Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Osnove numerične matematike

Bojan Orel

Ljubljana, 27. februar 2020

Predgovor

Ta knjiga je nastala na osnovi predavanj iz predmetov *Numerična matematika* v drugem letniku študija *Računalništvo in informatika* in *Numerične metode* v drugem letniku študija *Elektrotehnika* na Univerzi v Ljubljani. Pri pisanju knjige sem imel v mislih bralca, ki se je že srečal z matematiko, kot se predava v prvih letnikih študija tehniških ali naravoslovnih programov na univerzi, predvsem z linearno algebro, odvodi, integrali in diferencialnimi enačbami, pa tudi z osnovami računalništva, saj si uporabo numeričnih metod le težko zamislimo brez računalnika.

Vsebina knjige, kot je razvidno iz kazala, zajema osnovna poglavja numerične matematike. Reševanje sistemov linearnih enačb je gotovo osrednja tema numeričnega računanja, ki se ji učbenik te vrste nikakor ne more izogniti. Pri izboru ostalih poglavij, kot tudi pri izboru posameznih metod, sem poskušal izbrati tiste teme, ki bodo bralcu omogočile osnovno razumevanje tehnik, ki jih uporabljamo pri numeričnem računanju. Posebna pozornost je posvečena obravnavi in napovedovanju napak.

Ker je čas pri predavanjih omejen, je omejen tudi izbor tematike. Seznam poglavij, ki jih v tej knjigi ni, bi bil precej daljši od sedanjega kazala. Verjetno bi vsak, ki se je v praksi srečal s kakšnim konkretnim problemom, ki ga je bilo potrebno rešiti numerično, lahko kaj dodal na ta seznam. Omenimo le nekaj takih naslovov poglavij: optimizacija, Fourierova transformacija, parcialne diferencialne enačbe, itd. Tudi pri poglavjih, ki v tej knjigi so, marsikaj manjka. Kdor se bo lotil reševanja resnejših problemov, verjetno v tej knjigi ne bo našel primernega odgovora na svoja vprašanja in se bo moral posvetovati s kakšno izmed bolj poglobljenih knjig.

Ker je knjiga namenjena predvsem študentom tehnike, ki se bodo pri pri svojem bodočem delu večkrat srečevali z numeričnim reševanjem problemov, je v knjigi poudarjen predvsem 'uporabniški' pristop do numeričnih metod. Pri večini metod je poleg izpeljave dodan tudi algoritem, zapisan kot program v MATLABu. Tako zapisan algoritem nima samo namena prihraniti bralcu delo pri programiranju določene metode, ampak predvsem ilustrirati konstrukcijo algoritma. Vsakemu poglavju je dodanih tudi nekaj nalog. Nekatere izmed teh nalog neposredno dopolnjujejo izpeljavo v tekstu, druge pa so namenjene bolj preverjanju razumevanja snovi in so povzete po izpitnih nalogah.

V dodatku je kratek pregled dostopnih programov, uporabnih pri numeričnem reševanju raznih problemov. Navedeni so nekateri samostojni programi, kot tudi knjižnice podprogramov, od katerih je nekaj prosto dostopnih na računalniškem omrežju, drugi pa so komercialni, dostopni le proti plačilu.

Program predmeta *Numerična matematika* za študente računalništva obsega tri tedenske ure predavanj, zato njegova vsebina obsega vso snov, zajeto v tej knjigi. Študentje elektrotehnike pa imajo pri predmetu *Numerične metode* le dve uri predavanj na teden, zato je vsebina tega predmeta krajša za razdelke 2.6 Metoda Choleskega, 5.7 Ortogonalni polinomi nad sistemom točk, 6.5 Izbira koraka in 7.5 Kontrola koraka.

Nastanek tega dela je ves čas spremljal profesor Z. Bohte. Za njegove pripombe, ki so v marsičem izboljšale vsebino, sem mu od srca hvaležen. Prav tako se zahvaljujem za njune pripombe docentki N. Mramor Kosta in asistentu E. Žagarju za končni pregled. Zahvaljujem se tudi Založbi FE in FRI ter njenemu uredniku P. Šegi, ki sta omogočili izid tega dela.

Bojan Orel

Kazalo

Poglavje 1

Uvod

1.1 Zakaj numerične metode

Pri praktičnem reševanju problemov iz znanosti in tehnike pogosto srečamo matematične izraze, ki jih je težko ali pa jih sploh ni mogoče točno izračunati s klasičnimi analitičnimi metodami. V takih primerih se moramo navadno zateči po pomoč k računalniku in numeričnim metodam.

Pred dobo elektronskih računalnikov so matematiki reševali težje probleme tako, da so jih poenostavljali, zanemarjali vpliv količin, za katere so domnevali, da na končni rezultat nimajo bistvenega vpliva in nato z analitičnimi metodami poiskali rešitev, ki pa je bila zaradi poenostavitev le približna rešitev prvotnega problema.

Današnji zmogljivi računalniki nam omogočajo, da s pomočjo numeričnih metod rešujemo tudi mnogo zapletenejše probleme, kot bi jih zmogli z analitičnimi metodami, vendar pa se moramo tudi v tem primeru zaradi napak, ki so nujno povezane z numeričnim računanjem, sprijazniti z le približnimi rešitvami problemov.

Kdaj se splača uporabiti numerične metode? Predvsem takrat, ko problema drugače ne znamo rešiti, včasih pa tudi takrat, ko je analitična rešitev preveč zapletena in potrebujemo rešitev le za točno določene vrednosti podatkov. Zgodi pa se lahko, da problem lahko rešimo analitično, nato pa s primerno numerično metodo rezultate predelamo v lažje predstavljivo obliko (tabela, graf, ...) ali za nadaljnjo analizo.

Pri numeričnem reševanju problemov lahko ločimo več bolj ali manj različnih faz. Prva faza je običajno *formulacija* problema v obliki matematičnega

modela. Že v tej fazi se moramo zavedati, da bomo problem na koncu reševali z računalnikom, zato moramo pripraviti pravilne vhodne podatke, ugotoviti, kako bomo lahko preverjali pravilnost in smiselnost vmesnih rezultatov in se odločiti, kakšne, koliko in kako natančne končne rezultate bomo potrebovali.

Ko smo formulirali problem, se moramo odločiti, s katerimi numeričnimi metodami ga bomo reševali. Numerični metodi, pripravljeni za reševanje problema, bomo rekli *algoritem*. Algoritem je popoln in nedvoumen zapis postopkov, ki nas vodijo do rešitve problema. Z izbiro in konstrukcijo algoritmov, ustreznih za reševanje določenih problemov, se ukvarja *numerična analiza*. Ko se enkrat odločimo za primerno metodo, moramo oceniti velikost morebitnih napak, določiti primerne parametre numerične metode, kot so število iteracij, velikost koraka, ..., in predvideti sprotno preverjanje točnosti vmesnih rezultatov, kot tudi morebitne intervencije v primeru, ko so napake prevelike ali iteracije ne konvergirajo.

Naslednja faza v reševanju problema je *programiranje*. Izbrani algoritem moramo zapisati kot zaporedje ukazov, ki ga razume računalnik. Pri tem imamo na razpolago več možnosti. Najučinkovitejši je navadno zapis programa v kakšnem od programskih jezikov, kot so FORTRAN, C, Java, Python ... . Pri tem si lahko navadno pomagamo z bogatimi zbirkami že napisanih podprogramov, ki nam lahko močno olajšajo delo. Pri reševanju manjših problemov pa je navadno smiselna uporaba interaktivnih računalniških sistemov za numerično in/ali simbolično računanje, kot so Matlab, Mathematica in podobni. V nadaljevanju se bomo pri zapisovanju algoritmov pretežno držali stila, ki je običajen v programskem paketu Matlab.

Numerične metode so se z razvojem računalnikov tudi same v zadnjih petdesetih letih močno razvile, kljub temu pa se tudi danes še vedno uporabljajo metode, ki so poznane že več stoletij, le da so nekatere med njimi nekoliko prirejene. V zadnjem času so z razvojem vzporednih (večprocesorskih) računalnikov pravi preporod doživele nekatere stare, sicer že povsem odpisane metode. Ker se hiter razvoj računalništva še vedno nadaljuje, se bodo verjetno še nekaj časa intenzivno razvijale tudi numerične metode.

1.2 Napake in numerično računanje

Pri numeričnem računanju se ne moremo izogniti napakam. Če hočemo, da bo napaka pri končnem rezultatu manjša od maksimalne dopustne napake,

moramo poznati izvor napak pri numeričnem reševanju problemov in njihov vpliv na končni rezultat. Zato si najprej oglejmo možne izvore napak.

Napake v matematičnem modelu Kadar hočemo z matematičnim modelom opisati nek naravni pojav, se naš model le redko povsem sklada s fizikalno realnostjo. Pri računanju npr. položaja nebesnih teles v našem osončju bi morali za natančen model upoštevati Sonce, vse planete, njihove satelite, planetoide, komete in vsa druga telesa, ki se stalno ali občasno pojavljajo v našem osončju. Ker je teh teles ogromno in ker vseh niti ne poznamo, računamo le s tistimi telesi, za katera se nam zdi, da bistveno vplivajo na gibanje ostalih teles. Zato naš model ni povsem natančna slika pravega osončja in v računanju se zato pojavijo napake, za katere lahko le upamo, da končnega rezultata ne bodo preveč pokvarile.

Z napakami, ki nastanejo zaradi nepopolnega matematičnega modela, se numerična analiza navadno ne ukvarja, kljub temu pa vplivajo na pravilnost končnega rezultata.

Človeški faktor To je izvor napak, ki je skoraj vsakemu dobro znan. V predračunalniški dobi so bile to v glavnem izolirane napake pri posameznih aritmetičnih operacijah in pri računanju so bile potrebne zapletene navzkrižne kontrole za njihovo odkrivanje. Danes spadajo v to kategorijo predvsem napake pri programiranju. Da bi njihov vpliv čim bolj odpravili, moramo podrobno testirati vsak nov računalniški program. Najprej ga preizkusimo s podatki, za katere poznamo pravilen rezultat. Zapletene programe preizkušamo najlažje po posameznih, bolj ali manj samostojnih delih.

Napake v računalniku so neprijetne in zelo zahrbtne, saj običajno predpostavljamo, da procesor deluje pravilno. Kljub temu se lahko zgodi, da računalniški procesor vrne rezultat, ki je napačen. Znan je tako imenovani *Pentium bug*, napaka, ki jo je imela ena od začetnih serij Intelovih Pentium procesorjev `https://en.wikipedia.org/wiki/Pentium_FDIV_bug`.

Nenatančni podatki Često so vhodni podatki problemov rezultat meritev ali predhodnih izračunov in zato ne morejo biti natančni. Z metodami numerične analize teh napak ne moremo odpraviti, lahko pa ugotovimo, kako vplivajo na končni rezultat. Z izbiro ustreznih numeričnih algoritmov lahko

tudi do neke mere vplivamo na velikost napake, ki jo povzroča nenatančnost vhodnih podatkov.

Računanje z omejeno natančnostjo Števila v računalniku so vedno shranjena z omejeno natančnostjo. Kako so števila spravljena v računalniku, si bomo podrobneje ogledali v naslednjem razdelku. Zaradi te omejene natančnosti so nenatančne tudi vse aritmetične operacije v računalniku (zaokrožitvene napake). Pri reševanju nekaterih problemov, kot je reševanje sistemov linearnih enačb, so napake zaradi nenatančne aritmetike glavni izvor napak v končnem rezultatu. Znanih je več hudih nesreč, ki so nastale kot posledice napak zaradi računanja z omejeno natančnostjo (glej `http://www.ima.umn.edu/~arnold/disasters/patriot.html`, `http://www.ima.umn.edu/~arnold/disasters/ariane.html`).

Napake numeričnih metod To so napake, ki nastanejo, ko skušamo neskončne procese, ki nastopajo v matematiki, izračunati s končnim številom računskih operacij, npr. limito zaporedja nadomestimo z dovolj poznim členom, namesto odvoda vzamemo diferenčni kvocient, namesto integrala izračunamo končno integralsko vsoto. Te, tako imenovane *napake metode*, bomo poskušali oceniti pri vsaki posamezni numerični metodi, ki jo bomo obravnavali.

Absolutna in relativna napaka Napako pri določitvi vrednosti poljubne količine lahko opišemo na dva načina: kot *absolutno* napako ali kot *relativno* napako. Absolutna napaka je definirana kot

$$\text{Absolutna napaka} = \text{približna vrednost} - \text{točna vrednost},$$

relativna napaka pa kot

$$\text{Relativna napaka} = \frac{\text{Absolutna napaka}}{\text{točna vrednost}}.$$

Relativno napako lahko podamo direktno ali v odstotkih.

Primer 1.1. Kot ilustracijo si poglejmo znani približek

$$\pi \approx \frac{22}{7}.$$

V tem primeru je točna vredost $= \pi = 3.14159265\ldots$ in približna vrednost $= 22/7 = 3.1428571\ldots$ Zato je

$$\begin{aligned} \text{Absolutna napaka} &= \quad \tfrac{22}{7} - \pi \quad &= 0.00126\ldots, \\ \text{Relativna napaka} &= \quad (\tfrac{22}{7} - \pi)/\pi \quad &= 0.000402\ldots \approx 0.04\%. \end{aligned}$$

V tesni zvezi s pojmom relativne napake je število točnih mest v zapisu vrednosti neke količine. Kadar je

$$|\text{Relativna napaka}| \leq 0.5 \cdot 10^{-m},$$

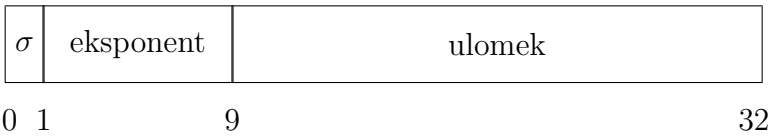
je v desetiškem zapisu vrednosti ustrezne količine pravilnih vsaj m mest. Npr. število 22/7 kot približek za število π ima točna 3 mesta.

1.3 Računalniška aritmetika

Realna števila so v računalniku običajno shranjena v obliki dvojiškega števila s premično piko, to je v obliki

$$x = \sigma \cdot \bar{x} \cdot 2^e, \tag{1.1}$$

kjer je predznak σ enak 1 ali -1, $\bar{x}$ je dvojiški ulomek (mantisa), za katerega velja $\frac{1}{2} \leq \bar{x} < 1$, e pa je celo število (dvojiški eksponent). Običajno računalniki zapišejo realno število v eno *računalnikovo besedo* (slika 1.1).



Slika 1.1: Zapis realnega števila v računalnikovo besedo.

Zapisi binarnih števil s premično piko se razlikujejo od računalnika do računalnika, kar povzroča nemalo težav pri prenašanju programov z enega

tipa računalnika na drugega. Na srečo se je v zadnjih desetih letih uveljavil standard IEEE, ki ga uporablja večina danes najbolj razširjenih mikroračunalnikov in delovnih postaj. Po standardu IEEE ima binarni ulomek $\bar{x}$ 24 binarnih mest, kar je dovolj za 7-mestno decimalno natančnost. Eksponent mora biti v mejah $-126 \leq e \leq 127$. Za večja števila ta standard uporablja simbol ∞, kar pomeni vrednost, ki je večja od največjega števila, ki ga v računalnikovi besedi še lahko zapišemo. Tako je $1/0 = \infty$ in $-1/0 = -\infty$. Posebna oznaka je predvidena tudi za nedoločeno število NaN, ki ga lahko dobimo npr, kot rezultat operacije $0/0$. Pri računanju s simboloma ∞ in NaN so določena natančna pravila, tako je, naprimer, $1/\infty = 0$, $1 \cdot \infty = \infty$, $\infty + \infty = \infty$, vendar $\infty - \infty = NaN$. V programu se seveda lahko vprašamo, ali ima določena spremenljivka vrednost ∞ ali NaN in temu primerno ukrepamo.

Poleg običajnih števil s premično piko ima večina računalnikov za natančnejše računanje na razpolago tudi števila z dvojno natančnostjo. Pri računalnikih, ki se držijo IEEE standarda, to pomeni, da ima ulomek $\bar{x}$ 53 binarnih mest, kar ustreza natančnosti skoraj 16 decimalnih mest, eksponent pa mora biti $-1022 \leq e \leq 1023$.

Zaokrožitvena napaka V računalniški pomnilnik lahko zapišemo realna števila navadno v binarni obliki s premično piko. Ker ima binarni ulomek v (1.1) le končno mnogo mest, lahko na ta način zapišemo le končno mnogo različnih realnih števil (pravimo jim *predstavljiva števila*), v splošnem pa moramo realno število x nadomestiti s primernim predstavljivim številom $fl(x)$. V navadi sta dva načina: *zaokrožanje* in *rezanje*. Pri zaokrožanju vzamemo za $fl(x)$ najbližje predstavljivo število, pri rezanju pa enostavno izpustimo odvečna binarna mesta. V obeh primerih povzročimo *zaokrožitveno napako*, ki je odvisna od velikosti števila x, zato jo zapišemo kot relativno napako

$$\delta = \frac{fl(x) - x}{x},$$

tako da velja

$$fl(x) = x(1 + \delta).$$

Relativna zaokrožitvena napaka δ je vedno omejena, njena natančna zgornja meja je odvisna od računalnika. Navadno jo označujemo z ε in jo imenujemo *osnovna zaokrožitvena napaka* ali *strojni epsilon* (glej [6]).

Zaokrožitvene napake se ne pojavljajo le pri vnosu realnih števil v računalnik, ampak tudi kot rezultat aritmetičnih operacij v računalniku samem. Naj

nam simbol $*$ označuje poljubno aritmetično operacijo (seštevanje, odštevanje množenje ali deljenje). Namesto pravega rezultata $x * y$ bomo dobili njegov približek $fl(x * y)$. Običajno lahko privzamemo, da tudi v tem primeru velja

$$fl(x * y) = (x * y)(1 + \delta),$$

kjer je $|\delta| \leq \varepsilon$. Osnove analize zaokrožitvenih napak so opisane v knjigi [6].

Primer 1.2. Izračunajmo vrednost izraza

$$c = a - (a - b),$$

kjer je $a = 10^9$ in $b \in (0, 1)$. Na računalniku, ki računa s standardno IEEE aritmetiko v enojni natančnosti (približno 7 decimalnih mest), bomo dobili napačen rezultat $c = 0$. Če pa račun nekoliko preuredimo: $c = (a - a) + b$, dobimo pravilen rezultat $c = b$.

Nekoliko drugačno sliko dobimo, če upoštevamo, da sta tudi oba operanda okužena z napakami. Tukaj bomo zaradi enostavnosti zanemarili napako, ki jo povzroči sama aritmetična operacija. Poglejmo si vsako operacijo posebej.

Množenje Vrednosti $x(1 + \delta_x)$ in $y(1 + \delta_y)$ naj predstavljata količini x in y, okuženi z relativnima napakama δ_x in δ_y. Izračunali bomo relativno napako produkta. Napaki δ_x in δ_y naj bosta dovolj majhni, da bomo lahko njune produkte δ_x^2, $\delta_x \cdot \delta_y$ in δ_y^2 zanemarili v primerjavi s samima okužbama δ_x in δ_y. Tako je

$$x(1 + \delta_x) \cdot y(1 + \delta_y) = x \cdot y(a + \delta_x + \delta_y + \delta_x\delta_y) \approx x \cdot y(1 + \delta_x + \delta_y),$$

od koder lahko preberemo, da je relativna napaka δ_{xy} produkta približno enaka

$$\delta_{xy} \approx \delta_x + \delta_y,$$

kar pomeni, da se pri množenju relativni napaki obeh faktorjev (približno) seštejeta v relativno napako produkta. S stališča razširjanja napak je to sprejemljivo, zato množenje smatramo za neproblematično operacijo.

Deljenje podobno kot pri množenju lahko izračunamo (če je le $y \neq 0$)

$$\frac{x(1 + \delta_x)}{y(1 + \delta_y)} = \frac{x}{y}(1 + \delta_x)(1 - \delta_y + \delta_y^2 - \cdots) \approx \frac{x}{y}(1 + \delta_x - \delta_y).$$

Za relativno napako kvocienta torej velja

$$\delta_{x/y} \approx \delta_x - \delta_y,$$

torej lahko tudi deljenje smatramo za neproblematično operacijo.

$x =$	$+$	e	1 0 1 0 0 1 1 1 1 1 1 g g
$y =$	$-$	e	1 0 1 0 0 1 1 1 1 0 1 g g
$x + y =$	$+$	e	0 0 0 0 0 0 0 0 0 1 0 g g
$=$	$+$	$e - 9$	1 0 g g ? ? ? ? ? ? ? ? ?

Slika 1.2: Seštevanje dveh števil nasprotnega predznaka, ki imata podobni absolutni vrednosti.

Seštevanje in odštevanje Ker imata lahko števili x in y poljuben predznak je dovolj, če si ogledamo samo seštevanje. Izračunamo

$$x(1+\delta_x) + y(1+\delta_y) = x + y + x\delta_x + y\delta_y = (x+y)(1 + \frac{x}{x+y}\delta_x + \frac{y}{x+y}\delta_y),$$

od koder dobimo (če predpostavimo, da je $x + y \neq 0$)

$$\delta_{x+y} = \frac{x}{x+y}\delta_x + \frac{y}{x+y}\delta_y.$$

Podobno kot pri množenju in deljenju je relativna napaka linearna kombinacija relativnih napak obeh sumandov, vendar koeficienta nista več ± 1,

ampak sta lahko poljubno velika. Če imata x in y isti predznak, potem sta oba koeficienta pozitivna in omejena. V tem primeru velja

$$|\delta_{x+y}| \leq |\delta_x| + |\delta_y|,$$

kar že pomeni, da je seštevanje v tem primeru tudi neproblematična operacija. Drugače pa je v primeru, ko sta x in y nasprotno predznačena, to je takrat, kadar je $|x+y|$ majhen v primerjavi z x in y, še prav posebej izrazito, ko sta x in y nasprotno predznačeni števili približno enake absolutne vrednosti. Tedaj je relativna napaka vsote lahko zelo velika, zato se moramo takega seštevanja, če je le mogoče izogibati. Iz slike 1.2 je razvidno, kako ta napaka nastane. Simboli g predstavljajo binarne števke, okužene z napako. Opazimo lahko, da končno normaliziranje rezultata premakne prvo nezanesljivo števko z 12. mesta na tretje.

