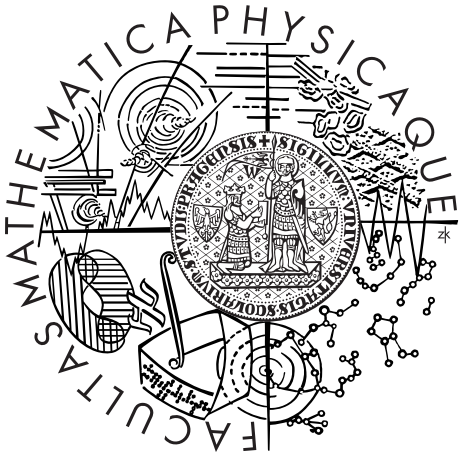
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Matěj Lébl

Numerické počítání s funkcemi pomocí Chebfun

Katedra numerické matematiky

Vedoucí bakalářské práce: RNDr. Petr Tichý, Ph.D.
Studijní program: Matematika
Studijní obor: Obecná matematika

Praha 2015

Rád bych poděkoval RNDr. Petru Tichému, Ph.D. za ochotu, trpělivost a čas, který mi v průběhu zpracování bakalářské práce věnoval. Mé poděkování patří i rodině a blízkým za pomoc a podporu.

Prohlašuji, že jsem tuto bakalářskou práci vypracoval 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 .....

Název práce: Numerické počítání s funkcemi pomocí Chebfun

Autor: Matěj Lébl

Katedra: Katedra numerické matematiky

Vedoucí bakalářské práce: RNDr. Petr Tichý, Ph.D., Katedra numerické matematiky

Abstrakt: Cílem práce je představit software Chebfun a myšlenky, na kterých je postaven. V první kapitole jsou shrnuty poznatky teorie polynomiální interpolace se zaměřením na Čebyševovy interpolanty. V druhé kapitole je představen software Chebfun, jeho základní příkazy a principy vytváření interpolantů. Třetí kapitola je věnována demonstraci tvrzení uvedených v první kapitole a ukázkám praktického použití Chebfunu při hledání kořenů funkce a řešení diferenciálních rovnic.

Klíčová slova: Chebfun, Čebyševovy interpolanty, numerické počítání s funkcemi

Title: Numerical computation with functions using Chebfun

Author: Matěj Lébl

Department: Department of Numerical Mathematics

Supervisor: RNDr. Petr Tichý, Ph.D., Department of Numerical Mathematics

Abstract: Goal of this work is to introduce Chebfun software and show ideas behind it. In the first chapter we summarize the theory of polynomial interpolation with focus on the Chebyshev interpolants. In the second chapter we introduce Chebfun software, its basic commands and principles of constructing interpolants. The third chapter is devoted to demonstrate theorems from the first chapter and to show practical applications of Chebfun when finding roots of a function and solving differential equations.

Keywords: Chebfun, Chebyshev interpolation, numerical computation with functions

Obsah

1 Čebyševovy řady a interpolanty **2**
1.1 Polynomiální interpolace 2
1.2 Čebyševovy body 5
1.3 Čebyševovy řady 8
1.4 Konvergence 10
1.5 Lebesgueova konstanta 12

2 Chebfun – numerické počítání s funkcemi **15**
2.1 Idea toolboxu Chebfun 15
2.2 Stručný průvodce 16
2.3 Jak Chebfun vytváří chebfun 21

3 Praktické ukázky využití softwaru Chebfun **26**
3.1 Obrázky ke kapitole 1 26
3.2 Hledání kořenů funkce 35
3.3 Spektrální metody 36
3.4 Experiment 37

Závěr **42**

Literatura **43**

Kapitola 1

Čebyševovy řady a interpolanty

V technické praxi se vyskytují problémy, kdy potřebujeme pracovat s velmi komplikovanými funkcemi. Výpočetní software je ovšem omezen jak přesností, tak časovou náročností. Hledáme tak vyjádření zadané funkce, se kterým budeme schopni počítat v rozumném čase s dostatečnou přesností.

V této kapitole se seznámíme se základy teorie aproximace funkcí. Konkrétně se zaměříme na polynomiální interpolaci v Čebyševových bodech – Čebyševovy interpolanty. Proč volíme zrovna aproximaci polynomy? Naším cílem je v konečné aritmetice reprezentovat zadanou funkci nějakou jednodušší strukturou, se kterou je možné dostatečně přesně a rychle počítat. K vyhodnocení polynomu v daném bodě stačí operace sčítání a násobení, které jsou v počítači hardwarově implementovány. Polynomy lze jednoduše sčítat, násobit, derivovat i integrovat.

1.1 Polynomiální interpolace

Mějme $(n+1)$-tici hodnot f_j, $j = 0, \ldots, n$ (například hodnoty měření) v bodech x_j (časy měření). Dvojice (x_j, f_j) odpovídají nějaké neznámé funkci $f(x)$. Naším cílem je potom ze znalosti $f_j = f(x_j)$ zkonstruovat aproximaci původní funkce f. Jedna z možných metod je interpolace. Interpolovat můžeme jednoduchou funkcí (například polynomem nebo racionální funkcí) nebo interpolant složit z více funkcí (například interpolace splinem).

Uvažujme úlohu nalezení polynomu stupně n – *interpolačního polynomu* – tak, aby nabýval zadaných $n+1$ funkčních hodnot v daných bodech – *interpolačních uzlech*. Ukážeme, že takový polynom existuje a je dán jednoznačně. Dále budeme studovat, jak dobře tento polynom aproximuje danou funkci i mimo uzly.

Označíme $\mathcal{P}_n$ množinu všech polynomů stupně nejvýše n. Každý polynom $q_n \in \mathcal{P}_n$ lze napsat jako lineární kombinaci polynomů z báze $\mathcal{P}_n$. Chceme najít jednoduchou bázi, která bude nějakým způsobem souviset s danými interpolačními uzly, konkrétně takovou, aby j-tý bázový polynom ℓ_j splňoval $\ell_j(x_i) = \delta_{ji}$. Tuto vlastnost mají Lagrangeovy polynomy

$$\ell_j(x) = \prod_{\substack{k=1 \\ k \neq j}}^{n} \frac{x - x_i}{x_j - x_i}\,.$$

Libovolný polynom q_n pak lze vyjádřit v této bázi,

$$q_n(x) = \sum_{i=0}^{n} a_i \ell_i(x).$$

Dosadíme-li za x uzel interpolace x_j, lehce spočítáme, že $a_j = f(x_j)$. Jinak řečeno hledaný interpolační polynom, který budeme nazývat *Lagrangeův interpolační polynom*, můžeme vyjádřit ve tvaru

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

Nalezli jsme polynom, který nabývá v uzlech x_j hodnot f_j. Ukážeme, že tento polynom je určen jednoznačně.

Věta 1.1. *Existuje právě jeden polynom $L_n \in \mathcal{P}_n$ takový, že $L_n(x_j) = f(x_j)$, kde $(x_j, f(x_j))$, $j = 0, \ldots, n$ jsou zadané body a hodnoty.*

Důkaz. Existence takového polynomu plyne z konstrukce výše. Ukážeme jednoznačnost.

Nechť $M_n \in \mathcal{P}_n$ také splňuje $M_n(x_j) = f(x_j)$, potom rozdíl $R_n = L_n - M_n \in \mathcal{P}_n$ a $R_n(x_j) = 0$ pro všechny $j = 0, \ldots, n$. Tedy R_n má alespoň $n + 1$ kořenů. Protože však $R_n \in \mathcal{P}_n$, musí být nutně identicky nulový, odtud potom $L_n = M_n$. □

Nyní máme jednoznačně určený interpolační polynom L_n. Chtěli bychom pro něj nalézt jiná vyjádření, která budou vhodnější pro vyhodnocování nebo umožní snadno přidávat nové interpolační uzly.

Označíme-li

$$\ell(x) := \prod_{i=0}^{n} (x - x_i), \qquad \lambda_j := \frac{1}{\ell'(x_j)},$$

můžeme psát

$$\ell_j(x) = \prod_{i \neq j}^{n} \frac{x - x_i}{x_j - x_i} = \frac{\ell(x)}{\ell'(x_j)(x - x_j)} = \frac{\ell(x)\lambda_j}{x - x_j}. \tag{1.2}$$

Získáme tak nové vyjádření $L_n(x)$ nazývané *modifikovaná Lagrangeova formule.*

$$L_n(x) = \ell(x) \sum_{i=0}^{n} f(x_i) \frac{\lambda_i}{x - x_i}. \tag{1.3}$$

Výhodou této formule je cena výpočtu $L_n(x)$; napočítáme-li si váhy λ_j ($O(n^2)$ operací), stojí nás potom výpočet hodnoty pro každé x jen $O(n)$ operací oproti $O(n^2)$ operacím pro každé x při počítání pomocí formule (1.1). Navíc pro vhodné rozložení bodů jsou váhy (1.3) známé, popřípadě lehce dopočitatelné.

Jelikož pro všechny $g \in \mathcal{P}_n$ platí $g = \sum_{i=0}^{n} g(x_i)\ell_i$, získáme pro $g \equiv 1$ vyjádření

$$1 = \sum_{i=0}^{n} \ell_i(x) = \ell(x) \sum_{i=0}^{n} \frac{\lambda_i}{x - x_i}.$$

Odtud $\ell(x) = 1/\sum_{i=0}^{n} \frac{\lambda_i}{x-x_i}$ a dosazením do (1.3) dostaneme

$$L_n(x) = \frac{\sum_{i=0}^{n} f(x_i)\frac{\lambda_i}{x-x_i}}{\sum_{i=0}^{n} \frac{\lambda_i}{x-x_i}}. \tag{1.4}$$

Toto vyjádření je označováno jako *barycentrická formule*.

Další způsob jak spočíst L_n je tzv. *Newtonova formule*. Označíme $f[x_0,\ldots,x_n]$ koeficient u x^n polynomu L_n interpolujícího funkci f z $C[a,b]$ v $n+1$ různých bodech $x_0,\ldots,x_n$, potom platí následující tvrzení. Poznamenejme, že koeficient $f[x_0,\ldots,x_n]$ se nazývá *poměrná diference*.

Věta 1.2. *Nechť $L_n \in \mathcal{P}_n$ splňuje $L_n(x_j) = f(x_j)$ $j = 0 \ldots n$, potom*

$$L_{n+1}(x) = L_n(x) + \prod_{i=0}^{n}(x - x_i) f[x_0,\ldots,x_{n+1}]\,,$$

kde $L_{n+1}(x_j) = f(x_j),\ j = 0,\ldots,n+1$.

Důkaz. Lze nalézt v [5, str.48, Theorem 5.2] □

Z tvrzení Věty 1.2 plyne indukcí

$$L_n(x) = f(x_0) + (x - x_0)f[x_0,x_1] + \ldots. + \prod_{i=0}^{n-1}(x - x_i) f[x_0,\ldots,x_n]. \tag{1.5}$$

Poznamenejme, že poměrnou diferenci lze jednoduše spočíst podle vzorce

$$f[x_j,\ldots,x_{j+k+1}] = \frac{f[x_{j+1},\ldots,x_{j+k+1}] - f[x_j,\ldots,x_{j+k}]}{x_{j+k+1} - x_j}\,,$$

viz například [5, str.50, Theorem 5.3]. Výhodou vyjádření (1.5) je možnost přidávat další body – interpolační uzly – bez změny předchozích koeficientů.

Věnujme se nyní aproximačním vlastnostem Lagrangeova interpolačního polynomu L_n. Zajímá nás, nakolik námi nalezený polynom L_n odpovídá hledané funkci f na celém intervalu a jak můžeme ovlivnit přesnost aproximace. Přímo z definice interpolačního polynomu plyne rovnost L_n a f ve všech uzlech interpolace. Relevantní otázkou je, zda s rostoucím n získáme lepší a lepší aproximace dané funkce. Jinak řečeno, zajímá nás, zda L_n konverguje stejnoměrně k f. Není však těžké najít funkci, pro kterou bude rozdíl $|f(x) - p(x)| > K$ pro všechna K z $\mathbb{R}$ a $x \neq x_j$, například funkce

$$f(x) = \begin{cases} K & x \neq x_j, j = 0,...,n\,, \\ -1 & \text{jinak}\,. \end{cases}$$

Vidíme tedy, že bez jakýchkoliv dodatečných předpokladů nejsme schopni říci nic o chybové funkci

$$e_n(x) := f(x) - L_n(x)\,,$$

a to dokonce ani pro spojité funkce.

Kvalitu aproximace zcela jistě ovlivňuje rozložení uzlů. Pokud by například byly všechny uzly (až na krajní) soustředěny v blízkosti jednoho bodu, můžeme

lokálně dostat velmi přesnou aproximaci, nicméně ve zbytku intervalu nemáme o chování funkce f téměř žádnou informaci. Stejně tak je důležitý počet uzlů a tedy i stupeň polynomu – je zbytečné aproximovat lineární funkci polynomem vysokého stupně, naopak pro funkci která kmitá bude potřeba uzlů více. Informace o kvalitě aproximace dostaneme analýzou chybové funkce e.

Chceme-li odhad chyby e_n, musíme klást nějaké požadavky na aproximovanou funkci f.

Pokud předpokládáme spojitost, můžeme chybovou funkci psát pomocí poměrných diferencí zmíněných výše. Mějme interpolační polynom L_n stupně n, přidáme další bod x_{n+1}. Jelikož lze za x_{n+1} zvolit libovolný bod x různý od předchozích interpolačních uzlů, platí díky Větě 1.2

$$f(x) = L_{n+1}(x) = L_n(x) + \prod_{i=0}^{n}(x - x_i) f[x_0, \ldots, x_{n+1}] \,,$$

(využíváme faktu, že $f(x_{n+1}) = L_{n+1}(x_{n+1})$). Celkem dostaneme [3, str. 245-248]

$$e_n(x) = f(x) - L_n(x) = \prod_{i=0}^{n}(x - x_i) f[x_0, \ldots, x_n, x] \,.$$

Předpokládáme-li navíc, že $f \in C^{n+1}[a,b]$, opakovanou aplikací Rolleovy věty dostáváme pro chybovou funkci vyjádření

$$e_n(x) = \frac{f^{(n+1)}(\xi_x)}{(n+1)!}\ell(x) \qquad \xi_x \in (a,b) \,.$$

Měníme-li rozložení bodů x_j, mění se hodnota ξ_x a člen $\ell(x)$. Chování $\ell(x)$ můžeme podstatně ovlivnit volbou uzlů, protože se jedná o monický polynom s kořeny právě v uzlech $x_0, \ldots, x_n$. I když rozložení uzlů ovlivní chování $\frac{f^{(n+1)}(\xi_x)}{(n+1)!}$, nedokážeme odhadnout, jak se tento člen změní. Předpokládáme-li rozumné chování $(n+1)$-ní derivace, potom je pro chování chyby podstatný člen $|\ell(x)|$.

Chybu Lagrangeovy interpolace lze pro analytické funkce vyjádřit ve tvaru [2, str. 67, sec 3.6]

$$e_n(x) = \frac{1}{2\pi \mathrm{i}} \int_\Gamma \frac{\ell(x) f(t)}{\ell(t)(t - x)} \mathrm{d}t \,,$$

kde Γ je jednoduchá uzavřená křivka obsahující body x_j ve svém vnitřku. Odtud je vidět, že chyba v závislosti na n záleží na poměru $\frac{\ell(x)}{\ell(t)}$ pro pevné libovolné x. Díky tomuto vyjádření lze teoreticky studovat chování L_n pro různá rozložení uzlů. Toto vyjádření chyby je rozebíráno také v [9, Kapitola 11].

1.2 Čebyševovy body

Výše jsme zmínili, že pro chování chyby je podstatný člen $|\ell(x)|$. Jeho velikost je ovlivněna volbou interpolačních uzlů. Chceme-li, aby chyba $e_n(x)$ byla malá na celém intervalu, potom je rozumnou strategií zvolit takové interpolační uzly x_j, pro které bude $|\ell(x)|$ minimální. V této sekci zavedeme Čebyševovy polynomy a Čebyševovy body. Dále ukážeme, že Čebyševovy body jsou ony hledané

interpolační uzly, pro které nabývá $|\ell(x)|$ nejmenší odchylky od nuly. Nakonec ukážeme, jak lze pro tuto volbu uzlů zjednodušit formuli určující interpolant.

Uvažujme zobrazení z $[-1,1]$ na $[0,\pi]$ přiřazující x hodnotu $\phi = \arccos(x)$, odtud máme vztah $x = \cos\phi$. Uvažujme na $[0,\pi]$ funkce $1, \cos(\phi), \cos(2\phi), \ldots, \cos(k\phi)$. Těm odpovídá sada funkcí $T_0(x) = 1$, $T_1(x) = \cos(\arccos(x)) = x$, $T_2(x) = \cos(2\arccos(x))$, $\ldots$, $T_k(x) = \cos(k\arccos(x))$. Použijeme součtové vzorce:

$$\cos((k+1)\phi) + \cos((k-1)\phi) = 2\cos(\phi)\cos(k\phi)\,,$$

odtud

$$T_{k+1}(x) + T_{k-1}(x) = 2xT_k\,.$$

Odvodili jsme rekurentní formuli pro výpočet T_{k+1} a zároveň vidíme, že T_{k+1} jsou polynomy pro všechna $k \in \mathbb{N}\cup\{0\}$ a vedoucí koeficient T_{k+1} je 2^k. Takto získanou posloupnost nazveme *posloupností Čebyševových polynomů*.

Vlastnosti Čebyševových polynomů nahlédneme přes transformaci $x = \cos(\phi)$. Pro kořeny polynomu T_k platí $0 = \cos(k\arccos(x))$, upravíme na $k\arccos(x) = \frac{(2j-1)\pi}{2}$ pro $j = 1,\ldots,k$. Další úpravou dostaneme vyjádření kořenů

$$x_j = \cos\left(\frac{(2j-1)}{k}\frac{\pi}{2}\right)\,.$$

Pro extrémy platí z definice $\max_{x\in[-1,1]}|T_k| = 1$ a těchto extrémů se střídavě nabývá v bodech

$$y_j = \cos\left(\frac{j}{k}\pi\right),\ j = 0,\ldots,k\,. \tag{1.6}$$

Čebyševovy polynomy jsou ortogonální s váhou $w(x) = \frac{1}{\sqrt{1-x^2}}$ ve smyslu

$$\int_{-1}^{1} T_k(x)T_j(x)\frac{1}{\sqrt{1-x^2}}\mathrm{d}x = \int_0^\pi \cos(k\phi)\cos(j\phi)\mathrm{d}\phi = \begin{cases} 0 & k \neq j\,,\\ \pi & k = j = 0\,,\\ \frac{\pi}{2} & k = j \neq 0\,. \end{cases}$$

Zajímavou (a užitečnou) vlastnost Čebyševových polynomů popisuje následující věta.

