Sveučilište u Zagrebu
Prirodoslovno–matematički fakultet

Mario Berljafa, Sara Muhvić, Melkior Ornik

Računanje Gaussovih integracijskih formula za sažimajuću bazu

Zagreb, 2011.

Ovaj rad izrađen je na Zavodu za numeričku matematiku i računarstvo na Matematičkom odsjeku Prirodoslovno–matematičkog fakulteta, pod vodstvom prof. dr. sc. Saše Singera i predan je na natječaj za dodjelu Rektorove nagrade u akademskoj godini 2010./2011.

Sadržaj

1 Uvod

Traženje integrala funkcije pokazuje se korisnim iz više razloga. Temeljni je razlog određivanje površine dijela ravnine omeđenog nekom krivuljom. Uvođenje pojma integrala pomaže nam formalizirati ideju površine i stvoriti efikasan način za određivanje površine. No, i u ovoj, naizgled vrlo jednostavnoj, primjeni integrala nailazimo na njihov osnovni problem – integrale nekih "prirodnih" funkcija, kao na primjer e^x/x, ne možemo egzaktno izračunati. Ipak, želimo nekako odrediti tu površinu – možda ne baš egzaktno, ali s greškom dovoljno malom da nam postane inkonzekventna.

Također, imamo još barem jednu drugu motivaciju za razmišljanje o integralima: diferencijalne jednadžbe. Očiti kontekst upotrebe ove matematičke teorije je u primjeni matematike u, na primjer, mehanici.

Ovo nas i dovodi do, šire gledano, problema kojim se ovaj rad bavi. Kao što smo već napomenuli, integrale nekih funkcija nećemo moći odrediti egzaktno. No dobro, ponekad nam egzaktnost neće ni biti moguće, a ni pretjerano bitno, postići, već iz metode kojom smo prikupili podatke o funkciji koju integriramo (sva stvarna fizikalna ili druga mjerenja su na diskretnom – dapače, konačnom – skupu podataka, pa iz njih tek možemo modelirati funkciju, a ne tvrditi da je egzaktno znamo). U svakom slučaju, smisleno nam je tražiti aproksimaciju integrala.

Nemoguće je stvoriti općeniti algoritam, pa ima smisla stvarati algoritme koji će služiti za rješavanje određene klase problema. Upravo fizikalni problem koji stoji u pozadini našeg rada, a radi se o problemu ponašanja štapa pri stiskanju s obje strane (v. [1], 20. poglavlje), garantira smisao i korist algoritma kojeg ćemo prezentirati.

Da bismo bolje opisali kontekst našeg rada, možda je najbolje krenuti sa stvarnim potrebama za algoritmom kojeg osmišljavamo. Motivacija za

problem dolazi iz područja običnih diferencijalnih jednadžbi generiranim gore spomenutim "problemom štapa", gdje se prirodno može pojaviti potreba za integriranjem funkcija iz vektorskog prostora razapetog skupom

$$\{1, x, x^2, \ldots, x^{2n-3}, \sin px, \cos px\},$$

kojeg zovemo *sažimajuća baza*. Budući da parametar p nije *a priori* poznat (tj. u svakoj praktičnoj primjeni može biti drugačiji), želimo napraviti algoritam koji prima p kao ulaz i nakon toga vrši integraciju u ovisnosti o p.

U ovom radu iznosimo algoritam, s implementacijom u programskom jeziku MATLAB/GNU Octave, kojim, za unos prirodnog broja n i parametra p, dobivamo n čvorova integracije i n težina Gaussove integracijske formule koja *egzaktno* integrira tražene funkcije na intervalu $[-1, 1]$. Naposljetku, predstavljamo rezultate dobivene korištenjem algoritma, uz neke prirodne parametre, te komentiramo veličinu dobivene izračunate pogreške integracije u sažimajućoj bazi.

Stoga, ovaj rad po prvi puta daje eksplicitni algoritam koji omogućava egzaktnu integraciju funkcija sažimajuće baze. S čisto matematičkog stajališta, takav algoritam je interesantan kao uspješan primjer modifikacije Gaussovih integracijskih formula na nepolinomne funkcije. S pozicije nestriktno-matematičke koristi, ovaj algoritam brzo i s velikom preciznošću daje gotove rezultate koji se dalje mogu odmah koristiti, na primjer, za točno računanje tzv. cikloidnih splajnova [2, 16]

2 Algoritam – osnovna ideja

Algoritam koji predstavljamo bazira se na poznatoj metodi Gaussove integracije. Može se pokazati da za sažimajuću bazu, uz ograničenje $p \in \langle 0, \pi \rangle$, postoje jedinstvene Gaussove formule [4, 9, 14], a očita je povezanost ova dva

problema. Generalnije, može nas zanimati proširivanje Gaussove integracijske metode na funkcije generirane skupom $\{1, x, x^2, \ldots, x^k, f(x), g(x)\}$. No, takav se zadatak pokazuje numerički znatno problematičnijim. Specifično, u našem ćemo algoritmu koristiti posebna svojstva trigonometrijskih funkcija: da je sinus neparna funkcija, a od velike se važnosti pokazuje i relativno lako određivanje Legendreovog reda funkcije kosinus.

Prije no što prijeđemo na kratak opis koncepata koje će naš kôd koristiti, svakako je potrebno diskutirati neke detalje algoritma kojeg predstavljamo. Da ponovimo, želimo s velikom preciznošću integrirati funkcije iz prostora $\operatorname{span}\{1, x, x^2, \ldots, x^{2n-3}, \sin px, \cos px\}$, na intervalu $[-1, 1]$[1], uz dane parametre $n \in \mathbb{N}, n \geq 2$, i $p \in \langle 0, \pi \rangle$.

Budući da je baza našeg vektorskog prostora integracije "većinom" polinomna, zanimat će nas modifikacija uobičajene matrice korištene u Gaussovoj integraciji:

$$A = \begin{pmatrix} \alpha_1 & \beta_1 & & & & & \\ \beta_1 & \alpha_2 & \beta_2 & & & & \\ & \beta_2 & \alpha_3 & \beta_3 & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & \ddots & \ddots & \ddots & \\ & & & & \beta_{n-2} & \alpha_{n-1} & \beta_{n-1} \\ & & & & & \beta_{n-1} & \alpha_n, \end{pmatrix}.$$

Zbog upravo spomenute "većinske" uloge polinoma u integracijskoj bazi i zbog neparnosti funkcije sinus, može se dobiti da vrijedi $\alpha_j = 0$ za sve j, a

$$\beta_j = \frac{j^2}{4j^2 - 1}, \qquad \text{za } j = 0, 1, \ldots, n-2.$$

[1]Uz tradicionalnu afinu transformaciju $f(x) = 2 \cdot \dfrac{x-a}{b-a} - 1$, algoritam se može lako proširiti na svaki interval $[a, b]$, za $a, b \in \mathbb{R}$.

U nastavku, algoritam je analogan onome za Gaussovu integraciju, uz dodatan problem traženja posljednjeg koeficijenta β_{n-1}. To činimo metodom bisekcije, pri čemu "točnost" koeficijenta određujemo greškom pri integraciji kosinusa. Naposljetku, čvorove integracije i njihove težine određujemo koristeći poznate formule preko svojstvenih vrijednosti i prvih komponenata svojstvenih vektora (tzv. Golub–Welsch algoritam [5, 8, 10, 11]).

Iz ovog kratkog opisa algoritma, jasno je koji će nam matematički koncepti biti od koristi. Prije svega, očito je diskusija o numeričkoj integraciji nezaobilazna. Specifično, naš se algoritam bazira na Gaussovoj metodi integracije.

