Univerza v Ljubljani
Fakulteta za računalništvo in informatiko
Fakulteta za matematiko in fiziko

Andrej Pangerčič

Programski paket Chebfun

diplomsko delo
univerzitetni študijski program prve stopnje
računalništvo in matematika

Mentor: izr. prof. dr. Gašper Jaklič

Ljubljana 2014

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

[1]Dodatno branje: http://creativecommons.si/licence.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

V diplomskem delu obravnavajte odprtokodni programski paket Chebfun. Najprej opišite osnove Čebiševih polinomov in interpolacijo s polinomi. Nato predstavite Chebfun in njegovo uporabo. Paket preizkusite na uporabnih primerih, npr. pri reševanju diferencialnih enačb.

Osnovna literatura: T. A. Driscoll, N. Hale, and L. N. Trefethen, editors, Chebfun Guide, Pafnuty Publications, Oxford, 2014.

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Andrej Pangerčič, z vpisno številko **63070099**, sem avtor diplomskega dela z naslovom:

Programski paket Chebfun

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom izr. prof. dr. Gašperja Jakliča,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela,
- soglašam z javno objavo elektronske oblike diplomskega dela na svetovnem spletu preko univerzitetnega spletnega arhiva.

V Ljubljani, dne 17. septembra 2014 Podpis avtorja:

V prvi vrsti se zahvaljujem staršem za finančno podporo v času študija v Ljubljani. Prav tako se zahvaljujem bratu Dejanu in kolegom iz IŠRM-ja za izjemno podporo in motivacijo pri zaključevanju študija. Hvala tudi mentorju, izr. prof. dr. Gašperju Jakliču, za pomoč pri izbiri teme diplomskega dela in koordinaciji ter za ves trud in čas, ki si ga je vzel zame. Hvala lepa za potrpežljivost, nasvete in vse popravke.

Kazalo

Povzetek

Skoraj vsak matematični problem, ki ga ne moremo več rešiti na roko zaradi naše omejene hitrosti reševanja, smo primorani rešiti z računalnikom. Za začetek moramo predstaviti problem v računalniški obliki in nato nad njim izvršiti neko matematično operacijo v obliki algoritma. Pri tem si želimo imeti algoritme, ki se izvajajo v realnem času. Funkcije največkrat aproksimiramo z interpolacijskimi polinomi, s katerimi najlažje in hitro računamo. Kaj kmalu ugotovimo, da osnovna uporaba interpolacije ni primerna, saj povzroča prevelike napake. V tem diplomskem delu se bomo posvetili Čebiševi vrsti, ki v praksi daje zelo dobre aproksimacije za lepe funkcije. Pogledali si bomo, kako je takšna aproksimacija implementirana v odprtokodnem programskem orodju Chebfun. Na koncu bomo pokazali nekaj uporabnih primerov, ki jih največkrat rešujemo na računalniku in njihovo reševanje s pomočjo programskega okolja.

Ključne besede: polinom, aproksimacija, interpolacija, Čebiševa vrsta, Matlab, Chebfun.

Abstract

Almost every mathematical problem that can not be solved manually due to our limited speed of calculating, we need to solve with the computer. We first present the problem in an electronic form and then execute a mathematical operation in a form of an algorithm that runs in real time. Usually we can approximate functions by interpolation with polynomials, because the polynoms are the easiest and fastest to compute. A significant drawback of this approach is in that large errors can arise thereby. To overcome this problem we in this thesis investigate Chebyshev series which in practice give a very good approximation for smooth functions. In particular we examine an approximation implemented in the open source software tool Chebfun. At the end we present a couple of examples used commonly in problem solving with the computer and solve them with the software tool.

Keywords: polynom, approximation, interpolation, Chebyshev series, Matlab, Chebfun.

Poglavje 1

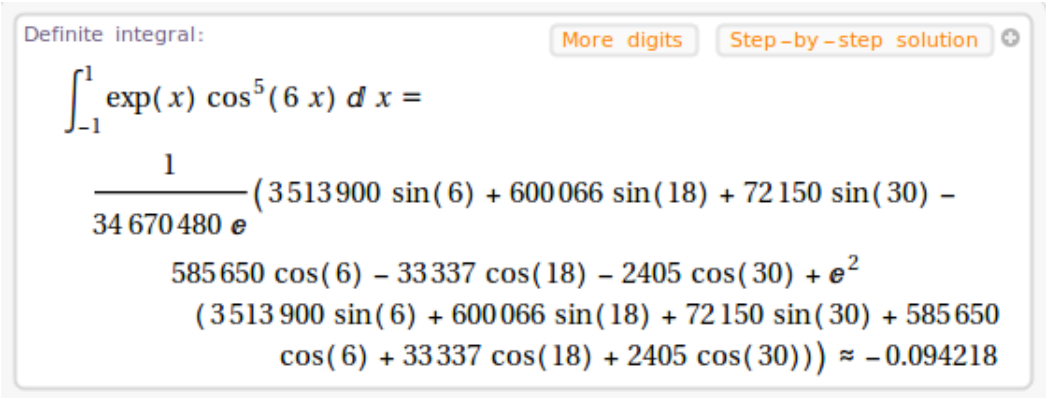
Uvod

Eden izmed najpogostejših problemov v analizi je izračun vrednosti funkcije $f(x)$ pri nekem $x \in [a, b]$ in matematične manipulacije, ki jih izvajamo nad njo. Funkcijo lahko predstavimo simbolično ali numerično.

Simbolično reševanje enačb je človeku bolj prijazno orodje, vendar ga je težje sprogramirati. Prav tako imamo lahko težave s simboličnimi rešitvami, saj pri malo bolj kompliciranih izrazih v praksi lahko dobimo eksponentne dolžine rešitev in čase reševanja. Primer takega reševanja lahko vidimo na sliki 1.1, kjer računamo integral preproste funkcije.

Pri numerični predstavitvi funkcije imamo dve možnosti; bodisi izračunamo veliko numeričnih približkov za funkcijo na nekem intervalu in jih shranimo v vektor števil, bodisi računamo z interpolacijskimi polinomi. Prva možnost je načeloma slabša, saj moramo najprej izračunati vse funkcijske vrednosti, katerih večinoma ne bomo eksplicitno uporabili, vendar jih bomo potrebovali kasneje, recimo za izračun odvoda. Pri izračunu odvoda moramo pognati ustrezen algoritem skozi cel vektor števil, čeprav bi si želeli poznati vrednost odvoda funkcije samo v eni točki.

Kakršnakoli uporaba interpolacije ni smiselna, ampak mora zadostovati vsaj dvema pogojema. Najprej mora biti enostavna za procesorsko oziroma računalniško računanje. V praksi si želimo izračun funkcijske vrednosti v realnem času, zavedati pa se moramo tudi, da nimamo vedno na razpolago



Slika 1.1: Primer izračuna določenega integrala $\int_{-1}^{1}\exp(-x)\cos^5(6x)\,\mathrm{d}x$ s pomočjo programa Mathematica na intervalu $[-1, 1]$.

najboljše računalniške arhitekture. Drugi pogoj govori o tem, da moramo znati oceniti napako interpolacije na vsakem koraku, saj nam lahko nekateri algoritmi izračunajo zelo slabe aproksimacije funkcij.

V diplomskem delu si bomo najprej ogledali lastnosti Čebiševih polinomov in metode za izgradnjo interpolacijskih funkcij. V nadaljevanju bomo predstavili Rungejev problem, ki pokaže, da je aproksimacija z uporabo polinomov in izbiro ekvidistančnih točk zelo slaba. Z uporabo Čebiševih točk bomo pokazali, da teh težav v splošnem nimamo več. V tretjem poglavju bomo prikazali osnove uporabe programskega paketa Chebfun in aproksimacijo nezveznih funkcij z uporabo zlepkov. Za konec bomo predstavili še nekaj uporabnih problemov in njihovo reševanje z uporabo programskega paketa.