Věta 1.3. *Monický polynom $W_k = \frac{1}{2^{k-1}}T_k$ minimalizuje maximum na $[-1,1]$ přes všechny monické polynomy z $\mathcal{P}_k$.*

Důkaz. Nechť existuje V_k monický tak, že $\max|V_k| < \max|W_k|$ na $[-1,1]$. Víme, že W_k nabývá extrémů v bodech $x_j = \cos\left(\frac{j}{k}\pi\right)$, $j = 0,\ldots,k$ a že v těchto bodech střídá znaménko. Potom nutně musí platit $V_k(x_0) < W_k(x_0)$, $V_k(x_1) > W_k(x_1), \ldots$. Tedy polynom $(W_k - V_k)$ mění k krát znaménko, tedy má k kořenů, to ale není možné, protože $(W_k - V_k)$ je stupně nejvýše $k-1$ (jedná se o monické polynomy). □

Zvolme nyní na $[-1,1]$ interpolační uzly jako kořeny Čebyševova polynomu T_{k+1}, tedy $x_j = \cos\left(\frac{(2(n-j)-1)}{n+1}\frac{\pi}{2}\right)$, $j = 0,\ldots,n$, seřazeny vzestupně podle velikosti. Potom je dle Věty 1.3 $\ell(x) = \frac{1}{2^n}T_{n+1}$ monický polynom s nejmenší odchylkou od nuly.

Výše zmíněné body se nazývají *Čebyševovy body prvního druhu*. V praxi se pro uzly interpolace používají více tzv. *Čebyševovy body druhého druhu* (dále jen Čebyševovy body), volené jako extrémy Čebyševova polynomu T_{k+1} namísto kořenů, viz (1.6). Tyto body mají analogické aproximační vlastnosti a navíc obsahují krajní body intervalu. *Čebyševovým interpolantem* funkce f potom nazveme polynom interpolující f v Čebyševových bodech.

Dokážeme si nyní dvě lemmata, pomocí nichž odvodíme jednoduché vyjádření Čebyševova interpolantu založené na barycentrické formuli.

Lemma 1.4. *Nechť* $x_j = \cos\left(\frac{j}{n}\pi\right),\ j = 0,\dots,n$ *jsou Čebyševovy body na* $[-1,1]$. *Potom pro* $0 \le m \le n$ *platí:*

$$T_m(x_j) = T_{2n-m}(x_j) = T_{2n+m}(x_j),\ j = 0,\dots,n\,. \tag{1.7}$$

Důkaz. Z definice si napíšeme m-tý Čebyševův polynom $T_m(x)$ a za x dosadíme Čebyševovy body:

$$T_m(x) = \cos(m\arccos(x)),\ T_m(x_j) = \cos\left(m\frac{j}{n}\pi\right)\,.$$

Totéž zopakujeme pro polynomy T_{2n-m} a T_{2n+m}:

$$T_{2n-m}(x_j) = \cos\left((2n-m)\frac{j}{n}\pi\right) = \cos\left(-m\frac{j}{n}\pi\right) = \cos\left(m\frac{j}{n}\pi\right)\,,$$

$$T_{2n+m}(x_j) = \cos\left((2n+m)\frac{j}{n}\pi\right) = \cos\left(m\frac{j}{n}\pi\right)\,.$$

Porovnáním výsledků dostáváme požadované tvrzení. □

Vlastnosti (1.7) říkáme *aliasing*. Konkrétně pro $m = n-1$ dostaneme

$$T_{n+1}(x_j) = T_{n-1}(x_j),\ j = 0,\dots,n\,.$$

Potom $\frac{1}{2^n}(T_{n+1}(x) - T_{n-1}(x))$ je monický polynom stupně $n+1$, který nabývá nuly v $n+1$ bodech, tedy je určen jednoznačně. Jelikož $\ell(x)$ je také monický a $\ell(x_j) = 0$, musí platit

$$\ell(x) = \frac{1}{2^n}(T_{n+1}(x) - T_{n-1}(x))\,.$$

Z $\ell_j(x) = \frac{\ell(x)\lambda_j}{x-x_j}$ dostaneme dosazením $\ell_j(x) = \frac{\lambda_j}{2^n}\frac{T_{n+1}(x)-T_{n-1}(x)}{x-x_j}$, dále, opět dosazením, získáme

$$\lambda_j = \frac{1}{\ell'(x_j)} = \frac{2^n}{(T_{n+1}(x_j))' - (T_{n-1}(x_j))'}\,.$$

Na základě tohoto vyjádření lze nyní dokázat následující lemma.

Lemma 1.5. *Pro Čebyševovy body je*

$$\lambda_j = \begin{cases} (-1)^j\frac{2^{n-1}}{n} & j = 1,\dots,n-1\,, \\ \frac{1}{2}(-1)^j\frac{2^{n-1}}{n} & j = 0,n\,. \end{cases} \tag{1.8}$$

Důkaz. Viz [9, str.34, Theorem 5.2]. □

Pomocí tohoto výsledku snadno dokážeme následující větu.

Věta 1.6. *Čebyševův interpolant funkce f na $[-1,1]$ v $n+1$ bodech je dán vztahem:*

$$L_n(x) = \frac{\sum'_{j=0}{}^n f(x_j)\frac{(-1)^j}{x-x_j}}{\sum'_{j=0}{}^n \frac{(-1)^j}{x-x_j}}\,, \tag{1.9}$$

kde $\sum'$ je suma, jejíž první a poslední člen je přenásoben $\frac{1}{2}$.

Důkaz. Ihned plyne z (1.4) (str. 4) dosazením. □

Z tvaru (1.9) je dobře vidět výhodnost barycentrické formule – výpočet $L_n(x)$ stojí pro každé x pouze $O(n)$ operací. Navíc se vyhneme počítání s velkými faktory $\frac{2^{n-1}}{n}$, které jsou společné pro všechny λ_j a díky tvaru barycentrické formule se vykrátí.

Zamysleme se nad tím, jak ovlivní lineární transformace $[-1,1]$ na $[a,b]$ formuli (1.9). Uvažujme přirozenou transformaci $y = \frac{x(b-a)+b+a}{2}$ pro $y \in [a,b]$, $a,b \in \mathbb{R}$. Dosadíme-li do obecného vzorce pro λ_i za $y_i = \frac{x_i(b-a)+b+a}{2}$, dostaneme po úpravě

$$\lambda_i = \left(\frac{2}{b-a}\right)^{n-1} \frac{1}{\prod_{j\neq i}^n (x_i - x_j)}\,.$$

To odpovídá λ_i na intervalu $[-1,1]$ přenásobené společným faktorem $\left(\frac{2}{b-a}\right)^{n-1}$. Můžeme ho tedy vytknout před sumy ve formuli (1.4) (str. 4) a zkrátit. Odtud je vidět, že vyjádření (1.9) lze použít i pro Čebyševův interpolant na obecném intervalu $[a,b]$, kde za x_j dosadíme lineárně transformované body y_j.

1.3 Čebyševovy řady

Čebyševovy polynomy tvoří množinu funkcí ortogonálních vzhledem ke skalárnímu součinu

$$\langle f,g\rangle := \int_{-1}^{1} f(x)g(x)\frac{1}{\sqrt{1-x^2}}\mathrm{d}x\,.$$

Ortogonální projekce funkce f do této množiny (uvažujeme-li prvních $n+1$ Čebyševových polynomů) je potom prvkem nejlepší aproximace funkce f (z $\mathcal{P}_n$) v indukované normě. Chtěli bychom vědět, kdy je funkce f součtem projekcí do jednotlivých prvků této množiny. Potom bychom jí mohli, podobně jako Taylorův rozvoj, zapsat sumou Čebyševových polynomů s vhodnými koeficienty. O tomto rozvoji hovoří následující věta.

Věta 1.7 (Čebyševova řada)**.** *Jestliže je funkce f Lipschitzovská na intervalu $[-1,1]$, lze ji jednoznačně vyjádřit pomocí Čebyševovy řady*

$$f(x) = \sum_{i=0}^{\infty} a_i T_i(x)\,, \tag{1.10}$$

která je absolutně stejnoměrně konvergentní s koeficienty danými vztahy

$$a_k = \frac{2}{\pi}\int_{-1}^{1} \frac{f(x)T_k(x)}{\sqrt{1-x^2}}\mathrm{d}x$$

pro $k > 0$, *pro* $k = 0$ *je koeficient* $\frac{2}{\pi}$ *zaměněn za* $\frac{1}{\pi}$.

Důkaz. Zavedeme modul spojitosti

$$\omega(f,[a,b],\delta) := \sup_{x_1,x_2\in[a,b];|x_1-x_2|\leq\delta} |f(x_1) - f(x_2)|\,.$$

Potom Dini-Lipschitzovo kritérium (viz například [6, str. 135]) říká: Jestliže spojitá funkce f na intervalu I splňuje

$$\lim_{n\to\infty} (\log n)\,\omega\left(f, I, \frac{1}{n}\right) = 0\,, \tag{1.11}$$

potom částečné součty Fourierovy řady (v našem případě Čebyševovy řady) konvergují stejnoměrně k f na I. Speciálně, je-li f Lipschitzovská s konstantou L, platí

$$\omega\left(f,[a,b],\frac{1}{n}\right) \leq \frac{1}{n}L$$

a Dini-Lipschitzovo kritérium je splněno.
Pro důkaz pomocí komplexní analýzy viz [9, str.15, Theorem 3.1.]. □

Každou Lipschitzovskou funkci lze vyjádřit Čebyševovou řadou (1.10). Částečný součet této řady nazveme *Čebyševovou projekcí*

$$f_n(x) := \sum_{i=0}^{n} a_i T_i(x)\,. \tag{1.12}$$

Z předchozího plyne, že jednoznačně určený Čebyševův interpolant lze vyjádřit v bázi Čebyševových polynomů:

$$p_n(x) = \sum_{i=0}^{n} c_i T_i(x)\,.$$

Zajímá nás kvalita aproximace obou těchto polynomů. Vztah f_n k původní funkci f je jasný. Z Věty 1.7 plyne, že pro Lipschitzovskou f bude f_n stejnoměrně konvergovat k f. Výpočet Čebyševovy projekce však bude náročný kvůli vyhodnocování integrálů v koeficientech a_k. Naproti tomu p_n umíme díky (1.9) vyhodnotit v $O(n)$ operacích, ale nevíme zatím nic o jeho konvergenci s rostoucím n. Chceme porovnat aproximační vlastnosti f_n a p_n. Pomocí Lemmatu 1.4 (str. 7) a následujícího tvrzení o vztahu koeficientů c_i a a_i ukážeme chování chybových funkcí $f - p_n$ a $f - f_n$.

Věta 1.8. *Nechť* f *je Lipschitzovská na* $[-1,1]$, p_n *je její Čebyševův interpolant v* $\mathcal{P}_n$, $n \geq 1$, c_i *a* a_i *jsou koeficienty jako výše. Potom*

$$c_0 = a_0 + a_{2n} + a_{4n} + \dots\,,$$

$$c_n = a_n + a_{3n} + a_{5n} + \dots\,,$$

a pro $1 \leq k \leq n-1$,

$$c_i = a_i + (a_{i+2n} + a_{i+4n} + \dots) + (a_{-i+2n} + a_{-i+4n} + \dots)\,.$$

Důkaz. Čebyševova řada stejnoměrně absolutně konverguje k f. Můžeme tedy její členy libovolně přerovnat. Vztahy pro $c_j,\ j = 0, \ldots, n$ tak jednoznačně dávají reálná čísla, jelikož odpovídají přerovnané Čebyševově řadě vyhodnocené v $x = 1$. Suma Čebyševových polynomů $T_0, \ldots, T_n$ s koeficienty $c_0, \ldots, c_n$ je rovna polynomu stupně n. Ten nabývá stejných hodnot jako f v Čebyševových bodech, zde využíváme Lemmatu 1.4 (str. 7), tedy se jedná o jednoznačně určený polynom $p_n \in \mathcal{P}_n$. □

Jako důsledek dostáváme absolutně konvergentní řady pro $f - f_n$ a $f - p_n$:

$$f(x) - f_n(x) = \sum_{i=n+1}^{\infty} a_i T_i(x), \tag{1.13}$$

$$f(x) - p_n(x) = \sum_{i=n+1}^{\infty} a_i (T_i(x) - T_{m_{i,n}}(x)), \tag{1.14}$$

kde $m_{i,n} = |(i + n - 1) (\mathrm{mod}\ 2n) - (n - 1)|$. Hodnota „prvního“ $m_{1,n}$ je potom $n - 1$, druhého $n - 2$ a tak dále až k $m_{n,n} = 0$, následující hodnoty jsou 1; 2; 3; ... a oscilují s periodou $2n$.

Hrubými odhady $||f - f_n|| \leq \sum_{i=n+1}^{\infty} |a_i|$ a $||f - p_n|| \leq \sum_{i=n+1}^{\infty} 2|a_i|$ dostáváme faktor poměru chyb 2

$$||f - p_n|| \leq 2||f - f_n||,$$

ukážeme též ve Větě 1.11 (str. 11), důkladnější analýzou dostaneme dokonce faktor $\frac{\pi}{2}$ viz [9, Theorem 16.1]. Jinak řečeno, p_n je až na (rozumnou) konstantu stejně kvalitní aproximace jako f_n, navíc je levnější na výpočet. Teoretické vlastnosti Čebyševova interpolantu tak lze studovat skrze Čebyševovy projekce.

1.4 Konvergence

V úvodu jsme zmínili Weierstrassovu větu. Ta nám zaručuje existenci polynomu, který danou spojitou funkci libovolně přesně stejnoměrně aproximuje. Zajímá nás, jak se budou chovat aproximace f_n a p_n obecné spojité funkce pro n jdoucí k nekonečnu na pevně dané množině interpolačních uzlů. Lze dokázat následující tvrzení:

Nechť máme pro každé n daný systém $n + 1$ různých interpolačních uzlů na intervalu $[a,b]$. Potom existuje $f \in C([a,b])$ taková, že $||f - p_n||_\infty$ diverguje pro $n \to \infty$, kde p_n je interpolantem funkce f v daných uzlech. Tento fakt dokázal Faber v roce 1914. Lze ho odvodit například jako důsledek Kharshiladze-Lozinskiho vět, viz [1, str.214-215].

Naštěstí pro většinu funkcí, mimo patologické případy na hranici spojitosti, Čebyševovy interpolanty stejnoměrně konvergují (dostatečně rychle pro praktické použití). Jak uvidíme v následujícím, se zvyšující se hladkostí funkce f roste i rychlost stejnoměrné konvergence Čebyševových řad a interpolantů pro $n \to \infty$. Za dostatečně hladké funkce pro praktické použití můžeme považovat například funkce f splňující Dini-Lipschitzovo kritérium (1.11).

Označme $p_n^* \in \mathcal{P}_n$ nejlepší stejnoměrnou aproximaci funkce f. Z teorie aproximace víme, že p_n^* existuje a je dána jednoznačně ($\mathcal{P}_n$ splňuje tzv. *Haarovu*

podmínku). Jacksonovy věty nám dávají rychlost konvergence $O(n^{-\nu})$ pro nejlepší aproximaci za předpokladu, že je funkce f ν-krát spojitě diferencovatelná. Pro mnoho funkcí dostáváme experimentálně (pro Čebyševovy interpolanty) konvergenci o jeden řád rychlejší. To lze pozorovat v případě, kdy derivace řádu $\nu+1$ sice není spojitá, ale má omezenou totální variaci. O tomto chování Čebyševových interpolantů hovoří následující věty.

Věta 1.9. *Označme $V \in \mathbb{R}$ totální variaci funkce f a nechť f má omezenou totální variaci. Pak pro koeficienty a_k, $k \geq 1$, Čebyševovy řady funkce f platí*

$$|a_k| \leq \frac{2V}{\pi k}\,.$$

Věta 1.10. *Uvažujme spojitou funkci f na intervalu $[-1,1]$. Nechť f má absolutně spojité derivace řádu $1,\ldots,\nu-1$, $\nu \in \mathbb{N}$. Označme $V \in \mathbb{R}$ totální variaci $f^{(\nu)}$ a nechť $f^{(\nu)}$ má omezenou totální variaci. Potom pro $k \geq \nu+1$ splňují koeficienty a_k Čebyševovy řady funkce f*

$$|a_k| \leq \frac{2V}{\pi k(k-1)\cdots(k-\nu)} \leq \frac{2V}{\pi(k-\nu)^{\nu+1}}\,.$$

Důkaz. Viz [9, str.48, Theorem 7.1.]

□

Odtud a z Věty 1.7 (str. 8) dostáváme odhad chyby Čebyševova interpolantu a projekce.

Věta 1.11. *Za předpokladů Věty 1.10 s $\nu \geq 1$ platí pro všechna $n > \nu$*

$$||f-f_n||_\infty \leq \frac{2V}{\pi\nu(n-\nu)^\nu}, \qquad ||f-p_n||_\infty \leq \frac{4V}{\pi\nu(n-\nu)^\nu}\,.$$

Důkaz. Chybu $||f-f_n||_\infty$ vyjádřenou ve tvaru (1.13) (str. 10) odhadneme sumou absolutních hodnot a dosadíme odhad pro koeficienty a_k z Věty 1.10 a dostaneme

$$||f-f_n||_\infty \leq \sum_{k=n+1}^{\infty} |a_k| \leq \frac{2V}{\pi} \sum_{k=n+1}^{\infty} \frac{1}{(k-\nu)^{\nu+1}}\,.$$

Tuto sumu můžeme odhadnout integrálem

$$\int_n^\infty \frac{1}{(x-\nu)^{\nu+1}}\mathrm{d}x = \frac{1}{\nu(n-\nu)^\nu}\,.$$

Pro $||f-p_n||_\infty$ použijeme (1.14) (str. 10) a analogický postup s $2|a_k|$ místo $|a_k|$.

□

Z výše dokázaného je přímo vidět, že má-li ν-tá derivace f omezenou totální variaci, potom je rychlost konvergence pro $n \to \infty$ řádu $O(n^{-\nu})$.

