonova metoda s backtrackingem 52
A.7 Metoda sečen 53
A.8 Mullerova metoda 54
A.9 Inverzní kvadratická interpolace 55

Úvod

Nelineární rovnice se objevují často jak v teoretických tak v praktických aplikacích matematiky. Typickým příkladem jsou polynomiální či transcendentní rovnice. V některých případech lze nalézt přesné řešení pomocí metod matematické analýzy nebo algebry, v mnohých případech ale přesné řešení nalézt neumíme, nebo jej ani nelze v radikálech vyjádřit, což například pro obecné polynomy stupně 5 a více plyne z Galoisovy teorie. V úvahu proto přichází hledání pouze přibližného řešení pomocí numerických metod. Velmi by se nám líbilo, kdyby existovala univerzální metoda aplikovatelná na jakoukoli nelineární rovnici. Takovou metodu bohužel zatím k dispozici nemáme, existuje však velké množství rozličných metod vhodných pro různé druhy nelineárních rovnic. V této práci se zaměříme na základní iterační metody pro řešení nelineárních algebraických rovnic jedné reálné proměnné a některé jejich modifikace. Vysvětlíme algoritmy, odvodíme podmínky, za kterých metody konvergují, a zhodnotíme výhody i nevýhody jejich použití a aplikovatelnost na různé rovnice.

V první kapitole nejprve specifikujeme problém, kterému se budeme věnovat. Dále zavedeme terminologii a některé důležité pojmy nezbytné pro podrobnější teoretické studium a uvedeme také několik základních vět z matematické analýzy, které budou potřebné pro důkazy teoretických vlastností metod v dalších kapitolách.

Ve druhé kapitole se seznámíme se třemi základními iteračními metodami. První z nich bude metoda půlení intervalu, která je nejzákladnější iterační metodou pro řešení nelineárních rovnic, a pomocí ní poté odvodíme metodu zvanou regula falsi. Podrobně bude rozebrána metoda fixed-point iterace, jenž bude mít velký význam pro studium pokročilejší Newtonovy metody ve třetí kapitole.

Třetí kapitola nese název Newtonova metoda a její modifikace. Nejprve se v ní čtenář seznámí s klasickou Newtonovou metodou a to včetně důkazu její lokální kvadratické konvergence a dále i s některými modifikacemi této metody, například pro vícenásobné kořeny. Na základě Newtonovy metody bude také odvozena metoda sečen, velmi známá metoda vhodná i pro nediferencovatelné funkce. Kapitola bude zakončena sekcí o metodách založených na kvadratické interpolaci.

Závěrečná, čtvrtá kapitola obsahuje numerické experimenty. Na různorodých příkladech bude pomocí výpočetního softwaru Matlab demonstrováno a porovnáváno chování metod uvedených ve druhé a třetí kapitole a to především na polynomiálních a transcendentních rovnicích. Bude provedeno nejen praktické ověření řádu konvergence metod na rovnicích splňujících předpoklady konvergenčních vět, ale také otestujeme chování metod aplikovaných na rovnice, o nichž nemáme z teoretické části žádné informace.

Kapitola 1

Základní teorie

V této kapitole si nejprve zadefinujeme úlohu, kterou se budeme dále zabývat. Podíváme se na množinu jejích řešení a poté na stejnou úlohu nahlédneme i z numerického hlediska. Nakonec uvedeme několik základních definic a vět známých z matematické analýzy, které budou důležité v následujících kapitolách pro studium teoretických vlastností různých metod řešení.

1.1 Specifikace úlohy

Mějme dánu nelineární spojitou skalární funkci $f : \mathbb{R} \to \mathbb{R}$. Naším úkolem je najít $x^* \in \mathbb{R}$ splňující

$$f(x^*) = 0. \tag{1.1}$$

Takové x^* budeme nazývat kořenem funkce f.

Poznámka. Speciálně lze někdy uvažovat $f : \mathbb{C} \to \mathbb{C}$. V závislosti na metodě řešení rovnice budeme také někdy přidávat další požadavky na funkci f, například existenci a spojitost derivací atp.

Funkce f bude většinou zadaná explicitně, například jako polynom či transcendentní funkce. Můžeme se však setkat i s implicitním zadáním, což znamená, že známe algoritmus pro výpočet $f(x)$ pro všechna $x \in \mathbb{R}$, ale neznáme explicitní vyjádření funkce f.

1.1.1 Počet řešení

Existence a jednoznačnost řešení obecné nelineární rovnice je na rozdíl od řešení rovnice lineární velmi složitý problém. Nelineární rovnice mohou mít libovolný počet řešení od žádného až po nekonečně mnoho. Obecná křivka (tj. graf funkce f) totiž může protnout osu x mnohem více způsoby, než přímka.

Příklad. Uveďme si pár typických příkladů nelineárních rovnic i s počtem řešení:

(i) $e^x + 1 = 0$ nemá žádné řešení,
(ii) $x^3 - 5 = 0$ má jedno reálné (a dvě komplexní) řešení,
(iii) $x^2 - \sin(7x) = 0$ má čtyři řešení,
(iv) $\cos(x) - 1 = 0$ má nekonečně mnoho řešení.

Nelineární rovnice mohou navíc mít i vícenásobné kořeny, což znamená, že nejen daná funkce, ale i její derivace (obecně až do řádu k) je v nějakém bodě nulová. Jako příklad uveďme rovnici (iv), která má zřejmě všechny své kořeny dvojnásobné.

1.2 Aproximace řešení

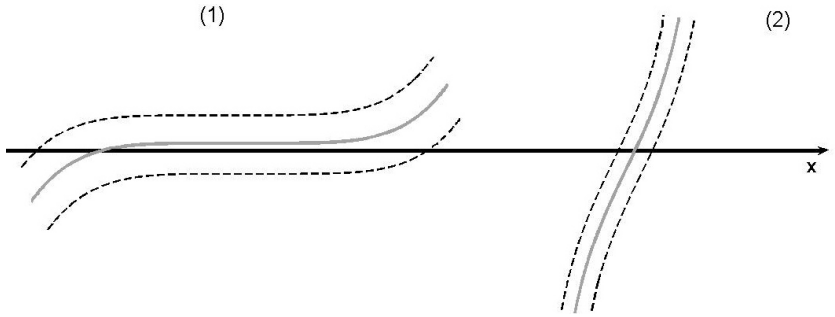
Pro některé funkce lze najít všechna přesná řešení rovnice (1.1), například pro faktorizovatelné polynomy, obecně však v něco podobného doufat rozhodně nemůžeme. Nejen, že nebudeme často schopni určit počet řešení rovnice, ale některá řešení nemusí jít ani v radikálech vyjádřit. Navíc v počítačové aritmetice (tj. v aritmetice s konečnou přesností) přesné řešení vůbec existovat nemusí. Naším cílem tedy nebude hledat všechna přesná řešení, ale budeme se snažit nalézt jedno dostatečně dobré přibližné řešení rovnice (1.1).

1.2.1 Podmíněnost úlohy

Vysvětlili jsme si, proč hledáme pouze přibližné řešení, nevysvětlili jsme si však, co tím myslíme. Nechť tedy f a x^* jsou jako v úvodu této kapitoly. Pak $\overline{x}$ nazveme přibližným řešením, pokud buď

$$(1)\ \|f(\overline{x})\| \approx 0 \quad \text{nebo} \quad (2)\ \|x^* - \overline{x}\| \approx 0.$$

První možnost říká, že máme malé reziduum, kdežto druhá možnost říká, že přibližné řešení je blízko přesnému řešení. Stejně jako například u řešení systémů lineárních rovnic tyto dvě možnosti nemusí nutně nastat zároveň. Ilustrujme si tento problém v jedné dimenzi na Obrázku 1.1. Obě funkce na obrázku mají



Obrázek 1.1: Ilustrace podmíněnosti funkce.

přibližně stejnou možnou odchylku ve svých hodnotách (vyznačenou čárkovaně), která může být způsobena například zaokrouhlovacími chybami v průběhu výpočtu. Zřejmě však je velký rozdíl v možné odchylce umístění jejich kořenů. Zatímco druhá funkce je dobře podmíněná, neboť malá změna ve vstupních datech způsobí pouze malou možnou změnu v poloze kořene, první funkce je evidentně špatně podmíněná [7].

Příklad. Špatně podmíněnou funkcí, se kterou se v praxi budeme často setkávat, jsou polynomy s vícenásobným kořenem.

1.2.2 Iterační metody

K výpočtu aproximace řešení budeme používat různé druhy iteračních metod. Definovat korektně obecnou iterační metodu je složité a pro nás poněkud zbytečné. Uveďme si tedy pouze, že iterační metodou budeme mít na mysli opakovanou aplikaci funkce F odvozené nějakým způsobem ze zadané funkce f. Funkce F může mít obecně k vstupních parametrů $\{x_i, x_{i+1},..., x_{i+k-1}\}$ (my budeme uvažovat $k \in \{1, 2, 3\}$) a jejím výstupem bude vždy jeden nový bod x_{i+k}, $i \in \mathbb{N} \cup \{0\}$. Parametry $\{x_0, x_1,..., x_{k-1}\}$ nutné pro inicializaci budou na počátku zadány. Konkrétní příklady iteračních metod rozebereme v následujících kapitolách.

1.2.3 Řád konvergence

Abychom mohli porovnávat efektivitu iteračních metod, potřebujeme charakterizovat rychlost konvergence.

Uvažme nějakou iterační metodu aplikovanou na rovnici (1.1) a x^*, přesné řešení této rovnice. Pak chybou v n-té iteraci budeme rozumět $e_n = x_n - x^*$. Pokud výsledkem n-té iterace dané metody není přibližné řešení, ale interval $[a_n, b_n]$ obsahující přesné řešení, pak chybou v n-té iteraci budeme rozumět $e_n = b_n - a_n$.

Definice 1 (řád konvergence)**.** Nechť $\{x_n\}$ je posloupnost daná nějakou iterační metodou a e_n je chyba v n-té iteraci. Řekneme, že metoda pro dané vstupní parametry $\{x_0, x_1,..., x_{k-1}\}$ konverguje k přesnému řešení x^*, jestliže

$$\lim_{n\to\infty} \|e_n\| = 0.$$

Dále řekneme, že metoda má *řád konvergence* r pro dané $\{x_0, x_1,..., x_{k-1}\}$, jestliže

$$\lim_{n\to\infty} \frac{\|e_{n+1}\|}{\|e_n\|^r} = C$$

pro nějakou kladnou reálnou konstantu C.
Konvergenci nazveme

- lineární, jestliže $r = 1$ a $C < 1$,
- superlineární, jestliže $r > 1$,
- kvadratickou, jestliže $r = 2$.

Poznámka. Konstantu C uvedenou v Definici 1 někdy také nazýváme rychlost konvergence.

Interpretace lineární konvergence je taková, že asymptoticky přidá lineárně konvergentní metoda v daném bodě v každém kroku konstantní počet platných cifer k výsledku. Superlineárně konvergentní posloupnost má asymptoticky v daném bodě v každé iteraci r-krát více platných číslic, než v předchozí iteraci. Speciálně tedy kvadraticky konvergentní metoda asymptoticky zdvojnásobuje počet platných číslic přibližného řešení každou iterací [7].

Lineární konvergence může být v praxi velmi pomalá, kdežto superlineární a kvadratická konvergence jsou většinou rychlé. Reálné chování metody závisí však

také na konstantě C. Například lineární konvergence s konstantou $C = 0.001$ může být celkem uspokojivá, kdežto lineární konvergence s konstantou $C = 0.9$ je velmi špatná [5].

Řád konvergence není jediným kritériem v posuzování efektivity dané metody. Důležitým faktorem pro určení ceny algoritmu je také počet operací provedených v každé iteraci, typicky počet vyhodnocování zadané funkce nebo nutnost výpočtu derivace funkce v jednotlivých iteracích. Jak uvidíme později, i pomaleji konvergující algoritmus může být efektivnější než jiný, rychleji konvergující algoritmus, za předpokladu, že v jednotlivých iteracích provádíme méně výpočetně náročných operací.

1.2.4 Zastavovací kritéria

Nelineární rovnice typicky nelze vyřešit v konečném počtu iterací, budeme proto volit takové metody, které nám dají posloupnost bodů konvergující k přesnému řešení nebo posloupnost intervalů obsahujících kořen a konvergujících ve velikosti k nule. Metodu zastavíme ve chvíli, kdy dostaneme dostatečně dobré přibližné řešení.

Otázkou je, na základě čeho poznáme, že obdržené přibližné řešení je dostatečně dobré. Jednou možností je sledovat hodnotu $f(x_n)$ a výpočet zastavit, je-li $|f(x_n)| < \epsilon$, kde ϵ je požadovaná přesnost. Možnou modifikací tohoto kritéria tak, aby více reflektovalo oblast, kde řešení hledáme, je $|f(x_n)| < \epsilon\,|f(x_0)| + \epsilon_1$, kde $\epsilon_1 > 0$ zabrání nesplnitelnosti kritéria pro $|f(x_0)|$ velmi malé [11]. Je-li však funkce f špatně podmíněná, pak nám toto ani předchozí kritérium nedávají jistotu, že $|x^* - x_n|$ je malé.

Další možností je sledovat rozdíl v po sobě jdoucích iteracích. První variantou tohoto kritéria je absolutní rozdíl, tj. metodu zastavíme pokud $|x_n - x_{n-1}| < \epsilon$. Druhou variantou je rozdíl relativní, tj. výpočet zastavujeme na základě velikosti $\frac{|x_n - x_{n-1}|}{|x_n|}$. Toto kritérium však není vhodné pro $|x_n| \approx 0$. V takovém případě je lepší jmenovatel nahradit výrazem $max\{x_{typ}, |x_n|\}$, kde x_{typ} je hodnota zadávaná uživatelem a často se volí například $x_{typ} = 1$. Metodu pak zastavíme, je-li $\frac{|x_n - x_{n-1}|}{max\{x_{typ}, |x_n|\}} < \epsilon$. Ani jedna z těchto dvou variant nám však obecně nedává informaci o $|x_n - x^*|$ ani o hodnotě $|f(x)|$. Přesto tato kritéria mají význam, neboť pokud se po sobě následující iterace liší o velmi málo, stojí za zvážení, zda má smysl počítat další iterace [4].

Důležitá je také přiměřená volba ϵ. Požadovat $\epsilon = \epsilon_{mach}$, kde ϵ_{mach} je strojová přesnost, je zřejmě přehnané. Rozumnou volbou je například $\epsilon = \sqrt{\epsilon_{mach}}$, viz [5].

Obecně vhodným zastavovacím kritériem je maximální možný počet iterací. Toto kritérium nám sice nedává žádnou informaci o kvalitě obdrženého řešení, avšak jeho použitím v kombinaci s dalšími kritérii zamezíme problémům, které mohou nastat při zadání neúměrných požadavků na přesnost hledané aproximace řešení [4].

Zastavovací kritérium rozhodně nelze volit obecně, musíme jej přizpůsobit dané úloze a zvolené metodě. Konkrétní příklady se zdůvodněním dané volby si uvedeme u jednotlivých metod v následujících kapitolách.

1.3 Věty z matematické analýzy

Věta 1 (o nabývání mezihodnot [8]). *Nechť $a, b \in \mathbb{R}$, $a \neq b$ a nechť funkce f je spojitá na intervalu $[a, b]$, $f(a) \neq f(b)$. Pak ke každému $c \in (f(a), f(b))$ existuje $\xi \in (a, b)$ takové, že platí $f(\xi) = c$.*

Věta o nabývání mezihodnot je důležitý teoretický nástroj, který nám při splnění daných předpokladů zaručí existenci řešení rovnice (1.1) v určitém intervalu.

Definice 2 (kontrakce). Nechť X je metrický prostor, ρ značí příslušnou metriku a $F : X \to X$ je zobrazení. Řekneme, že F je *kontrakce* na X, jestliže existuje $\gamma \in [0, 1)$ takové, že pro každé $x, y \in X$ platí

$$\rho(F(x), F(y)) \leq \gamma\rho(x, y).$$

Definice 3 (pevný bod). Nechť X je metrický prostor, $F : X \to X$ je zobrazení. Pak $x \in X$ splňující $F(x) = x$ nazveme *pevným bodem* zobrazení F.

Věta 2 (Banachova o kontrakci [8]). *Nechť X je úplný metrický prostor a $F : X \to X$ je kontrakce. Pak F má v X právě jeden pevný bod.*

Na první pohled se může zdát, že tato věta příliš nesouvisí s řešením úlohy (1.1). V Kapitole 2 si však ukážeme, že hledání kořenů rovnice můžeme snadno převést na hledání pevných bodů nějakého zobrazení. Banachova věta o kontrakci je proto pro nás velmi silný teoretický nástroj. Poznamenejme však, že v praktickém řešení nám tato věta často nepomůže, protože splnit její předpoklady může být obtížné.

Věta 3 (o střední hodnotě [8]). *Nechť $a, b \in \mathbb{R}$, $a < b$, f je funkce spojitá na intervalu $[a, b]$ a nechť $f'(x)$ existuje pro všechna $x \in (a, b)$. Pak existuje $\xi \in (a, b)$ splňující*

$$f'(\xi) = \frac{f(b) - f(a)}{b - a}.$$

Definice 4 (Lagrangeův interpolační polynom). Nechť $n \in \mathbb{N}$, $f : \mathbb{R} \to \mathbb{R}$ a $\{x_0, x_1,..., x_n\} \subset \mathbb{R}$. Pak *Lagrangeův interpolační polynom n-tého stupně* definujeme

$$L_n(x) = f(x_0)l_0(x) + f(x_1)l_1(x) + ... + f(x_n)l_n(x),$$

kde

$$l_i(x) = \frac{(x - x_0)}{(x_i - x_0)}...\frac{(x - x_{i-1})}{(x_i - x_{i-1})}\frac{(x - x_{i+1})}{(x_i - x_{i+1})}...\frac{(x - x_n)}{(x_i - x_n)}.$$

Poznámka. Lagrangeův interpolační polynom $n-$tého stupně splňuje $L_n(x_i) = f(x_i)$ a $l_i(x_j) = \delta_{ij}\ \forall\ i, j \in \{0,...,n\}$.

Věta 4 (zbytek Lagrangeova interpolačního polynomu [16]). *Nechť $n \in \mathbb{N}$, $[a, b] \subset \mathbb{R}$ je omezený neprázdný interval a f je spojitě diferencovatelná funkce až do řádu $n + 1$ na $[a, b]$, v krajích jednostranně. Nechť $x_0, x_1,..., x_n$ jsou navzájem různá čísla z intervalu $[a, b]$. Pak pro každé $x \in [a, b]$ existuje $\xi \in (a, b)$ takové, že*

$$f(x) = P(x) + \frac{f^{(n+1)}(\xi)}{(n + 1)!}(x - x_0)(x - x_1)...(x - x_n),$$

kde $P(x)$ je Lagrangeův interpolační polynom stupně nejvýše n splňující $f(x_i) = P(x_i)$ pro všechna $i \in \{1, 2,..., n\}$.

Kapitola 2

Základní metody

V této kapitole rozebereme tři základní iterační metody použitelné na různé typy rovnic. Vysvětlíme algoritmy jejich výpočtu, vhodnost aplikace na konkrétní rovnice a výhody či nevýhody při použití. Podíváme se také na řád konvergence a s ním spojenou výpočetní náročnost jednotlivých metod.