1.4 Računanje vrednosti polinoma

Na preprostem primeru računanja vrednosti polinoma si poglejmo, kako lahko s primerno organizacijo računskega algoritma povečamo učinkovitost računanja. Obenem se bomo srečali tudi z zapisom algortima v obliki, ki je podobna Matlabovemu programu, to je v obliki, ki jo bomo pogosto srečevali tudi v naslednjih poglavjih, imeli pa bomo tudi priložnost primerjati učinkovitost različnih algoritmov za izračun vrednosti polinoma.

Polinom je običajno podan v *standardni obliki*

$$p_n(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0. \tag{1.2}$$

Če poznamo *koeficiente* $a_0, a_1, \ldots, a_n$ polinoma p_n in vrednost neodvisne spremenljike x, nam ni težko izračunati vrednosti polinoma pri spremenljivki x:

Algoritem 1.1. Izračun vrednosti polinoma
Koeficienti polinoma p_n naj bodo $b(i+1) = a_i, i = 0, 1, \ldots, n$, in x vrednost neodvisne spremenljivke. Naslednji algoritem izračuna vrednost polinoma (1.2) v točki x:

```
1     p = b(1);  t = 1
2     for i = 1 : n
3           t = t * x
4           p = p + t * b(i + 1)
5     end
```

Oglejmo si ta algoritem podrobneje in komentirajmo posamezne ukaze:

1. vrstica: Spremenljivkama p in t damo začetne vrednosti: spremenljivka z imenom p (ko bo algoritem končan, bo v spremenljivki p iskana vrednost polinoma) dobi vrednost koeficienta $b(1) = a_0$ (indeks vektorja ali matrike mora biti naravno število, zato moramo koeficiente oštevilčiti z indeksi od 1 do $n + 1$) in spremenljivka t (ki jo bomo potrebovali za računanje zaporednih potenc) dobi vrednost 1.

2. vrstica: Začetek zanke: ker je izraz $1 : n$ vektor, katerega komponente so zaporedna naravna števila med 1 in n, se naslednje vrstice do pripadajočega ukaza **end** izvedejo zaporedoma za vrednosti spremenljivke i od 1 do n.

3. vrstica: Vrednost spremenljivke t se pomnoži z x. Ker je začetna vrednost spremenljivke t enaka 1, je po i-tem prehodu skozi zanko enaka x^i.

4. vrstica: Vrednost spremenljivke p se poveča za $t * b(i+1)$, to je za $x^i a_i$, kar pomeni, da smo spremenljivki p prišteli vrednost i-tega člena.

5. vrstica: Prišteli smo vse člene polinoma, v spremenljivki p je sešteta vrednost vseh členov polinoma p.

Preštejmo število operacij, potrebnih za izračun vrednosti polinoma z algoritmom 1.1. Jedro algoritma (vrstici 3 in 4) se izvede n-krat, v 3. vrstici imamo eno množenje, v 4. pa eno seštevanje in eno množenje, kar nam da skupaj

$$m = \sum_{i=1}^{n} (1 + 1) = 2n \quad \text{in} \quad s = \sum_{i=1}^{n} 1 = n,$$

torej $m = 2n$ množenj in $s = n$ seštevanj.

Polinom (1.2) pa lahko zapišemo tudi v *vgnezdeni* obliki. Če iz vseh členov, razen zadnjega, izpostavimo x, dobimo $p_n(x) = p_{n-1}(x)x + a_0$, kjer je $p_{n-1}(x)$ polinom stopnje $n-1$ s koeficienti $a_n, \ldots, a_1$. Tudi v polinomu $p_{n-1}(x)$ izpostavimo x iz vseh členov razen zadnjega, in tako nadaljujemo, dokler ne dobimo polinoma v vgnezdeni obliki

$$p_n(x) = ((\cdots(a_n x + a_{n-1})x + \cdots + a_2)x + a_1)x + a_0. \tag{1.3}$$

Metoda za izračun vrednosti polinoma, zapisanega v vgnezdeni obliki (1.3), je poznana kot *Hornerjeva metoda*.

Algoritem 1.2. Izračun vrednosti polinoma — Hornerjeva metoda [a] Koeficienti polinoma p_n so $b(i+1) = a_i, i = 0, 1, \ldots, n$ in x vrednost neodvisne spremenljivke. Naslednji algoritem izračuna vrednost polinoma (1.3) v točki x:

```
1    p = b(n + 1)
2    for i = n : −1 : 1
3        p = p ∗ x + b(i)
4    end
```

[a] William George Horner (1786 Bristol – 1837 Bath). Angleški učitelj in šolski ravnatelj. Njegov edini pomembni prispevek matematiki je Hornerjeva metoda za reševanje algebraičnih enačb, ki jo je objavil leta 1819. Podobno metodo je nekaj let pred njim odkril že italijanski matematik Paolo Ruffini, čeprav jo je kitajski matematik Zhu Shijie uporabljal že kakih 500 let prej.

Tudi ta algoritem si oglejmo podrobneje in ga komentirajmo:

1. vrstica: Spremenljivki p damo začetno vrednost $b(n+1) = a_n$.
2. vrstica: Začetek zanke: izraz $n : -1 : 1$ je vektor, katerega komponente so zaporedna naravna števila od n po -1 do 1, zato se naslednje vrstice do pripadajočega ukaza **end** izvedejo zaporedoma za vrednosti spremenljivke i od n do 1.
3. vrstica: Vrednost spremenljivke p pomnožimo z x, produktu prištejemo $b(i) = a_{i-1}$.
4. vrstica: Algoritem je končan, vrednost polinoma je spravljena v spremenljivki p.

Tudi tokrat preštejmo število operacij, ki so potrebne za izračun vrednosti polinoma z algoritmom 1.2. Jedro algoritma (vrstica 3) se izvede n-krat, pri tem pa potrebujemo eno seštevanje in eno množenje, tako da je

$$m = \sum_{i=1}^{n} 1 = n \quad \text{in} \quad s = \sum_{i=1}^{n} 1 = n,$$

torej imamo $m = n$ množenj in $s = n$ seštevanj, kar je n množenj manj, kot je bilo potrebno pri algoritmu 1.1. To pomeni, da je Hornerjeva metoda bolj ekonomična od direktnega izračuna vrednosti polinoma z algoritmom 1.1.

Poleg standardne (1.2) in vgnezdene (1.3) oblike pa polinom lahko zapišemo še v različnih drugih oblikah. V poglavju 5 bomo srečali polinome, zapisane v *Newtonovi* obliki,

$$\begin{aligned} p_n(x) &= a_0 + a_1(x - c_1) + a_2(x - c_1)(x - c_2) + \cdots + \\ &\quad a_n(x - c_1)(x - c_2)\cdots(x - c_n), \end{aligned} \tag{1.4}$$

ki je tudi primerna za zapis, podoben vgnezdeni obliki, kar pomeni, da lahko vrednost polinoma izračunamo z algoritmom, ki je podoben Hornerjevi metodi.

1.5 Problemi

1. Kolikšni sta absolutna in relativna napaka, če namesto števila $e \approx 2.718281828459\ldots$ (osnove naravnih logaritmov) vzamemo približek $19/7$? Kolikšna je natančnost približka?

2. Približno vrednost integrala

   $$I = \int_0^1 e^{x^2}\, dx$$

   lahko izračunamo tako, da integrand nadomestimo z ustreznim Taylorjevim polinomom:

   $$e^{x^2} \approx 1 + x^2 + \frac{x^4}{2} + \frac{x^6}{6} + \frac{x^8}{24}.$$

   Potem je

   $$I \approx \int_0^1 \left(1 + x^2 + \frac{x^4}{2} + \frac{x^6}{6} + \frac{x^8}{24}\right) dx.$$

Kolikšna je v tem primeru napaka metode? (Navodilo: Uporabi formulo za ostanek Taylorjevega polinoma.)

3. Izračunaj vrednost funkcije

$$f(x) = x\left(\sqrt{x+1} - \sqrt{x}\right)$$

za vrednosti spremenljivke $x = 10^i;\ i = 1, \ldots, 20$. Rezultate primerjaj z vrednostmi, ki jih dobiš kot

$$f(x) = x\left(\frac{\sqrt{x+1} - \sqrt{x}}{1}\right)\left(\frac{\sqrt{x+1} + \sqrt{x}}{\sqrt{x+1} + \sqrt{x}}\right) = \frac{x}{\sqrt{x+1} + \sqrt{x}}.$$

Kateri od obeh rezultatov ima manjšo napako?

4. Primerjaj vrednosti leve in desne strani naslednjih enačb pri majhnih vrednostih x. V vsakem od primeru ugotovi, katera vrednost je pravilnejša:

   (a) $\dfrac{1-\cos x}{x^2} = \dfrac{\sin^2 x}{x^2(1+\cos x)}$,

   (b) $\sin(a+x) - \sin a = 2\cos\frac{a+x}{2}\sin\frac{x}{2}$,

   (c) $1 - \cos^2 x = \sin^2 x$,

   (d) $\log(1 + 1/x) + \log x = \log(1+x)$,

   (e) $\sqrt{1+x} - 1 = \frac{x}{\sqrt{1+x}+1}$

5. Preveri veljavnost identitete

$$\sum_{i=1}^{n} \frac{1}{i(i+1)} = \frac{n}{n+1}$$

za $n = 10, 100, 1000$. Vsoto vrste računaj: a) od prvega člena proti zadnjemu; b) od zadnjega člena proti prvemu.

6. Vrednosti integralov

$$I_n = \int_0^1 x^n e^{-x}\, dx \tag{1.5}$$

lahko računamo iz rekurzivne formule, ki jo dobimo, če (1.5) integriramo po delih:

$$\int_0^1 x^n e^{-x}\,dx = \left[-x^n e^{-x}\right]_{x=0}^{x=1} + n\int_0^1 x^{n-1}e^{-x}\,dx = \frac{-1}{e} + nI_{n-1}.$$

Ker je $I_0 = 1 - 1/e$ (preveri!), vrednosti I_n sestavljajo zaporedje, ki je podano rekurzivno s prepisom

$$I_n = nI_{n-1} - \frac{1}{e}; \quad I_0 = 1 - \frac{1}{e}. \tag{1.6}$$

(a) Izračunaj vrednosti $I_1, \ldots, I_{20}$ s pomočjo rekurzivne formule (1.6)

(b) Zaporedje I_n je padajoče (premisli zakaj). Ali je tudi zaporedje, ki si ga izračunal, padajoče?

(c) Če rekurzivno formulo (1.6) obrnemo

$$I_n = \frac{I_{n+1} + 1/e}{n+1},$$

lahko vrednosti I_n računamo 'nazaj', če le poznamo vrednost nekega I_n pri dovolj velikem n. Izračunaj vrednosti $I_{20}, \ldots, I_0$, če vzameš približek $I_{30} \approx 0$.

(d) Kako se spremenijo vrednosti $I_{20}, \ldots, I_0$, če izberemo začetno vrednost $I_{30} \approx 10^6$?

7. Zapiši algoritem, ki izračuna vrednost polinoma, zapisanega v Newtonovi obliki (1.4).

8. Algoritem 1.2 dopolni tako, da bo poleg vrednosti polinoma izračunal tudi vrednost njegovega odvoda. Delovanje algoritma preskusi na polinomu $p(x) = 3x^4 - 4x^3 + x^2 - 2x + 5$ pri vrednostih $x = 1$, $x = 10$ in $x = 0.1$.

Poglavje 2

Sistemi linearnih enačb

Najpogostejši problem, na katerega naletimo pri numeričnem računanju, je reševanje sistema linearnih enačb. Tak sistem lahko dobimo direktno iz matematične formulacije originalnega problema, ali pa je vmesna stopnja pri reševanju kakšnega zapletenejšega problema, npr. aproksimacije funkcije s polinomom, reševanju navadnih ali parcialnih diferencialnih enačb z diferenčno metodo, reševanju sistemov nelinearnih enačb, ...

V tem poglavju bomo najprej predstavili osnovne pojme v zvezi s sistemi linearnih enačb in njihovo rešljivostjo, v nadaljevanju pa si bomo ogledali nekaj metod za numerično reševanje sistemov linearnih enačb. Začeli bomo z algoritmoma za reševanje spodnjetrikotnega in zgornjetrikotnega sistema, nato pa si bomo podrobneje ogledali Gaussovo eliminacijsko metodo in njeno varianto, LU razcep (algoritem 2.4). Da bi se izognili nekaterim težavam pri reševanju, bomo opisali postopek delnega pivotiranja, na kratko pa si bomo ogledali tudi, kako lahko ocenimo natančnost izračunane rešitve. Na koncu se bomo spoznali še z nekaterimi iteracijskimi metodami, kot so Jacobijevo, Gauss-Seidlova metoda, metoda SOR in metoda konjugiranih gradientov. Iteracijske metode imajo prednost pred Gaussovo eliminacijsko metodo predvsem pri reševanju zelo velikih sistemov linearnih enačb, ki jih najpogosteje srečujemo pri računanju rešitev parcialnih diferencialnih enačb.

2.1 Predstavitev problema

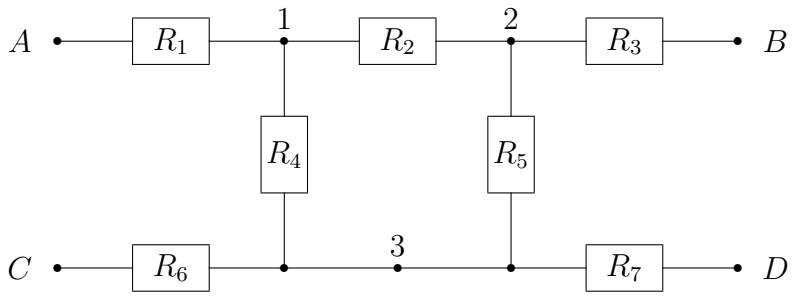
Sistem m linearnih enačb z n neznankami lahko zapišemo kot

$$\begin{array}{ccccccccc} a_{11}x_1 & + & a_{12}x_2 & + & \cdots & + & a_{1n}x_n & = & b_1, \\ a_{21}x_1 & + & a_{22}x_2 & + & \cdots & + & a_{2n}x_n & = & b_2, \\ \vdots & & \vdots & & & & \vdots & & \vdots \\ a_{m1}x_1 & + & a_{m2}x_2 & + & \cdots & + & a_{mn}x_n & = & b_m. \end{array} \tag{2.1}$$

Koeficienti sistema $a_{ij}\,(i = 1, \ldots, m; j = 1, \ldots, n)$ in *desne strani* $b_i\,(i = 1, \ldots, m)$ so dana števila. Poiskati moramo (če je to mogoče) števila $x_j\,(j = 1, \ldots, n)$ tako, da bodo enačbe (2.1) vse hkrati izpolnjene. Sistem enačb (2.1) navadno zapišemo v matrični obliki

$$A\boldsymbol{x} = \boldsymbol{b}, \tag{2.2}$$

kjer je $A = [a_{ij}]_{m\times n}$ matrika koeficientov (osnovna matrika sistema), $\boldsymbol{b} = [b_i]_m$ vektor desnih strani in $\boldsymbol{x} = [x_j]_n$ vektor neznank.



Slika 2.1: Električno vezje iz primera 2.1

Primer 2.1. Kot primer uporabe sistema linearnih enačb pri reševanju konkretnega problema vzemimo električno vezje na sliki 2.1. Pri danih vrednostih uporov $R_1, R_2, \ldots, R_7$ in danih napetostih U_A, U_B, U_C in U_D v točkah A, B, C in D nas zanimajo napetosti U_1, U_2 in U_3 v točkah 1, 2 in 3.
Za reševanje uporabimo *Kirchhoffov zakon* [a]: vsota električnih tokov, ki tečejo v vsako vozlišče električnega vezja je enaka 0. To zapišemo simbolično $\sum I = 0$, kjer je, po Ohmovem [b] zakonu, $I = \Delta U/R$.
Tako dobimo za vsoto tokov v točkah 1, 2 in 3 enačbe

$$\begin{aligned} \frac{U_1 - U_A}{R_1} + \frac{U_1 - U_2}{R_2} + \frac{U_1 - U_3}{R_4} &= 0 \\ \frac{U_2 - U_1}{R_2} + \frac{U_2 - U_B}{R_3} + \frac{U_2 - U_3}{R_5} &= 0 \\ \frac{U_3 - U_C}{R_6} + \frac{U_3 - U_1}{R_4} + \frac{U_3 - U_2}{R_5} + \frac{U_3 - U_D}{R_7} &= 0. \end{aligned}$$

Ko te enačbe preuredimo, dobimo linearni sistem

$$A\boldsymbol{U} = \boldsymbol{R},$$

kjer je

$$A = \begin{bmatrix} R_8 & -R_1R_4 & -R_1R_2 \\ -R_3R_5 & R_9 & -R_2R_3 \\ -R_5R_6R_7 & -R_4R_6R_7 & R_{10}, \end{bmatrix}$$

(tu smo pisali $R_8 = R_1R_2 + R_1R_4 + R_2R_4$, $R_9 = R_2R_3 + R_2R_5 + R_3R_5$ in $R_{10} = R_4R_5R_6 + R_4R_5R_7 + R_4R_6R_7 + R_5R_6R_7$),

$$\boldsymbol{R} = \begin{bmatrix} R_2R_4U_A \\ R_2R_5U_B \\ R_4R_5R_7U_C + R_4R_5R_6U_D \end{bmatrix} \quad \text{in} \quad \boldsymbol{U} = \begin{bmatrix} U_1 \\ U_2 \\ U_3. \end{bmatrix}.$$

[a] Gustav Robert Kirchhoff (1824 Königsberg – 1887 Berlin), nemški fizik, ki se je ukvarjal predvsem elektrotehniko in analizo spektrov. Svoja zakona o tokovih v električnih vezjih je objavil leta 1845.

[b] Georg Simon Ohm (1789 Erlangen – 1854 Munchen), nemški fizik. Ukvarjal se je predvsem z elektrotehniko in akustiko. Po njem se imenuje enota za električno upornost.

Preden se lotimo reševanja sistema (2.2), povzemimo znani rezultat iz

linearne algebre o rešljivosti sistemov linearnih enačb:

Izrek 2.1. *Za sistem linearnih enačb $A\boldsymbol{x} = \boldsymbol{b}$ veljajo naslednje trditve:*

1. *Sistem ima rešitev natanko tedaj, ko je vektor $\boldsymbol{b}$ linearna kombinacija stolpcev matrike A.*
2. *Sistem ima rešitev natanko tedaj, ko je rang razširjene matrike $[A \,|\, \boldsymbol{b}\,]$ enak rangu osnovne matrike:*
$$\text{rang}[A \,|\, \boldsymbol{b}\,] = \text{rang}A.$$
3. *Sistem ima največ eno rešitev natanko tedaj, ko ima ustrezen homogen sistem $A\boldsymbol{x} = \boldsymbol{0}$ samo trivialno rešitev $\boldsymbol{x} = \boldsymbol{0}$.*
4. *Vsak homogen sistem $A\boldsymbol{x} = \boldsymbol{0}$, ki ima manj enačb kakor neznank, ima netrivialne rešitve.*
5. *Če ima sistem (2.2) rešitev za poljuben vektor $\boldsymbol{b}$, potem število enačb ni večje od števila neznank $m \leq n$.*

Običajno nas zanimajo taki sistemi $A\boldsymbol{x} = \boldsymbol{b}$, ki imajo pri vsaki desni strani $\boldsymbol{b}$ natanko eno rešitev. Zaradi izreka 2.1 se lahko omejimo na sisteme enačb s kvadratno matriko, to je sisteme, kjer je število enačb enako številu neznank.

Izrek 2.2. *Naj bo število enačb m v sistemu $A\boldsymbol{x} = \boldsymbol{b}$ enako številu neznank n. Potem so enakovredne naslednje trditve:*

1. *Homogen sistem $A\boldsymbol{x} = \boldsymbol{0}$ ima le trivialno rešitev $\boldsymbol{x} = \boldsymbol{0}$.*
2. *Sistem (2.2) ima natanko eno rešitev za poljuben vektor $\boldsymbol{b}$.*
3. *Če sistem (2.2) za nek $\boldsymbol{b}$ ima rešitev, je ta ena sama.*
4. *Vrstice (stolpci) matrike A so linearno neodvisni vektorji.*
5. $\det A \neq 0$.
6. *Obstaja inverzna matrika A^{-1}, da je $AA^{-1} = A^{-1}A = I$.*
7. $\boldsymbol{x} = A^{-1}\boldsymbol{b}$.

Omejili se bomo na reševanje nesingularnih sistemov, to so sistemi z nesingularno ($\det A \neq 0$) matriko A. V tem primeru lahko rešitev zapišemo s pomočjo determinant po Cramerjevemu[1] pravilu, vendar s praktičnega stališča ta rešitev ni zanimiva, saj računanje vrednosti ene determinante zahteva približno ravno toliko računskih operacij, kot reševanje celotnega sistema linearnih enačb.

Pripomniti moramo tudi, da je pogoj $\det A \neq 0$ pri numeričnem reševanju zelo težko preverljiv, saj se vrednost determinante zelo spremeni, če vse enačbe sistema (2.1) pomnožimo z istim faktorjem α (kar seveda na rešitev sistema sploh ne vpliva). Za determinanto matrike, pomnožene s skalarjem α, velja

$$\det(\alpha A) = \alpha^n \det A.$$

Za primer vzemimo sistem s 30 neznankami (kar je dokaj majhen sistem za današnje standarde) in vse enačbe pomnožimo z $1/10$. Tako je

$$\det\left(\frac{1}{10}A\right) = 10^{-30} \det A,$$

kar pomeni, da deljenje vseh enačb sistema z 10 zmanjša determinanto sistema za faktor 10^{-30}, kar je praktično težko razlikovati od 0. Zato test

[1] Gabriel Cramer (1704 Ženeva – 1752 Bagnols-sur-Cèze), švicarski matematik, ukvarjal se je predvsem z geometrijo in zgodovino matematike. Svoje pravilo za rešitev sistema linearnih enačb je objavil leta 1750.

rešljivosti z determinanto uporabljamo bolj v teoretičnih razmišljanjih.