1.5 Lebesgueova konstanta

Uvažujme nyní operátor $\mathcal{A}$ přiřazující funkci f její interpolační polynom p_n daný formulí (1.1) (str. 3). Z jejího tvaru je přímo vidět, že $\mathcal{A}$ je **lineární**, navíc je **projekcí** na $\mathcal{P}_n$: pro $q \in \mathcal{P}_n$ máme $\mathcal{A}(q) = g$, g je polynom stupně n, který se rovná q v $n+1$ bodech, tedy q, g jsou identické. Na vlastnosti polynomiální interpolace tedy můžeme nahlížet skrze vlastnosti operátoru $\mathcal{A}$.

Při popisu konvergence, či možné divergence, polynomiálních interpolantů hraje důležitou roli tzv. *Lebesgueova konstanta* Λ. Tu definujeme jako ∞ - normu operátoru $\mathcal{A}$,

$$\Lambda := \sup_{||f||_\infty = 1} ||\mathcal{A}(f)||_\infty \,.$$

Lebesgueova konstanta nám říká, jaké maximální hodnoty může nabývat polynomiální interpolant funkce z jednotkového kruhu v $C([-1,1])$.

Z formule (1.1) (str. 3) můžeme Lebesgueovu konstantu vyjádřit pomocí Lagrangeových polynomů ℓ_j. Definujme *Lebesgueovu funkci* pro danou množinu x_j bodů z intervalu $[-1,1]$ jako

$$\lambda(x) := \sum_{i=0}^{n} |\ell_i(x)| \,.$$

Lemma 1.12. *Uvažujme $f \in C([-1,1])$, $||f||_\infty \leq 1$. Hodnota Lebesgueovy funkce je v každém bodě $x \in [-1,1]$ větší nebo rovna hodnotě interpolantu $p_n(x)$ k funkci f. Hodnota Lebesgueovy konstanty je maximum Lebesgueovy funkce na $[-1,1]$,*

$$\Lambda = \sup_{x \in [-1,1]} \lambda(x) \,.$$

Důkaz. Z (1.1) (str. 3) pro libovolné pevné $x \in [-1,1]$ dosáhneme maximální hodnoty $p_n(x)$, pokud f bude nabývat v bodech x_j hodnot ± 1 se stejným znaménkem jako má $\ell_j(x)$. Supremum hodnot $\lambda(x)$ na $[-1,1]$ je potom (z definice normy operátoru) Λ. □

Lebesgueova konstanta je významná při zkoumání vlastností interpolantu. Je-li totiž Lebesgueova konstanta dostatečně „malá“, potom je interpolant „dobrý“. Jestliže Λ je Lebesgueova konstanta pro interpolant p funkce f (bez újmy na obecnosti uvažujme $||f||_\infty \leq 1$) na dané množině interpolačních uzlů, potom $||p||_\infty \leq \Lambda$. Tedy $||f - p||_\infty$ může být rovna $\Lambda + 1$. Velká Lebesgueova konstanta potom znamená možnost velké chyby interpolace.

Pomocí chyby nejlepší aproximace a Lebesgueovy konstanty můžeme odhadnout chybu interpolantu.

Věta 1.13. *Nechť Λ je Lebesgueova konstanta lineární projekce $\mathcal{L}$ z $C([-1,1])$ do $\mathcal{P}_n$. Nechť $f \in C([-1,1])$, $p = \mathcal{L}(f)$ je interpolant a p^* je nejlepší polynomiální aproximace f. Potom*

$$||f - p|| \leq (\Lambda + 1)||f - p^*|| \,.$$

Důkaz. Uvažujme funkci $f-p^*$. Aplikujeme na ni operátor $\mathcal{L}$: $\mathcal{L}(f-p^*)=p-p^*$, tedy $p-p^*$ je interpolantem k $f-p^*$. Z definice Lebesgueovy konstanty máme:

$$||p-p^*|| \leq \Lambda ||f-p^*|| \,.$$

Dále platí $f-p=(f-p^*)-(p-p^*)$, po znormování

$$||f-p|| \leq ||(f-p^*)|| + ||(p-p^*)|| \leq (1+\Lambda)||(f-p^*)|| \,.$$

□

Uveďme ještě jedno tvrzení, hovořící obecněji o lineárních projekcích. To ukazuje, že Čebyševova interpolace je téměř optimální.

Věta 1.14. *Nechť $\mathcal{L}$ je omezená lineární projekce z $C([-1,1])$ do $\mathcal{P}_k$. Potom platí*

$$||\mathcal{L}|| \geq \frac{1}{2}||\mathcal{R}_0 + \mathcal{R}_k|| \,,$$

kde $\mathcal{R}_k$ přiřazuje funkci f její Čebyševovu projekci f_k.

Důkaz. Lze nalézt v [5, str.208, Theorem 17.4] □

Ukázali jsme, že operátor přiřazující funkci interpolant je lineární projekcí. Věta 1.13 nám dává odhad přesnosti interpolace v závislosti na Lebesgueově konstantě a chybě nejlepší aproximace. Jinými slovy, máme-li malou Λ, potom má námi zkonstruovaný interpolant normu chyby blízkou k normě chyby nejlepší aproximace a lze předpokládat, že je dobrou aproximací. Jak může být Λ malá? Jak se Λ_n chová pro $n \to \infty$? Odpovědi na tyto otázky nám prozradí mnoho o kvalitě Čebyševových interpolantů a projekcí.

Následující věta shromažďuje poznatky z celého předešlého století a jasně vymezuje chování polynomiálních interpolantů na Čebyševových a ekvidistantně dělených bodech pomocí Lebesgueovy konstanty.

Věta 1.15. *Lebesgueova konstanta Λ_n pro polynomiální interpolant stupně $n \geq 0$ na libovolné množině $n+1$ uzlů na intervalu $[-1,1]$ splňuje*

$$\Lambda_n \geq \frac{2}{\pi}\log(n+1) + 0.52125\ldots \,,$$

kde $0.52125\ldots$ je $\frac{2}{\pi}\left(\gamma + \log\frac{4}{\pi}\right)$, číslo $\gamma \approx 0.577$ je Eulerova konstanta.

Pro Čebyševovy body platí

$$\Lambda_n \leq \frac{2}{\pi}\log(n+1)+1 \qquad a \qquad \Lambda_n \sim \frac{2}{\pi}\log(n), \qquad n \to \infty \,.$$

Pro ekvidistantně dělené body platí

$$\Lambda_n > \frac{2^{n-2}}{n^2} \qquad a \qquad \Lambda_n \sim \frac{2^{n+1}}{en\log n}, \qquad n \to \infty \,,$$

s první nerovností pro $n \geq 1$.

Už jsme zmínili, že neexistuje sada bodů taková, aby polynomiální interpolant na těchto bodech konvergoval pro všechny spojité funkce. Věta 1.15 říká, že rychlost divergence Lebesgueovy konstanty je alespoň logaritmická. Zároveň nám však pro Čebyševovy body logaritmický růst Λ_n zaručuje i horním odhadem. Získáváme tak odpovědi na výše položené otázky: Pro Čebyševovy body je Λ_n i v nejhorším případě shora omezena rozumnou mezí a roste pomaleji než libovolný polynom – totiž logaritmicky. Pro ilustraci, Čebyševův interpolant v 100001 bodech nejhorších možných dat (spojitá funkce, v supremové normě menší nebo rovna jedné) je polynom stupně 100000 s maximální hodnotou menší než 9,3. Znamená to sice chybu velikosti přes 8, nicméně jedná se o nejhorší možný případ a polynom poměrně vysokého stupně. Naproti tomu pro ekvidistantní dělení máme z Věty 1.15 možnu divergenci Λ_n dokonce exponenciální. Stejné maximální hodnoty jako pro Čebyševovy body dosáhneme už pro 12 bodů! Navíc z Věty 1.13 dostaneme horní odhad chyby Čebyševova interpolantu: 9,3 násobek chyby nejlepší aproximace, tedy méně než jedno platné desetinné místo. Ztrátu přesnosti dvou cifer nepřekročíme ani pro polynom stupně 10^{66}. Pokud bychom měli nejlepší aproximaci p^* na úrovni strojové přesnosti ε a znali její stupeň n, potom Čebyševův interpolant zkonstruovaný na $n+1$ bodech by měl přesnost v nejhorším případě $10^2\varepsilon \approx 10^{-14}$.

Kapitola 2

Chebfun – numerické počítání s funkcemi

V předchozí kapitole jsme popsali teorii Čebyševových projekcí a Čebyševových interpolantů. Nyní představíme software využívající tuto teorii k aproximaci funkcí a pomocí něho ilustrujeme některé výsledky předchozí kapitoly. Dále provedeme několik drobných pozorování a nakonec se zaměříme na použitelnost tohoto softwaru v různých oblastech matematiky.

2.1 Idea toolboxu Chebfun

Chebfun je open source toolbox procedur rozšiřující MATLAB o možnost numerického počítání s funkcemi. Přesněji řečeno, Chebfun rozšiřuje MATLABovské příkazy pro počítání s vektory a maticemi na operace s funkcemi a operátory.

Zavádí proměnnou typu *chebfun*, což je struktura, v níž jsou uloženy vhodné parametry pro reprezentaci funkce jedné proměnné definované na intervalu $[a,b]$ pomocí polynomu. Přetížení MATLABovských příkazů nám umožňuje pracovat s proměnnou chebfun podobně jako s vektorem. Například `sum(f)` vrátí součet složek vektoru, pokud `f` je vektor. Pokud je `f` chebfun, dostaneme použitím `sum(f)` hodnotu určitého integrálu příslušné funkce na intervalu $[a,b]$. Chebfun v současné době není možné používat ve freewarovém Octave (který má syntax kompatibilní s MATLABem). Vývojáři Chebfunu zatím neplánují upravit toolbox tak, aby byl použitelný s jiným programem než MATLAB.

Implementace Chebfunu je založena na aproximaci funkce Čebyševovým interpolantem. K dosažení maximální přesnosti aproximace v dané aritmetice počítače je pro každou funkci potřeba jiný počet interpolačních uzlů. Jednoduché funkce stačí interpolovat v desítkách bodů, složitým funkcím nemusí stačit desetitisíce. Ke zjištění ideálního počtu bodů se využívá adaptivní algoritmus, který nastaví počet uzlů automaticky bez nutnosti zásahu ze strany uživatele.

V následujících kapitolách budeme používat výraz „*počítačově přesný*“ ve významu: přesný na maximální počet cifer v rámci dané aritmetiky. V double precision to znamená 15 platných cifer. Počítačově přesná aproximace potom neznamená chybu velikosti $\varepsilon \approx 10^{-15}$, ale relativní chybu vzhledem k velikosti funkčních hodnot. Pokud je například hodnota přesného řešení nějaké úlohy řádu 10^3, je spočtený výsledek lišící se o 10^{-12} považovatelný za počítačově přesný.

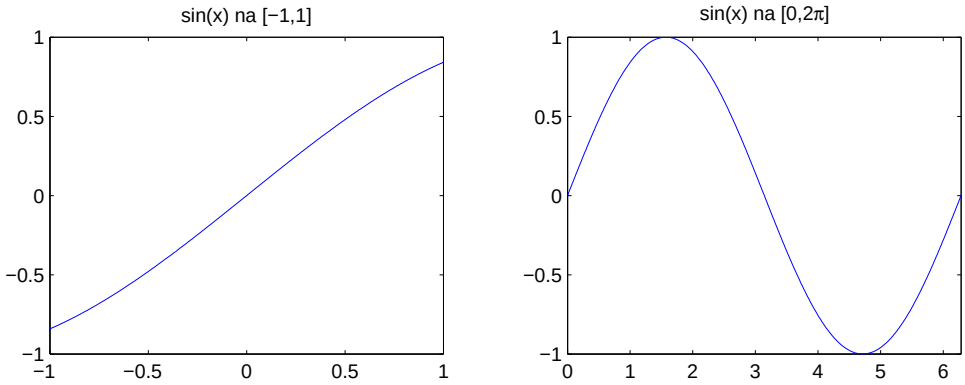
Velikost chyby je sice $\approx 10^{-12}$ nicméně jsme dosáhli maximální možné přesnosti v dané aritmetice.

2.2 Stručný průvodce

Zakladním a nejdůležitějším příkazem je `chebfun`, který vytvoří proměnnou typu chebfun ze vstupních dat. Těmi může být anonymní funkce, například `@(x) gamma(x)` nebo řetězec (string), například `'sin(15*x)'`. Anonymní funkcí zde rozumíme jakýkoliv objekt, který se chová jako funkce. Jednoduchou anonymní funkci si můžeme napsat i na řádce: `f = @(x) x.^2;`. Funkce f vrací druhou mocninu zadaného čísla, voláme ji jako klasickou funkci jedné proměnné: `f(v)`, `v` je vektor hodnot. Část `@(x)` udává jaký je vstupní parametr funkce – proměnnou. Pomocí ní a běžných operací definujeme libovolnou funkci. Můžeme také využít jednu z mnoha předdefinovaných funkcí jako v příkladu výše, kde používáme funkci gamma odpovídající $(x-1)!$ pro x přirozené.

Dalším nepovinným parametrem je interval na jakém chceme funkci aproximovat. Výchozí interval je nastaven na $[-1,1]$. Rozdíl v zadání funkce $\sin(x)$ na intervalech $[-1,1]$ a $[0,2\pi]$ je následující:

```
f = chebfun('sin(x)');
ff = chebfun('sin(x)',[0,2*pi]);
subplot (1,2,1), plot(f), title ('sin(x) na [-1,1]')
subplot (1,2,2), plot(ff), title ('sin(x) na [0,2\pi]')
```

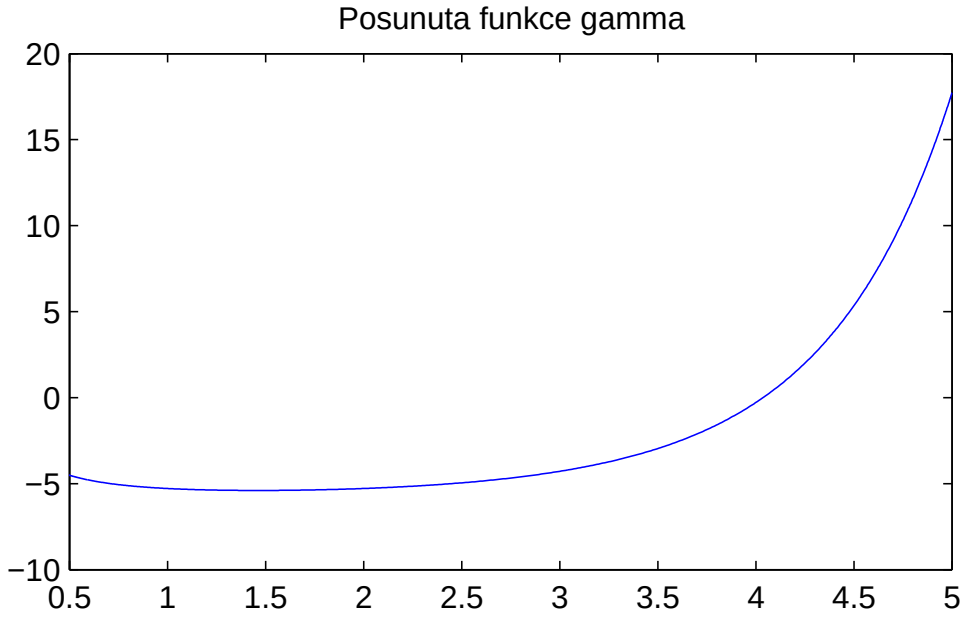


V chebfun-proměnné `f` nyní máme uloženou funkci se kterou budeme dále pracovat. Můžeme ji například vykreslit příkazem `plot(f)`. MATLAB může tento příkaz správně provést, přestože je `f` jiného typu než klasický `plot` vyžaduje. Toho je dosaženo přetížením příkazu, které spočívá v napsání nové procedury se stejným jménem jaké používá MATLAB. Při zavolání `plot(f)` se podle typu proměnné `f` volá buď procedura původní, nebo procedura nová – když je proměnná typu chebfun. Pokud si necháme vypsat výstup (vynecháním středníku) dozvíme se na jakém intervalu se pohybujeme, jakého stupně je polynom aproximující zadanou funkci, hodnoty v krajních bodech a velikost chyby aproximace:

```
f =
   chebfun column (1 smooth piece)
        interval       length   endpoint values
[       -1,        1]        14      -0.84      0.84
Epslevel = 1.110223e-15.  Vscale = 8.414710e-01.
```

Použijeme-li k vykreslení příkaz `plot(f,'.-')` uvidíme vyznačené hodnoty v Čebyševových bodech. Křivka mezi nimi je počítána pomocí barycentrické formule (1.9) (str. 8). Jak jsme zmínili výše, příkaz `sum(f)` vrátí hodnotu určitého integrálu z f přes interval $[-1,1]$. Dále uvedeme příklad s anonymní funkcí, tentokrát na uživatelem zvoleném intervalu [0.5,5]:

```
g = chebfun(@(x) gamma(x)-2*pi,[.5,5]);
```