2.1 Půlení intervalu (bisekce)

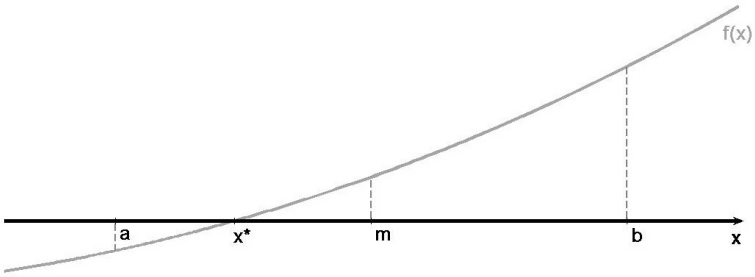
Bisekce je asi nejjednodušší a také nejznámější metoda pro řešení nelineárních rovnic. Jak jsme již zmínili v úvodní kapitole, pro nelineární funkce a obzvláště pak v aritmetice s konečnou přesností nemusí existovat přesné řešení rovnice (1.1). Jednou z možností, jak k takovému problému přistupovat, je hledat dostatečně malý interval $[a, b]$, o kterém víme, že obsahuje kořen.

2.1.1 Algoritmus

Myšlenka metody je snadná. Uvažujme neprázdný omezený interval $[a, b]$ a funkci f, která je na tomto intervalu spojitá a navíc splňuje $f(a)f(b) \leq 0$. Pak podle Věty 1 existuje bod $x^* \in [a, b]$ splňující $f(x^*) = 0$. Chceme nějak systematicky zmenšovat tento interval tak, aby stále obsahoval kořen. Jak název metody napovídá, zvolme bod m uprostřed intervalu $[a, b]$, tedy

$$m := a + \frac{(b - a)}{2}.$$

Pak buď $f(m) = 0$ a máme výsledek, nebo na jednom z intervalů $[a, m]$, $[m, b]$ má funkce f různá znaménka v koncových bodech. Takový podinterval zvolíme jako náš nový interval a provedeme další iteraci. Tento proces opakujeme tak dlouho, dokud velikost nového intervalu není menší než předem definovaná přesnost. Metoda je ilustrována na Obrázku 2.1.



Obrázek 2.1: Ilustrace bisekce.

2.1.2 Implementace

Předpokládejme, že na vstupu máme zadanou spojitou funkci f, interval $[a, b]$ tak, že $f(a)f(b) < 0$, a požadovanou přesnost tol. Ve zjednodušené syntaxi pak může algoritmus vypadat následovně

while $(b - a) > tol$ **do**
  $m := a + (b - a)/2$;
  if $sign(f(a)) = sign(f(m))$ **then**
    a:=m;
  else
    b:=m;
end

Poznamenejme pár důležitých věcí k uvedené implementaci algoritmu. Nejprve si všimněme volby výpočtu bodu m. Přímočařejším způsobem by jistě bylo počítat $m = (a + b)/2$, v aritmetice s končenou přesností by však tato volba mohla způsobit problémy. Hodnota $(a + b)/2$ vlivem aritmetiky s konečnou přesností nemusí nutně spadnout do intervalu $[a, b]$, kdežto hodnota $a + (b - a)/2$ bude vždy v intervalu $[a, b]$. Dalším zajímavým problémem implementace je testování, zda hodnoty $f(a), f(m)$ mají stejná znaménka. Matematicky je tento problém ekvivalentní testování zda $f(a)f(m) > 0$, v počítačové aritmetice by tato volba mohla být riskantní. Jsou-li hodnoty $f(a), f(m)$ nenulové avšak velmi malé (což jistě nastane, budeme-li se blížit ke kořeni), může jejich součin být numericky nulový. Bezpečnější tedy je využít předdefinované funkce *signum,* viz [2].

Za zmínku stojí i zastavovací kritérium. V Kapitole 1 jsme uvedli, že jeho volba by měla reflektovat zvolenou metodu i úlohu. U metody bisekce se často volí

$$|b_n - a_n| < \frac{|a| + |b|}{2}\sqrt{\epsilon_{mach}},$$

kde ϵ_{mach} je strojová přesnost. Zároveň můžeme omezit i maximální počet iterací, který by neměl překročit 40-50, viz [12].

2.1.3 Řád konvegrence

Nyní na základě [12] dokážeme, že metoda půlení intervalu konverguje lineárně. Odvodíme také, kolik iterací je nutné provést pro nalezení přibližného řešení zadané přesnosti.

Věta 5 (řád konvergence bisekce)**.** *Nechť $a, b \in \mathbb{R}$, $a < b$ a $f : [a, b] \to \mathbb{R}$ je spojitá funkce splňující $f(a)f(b) < 0$. Pak metoda bisekce konverguje lineárně s rychlostí $1/2$. Je-li navíc $[a_n, b_n]$ interval v $n-$té iteraci a $\delta \in \mathbb{R}$, pak je třeba provést $\log_2(\frac{b-a}{\delta}) + 1$ iterací, aby $|b_n - a_n| < \delta$.*

Důkaz. Zřejmě pro každé $n \in \mathbb{N}$ je po n iteracích $|b_n - a_n| = (b - a)/2^n$, a tedy

$$\lim_{n\to\infty} |b_n - a_n| = \lim_{n\to\infty} (b - a)/2^n = 0.$$

Zřejmě také přesné řešení x^* splňuje: $x^* \in [a_n, b_n]$ $\forall n \in \mathbb{N}$, tedy metoda konverguje. Navíc pro $r = 1$ platí

$$\lim_{n\to\infty} \frac{\|e_{n+1}\|}{\|e_n\|^r} = \lim_{n\to\infty} \frac{\frac{(b-a)}{2^{n+1}}}{\frac{(b-a)}{2^n}} = \frac{1}{2}.$$

Dokázali jsme, že metoda konverguje lineárně s rychlostí $C = \frac{1}{2}$.
Pro $\delta \in \mathbb{R}$ chceme: $|b_n - a_n| = (b-a)/2^n < \delta$, tedy musí platit $n > log_2(\frac{b-a}{\delta})$. □

2.1.4 Vlastnosti

V důkazu Věty 5 si můžeme všimnout, že na funkci f ani na vstupní interval neklademe žádné dodatečné podmínky. Metoda bisekce je tedy velmi robustní. Máme zaručeno, že vždy najde aproximaci řešení zadané přesnosti a to navíc v předem známém počtu iterací. Je-li v daném intervalu více kořenů, pak metoda konverguje k některému z nich, dopředu ovšem nedokážeme odhadnout ke kterému. Poznamenejme také, že metodu nelze použít pro funkce s kořeny se sudou násobností, neboť tyto funkce v kořeni nemění znaménko.

Výhodou metody je, že nepotřebujeme znát derivace funkce f a proto ji můžeme používat i pro hledání kořenů nediferencovatelných funkcí. Metoda půlení intervalu však využívá jen velmi málo vlastností vstupní funkce k nalezení jejích kořenů. To je hlavní důvod, proč tato metoda většinou konverguje pomaleji než jiné metody, kterými se budeme zabývat v následujících sekcích [7].

Protože je metoda spolehlivá, ale její konvergence je pomalá, používá se často u sofistikovanějších metod jako startovací, abychom získali z počátečního nepřesného odhadu mnohem lepší odhad pro polohu kořene a následně mohli aplikovat rychleji konvergující metodu, která vyžaduje přesnější vstupní data než bisekce [12]. Metodu lze navíc rozšířit i do více dimenzí, pak je ovšem výpočetní náročnost mnohonásobně vyšší.

2.2 Fixed-point iterace

V praxi se někdy může stát, že budeme chtít najít řešení rovnice

$$g(x) = x \tag{2.1}$$

pro nějakou známou funkci g. V následující sekci dokážeme existenci řešení tohoto problému, odvodíme algoritmus pro numerické řešení a podíváme se na jeho vlastnosti. Tato úloha bude mít pro nás velký význam při rozvíjení dalších metod pro řešení rovnice (1.1), neboť hledání kořene rovnice lze snadno převést na hledání pevného bodu nějakého zobrazení.

2.2.1 Existence řešení

Existenci řešení rovnice (1.1) nám zaručovala Věta o nabývání mezihodnot 1. Na základě knihy [4] nyní dokážeme její alternativou pro rovnici (2.1).

Věta 6 (existence pevného bodu)**.** *Nechť g je spojitá funkce na neprázdném intervalu $[a, b]$ splňující $g(x) \in [a, b]$ $\forall x \in [a, b]$. Pak g má v $[a, b]$ pevný bod.*

Důkaz. Je-li $g(a) = a$ nebo $g(b) = b$, věta zřejmě platí. Předpokládejme tedy, že $g(a) \neq a$ a $g(b) \neq b$. Pak z předpokladu věty plyne $g(a) > a$ a zároveň $g(b) < b$. Položme $h(x) := g(x) - x$. Pak funkce h je spojitá a splňuje $h(a) > 0$ a $h(b) < 0$. Podle Věty 1 proto existuje $x^* \in [a, b]$ splňující $h(x^*) = 0$ a tedy $g(x^*) = x^*$. □

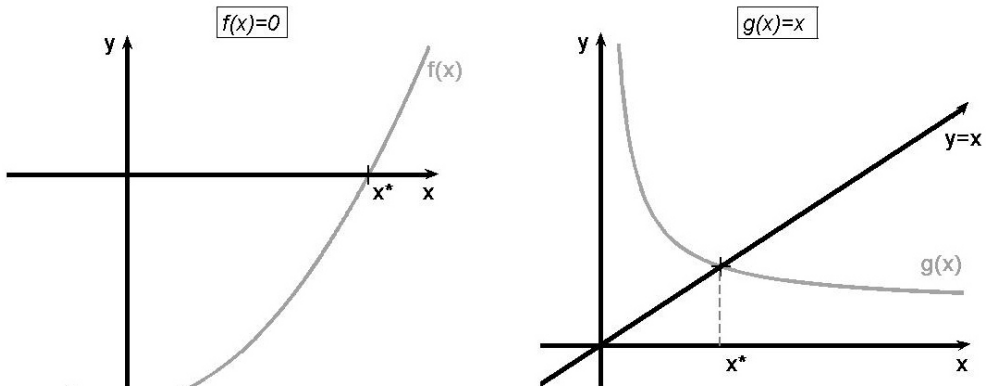
Nyní se zamysleme nad tím, jak převést problém hledání kořene rovnice na problém hledání pevného bodu. Mějme zadanou funkci f pro úlohu (1.1) a položme $g(x) = f(x) - x$. Pak zřejmě $f(x^*) = 0$ právě tehdy, když $g(x^*) = x^*$. Možností volby funkce g je však mnohem více, jak si ukážeme na následujícím příkladu.

Příklad. Uvažme rovnici $f(x) = x^2 - x - 2 = 0$. Funkci g můžeme zvolit například takto

(1) $g(x) = x^2 - 2$,
(2) $g(x) = \sqrt{x + 2}$,
(3) $g(x) = 1 + 2/x$,
(4) $g(x) = \frac{x^2+2}{2x-1}$.

Pak každé řešení rovnice $g(x) = x$ pro všechny uvedené funkce g je zároveň řešením rovnice $f(x) = 0$. Otázkou, které z těchto možností volby jsou vhodné a které ne, se budeme zabývat v Příkladu 1.

Rozdíl v hledání řešení rovnice $f(x) = 0$ a $g(x) = x$ lze snadno nahlédnout graficky, viz Obrázek 2.2. Při hledání kořene rovnice nás zajímá průsečík funkce f s osou x, kdežto při hledání pevného bodu nás zajímá průsečík funkce g s osou $y = x$.



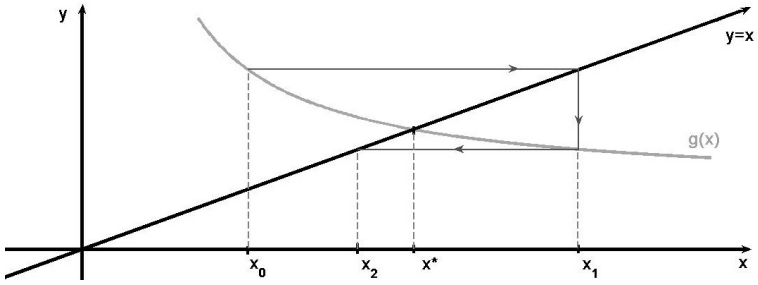
Obrázek 2.2: Rozdíl v hledání kořene funkce f a pevného bodu funkce g.

2.2.2 Algoritmus a implementace

Uvažujme funkci g, která je spojitá na neprázdném intervalu $[a, b]$ a splňuje $g(x) \in [a, b]\ \forall x \in [a, b]$. Pak dle Věty 6 existuje bod $x^* \in [a, b]$ splňující $g(x^*) = x^*$. Zvolme bod $x_0 \in [a, b]$ a položme $x_1 = g(x_0)$. Pak buď $x_1 = x_0$ a máme řešení, nebo provedeme další iteraci. Algoritmus zastavíme ve chvíli, kdy dostaneme dostatečně dobré přibližné řešení. Odvodili jsme algoritmus fixed-point iterace:

$$x_{n+1} = g(x_n).$$

Průběh výpočtu lze hezky ilustrovat graficky, první dvě iterace jsou znázorněny v Obrázku 2.3.



Obrázek 2.3: Ilustrace průběhu fixed-point iterace.

Přirozenou volbou zastavovacího kritéria je $|g(x_n) - x_n| < tol$, kde tol je požadovaná přesnost. Ve zjednodušené syntaxi může algoritmus vypadat takto

while $(|g(x_0) - x_0| > tol)$ **do**
  $x_0 := g(x_0)$;
end

Na první pohled si můžeme všimnout jednoduchosti algoritmu. V každém kroku potřebujeme pouze jedno vyhodnocení funkce g, nepotřebujeme počítat její derivaci ani provádět další výpočty. Cena každé iterace bude tedy záviset pouze na ceně vyhodnocení funkce g, která však někdy může být vysoká.

Poznámka. Název metody pochází z angličtiny a do češtiny se někdy překládá jako metoda prosté iterace, ale někdy také jako metoda pevného bodu. Vzhledem k neustálené terminologii se budeme dále držet označení fixed-point iterace.

2.2.3 Konvergence algoritmu

Z algoritmu vidíme, že požadavek existence neprázdného intervalu $[a, b]$ splňujícího $g(x) \in [a, b]\ \forall x \in [a, b]$ je nezbytný. Jedině tak máme pro každý vstupní bod $x_0 \in [a,b]$ zaručeno, že posloupnost $\{x_n\}$ daná algoritmem fixed-point iterace bude dobře definovaná pro všechna $n \in \mathbb{N}$. Tento požadavek zároveň se spojitostí funkce g nám však stále nedávají jistotu konvergence algoritmu. K tomu nám poslouží následující věta, která pochází z knihy [4].

Věta 7 (konvergence fixed-point iterace)**.** *Nechť $[a, b]$ je neprázdný interval a g je spojitě diferencovatelná funkce na $[a, b]$ splňující $g(x) \in [a, b]\ \forall x \in [a, b]$. Nechť*

navíc existuje $K \in [0,1)$ *takové, že* $|g'(x)| \leq K$, $\forall x \in [a,b]$. *Pak* g *má právě jeden pevný bod* $\xi \in [a,b]$ *a posloupnost* $\{x_n\}$ *generovaná fixed-point iterací konverguje ke* ξ *pro libovolné* $x_0 \in [a,b]$.

Důkaz. Existence pevného bodu plyne z Věty 6. Nechť tedy ξ je pevným bodem g v $[a,b]$, $x_0 \in [a,b]$ je libovolné. Označme $e_n = x_n - \xi$ chybu v $n-$té iteraci pro $n \in \mathbb{N} \cup \{0\}$. Protože $x_n = g(x_{n-1})$ a $\xi = g(\xi)$, platí

$$e_n = g(x_{n-1}) - g(\xi) = g'(\eta_n)(x_n - \xi) = g'(\eta_n)e_{n-1}$$

pro nějaké $\eta_n \in [\xi, x_{n-1}]$ dle Věty o střední hodnotě 3. Z předpokladu máme $|g'(\eta_n)| \leq K$, tedy $|e_n| \leq K\,|e_{n-1}|$ a indukcí dostáváme

$$|e_n| \leq K\,|e_{n-1}| \leq K^n\,|e_0|\,.$$

Neboť $0 \leq K < 1$, je $\lim_{n\to\infty} K^n = 0$ a proto

$$\lim_{n\to\infty} |e_n| = \lim_{n\to\infty} K^n\,|e_0| = 0.$$

Posloupnost x_n konverguje k pevnému bodu ξ. Zároveň jsme dokázali jednoznačnost, neboť pokud by ζ byl také pevný bod, položíme $x_0 = \zeta$ a dostaneme $x_1 = g(x_0) = \zeta$. Pak $|e_0| = |e_1| \leq K\,|e_0|$, tudíž $|e_0| = 0$, neboli $\xi = \zeta$. □

Poznámka. Pro spojitě diferencovatelnou $g : [a,b] \to [a,b]$ splňující $|g'(x)| < 1$ $\forall x \in [a,b]$ plyne z Věty o střední hodnotě 3, že pro každé $x, y \in [a,b]$ existuje $\eta \in [x,y]$ splňující $|g(x) - g(y)| = |g'(\eta)|\,|x - y| \leq |x - y|$ a tedy g je kontrakce. Existence a jednoznačnost pevného bodu ve Větě 7 tedy plyne přímo z Banachovy věty o kontrakci 2.

Ověřit podmínku, že g zobrazuje interval $[a,b]$ zpět do $[a,b]$ nemusí být v praxi snadné. V následujících dvou větách proto za mírnějších předpokladů nejprve dokážeme alespoň lokální konvergenci fixed-point iterace a poté také řád její konvergence. Vycházet budeme z knihy [7].

Definice 5 (lokální konvergence)**.** Řekneme, že iterační metoda *konverguje lokálně* k bodu ξ, jestliže existuje okolí U bodu ξ takové, že posloupnost $\{x_n\}$ daná touto metodou konverguje ke ξ pro každý počáteční bod $x_0 \in U$.

Věta 8 (lokální konvergence fixed-point iterace)**.** *Nechť funkce* g *je spojitě diferencovatelná na otevřeném intervalu* I *obsahujícím pevný bod funkce* g. *Nechť navíc* $|g'(\xi)| < 1$, *kde* ξ *je pevný bod* g. *Pak existuje* $\epsilon > 0$ *takové, že posloupnost* $\{x_n\}$ *daná fixed-point iterací pro funkci* g *konverguje ke* ξ *pro všechna* x_0 *splňující* $|x_0 - \xi| \leq \epsilon$.

Důkaz. Neboť g je spojitě diferencovatelná na okolí ξ a $|g'(\xi)| < 1$, existuje $\epsilon > 0$ takové, že $|g'(x)| < 1$ pro každé $x \in U := [\xi - \epsilon, \xi + \epsilon]$. Pak pro libovolné $x \in U$ z Věty o střední hodnotě 3 plyne $g(x) - \xi = g(x) - g(\xi) = g'(\eta)(x - \xi)$ pro nějaké $\eta \in [x, \xi]$ a tedy $|g(x) - \xi| \leq |g'(\eta)|\,|x - \xi| < |x - \xi|$. Proto $g(x) \in U$ pro každé $x \in U$ a tedy g splňuje na U předpoklady Věty 7. □

Věta 9 (řád konvergence fixed-point iterace)**.** *Nechť funkce* g *je spojitě diferencovatelná na otevřeném intervalu* I *obsahujícím pevný bod funkce* g. *Nechť navíc*

$0 < |g'(\xi)| < 1$, *kde* ξ *je pevný bod* g. *Pak posloupnost* $\{x_n\}$ *daná fixed-point iterací konverguje lokálně lineárně s rychlostí* $C = |g'(\xi)|$.