2.2 Trikotni sistemi

Preden se lotimo reševanja splošnega sistema (2.1), si oglejmo, kako lahko rešimo sistem s trikotno matriko. Matrika L je *spodnjetrikotna*, če za njene elemente l_{ij} velja, da je $l_{ij} = 0$ za $i < j$. Podobno je U *zgornje trikotna* matrika, če za njene elemente u_{ij} velja, da je $u_{ij} = 0$ za $i > j$. Preprosto povedano: zgornja trikotna matrika ima elemente, različne od 0, le na glavni diagonali in nad njo, spodnjetrikotna matrika pa ima elemente, različne od 0, le na glavni diagonali in pod njo. Tako spodnjetrikotno matriko L reda 4 lahko zapišemo kot

$$L = \begin{bmatrix} l_{11} & 0 & 0 & 0 \\ l_{21} & l_{22} & 0 & 0 \\ l_{31} & l_{32} & l_{33} & 0 \\ l_{41} & l_{42} & l_{43} & l_{44} \end{bmatrix}.$$

Poglejmo si najprej, kako rešimo sistem 3 linearnih enačb s spodnjetrikotno matriko. Najprej sistem enačb zapišimo:

$$\begin{aligned} l_{11}x_1 + 0 \quad\quad + 0 \quad\quad &= b_1, \\ l_{21}x_1 + l_{22}x_2 + 0 \quad\quad &= b_2, \\ l_{31}x_1 + l_{32}x_2 + l_{33}x_1 &= b_3. \end{aligned} \quad . \tag{2.3}$$

Ker je determinanta tega sistema enaka produktu diagonalnih elementov, $\det L = l_{11}l_{22}l_{33}$, je sistem enolično rešljiv natanko tedaj, ko so diagonalni elementi vsi različni od 0.

Iz prve enačbe izračunamo $x_1 = b_1/l_{11}$, to vrednost vstavimo v drugo enačbo in jo rešimo

$$x_2 = \frac{b_2 - l_{21}x_1}{l_{22}},$$

nato pa vstavimo dobljeni vrednosti za x_1 in x_2 v tretjo enačbo, iz katere lahko izračunano še vrednost neznanke x_3:

$$x_3 = \frac{b_3 - l_{31}x_1 - l_{32}x_2}{l_{33}}.$$

Če so koeficienti $l_{11},\ l_{22}$ in l_{33} različni od nič, smo rešili sistem (2.3).

Pri reševanju splošnega spodnjetrikotnega sistema $L\boldsymbol{x} = \boldsymbol{b}$ reda n moramo i-to enačbo rešiti glede na neznanko x_i:

$$x_i = \frac{b_i - \sum_{j=1}^{i-1} l_{ij} x_j}{l_{ii}}.$$

To moramo ponoviti po vrsti za $i = 1, \ldots, n$. Ker desno stran b_i potrebujemo le pri reševanju i-te enačbe, kasneje pa nič več, lahko na njeno mesto v pomnilniku zapišemo vrednost rešitve x_i. Tako dobimo algoritem, ki ga imenujmo *direktno vstavljanje*. Zapišimo ga kot zaporedje operacij nad elementi ustreznih matrik:

Algoritem 2.1. Direktno vstavljanje Naj bo L spodnjetrikotna nesingularna matrika reda n in $\boldsymbol{b}$ n-dimenzionalni vektor (stolpec). Naslednji algoritem na mesto, kjer je bil vektor $\boldsymbol{b}$, zapiše rešitev sistema $L\boldsymbol{x} = \boldsymbol{b}$.

$b(1) = b(1)/L(1,1)$
for $i = 2 : n$
    for $j = 1 : i - 1$
        $b(i) = b(i) - L(i,j) * b(j)$
    end
    $b(i) = b(i)/L(i,i)$
end

Isti algoritem lahko zapišemo v bolj kompaktni obliki, če uporabimo MATLABov zapis s podmatrikami:

$b(1) = b(1)/L(1,1)$
for $i = 2 : n$
    $b(i) = (b(i) - L(i, 1 : i-1) * b(1 : i-1))/L(i,i)$
end

Preštejmo aritmetične operacije, potrebne za izvedbo tega algoritma. V notranji zanki, ki je v kompaktnem zapisu skrita v skalarnem produktu $L(i, 1 : i-1) * b(1 : i-1)$, imamo eno množenje in eno odštevanje. Notranja zanka se izvede $i - 1$ krat, torej imamo $2(i-1)$ operacij. V zunanji zanki, ki se izvede $n - 1$ krat, imamo poleg notranje zanke še eno deljenje,

kar nam da skupaj

$$1 + \sum_{i=2}^{n}(1 + \sum_{j=1}^{i-1} 2) = n + 2\sum_{i=1}^{n-1} i = n + n(n-1) = n^2$$

osnovnih aritmetičnih operacij.

Ustreznemu algoritmu za reševanje sistema $U\boldsymbol{x} = \boldsymbol{b}$ z zgornjetrikotno matriko U pravimo *obratno vstavljanje*. Enačbe rešujemo od spodaj (od zadnje proti prvi), neznanko x_i pa izračunamo po formuli

$$x_i = \frac{b_i - \sum_{j=i+1}^{n} u_{ij}x_j}{u_{ii}}\,.$$

Tudi v tem primeru lahko vektor desnih strani $\boldsymbol{b}$ kar prepišemo z vektorjem rešitev x. Tako dobimo:

Algoritem 2.2. Obratno vstavljanje Naj bo U zgornjetrikotna nesingularna matrika reda n in $\boldsymbol{b}$ n-dimenzionalni vektor. Naslednji algoritem nadomesti vektor $\boldsymbol{b}$ z rešitvijo sistema $U\boldsymbol{x} = \boldsymbol{b}$.

```
b(n) = b(n)/U(n,n)
for i = n − 1 : −1 : 1
      for j = i + 1 : n
            b(i) = b(i) − U(i, j) ∗ b(j)
      end
      b(i) = b(i)/U(i, i)
end
```

Tudi ta algoritem zapišimo še v kompaktni obliki s podmatrikami:

```
b(n) = b(n)/U(n,n)
for i = n − 1 : −1 : 1
      b(i) = (b(i) − U(i, i + 1 : n) ∗ b(i + 1 : n))/U(i, i)
end
```

Podobno kot algoritem z direktnim vstavljanjem ima tudi algoritem z obratnim vstavljanjem časovno zahtevnost n^2.

2.3 Gaussova eliminacijska metoda

Kot smo ravnokar videli, je reševanje trikotnih sistemov enostavno, zato to poskusimo izkoristiti tudi pri sistemih, ki nimajo trikotne oblike. Ideja Gaussove[2] eliminacijske metode je splošen kvadratni sistem $A\boldsymbol{x} = \boldsymbol{b}$ preoblikovati v enakovreden trikotni sistem. To dosežemo s primernimi linearnimi kombinacijami enačb. Da bi se izognili težavam, bomo predpostavili, da ima matrika A vse vodilne poddeterminante $\det A(1:k, 1:k)$; $(k = 1, \ldots, n)$ različne od 0. Kako ravnamo, če ta pogoj ni izpolnjen, pa si bomo ogledali v razdelku 2.5.

Primer 2.2. Vzemimo kot primer naslednji sistem reda 2:

$$\begin{array}{rcrcl} 2x_1 & + & 3x_2 & = & 8 \\ 4x_1 & + & 7x_2 & = & 18. \end{array}$$

Če prvo enačbo, pomnoženo z 2, odštejemo od druge, dobimo enakovreden zgornjetrikotni sistem

$$\begin{array}{rrl} 2x_1 & +3x_2 & = 8, \\ & x_2 & = 2, \end{array}$$

od koder lahko z obratnim vstavljanjem izračunamo rešitev

$$x_2 = 2 \quad \text{in} \quad x_1 = \frac{8 - 3 \cdot 2}{2} = 1.$$

Če postopek, ki smo ga uporabili za reševanje primera 2.2 posplošimo na sistem n enačb, ga lahko formalno zapišemo kot:

[2] Johann Carl Friedrich Gauss (1777 Brunswick – 1855 Göttingen), nemški matematik, fizik in astronom, eden najpomembnejših matematikov. Ukvarjal se je skoraj z vsemi področji takratne matematike. Svojo metodo za reševanje sistemov linearnih enačb je razvil na osnovi stare kitajske metode, opisane v tekstu *Devet poglavij iz umetnosti matematike* iz obdobja okoli 200 pr. n. št., da bi lahko izračunal tirnico asteroida Pallas.

Algoritem 2.3. Gaussova eliminacija Naj bo dana kvadratna matrika A reda n z lastnostjo, da so vse njene glavne podmatrike $A(1\!:\!k, 1\!:\!k)$; $(k = 1, \ldots, n)$ nesingularne, in naj bo $\boldsymbol{b}$ n-dimenzionalni vektor. Naslednji algoritem preoblikuje sistem linearnih enačb $A\boldsymbol{x} = \boldsymbol{b}$ v enakovreden zgornjetrikotni sistem $U\boldsymbol{x} = \boldsymbol{c}$. Ko je algoritem končan, so elementi zgornjetrikotne matrike U zapisani v zgornjem trikotniku matrike A, preoblikovani vektor desnih strani $\boldsymbol{c}$ pa se nahaja na mestu vektorja $\boldsymbol{b}$.

for $k = 1 : n - 1$
  for $i = k + 1 : n$
    $M(i, k) = A(i, k)/A(k, k)$
    for $j = k + 1 : n$
      $A(i, j) = A(i, j) - M(i, k) * A(k, j)$
    end
    $b(i) = b(i) - M(i, k) * b(k)$
  end
end

Da bi izračunali rešitve prvotnega sistema $A\boldsymbol{x} = \boldsymbol{b}$, moramo le še z obratnim vstavljanjem rešiti zgornjetrikotni sistem $U\boldsymbol{x} = \boldsymbol{c}$.

Preštejmo še operacije, ki so potrebne za preoblikovanje sistema z Gaussovo eliminacijo

$$\sum_{k=1}^{n-1} \left(\sum_{i=k+1}^{n} \left(1 + \sum_{j=k+1}^{n} 2 \right) + 2 \right) = \frac{2n^3}{3} - \frac{n^2}{2} - \frac{7n}{6} \approx \frac{2n^3}{3},$$

torej ima Gaussova eliminacija časovno zahtevnost $2n^3/3$.

Primer 2.3. Sistem enačb

$$\begin{aligned} 3x_1 &+ 4x_2 &+ x_3 &= 6, \\ 5x_1 &+ 5x_2 &+ x_3 &= 6, \\ -2x_1 &+ 2x_2 &+ 4x_3 &= 10, \end{aligned}$$

rešimo z Gaussovo eliminacijsko metodo.
Začnimo z razširjeno matriko sistema

$$\left[\begin{array}{ccc|c} 3 & 4 & 1 & 6 \\ 5 & 5 & 1 & 6 \\ -2 & 2 & 4 & 10 \end{array}\right].$$

Po prvem koraku Gaussove eliminacije ($k = 1$) dobimo

$$\left[\begin{array}{ccc|c} 3 & 4 & 1 & 6 \\ 0 & -1.667 & -0.6667 & -4 \\ 0 & 4.667 & 4.667 & 14 \end{array}\right],$$

po drugem pa ($k = 2$)

$$\left[\begin{array}{ccc|c} 3 & 4 & 1 & 6 \\ 0 & -1.667 & -0.6667 & -4 \\ 0 & 0 & 2.8 & 2.8 \end{array}\right].$$

Končno z obratnim vstavljanjem izračunamo rešitev

$$x_1 = -1, \qquad x_2 = 2, \qquad x_3 = 1.$$

2.4 LU Razcep

V algoritmu 2.3 smo zgornji trikotnik (vključno z glavno diagonalo) matrike A prepisali z zgornjetrikotno matriko U, ki smo jo dobili kot rezultat Gaussove transformacije matrike A. Definirajmo še spodnjetrikotno matriko $L = [m_{ik}]$, katere (i, k)-ti element naj bo faktor m_{ik}, s katerim smo množili k-to vrstico,

preden smo jo odšteli od i-te, na glavni diagonali pa naj bodo enojke:

$$L = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ m_{21} & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ m_{n1} & \dots & m_{n,n-1} & 1 \end{bmatrix}. \tag{2.4}$$

Za matrike U, L in originalno matriko A velja naslednji rezultat:

Izrek 2.3. *Naj bo dana nesingularna matrika A, matrika U naj bo določena z algoritmom* 2.3*, matrika L pa s formulo* (2.4)*. Potem velja*

$$LU = A. \tag{2.5}$$

Dokaz: Naj bo $M_k\,(k = 1, \dots, n-1)$ matrika, ki jo dobimo, če v n razsežni enotski matriki v k-tem stolpcu pod glavno diagonalo namesto 0 pišemo $-m_{ik}$; $i = k+1, \dots, n$. Tako je

$$M_1 = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ -m_{21} & 1 & \cdots & 0 \\ -m_{31} & 0 & \cdots & 0 \\ \vdots & & \ddots & \vdots \\ -m_{n1} & 0 & \cdots & 1 \end{bmatrix}, \quad M_2 = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ 0 & -m_{32} & \cdots & 0 \\ & \vdots & \ddots & \vdots \\ 0 & -m_{n2} & \cdots & 1 \end{bmatrix}.$$

Izračunajmo

$$A_1 = M_1 A = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ -m_{21} & 1 & \cdots & 0 \\ -m_{31} & 0 & \cdots & 0 \\ \vdots & & \ddots & \vdots \\ -m_{n1} & 0 & \cdots & 1 \end{bmatrix} \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ a_{31} & a_{32} & \cdots & a_{3n} \\ & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix}$$

$$= \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ 0 & a'_{22} & \cdots & a'_{2n} \\ 0 & a'_{32} & \cdots & a'_{3n} \\ & \vdots & \ddots & \vdots \\ 0 & a'_{n2} & \cdots & a'_{nn} \end{bmatrix}$$

Tako je A_1 matrika, ki dobimo iz A po prvem koraku Gaussove eliminacije. Podobno izračunamo po vrsti še

$$A_2 = M_2 A_1, \ldots, A_{n-1} = M_{n-1} A_{n-2}$$

in ugotovimo, da je $A_{n-1} = U$ zgornjetrikotna matrika, ki jo dobimo pri Gaussovi eliminaciji. Tako imamo

$$M_{n-1} \cdots M_2 M_1 A = U.$$

Da bi dokazali trditev izreka, moramo to enačbo najprej pomnožiti z leve zaporedoma z $M_{n-1}^{-1}, \ldots, M_2^{-1}$ in M_1^{-1}, da dobimo

$$A = M_1^{-1} M_2^{-1} \cdots M_{n-1}^{-1} U,$$

nato pa izračunati produkt $M_1^{-1} M_2^{-1} \cdots M_{n-1}^{-1}$. Inverzne matrike M_k^{-1} dobimo iz matrik M_k tako, da zamenjamo predznake faktorjev m_{ij}, torej

$$M_1^{-1} = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ m_{21} & 1 & \cdots & 0 \\ m_{31} & 0 & \cdots & 0 \\ \vdots & & \ddots & \vdots \\ m_{n1} & 0 & \cdots & 1 \end{bmatrix}, \; M_2^{-1} = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ 0 & m_{32} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & m_{n2} & \cdots & 1 \end{bmatrix}, \tag{2.6}$$

in podobno za ostale, kar lahko enostavno preverimo z direktnim računom (glej problem 3).
Izračunati moramo še produkt $M_1^{-1} M_2^{-1} \ldots M_{n-1}^{-1}$. Najprej zmnožimo $M_1^{-1} M_2^{-1}$:

$$\begin{bmatrix} 1 & 0 & \cdots & 0 \\ m_{21} & 1 & \cdots & 0 \\ m_{31} & 0 & \cdots & 0 \\ \vdots & & \ddots & \vdots \\ m_{n1} & 0 & \cdots & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ 0 & m_{32} & \cdots & 0 \\ & \vdots & \ddots & \vdots \\ 0 & m_{n2} & \cdots & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ m_{21} & 1 & \cdots & 0 \\ m_{31} & m_{32} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ m_{n1} & m_{n2} & \cdots & 1 \end{bmatrix}.$$

Tako nadaljujemo, dokler na koncu ne dobimo spodnjetrikotne matrike $L = M_1^{-1} M_2^{-1} \ldots M_{n-1}^{-1}$, tako da velja enačba (2.5). Tako smo dokazali veljavnost izreka.

Zapišimo algoritem, ki nam izračuna LU razcep matrike A:

Algoritem 2.4. *LU* **razcep** Naj bo A kvadratna matrika reda n z lastnostjo, da so vse njene podmatrike $A(1:k, 1:k);\ k = 1, \ldots, n$ nesingularne. Naslednji algoritem izračuna razcep $A = LU$, kjer je L spodnjetrikotna matrika z enojkami na glavni diagonali, U pa zgornjetrikotna matrika. Ko je algoritem končan, so elementi matrike U zapisani v zgornjem trikotniku matrike A, elementi matrike L, razen enojk na diagonali, pa v spodnjem trikotniku matrike A.

for $k = 1 : n - 1$
  for $i = k + 1 : n$
    $A(i, k) = A(i, k)/A(k, k)$
    for $j = k + 1 : n$
      $A(i, j) = A(i, j) - A(i, k) * A(k, j)$
    end
  end
end

Preštejmo še operacije, ki so potrebne za LU razcep matrike z algoritmom 2.4:

$$\sum_{k=1}^{n-1} \left[\sum_{i=k+1}^{n} \left(1 + 2 \sum_{j=k+1}^{n} 1 \right) \right] =$$

$$\sum_{k=1}^{n-1} \left[n - k + 2(n-k)^2 \right] = \frac{n^3}{3} - \frac{n^2}{2} - \frac{n}{6} \approx \frac{2n^3}{3}.$$

Časovna zahtevnost algoritma za LU razcep je torej $2n^3/3$, prav tako kot časovna zahtevnost algoritma Gaussove eliminacije.

Kako si lahko z LU razcepom matrike A pomagamo pri reševanju sistema enačb (2.2)? Če smo matriko A razcepili v produkt LU, lahko sistem zapišemo kot

$$LU\boldsymbol{x} = \boldsymbol{b}$$

ki ga rešimo v dveh korakih. Najprej z direktnim vstavljanjem rešimo spodnjetrikotni sistem

$$L\boldsymbol{y} = \boldsymbol{b},$$

s pomožnim vektorjem $\boldsymbol{y} = U\boldsymbol{x}$, nato pa še z obratnim vstavljanjem zgornjetrikotni sistem

$$U\boldsymbol{x} = \boldsymbol{y}.$$

Izračunajmo potrebno število množenj/deljenj za rešitev sistema (2.2) po opisanem postopku. Za LU razcep potrebujemo $n^3/3 - n/3$ operacij, za direktno vstavljanje $n^2/2 - n/2$ (tukaj smo že upoštevali, da imamo na glavni diagonali enice) in $n^2/2 + n/2$ za obratno vstavljanje. Skupaj je to $n^3/3 + n^2 - n/3$ množenj/deljenj, kar je skoraj enako kot pri sami Gaussovi eliminaciji.

Glavnino operacij pri reševanju linearnega sistema predstavlja LU razcep, saj je število operacij za reševanje obeh trikotnih sistemov za velikostni red manjše. Tako se glavna prednost LU razcepa pokaže, kadar moramo zaporedoma rešiti več sistemov linearnih enačb (2.2) z isto matriko sistema A in različnimi desnimi stranmi. V tem primeru je potreben le en LU razcep, za vsak vektor desnih strani pa moramo rešiti po dva trikotna sistema, za kar pa je potrebno znatno manj (približno n^2) množenj/deljenj.

Primer 2.4. Za primer izračunajmo rešitev sistema linearnih enačb

$$\begin{aligned} 3x_1 + 2x_2 + 5x_3 + x_4 &= 1 \\ 6x_1 + 6x_2 + 15x_3 + 3x_4 &= -6 \\ -3x_1 + 4x_2 + 13x_3 + x_4 &= -17 \\ -6x_1 + 6x_2 + 15x_3 + 5x_4 &= -52. \end{aligned} \tag{2.7}$$

Najprej izračunajmo LU razcep matrike sistema

$$A = \begin{bmatrix} 3 & 2 & 5 & 1 \\ 6 & 6 & 15 & 3 \\ -3 & 4 & 13 & 1 \\ -6 & 6 & 15 & 5 \end{bmatrix}$$

s pomočjo algoritma 2.4. Spremljajmo, kaj se dogaja z matriko v posameznih korakih. Po prvem koraku ($k = 1$) dobimo namesto elementov matrike A

$$\begin{bmatrix} 3 & 2 & 5 & 1 \\ 2 & 2 & 5 & 1 \\ -1 & 6 & 18 & 2 \\ -2 & 10 & 25 & 7 \end{bmatrix},$$

po drugem koraku ($k = 2$) imamo

$$\begin{bmatrix} 3 & 2 & 5 & 1 \\ 2 & 2 & 5 & 1 \\ -1 & 3 & 3 & -1 \\ -2 & 5 & 0 & 2 \end{bmatrix},$$

po tretjem koraku ($k = 3$) pa se matrika ne spremeni, ker že ima 0 na mestu (4,3). Tako sta matriki L in U enaki

$$L = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ -1 & 3 & 1 & 0 \\ -2 & 5 & 0 & 1 \end{bmatrix}, \quad U = \begin{bmatrix} 3 & 2 & 5 & 1 \\ 0 & 2 & 5 & 1 \\ 0 & 0 & 3 & -1 \\ 0 & 0 & 0 & 2 \end{bmatrix}.$$

Rešiti moramo še oba trikotna sistema. Najprej $L\boldsymbol{y} = \boldsymbol{b}$ z direktnim vstavljanjem:

$$\begin{aligned} y_1 &= 1 \\ y_2 &= -6 - 2y_1 = -8 \\ y_3 &= -17 + y_1 - 3y_2 = 8 \\ y_4 &= -52 + 2y_1 - 5y_2 + 0y_3 = -10, \end{aligned}$$

nato pa še zgornjetrikotni sistem $U\boldsymbol{x} = \boldsymbol{y}$ z obratnim vstavljanjem:

$$\begin{aligned} x_4 &= \frac{-10}{2} = -5 \\ x_3 &= \frac{8 + x_4}{3} = 1 \\ x_2 &= \frac{-8 - x_4 - 5x_3}{2} = -4 \\ x_1 &= \frac{1 - x_4 - 5x_3 - 2x_2}{3} = 3 \end{aligned}$$

Rešitev sistema (2.7) je torej

$$\boldsymbol{x} = [3, -4, 1, -5]^T.$$

2.5 Pivotiranje

V algoritmih za Gaussovo eliminacijo in za LU razcep (algoritma 2.3 in 2.4) smo predpostavili, da ima matrika A vse vodilne poddeterminante $\det A(1:k, 1:k)$; $k = 1, \ldots, n$ različne od 0. S tem smo preprečili možnost, da bi bil kateri od diagonalni elementov, s katerim delimo, in ki ga običajno imenujemo *pivot*, lahko enak 0. Poglejmo si, kako se lahko izognemo delitvi z nič, če ta pogoj ni izpolnjen.