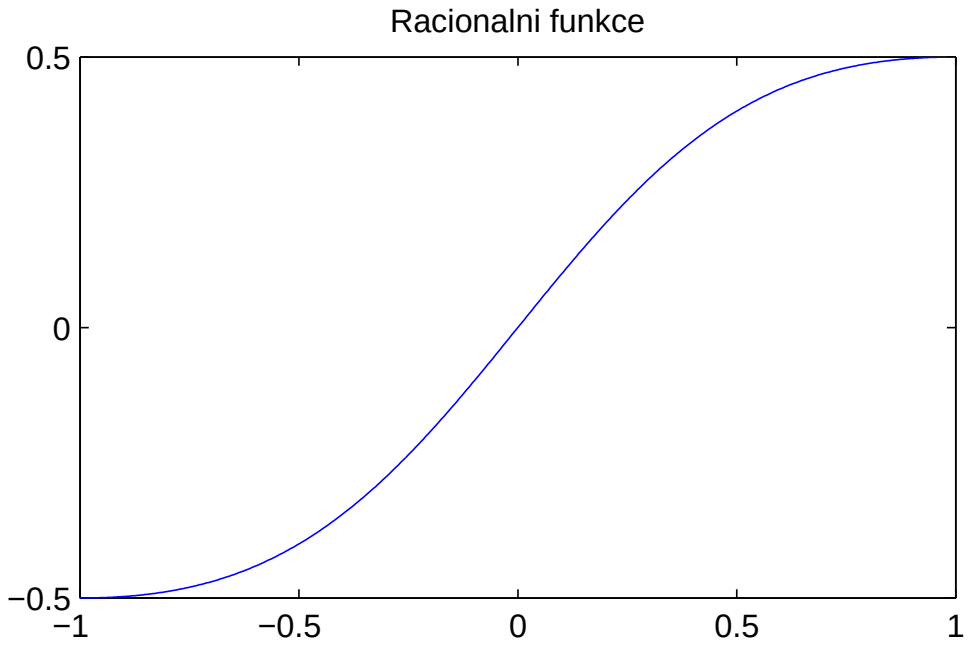
Pokud bychom chtěli najít kořeny této funkce, stačí zadat `r = roots(g)`. Příkaz `roots` hledá kořeny pomocí vlastních čísel tzv. *colleague matice*, podrobněji se o tomto postupu zmíníme ve třetí kapitole. Takto jednoduše jsme vyřešili netriviální úlohu nalezení kořenů funkce, která je definovaná integrálem. Nutno upozornit, že výsledek je spočítán numericky, ve standardní double precision aritmetice. Dosadíme-li však hodnoty do posunuté gamma funkce `gamma(r).-2*pi`, uvidíme, že se pohybujeme na úrovni počítačové přesnosti. Samozřejmě nás zajímá rychlost výpočtu. Pomocí dvojice příkazů `tic` a `toc` zjistíme, jak dlouho trvá vytvořit chebfun-proměnnou a spočítat kořeny gamma funkce. Dále se ještě podíváme na přesnost výpočtu dosazením `r` do posunuté funkce gamma:

```
clear
tic; g = chebfun(@(x) gamma(x)-2*pi,[0,5]); r = roots (g); toc
Elapsed time is 0.651924 seconds.

gamma(r)-2*pi
ans =
    1.7764e-15
```

Další možnost jak získat chebfun[1] je pomocí jiného chebfunu. Nadefinujeme si „proměnnou“ x: `x = chebfun('x')` (jedná se vlastně o lineární funkci $f(x) = x$) a pomocí ní definujeme chebfun. Pokud bychom chtěli pracovat s funkcí $\frac{x}{1+x^2}$, můžeme použít syntax:

[1]Proměnná typu chebfun je autory toolboxu často označovaná pouze jako chebfun, s malým ch. Toolbox je označován Chebfun s velkým Ch.

```
x = chebfun('x');
h = x./(1+x.^2);
```



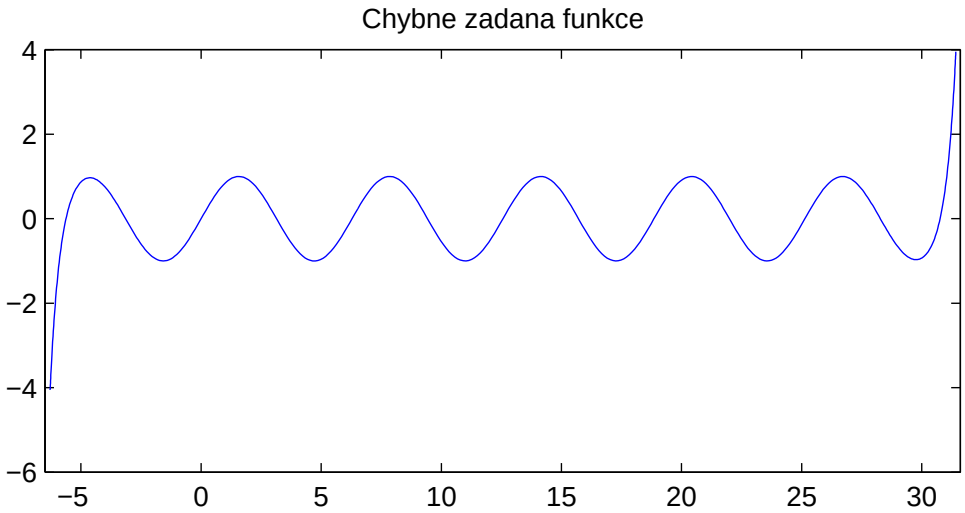
Tento způsob je ekvivalentní zadání `h = chebfun('x./(1+x.^2)');`. Proměnné typu chebfun se vždy konstruují stejným způsobem, bez závislosti na způsobu zadání. Nemusíme se omezovat jen na operace sčítání, násobení a dělení. Stejným způsobem bychom mohli vytvořit i náš první příklad:

```
f = sin(x);
```

Při tomto způsobu zadávání je třeba dát si pozor na tečku při zápisu mocnění, dělení a dalších operací, které MATLAB nemá definované pro vektory. Umístění tečky před takovou operaci říká, že se má pracovat po složkách. Dále musíme pozorně pracovat s intervaly, na kterých jsou chebfuny definované. Máme-li dva chebfuny `f` a `g`, oba na stejném intervalu, můžeme je mezi sebou sčítat, násobit, mocnit a podobně. Pokud jsou `f` a `g` definované na jiných intervalech, tyto operace použít nemůžeme. Chceme-li i přesto takovou operaci provést, musíme sáhnout k chebfunu složenému z více částí, viz níže. Používáme-li `x = chebfun('x')`, všechny chebfuny vytvořené pomocí `x` budou na intervalu $[-1,1]$. Chceme-li používat jiný interval, je nutné zadat na začátku:
`x = chebfun('x',[požadovaný_interval])`.
Předvedeme příklad špatného použití intervalů. Nejdříve chceme pracovat s funkci $\sin(x)$ na intervalu $[0,8\pi]$, potom si to rozmyslíme a zkusíme rozšířit funkci na větší interval $[-2\pi,10\pi]$.

```
x=chebfun('x',[0,8*pi]);
f=chebfun(sin(x),[-2*pi,10*pi]);
```

Vše vypadá v pořádku, nadefinujeme si „proměnnou“ x a pokusíme se vyrobit $\sin(x)$ ovšem na větším intervalu. Dostaneme tak:



Stalo se to, že Chebfun spočítal Čebyševův interpolant pro $\sin(x)$ na intervalu $[0,8\pi]$. Do takto získaného polynomu následně dosadil hodnoty $[-2\pi,10\pi]$ a ty vykreslil. Výsledek je vlastně výpočetně správný (na $[0,8\pi]$ odpovídá funkci $\sin(x)$), od požadovaného se ovšem výrazně liší.

Teď, když umíme sestrojit chebfun, se podíváme na operace, které s ním můžeme provádět. Zadáním `methods chebfun` si necháme vypsat většinu `příkazů`. Co který příkaz dělá zjistíme pomocí `help chebfun/příkaz`.

Během objevování možností Chebfunu brzy narazíme na funkce které nejsou hladké. Chceme-li například pracovat s $|x|$, dostaneme:

```
f = chebfun('abs(x)');
    Warning: Function not resolved using 65537 pts. Have you tried
     'splitting on'?
    > In chebfun.constructor>constructorNoSplit at 114
      In chebfun.constructor at 63
      In chebfun.chebfun>chebfun.chebfun at 219
```

Nabízené nastavení „splitting on“ dovoluje vytvořit chebfun z funkcí které jsou po částech hladké. Každé této části se říká „*fun*“. Upravíme-li vstup na:

```
f = chebfun('abs(x)','splitting','on');
```

dostaneme chebfun sestávající ze dvou funů. Vynecháním středníku si necháme vypsat počet funů a pro každý z nich stejné informace jako v případě hladké funkce:

```
f =
   chebfun column (2 smooth pieces)
       interval       length   endpoint values
[      -1, 1.2e-17]        2         1        0
[ 1.2e-17,       1]        2  -2.2e-16        1
Epslevel = 1.110223e-15. Vscale = 1.000000e+00.  Total length = 4.
```

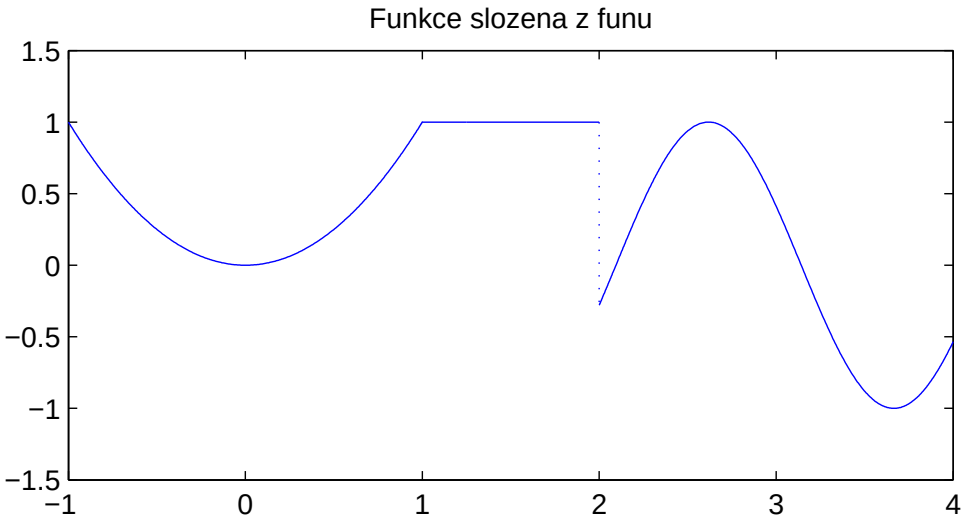
Funkce $|x|$ je důkladně studována v teorii aproximace[2]. Použijeme-li interpolaci polynomem na celém intervalu $[-1,1]$, dostaneme výrazně pomalejší konvergenci

[2]Stačí nahlédnout například do [1] nebo [9]

než pro interpolaci racionální funkcí. Rozdělení intervalu nám v tomto případě ještě více usnadní práci.

Po částech hladký chebfun můžeme také sestrojit přímo: do složené závorky oddělené čárkou zadáme hladké funkce, druhým argumentem jsou dělící body zapsané jako vektor.

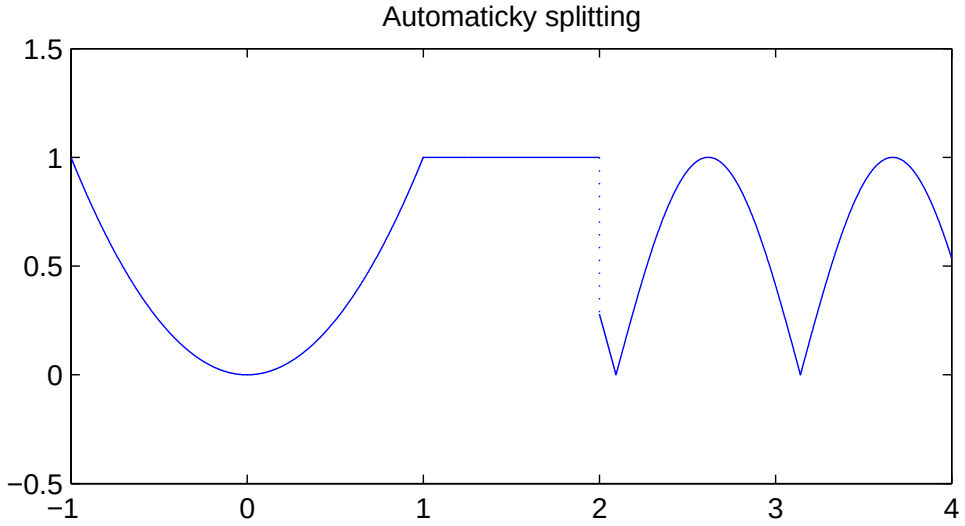
```
x = chebfun('x',[-1,4]);
f = chebfun({x.^2, 1, sin(3*x)},[-1 1 2 4]);
```



Vyrobili jsme funkci která odpovídá x^2 na $[-1,1]$, 1 na $[1,2]$ a $\sin(3x)$ na $[2,4]$.

Některé příkazy automaticky spustí hledání vhodných bodů pro rozdělení funkce na funy. Například `abs(f)` nejdříve najde kořeny `f` a v nich rozdělí zadanou funkci. Dalším takovým příkazem je `max(a,b)`, popřípadě `min(a,b)`. Použijeme `abs` na funkci f vytvořenou výše. Všimněme si, že nedostaneme chybovou hlášku.

```
h=abs(f);
```



Nutno ještě poznamenat, že ne vždy se vyplatí mít zapnuté dělení funkce. To lze pozorovat například na funkci $\sin(x)$ na intervalu $[0,1000\pi]$. Bez rozdělování dostaneme chebfun více jak 20 krát rychleji a potřebujeme celkově o třetinu méně interpolačních uzlů.

```
tic;f = chebfun('sin(x)',[0,1000*pi],'splitting','on'), toc

f = chebfun column (32 smooth pieces); Total length = 2748.
Elapsed time is 0.772135 seconds.

tic;f = chebfun('sin(x)',[0,1000*pi]), toc

f = chebfun column (1 smooth piece); Length = 1684.
Elapsed time is 0.035766 seconds.
```

2.3 Jak Chebfun vytváří chebfun

Data tvořící chebfun jsou koeficienty Čebyševovy řady (1.12) (str. 9) příslušné k zadané funkci. Hlavní problém je určit potřebný počet koeficientů n postačující k tomu, aby zadaná funkce byla pro počítač nerozlišitelná od Čebyševova interpolantu. Chceme mít n malé abychom neukládali zbytečně moc dat a neprováděli více operací, než je nezbytně nutné a zároveň dostatečně velké, abychom nepřicházeli o žádnou informaci a dostali co nejlepší aproximaci v rámci možností dané aritmetiky. Pro dané n Chebfun spočítá prvních n koeficientů Čebyševovy řady zadané funkce a zkoumá jejich velikost. Pokud několik z nich za sebou spadne pod hranici počítačové přesnosti, potom je toto n vyhodnoceno jako vyhovující a je provedeno případné odstranění přebytečných koeficientů. Tím upravíme – zmenšíme n. Chebfun postupně zkouší různá n, ty jsou z technických důvodů volena jako $n = 2^l + 1$. Během konstrukce je využíváno jednoznačného vztahu mezi hodnotami funkce v Čebyševových bodech a aproximacemi koeficientů Čebyševovy řady. Pomocí rychlé Fourierovy transformace (FFT) umíme rychle přejít od hodnot k aproximacím koeficientů[3]. Celý proces vypadá následovně: nejdříve Chebfun spočítá hodnoty funkce v 9 Čebyševových bodech, pomocí FFT spočte prvních 9 koeficientů Čebyševovy řady a zkontroluje, zdali jsou dostatečně malé. Jestliže je odpověď NE, Chebfun zkusí 17 bodů, potom 33, pak 65 a tak dále. Ve chvíli kdy je počet bodů vyhodnocen jako vhodný, odstraní se zbytek koeficientů, které jsou zanedbatelné – pod úrovní počítačové přesnosti. Získáme tak číslo k určující potřebný počet interpolačních bodů. Těm odpovídá k koeficientů Čebyševovy řady – těchto k čísel tvoří data definující konkrétní chebfun. Pro vykreslení by stačilo znát pouze číslo k a zadaný interval, uchování koeficientů však umožní dále pracovat s chebfunem (například hledat kořeny funkce). Pokud by nestačilo ani 65537 bodů, Chebfun se pokusí (pokud má tuto možnost povolenou) funkci rozdělit na více intervalů a na každém z nich aproximovat funkci zvlášť.

Vypišme si na ukázku koeficienty chebfunu odpovídajícího funkci $\cos(x)$. Ke zjištění koeficientů slouží příkaz `chebpoly`.

```
x = chebfun('x'); f = cos(x); a = chebpoly(f);
format long, a(end:-1:1)'
```

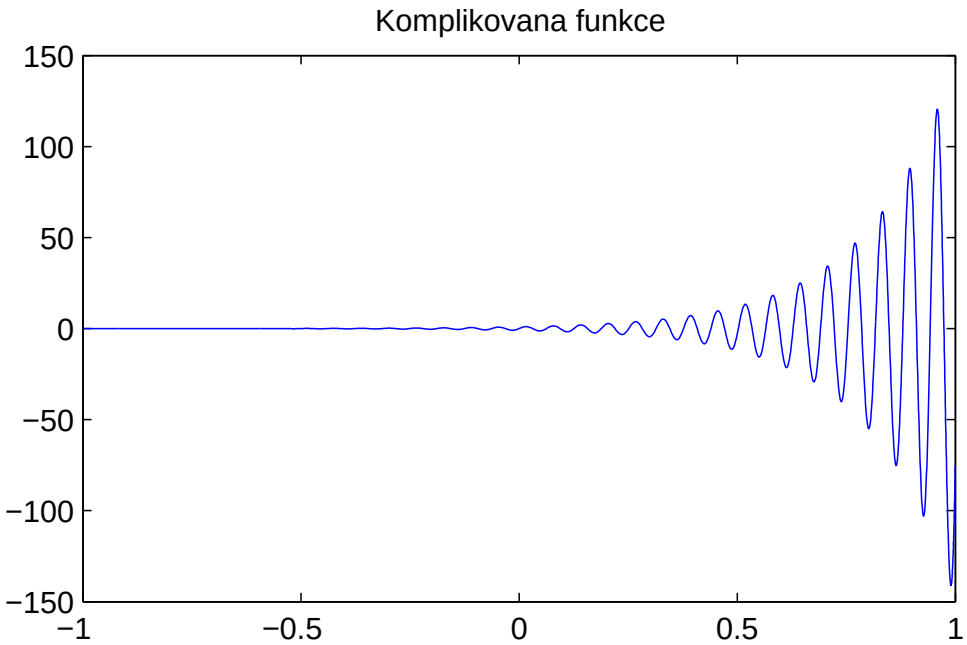
[3]Koeficienty jsou dány integrály. Ty nepočítáme přesně, ale pouze aproximujeme pomocí FFT.

```
ans =
    0.765197686557967
    0
   -0.229806969863801
    0
    0.004953277928220
    0
   -0.000041876676005
    0
    0.000000188446883
    0
   -0.000000000526123
    0
    0.000000000001000
    0
   -0.000000000000001
```

Všimněme si, že poslední koeficient (u T_{14}) už je na hranici počítačové přesnosti. Stačilo tedy vyzkoušet 9 a 17 bodů, výsledný polynom je stupně 14, použít všech 17 koeficientů by bylo zbytečné.