Důkaz. Lokální konvergenci jsme dokázali ve Větě 8. Dále z Věty o střední hodnotě 3 plyne $g(x_n) - \xi = g(x_n) - g(\xi) = g'(\eta_n)(x_n - \xi)$ pro $\eta_n \in [x_n, \xi]$. Pak

$$\lim_{n\to\infty} \frac{|x_{n+1} - \xi|}{|x_n - \xi|} = \lim_{n\to\infty} |g'(\eta_n)| = |g'(\xi)| = C < 1.$$

Existenci limity máme zaručenou díky již dokázané konvergenci algoritmu. □

Z uvedených vět dostáváme odpověď na otázku, jak volit funkci g pro fixed-point iteraci na základě zadané funkce f. Je zřejmé, že čím menší bude konstanta C, tím rychlejší konvergenci dostaneme. Pokud bude konstanta C ve Větě 9 nulová, pak lze dokonce ukázat, že konvergence bude alespoň kvadratická. Jakým způsobem volit funkci g, aby toto nastalo, se budeme zabývat v Kapitole 3.

2.2.4 Vlastnosti

Z důkazu Věty 8 plyne, že podmínka $|g'(\xi)| < 1$ je postačující pro lokální konvergenci, není však nutnou podmínkou. Metoda může konvergovat i pokud tato podmínka splněna není. Jako příklad si uveďme funkci $g(x) = \sin(x)$, jenž má pevný bod v 0. Posloupnost daná fixed-point iterací pro tuto funkci a počáteční bod $x_0 = 1$ konverguje k 0 i přesto, že $g'(0) = 1$, avšak konvergence je neúnosně pomalá. Dalším zajímavým způsobem chování metody při nesplnění postačujících podmínek pro konvergenci se budeme zabývat v Příkladu 7.

Důležitou vlastností algoritmu je, že jej lze velmi snadno rozšířit z jedné dimenze do více a průběh výpočtu zůstane stejný, viz [10].

2.3 Regula falsi

Metoda regula falsi nebo také false position method je další známou metodou pro řešení rovnice (1.1). Stejně jako u bisekce se místo hledání přesného řešení budeme snažit najít malý interval obsahující řešení. Již jsme se zmínili o tom, že pokud využíváme jenom málo vlastností vstupní funkce, metoda bude pravděpodobně konvergovat pomalu. Pokusíme se tedy modifikovat algoritmus bisekce tak, aby více využíval vlastností vstupních dat.

2.3.1 Algoritmus

Uvažujme opět neprázdný omezený interval $[a, b]$ a funkci f, která je na tomto intervalu spojitá a navíc splňuje $f(a)f(b) < 0$. Pak podle Věty 1 existuje bod $x^* \in [a, b]$ splňující $f(x^*) = 0$. Chceme systematicky zmenšovat tento interval tak, aby stále obsahoval kořen. Na rozdíl od metody bisekce, kde za nový bod bereme střed intervalu $[a, b]$, uvažme bod m odpovídající váženému průměru

$$m := \frac{|f(b)|\, a + |f(a)|\, b}{|f(b)| + |f(a)|}.$$

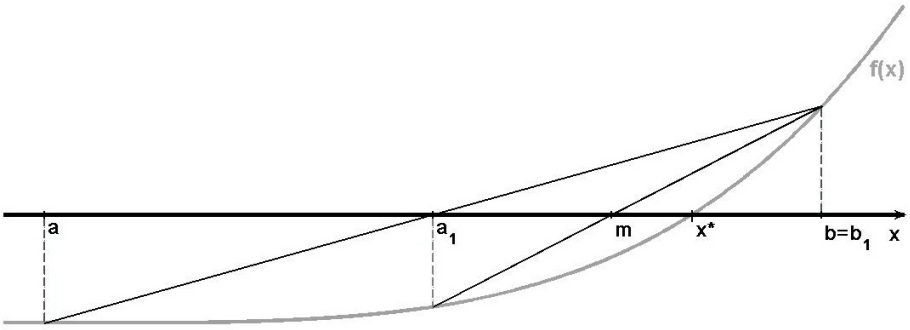
Neboť z předpokladu $f(a)$ a $f(b)$ mají opačná znaménka, můžeme psát

$$m = \frac{f(b)a - f(a)b}{f(b) - f(a)} = a - f(a)\frac{b-a}{f(b) - f(a)}.$$

Myšlenka této volby bodu je taková, že na mnohé nelineární funkce lze na malém intervalu [a, b] pohlížet jako na téměř lineární. Aproximujeme-li f lineární funkcí l splňující $l(a) = f(a)$, $l(b) = f(b)$, pak bod m je kořenem funkce l.

Máme-li zvolen bod m, dále již postupujeme stejně jako u bisekce. Buď $f(m) = 0$ a máme výsledek, nebo na jednom z intervalů $[a, m]$, $[m, b]$ má funkce f různá znaménka v koncových bodech. Takový podinterval zvolíme jako náš nový interval a provedeme další iteraci.

Nevýhoda metody je, že konvergence může být pouze jednostranná, což znamená, že délka intervalů v posloupnosti $[a_n, b_n]$ nejde pro $n \to \infty$ k 0. Pak ovšem buď $a_n \to x^*$ nebo $b_n \to x^*$, kde x^* je přesné řešení [4]. Ilustrujme si tuto situaci na Obrázku 2.4.



Obrázek 2.4: Jednostranná konvergence metody regula falsi.

Při volbě zastavovacího kritéria nám tedy nestačí pouze délka intervalu $[a_n, b_n]$, ale musíme sledovat, zda některá z krajních hodnot intervalu nekonverguje ke kořeni.

2.3.2 Implementace

Předpokládejme nyní, že na vstupu máme zadanou spojitou funkci f, interval $[a, b]$ tak, že $f(a)f(b) < 0$, a požadovanou přesnost tol. Ve zjednodušené syntaxi pak algoritmus může vypadat takto

while $(|f(a)| > tol$ **and** $|f(b)| > tol)$ **do**
  $m := a - f(a)\frac{b-a}{f(b)-f(a)}$;
  if $sign(f(a)) = sign(f(m))$ **then**
    $a := m$;
  else
    $b := m$;
end

Implementace algoritmu je velmi podobná jako u metody bisekce, poznámky k její implementaci je možné najít v sekci 2.1.

2.3.3 Řád kovergence

Metoda regula falsi konverguje vždy, někdy však dokonce pomaleji, než metoda půlení intervalu. V následující větě si na základě [9] dokážeme, že za jistých předpokladů konverguje metoda lineárně. Poznamenejme ještě, že metoda je nevhodná pro nelineární funkce, které mají v kořeni nulovou derivaci, neboť pro tyto funkce metoda konverguje neúnosně pomalu, viz Příklad 6.

Věta 10 (lineární konvergence metody regula falsi)**.** *Nechť f je spojitá funkce na neprázdném omezeném intervalu $[a,b]$, $f''(x)$ existuje na intervalu (a,b) a $\{[a_n,b_n]\}$ je posloupnost intervalů daná metodou regula falsi. Nechť navíc existuje $n \in \mathbb{N}$ tak, že*

(1) $a_n < b_n$,
(2) $f(b_n) > 0$ *a* $f(a_n) < 0$,
(3) $f''(x) > 0$

pro všechna $x \in [a_n, b_n]$. Pak metoda konverguje jednostranně a lineárně ke kořeni funkce f.

Důkaz. Nechť $m = \frac{f(b_n)a_n - f(a_n)b_n}{f(b_n)-f(a_n)}$. Pak buď $f(m) = 0$ a máme řešení, nebo $f(m)f(a_n) > 0$. To musí platit, neboť uvážíme-li $p(x)$ přímku procházející body $[a, f(a)], [b, f(b)]$, pak z Věty 4 plyne

$$f(x) - p(x) = (x - a_n)(x - b_n)\frac{f''(\eta)}{2}$$

pro $x \in [a_n, b_n]$ a vhodné $\eta \in [a_n, b_n]$. Tato rovnost zároveň s podmínkou (3) dává $f(x) - p(x) \leq 0$, což konkrétně pro bod m dává $f(m) \leq 0$, neboť $p(m) = 0$. Je-li $f(m) = 0$ metodu zastavíme, jinak máme $f(m) < 0$ a tedy $a_{n+1} := m$ a dostáváme

$$a_n < a_{n+1} < b_{n+1} = b_n.$$

Pak pro $f(m) \neq 0$ vidíme, že podmínky (1), (2) i (3) platí i pro $n+1$ a indukcí pak pro všechna $i \geq n$. Proto $b_i = b_n$ a

$$a_{i+1} = \frac{b_n f(a_i) - a_i f(b_n)}{f(a_i) - f(b_n)}$$

pro všechna $i \geq n$. Dále posloupnost $\{a_i\}$ je zřejmě monotonní, rostoucí a navíc omezená hodnotou b_n. Tedy $\lim_{i\to\infty} a_i := \xi$ existuje. Víme

$$f(\xi) \leq 0, \;\; f(b_n) > 0, \;\; \xi = \frac{b_n f(\xi) - \xi f(b_n)}{f(\xi) - f(b_n)},$$

což nám dává

$$(\xi - b_n)f(\xi) = 0, \quad a\ tedy \quad f(\xi) = 0,$$

protože $f(\xi) \leq 0 < f(b_n)$, je $\xi < b_n$. Dostáváme, že $\{a_i\}$ konverguje ke kořenu funkce f. Díky již dokázanému můžeme nyní řád konvergence zkoumat pomocí vztahu

$$a_{i+1} = \Phi(a_i), \quad kde \quad \Phi(x) = \frac{b_n f(x) - x f(b_n)}{f(x) - f(b_n)}.$$

Pak, protože $f(\xi) = 0$, můžeme psát

$$\Phi'(\xi) = \frac{-\left(b_n f'(\xi) - f(b_n)\right) f(b_n) + \xi f(b_n) f'(\xi)}{f(b_n)^2} = 1 - f'(\xi)\frac{\xi - b_n}{f(\xi) - f(b_n)}.$$

Z Věty o střední hodnotě 3 plyne, že existují η_1, η_2 tak, že

$$\frac{f(\xi) - f(b_n)}{\xi - b_n} = \frac{-f(b_n)}{\xi - b_n} = f'(\eta_1), \quad \xi < \eta_1 < b_n, \tag{2.2}$$

$$\frac{f(a_i) - f(\xi)}{a_i - \xi} = \frac{f(a_i)}{a_i - \xi} = f'(\eta_2), \quad a_i < \eta_2 < \xi. \tag{2.3}$$

Neboť $f''(x) > 0$, je $f'(x)$ ryze rostoucí na intervalu $[a_i, b_n]$ a tedy $f'(\eta_2) < f'(\xi) < f'(\eta_1)$. Protože $a_i < \xi$ a zároveň $f(a_i) < 0$, plyne z podmínky (2.3) $0 < f'(\eta_2)$. Proto $0 < f'(\xi) < f'(\eta_1)$ a dostáváme

$$0 < \Phi'(\xi) = 1 - \frac{f'(\xi)}{f'(\eta_1)} < 1.$$

Metoda regula falsi tedy konverguje lineárně podle Věty 9. □

Z předpokladů vidíme, že věta půjde aplikovat na poměrně širokou třídu funkcí. Máme-li dostatečně hladkou funkci a pokud tato funkce nemá v kořeni nulovou druhou derivaci, pak vždy najdeme okolí kořene, na kterém bude druhá derivace kladná respektive záporná. Od nějakého n pak bude metoda pro takové funkce konvergovat jednostranně a tedy dle předchozí věty lineárně. Samozřejmě ale můžeme najít i funkce, pro které bude konvergence oboustranná.

Kapitola 3

Newtonova metoda a její modifikace

Newtonova metoda je aktuálně jednou z nejvýznamnějších metod pro řešení nelineárních rovnic. Zároveň je také nejrychleji konvergující metodou z těch, kterými se v této práci zabýváme. V této kapitole se proto velmi podrobně podíváme na její vlastnosti a praktické použití a uvedeme i některé modifikace pro konkrétní případy.

3.1 Newtonova metoda

Nejprve se podíváme na klasickou Newtonovu metodu, odvodíme algoritmus výpočtu, podmínky nutné pro konvergenci a dokážeme, že metoda za jistých podmínek konverguje kvadraticky.

3.1.1 Algoritmus

Mějme spojitě diferencovatelnou funkci f, x^* přesné řešení rovnice (1.1) a x_0 počáteční odhad řešení. Uvažme Taylorův rozvoj funkce f v bodě x^*:

$$f(x^*) \approx f(x_0) + f'(x_0)(x^* - x_0).$$

Protože $f(x^*) = 0$, dostaneme pro $f'(x_0) \neq 0$

$$x^* \approx x_0 - \frac{f(x_0)}{f'(x_0)}. \tag{3.1}$$

Odvodili jsme iterační formuli pro Newtonovu metodu

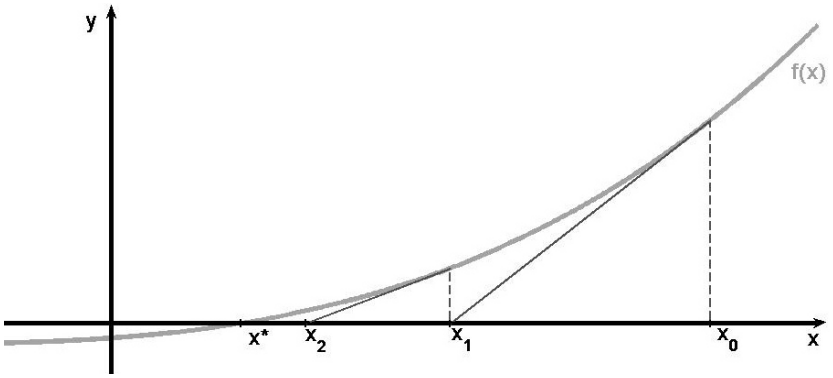
$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}. \tag{3.2}$$

Poznámka. Další možností, jak iterační formuli odvodit, je využít základní větu kalkulu a vhodnou aproximaci integrálu

$$f(x^*) = f(x_0) - \int_{x_0}^{x^*} f'(z)\,\mathrm{d}z \approx f(x_0) + f'(x_0)(x^* - x_0).$$

Úpravou pro $f'(x_0) \neq 0$ a $f(x^*) = 0$ dostaneme opět (3.1).

Vzorec (3.2) lze snadno interpretovat geometricky, jak můžeme vidět na Obrázku 3.1. Funkci f v každé iteraci v bodě x_n nahradíme tečnou ke grafu a kořen tečny pak zvolíme za novou aproximaci kořene.



Obrázek 3.1: Ilustrace Newtonovy metody.

Poznámka. Newtonova metoda využívá mnohem více informací o funkci f než například bisekce, je-li f lineární, pak najde kořen už v první iteraci, což plyne přímo z geometrického náhledu.

Než budeme algoritmus implementovat, zbývá vyřešit otázku zastavovacího kritéria. Pro Newtonovu metodu je typickou volbou $|f(x_n)| < tol_1$ nebo $|x_n - x_{n-1}| < tol_2$, pro zadaná tol_1, tol_2. Za předpokladu, že metoda konverguje kvadraticky (podmínky konvergence dokážeme později), nám druhá z možností dává dobrou informaci o $|x_n - x^*|$, neboť platí

$$|x_n - x^*| \leq |x_n - x_{n+1}| + |x_{n+1} - x^*| = |x_n - x_{n+1}| + O(|x_n - x^*|^2),$$

viz [11]. Dále kromě omezení maximálního počtu iterací je vhodné omezit i maximální možnou hodnotu $|x_n|$, abychom se vyvarovali případů, kdy nový bod v nějaké iteraci dostaneme velmi daleko od vstupního odhadu.

3.1.2 Implementace

Předpokládejme, že máme na vstupu zadanou spojitě diferencovatelnou funkci f, vstupní odhad kořene x_0 a požadovanou toleranci tol. Ve zjednodušené syntaxi pak může algoritmus vypadat takto

while $(|f(x_0)| > tol)$ **do**
  $x_0 := x_0 - f(x_0)/f'(x_0)$;
end

Prvním zjevným problémem je, že během výpočtu dělíme derivací funkce, tedy tato musí nutně být nenulová. Dostaneme-li v nějakém kroku $f'(x_n) = 0$, metoda není definovaná a algoritmus skočí chybou. Stejně tak může nastat problém, je-li $f'(x_n)$ v absolutní hodnotě kladná ale velmi malá, neboť může docházet k významným numerickým chybám ve výpočtu.

Poznámka. Důvod, proč je nulovost derivace problém, můžeme nahlédnout i geometricky. Má-li funkce f v nějakém bodě x_n nulovou derivaci, pak tečna ke grafu

f v tomto bodě je rovnoběžná s osou x a tedy se s ní nikdy neprotne. Pak bod x_{n+1} není definován.

Důležitou otázkou implementace algoritmu je samotný výpočet derivace. Máme-li vzorec pro výpočet derivace, pak jej můžeme zadat přímo na vstupu. Neznáme-li explicitní vzorec, například pokud je f daná výstupem z nějakého algoritmu, můžeme využít automatické diferencování, tj. techniku numerického vyhodnocování derivace funkce, která je zadaná počítačovým algoritmem. Hodnota derivace může být pomocí automatického diferencování vypočítána se stejnou přesností a při použití srovnatelného množství operací, jako při vyhodnocování funkce f [14]. Nemáme-li k dispozici ani automatické diferencování, pak můžeme využít metody konečných diferencí pro aproximaci derivace, kterou rozebereme podrobněji v jedné z následujících sekcí.

3.1.3 Konvergence

Rychlost konvergence je asi nejvýznamnější vlastností Newtonovy metody. Zajímavým pozorováním je, že na algoritmus lze pohlížet jako na fixed-point iteraci, kde funkci g volíme

$$g(x) = x - \frac{f(x)}{f'(x)}.$$

Iterační formule pak má tvar $x_{n+1} = g(x_n)$. S využitím tohoto pozorování nyní na základě poznámek z přednášky [12] a knihy [7] dokážeme řád konvergence Newtonovy metody.

Věta 11 (lokální konvergence Newtonovy metody)**.** *Nechť f je dvakrát spojitě diferencovatelná funkce na okolí x^* splňujícím $f(x^*) = 0$ a nechť $f'(x^*) \neq 0$. Pak existuje $\epsilon > 0$ takové, že posloupnost $\{x_n\}$ daná vzorcem (3.2) konverguje kvadraticky pro libovolné $x_0 \in [x^* - \epsilon, x^* + \epsilon]$.*

Důkaz. Označme $g(x) = x - f(x)/f'(x)$, pak $x_{n+1} = g(x_n)\ \forall n \in \mathbb{N}$.
Nejprve dokažme konvergenci algoritmu. Platí

$$g'(x) = 1 - \frac{f'(x)^2 - f(x)f''(x)}{f'(x)^2} = \frac{f(x)f''(x)}{f'(x)^2}.$$

Protože f je dvakrát spojitě diferencovatelná na okolí x^*, $f'(x^*) \neq 0$ a $f(x^*) = 0$, je $g'(\xi) = 0$. Pak podle Věty 8 existuje $\epsilon > 0$ takové, že posloupnost $\{x_n\}$ konverguje k x^* pro všechna $x_0 \in [x^* - \epsilon, x^* + \epsilon]$. Nyní dokažme, že metoda konverguje kvadraticky. Zvolme $x_0 \in [x^* - \epsilon, x^* + \epsilon]$. Pak posloupnost $\{x_n\}$ splňuje

$$f(x^*) = f(x_n) + f'(x_n)(x^* - x_n) + \frac{1}{2}f''(\eta_n)(x^* - x_n)^2, \text{ pro nějaké } \eta_n \in [x^*, x_n].$$

Neboť $f'(x_n) \neq 0$ a $f(x^*) = 0$, můžeme psát

$$0 = \frac{f(x_n)}{f'(x_n)} + (x^* - x_n) + \frac{1}{2}\frac{f''(\eta_n)}{f'(x_n)}(x^* - x_n)^2.$$

Dosazením ze vzorce (3.2) a převedením na levou stranu dostaneme

$$x_{n+1} - x^* = \frac{1}{2}\frac{f''(\eta_n)}{f'(x_n)}(x^* - x_n)^2.$$

Označíme-li $e_n = x_n - x^*$ chybu v n-té iteraci, dostaneme chybovou rovnici

$$e_{n+1} = \frac{1}{2}\frac{f''(\eta_n)}{f'(x_n)}e_n^2,$$

a odtud již zřejmě

$$\lim_{n\to\infty}\frac{|e_{n+1}|}{|e_n|^2} = \lim_{n\to\infty}\left|\frac{1}{2}\frac{f''(\eta_n)}{f'(x_n)}\right| = \frac{1}{2}\left|\frac{f''(x^*)}{f'(x^*)}\right| \in \mathbb{R}.$$

Existence limity plyne z již dokázané konvergence posloupnosti $\{x_n\}$ k x^*. □

Předchozí věta nám tedy dává jistotu kvadratické konvergence, pokud vstupní odhad kořene bude dostatečně blízko skutečnému kořeni. Problémem je, že dopředu nevíme, co pro danou funkci znamená dostatečně blízko. Uvedeme si tedy ještě bez důkazu jednu větu, která nám při splnění více požadavků dá jistotu konvergence pro libovolné x_0 z předem známého intervalu.