Nadalje, budući da tražimo zadnji koeficijent, pokazuje se da se radi o pitanju minimiziranja greške integracije – o tome više kasnije. U tu svrhu, morat ćemo prvo naći grešku, u ovisnosti o rečenom koeficijentu. Za to će nam trebati, kako smo gore napisali, svojstvene vrijednosti i svojstveni vektori niza matrica. Dakle, trebamo koristiti neki algoritam za određivanje svojstvenih vrijednosti. Naravno, postoje razni takvi algoritmi (npr., Jacobijev i inverzne iteracije), pri čemu ćemo mi diskutirati i koristiti QR algoritam, koji je najprirodniji za vrstu matrice[2] s kojom radimo.

Također, zbog određenih "loših" svojstava sažimajuće baze, koja ćemo isto spomenuti poslije, bit će potrebno koristiti razvoj funkcije kosinus po Legendreovim polinomima. Dakle, diskutirat ćemo i mogućnosti aproksimiranja funkcije s konačno mnogo članova reda.

Naposljetku, vratimo se minimizaciji same greške. Zapravo, radi se o traženju nultočke poznate, ali vrlo složene funkcije. Da bismo to postigli, koristit ćemo (a prije i diskutirati) vrlo jednostavan, ali u našim uvjetima idealan algoritam – metodu bisekcije.

Primijetimo, dakle, kako nam je za rješavanje ovog, vrlo jednostavno

[2]Kao što se vidi, radi se o simetričnim trodijagonalnim matricama.

postavljenog problema, potrebno znanje širokog spektra metoda numeričke matematike – od algoritama za svojstvene vrijednosti i vektore, do metoda za traženje nultočaka. Stoga, sljedećih nekoliko dijelova rada posvećujemo upravo spomenutim metodama.

3 Numerička integracija

3.1 Potreba za numeričkom integracijom

Neka je $f : I \to \mathbb{R}$ realna funkcija definirana na otvorenom intervalu $I \subseteq \mathbb{R}$. Kao problem nameće se pronaći integral $I(f)$ funkcije f na $[a, b] \subseteq I$,

$$I(f) = \int_a^b f(x)\,dx.$$

Ako s F označimo primitivnu funkciju od f, koristeći Newton–Leibnizovu formulu, imamo $I(f) = F(b) - F(a)$. No, naći primitivnu funkciju nije lagan zadatak. Dapače, ponekad ju je i nemoguće zapisati analitički. Također, u nekim drugim slučajevima možemo dobiti zatvorenu formulu za primitivnu funkciju, no s njom i dalje može biti vrlo teško računati. Uz navedeno, motivacija za aproksimacijom integrala jest nepoznavanje same funkcije f. Naime, funkciju ne moramo zadati analitički, već ona može biti rezultat nekog mjerenja – dakle, imamo (ne nužno niti sasvim točne) podatke o njoj samo na diskretnom skupu točaka.

Upravo se na ovom zadnjem bazira osnovna ideja aproksimacije integrala. Neka je n broj točaka $x_1, x_2, \ldots, x_n \in [a, b]$ u kojima znamo vrijednost funkcije f. Aproksimacija $I_n(f)$ integrala $I(f)$ je oblika

$$I_n(f) = \sum_{i=1}^{n} w_i f(x_i).$$

Točke $x_1, x_2, \ldots, x_n$ nazivamo čvorovima integracije, a brojeve $w_1, w_2, \ldots, w_n$ težinskim koeficijentima, ili samo težinama. Osnovni problem numeričke

integracije jest određivanje čvorova i težina takvih da je pritom napravljena greška po modulu što manja, odnosno, da je aproksimacija $I_n(f)$ što preciznija.

Grešku $E_n(f)$ napravljenu aproksimacijom definiramo relacijom

$$E_n(f) = I(f) - I_n(f).$$

Obično se traži da je aproksimacijska formula egzaktna na vektorskom prostoru polinoma $\mathcal{P}_m$ što veće dimenzije. Na taj će način aproksimacijska formula egzaktno integrirati početne članove Taylorovog reda funkcije f (ako pripadni Taylorov red postoji i konvergira) te će greška, u načelu, biti mala. Kako je integral linearni funkcional, dovoljno je odrediti koeficijente uzimajući kao uvjet da aproksimacijska formula integrira egzaktno npr. kanonsku bazu za $\mathcal{P}_m$. Ukoliko unaprijed zadamo čvorove integracije dobivamo Newton–Cotesove formule, a ako određujemo i čvorove, dobivamo Gaussove formule.

3.2 Newton–Cotesove formule

Kako smo već naveli, kod Newton–Cotesovih formula čvorove integracije zadajemo unaprijed. Uobičajeno, zadaje se ekvidistantna mreža:

$$h = \frac{b-a}{n-1}, \quad x_i = a + (i-1)h, \quad i = 1, 2, \ldots, n.$$

Dakle, za zadane $x_1, \ldots, x_n$, tražimo težine $w_1, \ldots, w_n$, takve da aproksimacijska formula I_n

$$I_n(f) = w_1 f(x_1) + w_2 f(x_2) + \cdots + w_n f(x_n)$$

integrira egzaktno na prostoru polinoma $\mathcal{P}_m$ što veće dimenzije. Budući da je potrebno odrediti n vrijednosti, potrebno nam je n linearno nezavisnih

jednadžbi. Stoga, $m = n - 1$. Postavljamo sustav jednadžbi:

$$\begin{cases} I(1) = \int_a^b 1\,dx \\ I(x) = \int_a^b x\,dx \\ \vdots \\ I(x^{n-1}) = \int_a^b x^{n-1}\,dx \end{cases} \tag{1}$$

Pokazuje se da su, za međusobno različite $x_1, \ldots, x_n$, jednadžbe (1) linearno nezavisne, odnosno, da je dobiveni linearni sustav regularan. Ovisno o izboru broja čvorova integracije, kao i izboru samih čvorova, dobivamo razne formule za integraciju, pri čemu su najjednostavnije i najpopularnije trapezna i Simpsonova, bilo u običnoj, bilo u tzv. produljenoj varijanti[3]. No, budući da algoritam koji ćemo predstaviti nije baziran na Newton–Cotesovim formulama, nećemo ih opisivati u više detalja.

3.3 Gaussove formule

Cilj Gaussovih formula, odnosno, Gaussove integracije je odrediti $2n$ nepoznatih parametara: n čvorova integracije i n težinskih koeficijenata. Lako proizlazi da, u ovom slučaju, možemo egzaktno integrirati polinome do stupnja $2n - 1$, uključivo. Budući da je stupanj polinoma na kojima se traži egzaktnost veći, ove su formule, generalno, preciznije. No, koeficijente je, dakako, teže odrediti. Naime, sistem jednadžbi koji dobivamo je, u ovom slučaju, nelinearan. Stoga se koeficijenti, općenito, ne računaju iz tog uvjeta, jer je to presložen problem.

[3]U produljenoj varijanti se interval integracije "cijepa" na više manjih intervala, a onda se na svakome od tih intervala primjenjuje odabrana formula.

Kod Gaussovih formula se integracijska formula zapisuje pomoću tzv. težinske funkcije $w \geq 0$ u obliku

$$I(f) = \int_a^b w(x) f(x)\, dx.$$

Ideja je "razdvojiti" podintegralnu funkciju na dva dijela, tako da eventualni singulariteti budu uključeni u w. Svih se $2n$ koeficijenata lako odredi pomoću ortogonalnih polinoma (uz standardni skalarni produkt, obzirom na w) na intervalu $[a, b]$. Naravno, svaku bazu za $\mathcal{P}_n$, pa tako i kanonsku, možemo ortogonalizirati, na primjer, Gram–Schmidtovim postupkom. Na taj način dobivamo ortogonalnu bazu $\{p_1, p_2, \ldots, p_n\}$. Može se pokazati (v. [7]) da su čvorovi integracije $x_1, x_2, \ldots, x_n$ nultočke polinoma p_n. Te nultočke su realne, jednostruke i leže u otvorenom intervalu $\langle a, b\rangle$. Za pripadne težine vrijedi:

$$w_i = \int_a^b w(x)\, \frac{p_n(x)}{(x - x_i)p_n'(x_i)}\, dx, \quad i = 1, 2, \ldots, n.$$

Za kraj, napomenimo samo da za Gaussove formule nema smisla promatrati produljenu varijantu, budući da se čvorovi integracije "biraju" na cijelom segmentu $[a, b]$ tako da formula najbolje aproksimira integral.