Pro složitější funkce by bylo nepřehledné vypisovat koeficenty. Lepší je vykreslit je do grafu. Ukažme jednu takovou komplikovanou funkci[4]

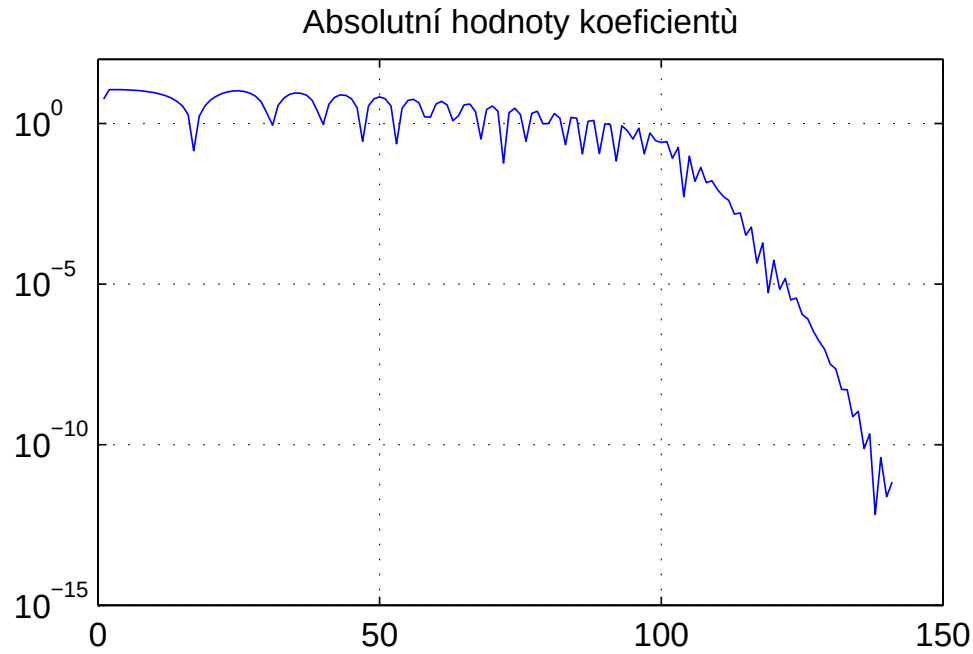
```
f = chebfun(exp(5*x).*sin(100*x));
```



Vykresleme si graf absolutních hodnot koeficientů:

```
a = chebpoly(f); semilogy(abs(a(end:-1:1)))
grid on, title('Absolutní hodnoty koeficientů')
```

[4]Vzali jsme rychle kmitající nekonečně hladkou funkci.



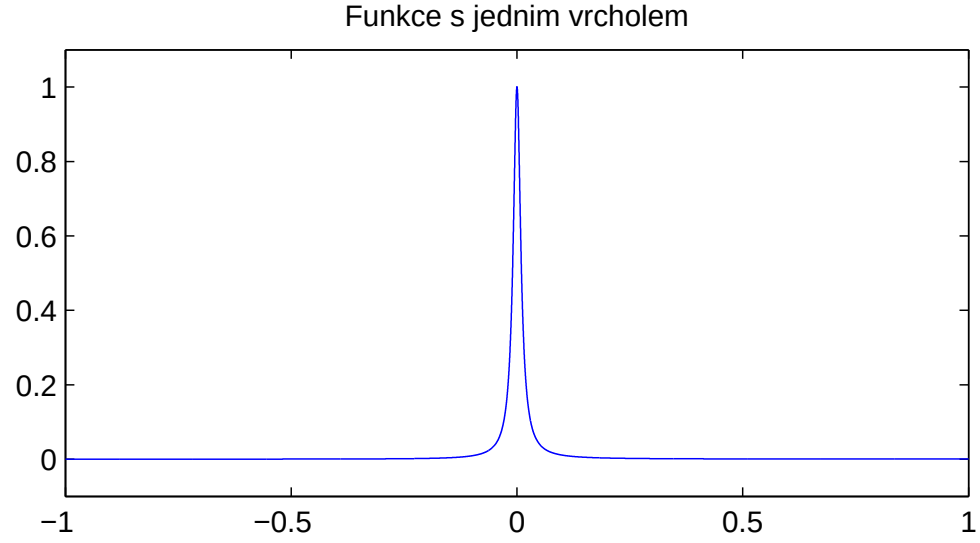
Z grafu vidíme, že až do stupně přibližně 100 jsou všechny koeficienty významné a nemají klesající tendenci. Musíme tedy vzít více koeficientů. Ty už rychle klesají k nule a kolem stupně 140 dosáhnou počítačové přesnosti. Přesnou hodnotu k získáme příkazem `length(f)`:

```
ans =
     141
```

Sestrojený chebfun je polynom na 141 bodech – tedy stupně 140. K tomuto výsledku bylo nutno spočítat koeficienty pro 257 bodů, Chebfun následně usoudil, že jsou již dostatečně malé a zanedbal koeficienty pro $k = 257, 256, \ldots, 142$.

Místo příkazů `chebpoly` a `semilogy` můžeme použít speciální Chebfunovský příkaz `chebpolyplot` se stejným efektem. To ukážeme na funkci s jedním vrcholem[5]:

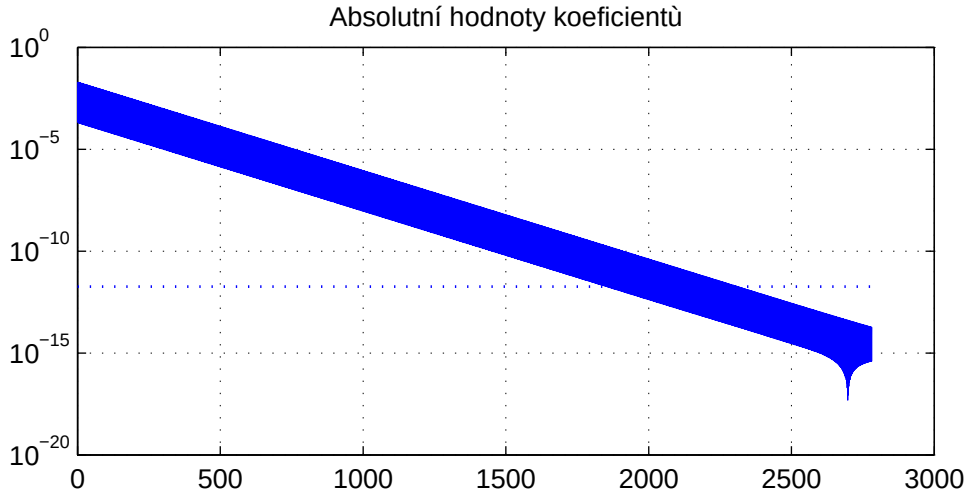
```
f = exp(x)./(1+10000*x.^2);
```



[5]Funkce ze cvičení v [9, Kapitola 3].

Její koeficienty vykreslíme:

```
chebpolyplot(f), grid on,
title('Absolutní hodnoty koeficientů')
```



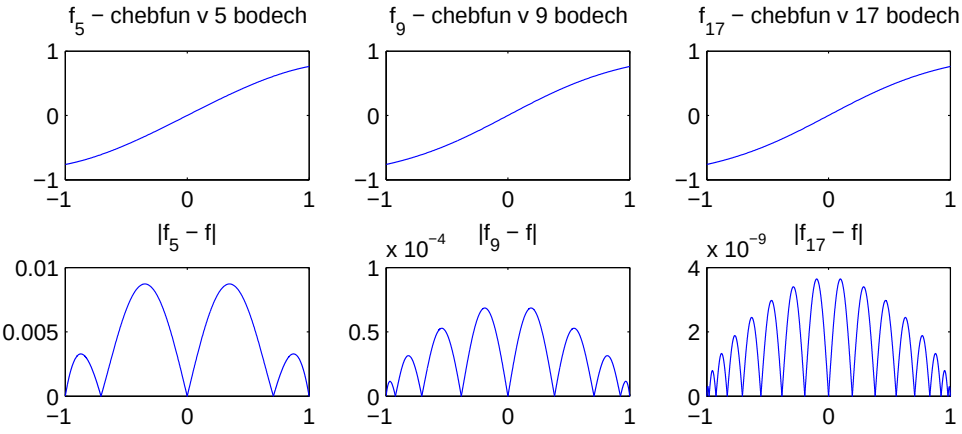
Přestože jde o napohled výrazně jednodušší funkci, pomocí `length(f)` zjistíme, že je potřeba 2783 bodů k dosažení uspokojivé přesnosti aproximace. Za to může chování jejích derivací.

Pro vyhodnocení chebfunu v konkrétním bodě je pak využita barycentrická formule (1.9) (str. 8), pro kterou stačí znát stupeň polynomu k. Stupeň polynomu můžeme také zadat ručně jako nepoviný parametr. Chceme-li aproximovat například funkci $\tanh(x)$ polynomem stupně k, stačí napsat:

```
f = chebfun('tanh(x)',k+1)
```

Zvolme $k = 4,8,16$, spočteme f_{k+1} – interpolanty v $k+1$ bodech a porovnejme výsledky s chebfunem f spočítaným v programem zvoleném počtu bodů.

```
f = chebfun('tanh(x)');
for n = 1 : 3
    pb = 2^(n+1)+1;
    g = chebfun('tanh(x)',pb);
    e = abs(f-g);
end
```

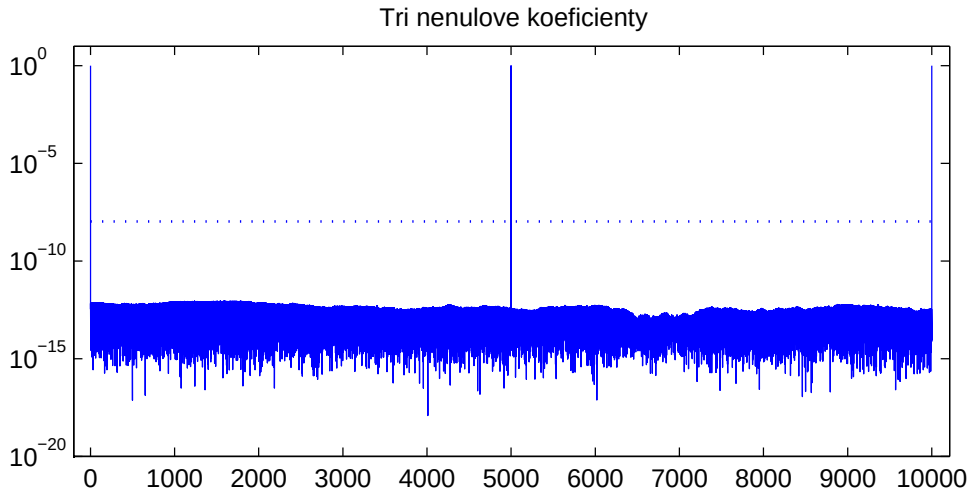


I když nepozorujeme žádný vizuální rozdíl, je chyba $|f_k - f|$ řádu postupně $10^{-2}, 10^{-4}, 10^{-9}$. Počet bodů Chebfunem považovaný za dostatečný je 30.

Vzhledem ke způsobu konstrukce chebfunu je přirozené si položit následující otázku. Co se stane, pokud od nějakého k budou koeficienty a_k zanedbatelně malé, ale velikost koeficientu $a_l, k \ll l$ bude srovnatelná s $\max\{a_1, \ldots, a_k\}$? Pokud bychom následovali výše popsaný postup, za potřebný počet interpolačních bodů by bylo vyhodnoceno k. Přišli bychom tedy o část informace a dostali bychom reprezentaci úplně jiné funkce. Udělejme tedy experiment s funkcí, pro kterou $a_1 = a_{5000} = a_{10000}$ a zbytek koeficientů je nulový:

```
p=chebfun('x+cos(5000*acos(x))+cos(10000*acos(x))');
```

a vykresleme velikost koeficientů:



Je jasně vidět, že byly použity všechny nenulové koeficienty. Chebfun pravděpodobně při vyhodnocování vhodného počtu bodů provádí zpětnou kontrolu. Jelikož Chebfun má k dispozici hodnoty zadané funkce f_j v Čebyševových bodech, není výpočetně náročné přejít od spočítaných koeficientů inverzní Fourierovou transformací zpět k hodnotám $\tilde{f}_j$. Potom je možné zkoumat velikost chyby $|f_j - \tilde{f}_j|$. Pokud je chyba velká, spočítali jsme aproximaci nepřesně a musíme vzít v úvahu více koeficientů.

Kapitola 3

Praktické ukázky využití softwaru Chebfun

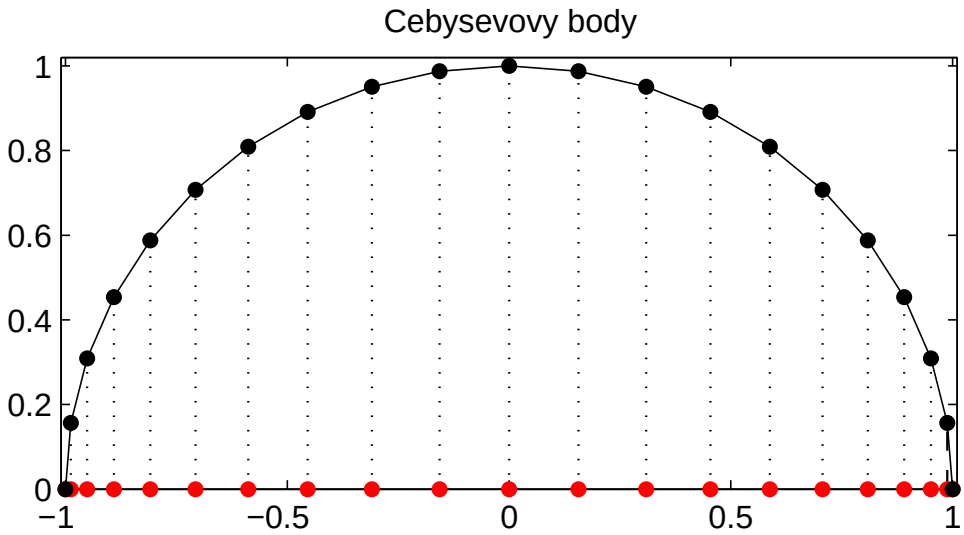
V poslední kapitole se zaměříme na možná využití softwaru Chebfun. V první řadě je to oblast teorie aproximací. Zde je Chebfun možné použít nejen k výzkumu ale také k výuce a s jeho pomocí demonstrovat dosažené výsledky. Na oficiálních stránkách programu `http://www.chebfun.org/examples/` jsou k nalezení zajímavé příklady z různých oblastí matematiky, například lineární algebry, obyčejných diferenciálních rovnic, statistiky nebo optimalizace. Dále je možné program použít například při hledání kořenů funkce nebo při řešení diferenciálních rovnic.

3.1 Obrázky ke kapitole 1

V této sekci použijeme Chebfun k ilustraci výsledků z první kapitoly. Začneme tím, že si necháme vykreslit Čebyševovy body a Čebyševovy polynomy.

Čebyševovy body jsou kolmou projekcí bodů rovnoměrně rozmístěných na jednotkové kružnici na osu x. Na spočítání samotných Čebyševových bodů má Chebfun příkaz `chebpts(počet_bodů)`. Vykreslíme si 21 Čebyševových bodů:

```
pom = linspace(0,pi,21);
bodykruh = cos(pom)+i*sin(pom);
chebbody = chebpts(21);
hold on
for j = 2:21
    plot ([chebbody(22-j) bodykruh(j)],'linewidth',0.7)
end
plot(chebbody,0*chebbody,'.')
plot(bodykruh,'.-')
```

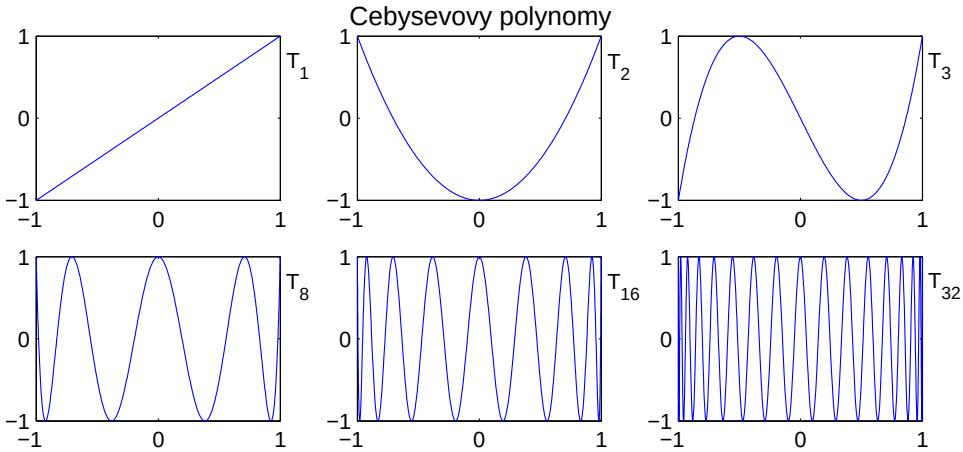


Z obrázku je patrné jejich hromadění u krajních bodů intervalu. Chebfun standardně používá Čebyševovy body druhého druhu. Kdybychom chtěli pracovat s Čebyševovými body prvního druhu, stačí zadat nepovinný parametr `chebkind(1)`. Je možné přenastavit výchozí typ bodů zadáním:

```
chebfunpref.setDefaults('tech',@chebtech1)
```

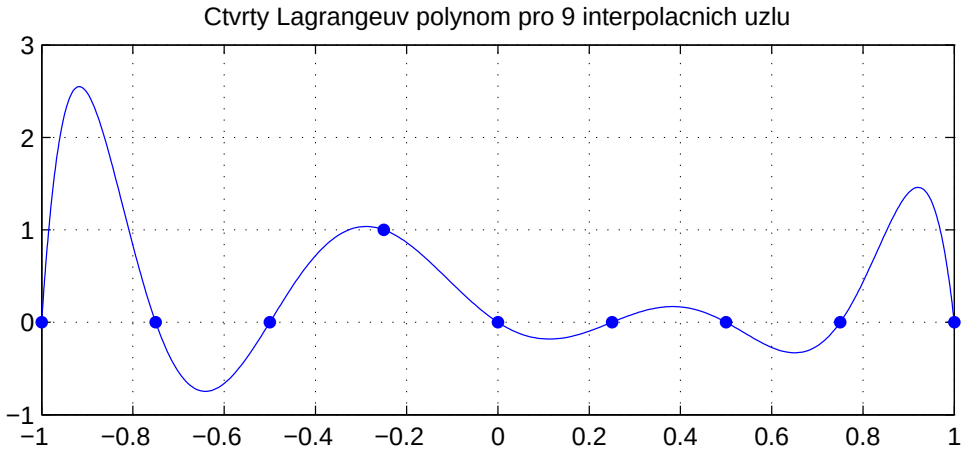
K vykreslení Čebyševových polynomů použijeme chebfunovský příkaz `chebpoly`. Vykreslíme si polynomy stupně 1, 2, 3, 8, 16 a 32.

```
for n = 1:3
    T = chebpoly(n);
    TT = chebpoly(2^(2+n));
    subplot (2,3,n), plot(T),
    subplot (2,3,3+n), plot(TT),
end
```



Tyto polynomy nabývají extrémů velikosti ± 1 v Čebyševových bodech. Pro porovnání si vykreslíme čtvrtý Lagrangeův polynom stupně 8 na devíti ekvidistantně dělených bodech.

```
d = domain(-1,1); s = linspace(-1,1,9); y= [0 0 0 1 0 0 0 0 0];
p = interp1(s,y,d);
plot(p), hold on, plot(s, p(s), '.')
```



Okamžitě vidíme, že maximální hodnota je větší než u odpovídajícího Čebyševova polynomu. Postupnou úpravou Lagrangeových polynomů jsme odvodili barycentrickou formuli (1.4) (str. 4), kterou jsme dále hojně využívali. Vyzkoušejme nyní experimentálně rychlost vyhodnocování Čebyševova interpolantu pomocí barycentrické formule. K tomuto účelu si sestrojíme velmi složitou funkci[1].

```
x = chebfun('x');
f = tanh(5*sin(20*exp(3*x)))./(3+sin(200*x).^3)+
                    +cos(3*x).*exp(4*sin(5*x))./(1+500*cos(x).^2);
length(f)
ans =
    35217
```

Nyní necháme vzniklý polynom vyhodnotit v 10 000 bodech, ty vykreslíme a změříme potřebný čas:

```
tic, plot(f,'numpts',10000,'.'); toc
Elapsed time is 0.028684 seconds.
```

Za takto krátkou dobu jsme schopni vyhodnotit polynom stupně 35 000 v 10 000 bodech. Zkusíme ručně nastavit stupeň polynomu na jeden milion[2] a podíváme se na změnu rychlosti:

```
p = chebfun(f, 1000001);
body = 2*rand(1,1000)-1;
tic, hodnoty = p(body); toc
Elapsed time is 1.617791 seconds.
```

Spočítali jsme takto tisíc hodnot polynomu stupně milion za přibližně jeden a půl vteřiny. Vidíme, že i pro opravdu velké stupně polynomů jsme pořád schopni počítat v řádu vteřin.