Poglavje 2

Čebiševi polinomi in interpolacija

Obstaja več vrst Čebiševih polinomov, pri njihovem poimenovanju pa si avtorji niso enotni. V diplomskem delu bomo omenjali Čebiševe polinome prve vrste oziroma enako kot jih imenuje Wikipedija [5]. Čebiševe polinome prve vrste (v nadaljevanju samo Čebiševe polinome) označujemo z notacijo T_n. V praksi uporabljamo tudi Čebiševe polinome druge vrste z oznako U_n. V teoriji se pojavljajo še Čebiševi polinomi tretje in četrte vrste, vendar nimajo nekih bistvenih prednosti pri uporabi.
Sedaj definirajmo Čebiševe bazne polinome, ki so oblike

$$T_n(x) = \cos(n \arccos(x)), \quad n \in \mathbb{N}, x \in [-1, 1] \tag{2.1}$$

oziroma

$$T_n(x) = \cosh(n \operatorname{arccosh}(x)), \quad n \in \mathbb{N}, x \in \mathbb{R} \setminus (-1, 1),$$

ki jih lahko zapišemo tudi v standardni bazi po naslednji izpeljavi. Najprej vpeljemo notacijo

$$\Theta = \arccos x, \text{ kjer je } \Theta \in [0, \pi], \tag{2.2}$$

ki jo vpeljemo v enačbo (2.1) in dobimo

$$T_n(\Theta(x)) \equiv T_n(\Theta) = \cos(n\Theta).$$

Omenimo lahko še Čebiševe polinome druge vrste

$$U_n(x) = \frac{\sin\left((n+1)\arccos x\right)}{\sin(\arccos x)}.$$

S pomočjo adicijskih izrekov in distributivnosti opazimo

$$T_{n+1}(\Theta) = \cos((n+1)\Theta) = \cos(n\Theta)\cos(\Theta) - \sin(n\Theta)\sin(\Theta), \tag{2.3}$$

$$T_{n-1}(\Theta) = \cos((n-1)\Theta) = \cos(n\Theta)\cos(\Theta) + \sin(n\Theta)\sin(\Theta). \tag{2.4}$$

Enačbi (2.3) in (2.4) seštejemo in dobimo

$$T_{n+1}(\Theta) + T_{n-1}(\Theta) = 2\cos(n\Theta)\cos(\Theta).$$

Če se vrnemo nazaj na spremenljivko x, dobimo

$$T_{n+1}(x) = 2\cos(n\arccos x)\cos(\arccos x) - T_{n-1}(x),$$
$$T_{n+1}(x) = 2x\cos(n\arccos x) - T_{n-1}(x).$$

Opazimo rekurzijo

$$T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x), \tag{2.5}$$

pričnemo pa z naslednjima začetnima pogojema

$$T_0(x) = 1 \text{ in } T_1(x) = x.$$

Nekaj začetnih Čebiševih baznih polinomov je

$$T_0(x) = 1$$
$$T_1(x) = x$$
$$T_2(x) = 2 * x^2 - 1$$
$$T_3(x) = 4 * x^3 - 3 * x$$
$$T_4(x) = 8 * x^4 - 8 * x^2 + 1$$
$$T_5(x) = 16 * x^5 - 20 * x^3 + 5 * x$$
$$\dots$$

Opazimo lahko, da so polinomi sode stopnje sode funkcije in polinomi lihe stopnje lihe funkcije.

2.1 Ostale lastnosti

Najprej se posvetimo produktu Čebiševih polinomov

$$2\,T_m(x)T_n(x) = T_{m+n}(x) + T_{m-n}(x). \tag{2.6}$$

Izraz je analogen trigonometričnemu izrazu

$$2\cos\alpha\cos\beta = \cos(\alpha+\beta) + \cos(\alpha-\beta), \tag{2.7}$$

kjer je $\alpha = m\arccos x$ in $\beta = n\arccos x$. V primeru $n = 1$ dobimo rekurzivno zvezo (2.5). Iz tega produkta lahko izpeljemo še tri pomembne zveze

$$\begin{aligned} T_{2n}(x) &= 2T_n^2(x) - T_0(x) = 2T_n^2(x) - 1, \\ T_{2n+1}(x) &= 2T_{n+1}(x)T_n(x) - T_1(x) = 2T_{n+1}(x)T_n(x) - x, \\ T_{2n-1}(x) &= 2T_{n-1}(x)T_n(x) - T_1(x) = 2T_{n-1}(x)T_n(x) - x. \end{aligned}$$

Omenimo še nekaj možnih oblik Čebiševih polinomov, med katerimi izstopajo naslednje

$$\begin{aligned} T_n(x) &= \begin{cases} \cos(n\arccos(x)), & |x| \le 1 \\ \cosh(n\,\mathrm{arccosh}(x)), & x \ge 1 \\ (-1)^n\cosh(n\,\mathrm{arccosh}(-x)), & x \le -1 \end{cases} \\ &= \frac{\left(x-\sqrt{x^2-1}\right)^n + \left(x+\sqrt{x^2-1}\right)^n}{2} \\ &= \sum_{k=0}^{\lfloor \frac{n}{2} \rfloor} \binom{n}{2k} \left(x^2-1\right)^k x^{n-2k} \end{aligned}$$

in povezave med Čebiševimi polinomi prve in druge vrste

$$\begin{aligned} \tfrac{d}{dx}\,T_n(x) &= nU_{n-1}(x)\ ,\ n = 1, 2\ldots \\ T_n(x) &= \tfrac{1}{2}(U_n(x) - U_{n-2}(x)). \end{aligned}$$

2.2 Ničle in ekstremi

Iz trigonometrije vemo, kje ležijo ničle kosinusne funkcije

$$\cos\left(\tfrac{\pi}{2}\,(2k-1)\right) = 0, \quad k \in \mathbb{Z}.$$

Ničle Čebiševega polinoma x_k lahko dobimo z uporabo zgornje zveze in enačbe (2.1),

$$\begin{aligned} \tfrac{\pi}{2}\,(2k-1) &= n \arccos(x_k) \\ \tfrac{\pi}{2n}\,(2k-1) &= \arccos(x_k) \\ x_k &= \cos\left(\tfrac{\pi}{2n}\,(2k-1)\right) \end{aligned}$$

Ekstreme $T_n(x)$ lahko poiščemo tako, da odvajamo funkcijo (2.1) in poiščemo ničle funkcije

$$\frac{n \sin(n \arccos(x))}{\sqrt{1-x^2}}.$$

Uporabimo zvezo $\sin(x) = 0$, ko je $x = k\pi$, za $k = 0, 1, \ldots$, in dobimo

$$\begin{aligned} n \arccos(\tilde{x}_k) &= k\pi \\ \arccos(\tilde{x}_k) &= \frac{k\pi}{n} \\ \tilde{x}_k &= \cos\left(\frac{k\pi}{n}\right), \quad k = 0, 1, \ldots, n. \end{aligned} \tag{2.8}$$

Ničle odvoda Čebiševega polinoma (2.8) imenujemo tudi Čebiševe točke.

2.3 Ortogonalnost

Ob primerno izbrani uteži

$$\frac{1}{\sqrt{1-x^2}}, \tag{2.9}$$

lahko na intervalu $[-1, 1]$ vidimo, da ima skalarni produkt Čebiševih baznih polinomov naslednje vrednosti

$$\int_{-1}^{1} T_n(x) T_m(x) \frac{\mathrm{d}x}{\sqrt{1-x^2}} = \begin{cases} 0 & : n \neq m \\ \pi & : n = m = 0 \\ \pi/2 & : n = m \neq 0 \end{cases} \tag{2.10}$$

Izraz lahko preverimo na naslednji način. Najprej upoštevamo zvezo (2.1) in dobimo

$$\int_{-1}^{1} T_n(x)T_m(x)\,\frac{\mathrm{d}x}{\sqrt{1-x^2}} = \int_{-1}^{1} \frac{\cos(n\arccos x)\cos(m\arccos x)\,\mathrm{d}x}{\sqrt{1-x^2}}.$$

Integral lahko izračunamo z uvedbo nove spremenljivke

$$u = \arccos x$$
$$\mathrm{d}u = -\frac{1}{\sqrt{1-x^2}}.$$

Spremenljivko vstavimo v intergral in upoštevamo adicijske izreke

$$\int_0^{\pi} \cos(nu)\cos(mu)\,\mathrm{d}u =$$
$$\int_0^{\pi} \frac{\cos((n+m)u) + \cos((n-m)u)}{2}\,\mathrm{d}u.$$

V primeru $n \neq m$ dobimo

$$\left[\frac{\sin((n+m)u)}{2(n+m)} + \frac{\sin((n-m)u)}{2(n-m)}\right]_{u=0}^{\pi} = 0.$$

V primeru $n = m \neq 0$ dobimo

$$\left[\frac{\sin((n+m)u)}{2(n+m)} + \frac{u}{2}\right]_{u=0}^{\pi} = \frac{\pi}{2}.$$

Če je $n = m = 0$, dobimo

$$\big[u\big]_{u=0}^{\pi} = \pi.$$

Iz rezultata lahko sklepamo, da so Čebiševi polinomi med seboj ortogonalni in jih lahko uporabljamo za bazo prostora polinomov.

2.4 Interpolacija

Pri interpolaciji imamo podane vrednosti $f_0, f_1, \ldots, f_n$ funkcije f v $n+1$ paroma različnih točkah $x_0, x_1, ..., x_n$. Radi bi poiskali interpolacijsko funkcijo $p_n(x)$, ki se vsaj v točkah x_i ujema s funkcijo f. Običajno za interpolacijsko funkcijo vzamemo polinome $P_n(x)$, katere najlažje predstavimo v računalniku. Pokažemo lahko, da obstaja natanko en polinom stopnje n ali manj, ki se v točkah x_i ujema s funkcijo f.