Što se tiče primjene gornjih formula za računanje čvorova i težina na problem koji rješavamo u ovom radu, napominjemo da se problem nalaženja Gaussovih integracijskih formula za sažimajuću bazu *ne može* riješiti na sličan način. Naime, ta baza nije polinomna. Zato se traže rješenja sustava jednadžbi (analognog sustavu (1) sa str. 7), nakon pojednostavljenja problema – kad iskoristimo da polinomi ipak čine najveći dio sažimajuće baze.

4 Aproksimacija funkcija

Već smo ranije spomenuli da će metode aproksimacije funkcija igrati veliku ulogu u razvijanju algoritma za problem koji smo postavili, počevši od

same motivacije za algoritam, pa do metoda računanja greške pri integraciji.

Važnost aproksimacije funkcija u stvarnim primjenama je jasna: obzirom na njihovu složenost, s nekim funkcijama u praksi lakše računamo, a s nekima teže. Stoga se često javlja potreba za "pojednostavljivanjem" polazne funkcije. Upravo to i jest ideja aproksimacije, na primjer, na prostoru polinoma određenog stupnja – širu klasu funkcija ćemo moći zadovoljavajuće točno aproksimirati polinomima. Generalno, cilj je naći takvu aproksimacijsku funkciju koja zadovoljava naše uvjete. Dakle, znamo ju integrirati i/ili derivirati i efikasno (brzo) izračunavati funkcijsku vrijednost za dani argument. Svakako, pritom napravljena greška mora biti što manja.

U nastavku promataramo neke od uobičajenih metoda za aproksimaciju te ističemo neke nešto manje poznate metode.

4.1 Taylorov polinom

Za funkciju f i točku x_0 iz domene od f, takvu da postoji derivacija svakog reda od f u x_0, definiramo Taylorov red $T(f, x_0)$ s:

$$T(f, x_0)(x) = \sum_{k=0}^{\infty} \frac{f^{(k)}(x_0)}{k!}(x - x_0)^k.$$

Osim evidentnog problema konvergencije tog reda (idealno, želimo da taj red konvergira upravo prema f, što stoji za široku klasu funkcija, ali ne i za sve funkcije), jasno je da direktna primjena ove definicije nije ni najmanje operativna u numeričkom smislu. Naime, radi se o "beskonačnoj sumi". Očito, najlakši način da se tome doskoči je da, umjesto reda, gledamo samo početni komad sume. Na taj način dobivamo Taylorov polinom za f u x_0:

$$T_n(f, x_0)(x) = \sum_{k=0}^{n} \frac{f^{(k)}(x_0)}{k!}(x - x_0)^k.$$

U ovom slučaju se i dalje zahtijeva postojanje do uključivo n-te derivacije funkcije f u točki x_0, što ograničava primjenu. Također, valja spomenuti da je ovakva aproksimacija funkcije, u većini slučajeva, dobra samo u nekoj okolini točke x_0.

Za trigonometrijske funkcije, uz $x_0 = 0$, direktno dobivamo

$$\sin(x) = \sum_{n=0}^{\infty}(-1)^n \frac{x^{2n+1}}{(2n+1)!}, \quad \cos(x) = \sum_{n=0}^{\infty}(-1)^n \frac{x^{2n}}{(2n)!},$$

pri čemu gornje tvrdnje vrijede, štoviše, na cijelom $\mathbb{R}$. Jednostavnom zamjenom varijabli $x \to px$ dobivamo Taylorove redove trigonometrijskih funkcija sažimajuće baze

$$\sin(px) = \sum_{n=0}^{\infty}(-1)^n p^{2n+1} \frac{x^{2n+1}}{(2n+1)!}$$

i, analogno,

$$\cos(px) = \sum_{n=0}^{\infty}(-1)^n p^{2n} \frac{x^{2n}}{(2n)!}.$$

Što se tiče osnovnog problema ovoga rada, uviđamo da polinomni dio sažimajuće baze ne mora činiti vektorski prostor velike dimenzije, jer bi u tom slučaju sami polinomi dobro aproksimirali ponašanje funkcija sinus i kosinus. Uvođenje funkcija sinus i kosinus ima smisla samo u problemima u kojima se trigonometrijske funkcije prirodno pojavljuju kao bitni dio rješenja, a ostatak "pojave" aproksimiramo polinomima relativno niskog stupnja. Dakle, u praktičnim je primjenama n mali.

4.2 Legendreovi polinomi

Nažalost, u algoritmu kojeg želimo konstruirati, zbog neortogonalnosti članova Taylorovog reda, odnosno, standardne baze potencija $\{1, x, x^2, \ldots\}$, aproksimacija funkcije Taylorovim polinomom ne pruža numerički zadovoljavajuće rezultate. Taj problem rješavamo prijelazom na ortogonalnu bazu

Legendreovih polinoma. Legendreovi polinomi su rješenja diferencijalne jednadžbe (v. [12])

$$\frac{d}{dx}\left[(1-x^2)\frac{d}{dx}P_n(x)\right] + n(n+1)P_n(x) = 0,$$

gdje je $n \in \mathbb{N}_0$, a s P_n označavamo n-ti "normalizirani"[4] Legendreov polinom.

Najvažnije svojstvo Legendreovih polinoma je da su međusobno ortogonalni na segmentu $[-1, 1]$ s težinskom funkcijom $w(x) = 1$,

$$\int_{-1}^{1} P_m(x)P_n(x)\,dx = 0, \quad \text{za } m \neq n,$$

što daje motivaciju za njihovo uvođenje. Što se tiče praktičnosti korištenja, i tu su prihvatljivi. Naime, Legendreove polinome možemo računati iz tročlane rekurzivne relacije

$$(n+1)P_{n+1}(x) - (2n+1)xP_n(x) + nP_{n-1}(x) = 0, \quad n \in \mathbb{N},$$

uz prva dva polinoma

$$P_0(x) = 1, \quad P_1(x) = x,$$

što, očito, znatno olakšava primjenu.

4.2.1 Razvoj funkcije kosinus po Legendreovim polinomima

Kao što smo spomenuli, pri računanju greške zanimat će nas koliko dobro, obzirom na neke čvorove i težine, Gaussova formula integrira funkciju kosinus. Zbog navedenih problema s Taylorovim razvojem funkcija, najbolja opcija u svojoj osnovi sadrži računanje razvoja funkcije po Legendreovim polinomima. No, taj je problem računski nešto teži od računanja koeficijenata u Taylorovim polinomima.

[4]Takav da vrijedi $P_n(1) = 1$, $\forall n \in \mathbb{N}_0$.

Općenito govoreći, razvoj funkcije f po Legendreovim polinomima P_k svodi se na određivanje koeficijenata d_k u sumi

$$f = \sum_{k=0}^{\infty} d_k P_k.$$

Postupak je sljedeći:

1. razvijemo funkciju u Taylorov red oko nule;
2. iskoristimo poznate razvoje potencija x^l po Legendreovim polinomima (v. [12, Problem 15.9] i [13, Problem 2.2.25]) i uvrstimo ih u Taylorov razvoj;
3. u dobivenoj dvostrukoj sumi zamijenimo poredak sumacije, tako da dobijemo razvoj funkcije po Legendreovim polinomima;
4. koeficijenti d_k su tada sume koje numerički računamo.