V sekci 1.3 porovnáváme chování p_n a f_n. Nyní si demonstrujeme, jak se tyto dva polynomy liší a jaký je vztah jejich chyb $p_n - f$ a $f_n - f$. K tomuto účelu nám poslouží jednodušší funkce $\sin(\mathrm{e}^{2x})$[3] (černě), k jejíž aproximaci Chebfun použije 49 bodů. My si zkonstruujeme Čebyševův interpolant

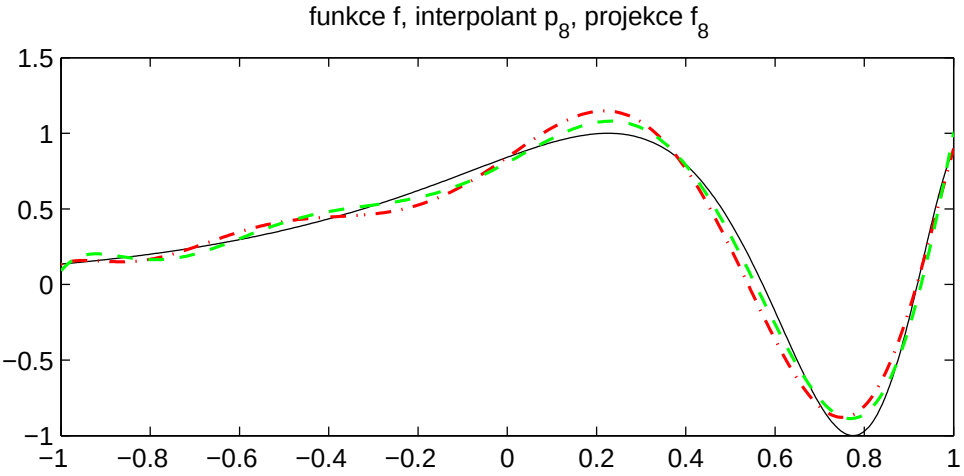
[1] Jde o komplikovanější variantu funkce použité v [9].

[2] Pokud zadáme počet bodů ručně, je možné obejít maximum 65537 bodů nastavené Chebfunem.

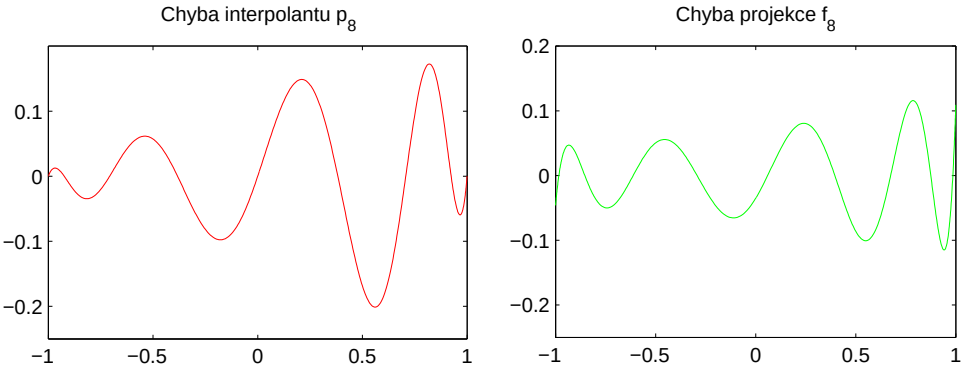
[3] Funkce, která se nejdříve mění pozvolně a postupně zrychluje své kmitání.

(červeně) a Čebyševovu projekci (zeleně), oboje stupně 8 a porovnáme přesnost aproximace a velikost chyby. K získání projekce použijeme nepovinný parametr `'trunc','počet_členů_řady'` příkazu `chebfun`.

```
f = sin(exp(2*x));
pn = chebfun(f,9);
fn = chebfun(f,'trunc',9);
```

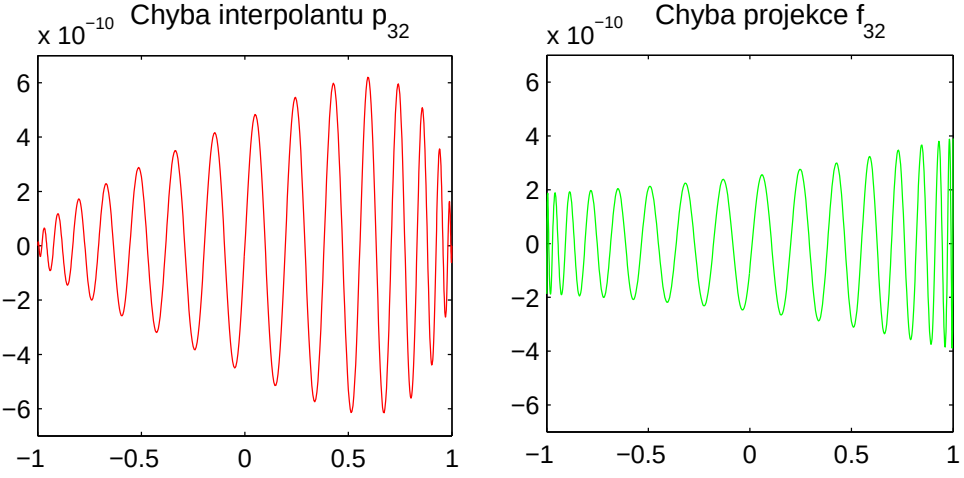


K tomu si vykreslíme obě chybové funkce $p_8 - f$ a $f_8 - f$.



Necháme si ještě vykreslit chybové funkce pro $p_{32} - f$ a $f_{32} - f$.

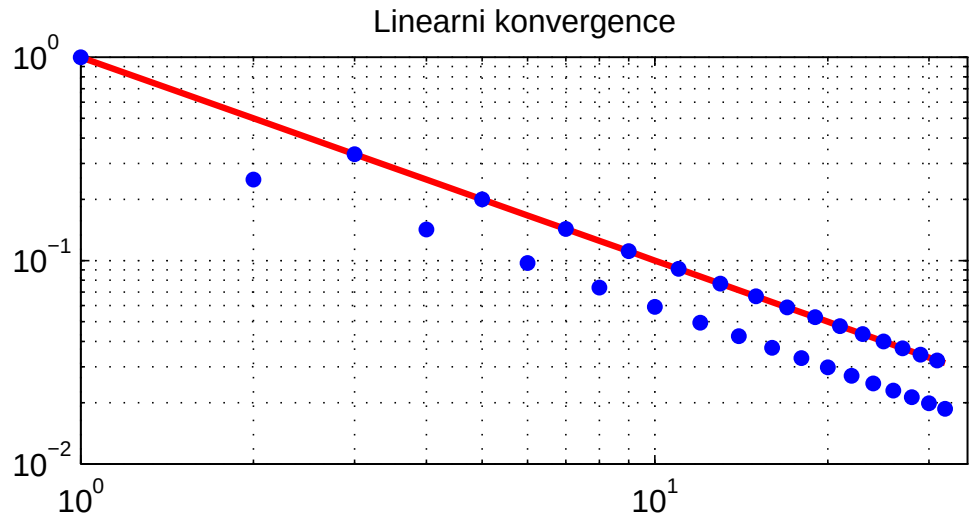
```
pn = chebfun(f,33);
fn = chebfun(f,'trunc',33);
```



Z obrázků jsou patrné o něco lepší aproximační vlastnosti projekce. Pro 33 interpolačních uzlů je však rozdíl v chybě aproximace velmi malý. Z druhého obrázku je také možné pozorovat poměr chyb $\frac{\pi}{2}$ zmiňovaný v první kapitole. Za povšimnutí stojí fakt, že obě chybové funkce kmitají téměř se stejnou frekvencí, liší se pouze hodnoty. To je důsledek skutečnosti, že Čebyševova interpolace je téměř nejlepší aproximací, jejíž chybová funkce kmitá (Čebyševova ekvioscilační věta).

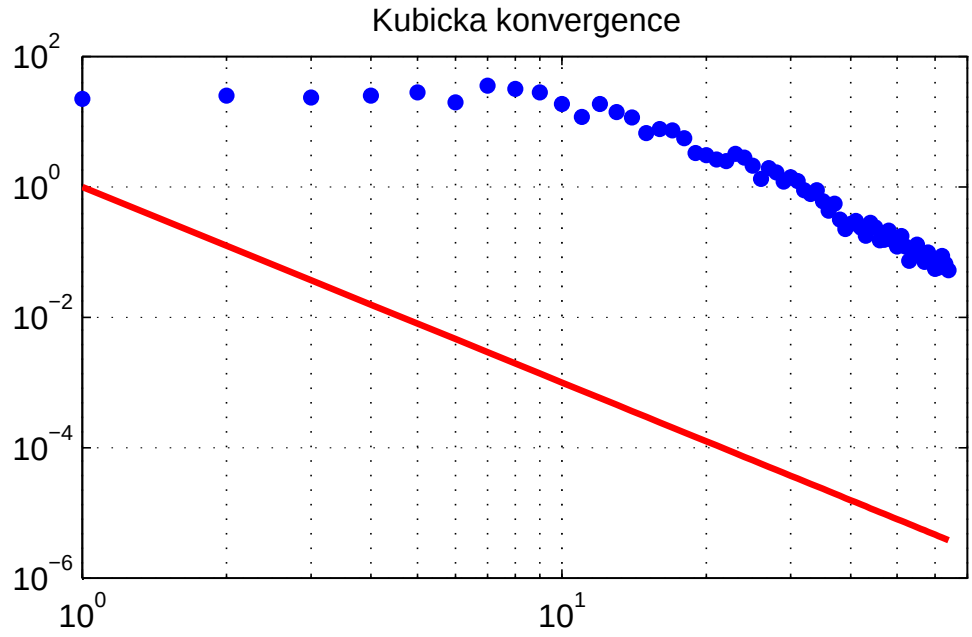
Dále jsme zkoumali rychlost konvergence. Ukážeme si lineární rychlost konvergence Čebyševova interpolantu funkce $|x|$.

```
f = abs(x); n = (1:32); err = [];
for i = 1:32
    pn = chebfun(f,i+1); err=[err, norm(f-pn,inf)];
end
```



Vykreslili jsme maximální chybu p_n pro n rovno 1 až 32 vůči n^{-1}. Vidíme, že pro sudá n rychlost konvergence přesně odpovídá n^{-1}, pro n lichá je chyba o něco málo menší, pokles je ovšem také lineární. Vezmeme nyní funkci, která má nespojitou třetí derivaci $f = |\mathrm{e}^{1.5x}\sin(10x)|^3$ a vykreslíme si chybu interpolanů p_n pro n rovno 1 až 64 oproti n^{-3}.

```
f = abs(exp(1.5*x).*sin(10*x)).^3;
n = (1:64);
err = [];
for i = 1:64
    pn = chebfun(f,i+1); err = [err, norm(f-pn,inf)];
end
loglog(n,n.^-3,'r'), hold on, loglog(n,err,'.')
```



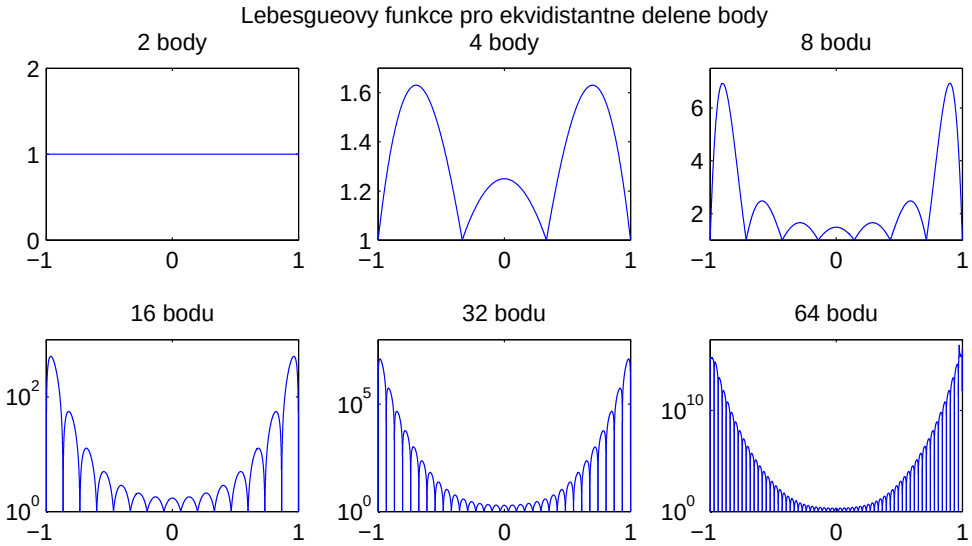
Vidíme, že maximální chyba se zmenšuje přibližně stejně rychle jako n^{-3}, pokles však nastává později. To je dáno velikostí totální variace třetí derivace funkce f. Pro funkci $|x|$ je hodnota totální variace první derivace rovna dvěma (snadno spočítáme). K určení totální variace třetí derivace druhé funkce využijeme příkaz `diff` a fakt, že totální variace funkce f je rovna $||f'||_1$.

```
tv = @(g) norm(diff(g),1);
tv(diff(f,3))
ans =
    2.6008e+06
```

Oba tyto výsledky odpovídají tvrzení Věty 1.11 a demonstrují vliv velikosti totální variace na velikost chyby $||f - p_n||_\infty$.

Mluvili jsme také o Lebesgueově konstantě a její roli při popisu konvergence. Pomocí Chebfunovského příkazu `lebesgue(uzly_interpolace)` jednoduše získáme jak Lebesgueovu funkci, tak i Lebesgueovu konstantu pro dané interpolační uzly. Nejdříve se podíváme na ekvidistantně dělené body.

```
for i = 1:6
    pts=linspace(-1,1,2^(i)); [lam, Lam] = lebesgue(pts);
end
```



Na obrázku vidíme rychlý nárůst hodnot Lebesgueovy funkce, zvláště u krajních bodů ± 1. Pro 16, 32 a 64 bodů je osa y škálovaná logaritmicky. Velikost příslušných Lebesgueových konstant je postupně:

```
1
1.6311
6.9297
512.3515
1.2658e+07
2.7543e+16
```

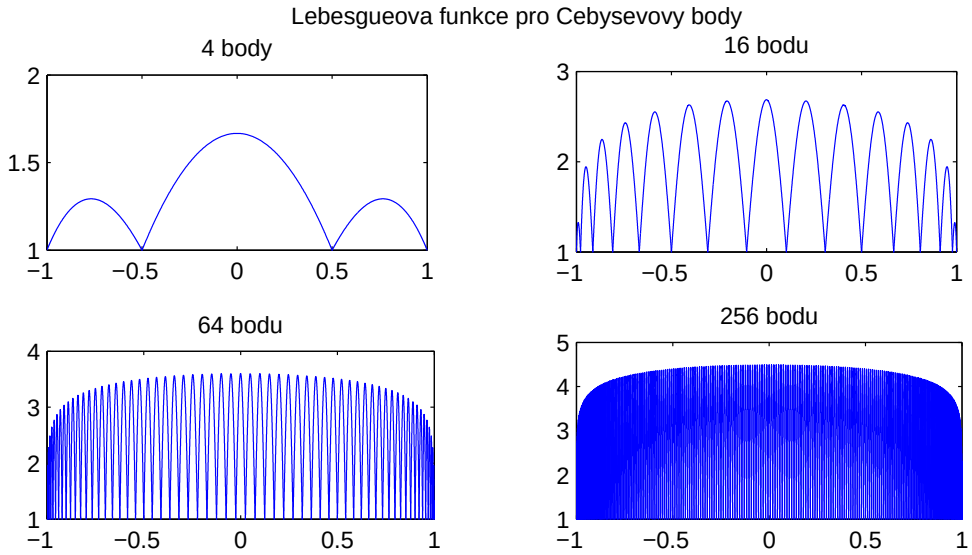
To přibližně odpovídá tvrzení Věty 1.15 (str. 13). Pro 64 bodů máme dolní odhad $\Lambda_{63} > \frac{2^{61}}{63^2} = 5.8 * 10^{14}$, a přibližný odhad

$$\Lambda_n \sim \frac{2^{64}}{63 * \mathrm{e} * \log 63} = 2.6 * 10^{16}.$$

To zhruba koresponduje se získanou hodnotou $2.7543 * 10^{16}$.