Izrek 2.1 *Za paroma različne točke $x_0, \ldots, x_n$ in vrednosti $f_0, \ldots, f_n$ obstaja natanko en polinom p_n stopnje n ali manj, za katerega velja $p_n(x_i) = f_i$ za $i = 0, 1, \ldots, n$.*

Dokaz. Eksistenco pokažemo s konstrukcijo. Za $i = 0, \ldots, n$ definirajmo polinome

$$\ell_{n,j}(x) = \frac{\prod_{k=0, k\neq j}^{n} (x - x_k)}{\prod_{k=0, k\neq j}^{n} (x_j - x_k)}, \tag{2.11}$$

ki so stopnje n, imenujemo pa jih Lagrangevi koeficienti. Če v definicijo po vrsti vstavimo točke $x_0, \ldots, x_n$, lahko hitro preverimo, da velja

$$\ell_{n,j}(x_k) = \begin{cases} 1, & j = k \\ 0, & j \neq k \end{cases} \qquad j, k = 0, \ldots, n.$$

Vsak izmed Lagrangevih koeficientov $\ell_{n,j}$ ima tako v eni izmed točk $x_0, \ldots, x_n$ vrednost 1, v preostalih pa 0. Skupaj tvorijo bazo za vse polinome stopnje n ali manj. Če sedaj definiramo

$$p_n(x) = \sum_{j=0}^{k} f_j \ell_{n,j}(x), \tag{2.12}$$

je to očitno polinom stopnje $\leq n$, prav tako pa velja $p_n(x_j) = f_j$, za vsak $j = 0, \ldots, n$

Pokažimo še enoličnost. Denimo, da imamo še en interpolacijski polinom $q_n(x)$ stopnje manjše ali enake n, za katerega prav tako velja $q_n(x_j) = f_j$ za $j = 0, \ldots, n$. Razlika $p_n(x) - q_n(x)$ je potem tudi polinom stopnje manjše ali enake n, ki pa ima vsaj $n+1$ ničel v točkah $x_0, \ldots, x_n$. To pa je seveda možno le, če je polinom $p_n(x) - q_n(x)$ identično enak 0 in prišli smo do protislovja. □

Na žalost je takšna interpolacija primerna samo v teoriji in za majhen n, kajti za vsak izračun polinomske vrednosti porabimo $O(n^2)$ seštevanj in množenj. Prav tako je algoritem problematičen, če hočemo dodati novo interpolacijsko vrednost, saj moramo postopek ponoviti od začetka. Algoritem

ima tudi težave z numerično stabilnostjo, kadar so interpolacijske točke tesno skupaj.

2.5 Newtonova oblika interpolacijskega polinoma

V prejšnjemu odseku smo ugotovili, da potrebujemo nov način interpolacije. V praksi največkrat uporabimo Newtonov interpolacijski polinom, za katerega porabimo samo $O(n)$ množenj in seštevanj pri izračunu vrednosti interpolacijskega polinoma.

Za izračun interpolanta moramo najprej izračunati tabelo deljenih diferenc, kot je na sliki 2.1. Deljene diference so določene z rekurzivno formulo

$$f[x_j, x_{j+1}, ..., x_{k-1}, x_k] = \frac{f[x_{j+1}, ..., x_k] - f[x_j, ..., x_{k-1}]}{x_k - x_j} \tag{2.13}$$

in začetnimi pogoji $f[x_j] = f_j$. Algoritem za izračun deljenih diferenc potrebuje n^2 odštevanj in $n^2/2$ deljenj.

Sedaj lahko za vsak x evaluiramo $p(x)$ po Newtonovi interpolacijski formuli

$$p(x) = f[x_0] + f[x_0, x_1](x - x_0) + f[x_0, x_1, x_2](x - x_0)(x - x_1) + ... \\ + f[x_0, x_1, ..., x_n](x - x_0)(x - x_1)...(x - x_n). \tag{2.14}$$

Za izračun vrednosti polinoma z uporabo vgnezdenega množenja porabimo samo n operacij seštevanja in množenja, kar je bistveno bolje kot pri Lagrangeovi formuli. Omeniti velja še, da lahko pri gradnji Newtonovega interpolanta uporabimo tudi vrednosti odvodov v točkah, kar nam omogoča izgradnjo boljših interpolacijskih funkcij.

$$
\begin{array}{lllll}
f[x_0] & & & & \\
 & f[x_0,x_1] & & & \\
f[x_1] & & f[x_0,x_1,x_2] & & \\
 & f[x_1,x_2] & & f[x_0,x_1,x_2,x_3] & \ddots \\
f[x_2] & & f[x_1,x_2,x_3] & \vdots & \qquad f[x_0,x_1,x_2,\ldots,x_n] \\
 & f[x_2,x_3] & & & \iddots \\
f[x_3] & & \vdots & f[x_{n-3},\ldots,x_n] & \\
\vdots & \vdots & f[x_{n-2},x_{n-1},x_n] & & \\
 & f[x_{n-1},x_n] & & & \\
f[x_n] & & & &
\end{array}
$$

Slika 2.1: Tabela deljenih diferenc

2.6 Izboljšava Lagrangeove formule

Za popravek in evaluacijo interpolacijskega polinoma bi radi porabili samo $O(n)$ časa. Najprej bomo na novo zapisali Lagrangeove bazne polinome (2.11). Števec ulomka zapišimo takole

$$l(x) = \frac{(x-x_0)(x-x_1)\ldots(x-x_n)}{x-x_j}. \tag{2.15}$$

Definirajmo baricentrične uteži

$$w_j = \frac{1}{\prod\limits_{k=0,k\neq j}^{n}(x_j-x_k)}. \tag{2.16}$$

Sedaj lahko zapišemo bazne polinome v obliki

$$\ell_j(x) = l(x)\frac{w_j}{x-x_j}. \tag{2.17}$$

Opazimo lahko, da produkt $l(x)$ ni odvisen od j in ga lahko v Lagrangeovi interpolacijski formuli pišemo pred sumandom

$$p(x) = l(x)\sum_{j=0}^{n}\frac{w_j}{x-x_j}f_j. \tag{2.18}$$

Sedaj porabimo $O(n)$ časa za izračun $l(x)$ in $O(n^2)$ časa za izračun w_j. Evaluacija dobljenega interpolanta za poljuben x v tem primeru zahteva samo $O(n)$ časa. Formulo (2.18) lahko poimenujemo: Prva oblika baricentrične interpolacijske formule.

Sedaj poglejmo še, kaj je potrebno narediti, če hočemo dodati novo funkcijsko vrednost f_{n+1} k našemu interpolantu. Najprej moramo deliti baricentrične uteži w_j, za $j = 0, 1, ..., n$ z izrazom $x_j - x_{n+1}$. Pri vsaki uteži w_j porabimo eno deljenje in eno odštevanje. Sledi še izračun uteži w_{n+1} z $n+1$ odštevanji in množenji ter enim deljenjem.

2.7 Baricentrična formula

Izraz (2.18) lahko poenostavimo v elegantnejšo formulo, ki se uporablja v Chebfunu.

Recimo, da poleg poljubne funkcije želimo tu interpolirati tudi konstantno funkcijo 1,

$$1 = l(x) \sum_{j=0}^{n} \frac{w_j}{x - x_y}. \tag{2.19}$$

Če delimo izraz (2.18) z (2.19), dobimo baricentrično formulo

$$p(x) = \frac{\sum_{j=0}^{n} \frac{w_j}{x - x_y} f_j}{\sum_{j=0}^{n} \frac{w_j}{x - x_y}}. \tag{2.20}$$

Izpeljava formule je bila povzeta po [1].
Opazimo lahko, da ima formula zelo lepo simetrijo. Uteži se pojavljajo tako v števcu kot v imenovalcu. Enake faktorje, ki nastopajo v obeh utežeh ulomka, lahko brez problema pokrajšamo in ne vplivajo na izgradnjo interpolacijskega polinoma. Prav tako lahko zopet porabimo samo $O(n)$ časa za izgradnjo novega interpolanta, če dodamo novo funkcijsko vrednost f_{n+1}.

Pri izbiri Čebiševih točk pri interpolaciji funkcije lahko uteži eksplicitno

izračunamo in dobimo

$$w_j = (-1)^j \delta_j, \quad \delta = \begin{cases} 1/2, & j = 0 \text{ ali } j = n \\ 1, & \text{sicer.} \end{cases} \tag{2.21}$$

Poglejmo si Matlabov primer, kjer bomo uporabili iterativni način izračuna Čebiševega interpolacijskega polinoma s pomočjo baricentrične formule. Vzemimo primer funkcije $f(x) = |x| + x/2 - x^2$, katere vrednosti poznamo v 51 točkah. V primeru, da algoritem poženemo v Matlabu, bomo na vsako desetinko sekunde videli novo stanje interpolacijskega polinoma.

```
n = 50;
fun = inline('abs(x)+.5*x-x.^2');

x = cos(pi*(0:n)'/n); % cebiseve tocke na [-1, 1]
f = fun(x); % funkcijske vrednosti funkcije na 51 tockah

c = [1/2; ones(n-1,1); 1/2].*(-1).^((0:n)'); % utezi
xx = linspace(-1,1,50)';
% cebiseve tocke interpolacijskega polinoma na [-1, 1]

numer = zeros(size(xx)); % stevec baricentricne formule
denom = zeros(size(xx));
% imenovalec baricentricne formule

for j = 1:n+1
    % interaktivno dodajanje nove funkcijske vrednosti
    xdiff = xx-x(j); % popravek utezi
    temp = c(j)./xdiff; % na novo izracunana utez
    numer = numer + temp*f(j); % popravek stevca
    denom = denom + temp; % popravek imenovalca
    ff = numer./denom; % koncno deljenje po komponentah
    clf; hold on % risanje grafa
    plot(xx,ff,'-r')
```

```
    plot(x,f,'.')
    pause(0.1)
end
```

2.8 Rungejev problem

Po občutku bi lahko dejali, da bomo z večanjem polinomske stopnje in hkrati večji izbiri funkcijskih vrednosti dobili vedno boljšo aproksimacijo dane funkcije f. Temu na žalost ni tako, saj pride do tako imenovanega Rungejevega problema, ki ga je leta 1901 odkril Carl David Tolmé Runge in nas še danes opozarja, da samo višanje polinomske stopnje ni primeren način za dobro aproksimacijo.