Věta 12. [4] *Nechť f je dvakrát spojitě diferencovatelná na neprázdném omezeném intervalu $[a, b]$ a nechť jsou splněny následující podmínky*

(1) $f(a)f(b) < 0$,
(2) $f'(x) \neq 0\ \forall x \in [a, b]$,
(3) $f''(x) \geq 0$ *nebo* $f''(x) \leq 0\ \forall x \in [a, b]$,
(4) $\frac{|f(a)|}{|f'(a)|} < b - a$ *a zároveň* $\frac{|f(b)|}{|f'(b)|} < b - a$.

Pak f má v intervalu $[a, b]$ právě jeden kořen a Newtonova metoda konverguje k tomuto kořeni pro libovolné $x_0 \in [a, b]$.

3.1.4 Vlastnosti

Nejvýznamnější vlastností Newtonovy metody je kvadratická konvergence, která je však pouze lokální. Při špatném počátečním odhadu kořene se může metoda chovat nevyzpytatelně, například velmi rychle divergovat. Může však také dojít k periodickému zacyklení, či ke konvergenci k jinému kořeni, než jsme očekávali, jak si ukážeme v Příkladu 8. V sofistikovanějších algoritmech se proto často používá v kombinaci s nějakou robustnější metodou, která v případě špatného chování zajistí konvergenci algoritmu.

Nevýhodou Newtonovy je její omezená aplikovatelnost, neboť potřebujeme znát derivaci funkce f a tedy na rozdíl od bisekce metoda není vhodná pro nediferencovatelné funkce. S tím souvisí i výpočetní náročnost jednotlivých iterací. V každé iteraci musíme provést jak vyhodnocení funkce f tak vyhodnocení její derivace, případně vypočítat aproximaci derivace v daném bodě. Tedy jeden výpočetní krok je dvakrát dražší, než například u bisekce.

Newtonovu metodu lze úspěšně rozšířit i na funkce více proměnných, viz [13], nebo [6].

3.2 Newtonova metoda pro vícenásobné kořeny

Z předpokladů vět v části o konvergenci je zřejmé, že je nelze aplikovat na funkce s vícenásobnými kořeny, neboť pro takové funkce platí $f'(x^*) = 0$. Na

základě [15] dokážeme obecnější větu o řádu konvergence, která nám navíc dá motivaci pro modifikaci algoritmu Newtonovy metody.

Věta 13 (řád konvergence Newtonovy metody pro vícenásobné kořeny)**.** *Nechť $m \in \mathbb{N}$, $m > 1$ a funkce f je spojitě diferencovatelná až do řádu $m+1$ na okolí U bodu x^*. Nechť x^* splňuje $f(x^*) = f'(x^*) = ... = f^{(m-1)}(x^*) = 0$, $f^{(m)}(x^*) \neq 0$ a posloupnost $\{x_n\}$ generovaná Newtonovou metodou konverguje k x^* pro nějaké $x_0 \in U$. Pak metoda konverguje lineárně s konstantou $\frac{m-1}{m}$.*

Důkaz. Označme $e_n = x_n - x^*$ a $\eta_n = -\frac{f(x_n)}{f'(x_n)}$ pro $n \in \mathbb{N}$. Pak zřejmě $e_{n+1} = e_n + \eta_n$. Uvažme Taylorovy rozvoje f a f'

$$f(x_n) = f(x^*) + f'(x^*)e_n + ... + \frac{1}{(m-1)!}f^{(m-1)}(x^*)e_n^{m-1} + \frac{1}{m!}f^{(m)}(x^* + \theta_1 e_n)e_n^m,$$

$$f'(x_n) = f'(x^*) + ... + \frac{1}{(m-2)!}f^{(m-1)}(x^*)e_n^{m-2} + \frac{1}{(m-1)!}f^{(m)}(x^* + \theta_2 e_n)e_n^{m-1}$$

pro nějaká $\theta_1, \theta_2 \in (0,1)$. Z předpokladu $f(x^*) = ... = f^{(m-1)}(x^*) = 0$ dostáváme

$$\eta_n = \frac{\frac{1}{m!}f^{(m)}(x^* + \theta_1 e_n)e_n^m}{\frac{1}{(m-1)!}f^{(m)}(x^* + \theta_2 e_n)e_n^{m-1}}.$$

Dále z Věty o střední hodnotě 3

$$f^{(m)}(x^* + \theta_1 e_n) = f^{(m)}(x^* + \theta_2 e_n) + f^{(m+1)}(x^* + \theta_3(\theta_1 - \theta_2)e_n)(\theta_1 - \theta_2)e_n$$

pro nějaké $\theta_3 \in [0,1]$. Odtud již

$$\eta_n = -\frac{1}{m}e_n - \left[\frac{f^{(m+1)}(x^* + \theta_3(\theta_1 - \theta_2)e_n)(\theta_1 - \theta_2)}{mf^{(m)}(x^* - \theta_2 e_n)}\right]e_n^2,$$

tudíž platí

$$e_{n+1} = \frac{m-1}{m}e_n + O(e_n^2)$$

a tedy metoda konverguje lineárně s konstantou $\frac{m-1}{m} < 1$. □

Poznámka. Pro $m = 1$ bychom z důkazu dostali, že Newtonova metoda pro jednoduché kořeny konverguje kvadraticky.

Důkaz předchozí věty nám dává motivaci pro modifikaci vzorce (3.2) pro funkci f s $m-$násobným kořenem

$$x_{n+1} = x_n - m\frac{f(x_n)}{f'(x_n)}. \tag{3.3}$$

Pak mírnou úpravou důkazu dostaneme, že konvergence modifikovaného Newtonova algoritmu daného vzorcem (3.3) je kvadratická.

Poznámka. Modifikovaná Newtonova metoda pro vícenásobné kořeny předpokládá, že násobnost kořene známe. Tuto informaci však v praxi často nemáme. V případě, že do vzorce dosadíme špatnou násobnost, může metoda rychle divergovat či konvergovat pomaleji, než standardní Newtonova metoda, viz Příklad 6.

3.3 Newtonova metoda s konečnými diferencemi

Ne vždy budeme mít pro zadanou funkci f k dispozici vzorec pro výpočet její derivace, neboť tuto funkci nemusíme mít zadanou explicitním vzorcem, ale výstupem z jiného výpočtu, měření a podobně. Podívejme se tedy na možnost aproximace derivace pomocí konečných diferencí. Uvažujme aproximaci

$$f'(x) \approx \frac{f(x+h)-f(x)}{h}, \text{ pro } h > 0 \text{ malé.} \tag{3.4}$$

Pak iterační schéma může mít tvar

$$x_{n+1} = x_n - \frac{f(x_n)}{a_n} \quad \text{pro} \quad a_n = \frac{f(x_n+h_n)-f(x_n)}{h_n}. \tag{3.5}$$

Výhodou této aproximace je, že vyhodnocujeme jenom funkci f, avšak potřebujeme ji v každém kroku vyhodnotit ve dvou různých bodech, což může být výpočetně náročné, je-li vyhodnocování funkce f drahé.

Otázkou je, jak vhodně zvolit h_n. Je zřejmé, že čím menší h_n budeme volit, tím lépe budeme aproximovat derivaci a tím rychleji by teoreticky měla metoda konvergovat, neboť limitně pro h jdoucí k nule dostaneme kvadraticky konvergující Newtonovu metodu. To však nemusí být pravda, neboť je nutné uvážit vliv aritmetiky s konečnou přesností. Je-li například $f(x_n) \neq 0$ a zvolíme $|h_n| < |x_n|\,\epsilon_{mach}$, kde ϵ_{mach} je strojová přesnost, pak numericky dostaneme $f(x_n+h_n) = f(x_n)$ a tedy x_{n+1} nebude definováno. Zvolíme-li h_n dostatečně velké tak, aby $f(x_n+h_n) \neq f(x_n)$, stále může docházet k velké ztrátě přesnosti. Více informací k této problematice je možné nalézt v knize [5]. Pro zajímavost však ještě uveďme, že rozumnou volbou pro h_n je

$$|h_n| = \sqrt{\epsilon_{mach}}\, max\{|x_n|, x_{typ}\},$$

kde x_{typ} je hodnota zadaná uživatelem.

Nakonec se podívejme na rychlost konvergence schématu (3.5). Uvedeme si pro informaci větu, která nám v závislosti na volbě h_n zaručí lokální lineární, superlineární či dokonce kvadratickou konvergenci. Důkaz si zde uvádět nebudeme. Pro specifickou volbu h_n dokážeme superlineární konvergenci v sekci o metodě sečen.

Věta 14 (konvergence Newtonovy metody s konečnými diferencemi [5])**.** *Nechť $a, b \in \mathbb{R}, a < b$, $f : (a,b) \to \mathbb{R}$ je Lipschitzovsky spojitá s konstantou γ na intervalu (a,b) a nechť existuje $\rho > 0$ takové, že $|f'(x)| \geq \rho\ \forall x \in (a,b)$. Je-li $x^* \in (a,b)$ takové, že $f(x^*) = 0$, pak existují kladné konstanty η_1, η_2 takové, že pro posloupnost $\{h_n\}$, $h_n \in \mathbb{R}\ \forall n \in \mathbb{N}$ splňující $0 < |h_n| \leq \eta_2$ a $|x^* - x_0| < \eta_1$ posloupnost $\{x_n\}$ definovaná*

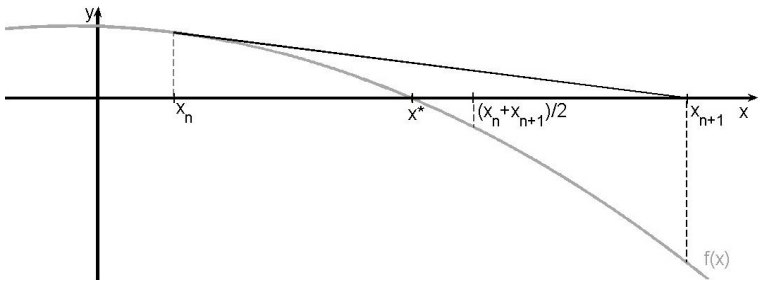
$$x_{n+1} = x_n - \frac{f(x_n)}{a_n}, \quad a_n = \frac{f(x_n+h_n)-f(x_n)}{h_n}$$

je dobře definovaná a konverguje lineárně k x^*. Je-li navíc $\lim_{n\to\infty} h_n = 0$, pak je konvergence superlineární. Existuje-li konstanta c splňující $|h_n| \leq c\,|x_n - x^*|$, pak je konvergence kvadratická.

3.4 Backtracking pro Newtonovu metodu

U vlastností Newtonovy metody jsme se zmínili, že při výpočtu někdy může nastat problém s rychlou divergencí, zacyklením nebo konvergencí k jinému kořenu, než jsme očekávali. Toto chování je demonstrováno na Příkladech 5 a 8. Možností modifikace standardní Newtonovy metody pro tento případ je více, my si nyní uvedeme jednu z nich.

Mějme opět spojitě diferencovatelnou funkci f a posloupnost $\{x_n\}$ danou vzorcem (3.2) pro nějaké x_0. Na vzorec pro výpočet x_{n+1} se lze dívat i tak, že nám dává informaci, v jakém směru od x_n leží bod x_{n+1}. Ačkoli může platit $|f(x_{n+1})| > |f(x_n)|$, bod x_{n+1} leží vždy ve směru, ve kterém funkce f v bodě x_n klesá (předpokládáme $f'(x_n) \neq 0$, jinak by x_{n+1} nebyl definován). Tedy existuje bod $z \in [x_{n+1}, x_n]$ splňující $|f(z)| < |f(x_n)|$. Nejlépe tento argument nahlédneme graficky z Obrázku 3.2, podrobněji viz [5].



Obrázek 3.2: Ilustrace backtrackingu.

Dostáváme návod pro modifikaci Newtonova algoritmu. Pokud během výpočtu pro nějaké $n \in \mathbb{N} \cup \{0\}$ splňuje x_{n+1} podmínku $|f(x_{n+1})| \geq |f(x_n)|$, nahradíme bod x_{n+1} bodem $z \in [x_{n+1}, x_n]$ splňujícím $|f(z)| < |f(x_n)|$. Možností, jak bod z najít, je například položit $z = \frac{x_n + x_{n+1}}{2}$. Pak buď $|f(z)| < |f(x_n)|$, nebo položíme $z = \frac{z + x_{n+1}}{2}$ a proces opakujeme dokud z nesplňuje $|f(z)| < |f(x_n)|$. Tomuto postupu se říká backtracking. Dále již položíme $x_{n+1} = z$ a provedeme další krok Newtonovy metody. Ve zjednodušené syntaxi můžeme pro zadanou funkci f, bod x_0 a požadovanou přesnost *tol* zapsat algoritmus následovně

while $(|f(x_n)| > tol)$ **do**
  $x_{n+1} := x_n - f(x_n)/f'(x_n)$;
  while $|f(x_{n+1})| \geq |f(x_n)|$ **do**
    $x_{n+1} = (x_{n+1} + x_n)/2$;
  end
  n=n+1;
end

Poznámka. Výše popsaná metoda je speciálním případem tzv. line search metod, kdy řešení hledáme pomocí řešení jednorozměrného problému, v tomto případě na úsečce $(x_n, x_n - f(x_n)/f'(x_n))$ [10].

Uvedený algoritmus funguje ve většině případů dobře, avšak v některých případech může způsobit i oscilaci kolem kořene. Na Příkladu 5 je demonstrováno dobré chování algoritmu, tj. zamezení divergenci či urychlení výpočtu. Výhodou algoritmu je, že pokud konverguje, pak je konvergence asymptoticky stejná, jako

u Newtonovy metody. Zpočátku metoda zajistí, aby nedošlo k rychlé divergenci a pokud se dostaneme dostatečně blízko ke kořeni, nezkazí kvadratickou konvergenci standardní Newtonovy metody [10].

3.5 Metoda sečen

V sekci o Newtonově metodě jsme odvodili lokálně kvadraticky konvergující algoritmus, který je však použitelný pouze pro diferencovatelné funkce. Naším cílem je nyní odvodit algoritmus, který bude konvergovat rychleji než lineárně a bude vhodný i pro nediferencovatelné funkce.

3.5.1 Algoritmus a implementace

Při výpočtu konečných diferencí u Newtonovy metody jsme se zmínili, že vyhodnocování funkce f dvakrát v každé iteraci se může výpočet prodražit. Pokusíme se proto algoritmus modifikovat tak, aby víc využíval již známé informace. Budeme přitom vycházet z knih [13] a [17]. Uvažme iterační schéma (3.5)

$$x_{n+1} = x_n - \frac{f(x_n)}{a_n}, \quad \text{pro} \quad a_n = \frac{f(x_n + h_n) - f(x_n)}{h_n}$$

a položme

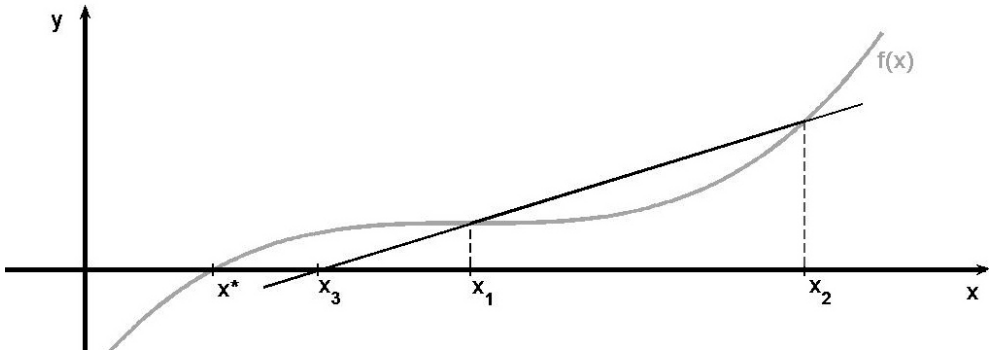
$$h_n = x_{n-1} - x_n.$$

Dosazením a úpravou dostaneme iterační formuli pro metodu sečen

$$x_{n+1} = x_n - f(x_n)\frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})}.$$

Pro inicializaci metody budeme zřejmě potřebovat dva počáteční body x_0 a x_1.

Volbu nového bodu můžeme nahlédnout opět geometricky, viz Obrázek 3.3. Bod x_{n+1} získáme jako průsečík osy x s přímkou vedenou body $[x_{n-1}, f(x_{n-1})]$, $[x_n, f(x_n)]$, tj. se sečnou grafu funkce f.



Obrázek 3.3: Ilustrace metody sečen.

Na sečnu vedenou body $[x_{n-1}, f(x_{n-1})]$, $[x_n, f(x_n)]$ můžeme pohlížet jako na lineární interpolaci funkce f mezi body x_{n-1} *a* x_n.

Možnosti pro volbu zastavovacího kritéria jsou podobné jako u Newtonovy metody. Typickou volbou jsou $|f(x_n)| < tol_1$ a $|x_n - x_{n-1}| < tol_2$, pro zadaná tol_1, tol_2. Druhé uvedené kritérium je pro metodu sečen velmi důležité. Je-li hodnota $|x_n - x_{n-1}|$ velmi malá, pak pro spojitou f je i hodnota $|f(x_n) - f(x_{n-1})|$ malá a při dělení touto hodnotou by mohlo docházet k velkým zaokrouhlovacím chybám [4].

Předpokládejme, že máme na vstupu zadanou funkci f, počáteční body x_0, x_1 a požadovanou přesnost tol. Pak algoritmus může vypadat takto

while $(|f(x_n)| > tol)$ **do**
  $x_{n+1} := x_n - f(x_n)\frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})}$;
  $n := n + 1$;
end

Výraz pro výpočet x_{n+1} lze ekvivalentně zapsat $\frac{f(x_n)x_{n-1} - f(x_{n-1})x_n}{f(x_n) - f(x_{n-1})}$. Možnost implementovaná v algoritmu výše však vede během výpočtu k menším zaokrouhlovacím chybám. V obou případech dělíme výrazem $f(x_n) - f(x_{n-1})$, není-li f prostá, tento výraz může být nulový a nový bod pak není vůbec definován.

3.5.2 Řád konvergence

Cílem této sekce bylo na základě Newtonovy metody odvodit algoritmus konvergující rychleji, než lineárně. V následující větě si dokážeme, že jsme cíl splnili. Budeme vycházet z poznámek k přednášce [3] a článku [9].