Denimo, da je na k-tem koraku LU razcepa element na diagonali $a_{kk} = 0$. Če je matrika A nesingularna, mora biti v k-tem stolpcu pod glavno diagonalo vsaj en element različen od 0. Denimo, da je $a_{ik} \neq 0$; $k + 1 \leq i \leq n$. V tem primeru lahko med seboj zamenjamo k-to in i-to vrstico, saj zamenjava dveh vrstic v matriki sistema ustreza zamenjavi dveh enačb, kar seveda rešitve sistema ne spremeni. Zgodi pa se, da tudi v primeru, ko so vsi pivotni elementi različni od 0, nismo zadovoljni z rešitvijo. Oglejmo si naslednji primer:

Primer 2.5. Rešimo sistem linearnih enačb

$$\begin{aligned} 6x_1 &+ 2x_2 &+ 2x_3 &= -2 \\ 2x_1 &+ \tfrac{2}{3}x_2 &+ \tfrac{1}{3}x_3 &= 1 \\ x_1 &+ 2x_2 &- x_3 &= 0, \end{aligned} \tag{2.8}$$

katerega točna rešitev je

$$x_1 = 2.6, \quad x_2 = -3.8 \quad \text{in} \quad x_3 = -5.0. \tag{2.9}$$

Računali bomo na štiri decimalna mesta z zaokrožanjem. Najprej zapišimo matriko sistema

$$\begin{bmatrix} 6.000 & 2.000 & 2.000 \\ 2.000 & 0.6667 & 0.3333 \\ 1.000 & 2.000 & -1.000 \end{bmatrix},$$

in izračunajmo njen LU razcep. Po prvem koraku je

$$\begin{bmatrix} 6.000 & 2.000 & 2.000 \\ 0.3333 & 0.0001000 & -0.3333 \\ 0.1667 & 1.667 & -1.333 \end{bmatrix}$$

in po drugem koraku

$$\begin{bmatrix} 6.000 & 2.000 & 2.000 \\ 0.3333 & 0.0001000 & -0.3333 \\ 0.1667 & 16670 & 5555 \end{bmatrix}.$$

Rešimo najprej spodnjetrikotni sistem

$$\begin{aligned} y_1 &= -2.000 \\ y_2 &= 1.000 - 0.3333y_1 = 1.667 \\ y_3 &= 0 - 0.1667y_1 - 16670y_2 = -27790, \end{aligned}$$

nato pa še zgornjetrikotnega

$$\begin{aligned} x_3 &= \frac{-27790}{5555} = -5.003 \\ x_2 &= \frac{1.667 + 0.3333x_3}{0.0001000} = 0.000 \\ x_1 &= \frac{-2.000 - 2.000x_3 - 2.000x_2}{6.000} = 1.335, \end{aligned}$$

kar se precej razlikuje od točne rešitve (2.9).
Vir težav v tem primeru je pivot na drugem koraku, ki bi moral biti enak 0, pa zaradi zaokrožitvene napake v prejšnjem koraku ni. Če bi pred drugim korakom zamenjali drugo in tretjo vrstico matrike, bi bil rezultat LU razcepa

$$\begin{bmatrix} 6.000 & 2.000 & 2.000 \\ 0.1667 & 1.667 & -1.333 \\ 0.3333 & 0.00005999 & -0.3332 \end{bmatrix}.$$

Sedaj dobimo iz spodnjetrikotnega sistema

$$\begin{aligned} y_1 &= -2.000 \\ y_2 &= 0 - 0.1667y_1 = 0.3334 \quad \text{(Zamenjali smo 2. in 3. enačbo!)} \\ y_3 &= 1.000 - 0.3333y_1 - 0.00006000y_2 = 1.667, \end{aligned}$$

nato pa še iz zgornjetrikotnega sistema

$$\begin{aligned} x_3 &= \frac{1.667}{-0.3332} = -5.003 \\ x_2 &= \frac{0.3334 + 1.333x_3}{1.667} = -3.801 \\ x_1 &= \frac{-2.000 - 2.000x_3 - 2.000x_2}{6.000} = 2.602, \end{aligned}$$

kar se mnogo bolje ujema s točno rešitvijo (2.9).

Problemom, kot v prejšnjem primeru, se večinoma lahko izognemo z zamenjavo vrstnega reda enačb (vrstic matrike), čemur pravimo *delno pivotiranje*. Na k-tem koraku Gaussove eliminacije (ali LU razcepa) izberemo za pivot po absolutni vredosti največji element v k-tem stolpcu od k-te do zadnje vrstice

$$c = \max_{k \le i \le n} |a_{ik}|.$$

Če je $|a_{kk}| < c$, potem zamenjamo k-to vrstico z eno od naslednjih, da dobimo po zamenjavi $|a_{kk}| = c$. Na ta način izberemo za pivot tisti element, ki je kar se da daleč od 0. Z delnim pivotiranjem dosežemo, da so vsi faktorji m_{ik}, torej vsi elementi matrike L, omejeni z 1

$$|l_{ij}| \le 1,$$

kar se pokaže kot ugodno tudi zaradi zaokrožitvenih napak.

Seveda pa izrek 2.5 za pivotiranje ne velja več v isti obliki. Pokazati se da, da je

$$PA = LU; \quad L\boldsymbol{y} = P\boldsymbol{b}; \quad U\boldsymbol{x} = \boldsymbol{y},$$

kjer je matrika P (pravimo ji *permutacijska matrika*) dobljena iz enotske matrike z istimi zamenjavami vrstic, kot jih zahteva delno pivotiranje za matriko A.

Dopolnimo algoritem 2.4 za izračun LU razcepa z delnim pivotiranjem:

Algoritem 2.5. *LU* **z delnim pivotiranjem** Naj bo A kvadratna, nesingularna matrika reda n. Naslednji algoritem izračuna razcep $PA = LU$, kjer je L spodnjetrikotna matrika z enojkami na glavni diagonali, U pa spodnjetrikotna matrika. Ko je algoritem končan, so elementi matrike U zapisani v zgornjem trikotniku matrike A, elementi matrike L, od katerih nobeden ni po absolutni vrednosti večji od 1 pa v spodnjem trikotniku matrike A. V celoštevilskem vektorju $\boldsymbol{p}$ so zapisane podrobnosti o zamenjavah vrstic.

```
for k = 1 : n − 1
    pivot = abs(A(k, k))
    p(k) = k
    for i = k + 1 : n
        if abs(A(i, k)) > pivot
            pivot = abs(A(i, k))
            p(k) = i
        end
    end                              %Element A_ik je največji
    if pivot = 0
        error('Matrika je singularna')
    end
    temp = A(k, :)                   %Zamenjamo k-to in p(k)-to vrstico
    A(k, :) = A(p(k), :)
    A(p(k), :) = temp
    for i = k + 1 : n
        A(i, k) = A(i, k)/A(k, k)
        for j = k + 1 : n
            A(i, j) = A(i, j) − A(i, k) ∗ A(k, j)
        end
    end
end
```

2.6 Metoda Choleskega

Splošni sistem linearnih enačb $A\boldsymbol{x} = \boldsymbol{b}$ najučinkoviteje rešimo tako, da faktoriziramo matriko $A = LU$, kjer je L spodnje trikotna in U zgornje trikotna matrika, nato pa rešimo dobljena trikotna sistema. Kadar pa je matrika A si-

metrična, je ugodno, če je tudi faktorizacija simetrična, na primer $A = R^T R$, kjer je R zgornje trikotna matrika. Žal na ta način ne moremo faktorizirati vsake simetrične matrike.

Poglejmo, kakšna mora biti matrika A, da bo imela simetrično faktorizacijo. Naj bo R nesingularna zgornje trikotna matrika, in $\boldsymbol{x}$ poljuben neničelen vektor. Potem je tudi $A = R^T R$ nesingularna in $\boldsymbol{y} = R\boldsymbol{x} \neq 0$. Tako je

$$\boldsymbol{x}^T A\boldsymbol{x} = \boldsymbol{x}^T R^T R\boldsymbol{x} = (R\boldsymbol{x})^T (R\boldsymbol{x}) = \boldsymbol{y}^T \boldsymbol{y} = \sum y_i^2 > 0.$$

Zato definirajmo:

Definicija 2.1. *Matrika A, za katero je kvadratna forma $\boldsymbol{x}^T A\boldsymbol{x} > 0$ za poljuben vektor $\boldsymbol{x} \neq 0$, se imenuje* pozitivno definitna.

Navedimo dve preprosti lastnosti pozitivno definitnih matrik, ki jih bomo kasneje s pridom uporabili:

Lastnost 1. *Pozitivno definitna matrika je nesingularna.*

Dokaz: Če je matrika A singularna, obstaja neničelen vektor $\boldsymbol{x}$, da je $A\boldsymbol{x} = 0$. Tako je tudi $\boldsymbol{x}^T A\boldsymbol{x} = 0$ in matrika A ni pozitivno definitna.

Lastnost 2. *Če je matrika $A = [a_{ij}]_{i,j=1}^n$ pozitivno definitna, potem je matrika $A_* = [a_{ij}]_{i,j=2}^n$ tudi pozitivno definitna in $a_{11} > 0$.*

Dokaz: Matriko A zapišimo v bločni obliki

$$A = \begin{bmatrix} a_{11} & a \\ b & A_* \end{bmatrix},$$

kjer je $a = [a_{12}, a_{13}, \ldots, a_{1n}]$ in $b = [a_{21}, a_{31}, \ldots, a_{n1}]^T$. Naj bo $y \in \mathbb{R}^{n-1}$ poljuben neničelen vektor in $x^T = [0, y^T]$. Potem je

$$0 < x^T A x = [0, y^T] \begin{bmatrix} a_{11} & a \\ b & A_* \end{bmatrix} \begin{bmatrix} 0 \\ y \end{bmatrix} = y^T A_* y$$

in A_* je pozitivno definitna. Če pa vzamemo $x = [1, 0, \ldots, 0]^T$, je

$$0 < x^T A x = [1, 0] \begin{bmatrix} a_{11} & a \\ b & A_* \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = a_{11}.$$

Pozitivno definitne simetrične matrike nastopajo pri reševanju najrazličnejših problemov dovolj pogosto, da se splača njihovo simetrijo izkoristiti pri reševanju sistemov linearnih enačb.

Algoritem za izračun elementov zgornje trikotne matrike R izpeljemo najlaže, če enačbo $A = R^T R$ zapišemo v bločni obliki tako, da prvi stolpec in prvo vrstico matrik A in R zapišemo posebej:

$$\begin{bmatrix} \alpha & a^T \\ a & A_* \end{bmatrix} = \begin{bmatrix} \rho & 0 \\ r & R_*^T \end{bmatrix} \cdot \begin{bmatrix} \rho & r^T \\ 0 & R_* \end{bmatrix}. \tag{2.10}$$

Ko to matrično enačbo zapišemo po blokih, dobimo tri enačbe

$$\begin{aligned} \alpha &= \rho^2 \\ a^T &= \rho r^T \\ A_* &= R_*^T R_* + r r^T, \end{aligned}$$

od koder je

$$\begin{aligned} \rho &= \sqrt{\alpha} \\ r^T &= a^T / \rho \\ R_*^T R_* &= A_* - r r^T \end{aligned} \tag{2.11}$$

S pomočjo prvih dveh enačb lahko izračunamo prvo vrstico matrike R, zadnjo enačbo pa si poglejmo podrobneje. Matrika

$$\hat{A}_* = A_* - rr^T = A_* - \frac{aa^T}{\alpha}$$

je simetrična, ker je

$$\hat{A}_*^T = (A_* - rr^T)^T = A_*^T - (rr^T)^T = A_* - rr^T = \hat{A}_*.$$

Lastnost 3. *Matrika $\hat{A}_*$ je pozitivno definitna.*

Dokaz: Pokazati moramo, da je za vsak neničelen vektor y

$$y^T \hat{A}_* y = y^T A_* y - (a^T y)^2/\alpha > 0.$$

Ker je matrika A pozitivno definitna, je za poljubno število η

$$0 < [\eta \quad y^T] \begin{bmatrix} \alpha & a^T \\ a & A_* \end{bmatrix} \begin{bmatrix} \eta \\ y \end{bmatrix} = \alpha\eta^2 + 2\eta a^T y + y^T A_* y.$$

Če si izberemo $\eta = -a^T y/\alpha$, dobimo oceno

$$0 < \alpha\eta^2 + 2\eta a^T y + y^T A_* y = y^T A_* y - (a^T y)^2/\alpha,$$

kar že pomeni, da je matrika $\hat{A}_*$ pozitivno definitna.

Dimenzija kvadratne matrike $\hat{A}_*$ je za 1 manjša od dimenzije prvotne matrike A, zato je matrika R_* v enačbi (2.11) faktor v razcepu Choleskega za matriko $\hat{A}_*$, ki ga lahko izračunamo rekurzivno.

Algoritem 2.6. Razcep Choleskega[a] Naj bo A simetrična pozitivno definitna matrika reda n. Naslednji algoritem izračuna zgornjo trikotno matriko R, da je $A = R^T R$. Ob koncu so elementi matrike R zapisani v zgornjem trikotniku matrike A.

for $k = 1 : n$
  $A(k,k) = \mathbf{sqrt}(A(k,k))$
  for $i = k+1 : n$
    $A(k,i) = A(k,i)/A(k,k)$
  end
  for $i = k+1 : n$
    for $j = i : n$
      $A(i,j) = A(i,j) - A(k,j) * A(k,i)$
    end
  end
end

Tudi ta algoritem zapišimo še v kompaktni obliki s podmatrikami:

for $k = 1 : n$
  $A(k,k) = \mathbf{sqrt}(A(k,k))$
  $A(k,k+1:n) = A(k,k+1:n)/A(k,k)$
  for $i = k+1 : n$
    $A(i,i:n) = A(i,i:n) - A(k,i:n) * A(k,i)$
  end
end

[a] Andre-Louis Cholesky (1875 Montguyon – 1918), francoski oficir in geodet. Metodo, danes poimenovano po njem, je razvil, da bi reševal normalne sisteme enačb, ki nastanejo pri uporabi metode najmanjših kvadratov. Padel je tri mesece pred koncem I. svetovne vojne.

Poglejmo še, koliko aritmetičnih operacij je potrebno za razcep Choleskega z algoritmom 2.6:

$$\sum_{k=1}^{n}\left(\sum_{i=k+1}^{n}1+\sum_{i=k+1}^{n}\left(\sum_{j=i}^{n}2\right)\right)=\sum_{k=1}^{n}\left(n-k+2\sum_{i=k+1}^{n}(n-i+1)\right)$$
$$=\sum_{k=1}^{n}(n-k+(n-k)(n-k+1))=\frac{(n-1)n(2n+5)}{6}\approx\frac{2n^3}{6},$$

poleg tega pa še n kvadratnih korenov. Število množenj in deljenj, potrebnih za razcep simetrične, pozitivno definitne matrike z algoritmom 2.6 je torej približno pol manjše kot za LU razcep z algoritmom 2.4.

Primer 2.6. Izračunajmo rešitev sistema linearnih enačb $A\boldsymbol{x} = \boldsymbol{b}$, kjer sta

$$A = \begin{bmatrix} 2 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & 2 \end{bmatrix}, \quad \boldsymbol{b} = [1, 0, 0, 1]^T.$$

Pri razcepu Choleskega dobimo

$$R = \begin{bmatrix} 1.414213562 & -0.707106781 & 0 & 0 \\ 0 & 1.224744871 & -0.816496581 & 0 \\ 0 & 0 & 1.154700538 & -0.866025404 \\ 0 & 0 & 0 & 1.118033989 \end{bmatrix},$$

po direktnem vstavljanju $R^T\boldsymbol{y} = \boldsymbol{b}$ in obratnem vstavljanju $R\boldsymbol{x} = \boldsymbol{y}$ izračunamo rešitev

$$\boldsymbol{x} = [1, 1, 1, 1]^T.$$

2.7 QR-razcep

Za reševanje sistemov linearnih enačb

$$A\mathbf{x} = \mathbf{b}, \quad A \in \mathbb{R}^{n\times n},\ \mathbf{x}, \mathbf{b} \in \mathbb{R}^n$$

se največkrat uporablja LU-razcep, ki je le varianta Gaussove eliminacije.

Matriko A zapišemo kot produkt spodnjetrikotne matrike L in zgornjetrikotne matrike U: $A = LU$. Tako namesto sistema $A\mathbf{x} = \mathbf{b}$ rešujemo sistem $LU\mathbf{x} = \mathbf{b}$

Najprej rešimo sistem $L\mathbf{y} = \mathbf{b}$, potem še $U\mathbf{x} = \mathbf{y}$.

Če bi matriko A lahko razcepili v produkt ortogonalne matrike Q in zgornje trikotne matrike R, bi morali rešiti sistem $QR\mathbf{x} = \mathbf{b}$. Če to enačbo pomnožimo z $Q^{-1} = Q^T$, moramo rešiti le trikotni sistem $R\mathbf{x} = Q^T\mathbf{b}$.

2.7.1 Gram-Schmidt

Za dano matriko A moramo torej poiskati takšno ortogonalno matriko Q, da bo

$$Q^T A = R$$

zgornjetrikotna matrika.

S pomočjo Gram-Schmidtovega postopka matriko Q konstruiramo po stolpcih iz stolpcev matrike A tako, da je vsak naslednji stolpec v matriki Q ortogonalen na vse prejšnje stolpce. To dosežemo tako, da od vsakega stolpca matrike A odštejemo njegove pravokotne projekcije na že konstruirane stolpce matrike Q.

Vzemimo kvadratno matriko a reda 3 s stolpci $\mathbf{a}$, $\mathbf{b}$ in $\mathbf{c}$. Za smer prvega vektorja sprejmemo $\mathbf{x} := \mathbf{a}$. Druga smer mora biti pravokotna na $\mathbf{x}$, zato od $\mathbf{b}$ odštejemo njegovo projekcijo na $\mathbf{x}$:

$$\mathbf{y} = \mathbf{b} - \frac{\mathbf{x}^T\mathbf{b}}{\mathbf{x}^T\mathbf{x}}\mathbf{x}\,.$$

Tretja smer mora biti pravokotna na prvi dve, zato od $\mathbf{c}$ odštejemo njegovi projekciji na $\mathbf{x}$ in $\mathbf{y}$:

$$\mathbf{z} = \mathbf{c} - \frac{\mathbf{x}^T\mathbf{c}}{\mathbf{x}^T\mathbf{x}}\mathbf{x} - \frac{\mathbf{y}^T\mathbf{c}}{\mathbf{y}^T\mathbf{y}}\mathbf{y}\,.$$

Nazadnje vsakega od vektorjev delimo z njegovo dolžino in dobimo ortogonalno matriko Q s stolpci

$$\mathbf{q}_1 = \mathbf{x}/||\mathbf{x}||, \quad \mathbf{q}_2 = \mathbf{y}/||\mathbf{y}||, \quad \mathbf{q}_3 = \mathbf{z}/||\mathbf{z}||$$

Ideja Gram-Schmidtovega procesa je torej preprosta: od vsakega novega vektorja odštejemo njegove projekcije na že določene smeri. Na koncu (ali pa že sproti) vsakega od vektorjev delimo z njegovo dolžino. Tako dobljeni vektorji so ortonormalni.