Zbog parnosti kosinusa, Legendreov razvoj sadrži samo parne polinome P_{2k}. Za dani parametar p, shodno tome, vrijedi

$$\cos(px) = \sum_{k=0}^{\infty} d_{2k}(p) P_{2k}(x).$$

Koeficijente $d_{2k}(p)$ određujemo preko relacije

$$d_{2k}(p) = f_k(p) \sum_{j=0}^{\infty} t_{k,j}(p),$$

gdje su $f_k(p)$ i $t_{k,j}(p)$ zadani rekurzivno s

$$f_0(p) = 1, \quad f_k(p) = -\frac{p^2}{(4k-1)\,(4k-3)} \cdot f_{k-1}(p), \quad k \in \mathbb{N},$$

odnosno,

$$t_{k,0}(p) = 1, \quad t_{k,j}(p) = -\frac{p^2}{2j\,(2j+4k+1)} \cdot t_{k,j-1}(p), \quad j \in \mathbb{N}.$$

5 Nelinearne jednadžbe

Osnovni dio algoritma kojeg predstavljamo bavi se minimiziranjem greške pri integraciji. Funkcija greške, koja kao argumente ima čvorove integracije i njihove težine, očito, nije linearna. Ipak, želimo biti sposobni na neki način odrediti njezinu nultočku. Metoda koju ćemo koristiti nije specifična za funkciju s kojom radimo, nego je primjenjiva na traženje nultočke široke klase funkcija.

Općenito gledajući, rješavanje nelinearnih jednadžbi predstavlja vrlo složen problem. U nekim je situacijama moguće raznim supstitucijama i transformacijama dobiti linearni sustav iz kojeg lako očitamo rješenja početnog nelinearnog sustava. Ipak, to nije moguće uvijek, pa je potrebno pronaći algoritme koji će rješavati takvu vrstu problema na puno većoj klasi funkcija.

Da formalno definiramo problem: za danu funkciju $f : I \to \mathbb{R}$, gdje je $I \subseteq \mathbb{R}$ neki interval, htjeli bismo pronaći $x \in I$ takav da je $f(x) = 0$[5].

Funkcija može imati i više nultočaka na danom intervalu. U tom slučaju moramo prvo, analizirajući tok funkcije, izolirati neku nultočku, odnosno, odrediti podsegment od I na kojem imamo samo jednu nultočku. Pretpostavljamo da su sve nultočke izolirane, odnosno, da za svaku možemo naći takav segment. Naravno, ta pretpostavka, iako jest u praksi vrlo prirodna, ne stoji za proizvoljnu neprekidnu funkciju f.

Nakon toga, preostaje odrediti samu nultočku. Dakako, u slučaju da funkcija f nije "jako posebna" (npr. polinom stupnja manjeg od 5), jedini način da se to učini jest numerički.

Postoje razni algoritmi za izrečeni problem. Spomenimo samo neke: me-

[5]Nužno zahtijevamo da je f neprekidna, jer bez tog ili sličnog uvjeta, nultočke, očito, nije moguće odrediti.

toda bisekcije, metoda pogrešnog položaja, metoda sekante i metoda tangente. Oni se, uz razlike u uvjetima na funkciju f, razlikuju po brzini konvergencije i po tome hoće li uopće uvijek konvergirati ili ne.

U algoritmu koristimo metodu bisekcije. Radi se o jednoj od najjednostavnijih metoda, koja koristi samo vrijednosti funkcije f, a ne i vrijednosti njezinih derivacija.

5.1 Metoda bisekcije

Metoda bisekcije je najprirodnija metoda određivanja nultočke funkcije i predstavlja svojevrstan numerički ekvivalent binarnog traženja. Metoda bisekcije zahtijeva da se nultočka izolira na segmentu $[a, b]$ tako da vrijedi $f(a)f(b) < 0$. Taj uvjet, garantira postojanje barem jedne nultočke na tom segmentu.

Algoritam u svakom koraku dijeli segment na upola manji, te se na taj način približava nultočki funkcije. Sam algoritam dan je u pseudo-kôdu.

Algoritam 5.1: Metoda bisekcije

$x = a$, $y = b$, $z = (x + y)/2$;
while $|f(z)| > \varepsilon$ **do**
    if $f(x) \cdot f(z) < 0$ **then** $y = z$;
    else $x = z$;
    $z = (x + y)/2$;
end while
return z;

Algoritam sigurno nalazi nultočku ako vrijede početni uvjeti, no ne nalazi nultočke u čijoj okolini nema promjene predznaka, na primjer, nultočke parnog reda. U kontekstu ovog rada, taj nedostatak neće smetati, jer na

zadanom intervalu tražimo jednostruku nultočku, što je posljedica jedinstvenosti težina i čvorova integracije. Naravno, treba napomenuti da metoda, kao i sve u numeričkoj matematici, nije "imuna" na neke patološke slučajeve u kojima, ovisno o preciznosti u kojoj radimo, može dati nezadovoljavajuće rezultate. No, ista napomena, u načelu, vrijedi i za ostale algoritme traženja nultočaka.

6 Numeričke greške

Nažalost, jedan od osnovnih problema računanja na računalu je da računala inherentno rade u konačnoj preciznosti. Mnogi programski jezici, kao što je C/C++, podržavaju baš tu (sklopovljem računala fiksiranu) jednostruku ili dvostruku preciznost. Viša preciznost (ali još uvijek konačna) može se postići softverski. Recimo, jedna takva biblioteka programa je i GNU Multiple Precision Arithmetic Library.

Uslijed toga, račun na računalu nije (uvijek) posve precizan. Naravno, moderna se računala trude minimizirati greške, ali one u svakom slučaju postoje. Najveći problem, predstavlja situacija u kojoj se greška poveća toliko da rezultati računa postanu besmisleni. Primjer za to je tzv. katastrofalno kraćenje, kad oduzimanjem dva broja sličnog reda veličine, kao rezultat dobijemo znatno manji broj.

Algoritam koji ćemo izložiti ima nekoliko ekvivalentnih matematičkih iskaza, međutim, oni ne moraju biti i numerički jednaki, što će se posebno očitovati kod računanja greške integracije. No, budući da su neke od metoda koje algoritam koristi manje, a neke više numerički stabilne, morat ćemo oprezno odabrati najbolju. Na taj ćemo način osigurati da greške računala ne utječu na korektnost predstavljenog algoritma.

7 QR algoritam

Osnovnu ulogu u metodi koju predstavljamo igra QR algoritam za nalaženje svojstvenih vrijednosti i vektora simetrične matrice[6]. Potrebu za takvim[7] algoritmom komentirali smo i ranije – njime se za polinomne Gaussove formule dobivaju odgovarajući čvorovi i težine, što onda proširujemo na integraciju funkcija sažimajuće baze. Općenito, algoritam se bazira na činjenici da za svaku matricu postoji rastav na produkt ortogonalne i gornjetrokutaste, odnosno donjetrokutaste matrice, tzv. QR (QL) dekompozicija.

Standardni QR algoritam počinje trodijagonalizacijom matrice (nizom Householderovih reflektora ili Givensovih rotacija)[8], a zatim se formira niz matrica koji konvergira prema dijagonalnoj matrici [6, 15]. Formalnije zapisano, jedna iteracija osnovnog QR algoritma izgleda ovako: A_k zapisujemo kao

$$A_k = Q_k R_k,$$

gdje je $Q_k = Q_k^*$, a R_k gornjetrokutasta[9]. Sljedeća iteracija je definirana s $A_{k+1} = R_k Q_k$. Može se pokazati[10] da ovako zadan niz međusobno sličnih matrica konvergira prema dijagonalnoj. Naravno, ta dijagonalna matrica

[6] On u naš kôd ulazi implicitno, preko MATLAB-ove funkcije *eig*, što osigurava da je izveden optimalno.

[7] Ponovno, mogli smo i raditi s, na primjer, Jacobijevim algoritmom, ali je QR algoritam u prednosti zbog trodijagonalnosti matrica s kojima radimo.

[8] Nama taj dio algoritma neće biti potreban budući da već "ulazimo" s trodijagonalnom matricom. Budući da "većina" greške QR algoritma proizlazi iz trodijagonalizacije, radi se o velikoj prednosti.

[9] U slučaju QL verzije algoritma, radi se, naravno, QL dekompozicija matrice, umjesto QR dekompozicije.