Zgoraj omenjeni problem najlažje prikažemo pri aproksimaciji Runge-jeve funkcije

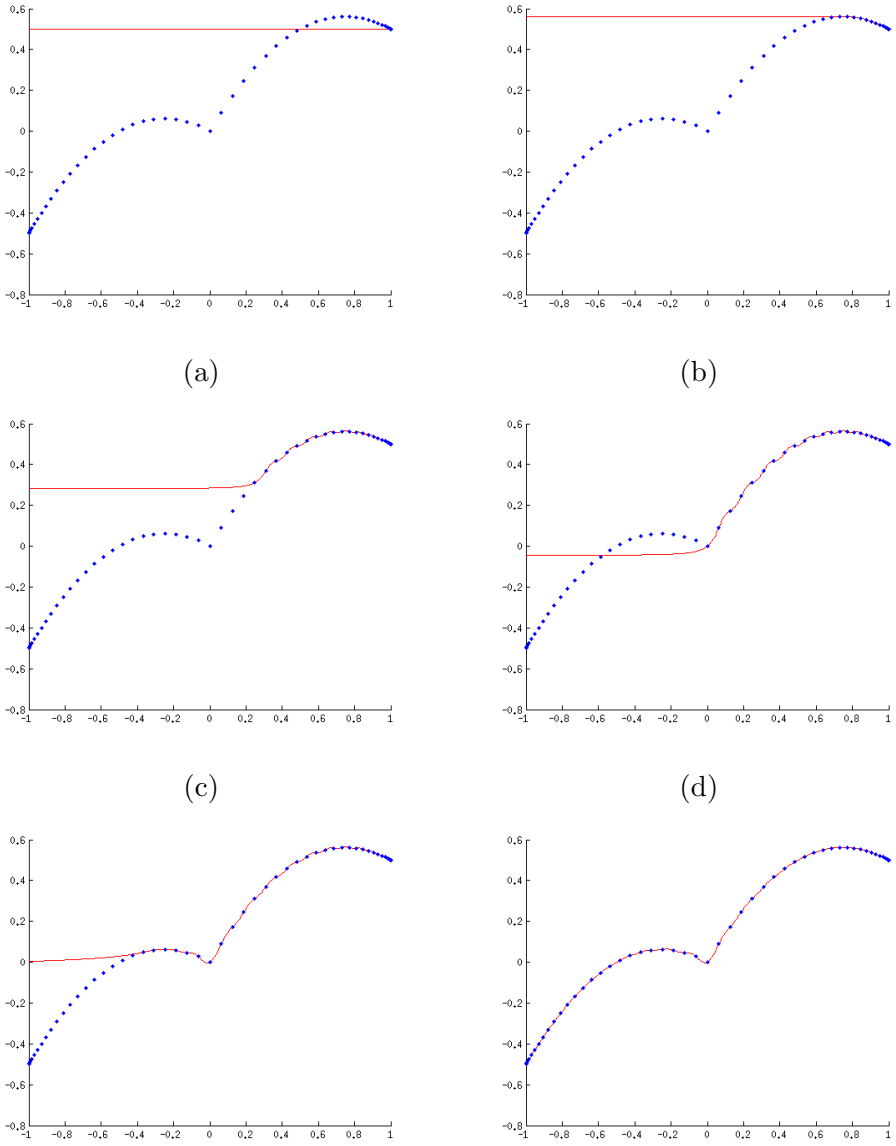
$$f(x) = \frac{1}{1+25x^2} \tag{2.22}$$

na ekvidistantnih točkah na intervalu $[-1, 1]$ in interpolacijskimi polinomi P_n stopnje $\leq n$. Dobljeni interpolanti močno oscilirajo pri robovih intervala. Prav tako je možno pokazati, da ne moremo omejiti napake pri interpolaciji, ko gre $n \to \infty$.

Možnost za popravek interpolacije je izbira ne-ekvidistantnih točk, ki pa jih ni mogoče izbrati na trivialen način. Eden izmed načinov je množica Čebiševih točk (**??**). V naslednjem primeru bomo definirali Rungejevo funkcijo z anonimno funkcijo in izračunali 184 funkcijskih vrednosti. Prav tako bomo izračunali tudi 184 funkcijskih vrednosti iz Čebiševega interpolanta in izračunali nekaj uporabnih norm nad vektorji: prvo, drugo, neskončno in Frobeniusovo.

```
s=linspace(-1,1, 184);
y0=1./(1+25.*s.*s);

cheb_fun=chebfun(@(x) 1/(1 + 25*x*x), 'vectorize');
```

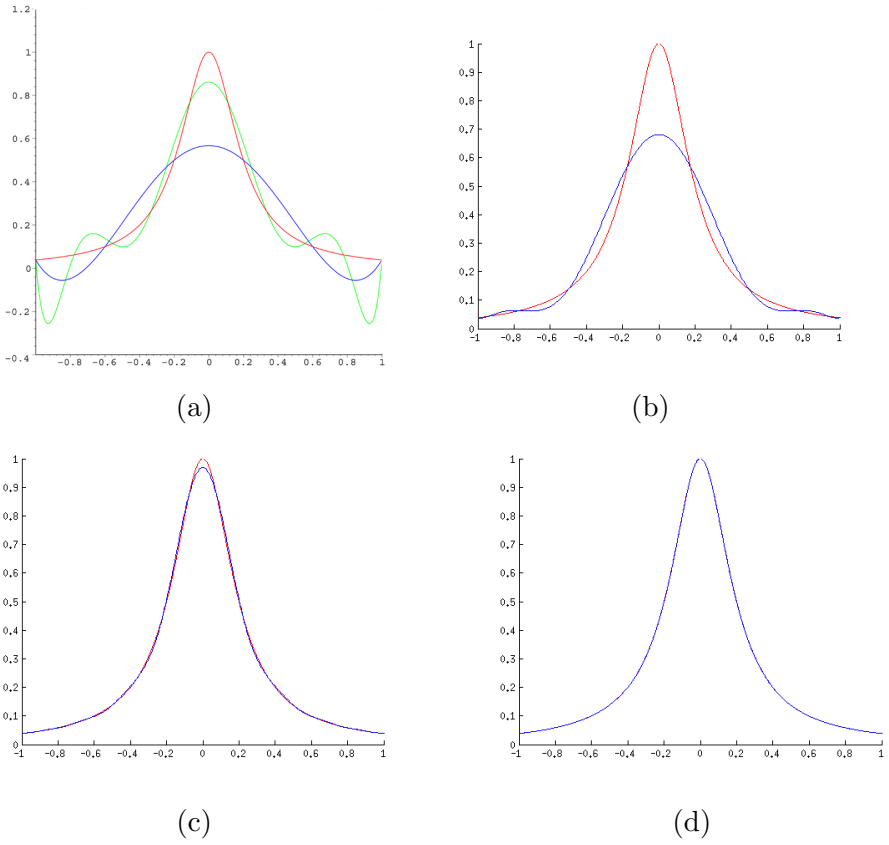


(a) (b)

(c) (d)

(e) (f)

Slika 2.2: Na sliki je šest vmesnih stopenj gradnje interpolacijskega polinoma, obarvanega z rdečo barvo. Opazimo lahko, da se polinom prilega iz leve proti desni, saj tako dodajamo nove interpolacijske točke v baricentrično formulo.



(a) (b)

(c) (d)

Slika 2.3: Na vseh slikah, ki sestavljajo sliko 2.3 vidimo Rungejevo funkcijo rdeče barve. Na sliki (a) je z modro barvo prikazan interpolacijski polinom pete stopnje in z zeleno barvo interpolacijski polinom devete stopnje. Na sliki (b) lahko vidimo z Čebišev interpolant skozi 10 Čebiševih točk (modre barve). Na sliki (c) lahko vidimo interpolacijo skozi 22 Čebiševih točk in na sliki (d) interpolacijo skozi 184 Čebiševih točk.

```
y1=feval(cheb_fun, s);

norm(abs(y0-y1), 1)
norm(abs(y0-y1), 2)
norm(abs(y0-y1), inf)
norm(abs(y0-y1), 'fro')
```

in dobimo rezultate

```
ans = 1.598721155460225e-14
ans = 1.735056510915909e-15
ans = 4.440892098500626e-16
ans = 1.735056510915909e-15
```

Vidimo lahko, da smo z normami zelo blizu ničli v dvojni natančnosti in da je Čebiševa interpolacija dobra.