Věta 15 (lokální konvergence metody sečen)**.** *Nechť funkce f je dvakrát spojitě diferencovatelná na okolí U bodu x^* splňujícím $f(x^*) = 0$ a zároveň $f'(x^*) \neq 0$. Pak existuje $\epsilon > 0$ takové, že posloupnost $\{x_n\}$ generovaná metodou sečen konverguje s řádem $r = \frac{1+\sqrt{5}}{2} \approx 1.618$ pro libovolná $x_0, x_1 \in [x^* - \epsilon, x^* + \epsilon]$.*

Důkaz. Označme $e_n = x_n - x^*$. Pak platí

$$e_{n+1} = x_{n+1} - x^* = \frac{f(x_n)x_{n-1} - f(x_{n-1})x_n}{f(x_n) - f(x_{n-1})} - x^* = \frac{e_{n-1}f(x_n) - e_n f(x_{n-1})}{f(x_n) - f(x_{n-1})}$$
$$= e_{n-1}e_n \left[\frac{\frac{f(x_n)}{e_n} - \frac{f(x_{n-1})}{e_{n-1}}}{f(x_n) - f(x_{n-1})} \right] =: e_{n-1}e_n A_n.$$

Označme $F(x) = \frac{f(x)}{x - x^*} = \frac{f(x) - f(x^*)}{x - x^*}$. Pak $A_n = \frac{F(x_n) - F(x_{n-1})}{f(x_n) - f(x_{n-1})}$. Z Věty o střední hodnotě 3 dostáváme

$$F(x_n) - F(x_{n-1}) = F'(\xi_n)(x_n - x_{n-1}) \text{ pro nějaké } \xi_n \in [x_n, x_{n-1}].$$

Dále z Taylorova rozvoje plyne $f(x^*) = f(x) + f'(x)(x^* - x) + f''(\eta)(x^* - x)^2$ pro nějaké $\eta \in [x^*, x]$, a proto

$$F'(x) = \frac{f'(x)(x - x^*) - f(x) + f(x^*)}{(x - x^*)^2} = \frac{1}{2}f''(\eta).$$

Odtud již plyne

$$F(x_n) - F(x_{n-1}) = \frac{1}{2}f''(\eta_n)(x_n - x_{n-1}) \text{ pro nějaké } \eta_n \in [x^*, \xi_n].$$

Dosazením a použitím Věty o střední hodnotě 3 dostaneme

$$A_n = \frac{1}{2} f''(\eta_n) \frac{(x_n - x_{n-1})}{f(x_n) - f(x_{n-1})} = \frac{1}{2} \frac{f''(\eta_n)}{f'(\theta_n)} \quad \text{pro nějaké } \theta_n \in [x_n, x_{n-1}].$$

Chybová rovnice má tvar

$$e_{n+1} = \frac{1}{2} \frac{f''(\eta_n)}{f'(\theta_n)} e_n e_{n-1}.$$

Neboť $f'(x^*) \neq 0$ a f'' je spojitá a tedy speciálně omezená na nějakém okolí U bodu x^*, existuje konstanta $M > 0$ taková, že $\left| \frac{1}{2} \frac{f''(\eta_n)}{f'(\theta_n)} \right| \leq M$ na U. Pak pro $x_0, x_1 \in U$ splňující $|e_0| < \frac{1}{M+1}, |e_1| < \frac{1}{M+1}$ plyne indukcí pro $n > 1$ z chybové rovnice $|e_{n+1}| < \frac{M}{M+1} |e_n|$, a tedy metoda konverguje.
Zbývá dokázat řád konvergence. Hledáme $p > 0$ splňující $\lim_{n\to\infty} \frac{|e_{n+1}|}{|e_n|^p} = C$, pro nějakou kladnou reálnou konstantu C. Označíme-li $E_n = \frac{|e_{n+1}|}{|e_n|^p}$, pak $\lim_{n\to\infty} E_n = C$ a platí

$$|e_{n+1}| = E_n |e_n|^p = E_n (E_{n-1} |e_{n-1}|^p)^p = E_n E_{n-1}^p |e_{n-1}|^{p^2}.$$

Odtud úpravou dostaneme

$$\frac{|e_{n+1}|}{|e_n| |e_{n-1}|} = E_n E_{n-1}^{p-1} |e_{n-1}|^{p^2-p-1}$$

a limitním přechodem na obou stranách dostáváme

$$\frac{1}{2} \frac{|f''(x^*)|}{|f'(x^*)|} = C C^{p-1} \lim_{n\to\infty} |e_{n-1}|^{p^2-p-1}.$$

Neboť výraz nalevo je obecně nenulový a $\lim_{n\to\infty} |e_{n-1}| = 0$, musí platit $p^2 - p - 1 = 0$. Jediný kladný kořen tohoto polynomu je $\frac{1+\sqrt{5}}{2}$. □

3.5.3 Vlastnosti

Výhodou metody sečen je její použitelnost pro nediferencovatelné funkce. Ačkoli její konvergence je pomalejší, než u Newtonovy metody, může být často její použití výhodnější. V n-té iteraci metody sečen potřebujeme pouze vyhodnocení funkce $f(x_n)$, neboť hodnotu $f(x_{n-1})$ již známe z předchozí iterace. U Newtonovy metody musíme vyhodnotit $f(x_n)$ i $f'(x_n)$. Je-li cena vyhodnocení funkce i její derivace přibližně stejná, pak jedna iterace Newtonovy metody je dvakrát dražší než iterace metody sečen.

Stejně jako u Newtonovy metody je konvergence metody sečen pouze lokální. Při špatně zvoleném počátečním odhadu může metoda velmi rychle divergovat. Tento problém lze v některých případech vyřešit aplikací backtrackingu, podobně jako u Newtonovy metody.

Nevýhodou metody sečen je, že ji nelze snadno modifikovat pro vícenásobné kořeny. Lze dokázat, že její konvergence je pro vícenásobné kořeny lineární a to s rychlostí odpovídající reálnému řešení z intervalu $(0, 1)$ rovnice $r^m + r^{m-1} - 1 = 0$, kde m je násobnost hledaného kořene [1].

3.6 Metody založené na kvadratické interpolaci

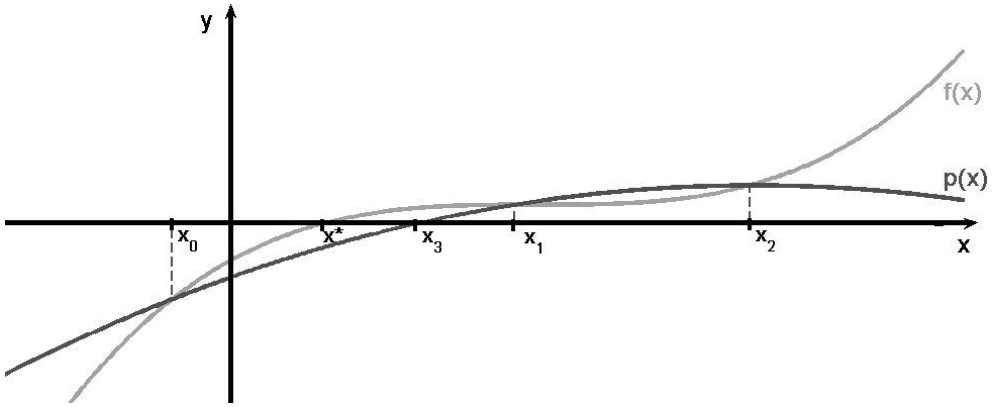
V části o metodě sečen jsme odvodili superlineárně konvergující algoritmus, který na základě dvou vstupních bodů definoval nový bod pomocí sečny grafu funkce. Přirozenou otázkou je, zda bychom mohli dosáhnout lepšího výsledku, pokud bychom využili většího množství vstupních bodů. Nyní si tedy ukážeme dvě myšlenkově velmi podobné metody, které na tuto otázku odpovídají.

3.6.1 Mullerova metoda

Mějme zadanou funkci f pro řešení rovnice (1.1) a body x_0, x_1, x_2. Uvažme polynom p stupně 2, který jako funkci proměnné x proložíme body $[x_0, f(x_0)]$, $[x_1, f(x_1)]$, $[x_2, f(x_2)]$. Využijeme-li vzorec pro Lagrangeův interpolační polynom 4, dostaneme

$$p(x) = f(x_0)\frac{(x-x_1)(x-x_2)}{(x_0-x_1)(x_0-x_2)} + f(x_1)\frac{(x-x_0)(x-x_2)}{(x_1-x_0)(x_1-x_2)} + f(x_2)\frac{(x-x_0)(x-x_1)}{(x_2-x_1)(x_2-x_0)}.$$

Úpravou a dosazením do vzorce pro výpočet kořenů kvadratické rovnice získáme body x_a, x_b splňující $p(x_a) = p(x_b) = 0$. Pak buď jeden z těchto bodů je přímo řešením zadané rovnice a algoritmus končí, nebo za nový bod x_3 zvolíme ten z bodů x_a, x_b, jehož vzdálenost od bodu x_2 je menší, a provedeme další iteraci pro body x_1, x_2, x_3. Tento algoritmus se nazývá Mullerova metoda. První iterace je pro názornost zobrazena v Obrázku 3.4.



Obrázek 3.4: Ilustrace Mullerovy metody.

Poznámka. Pro výpočet koeficientů polynomu p můžeme také využít numerický software, který má již předinstalovanou funkci pro prokládání bodů polynomem požadovaného stupně, v Matlabu je to například funkce "polyfit".

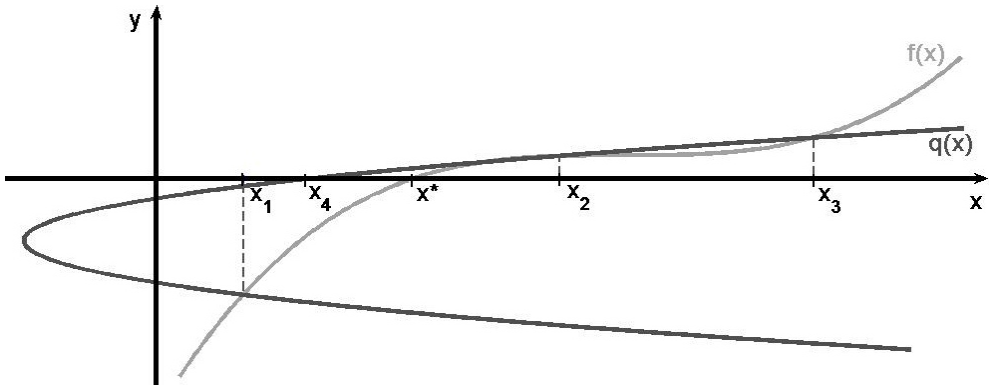
V průběhu výpočtu se může stát, že body, kterými prokládáme polynom, leží na jedné přímce a tedy je nelze proložit polynomem stupně 2. V takovém případě nový bod dostaneme jako průsečík přímky, na které tyto body leží, s osou x. Je-li tato přímka rovnoběžná s osou x, pak nový bod definován není a algoritmus končí chybou.

Zajímavým problémem algoritmu je, že polynom p nemusí mít reálné kořeny, ačkoli hledané řešení rovnice je reálné. Graf funkce p - parabola v takovém případě

leží buď celá nad nebo pod osou x a jejími kořeny jsou dvě komplexně sdružená čísla. Komplexní čísla se navíc ve výpočtu kořenů polynomu p mohou objevit také vlivem zaokrouhlovacích chyb. V takových případech je však většinou komplexní část přibližného řešení malá a je možné ji zanedbat [4]. Přesto se často budeme chtít vyhnout počítání s komplexními čísly, a proto odvodíme ještě jednu metodu.

3.6.2 Inverzní kvadratická interpolace

Mějme opět zadanou funkci f, body x_0, x_1, x_2 a uvažme q - polynom stupně 2, který je funkcí proměnné $y = f(x)$ a prochází body $[f(x_0), x_0]$, $[f(x_1), x_1]$, $[f(x_2), x_2]$. Pak q protíná osu x v jediném bodě $q(0)$. Položme tedy $x_3 = q(0)$. Algoritmus dále pokračuje stejně jako u Mullerovy metody. Buď x_3 je řešením zadané rovnice a algoritmus končí, nebo provedeme další iteraci pro body x_1, x_2, x_3. Tento algoritmus se nazývá inverzní kvadratická interpolace. První iterace je znázorněna v Obrázku 3.5.



Obrázek 3.5: Ilustrace inverzní kvadratické interpolace.

Poznámka. Koeficienty inverzního interpolačního polynomu q získáme stejně jako u Mullerovy metody, pouze zaměníme roli proměnných x_i a $y_i = f(x_i)$.

Vyřešili jsme tedy problém Mullerovy metody s komplexními kořeny, neboť inverzní parabola, je-li definována, vždy protne osu x právě v jednom bodě a tedy výpočet zůstane po celou dobu v reálných číslech. Stejně jako u Mullerovy metody se během výpočtu může stát, že interpolační body leží na přímce. Za předpokladu, že tato přímka není rovnoběžná s osou x, dostaneme nový bod jako její průsečík s osou x. Není-li zadaná funkce f prostá, může nastat i situace, že například $f(x_n) = f(x_{n+1})$. Pak funkce q není definována a algoritmus končí chybou. Problém může nastat i v případě, že hodnota $|f(x_n) - f(x_{n+1})|$ je nenulová ale malá. Graf funkce q v tomto případě může protnout osu x v bodě velmi vzdáleném od přesného řešení a algoritmus pak může rychle divergovat [17]. Tento fakt je dobré zohlednit při volbě zastavovacích kritérií omezením maximální hodnoty $|x_n|$.

3.6.3 Konvergence

Věta 16 (lokální konvergence inverzní kvadratické interpolace)**.** *Nechť f je třikrát spojitě diferencovatelná na okolí bodu x^* splňujícím $f(x^*) = 0$ a $f'(x^*) \neq 0$. Pak metoda inverzní kvadratické interpolace konverguje lokálně s řádem r, který odpovídá reálnému řešení rovnice $p^3 - p^2 - p - 1 = 0$ (tj. $r \approx 1.8393$).*

Důkaz. Myšlenka důkazu je podobná jako u metody sečen, vzhledem k většímu počtu proměnných je však důkaz technicky náročnější. V průběhu proto nebudeme rozepisovat všechny detaily.

Nejprve odvodíme chybovou rovnici. Pro zkrácení zápisu označme $a = x_{n-2}, b = x_{n-1}$, $c = x_n$ a $f_a = f(x_{n-2})$, $f_b = f(x_{n-1})$, $f_c = f(x_n)$, a nechť q je inverzní interpolační polynom procházející body $[f_a, a]$, $[f_b, b]$, $[f_c, c]$. Pak z Věty 4 plyne

$$e_{n+1} = x_{n+1} - x^* = q(0) - f^{-1}(0) = -\frac{1}{6}(f^{-1})^{(3)}(\xi_n) f_a f_b f_c$$

pro nějaké $\xi_n \in [min\{f_a, f_b, f_c\}, max\{f_a, f_b, f_c\}]$. Odtud z Věty 3 dostaneme chybovou rovnici

$$\begin{aligned} e_{n+1} &= -\frac{1}{6}(f^{-1})^{(3)}(\xi_n)(f_a - f(x^*))(f_b - f(x^*))(f_c - f(x^*)) \\ &= -\frac{1}{6}(f^{-1})^{(3)}(\xi_n) f'(\eta_a) f'(\eta_b) f'(\eta_c)(a - x^*)(b - x^*)(c - x^*) \\ &= A_n e_{n-2} e_{n-1} e_n \end{aligned}$$

pro nějaká $\eta_a \in [x^*, a], \eta_b \in [x^*, b], \eta_c \in [x^*, c]$. Postupným derivováním funkce f^{-1} a úpravou dostaneme

$$(f^{-1})^{(3)}(y) = \frac{-f^{(3)}(f^{-1}(y)) f'(f^{-1}(y)) + 3[f''(f^{-1}(y))]^2}{[f'(f^{-1}(y))]^5}.$$

Protože f je z předpokladu na okolí x^* třikrát spojitě diferencovatelná a $f'(x^*) \neq 0$, existuje okolí bodu x^* na kterém je hodnota $(f^{-1})^{(3)}$ dobře definovaná a omezená. Tedy na nějakém okolí bodu x^* je i člen A_n omezený konstantou a stejně jako u metody sečen lze ukázat, že pro x_0, x_1, x_2 zvolené dostatečně blízko x^* metoda konverguje.

Zbývá odvodit řád konvergence. Hledáme $p > 0$ splňující $\lim_{n\to\infty} \frac{|e_{n+1}|}{|e_n|^p} = C$, pro nějakou kladnou reálnou konstantu C. Označíme-li $E_n = \frac{|e_{n+1}|}{|e_n|^p}$, pak $\lim_{n\to\infty} E_n = C$ a platí

$$|e_{n+1}| = E_n |e_n|^p = E_n (E_{n-1} |e_{n-1}|^p)^p = E_n E_{n-1}^p |e_{n-1}|^{p^2} = E_n E_{n-1}^p (E_{n-2} |e_{n-2}|^p)^{p^2}.$$

Dosazením a úpravou dostáváme

$$\frac{|e_{n+1}|}{|e_n|\,|e_{n-1}|\,|e_{n-2}|} = E_n E_{n-1}^{p-1} E_{n-2}^{p^2-p-1} |e_{n-2}|^{p^3-p^2-p-1}.$$

Pak limitním přechodem na obou stranách a uvážením, že člen nalevo je obecně nenulový a že člen $|e_{n-2}|$ jde limitně pro $n \to \infty$ k nule, dostaneme podmínku $p^3 - p^2 - p - 1 = 0$, která má jediné reálné řešení. □

Důkaz konvergence pro Mullerovu metodu zde uvádět nebudeme, poznamenejme pouze, že obě metody konvergují lokálně a mají stejný řád konvergence [7].

Poznámka. Lze ukázat, že použitím interpolace vyššího řádu než kvadratické se již řád konvergence příliš nezvýší a že všechny metody založené na jednoduché inverzní interpolaci mají řád konvergence menší než 2 [1].

3.6.4 Vlastnosti

Obě metody během výpočtu nevyužívají derivace funkce f a jsou tedy vhodné i pro nediferencovatelné funkce. Navíc v každé iteraci u obou metod potřebujeme pouze jedno nové vyhodnocení funkce f, neboť zbylé hodnoty potřebné k výpočtu známe z předchozí iterace.

Počítání v komplexních číslech u Mullerovy metody může být v některých případech výhodou. Uvažujeme-li funkci $f : \mathbb{C} \to \mathbb{C}$, pak Mullerova metoda dokáže při splnění jistých požadavků spočítat i komplexní kořeny této funkce a to dokonce i v případě, že kořeny jsou vícenásobné. V praxi se proto využívá pro počítání kořenů polynomů [4].

Pro Mullerovu metodu lze ukázat, že její konvergence pro dvojnásobné kořeny je lokálně superlineární s řádem 1.23, pro kořeny s násobností vyšší než 2 je pak lokálně lineární [1].

Inverzní kvadratická interpolace se v kombinaci s metodou sečen a metodou půlení intervalu používá v Brentově algoritmu, který je implementovaný například i v Matlabu ve funkci "fzero"určené pro hledání kořenů funkcí [17].

Kapitola 4

Numerické experimenty

V této kapitole se podíváme na numerické řešení konkrétních nelineárních rovnic. Srovnáme chování algoritmů na funkci splňující předpoklady konvergence pro dané metody řešení a ukážeme, jak rychle reálně metody konvergují. Podíváme se také jaký vliv má nesplnění předpokladů spojitosti derivace na konvergenci algoritmů a uvedeme některé další zajímavé příklady. Všechny výpočty budou prováděny pomocí softwaru Matlab v double aritmetice a příslušné zdrojové kódy jsou k nalezení v Příloze A.

Příklad 1. Nejprve se vrátíme k příkladu uvedenému u metody fixed-point iterace, viz 2.2.1. Máme funkci $f(x) = x^2 - x - 2$ s kořeny -1 a 2. Předpokládejme, že chceme vypočítat kořen $x^* = 2$ pomocí následujících 4 funkcí

(1) $g(x) = x^2 - 2$,
(2) $g(x) = \sqrt{x+2}$,
(3) $g(x) = 1 + 2/x$,
(4) $g(x) = \frac{x^2+2}{2x-1}$.