[10] Dokaz se može naći u [15], Teorem 8.6.1., str. 158.

sadrži na dijagonali svoje svojstvene vrijednosti.

Ta se konvergencija na jednostavan način može i ubrzati – uvođenjem pomaka. Jedna iteracija QR algoritma s pomakom $\sigma \in \mathbb{R}$ je kako slijedi:

$$A_k - \sigma I = Q_k R_k$$
$$A_{k+1} = R_k Q_k + \sigma I.$$

Pomak se najčešće ne odabire neovisno o elementima matrice, čemu ćemo posvetiti više riječi kasnije. Primijetimo sada kako zbog ortogonalnosti matrice Q možemo pisati $R_k = Q_k^T(A_k - \sigma I)$, što dalje povlači

$$A_{k+1} = Q_k^T(A_k - \sigma I)Q_k + \sigma I = Q_k^T A_k Q_k,$$

tj. matrice A_{k+1} i A_k su slične za svaki k.

Prirodno je pitanje, nakon što smo riješili problem određivanja svojstvenih vrijednosti, može li ovaj algoritam dati i svojstvene vektore.

Nije teško pokazati da i u ovom slučaju QR producira zadovoljavajuće rezultate: i-ti svojstveni vektor[11] približno je jednak $a_i = Q_1 \cdots Q_k e_i$, gdje je k broj iteracija provedenog QR algoritma uz unaprijed zadanu točnost.

7.1 Korištenje u praksi

Naoko, nedostatak ove metode jest veliki broj QR faktorizacija, pri čemu je složenost svake $\mathcal{O}(n^3)$. No, u praksi se matrice $Q_1, \ldots, Q_k$ nikada ne računaju eksplicitno. Jedna iteracija QR algoritma može se ostvariti pomoću niza Givensovih rotacija.

Naime, može se pokazati[12] da, ukoliko je A realna simetrična matrica, $Q =$

[11] Svojstveni vektor koji odgovara i-toj svojstvenoj vrijednosti, tj. i-tom elementu konačne dijagonalne matrice.

[12] Tvrdnja slijedi direktno iz Implicitnog Q teorema koji se s dokazom može naći u [5], Teorem 9.7.3., str. 264.

$(q_1, \ldots, q_n)$ ortogonalna i $T = Q^T AQ$ nereducirana trodijagonalna simetrična matrica, tada su Q i T do na predznake stupaca određeni prvim stupcem matrice Q.

Ukratko ćemo opisati jednu iteraciju algoritma. Neka je G_1 zadana tako da vrijedi $G_1^T a_1 = \pm\|a_1\|_2 e_1$, pri čemu je a_1 prvi stupac matrice $A - \sigma I$.

Transformacija sličnosti $G_1^T A G_1$ pokvarit će trodijagonalnost: elementi na pozicijama $(1,3)$ i $(3,1)$ bit će različiti od nule. Njih onda poništavamo Givensovom matricom G_2, što, radi izbočine na pozicijama $(2,4)$ i $(4,2)$. Nastavljamo dalje te dolazimo do matrice G_{n-1} koja će poništiti i posljednje izbočine na mjestima $(n-2, n)$ i $(n, n-2)$. Ovaj postupak poznat je pod imenom "chasing the bulge". Po gore navedenom teoremu lako slijedi da su $\widetilde{Q} = G_1 G_2 \cdots G_{n-1}$ i Q iz "uobičajenog QR algoritma" jednake do na predznake stupaca.

7.2 Odabir pomaka

Prije smo komentirali kako se konvergencija može ubrzati "pomicanjem" matrice. Postoji više metoda za dobivanje pomaka, no u praksi je najčešći Wilkinsonov pomak [3, 17]. Za matricu

$$A = \begin{pmatrix} \alpha_1 & \beta_1 & & & & & \\ \beta_1 & \alpha_2 & \beta_2 & & & & \\ & \beta_2 & \alpha_3 & \beta_3 & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & \ddots & \ddots & \ddots & \\ & & & & \beta_{n-2} & \alpha_{n-1} & \beta_{n-1} \\ & & & & & \beta_{n-1} & \alpha_n, \end{pmatrix}$$

Wilkinsonov pomak ω dobivamo kao svojstvenu vrijednost podmatrice

$$\begin{pmatrix} \alpha_{n-1} & \beta_{n-1} \\ \beta_{n-1} & \alpha_n \end{pmatrix}$$

bližu α_n[13].

Eksplicitna, numerički stabilna formula za računanje pomaka je

$$\omega = a_1 - \text{sign}(d) - \frac{\beta_{n-1}^2}{|d| + \sqrt{d^2 + \beta_{n-1}^2}},$$

pri čemu je d jednak $(\alpha_{n-1} - \alpha_n)/2$.

7.3 Složenost

Ukoliko nije potrebno računati svojstvene vektore, složenost QR algoritma je $\mathcal{O}(n^2)$, a u suprotnom iznosi $\mathcal{O}(n^3)$. Algoritam se može, doduše, ponešto ubrzati tako da se prvo provede QR algoritam bez računanja svojstvenih vektora, te nakon što se dobiju svojstvene vrijednosti, algoritam provedemo još jednom, ali ovaj se puta za pomake uzimaju izračunate svojstvene vrijednosti, što osigurava da algoritam završi u točno n koraka, jer se u svakom od njih nađe točno jedna svojstvena vrijednost.

Dugogodišnje iskustvo pokazuje da s izborom Wilkinsonovog pomaka, QR algoritam treba približno 1.6 iteracija po svakoj svojstvenoj vrijednosti, u standardnoj točnosti računanja.

Ipak, kako smo komentirali ranije, u stvarnim je primjenama našeg algoritma n općenito vrlo mali. Stoga, brzina QR algoritma nam je samo od teoretskog interesa.

[13]U slučaju $\alpha_n = \alpha_{n-1}$ uzimamo manju svojstvenu vrijednost.

8 Algoritam – implementacija

Nakon što smo opisali sve potrebne numeričke metode i probleme, predstavljamo i kôd algoritma (zapravo skup funkcija koje treba pozvati), pisan u MATLAB/GNU Octave-u. Na početku, predstavljamo glavni program. Primjećujemo da se, nakon inicijaliziranja svih početnih koeficijenata, ovaj dio kôda svodi na metodu bisekcije (vidjeti Algoritam 5.1). No, time još nismo riješili osnovni problem algoritma – računanje greške. Funkciju *greska* predstavljamo nakon glavnog programa.

```
function [W, X] = algoritam(n, p)
    b0 = 2;
    J = zeros(n, n);
    for i = 1 : n − 2
      J(i, i + 1) = i / sqrt(4 ∗ i ∗ i − 1);
      J(i + 1, i) = J(i, i + 1);
    end
    lijevo = −1;
    desno = 1;
    greskaLijevo = greska(J, n, p, lijevo);
    greskaDesno = greska(J, n, p, desno);
    while (1)
      sredina = (lijevo + desno) / 2;
      greskaSredina = greska(J, n, p, sredina);
      if (abs(greskaSredina) < 1e−15)
        break;
      end
      if (greskaSredina * greskaLijevo < 0)
        desno = sredina;
        greskaDesno = greskaSredina;
```

```
    else
      lijevo = sredina;
      greskaLijevo = greskaSredina;
    end
  end
  J(n − 1, n) = sredina;
  J(n, n − 1) = sredina;
  [E,X] = eig(J);
  X = diag(X);
  W = zeros(n, 1);
  for i = 1 : n
    W(i) = b0 ∗ E(1, i) ∗ E(1, i);
  end
```