2.9 Čebiševa vrsta

Vsako dovolj lepo funkcijo lahko predstavimo s Čebiševo vrsto, ki je izraz

$$f(x) = \sum_{k=0}^{\infty} a_k T_k(x)$$

kjer a_k predstavljajo Čebiševe koeficiente, ki so podani s skalarnimi produkti

$$a_0 = \frac{1}{\pi} \int_{-1}^{1} \frac{f(x)T_0(x)}{\sqrt{1-x^2}} \, \mathrm{d}x \tag{2.23}$$

$$a_k = \frac{2}{\pi} \int_{-1}^{1} \frac{f(x)T_k(x)}{\sqrt{1-x^2}} \, \mathrm{d}x, k = 1, 2, ... \tag{2.24}$$

Za obstoj vrste se zahteva, da je funkcija dovolj gladka oziroma vsaj Lipschitzovo zvezna. To je pogojeno s tem, da za vsak $x \in [a, b]$ obstajata $y \in [a, b]$ in realna kostanta $K \geq 0$, ki ustrezajo pogoju

$$|f(x) - f(y)| \leq K|x - y|.$$

Takšna vrsta konvergira absolutno in enakomerno na celotnem intervalu.

Celotna Čebiševa vrsta je problematična za predstavitev v računalniku, saj smo omejeni s končnim prostorom na spominskem mediju. Zato se odločimo, da ga bomo pri zadosti velikem N-ju preprosto odrezali,

$$f_N(x) = \sum_{k=0}^{N} a_k T_k(x).$$

2.10 Primer

V naslednjem osnovnem primeru bomo pogledali, kako izgleda Čebiševa vrsta za funkcijo $f(x) = x^3$.

Najprej izračunajmo koeficiente po enačbah (2.24)

$$\begin{aligned}
a_0 &= \frac{1}{\pi}\int_{-1}^{1} \frac{x^3 * 1}{\sqrt{1-x^2}}\,\mathrm{d}x = 0\\
a_1 &= \frac{2}{\pi}\int_{-1}^{1} \frac{x^3 * x}{\sqrt{1-x^2}}\,\mathrm{d}x = 0.75\\
a_2 &= \frac{2}{\pi}\int_{-1}^{1} \frac{x^3 * (2x^2-1)}{\sqrt{1-x^2}}\,\mathrm{d}x = 0\\
a_3 &= \frac{2}{\pi}\int_{-1}^{1} \frac{x^3 * (4x^3-3x)}{\sqrt{1-x^2}}\,\mathrm{d}x = 0.25\\
a_k &= \frac{2}{\pi}\int_{-1}^{1} \frac{x^3 * T_k(x)}{\sqrt{1-x^2}}\,\mathrm{d}x = 0, \quad k = 4, 5, ...
\end{aligned}$$

Sedaj lahko s pomočjo Čebiševih baznih polinomov zapišemo Čebiševo vrsto

$$f(x) = a_1 T_1(x) + a_3 T_3(x) = 0.75x + 0.25(4x^3 - 3x).$$

Na žalost takšen način ni primeren za interpolacijo zapletenih funkcij, saj imamo lahko probleme pri računanju skalarnih produktov. Težava se pojavi tudi pri izračunu skalarnega produkta, kjer moramo uporabiti funkcijo, katero želimo aproksimirati.

2.11 Interpolacija s Čebiševo vrsto

Ugotovili smo, da za interpolacijo s Čebiševo vrsto potrebujemo drugačen algoritem. Na začetku lahko k problemu pristopimo s pomočjo Vandermon-

dove matrike. Recimo, da imamo funkcijo $f(x) = \cos(x)$ na $[-1, 1]$. Radi bi izračunali Čebiševo vrsto z sedmimi koeficienti. Najprej izračunamo sedem Čebiševih točk, nato pa funkcijo f evaluiramo v pravkar izračunanih točkah

```
>> n = 6;
>> s = cos(pi*(0:n)'/n)
s =
    1.000000000000000
    0.866025403784439
    0.500000000000000
    0.000000000000000
   -0.500000000000000
   -0.866025403784439
   -1.000000000000000
>>fval = cos(s)
fval =

    0.540302305868140
    0.647859344852457
    0.877582561890373
    1.000000000000000
    0.877582561890373
    0.647859344852457
    0.540302305868140.
```

Sedaj nastavimo Čebiševe bazne polinome, jih evaluiramo v Čebiševih točkah in vstavimo v Vandermondovo matriko.

```
>> t0 = @(x) 0*x + 1;
>> t1 = @(x) x;
>> t2 = @(x) 2*x.^2 - 1;
>> t3 = @(x) 4*x.^3 - 3*x;
>> t4 = @(x) 8*x.^4 - 8*x.^2 + 1;
>> t5 = @(x) 16*x.^5 - 20*x.^3 + 5*x;
```

```
>> t6 = @(x) 32*x.^6 - 48*x.^4 + 18*x.^2 - 1;
>> V = [t0(s), t1(s), t2(s), t3(s), t4(s), t5(s), t6(s)];
```

Na koncu še poračunamo dobljeni sistem, obrnemo vektor rešitev in dobimo

```
>> coef = V\fval;
>> coef = flipud(coef)
coef =
   -0.000041876676005
                     0
    0.004953466375103
                     0
   -0.229806970389925
                     0
    0.765197686558967.
```

Izračunani koeficienti si sledijo od najnižjega (a_0) proti najvišjemu (a_6). Za primerjavo lahko še pogledamo koeficiente, ki jih izračuna Chebfun

```
>> f = chebfun(@(x) cos(x), 7);
>> get(f, 'coeffs')
ans =
   -0.000041876676005
   -0.000000000000000
    0.004953466375103
    0.000000000000000
   -0.229806970389925
   -0.000000000000000
    0.765197686558966.
```

Opazimo lahko, da so si rezultati skoraj identično enaki. Omeniti je potrebno, da izračun koeficientov s pomočjo Vandermondove matrike pri velikih n ni primeren, saj je časovno potraten. Prav tako imamo tudi težave z numerično stabilnostjo, kadar so Čebiševe točke tesno skupaj. V programskem paketu Chebfun se namesto Vandermondove matrike uporablja diskretna kosinusna

transformacija, ki pa jo na tem mestu ne bomo predstavili.

Poglavje 3

Začetki s programskim paketom Chebfun

Chebfun je odprto kodni programski paket za Matlab, kjer numerično računamo s funkcijami. V paketu lahko interpoliramo Lipschitz-ove realne funkcije s končnim številom točk nezveznosti na poljubnem intervalu. Nad Čebiševim interpolantom lahko izvajamo matematične manipulacije kot so: integriranje, odvajanje, iskanje ničel, iskanje lokalnih in globalnih ekstremov, risanje funkcij na grafu... Podobne operacije lahko naredimo tudi v kompleksni ravnini. Primer posebne uporabe je ukaz *scribble*, ki v konstruktor sprejme poljubni niz in iz njega izračuna kompleksni interpolant, ki prikazuje tekst v kompleksni ravnini. Chebfun podpira tudi matrike in vse osnovne operacije nad njimi: iskanje lastnih vrednosti in vektorjev, izračun sledi, ranga, norm... Izračuna lahko tudi QR ali singularni razcep. Z njegovo uporabo lahko rešujemo tudi navadne in parcialne diferencialne enačbe ene ali dveh spremenljivk. Podpira tudi matematične operacije nad funkcijami dveh spremenljivk. Spisek vseh ukazov lahko dobimo z ukazoma

```
>> methods chebfun
```

in

```
>> methods chebfun2
```

Sedaj opišimo razliko med besedama Chebfun in chebfun. Prva opisuje odprtokodno programsko okolje, druga pa opisuje aproksimacijski polinom ene spremenljivke, definiran na intervalu $[a, b]$. Cilj programskega paketa je dobiti občutek, kot da računamo simbolno, vendar je hitrost izračuna taka, kot da delamo s števili ("feel symbolic but run at the speed of numerics"). Vizija razvoja paketa gre proti temu, da se za funkcije doseže tisto, kar velja za plavajočo vejico pri aritmetičnih operacijah: hitro računanje in najmanjša matematična napaka pri standardu IEEE 754 za števila v dvojni natačnosti v vsakem koraku [2]. IEEE 754 je računalniški standard za računanje s števili v plavajoči vejici. Standard je danes uporabljen skoraj v vseh novejših računalnikih, kjer so definirani formati števil, matematične operacije, načine zaokroževanja...

Za enostavne funkcije je zadosti 20 do 30 aproksimacijskih točk, algoritmi pa so dobri in stabilni tudi pri zahtevnejših funkcijah, ki zahtevajo 1000 ali 1.000.000 aproksimacijskih točk. Programski paket vsebuje adaptivne metode, katerih namen je, da same najdejo zadostno število aproksimacijskih točk in da je na vsakem koraku natančnost vrednosti funkcije na 15 decimalnih mest. Algoritem najprej poskusi narediti chebfun stopnje 8 skozi 9 Čebiševih točk in nato pogleda, ali se zadnji koeficienti približujejo ničli. V primeru, da temu ni tako, poskusi narediti chebfun stopnje 16, 32, 64 in tako naprej. Recimo, da imamo primer, kjer je polinom stopnje 256 ustrezen, vendar je že 151 koeficient zadosti majhen, prav tako pa obstaja tudi delno zaporedje koeficientov, ki gredo proti ničli v dvojni natančnosti. V tem primeru koeficiente od 151 člena naprej odrežemo in kot rezultat dobimo interpolant 150 stopnje.