Zvolme počáteční odhad kořene $x_0 = 3$ a zastavovací kritérium $|g(x_n) - x_n| < 10^{-6}$. V Tabulce 4.1 jsou uvedeny hodnoty $\{x_n\}$, pro jednotlivé funkce a z nich vypočítaná velikost chyby $\{|e_n|\}$.

	x^2-2		$\sqrt{x+2}$		$1+\frac{2}{x}$		$\frac{x^2+2}{2x-1}$	
n	x_n	$\lvert e_n\rvert$	x_n	$\lvert e_n\rvert$	x_n	$\lvert e_n\rvert$	x_n	$\lvert e_n\rvert$
1	3	1	2.236067	0.236067	1.666666	0.333333	2.200000	0.200000
2	7	6	2.058171	0.058171	2.200000	0.200000	2.011764	0.011764
3	43	45	2.014490	0.014490	1.909090	0.090909	2.000045	0.000045
4	2207	2205	2.003619	0.003619	2.047619	0.047619	2.000000	0.000000
5			2.000904	0.000904	1.976744	0.023255		
6			2.000226	0.000226	2.011764	0.011764		
7			2.000056	0.000056	1.994152	0.005847		
8			2.000014	0.000014	2.002932	0.002932		
9			2.000003	0.000003	1.998535	0.001464		
10			2.000000	0.000000	2.000732	0.000732		
...					...	...		
20					2.000000	0.000000		

Tabulka 4.1: Velikost chyby pro různé funkce g, Příklad 1.

Z hodnot uvedených v Tabulce 4.1 můžeme dále pro metody, které konvergují, pomocí algebraického systému rovnic

$$\frac{|e_n|}{|e_{n-1}|^{r_n}} = C_n, \qquad \frac{|e_{n-1}|}{|e_{n-2}|^{r_n}} = C_n$$

vypočítat odhad řádu konvergence r_n a konstanty C_n pro $n > 2$. Tyto hodnoty jsou uvedeny v Tabulce 4.2.

	$\sqrt{x+2}$		$1+\frac{2}{x}$		$\frac{x^2+2}{2x-1}$	
n	r_n	C_n	r_n	C_n	r_n	C_n
3	0.970281	0.236067	0.464973	0.333333	1.760374	0.200000
4	0.992274	0.243683	1.543496	1.090096	1.958580	0.275150
5	0.998049	0.247719	0.820116	0.340286	1.998597	0.328685
6	0.999511	0.249257	1.108332	0.679187		
7	0.999877	0.249771	0.950847	0.420494		
8	0.999969	0.249932	1.025770	0.557371		
9	0.999992	0.249980	0.987420	0.470057		
10	0.999998	0.249994	1.006365	0.518149		
...			...	...		
20			1.000006	0.500038		

Tabulka 4.2: Odhad řádu konvergence, Příklad 1.

Nyní se zamysleme, zda vypočítané výsledky odpovídají teoretickým poznatkům uvedeným v Kapitole 2.
-(i) Pro $|x| > \frac{1}{2}$ je $g'(x) > 1$ a tedy z teorie nevíme o konvergenci nic. V našem případě metoda pro tuto funkci velmi rychle divergovala. Lze ukázat, že metoda diverguje pro libovolnou volbu x_0 [4].
-(ii) Pro $x \in [0,7]$ je $0 \leq g'(x) \leq 1/\sqrt{8} < 1$ a $g(x) \in [0,7]$. Jsou tedy splněny předpoklady Věty 7 a pro libovolné $x_0 \in [0,7]$ posloupnost generovaná fixed-point iterací pro tuto funkci g konverguje k pevnému bodu $x^* = 2$ a to lineárně s konstantou $C = g'(2) = \frac{1}{4}$ podle Věty 9. Experiment tedy souhlasí s teorií, dle hodnot uvedených v Tabulce 4.2 metoda konverguje lineárně s konstantou blížící se $\frac{1}{4}$.
-(iii) Pro $x \in [\sqrt{2}, 4]$ splňuje funkce g také předpoklady Věty 7 i Věty 9 a tedy metoda konverguje lineárně s konstantou $C = g'(2) = 1/2$. Tomu odpovídá provedený numerický experiment. Navíc můžeme vidět, že konstanta C je v tomto případě větší, než v (ii) a tedy konvergence by měla být pomalejší, což odpovídá hodnotám v Tabulce 4.1.
-(iv) Tato volba funkce g odpovídá podle Kapitoly 3 Newtonově metodě a na intervalu $[\frac{3}{4}, 6]$ splňuje předpoklady Věty 12, metoda tedy konverguje. Z numerického experimentu navíc vidíme, že konvergence je téměř kvadratická, což odpovídá teorii, neboť $g'(2) = 0$.

Příklad 2. Praktickou aplikací numerického řešení nelineárních rovnic může být výpočet odmocniny. Mějme funkci $f(x) = x^3 - 3$. Jediným reálným řešením rovnice $f(x) = 0$ je $\sqrt[3]{3}$. Uvažujme počáteční odhad $x_0 = 1$ a dále $x_1 = 2$ pro metodu sečen a také $x_2 = 3$ pro Mullerovu metodu a inverzní kvadratickou interpolaci, respektive $[a_0, b_0] = [1, 2]$ pro bisekci a metodu regula falsi, a zastavovací kritérium $|f(x_n)| < 10^{-6}$, respektive $|b_n - a_n| < 10^{-6}$. V Tabulce 4.3 jsou uvedeny hodnoty

$|e_n|$ a v Tabulce 4.4 pak odhad řádu konvergence r_n, pro metodu regula falsi i hodnota C_n. Pro metodu půlení intervalu hodnoty nejsou v Tabulce 4.4 uvedeny, neboť již víme, že konvergence je lineární s konstantou $\frac{1}{2}$.

n	Bisekce	Regula falsi	Newtonova metoda	Metoda sečen	Mullerova metoda	Inverz. kvard. interpolace
1	0.500000	0.442249	0.224417	-	-	-
2	0.250000	0.156535	0.028861	0.156535	-	-
3	0.125000	0.050190	0.000562	0.050190	0.057750	0.121833
4	0.062500	0.015515	0.000000	0.006015	0.009674	0.029185
5	0.031250	0.004740		0.000213	0.000141	0.002080
6	0.015625	0.001442		0.000001	0.000000	0.000006
7	0.007812	0.000438		0.000000		0
...	...	...				
12	0.000244	0.000001				
13	0.000122	0.000000				
...						
19	0.000001					
20	0.000000					

Tabulka 4.3: Velikost chyby, Příklad 2.

	Regula falsi		Newtonova metoda	Metoda sečen	Mullerova metoda	Inverz. kvadr. interpolace
n	r_n	C_n	r_n	r_n	r_n	r_n
3	1.095198	0.382539	1.919956	-	-	-
4	1.032073	0.340280	1.993434	1.865050	-	-
5	1.010076	0.318605		1.573311	2.366213	1.848375
6	1.003093	0.309471		1.642539	2.204818	2.178880
7	1.000943	0.305930		1.608306		1.722337
8	1.000286	0.304619				-
9	1.000087	0.304149				
...	...	...				
12	1.000002	0.303909				
13	1.000000	0.303902				

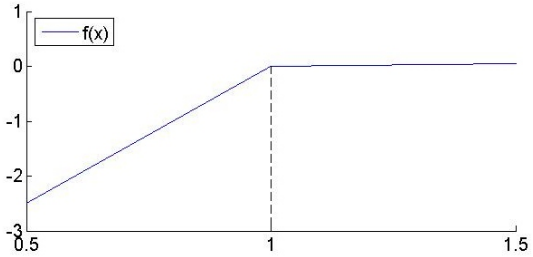
Tabulka 4.4: Odhad řádu konvergence, Příklad 2.

Z hodnot uvedených v Tabulce 4.3 je zřejmé, že všechny testované metody konvergují. Metoda regula falsi v tomto případě podle hodnot z Tabulky 4.4 konverguje lineárně s rychlostí přibližně 0.3 a tedy rychleji, než metoda půlení intervalu.

Zadaná funkce je třikrát spojitě diferencovatelná a platí $f'(\sqrt[3]{3}) \neq 0$. Pro Newtonovu metodu, metodu sečen, Mullerovu metodu i metodu inverzní kvadratické interpolace jsou tedy splněny předpoklady pro lokální konvergenci. Newtonova metoda konverguje nejrychleji z uvedených metod, podle Tabulky 4.4 je konvergence téměř kvadratická již od druhé iterace. Metoda sečen konverguje zřejmě superlineárně s řádem blížícím se hodnotě 1.618, což také odpovídá teorii uvedené v Kapitole 3. Mullerova metoda konverguje zpočátku rychleji, než kvadraticky, ale při výpočtu řešení s větší přesností již konverguje superlineárně. Inverzní kvadratická interpolace nalezne dokonce přesné řešení, což je ovšem pouze náhoda.

Příklad 3. V Příkladu 2 jsme porovnávali konvergenci metod pro třikrát spojitě diferencovatelnou funkci. Zkusme nyní otestovat funkci, která je spojitá, ale není spojitě diferencovatelná. Uvažme jednoduchou po částech lineární funkci zobrazenou na Obrázku 4.1 s předpisem

$$f(x) = \begin{cases} 5x - 5, & \text{pro } x \leq 1, \\ 0.1x - 0.1, & \text{pro } x > 1. \end{cases}$$



Obrázek 4.1: Graf funkce f, Příklad 3.

Tato funkce má kořen v bodě 1 a v tomto bodě není diferencovatelná. Zvolme počáteční odhad $x_0 = 0.5$, ($x_1 = 2$, $x_2 = 3$), respektive $[a_0, b_0] = [0.5, 1]$ pro bisekci a metodu regula falsi, a zastavovací kritérium $|f(x_n)| < 10^{-6}$, respektive $|b_n - a_n| < 10^{-6}$.

n	Bisekce	Regula falsi	Newtonova metoda	Metoda sečen	Mullerova metoda	Inverz. kvadr. interpolace
1	0.750000	0.942307	0	-	-	-
2	0.375000	0.889918		0.942307	-	-
3	0.187500	0.842142		0	2.273272	0.069800
4	0.093750	0.798404			0.000000	0.000000
5	0.046875	0.758222				
...	...	...				
21	0.000001	0.386792				
22	0.000000	0.373281				
...		...				
516		0.000009				

Tabulka 4.5: Velikost chyby, Příklad 3.

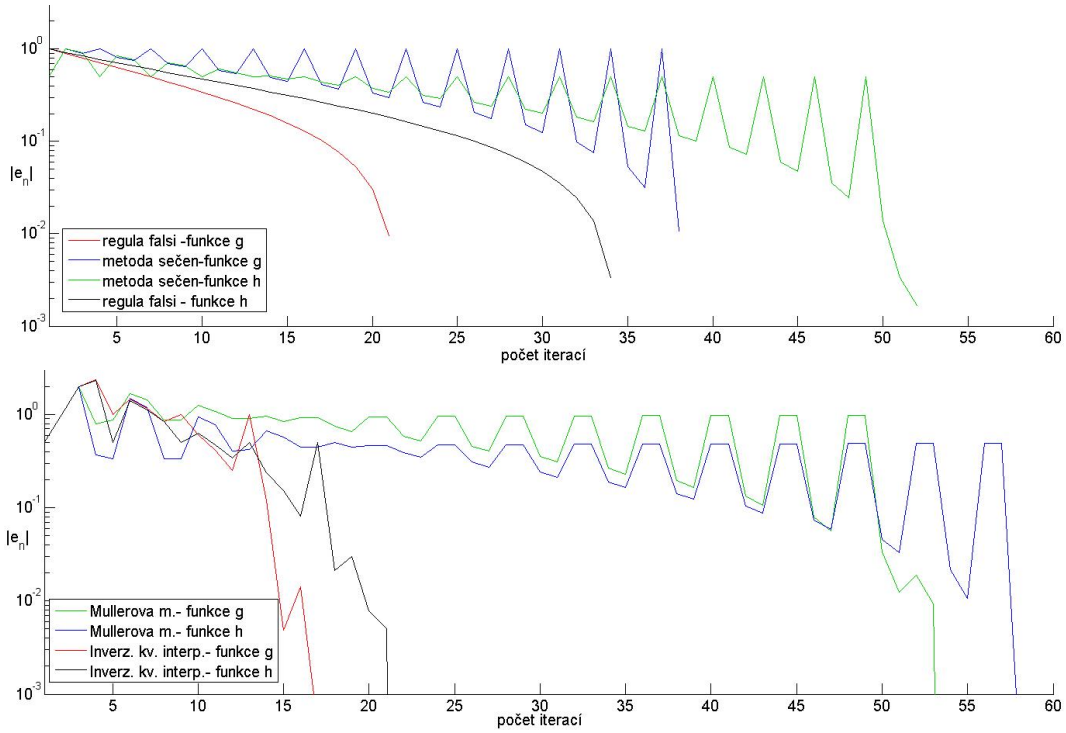
Z Tabulky 4.5 je zřejmé, že Newtonova metoda nalezne přesné řešení v první iteraci. To odpovídá geometrickému náhledu, neboť pro libovolný počáteční bod tečna kopíruje graf funkce až do kořene. Podobně metoda sečen nalezne přesné řešení nejpozději ve třetí iteraci, neboť příslušné body x_n, x_{n-1} leží buď oba napravo nebo nalevo od přesného řešení a sečna tedy také kopíruje graf funkce až do kořene. Totéž platí pro Mullerovu metodu a inverzní kvadratickou interpolaci, neboť nejpozději ve čtvrté iteraci leží všechny interpolační body napravo nebo nalevo od kořene a tedy interpolační polynom je přímka procházející kořenem. Metoda regula falsi konverguje v tomto případě lineárně, avšak s rychlostí přibližně 0.9537 a tedy velmi pomalu.

Uvažme pro srovnání dvě další po částech lineární funkce s předpisy

$$g(x) = \begin{cases} 5x - 5, & \text{pro } x \leq 50/49, \\ 0.1x, & \text{pro } x > 50/49, \end{cases} \qquad h(x) = \begin{cases} 5x - 5, & \text{pro } x \leq 495/490, \\ 0.1x - 0.05, & \text{pro } x > 495/490. \end{cases}$$

Obě mají též kořen v 1, avšak bod, kde funkce nemají derivaci, neleží v kořeni. Pro funkci g dochází ke zlomu přibližně v bodě 1.0204, pro funkci h v bodě 1.0100. Zvolme stejné počáteční odhady a zastavovací kritérium jako pro funkci f.

Metoda bisekce se pro obě tyto funkce chová stejně, jako pro funkci f, Newtonova metoda najde kořen nejpozději ve druhé iteraci. Zajímavé je však srovnání chyby ostatních metod. Velikost chyby tentokrát nebudeme vypisovat do tabulky, lépe bude její vývoj vidět z grafů v Obrázku 4.2, kde na ose y volíme logaritmickou škálu.

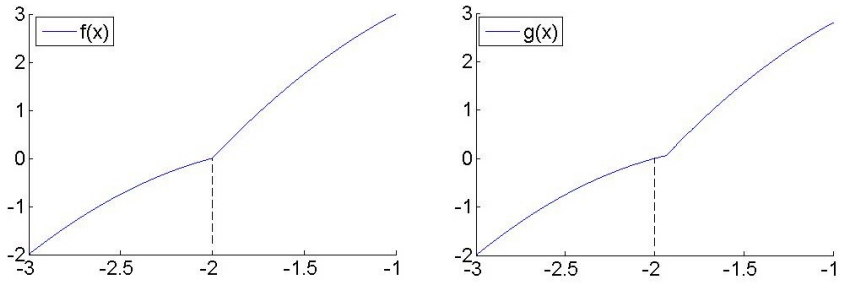


Obrázek 4.2: Graf velikosti chyby pro funkce g, h, Příklad 3.

Z grafů v Obrázku 4.2 vidíme, že čím blíže je bod, ve kterém dochází ke zlomu funkce, ke kořeni, tím více iterací všechny testované metody potřebují pro nalezení kořene. Zajímavý je především vývoj velikosti chyby pro metodu sečen a Mullerovu metodu. Pro obě porovnávané funkce dochází u obou metod k výrazné oscilaci a to až do doby, než je nalezeno přesné řešení.

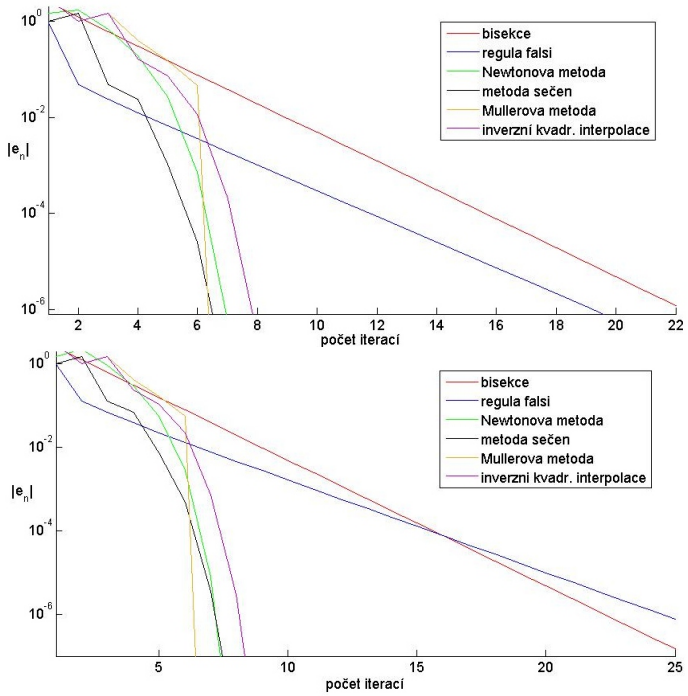
Příklad 4. V Příkladu 3 jsme porovnávali chování metod na lineárních lomených funkcích. Uvažme nyní dvě po částech kvadratické funkce zobrazené na Obrázku 4.3 a s předpisy

$$f(x) = \begin{cases} -x^2 - 3x - 2, & \text{pro } x \leq -2, \\ -x^2 + 4, & \text{pro } x > -2, \end{cases} \qquad g(x) = \begin{cases} -x^2 - 3x - 2, & \text{pro } x \leq -55/30, \\ -x^2 + 3.5, & \text{pro } x > -55/30. \end{cases}$$



Obrázek 4.3: Grafy funkcí f a g, Příklad 4.

Obě funkce mají kořen v bodě -2, funkce f navíc v tomto bodě nemá derivaci, g nemá derivaci v bodě $-\frac{55}{30} \approx -1.8333$. Zvolme počáteční odhady pro kořen $x_0 = -3$, ($x_1 = -0.5$, $x_2 = -1$), respektive $[a_0, b_0] = [-3, -0.5]$ a zastavovací kritérium $|f(x_n)| < 10^{-6}$, $|g(x_n)| < 10^{-6}$, respektive $|b_n - a_n| < 10^{-6}$. Místo zadávání výsledků do tabulek opět použijeme grafy závislosti velikosti chyby na počtu iterací s logaritmickou škálou na ose y.

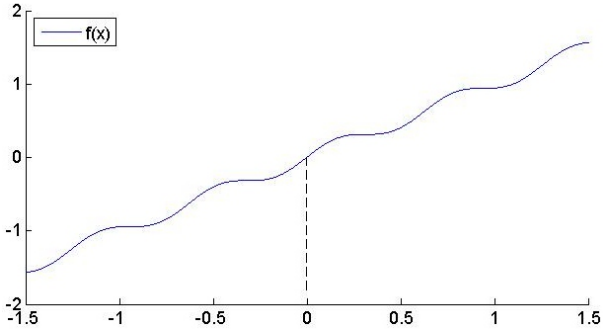


Obrázek 4.4: Grafy velikosti chyby, první graf je pro funkci f, druhý pro funkci g, Příklad 4.