8.1 Računanje greške integracijske formule

Kada bi račun bio proveden u egzaktnoj aritmetici, pogrešku integracije ne bi bilo teško izračunati. No, ograničena preciznost zahtijeva numerički stabilnu metodu. Zbog "skoro–kolinearnosti" kosinusa s kanonskim polinomima na ovom intervalu, funkcija kosinus razvija se po Legendreovim polinoma. Preciznije, uzima se funkcija c_p koja je ortogonalna na polinomni dio baze, dakle, ostatak Legendreovog razvoja funkcije $\cos(px)$, bez početnog dijela koji je već u prostoru polinoma do stupnja $2n-3$, odnosno,

$$c_p = \sum_{k=n-1}^{\infty} d_{2k}(p) P_{2k}.$$

Kako koeficijenti $d_{2k}(p)$ vrlo brzo padaju za malo veće n, da bi se postigla visoka relativna točnost izračunate greške, potrebno je "skalirati" funkciju c_p. Skaliranje se vrši obzirom na sumu apsolutnih vrijednosti koeficijenata u tom razvoju, budući da takvo skaliranje ne stvara problem u smislu

složenosti. Naime, te koeficijente ionako treba računati dok ne padnu ispod željene točnosti. Pravi izgled funkcije c_p je, zapravo,

$$c_p = \frac{1}{scale} \sum_{k=n-1}^{\infty} d_{2k}(p) P_{2k},$$

gdje je

$$scale = \sum_{k=n-1}^{\infty} |d_{2k}(p)|.$$

Slijedi algoritam za računanje greške. Za računanje funkcije c_p poziva se funkcija *koeficijenti* koja vraća polje D (i veličinu – *ind*) koeficijenata $d_{2k}(p)$ kao i sumu njihovih modula – *scale*.

```
function [t] = greska(J, n, p, x)
    b0 = 2;
    J(n, n − 1) = x;
    J(n − 1, n) = x;
    [W, X] = eig(J);
    X = diag(X);
    t = 0;
    [D, scale, ind] = koeficijenti(n, p);
    for j = 1 : n
      if abs(X(j)) > 1
        t = 5000;
        return
      end
      s = 0;
      for i = 1 : ind
        u = legendre(2 ∗ n − 4 + 2 ∗ i, X(j));
        u = u(1);
        s = s + D(i) ∗ u;
      end
```

```
    s = s * b0 * W(1, j) * W(1, j);
    t = t + s;
  end
  t = t / scale;
  end
```

Izuzev u MATLAB-u implementirane funkcije *legendre*, implementirali smo i funkciju *koeficijenti*. Nakon potrebne inicijalizacije, ta funkcija u potpunosti[14] prati prethodno opisan razvoj kosinusa za određivanje potrebnih koeficijenta, pri čemu se izrazi f_k spremaju u *factor*, izrazi $t_{k,j}$ u *term*, a njihova suma u *sum*.

```
function [D, scale, ind] = koeficijenti(n, p)
  D = zeros(100, 1);
  ind = 1;
  factor = 1;
  scale = 0;
  k = n - 1;
  while 1
    term = 1;
    sum = term;
    k4p1 = 4 * k + 1;
    j2 = 2;
    while abs(term) > eps * abs(sum)
      term = term * (-p / j2) * (p / (k4p1 + j2));
      sum = sum + term;
      j2 = j2 + 2;
    end
    if k >= n
```

[14]S potrebnim translacijama za 1 zbog indeksiranja.

```
      factor = factor * (-p / (k4p1 - 2)) * (p / (k4p1 - 4));
    end
    D(ind) = factor * sum;
    scale = scale + abs(D(ind));
    if abs(D(ind)) <= eps * abs(D(1))
      break;
    end
    k = k + 1;
    ind = ind + 1;
  end
  ind = ind - 1;
end
```

9 Testiranje i rezultati

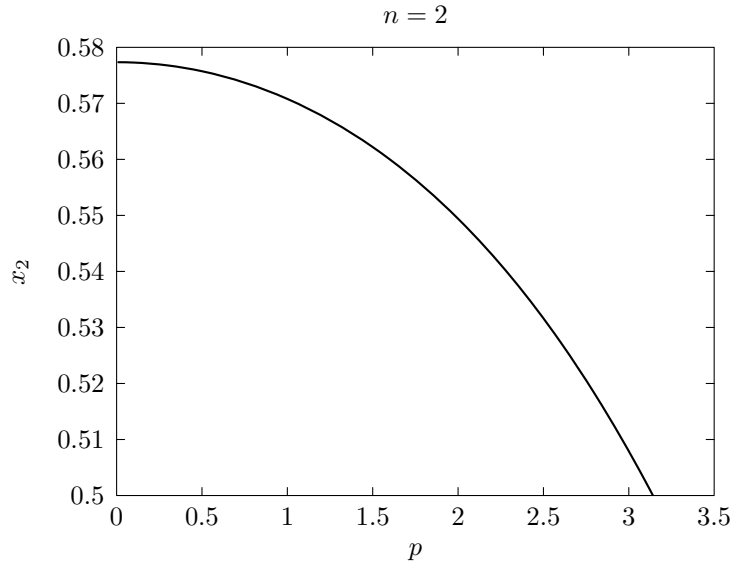
U ovom dijelu ukratko prezentiramo neke od rezultata dobivenih testiranjem gore opisanog i implementiranog algoritma. Primijetimo da su za praktičnu primjenu najkorisniji rezultati testiranja za $n < 10$. Posebno, rezultati testiranja za $n = 2$, $n = 3$ i $n = 5$, za različite parametre p, dani su u Tablicama 1–3. Budući da je kôd programa objavljen unutar ovog rada, dodatna testiranja nije teško provesti. U Tablicama 1–3 dane su vrijednosti čvorova i težina samo za nenegativne čvorove x_i. Zbog simetrije za te čvorove i težine vrijedi

$$x_i = -x_{n-i}, \quad w_i = w_{n-i}, \quad i = 0, 1, \ldots, \left\lfloor \frac{n}{2} \right\rfloor .$$

Slika 1 prikazuje za $n = 2$ graf ovisnosti najvećeg čvora integracije x_2 o parametru p.

Tablica 1: Čvorovi i težine za $n = 2$ i različite vrijednosti p.

p		$i = 1$	p		$i = 1$
0.01	x_i	0.577349627687952	1.60	x_i	0.560003836255995
	w_i	1.000000000000000		w_i	1.000000000000000
0.10	x_i	0.577286105411754	2.00	x_i	0.549409279565755
	w_i	1.000000000000000		w_i	1.000000000000000
0.30	x_i	0.576771804348972	2.60	x_i	0.527386157386079
	w_i	1.000000000000000		w_i	1.000000000000000
0.50	x_i	0.575737903249122	2.80	x_i	0.518168014888067
	w_i	1.000000000000000		w_i	1.000000000000000
1.00	x_i	0.570796326794897	3.10	x_i	0.502381550855563
	w_i	1.000000000000000		w_i	1.000000000000000
1.50	x_i	0.562204655981295	3.14	x_i	0.500092074034565
	w_i	1.000000000000000		w_i	1.000000000000000



Slika 1: Graf ovisnosti najvećeg čvora integracije o parametru p.