Primer 2.7. Za vektorje $\mathbf{a} = [1, -1, 0]^T$, $\mathbf{b} = [2, 0, -2]^T$ in $\mathbf{c} = [3, -3, 3]^T$ izračunajmo z Gram-Schmidtovim postopkom ustrezne ortonormirane vektorje.
Ortogonalne vektorje $\mathbf{x}$, $\mathbf{y}$ in $\mathbf{z}$ določimo kot

$$\begin{aligned}
\mathbf{x} &= \mathbf{a} &&= \begin{bmatrix} 1 \\ -1 \\ 0 \end{bmatrix} \\
\mathbf{y} &= \mathbf{b} - \tfrac{\mathbf{x}^T\mathbf{b}}{\mathbf{x}^T\mathbf{x}}\mathbf{x} &&= \begin{bmatrix} 2 \\ 0 \\ -2 \end{bmatrix} - \frac{2}{2}\begin{bmatrix} 1 \\ -1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ -2 \end{bmatrix} \\
\mathbf{z} &= \mathbf{c} - \tfrac{\mathbf{x}^T\mathbf{c}}{\mathbf{x}^T\mathbf{x}}\mathbf{x} - \tfrac{\mathbf{y}^T\mathbf{c}}{\mathbf{y}^T\mathbf{y}}\mathbf{y} &&= \begin{bmatrix} 3 \\ -3 \\ 3 \end{bmatrix} - \frac{6}{2}\begin{bmatrix} 1 \\ -1 \\ 0 \end{bmatrix} - \frac{-6}{6}\begin{bmatrix} 1 \\ 1 \\ -2 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}.
\end{aligned}$$

Končno še vse tri vektorje normiramo, da dobimo ortonormirane vektorje

$$\begin{aligned}
\mathbf{q}_1 &= \tfrac{\mathbf{x}}{||\mathbf{x}||} &&= \frac{1}{\sqrt{2}}\begin{bmatrix} 1 \\ -1 \\ 0 \end{bmatrix} \\
\mathbf{q}_2 &= \tfrac{\mathbf{y}}{||\mathbf{y}||} &&= \frac{1}{\sqrt{6}}\begin{bmatrix} 1 \\ 1 \\ -2 \end{bmatrix} \\
\mathbf{q}_3 &= \tfrac{\mathbf{z}}{||\mathbf{z}||} &&= \frac{1}{\sqrt{3}}\begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}.
\end{aligned}$$

Zapišimo algoritem:

Algoritem 2.7. Gram-Schmidtova QR faktorizacija Za dano matriko A reda $m \times n$ algoritem izračuna matriko Q, prav tako reda $m \times n$ z ortonormalnimi stolpci in zgornjetrikotno matriko R reda $n \times n$, da velja $A = QR$.

$R(1,1) = ||A(:,1)||_2$
$Q(:,1) = A(:,1)/R(1,1)$
for $i = 2 : n$
    $R(1 : i-1, i) = Q(:, 1 : i-1)' * A(:, i)$
    $z = A(:, i) - Q(:, 1 : i-1)' * R(1 : i-1, i)$
    $R(i, i) = ||z||_2$
    $Q(:, i) = z/R(i, i)$
end

Imamo matriko A (n vrstic) in trije stolpci $\mathbf{a}$, $\mathbf{b}$ in $\mathbf{c}$. Ko na stolpcih naredimo Gram-Schmidtov postopek, dobimo matriko Q (n vrstic) z ortonormiranimi stolpci $\mathbf{q}_1$, $\mathbf{q}_2$ in $\mathbf{q}_3$. Kako sta ti dve matriki povezani?

Ker so vsi trije vektorji q linearne kombinacije stolpcev matrike A, obstaja taka matrika R, da je $A = QR$. Kaksna je matrika R?

Ker je

1. $\mathbf{q}_1$ v smeri $\mathbf{a}$,
2. $\mathbf{q}_2$ v ravnini, ki jo določata $\mathbf{a}$ in $\mathbf{b}$ in
3. $\mathbf{q}_3$ v podprostoru, ki ga razpenjajo $\mathbf{a}$, $\mathbf{b}$ in $\mathbf{c}$,

je matrika R trikotna.

Kaj so elementi zgornje trikotne matrike R?

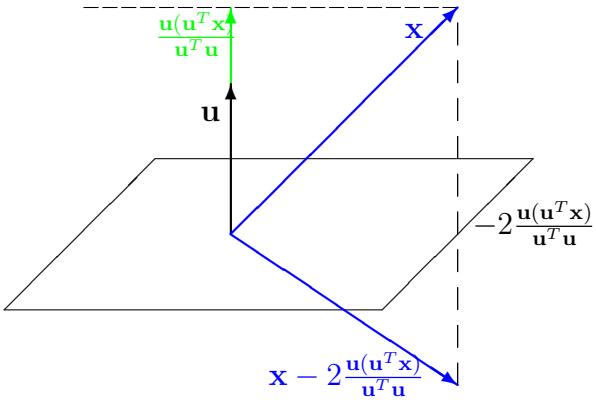
Enačbo $A = QR$ z leve pomnožimo z Q^T. Ker ima Q ortogonalne stolpce, je $Q^TQ = I$, zato je $Q^TA = R$, kar pomeni, da enačba $A = QR$ izgleda kot

$$\begin{bmatrix} \vdots & \vdots & \vdots \\ \mathbf{a} & \mathbf{b} & \mathbf{c} \\ \vdots & \vdots & \vdots \end{bmatrix} = \begin{bmatrix} \vdots & \vdots & \vdots \\ \mathbf{q}_1 & \mathbf{q}_2 & \mathbf{q}_3 \\ \vdots & \vdots & \vdots \end{bmatrix} \begin{bmatrix} \mathbf{q}_1^T\mathbf{a} & \mathbf{q}_1^T\mathbf{b} & \mathbf{q}_1^T\mathbf{c} \\ & \mathbf{q}_2^T\mathbf{b} & \mathbf{q}_3^T\mathbf{c} \\ & & \mathbf{q}_3^T\mathbf{c} \end{bmatrix}.$$

Elementi matrike R so torej skalarni produkti stolpcev matrike Q s stolpci matrike A.

Žal Gram-Schmidtov postopek zaradi problemov s stabilnostjo ni primeren za reševanje večjih sistemov linearnih enačb.

2.7.2 Matrika elementarnih zrcaljenj



Slika 2.2: Vektor $\mathbf{x}$ zrcalimo prek ravnine, pravokotne na vektor $\mathbf{u}$.

Množenje z matriko

$$H = I - 2\frac{\mathbf{u}\mathbf{u}^T}{\mathbf{u}^T\mathbf{u}}$$

vektor $\mathbf{x}$ prezrcali prek (hiper)ravnine skozi koordinatno izhodišče, ki je pravokotna na vektor $\mathbf{u}$.

Veljajo naslednje lastnosti, kar se lahko prepriča bralec sam:

Lastnost 4.

1. *H je ortogonalna matrika: $H^T H = I$;*
2. *$H^2 = I$: Vektor $H\mathbf{x}$ dobimo z zrcaljenjem vektorja $\mathbf{x}$ preko (hiper)ravnine. $H^2\mathbf{x} = H(H\mathbf{x})$ pa ga prezrcali nazaj v $\mathbf{x}$;*
3. *$H\mathbf{y} = \mathbf{y}$ za vsak vektor $\mathbf{y}$ iz (hiper)ravnine (to pomeni, da je $\mathbf{y}^T\mathbf{u} = 0$);*
4. *Matrika H ima enostavno lastno vrednost -1 z lastnim vektorjem $\mathbf{u}$ in $n-1$-kratno lastno vrednost 1 z $n-1$ linearno neodvisnimi lastnimi vektorji iz (hiper)ravnine;*
5. $\det(H) = -1$;
6. *$||H\mathbf{x}||_2 = ||\mathbf{x}||_2$: zrcaljenje ne spreminja (evklidske) dolžine vektorja.*

Pomen matrike zrcaljenja H (Householderjeve[3] matrike) je v tem, da z njo lahko v vektorju z n komponentami pridelamo $n-1$ ničel.

Izrek 2.4. *Za vsak neničelni vektor $\mathbf{x} \neq \mathbf{e}_1$ obstaja taka Householderjeva matrika H, da je $H\mathbf{x}$ kolinearen z $\mathbf{e}_1$.*

Dokaz: Vektor $\mathbf{u}$ sestavimo kot

$$\mathbf{u} = \mathbf{x} + \operatorname{sign}(x_1)||\mathbf{x}||_2\mathbf{e}_1 = [x_1 + \operatorname{sign}(x_1)||x||_2,\ x_2\ \ldots,\ x_n]^T$$

in naj bo

$$H = I - 2\frac{\mathbf{u}\mathbf{u}^T}{\mathbf{u}^T\mathbf{u}}\,.$$

Z direktnim izračunom se prepričamo, da je $H\mathbf{x} = [-\operatorname{sign}(x_1)||\mathbf{x}||_2,\ 0\ \ldots,\ 0]^T$.

[3] Alston Scott Householder (1904–1993), ameriški matematik

Primer 2.8. Naj bo $\mathbf{x} = [2,\ 1,\ 2]^T$. Iščemo tak vektor $\mathbf{u}$, da bo $H\mathbf{x} = [\sigma,\ 0,\ 0]^T$, kjer je $H = I - 2\frac{\mathbf{u}\mathbf{u}^T}{\mathbf{u}^T\mathbf{u}}$.
Imamo $||\mathbf{u}||_2 = 3$, zato je $\mathbf{u} = [2+3, 1, 2]$.
Izračunamo najprej $\mathbf{u}^T\mathbf{u} = 25+1+4 = 30$ in $\mathbf{u}^T\mathbf{x} = 10+1+4 = 15$, zato je

$$H\mathbf{x} = \mathbf{x} - 2\frac{\mathbf{u}}{\mathbf{u}^T\mathbf{u}}\mathbf{u}^T\mathbf{x} = \mathbf{x} - 2\mathbf{u}\frac{15}{30} = \mathbf{x} - \mathbf{u} = [-3,\ 0,\ 0]^T = -3\mathbf{e}_1\,.$$

2.7.3 QR razcep matrike

Dana je matrika A. Naša naloga je s pomočjo Householderjevih matrik izračunati ortogonalno matriko Q in zgornjetrikotno matriko R, da bo $A = QR$.

1. korak: Konstruiramo tak vektor $\mathbf{u}_1$ in s tem implicitno določimo Householderjevo matriko $H_1 = I - 2\frac{\mathbf{u}_1\mathbf{u}_1^T}{\mathbf{u}_1^T\mathbf{u}_1}$, da bo imela $A_1 := H_1A$ v prvem stolpcu pod glavno diagonalo same ničle.

2. korak: Konstruiramo tako Householderjevo matriko H_2, da bo imela $A_2 := H_2A_1$ v drugem stolpcu pod glavno diagonalo same ničle. Pri tem moramo ohraniti ničle, ki smo jih pridelali v prejšnjem koraku.

Matriko Najprej določimo vektor $\mathbf{u}_2 \in \mathbb{R}^{n-1}$, da bo matrika $\hat{H}_2 = I_{n-1} - 2\mathbf{u}_2\mathbf{u}_2^T/\mathbf{u}_2^T\mathbf{u}_2$ reda $n-1$ tista, ki drugi stolpec matrike A_2 pod diagonalo napolni z $n-2$ ničlami. Matrika H_2 je

$$H_2 = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & & & \\ \vdots & & \hat{H}_2 & \\ 0 & & & \end{bmatrix}.$$

k-ti korak: Najprej sestavimo vektor $\mathbf{u}_k \in \mathbb{R}^{n-k+1}$, da bo Householderjeva matrika $\hat{H}_k = I_{n-k+1} - 2\mathbf{u}_k\mathbf{u}_k^T/\mathbf{u}_k^T\mathbf{u}_k$, pri množenju s A_{k-1} k-ti stolpec pod glavno diagonalo napolnila s samimi ničlami. Matriko H_k potem sestavimo kot

$$H_k = \begin{bmatrix} I_{k-1} & 0 \\ 0 & \hat{H}_k \end{bmatrix}$$

Tako ohranimo ničle, ki smo jih pridelali v prejšnjih korakih.

Matrika A_{n-1}, ki jo dobimo po $n-1$-vem koraku, je zgornje trikotna matrika R:

$$H_{n-1} \cdots H_2 H_1 A = R.$$

Produkt ortogonalnih matrik $H_{n-1} \cdots H_2 H_1$ je spet ortogonalna matrika, zapišimo jo z Q^T:

$$Q^T A = R \quad \text{ali} \quad A = QR.$$

Tako smo s pomočjo Householderjevih matrik prvotno matriko A v $n-1$ koraku preoblikovali v zgornjetrikotno. Za razliko od Gaussove eliminacije se transformacija na trikotno obliko s Householderjevimi matrikami vedno uspešno konča.

Praktične pripombe

1. Vsaka Householderjeva matrika H_i je natanko določena z vektorjem $\mathbf{u}_i$, zato je dovolj, če shranimo le vektorje $\mathbf{u}_i$.

2. Vektor $\mathbf{u}_{n-k+1}$, ki določa matriko H_k ima $n-k+1$ komponento, naredi pa $n-k$ ničel v matriki A. Zato lahko zadnjih $n-k$ komponent vektorja $\mathbf{u}$ spravimo na mesta $(k+1,k)$ do (n,k) matrike A (vektor $\mathbf{u}$ razen prvega elementa tako ali tako vsebuje elemente, ki so že na tistih mestih), prvi element pa spravimo poseben vektor.

3. Matrik Q in H_k nikoli ne formiramo eksplicitno. Mnogo bolj ekonomično je, da računamo kar direktno z vektorji $\mathbf{u}_i$.

Zapišimo še algoritem za izračun QR razcepa kvadratne matrike A:

Algoritem 2.8. Householderjeva QR faktorizacija Naj bo A kvadratna matrika reda n. Naslednji algoritem izračuna Householderjeve matrike $H_1, H_2, \ldots, H_{n-1}$ in zgornje trikotno matriko R, da je $A = QR$, kjer je $Q = H_1 H_2 \cdots H_{n-1}$. Ko je algoritem končan, so elementi $u_{k+1,k}$ do $u_{n,k}$ vektorja $\mathbf{u}_k$, ki določa matriko H_k, so zapisani na pozicijah $(k+1, k)$ do (n, k) v spodnjem trikotniku matrike A. Komponente $u_{k,k}$ se nahajajo v vektorju $\mathbf{s} = [u_{1,1}, u_{2,2}, \ldots, u_{n-1,n-1}, s_n]$, matrika R pa v zgornjem trikotniku matrike A.

for $k = 1 : n-1$
$s(k) = sign(A(k,k)) * norm(A(k:n,k))$ % Poiščemo vektor $\mathbf{u}_k$
$u = [s(k) + A(k,k); A(k+1:n,k)]$
$A(k,k) = -s(k)$
% Popravimo elemente matrike A v vrsticah k do n in stolpcih k+1 do n
$A(k:n, k+1:n) = A(k:n, k+1:n) - 2u * \frac{u^T * A(k:n,k+1:n)}{u^T * u}$
end

Za izračun zgornje trikotne matrike R s te algoritmo potrebujemo približno $2n^3/3$ aritmetičnih operacij. Pri tem nismo šteli operacij, ki bi jih potrebovali za izračun matrike Q. Za rešitev sistema enačb matrike Q ne potrebujemo eksplicitno, dovolj je, da poznamo vektorje $\mathbf{u}_k$, ki implicitno določajo matriko Q. Če pa bi matriko Q potrebovali eksplicitno, jo lahko izračunamo iz vektorjev $\mathbf{u}_k$ s približno $2n^3/3$ aritmetičnimi operacijami.

2.8 Napaka in ostanek

Zaradi zaokrožitvenih napak je vsaka izračunana rešitev sistema linearnih enačb le približek k pravi rešitvi. V tem razdelku si bomo ogledali, kako lahko ocenimo napako izračunane rešitve.

Naj bo $\boldsymbol{x}$ točna in $\hat{\boldsymbol{x}}$ izračunana rešitev linearnega sistema $A\boldsymbol{x} = \boldsymbol{b}$. *Napaka* $\boldsymbol{e}$ izračunane rešitve je torej enaka

$$\boldsymbol{e} = \hat{\boldsymbol{x}} - \boldsymbol{x}. \tag{2.12}$$

Ker je napaka vektor, potrebujemo primerno mero za ocenitev njegove velikosti. Tako kot 'velikost' realnega števila določa njegova absolutna vrednost, določa 'velikost' vektorja njegova *norma*.

Definicija 2.2. Norma vektorja (*zapišemo jo s simbolom* $||\ .\ ||$) *je vsaka funkcija z definicijskim območjem v vektorskem prostoru in zalogo vrednosti v realnih številih, ki zadošča naslednjim zahtevam*[4]*:*

1. $||\boldsymbol{x}|| \geq 0$ *za vsak vektor* $\boldsymbol{x}$ *in* $||\boldsymbol{x}|| = 0$ *le za* $\boldsymbol{x} = \boldsymbol{0}$.
2. $||\alpha\boldsymbol{x}|| = |\alpha| \cdot ||\boldsymbol{x}||$ *za vsak skalar* α *in vsak vektor* $\boldsymbol{x}$.
3. $||\boldsymbol{x} + \boldsymbol{y}|| \leq ||\boldsymbol{x}|| + ||\boldsymbol{y}||$ *za vsak par vektorjev* $\boldsymbol{x}$ *in* $\boldsymbol{y}$ *istih dimenzij.*

Prva lastnost zahteva, da je norma neničelnih vektorjev pozitivna. Druga lastnost zahteva, da imata vektorja $\boldsymbol{x}$ in $-\boldsymbol{x}$ isto normo in da se norma vektorja pomnoži z absolutno vrednostjo skalarja, če pomnožimo vektor s skalarjem. Tretja lastnost norme je poznana pod imenom *trikotniška neenakost*, ker zahteva, da mora biti dolžina vsake stranice trikotnika manjša od vsote dolžin ostalih dveh stranic.

Iz linearne algebre je znana *evklidska norma*

$$||\boldsymbol{x}||_2 = \sqrt{\sum_{i=1}^{n} (x_i)^2},$$

pri numeričnem računanju pa največkrat uporabljamo tako imenovano *maksimum* normo

$$||\boldsymbol{x}||_\infty = \max_{1 \leq i \leq n} |x_i|.$$

Pogosto uporabljamo tudi *taksi normo*

$$||\boldsymbol{x}||_1 = \sum_{i=1}^{n} |x_i|\ .$$

Lahko se je prepričati, da tudi za maksimum normo in taksi normo veljajo vse tri lastnosti norme.

[4]Tem zahtevam ustrezajo tudi absolutne vrednosti realnih in kompleksnih števil ter dolžine vektorjev

Primer 2.9. Za vektor $\boldsymbol{a} = [1, 2, 3]^{\mathrm{T}}$ je

$$\begin{aligned} ||\boldsymbol{a}||_1 &= 1 + 2 + 3 = 6, \\ ||\boldsymbol{a}||_2 &= \sqrt{1 + 4 + 9} = \sqrt{14} \approx 3.74166 \quad \text{in} \\ ||\boldsymbol{a}||_\infty &= 3. \end{aligned}$$

Podobno kot pri vektorjih, lahko tudi 'velikost' matrik merimo z *matrično* normo.

Definicija 2.3. *Matrična norma je vsaka funkcija iz prostora kvadratnih matrik v realna števila, ki zadošča naslednjim lastnostim:*

1. *Za vsako kvadratno matriko A je $||A|| \geq 0$ in $||A|| = 0$ le za ničelno matriko $A = 0$.*
2. *$||\alpha A|| = |\alpha| \cdot ||A||$ za vsako kvadratno matriko A in vsak skalar α.*
3. *$||A + B|| \leq ||A|| + ||B||$ za vsak par kvadratnih matrik A in B istih dimenzij.*
4. *$||AB|| \leq ||A|| \cdot ||B||$ za vsak par kvadratnih matrik A in B istih dimenzij.*

Iz vsake vektorske norme lahko dobimo ustrezno matrično normo s predpisom

$$||A|| = \max_{\boldsymbol{x} \neq 0} \frac{||A\boldsymbol{x}||}{||\boldsymbol{x}||}, \tag{2.13}$$

tako da velja ocena

$$||A\boldsymbol{x}|| \leq ||A|| \cdot ||\boldsymbol{x}||.$$

Na ta način je matrična maksimum norma definirana s pomočjo vektorske maksimum norme [3]

$$||A||_\infty = \max_{\boldsymbol{x} \neq 0} \frac{||A\boldsymbol{x}||_\infty}{||\boldsymbol{x}||_\infty},$$

in je enaka

$$||A||_\infty = \max_{1 \leq i \leq n} \sum_{j=1}^{n} |a_{ij}|.$$

Podobno lahko je izračunljiva tudi matrična norma, definirana s pomočjo vektorske taksi norme

$$||A||_1 = \max_{1\leq j\leq n} \sum_{i=1}^{n} |a_{ij}|\,,$$

medtem ko je matrično normo, definirano s pomočjo vektorske evklidske norme

$$||A||_2 = \max_{\boldsymbol{x}\neq 0} \frac{||A\boldsymbol{x}||_2}{||\boldsymbol{x}||_2}$$

običajno precej teže izračunati. Pokazati se da, da je $||A||_2 = \sqrt{|\lambda_{max}|}$, kjer je λ_{max} tista lastna vrednost matrike A^TA, ki ima največjo absolutno vrednost.

Primer 2.10. Za matriko

$$A = \begin{bmatrix} 4 & -6 & 2 \\ 0 & 4 & 1 \\ 1 & 3 & 3 \end{bmatrix}$$

je $||A||_\infty = \max\{12, 5, 6\} = 12$ in $||A||_1 = \max\{5, 13, 6\} = 13$, medtem ko je $||A||_2 \approx 8.3427$.

Vrnimo se k obravnavi ocen za napako (2.12) rešitve sistema linearnih enačb. Če izračunano rešitev vstavimo v sistem (2.2), dobimo *ostanek* $\boldsymbol{r} = A\hat{\boldsymbol{x}} - \boldsymbol{b}$, za katerega velja

$$\boldsymbol{r} = A\hat{\boldsymbol{x}} - A\boldsymbol{x} = A(\hat{\boldsymbol{x}} - \boldsymbol{x}) = A\boldsymbol{e}, \tag{2.14}$$

oziroma $\boldsymbol{e} = A^{-1}\boldsymbol{r}$, od koder hitro dobimo oceno za napako izračunane rešitve

$$||\boldsymbol{e}|| = ||A^{-1}\boldsymbol{r}|| \leq ||A^{-1}|| \cdot ||\boldsymbol{r}||. \tag{2.15}$$

Podobno oceno za relativno napako $||\boldsymbol{e}||/||\boldsymbol{x}||$ dobimo tako, da enačbo (2.15) delimo z $||\boldsymbol{x}||$ in upoštevamo, da je $||A|| \cdot ||\boldsymbol{x}|| \geq ||\boldsymbol{b}||$:

$$\frac{||\boldsymbol{e}||}{||\boldsymbol{x}||} \leq ||A^{-1}|| \cdot ||A|| \frac{||\boldsymbol{r}||}{||\boldsymbol{b}||}, \tag{2.16}$$

Ta ocena velja za poljubno vektorsko normo in z njeno pomočjo definirano matrično normo. Povejmo še z besedami: relativna napaka je kvečjemu za faktor $||A^{-1}|| \cdot ||A||$ večja od relativnega ostanka. Število $||A^{-1}|| \cdot ||A||$ je

značilno za posamezno matriko. Imenujemo ga *število občutljivosti* (angl. *condition number*) in ga zapišemo

$$\text{cond}(A) = ||A^{-1}|| \cdot ||A||,$$

če pa je matrika A singularna (in A^{-1} ne obstaja), tedaj je $\text{cond}(A) = \infty$.