Z grafů v Obrázku 4.4 je vidět, že chování všech metod se v závislosti na zvolené funkci příliš nemění, nezáleží na tom, zda singularita je v kořeni či v okolí kořene. Newtonova metoda konverguje v obou případech kvadraticky a to i přesto, že v prvním případě nejsou splněny předpoklady Věty 11 a v druhém případě sice existuje okolí, na kterém jsou předpoklady Věty 11 splněny, ale počáteční odhad x_0 ležel mimo toto okolí. Zajímavé je chování Mullerovy metody, která v obou případech konverguje zpočátku pomalu a pak v jedné iteraci z přibližného řešení řádu 10^{-1} vypočítá řešení o 7 řádů přesnější. To lze snadno vysvětlit, neboť

interpolační body v šesté iteraci leží v obou případech všechny nalevo od "bodu zlomu"a tedy parabola daná Mullerovou metodou kopíruje graf funkce až do kořene. Přesné řešení v této iteraci není nalezeno vlivem zaokrouhlovacích chyb.

Příklad 5. Uvažujme nyní funkci $f(x) = x+0.1\sin(10x)$ znázorněnou v Obrázku 4.5. Tato funkce je na celém svém definičním oboru třikrát spojitě diferencova-



Obrázek 4.5: Graf funkce f, Příklad 5.

telná a jediným řešením příslušné rovnice (1.1) je $x^* = 0$. Navíc na intervalu $(-\frac{\pi}{10}, \frac{\pi}{10})$ má nenulovou derivaci a tedy splňuje předpoklady vět pro konvergenci Newtonovy metody, metody sečen i metod založených na kvadratické interpolaci. Nás však nyní bude zajímat, jak se metody chovají mimo toto okolí.

Nejzajímavější je Newtonova metoda, při volbě náhodných počátečních bodů z intervalu $[-5, 5]$ metoda v některých případech rychle konvergovala, v některých případech však také velice rychle divergovala. V Tabulce 4.6 je pro ilustraci uvedeno chování metody na okolí bodu 1. Zastavovacím kritériem pro konvergenci je $|f(x)| < 10^{-6}$. Přesáhne-li počet iterací 100, nebo pokud $|x_n| > 10^5$, metoda bude zastavena a výsledkem bude divergence.

x_0	konvergence/ divergence	počet iterací		x_0	konvergence/ divergence	počet iterací
1.0000	diverguje	18		1.0033	konverguje	7
1.0010	diverguje	2		1.0035	konverguje	4
1.0020	diverguje	6		1.0037	konverguje	32
1.0028	diverguje	9		1.0038	diverguje	12
1.0029	konverguje	9		1.0040	diverguje	27
1.0030	konverguje	53		1.0042	diverguje	91
1.0031	konverguje	16		1.0043	konverguje	7

Tabulka 4.6: Výsledky Newtonovy metody, Příklad 5.

Z Tabulky 4.6 je zřejmé, že chování Newtonovy metody, pokud nejsme na dostatečně malém okolí kořene, je v tomto případě jen obtížně předvídatelné.

Metoda sečen je pro tento příklad mnohem spolehlivější. Při volbě dvaceti náhodných počátečních bodů z intervalu $[-30, 30]$ metoda sečen v devatenácti případech našla řešení s přesností 10^{-6} do osmi iterací. Volbou, jenž způsobila divergenci, byla $x_0 = 0.314$, $x_1 = 0.315$, tj. oba body leží v blízkosti bodu, kde má f nulovou derivaci. Mullerova metoda a metoda kvadratické interpolace vykazovaly ještě lepší chování, než metoda sečen. Pro všechny testované body obě vždy našly řešení s přesností 10^{-6} a to nejpozději ve čtrnácté iteraci.

Otestujme nyní, zda špatné chování Newtonovy metody lze při použití stejných zastavovacích kritérií i počátečních odhadů zlepšit aplikací backtrackingu, modifikací Newtonovy metody uvedenou v Kapitole 3. V Tabulce 4.7 je kromě počtu iterací uveden také počet provedených "backtrackingových kroků", tj. kolikrát během výpočtu bylo třeba nahradit bod x_n daný Newtonovou metodou bodem ležícím blíže k x_{n-1}. Pro testované volby x_0 modifikovaný algoritmus vždy konvergoval.

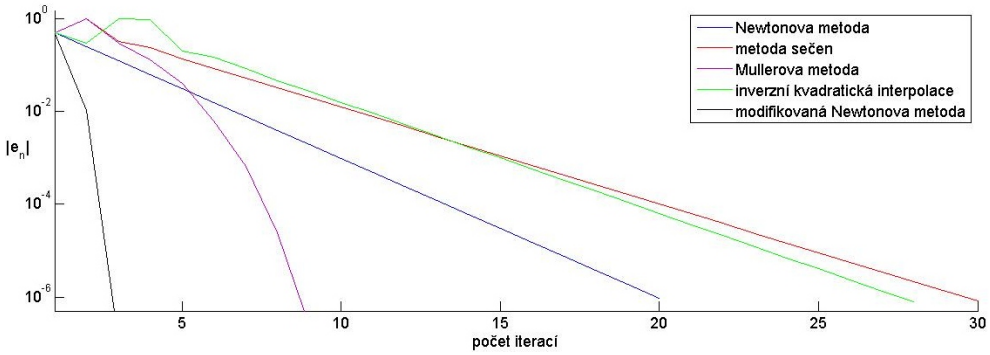
x_0	počet backtr. kroků	počet iterací		x_0	počet backtr. kroků	počet iterací
1.0000	2	5		1.0033	15	4
1.0010	4	5		1.0035	11	5
1.0020	4	5		1.0037	8	5
1.0028	7	4		1.0038	8	6
1.0029	8	8		1.0040	7	7
1.0030	8	6		1.0042	6	4
1.0031	9	6		1.0043	6	6

Tabulka 4.7: Výsledky backtrackingu, Příklad 5.

Z Tabulky 4.7 vidíme, že použitá modifikace vedla k výraznému zlepšení výsledků. Ve všech případech zamezila divergenci standardní Newtonovy metody a řešení požadované přesnosti bylo vždy nalezeno do osmi iterací.

Příklad 6. Nyní se podíváme na funkce s vícenásobnými kořeny. Porovnáme chování standardních metod a také modifikované Newtonovy metody pro vícenásobné kořeny. Vzhledem ke špatné podmíněnosti obou funkcí nebudeme volit zastavovací kritérium na základě $|f(x_n)|$, ale zvolíme $|x_n - x_{n-1}| < 10^{-6}$.

Nejprve uvažme funkci $f(x) = 1 - \cos(x)$, která má v bodě 0 dvojnásobný kořen a počáteční odhad $x_0 = 0.5$, ($x_1 = 1$, $x_2 = -0.3$). Pro metodu inverzní kvadratické interpolace tato volba vede k divergenci, speciálně pro ni proto zvolíme $x_0 = -0.5$, $x_1 = 0.3$, $x_2 = 1$. Neboť funkce v kořeni nemění znaménko, metoda půlení intervalu a metoda regula falsi nejsou definovány. Velikost naměřené chyby je znázorněna v grafu v Obrázku 4.6 a odhad řádu konvergence v Tabulce 4.8.



Obrázek 4.6: Graf velikosti chyby pro funkci f, Příklad 6.

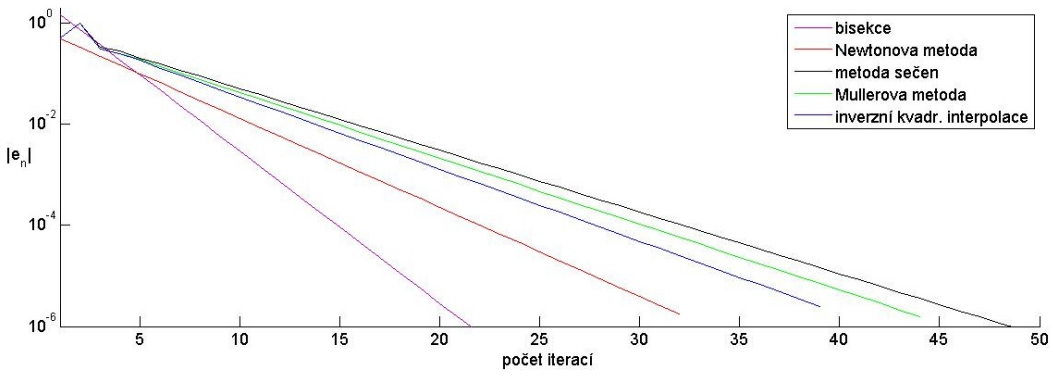
Z grafu v Obrázku 4.6 i Tabulky 4.8 je patrné, že metoda sečen a metoda inverzní kvadratické interpolace konvergují pomaleji, než Newtonova metoda. Podle

n	Newtonova metoda		Metoda sečen		Mullerova metoda	Inverzní kvadr. interpolace	
	r_n	C_n	r_n	C_n	r_n	r_n	C_n
3	0.976819	0.481516	-	-	-	-	-
4	0.994566	0.493699	0.266587	0.318522	-	-	-
5	0.998661	0.497976	1.831309	1.908041	1.363819	33.084371	-
6	0.999916	0.499379	0.814665	0.437318	1.580917	0.183101	-
7	0.999979	0.499816	1.081828	0.747709	1.230772	2.042350	-
8	0.999994	0.499946	0.969783	0.567415	1.446561	0.984767	0.540144
9	0.999998	0.499984	1.01139	0.641649	1.413562	0.924769	0.465155
...	...	...	...	...		...	...
20	0.999993	0.499959	0.999999	0.618032		0.999998	0.576461
...			...	...		...	...

Tabulka 4.8: Odhad řádu konvergence pro funkci f, Příklad 6.

Věty 13 Newtonova metoda pro dvojnásobný kořen konverguje lineárně s rychlostí 1/2, čemuž odpovídají i hodnoty v Tabulce 4.8. Konvergence metody sečen by měla být též lineární a s rychlostí C splňující $C^2 + C - 1 = 0$, $C \in (0, 1)$, tj. $C \approx 0.618$, čemuž odpovídají hodnoty v Tabulce 4.8. Mullerova metoda v tomto případě konverguje superlineárně, avšak poznamenejme, že posloupnost $\{x_n\}$ daná touto metodou je komplexní a velikost komplexní části čísel x_n je ve většině iterací řádově srovnatelná s reálnou částí. Nejrychleji podle očekávání konverguje modifikovaná Newtonova metoda a to díky faktu, že známe násobnost kořene. Zadáme-li do algoritmu místo 2 násobnost kořene $m = 3$, pak metoda konverguje pomaleji než standardní Newtonova metoda, pro $m = 4$ metoda diverguje.

Nyní uvažme funkci $g(x) = x^3$, která má v bodě 0 trojnásobný kořen, a počáteční odhad $x_0 = -0.5$, ($x_1 = 1$, $x_2 = -0.3$). Absolutní hodnota chyby jednotlivých metod v závislosti na iteraci je zanesena do grafu v Obrázku 4.7.



Obrázek 4.7: Graf velikosti chyby pro funkci g, Příklad 6.

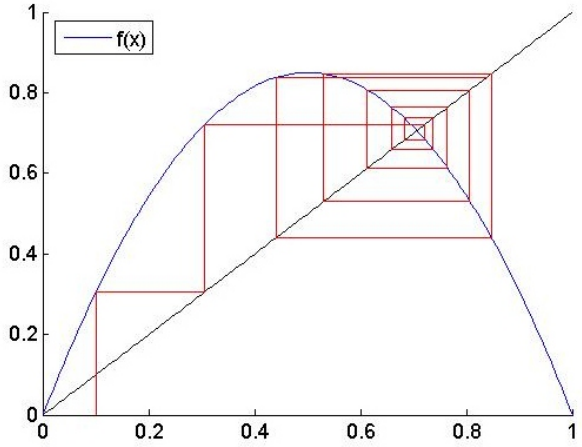
Metoda regula falsi v grafu není uvedena, neboť její konvergence je velmi pomalá (po vypočítání 5000 iterací je aproximace řešení daná touto metodou 0.01). Hodnota chyby pro modifikovanou Newtonovu metodu v grafu také není zanesena, neboť tato metoda najde přesné řešení již v první iteraci a to dokonce pro libovolný odhad x_0, neboť platí

$$x_1 = x_0 - 3\frac{x_0^3}{3x_0^2} = 0.$$

Ostatní metody konvergují podle grafu v Obrázku 4.7 lineárně. Nejrychleji z nich metoda bisekce s rychlostí konvergence 0.5. Dále Newtonova metoda s rychlostí 0.667, což opět odpovídá Větě 13. Pro informaci ještě uveďme rychlost konvergence pro zbylé metody: inverzní kvadratická interpolace 0.720, Mullerova metoda 0.741 a metoda sečen 0.755.

Příklad 7. V Kapitole 3 jsme uvedli, že metody se při nesplnění předpokladů konvergenčních vět mohou chovat i chaoticky. Tuto situaci můžeme hezky ilustrovat na fixed-point iteraci.

Zvolme funkci $g(x) = 3.4x(1-x)$, která má dva průsečíky s osou $y = x$, první v bodě 0 a druhý přibližně v bodě 0.7058. V obou těchto bodech má funkce g derivaci ostře větší než 1 a tedy nesplňuje předpoklady Věty 8. Položme $x_0 = 0.4$ a zastavovací kritérium $|g(x_n) - x_n| < 10^{-6}$. Metoda nenalezne řešení ani po 300 iteracích a průběh výpočtu je chaotický. Prvních dvacet iterací je znázorněno v Obrázku 4.8.



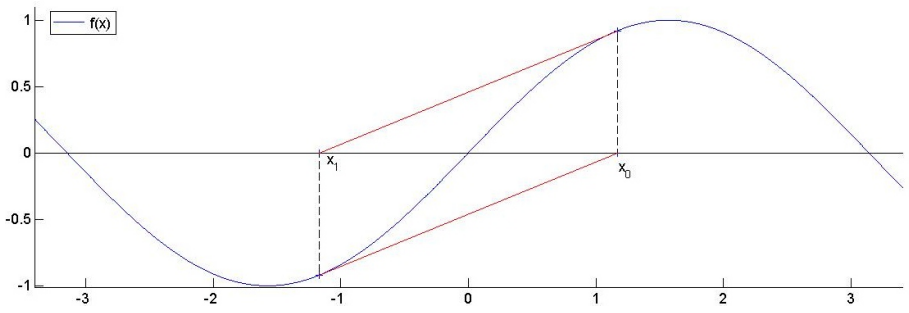
Obrázek 4.8: Graf chaotického chování, Příklad 7.

Grafy tohoto typu se nazývají Verhulstovy diagramy, nebo také cobweb plots (z angličtiny cobweb=pavučina).

Příklad 8. Dalším zajímavým typem chování některých metod je zacyklení, či konvergence k jinému kořeni, než bychom očekávali. Tento případ ilustrujme na Newtonově metodě pro funkci $f(x) = \sin(x)$.

Předpokládejme, že hledáme kořen $x^* = 0$. Z Obrázku 4.9 vidíme, že aby nastalo cyklení Newtonovy metody mezi body x_0, x_1, musí tyto body splňovat

$$x_1 = x_0 - \frac{\sin(x_0)}{\cos(x_0)}, \quad x_1 = -x_0.$$



Obrázek 4.9: Graf funkce f, Příklad 8.

Bod x_0 tedy musí splňovat rovnici

$$2x_0 = \tan(x_0),$$

která má na intervalu $[-\pi,\pi]$ dvě řešení $\tilde{x_0}$ a $-\tilde{x_0}$, kde $\tilde{x_0} \approx 1.165561185$. Pro tyto dvě volby x_0 nebude tedy Newtonova metoda konvergovat, ale zacyklí se. Zajímavé je i chování Newtonovy metody na okolí těchto bodů. Zvolme několik různých bodů x_0 v okolí $\tilde{x_0}$ a zastavovací kritérium $|f(x_n)| < 10^{-6}$. V Tabulce 4.9 je zanesena závislost počtu iterací a vypočítaného řešení na volbě x_0.

x_0	počet iterací	kořen
1.164000000	7	0
1.165500000	10	0
1.165561185	17	0
$\tilde{x_0}$	-	-
1.165561186	21	25π
1.165561187	18	π
1.165561188	15	-13π
1.300000000	4	$-\pi$

Tabulka 4.9: Výsledky Newtonovy metody, Příklad 8.

Z Tabulky 4.9 vidíme, že pro x_0 zvolená v intervalu $[-\tilde{x_0}, \tilde{x_0}]$ metoda konverguje ke kořeni v bodě 0, což odpovídá geometrickému náhledu z Obrázku 4.9. Pro body mimo tento interval se metoda chová nepředvídatelně a i velmi malá změna ve volbě x_0 vede ke konvergenci k velmi odlišným kořenům. Ve většině těchto případů lze chování algoritmu zlepšit použitím backtrackingu, ale například pro volbu $x_0 = 1.3$ i modifikovaný algoritmus konverguje ke kořeni $-\pi$.

Závěr

Cílem této bakalářské práce bylo shrnout základní metody řešení skalárních nelineárních rovnic a dále tyto metody implementovat a porovnat pomocí numerických experimentů. Nejprve byly zavedeny základní matematické pojmy související s tématem. Dále se čtenář seznámil se základními metodami a to především s metodou bisekce a metodou fixed-point iterace a v návaznosti na ně s důležitou Newtonovou metodou, jejími modifikacemi a dalšími algoritmy z ní vycházejícími. V závěrečné čtvrté kapitole byly popsány výsledky numerických experimentů provedených pomocí softwaru Matlab. Testovací příklady byly voleny tak, aby bylo možné porovnat numerické výsledky s poznatky uvedenými v teoretické části. Všechny provedené experimenty potvrdili odvozené teoretické vlastnosti metod. Navíc bylo uvedeno několik příkladů rovnic nesplňujících konvergenční předpoklady vybraných iteračních metod, aby bylo ilustrováno zajímavé chování těchto metod.

Existuje velké množství dalších metod, které nebyly v práci uvedeny ani testovány. Zajímavým algoritmem pro studium by mohl být například Brentův algoritmus, který kombinuje tři v této práci uvedené metody a to metodu půlení intervalu, metodu sečen a metodu inverzní kvadratické interpolace. Jiným navazujícím problémem by mohlo být řešení soustav nelineárních rovnic a tedy i možnosti rozšíření uvedených metod do více dimenzí.

Literatura

[1] Antia, H. (2002). *Numerical Methods for Scientists and Engineers*. Birkhäuser, Basilej.

[2] Bhardwaj, D. (2003). *Solution of Nonlinear Equations*. Online. URL `www.cse.iitd.ernet.in/~dheerajb/CS210_lect09.pdf`.

[3] Bhardwaj, D. (2003). *Solution of Nonlinear Equations II*. Online. URL `www.cse.iitd.ernet.in/~dheerajb/CS210_lect10.pdf`.

[4] Conte, S. D. a de Boor, C. (1980). *Elementary Numerical Analysis: An Algoritmic Approach*. The McGraw-Hill Companies, New York.

[5] Dennis, J. J. a Schnabel, R. B. (1996). *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*. SIAM, Philadelphia.

[6] Deuflhard, P. (2004). *Newton Methods for Nonlinear Problems*. Springer, Berlin.

[7] Heath, M. J. (1997). *Scientific Computing An Introductory Survey*. The McGraw-Hill Companies, New York.

[8] Jarník, V. (1974). *Diferenciální počet 1*. Academia, Praha.

[9] Jia, Y.-B. (2004). *Convergence Rates on Root Finding*. Online. URL `www.cs.iastate.edu/~cs577/handouts/convergence.pdf`.