Tablica 2: Čvorovi i težine za $n = 3$ i različite vrijednosti p.

p		$i = 1$	$i = 2$
0.01	x_i	0.000000000000000	0.774596521699107
	w_i	0.888888465607905	0.555555767196048
0.10	x_i	0.000000000000000	0.774581913485105
	w_i	0.888846555235741	0.555576722382129
0.30	x_i	0.000000000000000	0.774463756889686
	w_i	0.888507481594047	0.555746259202976
0.50	x_i	0.000000000000000	0.774226852813678
	w_i	0.887827170925321	0.556086414537340
1.00	x_i	0.000000000000000	0.773105769849486
	w_i	0.884599302029966	0.557700348985017
1.50	x_i	0.000000000000000	0.771197736104239
	w_i	0.879073209804319	0.560463395097841
1.60	x_i	0.000000000000000	0.770716733302532
	w_i	0.877673636782421	0.561163181608789
2.00	x_i	0.000000000000000	0.768440613094761
	w_i	0.871015131341432	0.564492434329284
2.60	x_i	0.000000000000000	0.763878150564730
	w_i	0.857488539804404	0.571255730097799
2.80	x_i	0.000000000000000	0.762018034787198
	w_i	0.851903901781520	0.574048049109240
3.10	x_i	0.000000000000000	0.758874183036837
	w_i	0.842371569329267	0.578814215335367
3.14	x_i	0.000000000000000	0.758421077846743
	w_i	0.840987947090215	0.579506026454894

Tablica 3: Čvorovi i težine za $n = 5$ i različite vrijednosti p.

p		$i = 2$	$i = 3$	$i = 4$
0.01	x_i	0.000000000000000	0.538469271485405	0.906179829569015
	w_i	0.568888831425331	0.478628664206978	0.236926920080356
0.10	x_i	0.000000000000000	0.538465447888465	0.906178208919588
	w_i	0.568883142201338	0.478628041289405	0.236930387609925
0.30	x_i	0.000000000000000	0.538434536370678	0.906165108824994
	w_i	0.568837144558161	0.478623009712285	0.236958418008634
0.50	x_i	0.000000000000000	0.538372639749139	0.906138887593951
	w_i	0.568745020318735	0.478612957815702	0.237014532024931
1.00	x_i	0.000000000000000	0.538081184927334	0.906015600321855
	w_i	0.568310882444455	0.478566042757814	0.237278516019960
1.50	x_i	0.000000000000000	0.537590559444250	0.905808736813741
	w_i	0.567578764456294	0.478488620772332	0.237721996999520
1.60	x_i	0.000000000000000	0.537467931722319	0.905757164685021
	w_i	0.567395521301074	0.478469574400712	0.237832664948750
2.00	x_i	0.000000000000000	0.536893266176048	0.905516182517368
	w_i	0.566535424569318	0.478381942461957	0.238350345253384
2.60	x_i	0.000000000000000	0.535768099309348	0.905047660418356
	w_i	0.564844813692083	0.478218125663713	0.239359467490244
2.80	x_i	0.000000000000000	0.535319096333812	0.904861907126787
	w_i	0.564167718035092	0.478155626418279	0.239760514564174
3.10	x_i	0.000000000000000	0.534572312643117	0.904554479552798
	w_i	0.563038451887802	0.478055312208544	0.240425461847556
3.14	x_i	0.000000000000000	0.534465894855176	0.904510824234260
	w_i	0.562877211479664	0.478041387170458	0.240425461847556

Kao što se može vidjeti, rezultati pokazuju da se položaj čvorova integracije i njihove težine ne mijenjaju znatno (ali nisu posve isti, pogotovo za iznimno male n, što daje očekivani smisao ovakvom algoritmu) u ovisnosti o parametru p.

Da bismo provjerili dobivene rezultate i minimizirali greške, tijekom konstruiranja algoritma korištena je sljedeća funkcija koja računa i ispisuje greške pri integraciji funkcija sažimajuće baze. U idealnom slučaju, pri egzaktnom računanju, te bi greške bile 0. No, zbog već diskutiranih, inherentno loših, svojstava aritmetike računala, to je, očito, nemoguće.

```
function provjera(n, p, W, X, sredina)
    fprintf('Sredina: %g\n', sredina);
    fprintf('————\n');
    fprintf('Greska kod:\n');
    for i = 0 : (2 * n - 3)
      gr = 1 / (i + 1) - (-1)^(i +1) / (i + 1);
      for j = 1 : n
        gr = gr - W(j) * X(j)^i;
      end
      fprintf('f(x) = x^%d:\t %g\n', i, abs(gr));
    end
    gr = 0;
    for j = 1 : n
      gr = gr - W(j) * sin(p * X(j));
    end
    fprintf('f(x) = sin(px):\t %g\n', abs(gr));
    gr = 2 * sin(p) / p;
    for j = 1 : n
      gr = gr - W(j) * cos(p * X(j));
```

end

fprintf('f(x) = **cos**(px):\t %g\n\n', **abs**(gr));

end

Dalje za primjer predstavljamo neke ispise dobivene ovom funkcijom. Prvo, algoritam testiramo za $n = 2$,$p = 1$. Ispisan je posljednji koeficijent, dobiven metodom bisekcije, kao i greška integracije na funkcijama sažimajuće baze.

```
Sredina: -0.570796
_____________

Greska kod:
f(x)=x^0:          4.44089e-016
f(x)=x^1:          0
f(x)=sin(px):      0
f(x)=cos(px):      4.44089e-016
```

Slijede rezultati dobiveni provjerom algoritma za $n = 6$, $p = 3.141$.

```
Sredina: -0.497627
_____________

Greska kod:
f(x)=x^0:          2.77556e-017
f(x)=x^1:          1.11022e-016
f(x)=x^2:          9.15934e-016
f(x)=x^3:          1.38778e-016
f(x)=x^4:          6.93889e-016
f(x)=x^5:          1.24900e-016
f(x)=x^6:          5.41234e-016
f(x)=x^7:          6.93889e-017
f(x)=x^8:          4.71845e-016
f(x)=x^9:          2.77556e-017
```

```
f(x)=sin(px):      3.05311e-016
f(x)=cos(px):      1.88738e-015
```

Naposljetku, rezultati dobiveni za $n = 9$, $p = 0.001$, pokazuju da algoritam korektno radi i za $p \sim 0$.

```
Sredina:  -0.500979
```

```
Greska kod:
f(x)=x^0:          1.29063e-015
f(x)=x^1:          1.38778e-017
f(x)=x^2:          3.05311e-016
f(x)=x^3:          2.35922e-016
f(x)=x^4:          3.05311e-016
f(x)=x^5:          1.80411e-016
f(x)=x^6:          2.49800e-016
f(x)=x^7:          9.71445e-017
f(x)=x^8:          2.91434e-016
f(x)=x^9:          9.02056e-017
f(x)=x^10:         2.84495e-016
f(x)=x^11:         9.02056e-017
f(x)=x^12:         2.77556e-016
f(x)=x^13:         4.16334e-017
f(x)=x^14:         2.42861e-016
f(x)=x^15:         4.16334e-017
f(x)=sin(px):      4.06576e-020
f(x)=cos(px):      1.24900e-015
```

Budući da su sve greške reda veličine 10^{-15} ili manje, što je blizu preciznosti MATLAB-a (dvostruke preciznosti), možemo zaključiti da algoritam daje zadovoljavajuće rezultate. Računanjem uz pomoć alata višestruke pre-

ciznosti (na primjer, već prije spomenutim GNU MP-om), ove se greške mogu i proizvoljno smanjiti.

Naravno, ovo testiranje, osim što pokazuje da algoritam korektno radi, nije od velike teoretske važnosti. Za opsežniju diskusiju o svojstvima predstavljenog algoritma, trebalo bi provesti sistematične testove, posebno u nekim graničnim slučajevima.

10 Zaključak

Cilj ovog rada bio je dati, prvi opis algoritma za dobivanje integracijske formule u Gaussovom smislu za funkcije iz

$$\operatorname{span}\{1, x, x^2, \ldots, x^{2n-3}, \sin px, \cos px\},$$

gdje je $p \in \langle 0, \pi \rangle$ na intervalu $[-1, 1]$. To smo i učinili – objasnili smo osnovnu ideju algoritma te u nešto više detalja ušli u eventualne probleme s numeričkom izvedbom algoritma. Uzevši te probleme u obzir, algoritam smo implementirali u programskom jeziku MATLAB/GNU Octave te ga testirali na većem broju vrijednosti parametara. Važno je primijetiti da je konstrukcija gore spomenutog algoritma zahtijevala primjenu brojnih tehnika i algoritama prisutnih u numeričkoj matematici. U prvom redu, to su "obične" Gaussove integracijske formule, tj. formule za egzaktnu integraciju nad prostorom polinoma do stupnja n uključivo. Te su nam formule polazište u pronalaženju formula za funkcije iz naše sažimajuće baze budući da se ta baza sastoji samo od dvije nepolinomne funkcije. Daljnji potrebni postupci bili su metoda bisekcije za pronalaženje nultočaka nelinearnih funkcija, koja je zbog svojih minimalnih zahtjeva na funkciju nad kojom radi bila dobar izbor za numeričko nalaženje posljednjeg $(n-1)$-og koeficijenta. Glavni dio algoritma se završava iteriranjem QR algoritma za nalaženje svojstvenih vrijednosti i

vektora simetrične trodijagonalne matrice. Što se numeričke stabilnosti tiče, najosjetljivija točka bila je računanje greške kod provođenja bisekcije. To smo učinili na nešto kompliciraniji, ali numerički smisleniji način, koristeći Legendreove polinome.