Zaradi lastnosti matrične norme je $||I|| = ||A^{-1}A|| \leq ||A^{-1}|| \cdot ||A||$, po drugi strani pa je, za matrične norme, ki so definirane s pomočjo vektorske norme, $||I|| = \max ||A\boldsymbol{x}||/||\boldsymbol{x}|| = 1$, zato je število občutljivosti $\text{cond}(A) \geq 1$ za vsako matriko A. Matrikam, katerih število občutljivosti je majhno (to pomeni: ne dosti večje od 1), pravimo *dobro pogojene* matrike. Nasprotno so *slabo pogojene* tiste matrike, katerih število občutljivosti je veliko.

Ocena (2.16) pomeni, da je pri dobro pogojenih matrikah ostanek dobra ocena za napako v rešitvi, pri slabo pogojenih matrikah pa iz majhnega ostanka ne moremo sklepati, da je tudi napaka majhna.

Da bi lahko izračunali število občutljivosti neke matrike A, bi morali poznati njeno inverzno matriko A^{-1}, kar pa je navadno še težja naloga kot rešiti sistem linearnih enačb.

Število občutljivosti je obratno sorazmerno oddaljenosti matrike od najbljižje singularne matrike:

Izrek 2.5. *[10] Naj bo $A \in \mathbb{R}^{n\times n}$ nesingularna matrika. Za vsako singularno matriko $B \in \mathbb{R}^{n\times n}$ velja*

$$\frac{1}{\text{cond}(A)} \leq \frac{||A - B||}{||A||}.$$

Dokaz: Zaradi definicije matrične norme (2.13) je

$$||A^{-1}|| = \max_{\boldsymbol{x}\neq 0} \frac{||A^{-1}\boldsymbol{x}||}{||\boldsymbol{x}||} \geq \frac{||A^{-1}\boldsymbol{x}||}{||\boldsymbol{x}||}$$

za vsak vektor $\boldsymbol{x} \in \mathbb{R}^n$, torej tudi

$$||A^{-1}|| \geq \frac{||\boldsymbol{y}||}{||A\boldsymbol{y}||}$$

za vsak vektor $\boldsymbol{y} \in \mathbb{R}^n$. Od tod sklepamo, da tudi ocena

$$\frac{1}{\mathrm{cond}(A)} \leq \frac{1}{||A||}\frac{||A\boldsymbol{y}||}{||\boldsymbol{y}||}$$

velja za vsak vektor $\boldsymbol{y}$. Izberimo neničelen vektor $\boldsymbol{y}$ tako, da bo $B\boldsymbol{y} = 0$. Tak vektor zagotovo obstaja, ker je matrika B singularna. Tako velja

$$\frac{1}{\mathrm{cond}(A)} \leq \frac{||(A-B)\boldsymbol{y}||}{||A||\cdot||\boldsymbol{y}||} \leq \frac{||A-B||\cdot||\boldsymbol{y}||}{||A||\cdot||\boldsymbol{y}||} = \frac{||A-B||}{||A||}.$$

S pomočjo izreka 2.5 lahko ugotovimo, kako daleč je matrika A od najbližje singularne matrike: če je $\mathrm{cond}(A)$ veliko število, je A skoraj singularna. Zato pri reševanju sistema linearnih enačb z zelo občutljivo matriko lahko dobimo velike napake.

Tudi brez poznavanja števila občutljivosti lahko ugotovimo, ali je sistem slabo ali dobro pogojen, ob tem pa še izboljšamo točnost izračunane rešitve s tehniko *iterativnega izboljševanja*.

Vzemimo, da imamo približno rešitev $\hat{\boldsymbol{x}}$ sistema (2.2), in da smo izračunali ostanek $\boldsymbol{r}^1 = A\hat{\boldsymbol{x}} - \boldsymbol{b}$. Vemo, da v tem primeru napaka izračunane rešitve zadošča enačbi (2.14), katere matrika je ista kot matrika prvotnega sistema. Ker smo prvotni sistem že rešili, torej poznamo njegov LU razcep, lahko tudi do rešitve sistema (2.14) pridemo z rešitvijo dveh trikotnih sistemov, kar pomeni le n^2 množenj. Ko tako poznamo napako $\boldsymbol{e}$, lahko 'popravimo' izračunano rešitev, saj je

$$\hat{\boldsymbol{x}}^1 = \hat{\boldsymbol{x}} - \boldsymbol{e}$$

boljši približek za rešitev kot $\hat{\boldsymbol{x}}$. Na ta način lahko nadaljujemo, da dobimo zaporedje rešitev $\hat{\boldsymbol{x}}$, $\hat{\boldsymbol{x}}^1$, $\hat{\boldsymbol{x}}^2, \ldots$, kot tudi zaporedje ustreznih ostankov

$\hat{\boldsymbol{r}}$, $\hat{\boldsymbol{r}}^1$, $\hat{\boldsymbol{r}}^2, \ldots$. Iz števila pravilnih mest približkov $\hat{\boldsymbol{x}}^i$, kot tudi iz konvergence zaporedja ostankov lahko sklepamo na natančnost teh približkov rešitve. Kadar je število $\mathrm{cond}(A)$ veliko, zaporedje rešitev počasneje ali sploh ne konvergira proti pravi rešitvi. Tako lahko sklepamo, da je sistem zelo slabo pogojen, kadar zaporedje ostankov le počasi ali pa sploh ne konvergira proti 0.

Pozor! Posebej velja opozoriti, da moramo pri iterativnem izboljševanju rešitve ostanke $\boldsymbol{r}^i$ računati čim bolj natančno, navadno v dvojni natančnosti.

Algoritem 2.9. Iterativno izboljšanje Naj bo dan linearen sistem $A\boldsymbol{x} = \boldsymbol{b}$, LU razcep matrike A in približna rešitev $\hat{\boldsymbol{x}}$. Naslednji algoritem izračuna boljši približek za rešitev:

Izračunaj ostanek $\boldsymbol{r} = A\hat{\boldsymbol{x}} - \boldsymbol{b}$ v dvojni natančnosti
Reši sistem $A\boldsymbol{e} = \boldsymbol{r}$ z uporabo algoritmov z direktnim in obratnim vstavljanjem
$\hat{\boldsymbol{x}} = \hat{\boldsymbol{x}} - \boldsymbol{e}$
Če je $||\boldsymbol{e}||/||\hat{\boldsymbol{x}}||$ dovolj majhno število, naj bo $\hat{\boldsymbol{x}}$ rešitev, sicer ponovi postopek

Če smo že izračunali rešitev $\hat{\boldsymbol{x}}$ s pomočjo LU razcepa matrike A, se skoraj vedno izplača naknadno iterativno izboljšanje rešitve, saj zanjo za vsak korak iteracije potrebujemo le rešitev dveh trikotnih sistemov. Iz konvergence iterativnega izboljšanja pa lahko sklepamo na občutljivost sistema.

S številom občutljivosti $\mathrm{cond}(A)$ lahko tudi ocenimo, kako je rešitev linearnega sistema odvisna od spremembe desne strani $\boldsymbol{b}$.

Izrek 2.6. *Naj bo matrika A nesingularna. Za točne rešitve sistemov $A\boldsymbol{x} = \boldsymbol{b}$ in $A\hat{\boldsymbol{x}} = \hat{\boldsymbol{b}}$ velja ocena*

$$\frac{||\boldsymbol{x} - \hat{\boldsymbol{x}}||}{||\boldsymbol{x}||} \leq \mathrm{cond}(A) \frac{||\boldsymbol{b} - \hat{\boldsymbol{b}}||}{||\boldsymbol{b}||}. \tag{2.17}$$

Dokaz: Odštejmo $A\hat{\boldsymbol{x}} = \hat{\boldsymbol{b}}$ od $A\boldsymbol{x} = \boldsymbol{b}$

$$A(\boldsymbol{x} - \hat{\boldsymbol{x}}) = \boldsymbol{b} - \hat{\boldsymbol{b}},$$

oziroma

$$\boldsymbol{x} - \hat{\boldsymbol{x}} = A^{-1}(\boldsymbol{b} - \hat{\boldsymbol{b}}).$$

Zaradi zveze med vektorsko in matrično normo je

$$||\boldsymbol{x} - \hat{\boldsymbol{x}}|| = ||A^{-1}(\boldsymbol{b} - \hat{\boldsymbol{b}})|| \leq ||A^{-1}||\,||\boldsymbol{b} - \hat{\boldsymbol{b}}||.$$

Delimo obe strani neenačbe z $||\boldsymbol{x}||$

$$\frac{||\boldsymbol{x} - \hat{\boldsymbol{x}}||}{||\boldsymbol{x}||} \leq \frac{||A^{-1}||\,||\boldsymbol{b} - \hat{\boldsymbol{b}}||}{||\boldsymbol{x}||} = \frac{||A||\,||A^{-1}||\,||\boldsymbol{b} - \hat{\boldsymbol{b}}||}{||A||\,||\boldsymbol{x}||}. \tag{2.18}$$

Ker, spet zaradi zveze med vektorsko in matrično normo, velja tudi

$$||\boldsymbol{b}|| = ||A\boldsymbol{x}|| \leq ||A||\,||\boldsymbol{x}||,$$

kar uporabimo na desni strani neenačbe (2.18), in dobimo oceno (2.17).

Na tem mestu le omenimo, da lahko s pomočjo LU razcepa izračunamo tudi determinanto kvadratne matrike in inverzno matriko. Iz razcepa $A = LU$ je očitno, da je $\det A = \det L \cdot \det U$, ker pa je $\det L = 1$ (matrika L je trikotna z enojkami na diagonali), je

$$\det A = \prod_{i=1}^{n} u_{ii},$$

saj je tudi U trikotna matrika.

Inverzna matrika A^{-1} je rešitev matrične enačbe $AA^{-1} = I$, ki jo lahko zapišemo po stolpcih kot zaporedje linearnih sistemov z isto matriko A:

$$A\boldsymbol{x}^{(j)} = \boldsymbol{e}^{(j)}; \quad j = 1, \ldots, n,$$

kjer so $\boldsymbol{x}^{(j)}$ stolpci inverzne matrike A^{-1}, $\boldsymbol{e}^{(j)}$ pa ustrezni stolpci enotske matrike I. Opozoriti velja, da inverzno matriko le redkokdaj potrebujemo, saj je praviloma učinkoviteje rešiti ustrezen sistem linearnih enačb. Tako namesto računanja produkta $C = A^{-1}B$ raje rešujemo matrično enačbo $AC = B$.

2.9 Iterativne metode

Pri reševanju nekaterih problemov, predvsem pri reševanju parcialnih diferencialnih enačb, naletimo na linearne sisteme z zelo velikim številom neznank (tudi $n = 10000$ ali več). Običajno ima pri teh sistemih matrika A le nekaj elementov v vsaki vrstici različnih od 0 (taki matriki pravimo *razpršena matrika* (angl. *sparse matrix*) za razliko od *goste matrike* (angl. *dense matrix*), pri kateri je večina elementov različna od 0). Gaussova eliminacijska metoda za velike razpršene sisteme ni primerna, ker zahteva preveč računanja (spomnimo se: za rešitev sistema reda n potrebujemo približno $n^3/3$ množenj in deljenj), poleg tega pa se med računanjem 'napolnijo' tudi tisti elementi, ki so bili v prvotni matriki A enaki 0, kar lahko predstavlja znatne težave pri shranjevanju elementov v pomnilniku računalnika.

Zaradi tega za reševanje zelo velikih razpršenih sistemov raje od direktnih metod, kot je Gaussova eliminacija ali LU razcep, uporabljamo *iteracijske metode*. To so metode, pri katerih tvorimo zaporedje približnih rešitev $\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots$, ki pri določenih pogojih konvergira proti rešitvi prvotnega sistema. Da bo iteracijska metoda učinkovita, mora biti izračun novega približka enostaven, z majhnim številom računskih operacij, zaporedje približkov pa mora dovolj hitro konvergirati proti pravilni rešitvi. Ogledali si bomo dve enostavni iteracijski metodi, Jacobijevo in Gauss-Seidlovo.

Jacobijeva iteracija *Jacobijevo metodo* [5] najlaže izpeljemo tako, da v linearnem sistemu (2.2) i-to enačbo rešimo glede na spremenljivko x_i. Tako dobimo

$$x_i = \frac{1}{a_{ii}} \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j - \sum_{j=i+1}^{n} a_{ij} x_j \right); \quad i = 1, \ldots, n. \tag{2.19}$$

Izberimo si začetni približek k rešitvi $\boldsymbol{x}^{(0)}$ (zgornji indeksi v oklepaju bodo pomenili zaporedno številko iteracije). Novi, izboljšani približek k rešitvi dobimo tako, da prejšnji približek vstavimo v desno stran enačbe (2.19). Tako pridemo do splošnega koraka Jacobijeve iteracijske sheme

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k)} \right); \quad i = 1, \ldots, n, \tag{2.20}$$

[5] Carl Gustav Jacob Jacobi (1804 Potsdam – 1851 Berlin), namški matematik, poznan predvsem po teoriji eliptičnih funkcij in funkcijskih determinantah. Iterativno metodo za reševanje sistemov linearnih enačb je objavil leta 1845 v astronomski reviji.

ki jo lahko nekoliko poenostavimo, če na desni strani prištejemo in odštejemo $x_i^{(k)}$:

$$x_i^{(k+1)} = x_i^{(k)} + \frac{1}{a_{ii}} \left(b_i - \sum_{j=1}^{n} a_{ij} x_j^{(k)} \right); \quad i = 1, \ldots, n. \tag{2.21}$$

Poglejmo si še, kako Jacobijevo iteracijsko shemo zapišemo v matrični obliki. Najprej matriko A zapišimo v obliki $A = S + D + Z$, kjer je matrika S spodnji trikotnik matrike A brez diagonale, D diagonalna matrika, ki vsebuje diagonalne elemente matrike A, matrika Z pa zgornji trikotnik matrike A brez diagonale. Sistem (2.20) lahko sedaj zapišemo kot

$$D\boldsymbol{x}^{(k+1)} = \boldsymbol{b} - (S + Z)\boldsymbol{x}^{(k)}. \tag{2.22}$$

Ker je matrika D diagonalna, je ta sistem lahko rešljiv, če so le vsi diagonalni koeficienti $a_{ii} \neq 0$. Uporabo Jacobijeve metode si oglejmo na primeru:

Primer 2.11. Rešujemo sistem enačb

$$\begin{bmatrix} 2 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix}, \tag{2.23}$$

katerega rešitev je $x_1 = x_2 = x_3 = x_4 = 1$.
Vzemimo začetni približek $x_1 = x_2 = x_3 = x_4 = 0$, naslednje približke pa računamo po formuli (2.20). Iz rezultatov, ki so v tabeli 2.1, vidimo, da napaka s številom iteracij pada in da je po 20 iteracijah približek k rešitvi točen na 2 decimalni mesti, po 50 iteracijah točen na 4 decimalna mesta, po 100 iteracijah pa že na več kot 9 decimalnih mest.

Pogoje za konvergenco Jacobijeve metode je v praksi težko natančno opredeliti. Velja sicer naslednji izrek o konvergenci, ki ga navajamo brez dokaza (glej [11], Theorem 10.1.1):

k	x_1	x_2	x_3	x_4
1	0.50000000	0	0	0.50000000
2	0.50000000	0.25000000	0.25000000	0.50000000
3	0.62500000	0.37500000	0.37500000	0.62500000
4	0.68750000	0.50000000	0.50000000	0.68750000
5	0.75000000	0.59375000	0.59375000	0.75000000
6	0.79687500	0.67187500	0.67187500	0.79687500
⋮				
20	0.99155473	0.98633527	0.98633527	0.99155473
21	0.99316763	0.98894500	0.98894500	0.99316763
22	0.99447250	0.99105632	0.99105632	0.99447250
⋮				
50	0.99998536	0.99997632	0.99997632	0.99998536
51	0.99998816	0.99998084	0.99998084	0.99998816
52	0.99999042	0.99998450	0.99998450	0.99999042
⋮				
100	1.00000000	1.00000000	1.00000000	1.00000000

Tabela 2.1: Približki rešitve enačbe (2.23), izračunani z Jacobijevo iteracijsko metodo

Izrek 2.7. *Zaporedje približkov k rešitvi linearnega sistema* (2.2), izračunano z Jacobijevo metodo (2.22) pri poljubnem začetnem približku $\boldsymbol{x}^{(0)}$ konvergira proti točni rešitvi natanko tedaj, ko vse lastne vrednosti λ matrike $M = D^{-1}(S + Z)$ zadoščajo neenačbi

$$|\lambda| < 1.$$

V praksi je zelo težko preverjati, ali je pogoj tega izreka izpolnjen. Navadno si pomagamo s kriterijem *diagonalne dominantnosti*:

Definicija 2.4. *Matrika A je* diagonalno dominantna po vrsticah, *če je*

$$\sum_{\substack{j=1\\ j\neq i}}^{n} |a_{ij}| < |a_{ii}|; \quad i = 1, \ldots, n.$$

Če je matrika A diagonalno dominantna, potem Jacobijeva metoda konvergira proti točni rešitvi pri vsakem začetnem približku. Navadno je konvergenca hitrejša, čim bolj je sistem diagonalno dominanten.

Gauss-Seidlova iteracija Pri Jacobijevi metodi računamo komponente novega približka $\boldsymbol{x}^{(k+1)}$ iz starega približka $\boldsymbol{x}^{(k)}$. Konvergenco iteracije bi lahko pospešili, če bi v formuli (2.20) na desni strani uporabili tiste komponente novega približka $\boldsymbol{x}^{(k+1)}$, ki smo jih že izračunali. Tako dobimo *Gauss-Seidlovo* [6] iteracijsko shemo

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k)} \right); \quad i = 1, \ldots, n. \tag{2.24}$$

Če matriko A razdelimo na vsoto matrik $A = S + D + Z$, kot pri Jacobijevi metodi, lahko Gauss-Seidlovo iteracijo zapišemo v matrični obliki kot

$$(S + D)\boldsymbol{x}^{(k+1)} = \boldsymbol{b} - Z\boldsymbol{x}^{(k)}. \tag{2.25}$$

Matrika $S + D$ je spodnjetrikotna, zato sistem enačb (2.25) lahko enostavno rešimo z direktnim vstavljanjem.

Na istem primeru si oglejmo še delovanje Gauss-Seidelove metode:

Primer 2.12. Rešujemo linearni sistem (2.23) z Gauss-Seidlovo iteracijsko metodo. Vzemimo začetno rešitev $x_1 = x_2 = x_3 = x_4 = 0$, naslednje približke pa računamo po formuli (2.24). Rezultati so prikazani v tabeli 2.2. Iz rezultatov vidimo, da v tem primeru Gauss-Seidlova iteracija res konvergira hitreje kot Jacobijeva iteracija.

[6] Philipp Ludwig von Seidel (1821 Zweibrücken – 1896 München), nemški matematik in astronom. Iteracijsko metodo, danes znano pod imenom Gauss-Seidel je objavil leta 1874 kot rezultat sodelovanja z Jacobijem. Gauss je podobno metodo opisal že leta 1845.

k	x_1	x_2	x_3	x_4
1	0.50000000	0.25000000	0.12500000	0.56250000
2	0.62500000	0.37500000	0.46875000	0.73437500
3	0.68750000	0.57812500	0.65625000	0.82812500
4	0.78906250	0.72265625	0.77539062	0.88769531
5	0.86132812	0.81835937	0.85302734	0.92651367
6	0.90917968	0.88110351	0.90380859	0.95190429
⋮				
20	0.99975953	0.99968523	0.99974534	0.99987267
21	0.99984261	0.99979398	0.99983332	0.99991666
22	0.99989699	0.99986515	0.99989091	0.99994545
⋮				
50	0.99999997	0.99999996	0.99999997	0.99999998
51	0.99999998	0.99999997	0.99999998	0.99999999
52	0.99999999	0.99999998	0.99999999	0.99999999

Tabela 2.2: Približki rešitve enačbe (2.23), izračunani z Gauss-Seidlovo iteracijsko metodo

Če je matrika sistema diagonalno dominantna, potem zaporedje približkov, izračunano z Gauss-Seidlovo iteracijsko metodo, konvergira proti točni rešitvi vsaj tako hitro, kot zaporedje, izračunano z Jacobijevo metodo, običajno pa hitreje.

Metoda SOR Jacobijeva in Gauss-Seidelova metoda sta v praksi največkrat obupno počasni. Hitrost konvergence Gauss-Seidelove metode pa lahko pogosto močno pospešimo tako, da za novi približek vzamemo primerno uteženo povprečje novega približka, izračunenega po Gauss-Seidelovi metodi, in prejšnjega približka. Tako dobimo *metodo SOR* (iz angl. Successive Over-Relaxation). Izberimo si utež $\omega \in \mathbb{R}$ in

$$x_i^{(k+1)} = \frac{\omega}{a_{ii}} \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k)} \right) + (1-\omega) x_i^j; \quad i = 1, \ldots, n, \tag{2.26}$$

kar lahko v matrični obliki zapišemo kot

$$(\omega S + D)\boldsymbol{x}^{(k+1)} = \omega\boldsymbol{b} + ((1-\omega)D - \omega Z)\boldsymbol{x}^{(k)}. \tag{2.27}$$

k	x_1	x_2	x_3	x_4
1	0.65000000	0.42250000	0.27462500	0.82850625
2	0.72962500	0.52601250	0.79804968	0.92018042
3	0.77302062	0.86339195	0.91990713	0.97188551
4	0.97929858	0.97546613	0.98980642	1.00180852
5	0.99026341	0.99440555	1.00059722	0.99984563
6	0.99928458	1.00160151	1.00076147	1.00054127
⋮				
14	0.99999972	0.99999980	0.99999985	0.99999997
15	0.99999996	0.99999994	0.99999999	1.00000000

Tabela 2.3: Približki rešitve enačbe (2.23), izračunani z iteracijsko metodo SOR ($\omega = 1.3$).