Programski paket Chebfun je objektno orientiran in definiran znotraj direktorija *@chebfun*, ki ga prenesemo s spletne strani `http://www.chebfun.org/download/`. Njegovi ukazi so podobni ukazom, ki jih Matlab izvaja nad vektorji števil, vendar pa so ukazi v programskem paketu Chebfun namenjeni manipulaciji nad funkcijami. Ukazom se spremeni pomen, analogija med diskretnim in zveznim prostorom pa ostaja ista, na primer: navadni ukaz *sum*

sešteje števila v vektorju, pri Čebiševem konstruktorju pa izračuna integral funkcije, oziroma ukaz *min* najde najmanjše število v vektorju, pri Čebiševem konstruktorju pa najde minimum funkcije. Najbolj osnovna operacija tega paketa je izgradnja chebfun-a s klicem njegovega konstruktorja. Na primer

```
>> f = chebfun('cos(20*x)').
```

V argument konstruktorja lahko podamo funkcijo v obliki niza *'sin(x)'*, kot anonimno funkcijo; na primer $@(x)x.^2 + 2 * x + 1$ ali kot vektor števil, ki predstavljajo funkcijske vrednosti. V prvih dveh primerih se bo funkcija "vektorizirala" v smislu, da se izračuna nekaj funkcijskih vrednosti. Dobra zasnovanost paketa Chebfun-a se kaže tudi v tem, da lahko sestavimo nov chebfun s pomočjo že obstoječega chebfun-a, na primer

```
>> y = chebfun('x')
y =
    chebfun column (1 smooth piece)
        interval        length    endpoint values
[       -1,         1]          2         -1          1
Epslevel = 1.110223e-15.   Vscale = 1.

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

Osnova izgradnje novega Chefuna je preprosto dejstvo, da se vsak nov interpolacijski polinom sestavi iz funkcijskih vrednosti na nekem intervalu. V našem primeru lahko eksplicitno zahtevamo, da se funkcija evaluira v nekaj točkah in dobimo isti rezultat.

```
>> z = chebfun(@(x) sin(feval(y, x)))
z =
```

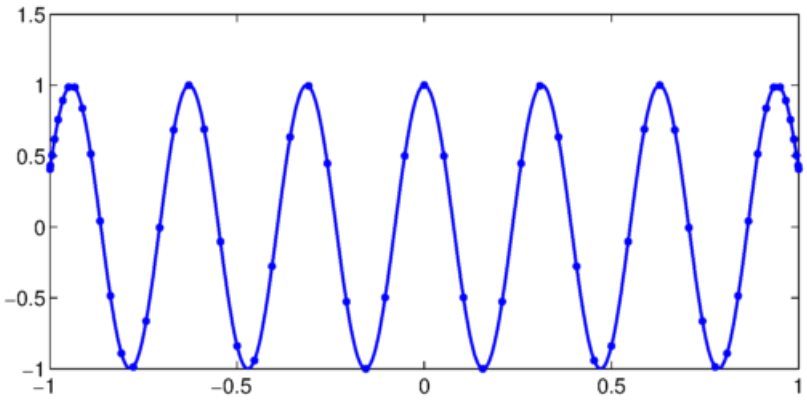
```
    chebfun column (1 smooth piece)
         interval       length     endpoint values
[        -1,         1]        14       -0.84        0.84
Epslevel = 1.110223e-15.   Vscale = 8.414710e-01.
```

V primeru, da eksplicitno ne podamo intervala, se privzame interval $[-1, 1]$, v nasprotni situaciji se poljubni interval $[a, b]$ skrči na $[-1, 1]$ po naslednji zvezi

$$x = \frac{2\,t - a - b}{b - a}, \quad t \in [a, b].$$

Interpolacijski polinom lahko tudi narišemo z ukazom

```
$ plot(f,'.-')
```



Slika 3.1: Opazimo lahko, da imamo v sredini interpolacijski polinom poračunan v 5-tih točkah, kar je minimalno potrebno, da lahko predstavimo eno kosinusno periodo na 15 decimalk natančno.

Poglavje 4

Interpolacija odsekoma zveznih funkcij

V praksi se velikokrat soočamo s funkcijami, ki na željenem intervalu niso Lipschitzovo zvezne. V tem primeru lahko sestavimo chebfun iz več kosov, ki jih imenujemo "fun". Vsak fun je sestavljen iz svojega zveznega polinoma na podintervalu. V primeru, da podamo v konstruktor chebfun-a funkcijo, ki ni zvezna na danem intervalu, bo chebfun poskusil sestaviti ustrezni interpolacijski polinom, vendar v večini primerov ne bo uspel in nam bo posredoval naslednje sporočilo.

```
Warning: Function not resolved using 65537 pts.
Have you tried 'splitting on'?
```

V izogib težavam lahko uporabimo Chebfun-ov vgrajen algoritem, ki je sposoben zaznavati točke nezveznosti na interpolacijskem polinomu. Uporabimo ga lahko z naslednjim ukazom

```
>> f = chebfun('abs(x-3)','splitting','on')

f =
    chebfun column (2 smooth pieces)
         interval        length    endpoint values
[       -1,      0.3]          2         1.3   4.4e-16
```

```
[      0.3,      1]      2   -1.1e-16      0.7
Epslevel = 1.110223e-15.   Vscale = 1.300000e+00.
Total length = 4.
```

Opazimo lahko, da smo dobili dva fun-a. Prvi živi na intervalu $[-1, 0.3]$ in drugi na $[0.3, 1]$. Prav tako lahko sklepamo, da vsak fun predstavlja premico, saj je dolžina vsakega segmenta enaka 2 in tudi končne vrednosti se prilegajo grafu funkcije. V primeru, da dobro poznano funkcijo, ki jo želimo interpolirati, je bolje, da v konstruktor podamo sestavljeno funkcijo in podintervale. Prav tako se ne smemo zanašati na algoritem iskanja točk nezveznosti, kajti hitro lahko dobimo chebfun s polno fun-i pri zvezni funkciji na celotnem intervalu. Algoritem za iskanje točk nezveznosti uporabi tri koncepte. Prvi je ta, da poskuša najti majhen interval na osi x, kjer se vrednosti funkcije razlikujejo za več kot osnovni ϵ. V drugem primeru gleda vrednost odvoda funkcije v neki točki. Če je ta zelo blizu maksimalnega ali minimalnega še predstavljivega števila, ga lahko detektira kot točko nezveznosti. Kot zadnji primer detektira minimume ali maksimume funkcij, katerih se ne da več predstaviti v dvojni natančnosti.

Poglavje 5

Primeri uporabe programskega paketa

Na tem mestu bi bilo smiselno omeniti in pokazati še primere uporabe Chebfuna pri manipulacijah nad interpolacijskimi funkcijami, kjer lahko začutimo, da lahko delamo samo s funkcijami v simbolni obliki, čeprav se v ozadju vedno opravlja računanje z vektorji števil. Recimo, da imamo funkciji f in g podani na naslednji način

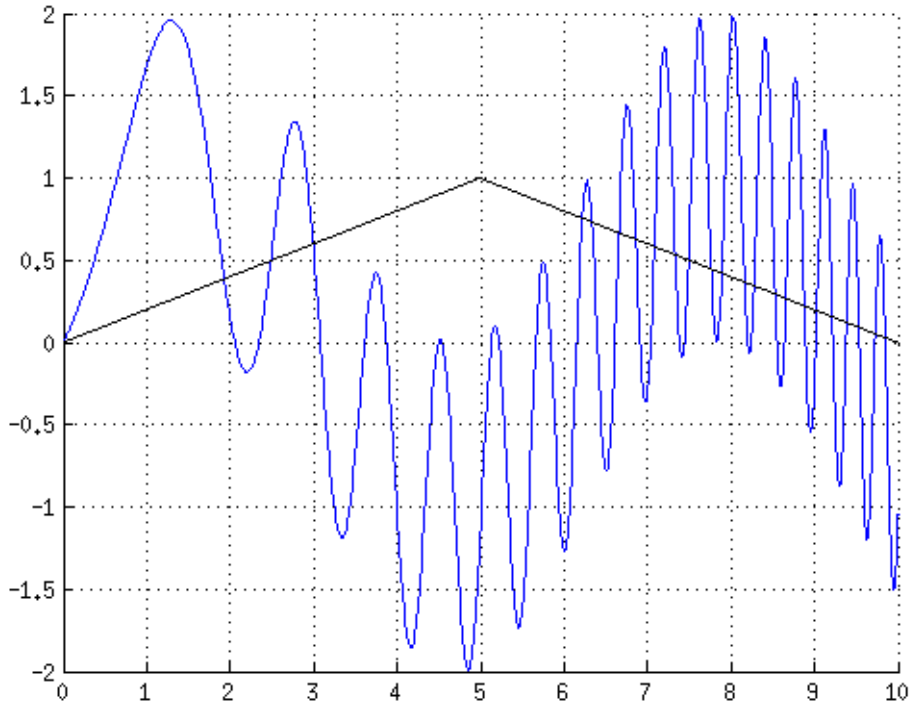
```
x = chebfun('x', [0, 10]);
f = sin(x) + sin(x.^2);
g = 1 - abs(x-5)/5;
```

Radi bi izračunali maksimum funkcij na intervalu $[0, 10]$. To naredimo preprosto z ukazom

```
>> h = max(f, g)
```

in dobimo funkcijo, ki ima 25 fun-ov na intervalu $[0, 10]$.