Naposljetku, postavlja se još jedno prirodno pitanje – što dalje? Eventualnog (ali nikako potpunog!) odgovora dotakli smo se na nekoliko mjesta u našem radu. Što se predstavljenog algoritma tiče, ima smisla i dalje ispitivati rezultate koje daje. U svojevrsnim graničnim slučajevima ($p \to 0$, $p \to \pi$) ovi nam rezultati mogu dati i dobru teorijsku podlogu za razumijevanje problema integracije funkcija sažimajuće baze. No, pojavljuje se i još jedno, daleko opsežnije pitanje. U ovom smo radu, naime, polinomnu bazu "nadogradili" trigonometrijskom. Ta se baza prirodno pojavljuje u tumačenju fizikalnih pojava te je stoga posebno zanimljiva. Ipak, prirodno je zapitati se i kojim još funkcijama bismo mogli nadoknaditi polinomnu bazu te koje uvjete na te funkcije moramo zahtijevati da bi algoritam analogan ovome i numerički funkcionirao. Neke od uvjeta spomenuli smo ranije, ali nismo ulazili u detalje.

Obzirom na izrečeno, možemo zaključiti da, iako je u radu predstavljen korektan i originalan algoritam za rješavanje konkretnog problema, sasvim sigurno ima prostora za daljnji nastavak istraživanja općenitog pitanja numeričke integracije funkcija sažimajuće i, općenito, "većinski polinomne" baze, kako s teorijskog, tako i sa stajališta numeričke izvedivosti.

Zahvale

Autori se zahvaljuju prof. dr. sc. Saši Singeru na puno utrošenog vremena i truda, mnogim objašnjenjima i savjetima, te na velikim količinama motivacije. Također, zahvale idu i prof. dr. sc. Sanji Singer na brojnim korekcijama, lektoriranju i tehničkom opremanju rada.

Literatura

[1] I. Aganović and K. Veselić, *Jednadžbe matematičke fizike*, Školska knjiga, Zagreb, 1985.

[2] T. Bosner and M. Rogina, *Numerically stable algorithm for cycloidal splines*, Ann. Univ. Ferrara, 53 (2007), pp. 189–197.

[3] H. Bowdler, R. S. Martin, C. Reinsch, and J. H. Wilkinson, *The QR and QL algorithms for symmetric matrices*, Numer. Math., 11 (1968), pp. 293–306.

[4] D. Braess, *Nonlinear Approximation Theory*, Springer–Verlag, Berlin, 1986.

[5] G. Dahlquist and Å. Björck, *Numerical Methods in Scientific Computing. Vol. 1*, SIAM, Philadelphia, 2008.

[6] J. W. Demmel, *Applied Numerical Linear Algebra*, SIAM, Philadelphia, 1997.

[7] W. Gautschi, *Orthogonal Polynomials: Computation and Approximation*, Numerical Mathematics and Scientific Computation, Oxford University Press, Oxford, 2004.

[8] G. H. Golub and J. H. Welsch, *Calculation of Gauss quadrature rules*, Math. Comput., 23 (1969), pp. 221–230, s1–s10.

[9] S. Karlin and W. J. Studden, *Tchebycheff Systems: With Applications in Analysis and Statistics*, vol. 15 of Pure and Applied Mathematics, Intersecience Publishers, a division of John Wiley & Sons, New York, 1966.

[10] D. P. Laurie, *Accurate recovery of recursion coefficients from Gaussian quadrature formulas*, J. Comput. Appl. Math., 112 (1999), pp. 165–180.

[11] ——, *Computation of Gauss–type quadrature formulas*, J. Comput. Appl. Math., 127 (2001), pp. 201–217.

[12] D. P. Mitrinović, *Uvod u specijalne funkcije*, Građevinska knjiga, Beograd, 1972.

[13] D. P. Mitrinović, D. Đ. Tošić, and R. R. Janić, *Specijalne funkcije, Zbornik zadataka i problema*, Građevinska knjiga, Beograd, 1972.

[14] G. Nürnberger, *Approximation by Spline Functions*, Springer–Verlag, Berlin, 1989.

[15] B. N. Parlett, *The Symmetric Eigenvalue Problem*, no. 20 in Classics in Applied Mathematics, SIAM, Philadelphia, 1998.

[16] G. Wang, Q. Chen, and M. Zhou, *NUAT B-spline curves*, Comput. Aided Geom. Design, 21 (2004), pp. 193–205.

[17] J. H. Wilkinson, *The Algebraic Eigenvalue Problem*, Monographs on Numerical Analysis, Oxford University Press, Oxford, 1965. (Reprinted with corrections, 1977.).

Sažetak

Mario Berljafa, Sara Muhvić, Melkior Ornik:

Računanje Gaussovih integracijskih formula za sažimajuću bazu

U radu se diskutira problem određivanja čvorova i težina koji, u Gaussovom smislu, točno integriraju funkcije iz vektorskog prostora generiranog tzv. sažimajućom bazom.

Na početku rada daje se kratka motivacija za rješavanje ovog problema. Također, predstavlja se osnovna ideja algoritma koji ga rješava. Kao što se pokazuje, algoritam ovisi o poznavanju i kvalitetnoj implementaciji nekoliko koncepata numeričke matematike, koje opisujemo u nastavku rada.

Drugi dio rada, stoga, započinjemo diskutiranjem šireg područja o kojem ovaj rad govori: numeričke integracije. Nakon toga, govorimo o aproksimaciji funkcija, pri čemu se od posebne važnosti za algoritam pokazuje razvoj funkcije kosinus po Legendreovim polinomima. Također, komentiramo još dva prisutna problema: rješavanje nelinearnih jednadžbi, s posebnim osvrtom na metodu bisekcije, kao i numeričke greške. Drugi dio završavamo opisivanjem QR algoritma, koji koristimo u rješavanju problema određivanja svojstvenih vrijednosti i vektora.

U završnom dijelu rada predstavljamo kôd, napisan u programskom jeziku MATLAB/GNU Octave, kojim implementiramo spomenuti algoritam. Testiramo algoritam te ukratko komentiramo dobivene rezultate.

Ključne riječi: *Gaussove integracijske formule, sažimajuća baza, numerička integracija, algoritam*

Summary

Mario Berljafa, Sara Muhvić, Melkior Ornik:

Computation of Gaussian Quadrature Formulae for Compression Basis

In the paper, we discuss the problem of determining nodes and weights that, in the Gaussian sense, exactly integrate functions in the vector space generated by the compression basis.

At the beginning of the paper, short motivation for solving the problem is given. Also presented is the basic idea for the algorithm which solves it. As it is shown, the algorithm depends on knowledge and the quality of implementation of several numerical concepts which we describe later.

The second part of the paper, thus, starts with a discussion of a wider area of numerical mathematics this paper belongs to: numerical integration. Further on, we discuss approximations of functions. With regard to this algoritm, representation of cosine in Legendre polynomials is shown to have special importance. We also discuss two other problems present in the algorithm: solving nonlinear equations, especially with regard to the bisection method, as well as numerical errors. We finish the second part with a description of QR algorithm which we use in solving eigenvalue and eigenvector problems.

In the final part of the paper we present the code (written in MATLAB/GNU Octave) which we use to implement the aforementioned algorithm. We test the algorithm and briefly comment on the obtained results.

Keywords: *Gaussian quadrature formulae, compression basis, numerical integration, algorithm*