Primer 2.13. Rešujemo isti linearni sistem (2.23) z iteracijsko metodo SOR. Za vrednost relaksacijskega parametra ω bomo izbrali 1.3. Vzemimo začetno rešitev $x_1 = x_2 = x_3 = x_4 = 0$, naslednje približke pa računamo po formuli (2.26). Rezultati so prikazani v tabeli 2.3. Iz rezultatov vidimo, da v tem primeru iteracija SOR res konvergira precej hitreje kot Gauss-Seidelova iteracija.

Ostane še vprašanje, kako izbrati vrednost relaksacijskega parametra ω, od katerega je bistveno odvisna hitrost iteracije. Za vrednosti ω zunaj intervala $(0, 2)$ iteracija SOR zagotovo ne konvergira, v primeru, ko je matrika sistema simetrična in pozitivno definitna, pa iteracija SOR konvergira za vse $0 < \omega < 2$ pri poljubni izbiri začetnega približka $\mathbf{x}^{(0)}$.

Težji problem je določiti vrednost parametra ω tako, da bo iteracija SOR najhitreje konvergirala. Žal splošno pravilo za izbor optimalne vrednosti ω_{opt} ni znano, zato je pri vsakem konkretnem primeru približno vrednost za ω_{opt} večkrat potrebno določiti eksperimentalno.

Metoda konjugiranih gradientov *Metoda konjugiranih gradientov* je uporabna za reševanje sistemov linearnih enačb $A\mathbf{x} = \mathbf{b}$, kjer je matrika A

simetrična in pozitivno definitna (kar pomeni, da so vse njene lastne vrednosti pozitivne).

Metodo sta prva opisala Hestend in Stiefel (1952) in se od tedaj redno uporablja za reševanje zelo velikih, razpršenih simetričnih pozitivno defnitnih sistemov linearnih enačb. Teoretično (brez zaokrožitvenih metod) bi metoda izračunala točno rešitev sistema n enačb z n neznankami v n korakih, v praksi pa je, zaradi zaokrožitvenih napak, iterativna. Njena ideja je v naslednjem rezultatu [8]:

Izrek 2.8. *Za simetrično in pozitivno definitno matriko A je reševanje linearnega sistema*

$$A\mathbf{x} = \mathbf{b}$$

enakovredno iskanju minimuma kvadratne funkcije

$$\varphi(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T A\mathbf{x} - \mathbf{x}^T\mathbf{b}\,.$$

Funkcija $\varphi(\mathbf{x})$ doseže svojo minimalno vrednost $-\frac{1}{2}\mathbf{b}^T A^{-1}\mathbf{b}$ pri

$$\mathbf{x} = A^{-1}\mathbf{b}.$$

Podobne optimizacijske probleme navadno rešujemo tako, da konstruiramo zaporedje vektorjev

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k\mathbf{p}_k$$

tako, da se vrednost kriterijske funkcije $\varphi(\mathbf{x}_k)$ zmanjšuje s k. Zato vektorje $\mathbf{p}_k$ (*smerne vektorje*) izbiramo tako, da kažejo v smer padanja funkcije φ, skalarje α_k pa tako, da v smeri vektorja $\mathbf{p}_k$ minimizirajo funkcijo $\varphi_\alpha(\mathbf{x}_k + \alpha\mathbf{p}_k)$. Da se pokazati [8], da je najprimernejša vrednost

$$\alpha = \alpha_k = \frac{\mathbf{p}_k^T(\mathbf{b} - A\mathbf{x}_k)}{\mathbf{p}_k^T A\mathbf{p}_k} = \frac{\mathbf{p}_k^T\mathbf{r}_k}{\mathbf{p}_k^T A\mathbf{p}_k}\,,$$

kjer je ostanek $\mathbf{r}_k = b - A\mathbf{x}_k$.

Kako pa izberemo smerne vektorje $\mathbf{p}_k$? Pri metodi konjugiranih gradientov jih izbiramo tako, da so med seboj ortogonalni glede na skalarni produkt

$\mathbf{p}^T A\mathbf{p} = 0$, ki ga določa pozitivno defninitna matrika A. Vektorjem, ki so ortogonalni glede na ta skalarni produkt, pravimo *konjugirani vektorji*.

Zapišimo algoritem za metodo konjugiranih gradientov.

Algoritem 2.10. Metoda konjugiranih gradientov Naj bo A simetrična pozitivno definitna matrika reda n. Pri danem začetnem približku $\mathbf{x}_0$ naslednji algoritem izračuna zaporedje približkov $\mathbf{x}_k$ rešitve sistema linearnih enačb $A\mathbf{x} = \mathbf{b}$ tako, da je $||\mathbf{x}_{k+1} - \mathbf{x}||_2 < ||\mathbf{x}_k - \mathbf{x}||_2$ in $||A\mathbf{x}_k - \mathbf{b}||_2 < \epsilon$, kjer je $\mathbf{x}$ točna rešitev sistema.

$r_0 = b - A * x$
$p_0 = r_0$
for $k = 0 : Nmax$
  $w = A * p_k$
  $\alpha_k = (\mathbf{norm}(r_k))^2/(p_k^T * w)$
  $x_{k+1} = x_k + \alpha_k * p_k$
  $r_{k+1} = r_k - \alpha_k * w$
  if $\mathbf{norm}(r_{k+1})^2 < \epsilon$, **break, endif**
  $\beta_k =$**norm**$(r_{k+1})^2/$**norm**$(r_k)^2$
  $p_{k+1} = r_{k+1} + \beta_k * p_k$
end

Poglejmo, kako metoda konjugiranih gradientov deluje na preprostem primeru.

Primer 2.14. Rešujemo sistem enačb

$$\begin{bmatrix} 4 & 1 & 2 \\ 1 & 5 & 3 \\ 2 & 3 & 6 \end{bmatrix} \mathbf{x} = \begin{bmatrix} 7 \\ 9 \\ 11 \end{bmatrix} \tag{2.28}$$

z metodo konjugiranih gradientov. Za začetni približek izberimo kar desno stran sistema $\mathbf{x}_0 = [7\ 9\ 11]$. Zaporedje približkov $\mathbf{x}_k$ in ostankov $\mathbf{r}_k$, ki ga dobimo z metodo konjugiranih gradientov, je v Tabeli 2.4.

2.10 Povzetek

V splošnem delimo numerične metode za reševanje sistemov linearnih enačb na direktne, pri katerih računamo razcep matrike in pridemo do rešitve s

$\mathbf{x}_k$			$\mathbf{r}_k$		
7.0000	9.0000	11.0000	−52.0000	−76.0000	−96.0000
1.4780	0.9293	0.8055	−1.4521	0.4590	0.4232
1.0183	1.0705	0.9345	−0.0128	−0.1743	0.1449
1.0000	1.0000	1.0000	0.0000	0.0000	0.0000

Tabela 2.4: Zaporedje približkov k rešitvi sistema linearnih enačb (2.28)

končnim številom aritmetičnih operacij, in iterativne, pri katerih tvorimo zaporedje približkov, ki pod določenimi pogoji konvergira proti rešitvi.

Od direktnih metod smo spoznali Gaussovo eliminacijsko metodo z njeno varianto, LU razcepom (algoritem 2.5), ki je primernejša za reševanje sistemov, ki imajo pri istih koeficientih različne desne strani, metodo Choleskega (algoritem 2.6) za reševanje simetričnih pozitivno definitnih sistemov in QR razcep (algoritem 2.8), ki je sicer računsko zahtevnejša od LU razcepa, zato pa bolj stabilna. Direktne metode se uporabljajo predvsem pri reševanju manjših sistemov in sistemov, pri katerih je večina koeficientov različna od 0. Pri reševanju sistemov linearnih enačb z direktnimi metodami je praviloma priporočljivo, včasih pa celo potrebno, delno pivotiranje.

Včasih lahko pri reševanju linearnih sistemov s pridom izkoristimo posebno obliko matrike koeficientov. Oglejmo si dva značilna primera:

- *Po stolpcih diagonalno dominantne matrike,* to so matrike, pri katerih je
$$|a_{ii}| > \sum_{\substack{j=1 \\ j\neq i}}^{n} |a_{ji}| \quad i = 1, \ldots, n,$$
so v praksi pogoste. Zanje je značilno, da pivotiranje ni potrebno.

- *Pasovne matrike* s širino pasu $2p + 1$ so matrike, za katere je
$$a_{ik} = 0 \quad \text{za} \quad |i - k| > p.$$
Take matrike so zelo pogoste pri numeričnem reševanju diferencialnih enačb. Tudi pasovno strukturo, posebej ko je $p \ll n$ lahko s pridom izkoristimo pri reševanju.

Izmed iterativnih metod smo spozanali Jacobijevo, Gauss-Seidlovo metodo, metodo SOR in metodo konjugiranih gradientov. Iterativne metode se

uporabljajo v glavnem pri reševanju velikih sistemov z razpršeno matriko. Jacobijeva, Gauss-Seidelova in metoda SOR konvergirajo, kadar je matrika sistema diagonalno dominantna, metodo konjugiranih gradientov pa lahhko uporabimo, kadar je matrika sistema simetrična in pozitivno definitna. Take matrike nastopajo pogosto pri numeričnem reševanju parcialnih diferencialnih enačb.

2.11 Problemi

1. Reši problem iz primera 2.1 za podatke: $R_1 = 10k\Omega$, $R_2 = 25k\Omega$, $R_3 = 180k\Omega$, $R_4 = 1.5M\Omega$, $R_5 = 37k\Omega$, $R_6 = 1M\Omega$, $R_7 = 0.5M\Omega$, $U_A = 25V$, $U_B = 70V$, $U_C = 250V$ in $U_D = -75V$.

2. Preštej potrebne operacije (množenja in deljenja) za rešitev zgornjetrikotnega sistema enačb $U\boldsymbol{x} = \boldsymbol{b}$ z algoritmom 2.2.

3. Preveri, da za matrike M_i v dokazu izreka 2.3 veljajo enakosti (2.6).

4. Preveri, da v dokazu izreka 2.3 res velja $L = M_1^{-1} M_2^{-1} \cdots M_{n-1}^{-1}$.

5. Matrika A je *tridiagonalna*, če za njene elemente a_{ij} velja $a_{ij} = 0$ kadar je $|i - j| \geq 2$. Algoritem za LU razcep priredi za reševanje sistemov s tridiagonalno matriko. Kolikšno je število operacij, ki je potrebno za reševanje tridiagonalnega sistema reda n?

6. Matrika A je *zgornja Hessenbergova*[7], če je $a_{ij} = 0$ kadarkoli je $i-j \geq 2$.

   (a) Zapiši eno zgornjo Hessenbergovo matriko reda 5.

   (b) Priredi algoritem za LU razcep za reševanje sistema $A\boldsymbol{x} = \boldsymbol{b}$, kjer je A zgornja Hessenbergova matrika.

   (c) preštej aritmetične operacije, potrebne za rešitev sistema linearnih enačb z zgornjo Hessenbergovo matriko.

7. Izračunaj evklidsko ($||.||_2$), taksi ($||.||_1$) in maksimum normo ($||.||_\infty$) naslednjih vektorjev:

   (a) $\boldsymbol{a} = [1, 3, 5, 2, 3, 1]^T$

[7]Karl Hessenberg (1904 – 1959), nemški matematik, ukvarjal se je predvsem z numeričnimi metodami za izračun lastnih vrednosti matrike.

(b) $\boldsymbol{a} = [1, -1, 2, -2]^T$

(c) $\boldsymbol{a} = [2.31, 3.23, 4.87, -1.22, 2.92]^T$

Poglavje 3

Numerični problem lastnih vrednosti

V tem poglavju si bomo ogledali nekaj metod za računanje lastnih vrednosti in lastnih vektorrjev matrik, Ti problemi so ključ do rešitve pri nekaterih pomembnih uporabah v gradbeništvu, strojništvu, ststistiki in ekonomiji.

Naj bo $A \in \mathbb{R}^{n\times n}$ neka kvadratna matrika. Zanima nas preslikava f_A : $\mathbf{x} \to A\mathbf{x}$, ki vsakemu vektorju $\mathbf{x}$ iz n-razsežnega vektorskega prostora V (ki je lahko realni vektorski prostor $\mathbb{R}^n$, včasih pa tudi kompleksni vektorski prostor $\mathbb{C}^n$) priredi vektor $A\mathbf{x}$, ki tudi leži v prostoru V. Kadar vektor $\mathbf{x}$ izberemo naključno, se bo le redkokdaj zgodilo, da bosta $\mathbf{x}$ in njegova slika $A\mathbf{x}$ kolinearna. Pri problemu lastnih vrednosti in lastnih vektorjev pa nas zanima prav to: za katere vektorje $\mathbf{x} \in V$ sta original $\mathbf{x}$ in njegova slika $A\mathbf{x}$ kolinearna. Z drugimi besedami: za katere vektorje iz prostora V množenje z matriko A ohranja smer.

3.1 Lastne vrednosti in lastni vektorji

Pregled nekaterih dejstev, povezanih z lastnimi vrednostmi in lastnimi vektorji začnimo z definicijo lastne vrednosti in lastnega vektorja.

Definicija 3.1. *Za matriko* $A \in \mathbb{R}^{n\times n}$ *je število* λ lastna vrednost, *če obstaja tak vektor* $\mathbf{x} \in V$, *da je*

$$A\mathbf{x} = \lambda\mathbf{x}\,.$$

Vektor $\mathbf{x}$ *je* (desni) lastni vektor *matrike* A, *ki pripada lastni vrednosti* λ. *Lastno vrednost skupaj s pripadajočim lastnim vektorjem* $(\lambda, \mathbf{x})$ *imenujemo* lastni par *matrike* A.

Pozor! Čeprav govorimo o lastnem vektorju, mislimo na več kot en sam vektor. Če je vektor $\mathbf{x}$ lastni vektor matrike A, je tudi vsak njegov neničelni mnogokratnik $\alpha\mathbf{x}$ tudi lastni vektor iste matrike A, ki pripada isti lastni vrednosti λ, saj iz

$$A\mathbf{x} = \lambda\mathbf{x}$$

sledi, da je tudi

$$A(\alpha\mathbf{x}) = \alpha A\mathbf{x} = \alpha\lambda\mathbf{x} = \lambda(\alpha\mathbf{x})\,.$$

Velja pa še več: če lastni vrednosti λ pripada več linearno neodvisnih lastnih vektorjev $\mathbf{x}_1, \ldots, \mathbf{x}_k$, so tudi vse linearne kombinacije teh vektorjev lastni vektorji. Zato bi bilo bolje kot o lastnih vektorjih govoriti o lastnih podprostorih matrike.

Iz same definicije lastne vrednosti in lastnega vektorja lahko enostavno ugotovimo:

Izrek 3.1. *Če ima matrika A lastno vrednost λ in pripadajoči lastni vektor $\mathbf{x}$, potem ima*

1. *matrika A^2 lastno vrednost λ^2 in lastni vektor $\mathbf{x}$;*
2. *matrika A^n lastno vrednost λ^n in lastni vektor $\mathbf{x}$;*
3. *matrika A^{-1} lastno vrednost λ^{-1} in lastni vektor $\mathbf{x}$;*
4. *matrika $A + \sigma I$ lastno vrednost $\lambda + \sigma$ in lastni vektor $\mathbf{x}$.*

Ponovimo nekaj lastnosti lastnih vrednosti in lastnih vektorjev, ki smo jih že srečali pri linearni algebri.

Izrek 3.2. *Lastne vrednosti λ matrike A so ničle* karakterističnega polinoma

$$\det(A - \lambda I) = 0\,.$$

Lastne vektorje $\mathbf{x}$, ki ustrezajo lastni vrednosti λ_i lahko izračunamo kot rešitve homogenega sistema linearnih enačb

$$(A - \lambda I)\mathbf{x} = \mathbf{0}\,.$$

Pozor! Računanje lastnih vrednosti kot ničel karakterističnega polinoma se v praksi ne uporablja, saj je lahko že pri majhnih dimenzijah matrik zelo nestabilno.

Kadar je matrika realna, ima tudi karakteristični polinom realne koeficiente, zato velja:

Posledica 3.3. *Kvadratna matrika z n vrsticami ima n lastnih vrednosti, če jih štejemo z njihovo večkratnostjo.*
Če je matrika realna, so njene lastne vrednosti realna števila ali konjugirani pari kompleksnih števil.

Večkratne lasnte vrednosti matrike lahko povzročajo probleme, predvsem kadar jim pripada premalo linearno neodvisnih lastnih vektorjev.

Izrek 3.4. *Lastni vektorji, ki pripadajo različnim lastnim vrednostim, so linearno neodvisni.*

Posebej enostavno lahko ugotovimo, kaj so lastne vrednosti trikotne matrike:

Izrek 3.5. *Lastne vrednosti trikotne matrike so njeni diagonalni elementi.*

Podobna trditev velja za bločno trikotne matrike.

Izrek 3.6. *Matrika A naj bo*

$$A = \begin{bmatrix} A_{11} & A_{12} & \cdots & A_{1k} \\ 0 & A_{22} & \cdots & A_{2k} \\ & & \ddots & \vdots \\ 0 & 0 & \dots & A_{kk} \end{bmatrix},$$

kjer je vsaka od matrik A_{ii} kvadratna. Potem je vsaka od lastnih vrednosti matrik A_{ii} tudi lastna vrednost matrike A. Njena večkratnost v matriki A je vsota večkratnosti te lastne vrednosti v vseh matrikah A_{ii}.

Zanimive so matrike, ki imajo enake lastne vrednosti.

Izrek 3.7. *Naj bo A poljubna $n \times n$ matrika, T pa obrnljiva matrika iste dimenzije. Potem ima matrika*

$$B = T^{-1}AT$$

iste lastne vrednosti kot matrika A.

Relacija med dvema matrikama iz prejšnjega izreka je dovolj pomembna, da zasluži posebno ime.

Definicija 3.2. *Kvadratni matriki A in B reda n sta* podobni, *kadar obstaja obrnljiva matrika T, da je $B = T^{-1}AT$.*

V primeru, da ima matrika dovolj linearno neodvisnih lastnih vektorjev, je podobna diagonalni matriki.

Izrek 3.8. *Denimo, da ima matrika $A \in \mathbb{R}^{n\times n}$ n linearno neodvisnih lastnih vektorjev $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n$. Če jih zložimo kot stolpce v matriko S*

$$S = [\mathbf{x}_1\ \mathbf{x}_2\ \cdots\ \mathbf{x}_n],$$

potem je $\Lambda = S^{-1}AS$ diagonalna matrika z lastnimi vrednostmi λ_i, $i = 1, \ldots, n$ na diagonali

$$S^{-1}AS = \Lambda = \begin{bmatrix} \lambda_1 & & \\ & \ddots & \\ & & \lambda_n \end{bmatrix}.$$

Matriki, ki zadošča pogojem izreka 3.8, pravimo *diagonalizabilna* matrika.

Pri simetričnih matrikah je skrb, da lastne vrednosti ne bi bile realne, odveč.

Izrek 3.9. *Če je matrika A simetrična ($A^T = A$), potem so vse njene lastne vrednosti realne, lastni vektorji, ki pripadajo različnim lastnim vrednostim pa so ortogonalni. Vsaki lastni vrednosti pripada toliko lastnih vektorjev, kolikor je večkratnost te lastne vrednosti.*

Izrek 3.9 zagotavlja, da so lastni vektorji, ki pripadajo različnim lastnim vrednostim, ortogonalni. Ker lahko lastne vektorje, ki pripadajo isti večkratni lastni vrednosti vedno izberemo tako, da so ortogonalni, in ker je vseh linearno neodvisnih lastnih vektorjev (niso le neodvisni, ampak celo paroma ortogonalni) dovolj, lahko izrek 3.8 o diagonalizaciji za simetrične matrike zapišemo enostavneje.

Izrek 3.10. *Za vsako realno simetrično matriko $A = A^T$ obstaja taka ortogonalna matrika Q, da je $\Lambda = Q^T AQ$ diagonalna matrika, katere diagonalni elementi so lastne vrednosti matrike A.*

Za nesimetrične matrike diagonalizacija ne obstaja vedno. Zadovoljiti se moramo z nekoliko manj enostavnimi oblikami matrik. Za praktično računanje lastnih vrednosti in lastnih vrednosti je najpomembnejša zgornjetrikotna oblika matrike, o kateri govori *Schurov*[1] *izrek*

Izrek 3.11. Schurov izrek *Za vsako kvadratno matriko A obstaja taka unitarna matrika U, da je*

$$U^H A U = T$$

zgornjetrikotna matrika z lastnimi vrednostmi matrike A na diagonali.

Ker so lastne vrednosti realne matrike lahko kompleksne (v konjugiranih parih), sta tudi za realno matriko A matriki U in T lahko kompleksni. Kljub temu lahko ostanemo pri realnih matrikah, če le malo popustimo pri zahtevi o trikotnosti matrike T:

Izrek 3.12. Realni Schurov izrek *Za vsako realno kvadratno matriko A obstaja taka ortogonalna matrika Q, da je*

$$Q^T A Q = R = \begin{bmatrix} R_{11} & R_{12} & \cdots & R_{1k} \\ 0 & R_{22} & \cdots & R_{2k} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & R_{kk} \end{bmatrix},$$

kjer so R_{ii} skalarji ali matrike 2×2. Skalarni elementi na diagonali so realne lastne vrednosti matrike A, matrike 2×2 na diagonali pa ustrezajo konjugiranemu paru kompleksnih lastnih vrednosti.

Za dokaze navedenih izrekov bralec lahko pogleda v ustrezen učbenik linearne algebre.