Tento výsledek srovnáme s Čebyševovými body:

```
for i = 1:4
    pts=chebpts(2^(2*i)); [lam, Lam] = lebesgue(pts);
end
```



Maximální hodnoty je oproti ekvidistantním bodům nabýváno uprostřed intervalu. Na první pohled je vidět, že i pro 256 bodů nabývá Lebesgueova funkce poměrně nízkých hodnot. To ilustruje vhodnost Čebyševových bodů pro stejnoměrnou aproximaci funkce. Odpovídající Lebesgueovy konstanty jsou (přidáme ještě Lebesgueovu konstantu pro 1024 a 4096 bodů, vykreslovat pro tolik bodů Lebesgueovu funkci by bylo zbytečné):

```
1.6667
2.6867
3.6001
4.4902
5.3746
6.2576
```

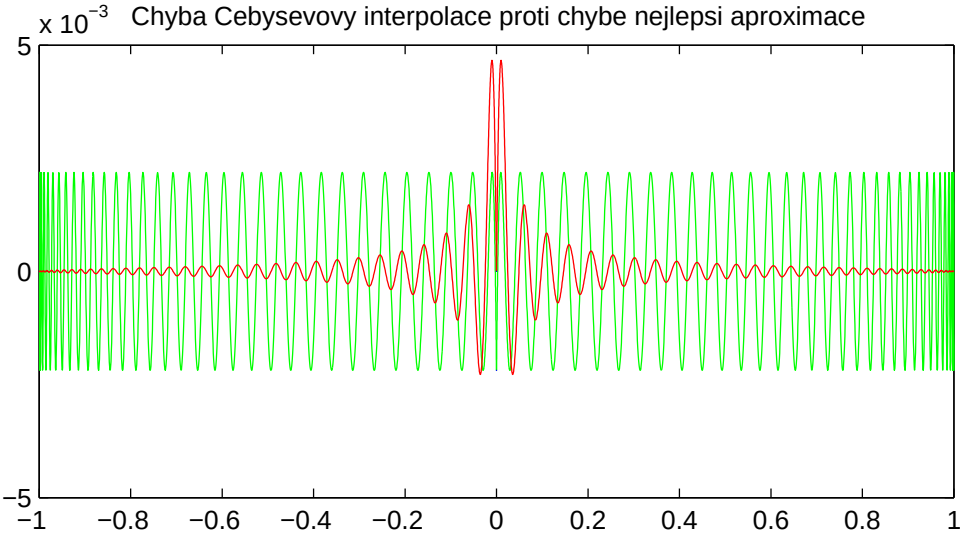
Ty opět odpovídají odhadům z Věty 1.15 (str. 13). Pro 1024 bodů máme horní odhad $\Lambda_{1023} \leq \frac{2}{\pi}\log(1024)+1 = 5.4127$ a spočtenou hodnotu 5.3746. Pro 4096[4] bodů je přibližná hodnota $\Lambda_{4095} \sim \frac{2}{\pi}\log(4095) = 5.295$ ještě stále podhodnocená.

Čebyševovy interpolanty vykazují dobré aproximační vlastnosti. Na závěr je proto porovnáme s nejlepší aproximací.

Ke spočtení nejlepší aproximace funkce f (reprezentované chebfunem) stupně n použijeme příkaz `remez(f,n)`. Jako testovací funkci vezmeme $|x|$ s nespojitou derivací a spočteme obě aproximace stupně 128. Podíváme se na rychlost výpočtu a vykreslíme si chybové funkce. Chyba Čebyševovy interpolace je vykreslena červeně, chyba nejlepší aproximace zeleně.

```
f = abs(x);
tic, best = remez(f,128); toc
tic, cheb = chebfun(f,129); toc
Elapsed time is 0.830048 seconds.
Elapsed time is 0.010085 seconds.
```

[4]Výpočet Lebesgueovy funkce a konstanty pro 4096 bodů trval několik hodin.



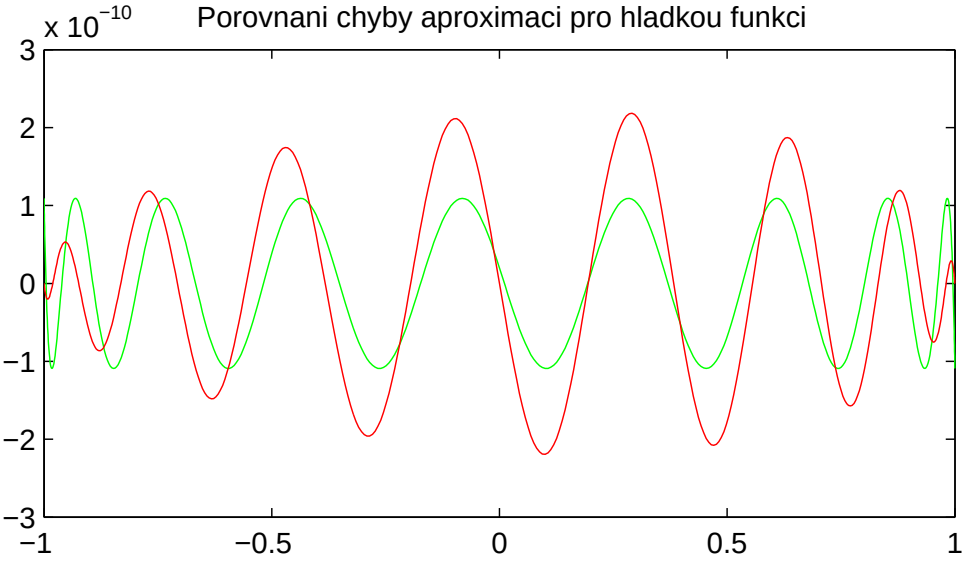
Maximální chyba nejlepší aproximace je samozřejmě menší. Nicméně mimo okolí bodu nespojitosti derivace je Čebyševův interpolant dokonce lepší než nejlepší aproximace. Navíc, spočíst nejlepší aproximaci pomocí Remezova algoritmu je osmdesátkrát pomalejší. Porovnáme ∞ a 2-normu[5] obou chyb. Dvojkovou normu zde volíme jako nástroj ke změření celkové velikosti chyby, oproti porovnávání maxima v případě ∞-normy.

```
chybabestinf = norm(f-best,inf)
chybachebinf = norm(f-cheb,inf)
chybabest2 = norm(f-best,2)
chybacheb2 = norm(f-cheb,2)

chybabestinf =
    0.0022
chybachebinf =
    0.0047

chybabest2 =
    0.0022
chybacheb2 =
    0.0009
```

I když je nejlepší aproximace dvakrát lepší v ∞-normě, ve dvojkové normě je dvakrát horší. Vezmeme ještě nekonečněkrát diferencovatelnou funkci $\sin(3x)\mathrm{e}^{2x}$. Tentokrát chceme aproximace stupně 16.

```
f = sin(3*x).*exp(2*x);
tic, best = remez(f,16); toc
tic, cheb = chebfun(f,17); toc
Elapsed time is 0.093442 seconds.
Elapsed time is 0.007466 seconds.
```

[5] Uvažujeme standardní L^2 normu.



Opět vidíme výrazný rozdíl v potřebném výpočetním čase. Podíváme se ještě na normy chyb.

```
chybabestinf =
   1.0921e-10
chybachebinf =
   2.1944e-10

chybabest2 =
   1.0916e-10
chybacheb2 =
   1.7744e-10
```

Jako v předchozím případě je chyba nejlepší aproximace v ∞-normě dvakrát menší. Ve dvojkové normě je opět lepší Čebyševův interpolant. Tyto výsledky ukazují výhodnost použití Čebyševových bodů a interpolantů pro numerické počítání. Za krátký výpočetní čas dostáváme aproximaci srovnatelnou s nejlepší aproximací (ne-li lepší). Navíc se nemusíme omezovat na pouhé počítání s funkcemi. Některá možná využití Chebfunu si představíme v následující sekci.

3.2 Hledání kořenů funkce

Populární způsob, jak nalézt kořeny monického polynomu vyjádřeného v monomiální bázi, je řešit úlohu nalezení vlastních čísel *companion* matice sestavené z koeficientů polynomu. Takto funguje MATLABovský příkaz `roots`, který k nalezení vlastních čísel používá QR algoritmus. Tato metoda je za určitých předpokladů numericky stabilní – nalezne kořeny polynomu, který je blízký tomu zadanému. Jsou však velmi dobře známy příklady, kdy i velmi malá změna koeficientů může znamenat velkou rozdílnost kořenů, viz například [10]. Jinak řečeno, kořeny polynomů mohou být velmi citlivé na malé změny koeficientů a blízkost koeficientů polynomů obecně nezaručuje blízkost kořenů. Jednou z možností, jak zamezit numerickým problémům, je reprezentovat polynom v jiné polynomiální bázi takové, že kořeny polynomu jsou lépe podmíněnou funkcí koeficientů polynomu v této bázi. Dobrým kandidátem je báze Čebyševových polynomů. Matice

složená z koeficientů polynomu reprezentovaného v této bázi se nazývá *colleague* matice. Vanni Noferini a Javier Pérez v současném článku zaslaném k publikaci [4] dokázali, že touto metodou pro polynom p spočteme kořeny polynomu $\tilde{p}$, pro který platí

$$\frac{||p-\tilde{p}||}{||p||} = O(\varepsilon)||p||\,,$$

kde ε značí strojovou přesnost a $||p||$ je definovaná jako $\sqrt{\sum_{k=0}^{n-1} a_k^2}$, a_k jsou koeficienty polynomu v bázi Čebyševových polynomů. Počítání kořenů polynomu p pomocí *colleague* matice je zpětně stabilní, pokud je $||p||$ rozumně velká.

Jelikož Chebfun reprezentuje všechny funkce jako polynomy a ukládá příslušné koeficienty, nepotřebujeme k sestavení *colleague* matice počítat nic navíc. Implementace v Chebfunu navíc obsahuje rekurzivní dělení intervalu na podintervaly v případě, že je příslušný chebfun stupně víc jak 100. Díky tomu neřešíme problém vlastních čísel pro matici větší jak 100×100. Podrobnější popis je možné nalézt v [9, Kapitola 18].

3.3 Spektrální metody

V této sekci nastíníme princip řešení diferenciálních rovnic pomocí spektrálních metod. Ve čtvrté sekci pak ukážeme, jak lze takové rovnice řešit pomocí Chebfunu a pokusíme se numericky vyřešit problém konvekce-difúze, u kterého očekáváme numerické problémy. Podrobný popis spektrálních metod lze nalézt v Trefethenově knize [8], jejich implementaci v Chebfunu je věnována kapitola 21 v [9].

Libovolný polynom $q \in \mathcal{P}_n$ je jednoznačně určen svými hodnotami na $n+1$ bodech. Stejně tak i jeho derivace q'. Nechť $\{x_j\}$ je daná množina $n+1$ bodů. Přechod od hodnot $q(x_j)$ k hodnotám $q'(x_j)$ je lineární, dá se tedy realizovat pomocí násobení maticí $(n+1) \times (n+1)$. Tato matice se nazývá *diferenční matice* a značíme jí D_n. Uvažujme množinu dvojic (x_j,f_j), $j = 1,\ldots,n$. Nechť p je polynom, pro který platí $p(x_j) = f_j$. Potom vektor $\mathbf{p}' = (p'(x_0),\ldots,p'(x_n))^T$ můžeme spočíst jako $\mathbf{p}' = D_n\mathbf{p}$, kde $\mathbf{p} = (p(x_0),\ldots,p(x_n))^T$. Prvky matice $D_n = \{d_{ij}\}_{i,j=0}^n$ lze přímo určit z (1.1) a (1.2) (str. 3). Uvažujme polynom p ve tvaru (1.1), potom $p'(x) = \sum_{i=0}^n f_i\ell_i'(x)$ a hodnota d_{ij} je rovna $\ell_j'(x_i)$. Ke spočtení $\ell_j'(x_i)$ použijeme vzorec (1.2). Pro prvky matice d_{ij} dostáváme vyjádření

$$d_{ij} = \begin{cases} \ell_j'(x_i) = \frac{\ell'(x_i)\lambda_j}{x_i-x_j} - \frac{\ell(x_i)\lambda_j}{(x_i-x_j)^2} = \frac{\lambda_j}{\lambda_i(x_i-x_j)} & \text{pro } i \neq j, \\ \ell_j'(x_j) = \sum_{k\neq j} \frac{1}{(x_k-x_j)} & \text{pro } i = j. \end{cases}$$

Pro Čebyševovy body je λ_j dáno vzorcem (1.8) (str. 7). Hodnoty vyšších derivací lze získat opakovanou aplikací diferenční matice D_n, ν-té derivaci odpovídá $D_n^\nu\mathbf{p}$.

Ilustrujme nyní použití diferenčních matic. V kapitole 2 jsme zmínili, že pro aproximaci funkce $f(x) = \tanh(x)$ je potřeba 30 bodů, její interpolant p_f je jednoznačně určen vektorem hodnot $\mathbf{p}_f$ v 30 bodech. Vezmeme-li diferenční matici $D = D_{30}$, můžeme operátor $\mathcal{L} : f \to 5f - 2f' + f''$ aproximovat maticí $L = 5I - 2D + D^2$, kde I je jednotková matice. Aproximací $\mathcal{L}(f)$ je polynom jednoznačně určený hodnotami $L\mathbf{p}_f$. Toho lze využít při řešení diferenciálních

rovnic, kde potřebujeme nalézt aplikaci inverze daného operátoru na funkci pravé strany. Řešíme-li například rovnici $5u-2u'+u''=g$, hledáme vlastně $\mathcal{L}^{-1}(g)$. Při reprezentaci operátoru $\mathcal{L}$ maticí převedeme problém na řešení soustav lineárních rovnic $L\mathbf{p}_u = \mathbf{p}_g$.

Chebfun umí příkazem `D = chebop(@(u) diff(u));` sestrojit operátor odpovídající derivaci, reprezentovaný diferenční maticí D. Matici konkrétního rozměru N vytvoříme zadáním `DN = D(N)`. Nemusíme se omezovat pouze na jednoduchou derivaci, operátor $\mathcal{L}$ zmíněný výše sestrojíme příkazem

```
L = chebop(@(u) 5*u - 2*diff(u) + diff(u,2));
```

Objekty `D` a `L` se nazývají „linear chebop“ a v Chebfunu nejsou uloženy jako matice, ale jako předpisy jak sestrojit matici daného rozměru. Pokud je do `D` zadán argument (hodnota N), Chebfun sestrojí matici typu $N \times N$. Pokud je `D` aplikován na chebfun `f` pomocí operace násobení `D*f`, je sestrojena matice velikosti `length(f)` a výstupem je chebfun odpovídající derivaci `f`.

Pro řešení diferenciálních rovnic potřebujeme ještě zadat okrajové podmínky. Uvažujme rovnici $5u - 2u' + u'' = 0$ s Dirichletovými okrajovými podmínkami $u(-1) = u(1) = 0$. Ty zadáme příkazem `L.bc = 'dirichlet';`. Zadanou rovnici vyřešíme jednoduše pomocí zpětného lomítka a pravé strany: `u = L\0;`. Takto získané `u` je proměnná typu chebfun. Velikost matice se v tomto případě určuje automaticky tak, aby bylo dosaženo počítačové přesnosti. Změníme-li pravou stranu na $\sin(x)$ a počáteční podmínky na $u(-1) = 1$, $u(1) = 2$, bude zadání vypadat následovně:

```
L = chebop(@(u) 5*u - 2*diff(u) + diff(u,2));
L.lbc = 1; L.rbc = 2;
x = chebfun('x');
f = sin(x);
u = L\f;
```

Neumannovy okrajové podmínky zadáme příkazem `L.bc = 'neumann';`, pro jiný interval než [−1,1] použijeme nepovinný parametr při definování operátoru:

```
D = chebop(@(u) diff(u),[a,b]);
```

Můžeme definovat i operátor s proměnnými koeficienty, například $\mathcal{L} = u' + xu$ vytvoříme příkazem

```
L = chebop(@(u,x) diff(u) + x.*u);
```

Chebfun zvládá řešit i nelineární problémy, této problematice je věnována samostatná kapitola v `http://www.chebfun.org/docs/guide/chebfun_guide.pdf`.

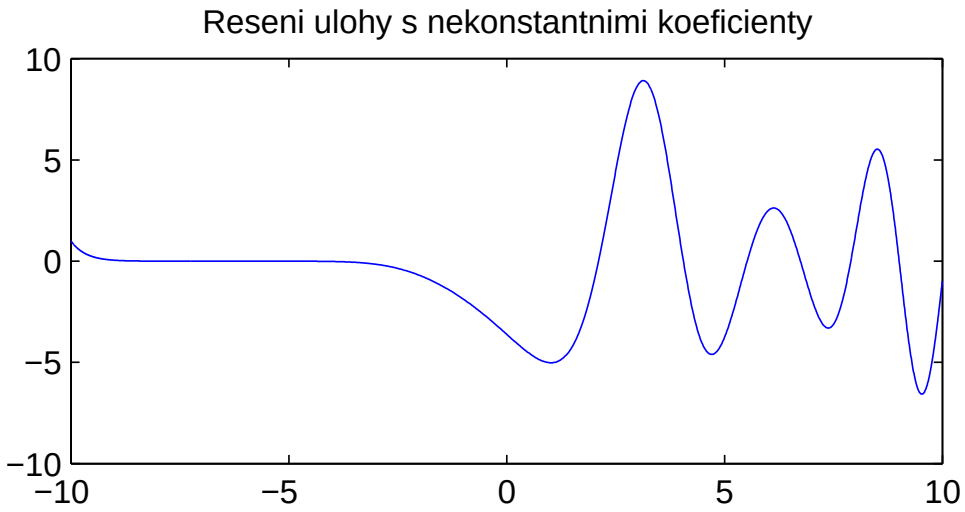
3.4 Experiment

V poslední sekci použijeme Chebfun k řešení dvou obyčejných diferenciálních rovnic. Jako první vyřešíme rovnici s nekonstantními koeficienty

$$u(x)'' - \sin(x)u(x)' + xu = 0$$

na intervalu [−10,10] s okrajovými podmínkami $u(-10) = 1$, $u(10) = -1$.

```
L = chebop(@(x,u) diff(u,2) - sin(x).*diff(u) + x.*u, [-10, 10]);
L.bc = 'dirichlet'; L.lbc = 1; L.rbc = -1;
u=L\0;
```



Podíváme se ještě na přesnost numerického řešení pomocí příkazu `norm(L*u)`.

```
ans =
   5.1291e-10
```

Obdrželi jsme poměrně přesné řešení netriviální úlohy.