```
h =
    chebfun column (25 smooth pieces)
         interval       length    endpoint values
[         0,     1.9]        24    1.2e-15      0.39
[       1.9,     2.5]         2       0.39       0.5
```



Slika 5.1: Z modro barvo je narisana funkcija f, ki je zvezni interpolant stopnje 119. S črno barvo pa je narisana funkcija g, ki jo zaradi absolutne vrednosti sestavljata 2 fun-a prve stopnje.

[2.5,	3]	16	0.5	0.6
[3,	5]	2	0.6	1
[5,	6.2]	2	1	0.76
[6.2,	6.3]	13	0.76	0.73
[6.3,	6.7]	2	0.73	0.67
[6.7,	6.9]	16	0.67	0.63
[6.9,	7.1]	2	0.63	0.58
[7.1,	7.3]	17	0.58	0.53
[7.3,	7.5]	2	0.53	0.5
[7.5,	7.8]	18	0.5	0.45
[7.8,	7.9]	2	0.45	0.42

```
[     7.9,     8.2]      18      0.42      0.37
[     8.2,     8.3]       2      0.37      0.34
[     8.3,     8.5]      18      0.34      0.29
[     8.5,     8.7]       2      0.29      0.27
[     8.7,     8.9]      17      0.27      0.22
[     8.9,       9]       2      0.22      0.19
[       9,     9.2]      17      0.19      0.16
[     9.2,     9.4]       2      0.16      0.12
[     9.4,     9.5]      16      0.12     0.093
[     9.5,     9.7]       2     0.093     0.055
[     9.7,     9.8]     118     0.055     0.031
[     9.8,      10]       2     0.031         0
Epslevel = 3.330669e-15.   Vscale = 1.985442e+00.
Total length = 334.
```

Če želimo najti vse lokalne ekstreme, funkcijo najprej odvajamo, nato pa poiščemo njene ničle. Odvod funkcije lahko izračunamo z ukazom *diff*, ničle funkcije pa najdemo z ukazom *roots*.

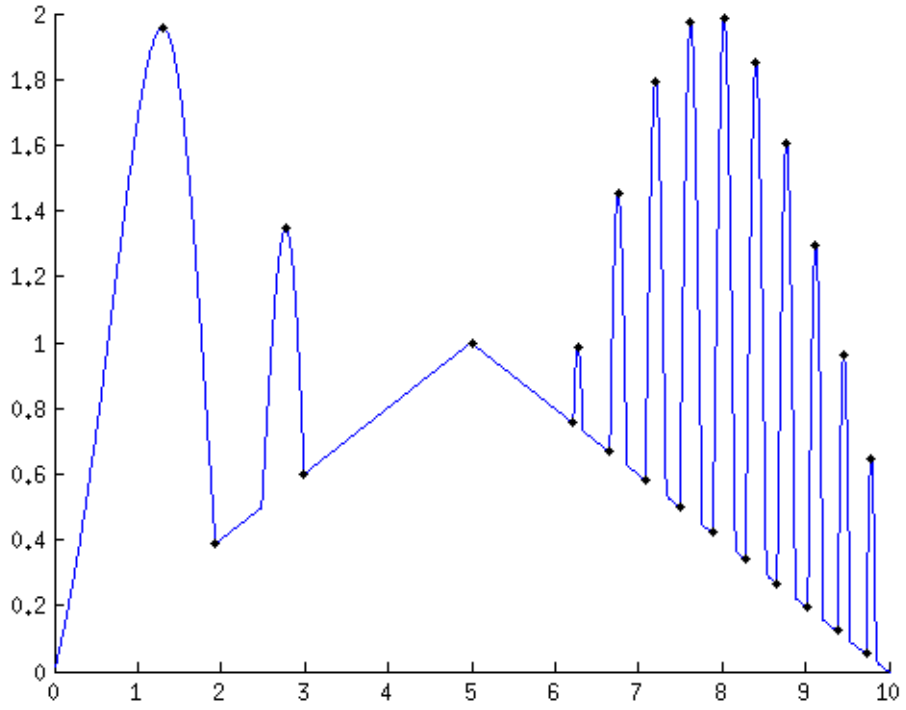
```
hp = diff(h);
extrema = roots(hp);
```

Za lažjo predstavo bomo narisali funkcijo in vse njene lokalne ekstreme (slika 5.2) z naslednjimi tremi ukazi.

```
hold on
plot(h)
plot(extrema, h(extrema), '.k')
```

Naprej poiščemo vse lokalne maksimume funkcije h (slika 5.3) na naslednji način

```
hpp = diff(h,2);
mx = extrema(hpp(extrema) < 0);
```



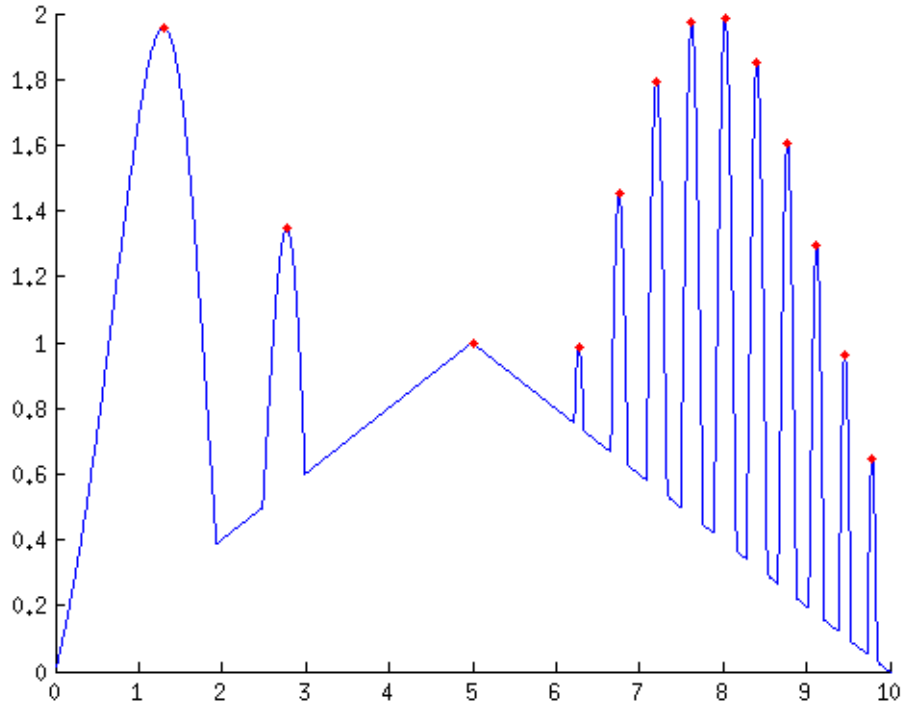
Slika 5.2: Lokalni ekstremi funkcije $h = \max(f, g)$.

5.1 Integriranje

V naslednjem primeru lahko vidimo numerično stabilnost algoritma za integriranje in točno vrednost rezultata na 15 decimalk natančno. Ukaz za integriranje funkcij na intervalu $[-1, 1]$ je *sum*.

```
f = chebfun(@(x) log(1+tan(x)),[0 pi/4]);
format long
I = sum(f)
Iexact = pi*log(2)/8
```

```
I =
   0.272198261287950
Iexact =
```



Slika 5.3: Lokalni maksimumi funkcije $h = \max(f, g)$.

```
0.272198261287950
```

Na žalost pa se izračun integrala dosti počasneje izvede, če imamo chebfun sestavljen iz več funov. Kot primer izračunajmo integral absolutne Besselove funkcije prvega reda na intervalu $[0, 20]$.

Najprej definiramo funkcijo kot anonimno Matlab-ovo funkcijo in kot Čebišev interpolant, ter nato poženimo nekaj algoritmov za izračun integrala, ter izmerimo njihove čase.