[10] Kelley, C. (1995). *Iterative Methods for Linear and Nonlinear Equations*. SIAM, Philadelphia.

[11] Kelley, C. (2003). *Solving Nonlinear Equations with Newtons Method*. SIAM, Philadelphia.

[12] Kopecky, K. (2007). *Root Finding Methods*. Online. URL `www.karenkopecky.net/Teaching/eco613614/Notes_RootFindingMethods.pdf`.

[13] Orthega, J. a Rheinbodt, W. (2000). *Iterative Solution of Nonlinear Equations in Several Variables*. SIAM, Philadelphia.

[14] Radul, A. (2013). *Introduction to Automatic Differentiation*. Online. URL `alexey.radul.name/ideas/2013/introduction-to-automatic-differentiation/`.

[15] Rall, L. B. (1966). Convergence of the newton process to multiple solutions. *Numerische Mathematik*, **9**(1), 23–37.

[16] Segethová, J. (2000). *Základy numerické matematiky*. Karolinum, Praha.

[17] Spiteri, R. J. (2013). *Solution of Nonlinear Equations*. Online. URL `www.cs.usask.ca/~spiteri/M211/notes/chapter4.pdf`.

Příloha A

Zdrojové kódy

A.1 Bisekce

```
function [z,n,x]=bisekce(f,x1,x2,tol,maxit)
  %metoda puleni intervalu
  %povinne vstupni parametry - funkce f, interval [x1,x2]
  %volitelne vstupni parametry: tol - pozadovana presnost
  %                             maxit - maximalni pocet iteraci
  %vystup: z-hledany interval
  %        n-pocet provedenych iteraci
  %        x-vektor mezivypoctu

  if nargin<5                        %neni-li zadan max.pocet iteraci
    maxit=100;
  end
  if nargin<4                     %neni-li zadana pozadovana presnost
    tol=1e-8;
  end

  n=0;
  x(1,1)=x1;             %inicializace vystupniho vektoru mezivypoctu
  x(2,1)=x2;
  if (sign(feval(f,x1))==sign(feval(f,x2)))           %spatne zadani
    error('Vstupni funkce nemeni na zadanem intervalu znamenko.')
  else
    while (n<maxit)
        m=x1+(x2-x1)/2;                %novy bod uprostred intervalu
        if feval(f,m)==0                          %bod m je korenem f
            z=[m,m];
            n=n+1;
            display('Nalezeno presne reseni');
            return;
        end
        if (sign(feval(f,x1))==sign(feval(f,m)))      %dalsi iterace
             x1=m;
             x(1,n+2)=m;
             x(2,n+2)=x(2,n+1);
        else x2=m;
             x(2,n+2)=m;
             x(1,n+2)=x(1,n+1);
        end
```

```
        if (abs(x(2,n+2)-x(1,n+2))<tol) %interval je dostatecne maly
           z=[x(1,n+2),x(2,n+2)]; %hledany interval obsahujici koren
           n=n+1;
           return;
        end
        n=n+1;
     end
  error('Prilis mnoho iteraci');%nenalezeno dostatecne dobre reseni
  end
end
```

A.2 Fixed-point iterace

```
function [z,n,x]=fixed_point(f,x1,tol,maxit)
 %metoda fixed-point iterace
 %povinne vstupni parametry - funkce f, odhad korene x1
 %volitelne vstupni parametry: tol - pozadovana presnost
 %                             maxit - maximalni pocet iteraci
 %vystup: z-hledany interval
 %        n-pocet provedenych iteraci
 %        x-vektor mezivypoctu

  if nargin<4                        %neni-li zadan max.pocet iteraci
   maxit=100;
  end
  if nargin<3                      %neni-li zadana pozadovana presnost
   tol=1e-8;
  end

  n=0;
  x(1)=x1;                           %inicializace vystupniho vektoru

  if (abs(feval(f,x1)-x1)<tol)         %vstupni odhad x1 je pevny bod
   z=x1;
   return;
  else
   while (n<maxit)
        m=feval(f,x1);                          %vypocet noveho bodu m
        if  (abs(feval(f,m)-m)<tol)     %m je dobre priblizne reseni
             z=m; x(n+2)=m; n=n+1;
             return;
        end
        x1=m;           %zatim nemame reseni, provedeme dalsi iteraci
        n=n+1;
        x(n+1)=m;
   end
   error('Prilis mnoho iteraci');%nenalezeno dostatecne dobre reseni
  end
end
```

A.3 Regula falsi

```
function [z,n,x]=regula_falsi(f,x1,x2,tol,maxit)
  %metoda regula falsi
  %povinne vstupni parametry - funkce f, interval [x1,x2]
  %volitelne vstupni parametry: tol - pozadovana presnost
  %                             maxit - maximalni pocet iteraci
  %vystup: z-hledany interval
  %        n-pocet provedenych iteraci
  %        x-vektor mezivypoctu

  if nargin<5                        %neni-li zadan max.pocet iteraci
    maxit=10000;
  end
  if nargin<4                     %neni-li zadana pozadovana presnost
    tol=1e-8;
  end

  n=0;
  x(1,1)=x1;             %inicializace vystupniho vektoru mezivypoctu
  x(2,1)=x2;
  if (sign(feval(f,x1))==sign(feval(f,x2)))           %spatne zadani
      error('Vstupni funkce nemeni na zadanem intervalu znamenko.')
  else
    while (n<maxit)
       m=x1-feval(f,x1)*(x2-x1)/(feval(f,x2)-feval(f,x1)); %novy bod
       if feval(f,m)==0                                  %m je korenem f
            z=[m,m];
            n=n+1;
            display('Nalezeno presne reseni');
            return;
       end
       if (sign(feval(f,x1))==sign(feval(f,m)))       %dalsi iterace
            x1=m;
            x(1,n+2)=m;
            x(2,n+2)=x(2,n+1);
       else x2=m;
            x(2,n+2)=m;
            x(1,n+2)=x(1,n+1);
       end
       if (abs(feval(f,m))<tol)            %jednostranna konvergence
            z=[m,m]; n=n+1;               %bod m je priblizne reseni
            return;
       end
       if (abs(x(2,n+2)-x(1,n+2))<tol)      %oboustranna konvergence
            z=[x(1,n+2),x(2,n+2)];%hledany interval obsahujici koren
            n=n+1;
            return;
       end
       n=n+1;
    end
  error('Prilis mnoho iteraci'); %nenalezeno dostatecne dobre reseni
  end
end
```

A.4 Newtonova metoda

```
function [z,n,x]=newtonova_metoda(f,df,x1,tol,maxit,maxval)
 %Newtonova metoda
 %povinne vstupni parametry - funkce f
 %                          - derivace funkce f -df
 %                          - pocateni odhad korene x1
 %volitelne vstupni parametry: tol - pozadovana presnost
 %                             maxit - maximalni pocet iteraci
 %                             maxval - maximalni povolena hodnota
 %vystup: z-hledany interval
 %        n-pocet provedenych iteraci
 %        x-vektor mezivypoctu
  if nargin<6                      %neni-li zadana maximalni hodnota
     maxval=100000;
  end
  if nargin<5                  %neni-li zadan maximalni pocet iteraci
     maxit=100;
  end
  if nargin<4                     %neni-li zadana pozadovana presnost
     tol=1e-8;
  end

  n=0;
  x(1)=x1;                          %inicializace vystupniho vektoru

  if (abs(feval(f,x1))<tol)%x1 je dostatecne dobre priblizne reseni
     z=x1;
     return;
  end
  while (n<maxit)
     m=x1-feval(f,x1)/(feval(df,x1));          %vypocet noveho bodu
     if (abs(m)>maxval)                %nova hodnota je prilis velka
         error('Metoda diverguje');
         abort;                                    %vypocet ukoncime
         return;
     end
     if (abs(feval(f,m))<tol)      %novy bod dostatecne dobre reseni
         z=m; x(n+2)=z; n=n+1;
      return;
     end
     if (abs(x(n+1)-m)<tol)       %metoda stagnuje, vypocet ukoncime
         z=m; x(n+2)=z; n=n+1;
         return;
     end
     x1=m;                  %ulozeni dat pro provedeni dalsi iterace
     n=n+1;
     x(n+1)=x1;
  end
  error('Prilis mnoho iteraci');%nenalezeno dostatecne dobre reseni
end
```

A.5 Newtonova metoda pro vícenásobné kořeny

```
function [z,n,x]=newtonova_metoda_2(f,df,x1,k,tol,maxit,maxval)
 %Modifikovana Newtonova metoda pro k-nasobny koren
 %povinne vstupni parametry - funkce f
 %                          - derivace funkce f -df
 %                          - pocateni odhad korene x1
 %                          - nasobnost hledaneho korene k
 %volitelne vstupni parametry: tol - pozadovana presnost
 %                             maxit - maximalni pocet iteraci
 %                             maxval - maximalni povolena hodnota
 %vystup: z-hledany interval
 %        n-pocet provedenych iteraci
 %        x-vektor mezivypoctu

  if nargin<7                %neni-li zadana maximalni mozna hodnota
     maxval=10000;
  end
  if nargin<6                 %neni-li zadan maximalni pocet iteraci
     maxit=100;
  end
  if nargin<5                    %neni-li zadana pozadovana presnost
     tol=1e-8;
  end

  n=0;
  x(1)=x1;              %inicializace vystupniho vektoru mezivypoctu

  if (abs(feval(f,x1))<tol)           %x1 je dostatecne dobre reseni
     z=x1;
     return;
  end
  while (n<maxit)
     m=x1-k*feval(f,x1)/(feval(df,x1));       %vypocet noveho bodu m
     if (abs(m)>maxval)                 %nova hodnota je prilis velka
         error('Metoda diverguje');
         abort;                                      %vypocet ukoncime
     end
     if (abs(feval(f,m))<tol)                     %novy bod je koren
         z=m; x(n+2)=z; n=n+1;
         return;
     end
     if (abs(x(n+1)-m)<tol)       %metoda stagnuje, vypocet ukoncime
         z=m; x(n+2)=z; n=n+1;
         return;
     end
     x1=m;              %ulozeni vypoctu pro provedeni dalsi iterace
     n=n+1;
     x(n+1)=x1;
   end
   error('Prilis mnoho iteraci');%nenalezeno dostatecne dobre reseni
end
```

A.6 Newtonova metoda s backtrackingem

```
function [z,n,x,k]=newtonova_metoda_b(f,df,x1,tol,maxit,maxval)
 %Newtonova metoda s backtrackingem
 %povinne vstupni parametry - funkce f
 %                          - derivace funkce f -df
 %                          - pocateni odhad korene x1
 %volitelne vstupni parametry: tol - pozadovana presnost
 %                             maxit - maximalni pocet iteraci
 %                             maxval - maximalni povolena hodnota
 %vystup: z-hledany interval
 %        n-pocet provedenych iteraci
 %        x-vektor mezivypoctu
 %        k-pocet provedenych zpetnych kroku pro zamezeni divergenci

  if nargin<6                        %neni-li zadana maximalni hodnota
      maxval=100000;
  end
  if nargin<5                  %neni-li zadan maximalni pocet iteraci
    maxit=100;
  end
  if nargin<4                     %neni-li zadana pozadovana presnost
    tol=1e-8;
  end

  n=0;
  x(1)=x1;                           %inicializace vystupniho vektoru
  k=0;

  if (abs(feval(f,x1))<tol) %x1 je dostatecne dobre priblizne reseni
    z=x1;
    return;
  end
  while (n<maxit)
    m=x1-feval(f,x1)/(feval(df,x1));            %vypocet noveho bodu
    while abs(feval(f,m))>abs(feval(f,x1))  %provedeni backtrackingu
        m=(m+x1)/2;
        k=k+1;
    end
    if (abs(m)>maxval)                   %nova hodnota je prilis velka
        error('Metoda diverguje');
        abort;                                        %vypocet ukoncime
        return;
    end
    if (abs(feval(f,m))<tol)       %novy bod dostatecne dobre reseni
        z=m; x(n+2)=z; n=n+1;
        return;
    end
    if (abs(x(n+1)-m)<tol)         %metoda stagnuje, vypocet ukoncime
        z=m; x(n+2)=z; n=n+1;
        return;
    end
    x1=m;                     %ulozeni dat pro provedeni dalsi iterace
    n=n+1;
    x(n+1)=x1;
  end
```

```
  error('Prilis mnoho iteraci');%nenalezeno dostatecne dobre reseni
end
```

A.7 Metoda sečen

```
function [z,n,x]=metoda_secen(f,x1,x2,tol,maxit,maxval)
 %Metoda secen
 %povinne vstupni parametry - funkce f
 %                          - pocateni odhady korene x1, x2
 %volitelne vstupni parametry: tol - pozadovana presnost
 %                             maxit - maximalni pocet iteraci
 %                             maxval - maximalni povolena hodnota
 %vystup: z-hledany interval
 %        n-pocet provedenych iteraci
 %        x-vektor mezivypoctu

  if nargin<6                %neni-li zadana maximalni mozna hodnota
     maxval=10000;
  end
  if nargin<5                 %neni-li zadan maximalni pocet iteraci
     maxit=100;
  end
  if nargin<4                    %neni-li zadana pozadovana presnost
     tol=1e-8;
  end

  n=0;
  x(1)=x1;             %inicializace vystupniho vektoru  mezivypoctu
  x(2)=x2;

  if (abs(feval(f,x1))<=tol)              % prvni krajni bod je koren
     z=x1;
     return;
  end
  if (abs(feval(f,x2))<=tol)              % druhy krajni bod je koren
     z=x2;
     return;
  end
  while (n<maxit)
     if (abs(x1-x2)<tol)                     %metoda zacina stagnovat
        z=x1; x(n+2)=z;
        display('Metoda stagnuje');
        return;
     end
                                                %vypocet noveho bodu m
     m=x2-feval(f,x2)*(x2-x1)/(feval(f,x2)-feval(f,x1));
     if (abs(m)>maxval)                 %nova hodnota je prilis velka
        error('Metoda diverguje');
        abort;                                        %vypocet ukoncime
     end
     if (abs(feval(f,m))<tol)    %novy bod je dostatecne dobre reseni
        z=m;
        x(n+3)=z;
        n=n+1;
```

```
        return;
     end
        x(n+3)=m;              %ulozeni mezivypoctu pro dalsi iteraci
        x1=x2;
        x2=m;
        n=n+1;
  end
  error('Prilis mnoho iteraci'); %nenalezeno dostatecne dobre reseni
end
```

A.8 Mullerova metoda

```
function [z,n,x]=mullerova_metoda(f,x1,x2,x3,tol,maxit,maxval)
 %Mullerova metoda
 %povinne vstupni parametry - funkce f
 %                          - pocateni odhady korene x1, x2, x3
 %volitelne vstupni parametry: tol - pozadovana presnost
 %                             maxit - maximalni pocet iteraci
 %                             maxval - maximalni povolena hodnota
 %vystup: z-hledany interval
 %        n-pocet provedenych iteraci
 %        x-vektor mezivypoctu
  if nargin<7                       %neni-li zadana maximalni hodnota
    maxval=10000;
  end
  if nargin<6                  %neni-li zadan maximalni pocet iteraci
    maxit=1000;
  end
  if nargin<5
    tol=1e-8;                    %neni-li zadana pozadovana presnost
  end

  n=0;
  x(1)=x1;                          %inicializace vystupniho vektoru
  x(2)=x2;
  x(3)=x3;

  if (abs(feval(f,x1))<tol)           %x1 je dostatecne dobre reseni
    z=x1;
    return;
  end
  if (abs(feval(f,x2))<tol)           %x2 je dostatecne dobre reseni
    z=x2;
    return;
  end
  if (abs(feval(f,x3))<tol)           %x3 je dostatecne dobre reseni
    z=x3;
    return;
  end
  while (n<maxit)
              %pomocne vektory pro vypocet interpolacniho polynomu p
     yx=[x1,x2,x3];
     yy=[feval(f,x1),feval(f,x2),feval(f,x3)];
                 % prolozeni polynomem stupne 2 (nemusi existovat)
```

```
      p=polyfit(yx,yy,2);
                  %je-li koeficient u x^2 velmi maly, zanedbame jej
                              %a uvazime interpolaci pomoci primky
      if abs(p(1))<1e-10
         xa=-p(3)/p(2);                          %vypocet korenu primky
         xb=xa;
      else
                                    %vypocet korenu polynomu stupne 2
      xa=(-p(2)+sqrt(p(2)^2-4*p(1)*(p(3))))/(2*p(1));
      xb=(-p(2)-sqrt(p(2)^2-4*p(1)*(p(3))))/(2*p(1));
      if abs(xa-x3)<abs(xb-x3)                   %vyber blizsiho korenu
        m=xa;
       else
        m=xb;
      end
      if (abs(m)>maxval)                 %nova hodnota je prilis velka
         error('Metoda diverguje');
         abort;
      end
      if (abs(feval(f,m))<tol)%novy bod m je dostatecne dobre reseni
         z=m; x(n+4)=z;
         n=n+1;
         return
      end
      if (abs(x(n+3)-m)<tol)                          %metoda stagnuje
         z=m; x(n+4)=z; n=n+1;
         return;                                     %ukoncime vypocet
      end
      x1=x2;                   %ulozeni mezivysledku pro dalsi iteraci
      x2=x3;
      x3=m;
      x(n+4)=m;
      n=n+1;
  end
  error('Prilis mnoho iteraci'); %nenalezeno dostatecne dobre reseni
end
```

A.9 Inverzní kvadratická interpolace

```
function [z,n,x]=inverz_kv_interpolace(f,x1,x2,x3,tol,maxit,maxval)
 %Inverzni kvadraticka interpolace
 %povinne vstupni parametry - funkce f
 %                          - pocateni odhady korene x1, x2, x3
 %volitelne vstupni parametry: tol - pozadovana presnost
 %                             maxit - maximalni pocet iteraci
 %                             maxval - maximalni povolena hodnota
 %vystup: z-hledany interval
 %        n-pocet provedenych iteraci
 %        x-vektor mezivypoctu

  if nargin<7                        %neni-li zadana maximalni hodnota
    maxval=10000;
  end
  if nargin<6                 %neni-li zadan maximalni pocet iteraci
```

```
    maxit=1000;
  end
  if nargin<5
    tol=1e-8;                      %neni-li zadana pozadovana presnost
  end

  n=0;
  x(1)=x1;
  x(2)=x2;
  x(3)=x3;
  if (abs(feval(f,x1))<tol)           %x1 je dostatecne dobre reseni
    z=x1;
    return;
  end
  if (abs(feval(f,x2))<tol)           %x2 je dostatecne dobre reseni
    z=x2;
    return;
  end
  if (abs(feval(f,x3))<tol)           %x3 je dostatecne dobre reseni
    z=x3;
    return;
  end
  while (n<maxit)
     yx=[x1,x2,x3];       %pomocne vektory pro prolozeni polynomem q
     yy=[feval(f,x1),feval(f,x2),feval(f,x3)];
     q=polyfit(yy,yx,2);               %prolozeni polynomem stupne 2
     m=q(3);                                    %vypocet noveho bodu
     if (abs(m)>maxval)                %nova hodnota je prilis velka
       error('Metoda diverguje');
       abort;
     end
     if (abs(feval(f,m))<tol)            % m dostatecne dobre reseni
       z=m; x(n+4)=z;
       n=n+1;
       return
     end
     if (abs(x(n+3)-m)<tol)                          %metoda stagnuje
       z=m; x(n+4)=z; n=n+1
       return;                                     %vypocet ukoncime
     end
     x1=x2;                %ulozeni hodnot pro vypocet dalsi iterace
     x2=x3;
     x3=m;
     x(n+4)=m;
     n=n+1;
  end
 error('Prilis mnoho iteraci'); %nenalezeno dostatecne dobre reseni
end
```