Pro druhý experiment zvolíme rovnici konvekce-difúze. Ta popisuje například šíření nečistot v řece. Nečistoty jsou unášeny proudem – konvektivní složka popsaná první derivací – a zároveň probíhá difúze – tu reprezentuje druhá derivace. V jedné dimenzi dostáváme rovnici

$$-\varepsilon u''(x) + a(x)u'(x) + b(x)u(x) = f(x) \qquad 0 < x < 1 \qquad u(0) = u(1) = 0,$$

kde a, b, f jsou zadané funkce a ε je kladná konstanta. Protože proudění má dominantní vliv na šíření nečistot, uvažujeme $\varepsilon \ll 1$. Podrobný popis problému konvekce-difúze a jeho numerického řešení lze nalézt například v [7].

Uvažujme konkrétně problém

$$-\varepsilon u''(x) + u'(x) = 1 \qquad 0 < x < 1 \qquad u(0) = u(1) = 0, \tag{3.1}$$

položíme-li $\varepsilon = 0$ dostáváme rovnici $u'(x) = 1$, která pro zadané okrajové podmínky nemá řešení v množině spojitých funkcí. Odtud lze tušit numerické problémy při řešení rovnice (3.1) pro malé hodnoty ε. A vskutku, pro přesné řešení (3.1)

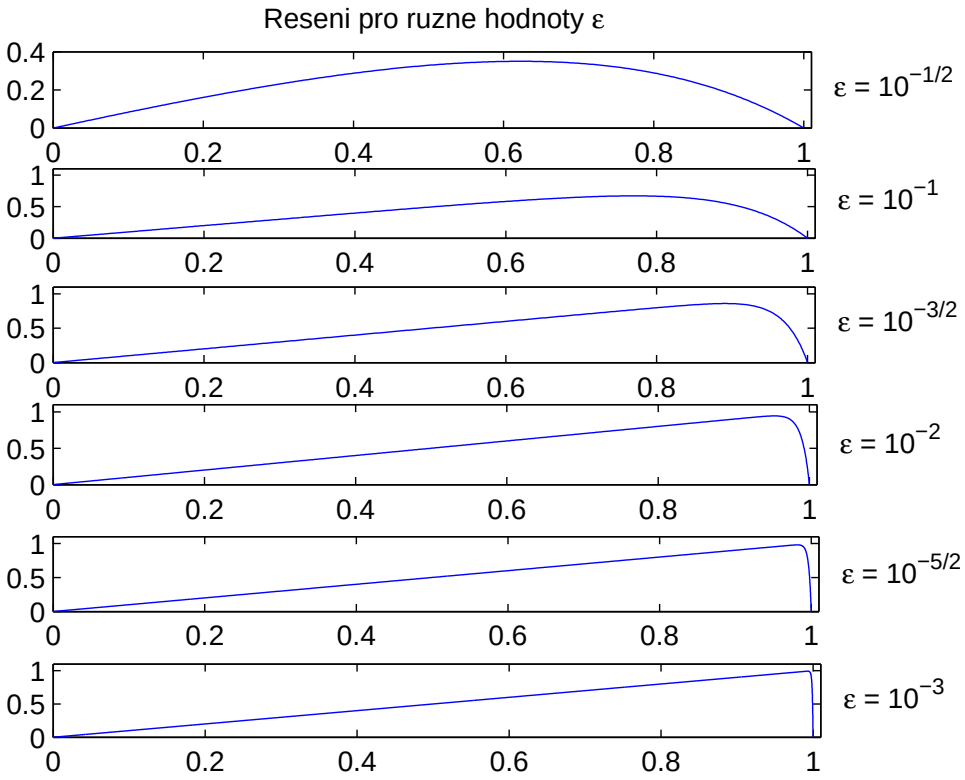
$$u(x) = x - \frac{\exp(-\frac{1-x}{\varepsilon}) - \exp(-\frac{1}{\varepsilon})}{1 - \exp(-\frac{1}{\varepsilon})}$$

platí

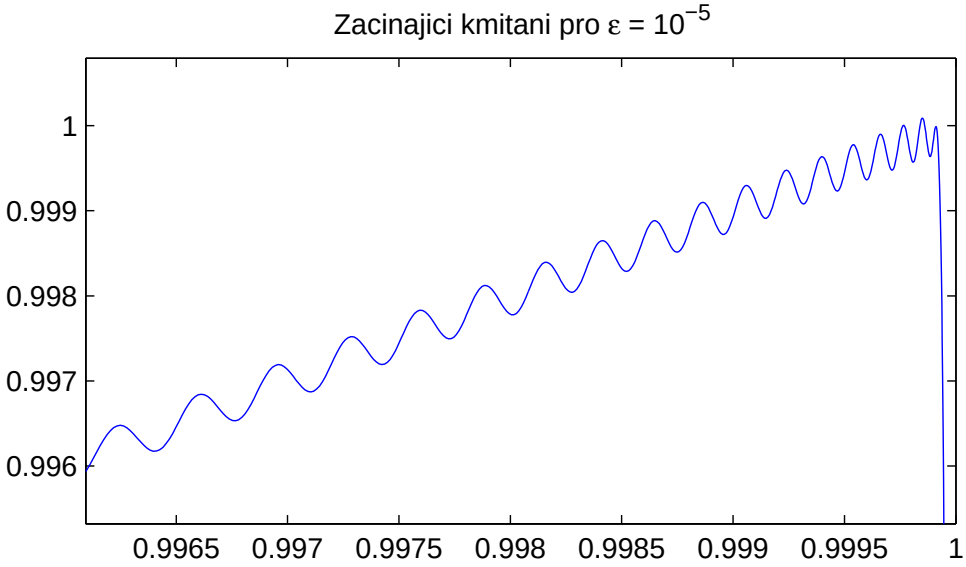
$$1 = \lim_{x\to 1}\lim_{\varepsilon\to 0} u(x) \neq \lim_{\varepsilon\to 0}\lim_{x\to 1} u(x) = 0\,.$$

Odtud vyplývá, že řešení se rychle mění s x blížícím se k 1. Podívejme se, jak si s touto rovnicí poradí Chebfun pro různé volby ε.

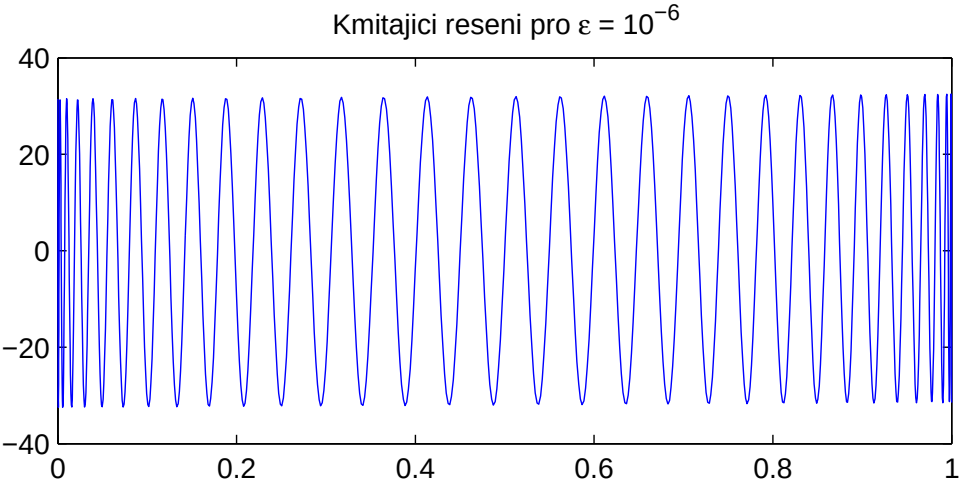
```
for i = 1 : 6
    L = chebop(@(u) -1*(10^-(i/2))*diff(u,2) + diff(u),[0,1]);
    L.bc = 'dirichlet';
    L.lbc = 0; L.rbc=0;
    u=L\1;
    subplot(6,1,i), plot(u)
end
```



Vidíme, jak se při zmenšujícím se ε vytváří výrazný propad řešení těsněji u krajního bodu 1. Přiblížíme-li si numerické řešení pro $\varepsilon = 10^{-5}$, můžeme vidět začínající kmitání.

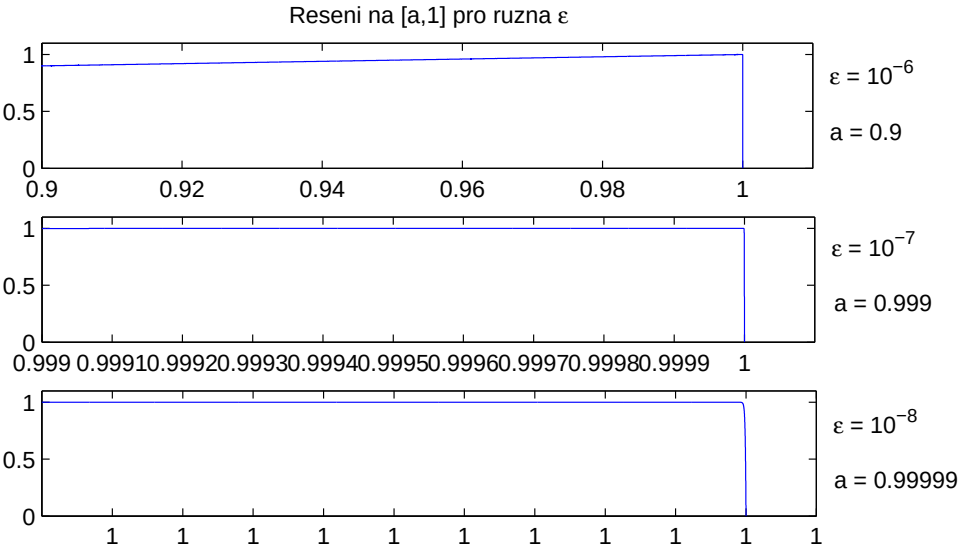


Zvolíme-li $\varepsilon = 10^{-6}$ numerické řešení začne výrazně kmitat a přestane odpovídat přesnému řešení.



V [7, str. 127] je popsáno řešení pomocí Shishkinových sítí. Důležitou roli zde hraje tzv. *mesh transition point* ležící, v našem případě, mezi bodem $\frac{1}{2}$ a 1. Inspirováni tímto, zkusíme zpřesnit numerické řešení pro $\varepsilon \leq 10^{-6}$ soustředěním se pouze na interval $[a,1]$. Potřebujeme ještě okrajovou podmínku v bodě a, tu zvolíme jednoduše jako $u(a) = a$ (dosazením a do přesného řešení můžeme ověřit, že na většině intervalu se přesné řešení chová pro malá ε lineárně). Po vyzkoušení několika hodnot nebylo těžké určit pro $\varepsilon = 10^{-6}, 10^{-7}, 10^{-8}$ vhodné a po řadě 0.9, 0,999 a 0,99999.

```
for i = 1 : 3
    j = 5+i; k = 2*i-1;
    L = chebop(@(u) -1*(10^-j)*diff(u,2) + diff(u),[1-10^-k,1]);
    L.lbc = 1-10^-(2*i-1); L.rbc=0;
    u=L\1;
    subplot(3,1,i), plot(u)
end
```



Jednoduchým zápisem na pár řádků dostaneme řešení v několika sekundách. Spočíst řešení pro všechna výše uvedená ε (oba `for` cykly) zabralo celkem

```
clear
tic
.
```

```
.
.
toc
Elapsed time is 3.636012 seconds.
```

Výše zmíněná úprava nám dovolila získat rozumné numerické řešení pro větší rozsah hodnot ε, nedá se však aplikovat na všechny problémy. V obecném případě (interval [0,1], zadané okrajové podmínky $u(0)$ a $u(1)$) bychom mohli zkusit například Schwarzovu metodu na dvou překrývajících se oblastech. V intervalu zvolíme dva body a, b, $a < b$ a postupujeme iteračně. V prvním kroku spočteme řešení u_{b1} na intervalu $[0,b]$ s libovolnou okrajovou podmínkou $u_{b1}(b)$. V druhém kroku spočítáme řešení u_{a1} na intervalu $[a,1]$ s okrajovou podmínkou $u_{a1}(a) = u_{b1}(a)$. Dále spočteme u_{b2} s okrajovou podmínkou $u_{b2}(b) = u_{a1}(b)$... Pokračujeme, dokud $u_{an}(b) = u_{bn}(b) \pm e$, e je zvolená tolerance (popřípadě $u_{bn+1}(a) = u_{an}(a) \pm e$).

```
Lb = chebop(dany_operator,[0,b]);
La = chebop(dany_operator,[a,1]);
    Lb.lbc = dana_podminka; Lb.rbc=0;
    ub=L\prava_strana;
for i = 1:potrebny_pocet_iteraci
    La.lbc = ub(a); La.rbc=dana_podminka;
    ua=L\prava_strana;
    Lb.lbc = dana_podminka; Lb.rbc=ua(b);
    ub=L\prava_strana;
end
```

Tato úloha představuje demonstraci toho, že na některé problémy nelze použít Chebfun přímo. Lze jej ale využít při konstrukci metody pro řešení daného problému. V našem případě jsme se omezili na část intervalu a odhadli jsme okrajové podmínky. Navrhli jsme také Schwarzovu metodu se dvěma překrývajícími se oblastmi. Pro náš problém se však ukázalo, že Schwarzova metoda je velmi citlivá na volbu hranic obou oblastí a na příslušné okrajové podmínky. Nepodařilo se nám nalézt vhodné parametry, které by vedly k numericky stabilnímu výpočtu aproximace řešení. Řešíme-li úlohu s takto problematickým chováním numerického řešení, je zcela jistě potřeba založit numerickou metodu na dobré znalosti a analýze dané diferenciální rovnice. Takový požadavek však přesahuje hranice této bakalářské práce.

Závěr

Autoři Chebfunu se snaží vytvořit efektivní nástroj, který by umožňoval počítat v MATLABu s funkcemi podobně jako s čísly. Ideou je na patnáct platných cifer aproximovat zadané funkce a tuto přesnost udržet při každé operaci s nimi, to všechno bez větších časových nároků. Do velké míry se jim to podařilo. Přesnost výpočtů je obvykle opravdu 15 cifer, u komplikovaných funkcí na hraně spojitosti nebo u nehladkých funkcí může docházet, a také dochází, ke ztrátě několika platných cifer, viz například závěr kapitoly 5 v [9]. Rychlost, s jakou jsou prováděny operace, je vynikající. Na běžném počítači trvalo spočítat a vykreslit všechny výše uvedené experimenty (s výjimkou výpočtu Lebesgueovy funkce a konstanty pro 1024 a 4096 bodů) méně jak minutu. Pokusy zadat Chebfunu funkci tak, aby ji nebyl schopný aproximovat byly neúspěšné (nepočítáme-li například Weierstrassovu funkci), na konci kapitoly 2 jsme ukázali jeden takový pokus a schopnost Chebfunu zpětně kontrolovat své výpočty.

Samotné používání Chebfunu bylo občas zkomplikováno ne vždy přehlednou nápovědou a nemožností provádět některé základní operace s chebfuny. Není například možné sečíst dva chebfuny na intervalech $[a,b]$, $[b,c]$ a vytvořit tak chebfun složený ze dvou funů na intervalu $[a,c]$. Tento proces se dá obejít složitější konstrukcí. Dalším příkladem je absence možnosti dělit interval při řešení diferenciálních rovnic. Možné řešení jsme nastínili na konci předchozí kapitoly.

Existují programy počítající s funkcemi analyticky. Výstupem potom často bývá velmi složitý výraz, jehož vyhodnocení je časově náročné. Některé problémy nejsou dokonce analyticky vůbec řešitelné. Numerické řešení zase poskytuje pouze číselné hodnoty, nikoliv funkci se kterou se dá dále pracovat. Chebfun nabízí kombinaci analytického a numerického přístupu. MATLAB je program využívaný mnoha firmami a Chebfun jako jeho rozšíření představuje další možnost jak rychle řešit problémy. Jejich převedením na polynomiální problém zjednoduší způsob řešení. Chebfun však má svá omezení, zejména funkce, které nejsou hladké nebo spojité. Celkově je Chebfun užitečný rychlý praktický i teoretický nástroj kladoucí vyšší nároky na uživatele, zejména při zadávání příkazů a kontrole obdržených výsledků.

`http://www.chebfun.org`

Literatura

[1] E. W. Cheney, *Introduction to approximation theory*, AMS Chelsea Publishing, Providence, RI, 1998. Reprint of the second (1982) edition.

[2] P. J. Davis, *Interpolation and approximation*, Blaisdell Publishing Co. Ginn and Co. New York-Toronto-London, 1963.

[3] E. Isaacson and H. B. Keller, *Analysis of numerical methods*, Dover Publications, Inc., New York, 1994. Corrected reprint of the 1966 original [Wiley, New York; MR0201039 (34 #924)].

[4] V. Noferini and J. P. Alvaro, *2015.24: Chebyshev rootfinding via computing eigenvalues of colleague matrices: when is it stable?*, Tech. Report MIMS EPrint: 2015.24, The University of Manchester, 2015.

[5] M. J. D. Powell, *Approximation theory and methods*, Cambridge University Press, Cambridge-New York, 1981.

[6] T. J. Rivlin, *The Chebyshev polynomials*, Wiley-Interscience [John Wiley & Sons], New York-London-Sydney, 1974. Pure and Applied Mathematics.

[7] H.-G. Roos, M. Stynes, and L. Tobiska, *Robust numerical methods for singularly perturbed differential equations*, vol. 24 of Springer Series in Computational Mathematics, Springer-Verlag, Berlin, second ed., 2008. Convection-diffusion-reaction and flow problems.

[8] L. N. Trefethen, *Spectral methods in MATLAB*, vol. 10 of Software, Environments, and Tools, Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2000.

[9] ——, *Approximation theory and approximation practice*, Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2013.

[10] J. H. Wilkinson, *The perfidious polynomial*, in Studies in numerical analysis, vol. 24 of MAA Stud. Math., Math. Assoc. America, Washington, DC, 1984, pp. 1–28.