```
F = @(x) abs(besselj(0,x));
f = chebfun(@(x) abs(besselj(0,x)),[0 20],
    'splitting','on');

tol = 3e-14;
```

```
tic, I = quad(F,0,20,tol); t = toc;
  fprintf('   QUAD:  I = %17.15f  time = %5.3f secs\n',I,t)
tic, I = quadl(F,0,20,tol); t = toc;
  fprintf('  QUADL:  I = %17.15f  time = %5.3f secs\n',I,t)
tic, I = quadgk(F,0,20,'abstol',tol,'reltol',tol); t = toc;
  fprintf('QUADGK:  I = %17.15f  time = %5.3f secs\n',I,t)
tic,
I = sum(chebfun(
          @(x) abs(besselj(0,x)),[0,20],'splitting','on'));
t = toc;
  fprintf('CHEBFUN:  I = %17.15f  time = %5.3f secs\n',I,t)
```

V nadaljevanju so rezultati ploščin in časi merjenj

```
QUAD:     I = 4.445031603001505  time = 0.154 secs
QUADL:    I = 4.445031603001576  time = 0.076 secs
QUADGK:   I = 4.445031603001578  time = 0.026 secs
CHEBFUN:  I = 4.445031603001566  time = 2.189 secs
```

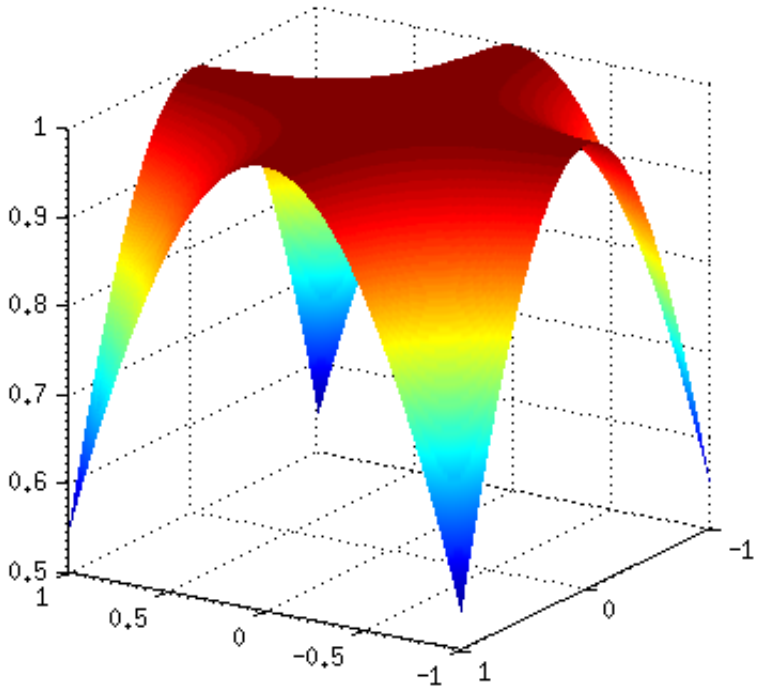
Uporabili smo tri vgrajene Matlabove algoritme za integriranje funkcij in ukaz *sum* za integriranje Čebiševega interpolacijskega polinoma. Ker je interpolacijski polinom sestavljen iz sedmih fun-ov, se čas računanja zavleče za 10 do 20 krat. Chebfun je v osnovi orodje za manipulacijo funkcij in integriranje je samo del orodja. Prav tako trenutno še nima zadosti podpore s strani razvijalcev, ki bi lahko naredili več na optimizaciji algoritmov.

Podobno lahko vidimo, da lahko integriramo tudi funkcije dveh spremeljivk(slika 5.4) z ukazom *sum2*, na primer

```
f = chebfun2(@(x,y) cos(x.*y));
sum2(f)
ans =  3.784332281468732.
```

Če preverimo še rešitev v programu Mathematica, opazimo, da dobimo enak rezultat

```
In[1]:= Integrate[Cos[x*y], {y, -1, 1}, {x, -1, 1}]
```



Slika 5.4: Graf funkcije $f(x, y) = \cos(xy)$ na intervalu $[-1, 1] \times [-1, 1]$.

```
Out[1]:= 3.784332281468732.
```

5.2 Reševanje diferencialnih enačb

Način reševanja navadnih linearnih diferencialnih enačb si lahko ogledamo v naslednjem primeru. Recimo, da imamo funkcijo $0.0001u'' + xu = 0$ na intervalu $[-1, 1]$ in z robnima pogojema $u(-1) = 0$ in $u(1) = 1$.

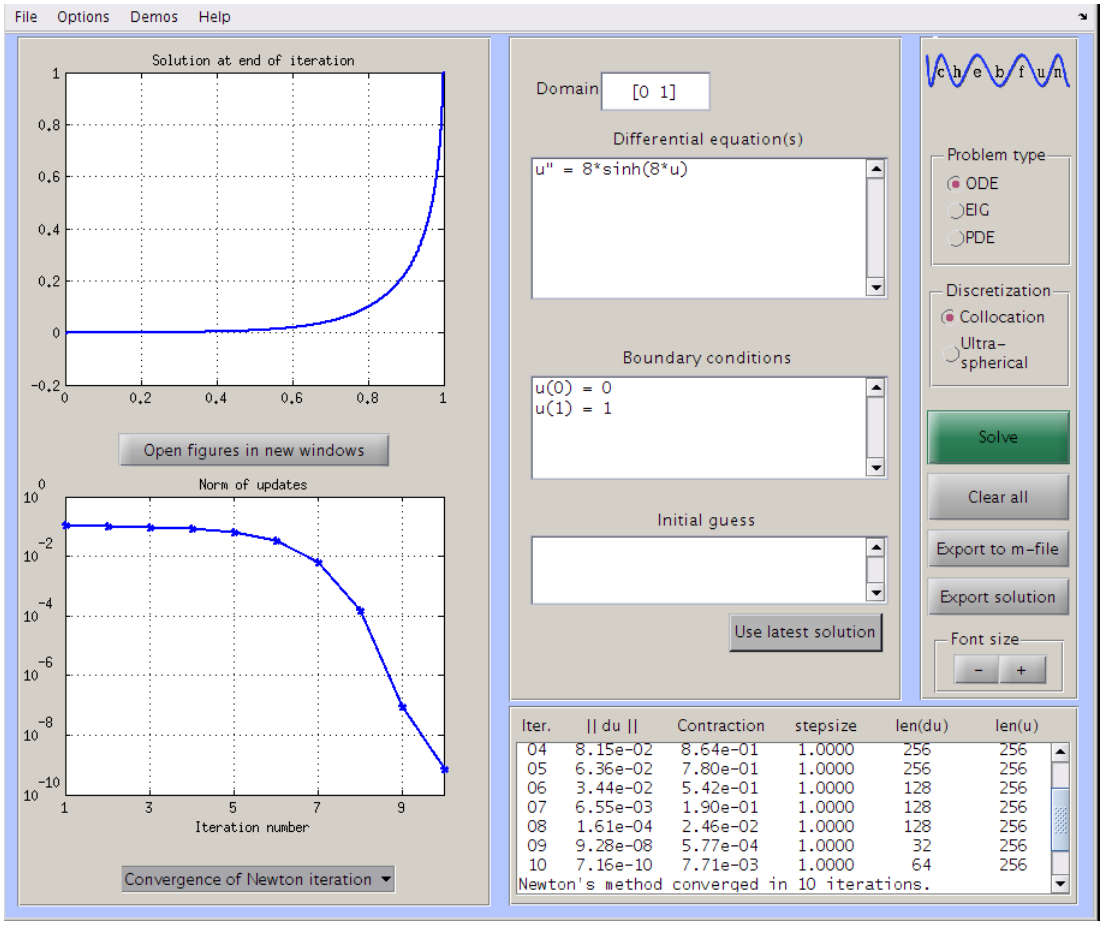
Problem predstavimo s konstruktorjem *chebop*, kateri omogoča enostavno reševanje s klicem operatorja \. *Chebop* predstavlja ogrodje, v katerega podamo vse karakteristike diferencialne enačbe: funkcijo, ki želimo rešiti, inter-

val, na kateri je definirana, robne in začetne pogoje,

```
L = chebop(-1, 1);
L.op = @(x,u) 0.0001*diff(u,2) + x.*u;
L.lbc = 0;
L.rbc = 1;
u = L\0;
```

Diferencialne enačbe lahko rešujemo s pomočjo ukaza *chebgui*, ki odpre grafični vmesnik, kot je prikazano na sliki 5.5. Grafični vmesnik poenostavi vnos vhodnih podatkov za reševanje diferencialnih enačb. Omogoča tudi, da vnešeno diferencialno enačbo lahko izvozimo kot zaporedje ukazov, ki jih lahko shranimo v Matlabovo skripto. Skripto lahko naknadno poženemo iz Matlabove komandne vrstice.

Več praktičnih primerov matematični problemov lahko najdemo na uradni spletni strani projekta [4].



Slika 5.5: Grafični vmesnik za pomoč pri reševanju diferencialnih enačb.

Literatura

[1] J. Berrut and L. Trefethen. Barycentric lagrange interpolation. SIAM Review, 46(3):501–517, 2004.

[2] Lloyd N. Trefethen. Computing numerically with functions instead of numbers. Mathematics in Computer Science, 1(1):9–19, 2007.

[3] T. A. Driscoll, N. Hale, and L. N. Trefethen, editors, Chebfun Guide, Pafnuty Publications, Oxford, 2014.

[4] http://www.chebfun.org, dostop 1-9-2014.

[5] http://en.wikipedia.org/wiki/Chebyshev_polynomials, dostop 1-9-2014.

[6] https://github.com/chebfun/examples/blob/development/approx/ChebfunFFT.m, dostop 1-9-2014.