[1] Issai Schur, (1875, Mogilev (Rusija, sedaj v Belorusiji) – 1941, Tel Aviv (Izrael)), Nemški matematik, poznan predvsem po svojih rezultatih v reprezentacijah grup in matričnem razcepu, imenovanem po njem.

3.2 Lokalizacija lastnih vrednosti

Preden dejansko računamo lastne vrednosti matrike je koristno poznati vsaj vsaj približno distribucijo lastnih vrednosti ali približne vrednosti nekaj določenih lastnih vrednosti. Včasih nas je dovolj, če namesto točnih lastnih vrednosti le približno vemo, kje na kompleksni vrednosti se dejanske lastne vrednosti nahajajo. V tem razdelku si bomo ogledali nekaj rezultatov, s katerimi lahko dobimo nekaj informacij o lokaciji lastnih vrednosti, ne da bi jih dejansko izračunali.

3.2.1 Lastne vrednosti in matrične norme

Osnovno informacijo o lastnih vrednostih matrike lahko dobimo že iz matrične norme.

Izrek 3.13. *Naj bo λ lastna vrednost matrike A. Potem za vsako matrično normo, definirano s pomočjo vektorske norme (2.13) velja*

$$|\lambda| \leq ||A|| \,.$$

Dokaz: Za vsako lastno vrednost λ obstaja ustrezni lastni vektor $\mathbf{x}$, da je $A\mathbf{x} = \lambda\mathbf{x}$, zato tudi

$$||A\mathbf{x}|| = ||\lambda\mathbf{x}|| = |\lambda| \, ||\mathbf{x}|| \,.$$

Ker za vsako matrično normo, dobljeno iz vektorske norme velja, da je $||A\mathbf{x}|| \leq ||A|| \, ||x||$, je tudi

$$|\lambda| \, ||\mathbf{x}|| = ||A\mathbf{x}|| \leq ||A|| \, ||\mathbf{x}|| \,,$$

od koder zlahka zaključimo, da mora biti $|\lambda| \leq ||A||$, saj je vedno $||\mathbf{x}|| > 0$.

Iz lastnosti matričnih norm lahko dobimo praktično oceno za zgornjo mejo vseh lastnih vrednosti matrike.

Posledica 3.14. *Za lastne vrednosti λ matrike A velja*

$$|\lambda| \leq \min \left(\max_i \sum_{j=1}^{n} |a_{ij}|, \ \max_j \sum_{i=1}^{n} |a_{ij}| \right) .$$

3.2.2 Izrek o Geršgorinovih diskih

Naj bo $A = [a_{ij}]_{ij=1}^n$ matrika. Izračunajmo vsoto absolutnih vrednosti izvendiagonalnih elementov v vsaki vrstici

$$r_i = \sum_{j=1}^{n} |a_{ij}| \quad \text{za} \quad i = 1, \ldots, n\,.$$

Geršgorinov[2] disk G_i, ki pripada i-ti vrstici matrike A je množica kompleksnih števil

$$G_i = \{z \in \mathbb{C};\ |z - a_{ii}| \leq r_i\}\,,$$

ki so od diagonalnega elementa a_{ii} v tisti vrstici oddaljene ne več kot r_i.

Izrek 3.15. Prvi Geršgorinov izrek *Vse lastne vrednosti matrike A so vsebovane v uniji Geršgorinovih diskov*

$$\forall i:\ \lambda_i \in \cup_{j=1}^{n} G_j$$

Dokaz: Za lastno vrednost λ matrike A in ustrezni lastni vektor $\mathbf{x} = [x_1, \ldots, x_n]^T$ zaradi $A\mathbf{x} = \lambda\mathbf{x}$, oziroma $(\lambda I - A)\mathbf{x} = \mathbf{0}$ velja

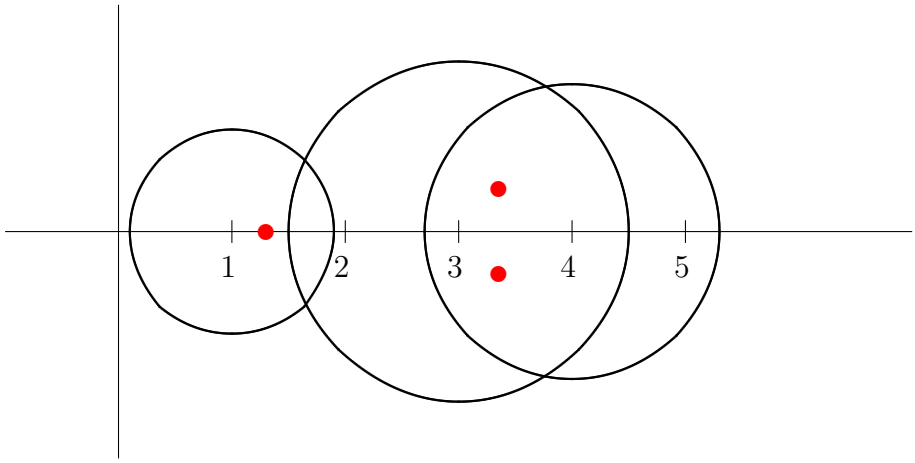
$$(\lambda - a_{ii})x_i = \sum_{\substack{j=1 \\ j\neq i}}^{n} a_{ij}x_j\,.$$

Če je k komponenta vektorja $\mathbf{x}$ z največjo absolutno vrednostjo ($|x_k| \geq |x_j|$ za vse $j = 1, \ldots, n$), je $|x_j|/|x_k| \leq 1$ za vse j, zato tudi

$$|\lambda - x_k| \leq \sum_{\substack{j=1 \\ j\neq i}}^{n} |a_{kj}| \frac{|x_j|}{|x_k|} \leq \sum_{\substack{j=1 \\ j\neq i}}^{n} |a_{kj}|\,,$$

in λ mora biti znotraj diska $\{\lambda\,;\ |\lambda - a_k k| \leq r_k\}$.

[2]Semjon Aranovič Geršgorin, (1901, Pružani (Rusija, sedaj v Belorusiji) – 1933 Leningrad (Rusija)), sovjetski matematik, ukvarjal se je z uporabno matematiko, predvsem z reševanjem parcialnih diferencialnih enačb.



Slika 3.1: Geršgorinovi diski za Primer 3.1. Rdeče pike predstavljajo lastne vrednosti.

Primer 3.1. Matrika A naj bo

$$A = \begin{bmatrix} 1.0 & 0.5 & -0.4 \\ -0.5 & 3.0 & 1.0 \\ 0.8 & -0.5 & 4.0 \end{bmatrix} \tag{3.1}$$

ima lastne vrednosti $\lambda_1 \approx 1.2982$, $\lambda_2 \approx 3.3509 + 0.3759\,i$ in $\lambda_3 \approx 3.3509 - 0.3759\,i$.
Geršgorinovi diski matrike A so:

1. G_1 ima središče v $(1, 0)$ in polmer 0.9
2. G_2 ima središče v $(3, 0)$ in polmer 1.5
3. G_1 ima središče v $(4, 0)$ in polmer 1.3

Vsi trije so na Sliki 3.1. Lastne vrednosti so označene z rdečimi pikami.

Naslednji izrek, ki ga navajamo brez dokaza, nam pove še nekoliko več o razporeditvi lastnih vresdnosti v Geršgorinovih diskih.

Izrek 3.16. Drugi Geršgorinov izrek *Kadar skupina r Geršgorinovih diskov nima skupnega preseka z ostalimi, v uniji teh r diskov leži natanko r lastnih vrednosti.*

Ker so lastne vrednosti matrike A^T iste kot lastne vrednosti matrike A, lahko Geršgorinove diske računamo tudi po stolpcih:

$$G_j = \{z \in \mathbb{C};\ |z - a_{jj}| \leq \sum_{i=1}^{n} |a_{ij}|\}\,.$$

Lastne vrednosti matrike A morajo še vedno ležati v uniji vseh Geršgorinovih diskov (Izrek 3.15) in v vsaki skupini r Geršgorinovih diskov, disjunktnih z unijo ostalih $n - r$ diskov, leži natanko r lastnih vrednosti (Izrek 3.16).

3.3 Računanje izbranih lastnih vrednosti in lastnih vektorjev

Pri veliko aplikacijah, kjer potrebujemo lastne vrednosti je dovolj, če poznamo le nekaj (največjih ali najmanjših) lastnih vrednosti. Če nas zanima obnašanje zaporedja vektorjev, definiranih z rekurzivno relacijo

$$\mathbf{x}_{n+1} = A\mathbf{x}_n$$

kjer je $\mathbf{x}_0$ dan začetni vektor in A matrika, je največkrat dovolj, da poznamo dve, po absolutni vrednosti največji lastni vrednosti. Prav tako nas pri *analizi glavnih komponent (Principal component analysis, PCA)* zanima le nekaj največjih lastnih vrednosti.

Pozor! Lastne vrednosti matrike A reda n so ničle njenega karakterističnega polinoma, ki je stopnje n. Zaradi znanega rezultata (ki sta ga neodvisno dokazala N. H. Abel[3]in E. Galois[4]), da se ničel poljubnega polinoma stopnje pet ali višje ne da izračunati v končnem številu operacij, morajo biti numerične metode za računanje lastnih vrednosti poljubnih matrik nujno iterativne.

[3]Nils Henrik Abel (1802, Nedstrand (v bližini Stavangerja, Norveška) – 1829, Froland (v bližini Arendala, Norveška)) norveški matematik, ukvarjal se je z eliptičnimi integrali, začetnik teorije eliptičnih funkcij, skupaj z Galoisem začetnik teorije grup.

[4]Évariste Galois (1811, Bourg La Reine (v bližini Pariza, Francija) – 1832, Pariz (Fran-

3.3.1 Potenčna metoda

Potenčno metodo pogosto uporabljamo za izračun *dominantne lastne vrednosti* (t.j. tista lastna vrednost, ki ima največjo absolutno vrednost). Predvsem je primerna za razpršene matrike, ker je množenje matrike z vektorjem glavna operacija. Metoda je dobila ime po potencah matrike, ki jih konstruiramo implicitno.

Naj bodo lastne vrednosti matrike A take, da je

$$|\lambda_1| > |\lambda_2| \geq |\lambda_3| \geq \cdots \geq |\lambda_n|\,,$$

tako da je λ_1 dominantna lastna vrednost, in naj bo $\mathbf{x}_1$ ustrezni lastni vektor.

Izrek 3.17. *Matrika A naj bo diagonalizabilna, vektor $\mathbf{y}^0$ izberimo tako, da ni ortogonalen na $\mathbf{x}_1$. Zaporedje vektorjev $\mathbf{y}^{(i)}$, ki ga tvorimo rekurzivno*

$$\begin{aligned} \hat{\mathbf{y}}^{(k)} &= A\mathbf{y}^{(k-1)} \\ \mathbf{y}^{(k)} &= \hat{\mathbf{y}}^{(k)} / \max_{1\leq i\leq n} \hat{\mathbf{y}}_i^{(k)} \quad za \quad k = 1, 2, \ldots \end{aligned}$$

konvergira proti dominantnemu lastnemu vektorju $\mathbf{x}_1$, $\max_{1\leq i\leq n} \hat{\mathbf{y}}_i^{(k)}$ pa proti dominantni lastni vrednosti λ_1:

$$\lim_{k\to\infty} \frac{y_i^{(k+1)}}{y_i^{(k)}} = \lambda_1 \quad za\ vse \quad i\,.$$

cija)), francoski matematik. Skupaj z Abelom začetnik teorije grup.

Dokaz: Ker je matrika A diagonalizabilna, lahko smatramo, da so lastni vektorji $\mathbf{x}_i$, ki pripadajo lastnim vrednostim λ_i, linearno neodvisni, torej so baza prostora $\mathbb{R}^n$. Zato lahko začetni vektor $\mathbf{y}^0$ razvijemo po tej bazi

$$\mathbf{y}^{(0)} = \alpha_1\mathbf{x}_1 + \cdots + \alpha_n\mathbf{x}_n.$$

Potem je

$$\begin{aligned} A^k\mathbf{y}^{(0)} &= A^k(\alpha_1\mathbf{x}_1 + \cdots + \alpha_n\mathbf{x}_n) \\ &= \alpha_1\lambda_1^k\mathbf{x}_1 + \cdots + \alpha_n\lambda_n^k\mathbf{x}_n \\ &= \lambda_1^k\left[\alpha_1\mathbf{x}_1 + \alpha_2\left(\frac{\lambda_2}{\lambda_1}\right)^k\mathbf{x}_2 + \cdots + \alpha_n\left(\frac{\lambda_n}{\lambda_1}\right)^k\mathbf{x}_n\right]. \end{aligned}$$

Ker ima lastna vrednost λ_1 med vsemi λ_i največjo absolutno vrednost, je

$$\lim_{k\to\infty}\left(\frac{\lambda_i}{\lambda_1}^k\right) = 0 \quad \text{za vse} \quad i = 2, \ldots, n\,,$$

zato tudi

$$\mathbf{y}^{(k)} = \frac{A^k\mathbf{y}^{(0)}}{\max(A^k\mathbf{y}^{(0)})} \to \sigma\mathbf{x}_1,$$

kjer je σ neka konstanta, pa tudi

$$\max\hat{\mathbf{y}}^{(k)} \to \lambda_1\,.$$

Da potenčna metoda konvergira, smo dokazali za primer, ko matrika izpolnjuje dva dodatna pogoja:

1. $\alpha_1 \neq 0$ in
2. λ_1 je *edina* dominantna lastna vrednost.

Prvi pogoj ($\alpha_1 \neq 0$) v praksi ne predstavlja resne ovire, saj običajno zaokrožitvene napake povzročijo, da je že po nekaj iteracijah izpolnjen.

Drugi pogoj potrebuje le nekoliko več razmisleka. Če je dominantna lastna vrednost $\lambda_1 = \lambda_2 = \cdots = \lambda_r$ r-kratna, $|\lambda_1| > |\lambda_{r+1}| \leq \cdots |\lambda_n|$ in lastni vrednosti λ_1 pripada r linearno neodvisnih lastnih vektorjev $\mathbf{x}_1, \ldots, \mathbf{x}_r$, po-

tem je (ker je $(\lambda_i/\lambda_1)^k$ za velike k poljubno majhen)

$$\begin{aligned} A^k\mathbf{y}^{(0)} &= \lambda_1^k\left(\sum_{i=1}^{r}\alpha_i\mathbf{x}_i + \sum_{i=r+1}^{n}\alpha_i\left(\frac{\lambda_i}{\lambda_1}^k\right)\mathbf{x}_i\right) \\ &\approx \lambda_1^k\sum_{i=1}^{r}\alpha_i\mathbf{x}_i\,. \end{aligned}$$

To kaže, da potenčna metoda v tem primeru konvergira proti nekemu vektorju v lastnem podprostoru, ki pripada dominantni lastni vrednosti.

Kadar pa imamo več različnih dominantnih lastnih vrednosti, npr.

$$|\lambda_1| = |\lambda_2| \quad \text{vendar} \quad \lambda_1 \neq \lambda_2 \quad \text{in} \quad |\lambda_1| > |\lambda_3| \geq \cdots \geq |\lambda_n|\,,$$

pa potenčna metoda ne deluje.

Zapišimo algoritem za potenčno metodo:

Algoritem 3.1. Potenčna metoda Matrika $A \in \mathbb{C}^{n\times n}$ naj bo diagonalizabilna, tako da obstaja taka matrika $X = [\mathbf{x}_1, \ldots, \mathbf{x}_n]$, da je $X^{-1}AX = diag(\lambda_1, \ldots, \lambda_n)$ diagonalna matrika z edino dominantno lastno vrednostjo λ_1. Dan naj bo n-dimenzionalni vektor $\mathbf{y}^{(0)}$ z $||\mathbf{y}^{(0)}||_1 = 1$. Naslednji algoritem izračuna zaporedje vektorjev $\mathbf{y}^{(k)}$, ki konvergirajo proti lastnemu vektorju $\mathbf{x}_1$, ki pripada dominantni lastni vrednosti λ_1.

$ls = \max_i \mathbf{y}_i^{(0)}$
for $k = 1 : Nmax$
  $\hat{\mathbf{y}}^{(k)} = A\mathbf{y}^{(k-1)}$
  $ln = \max_i \hat{\mathbf{y}}_i^{(k)}$
  $\mathbf{y}^{(k)} = \hat{\mathbf{y}}^{(k)}/ln$
  if (**abs**$(ln - ls)) < tol$
    break
  endif
  $ls = ln$
end
$\lambda = ln$
$\mathbf{x} = \mathbf{y}^{(k)}$

Hitrost konvergence potenčne metode je odvisna od absolutne vrednosti razmerja med dominantno lastno vrednostjo in po velikosti naslednjo lastno vrednostjo matrike. Čim večje je to razmerje, hitrejša je konvergenca.

Primer 3.2. Matrika

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 5 & 2 \\ 3 & 2 & 5 \end{bmatrix}$$

ima lastne vrednosti $\lambda_1 = 8.6526$, $\lambda_2 = 3.0944$ in $\lambda_3 = -0.7470$. Dominantni lastni vrednosti pripada lastni vektor $x = [0.62450,\ 0.88952,\ 1.00000]^T$ (normiran v $||.||_\infty$ normi).
Če izberemo začetni približek k lastnemu vektorju $\mathbf{y}^{(0)} = [1,\ 1,\ 1]^T$, zaporedoma dobimo
$k = 1$

$$\begin{aligned} \hat{\mathbf{y}}^{(1)} &= A * \mathbf{y}^{(0)} = [6,\ 9,\ 10] \\ ln &= \max \hat{\mathbf{y}}^{(1)} = 10 \\ \mathbf{y}^{(1)} &= \hat{\mathbf{y}}^{(1)}/ln = [0.6,\ 0.9,\ 1] \end{aligned}$$

$k = 2$

$$\begin{aligned} \hat{\mathbf{y}}^{(2)} &= A * \mathbf{y}^{(1)} = [5.4,\ 7.7,\ 8.6] \\ ln &= \max \hat{\mathbf{y}}^{(2)} = 8.6 \\ \mathbf{y}^{(2)} &= \hat{\mathbf{y}}^{(2)}/ln = [0.62791,\ 0.89535,\ 1] \end{aligned}$$

$k = 3$

$$\begin{aligned} \hat{\mathbf{y}}^{(3)} &= A * \mathbf{y}^{(2)} = [5.4186,\ 7.7326,\ 8.6744] \\ ln &= \max \hat{\mathbf{y}}^{(3)} = 8.6744 \\ \mathbf{y}^{(3)} &= \hat{\mathbf{y}}^{(3)}/ln = [0.62466,\ 0.89142,\ 1] \end{aligned}$$

$k = 4$

$$\begin{aligned} \hat{\mathbf{y}}^{(4)} &= A * \mathbf{y}^{(3)} = [5.4075,\ 7.7064,\ 8.6568] \\ ln &= \max \hat{\mathbf{y}}^{(4)} = 8.6568 \\ \mathbf{y}^{(4)} &= \hat{\mathbf{y}}^{(4)}/ln = [0.62465,\ 0.89021,\ 1]\,. \end{aligned}$$

Vidimo, da ln konvergira proti dominantni lastni vrednosti, medtem ko $\mathbf{y}^{(k)}$ konvergira proti pripadajočemu lastnemu vektorju.

Potenčna metoda je primerna predvsem, kadar hočemo izračunati dominantno lastno vrednost velike razpršene (večina elementov v matriki enakih

0) matrike in ustrezen lastni vektor.

3.3.2 Inverzna iteracija

Huda pomanjkljivost potenčne metode je, da z njo lahko izračunamo samo dominantno lastno vrednost in da je hitrost njene konvergence odvisna od razmerja med po absolutni vrednosti drugo lastno vrednostjo in dominantno lastno vrednostjo, ki je odvisno le od matrike in nanj ne moremo vplivati. To pomanjkljivost lahko odpravimo tako, da potenčno metodo uporabimo na matriki $(A - \sigma I)^{-1}$ kjer število σ imenujemo *pomik*.

Algoritem 3.2. Inverzna iteracija Naj bo pomik σ tak približek lastne vrednosti λ_i matrike A, da je $|\lambda_i - \sigma| \ll |\lambda_j - \sigma|$ za vse $j \neq i$ (σ je veliko bliže λ_i kot katerikoli drugi lastni vrednosti). Dan naj bo n-dimenzionalni vektor $\mathbf{y}^{(0)}$ z $||\mathbf{y}^{(0)}||_1 = 1$.
Naslednji algoritem izračuna zaporedje vektorjev $\mathbf{y}^{(k)}$, ki konvergirajo proti lastnemu vektorju $\mathbf{x}_i$, ki pripada lastni vrednosti λ_i in zaporedje števil $l^{(k)}$, ki konvergirajo proti lastni vrednosti λ_i.

$[L, U, P] = \mathbf{lu}(A - \sigma I)$
for $k = 1 : Nmax$
  $\mathbf{z}^{(k)} = L \setminus P * \mathbf{y}^{(k-1)}$
  $\hat{\mathbf{y}}^{(k)} = U \setminus \mathbf{z}^{(k-1)}$
  $\mathbf{y}^{(k)} = \hat{\mathbf{y}}^{(k)} / \max_i \hat{\mathbf{y}}_i^{(k)}$
  $l^{(k)} = \left(\mathbf{y}^{(k)}\right)^T A\, \mathbf{y}^{(k)}$
  if $(||(A - l^{(k)} I)\mathbf{y}^{(k)}||_\infty < tol||A||_\infty)$
    break
  endif
end

Izrek 3.18. *Zaporedje vektorjev $\mathbf{y}^{(k)}$ iz Algoritma 3.2 konvergira proti lastnemu vektorju $\mathbf{x}_i$, zaporedje $l^{(k)}$ pa proti lastni vrednosti λ_i.*

Dokaz: Lastne vrednosti matrike $(A-\sigma I)^{-1}$ so (glej Izrek 3.1) $(\lambda_1-\sigma I)^{-1}$, $(\lambda_2-\sigma I)^{-1}$,... $(\lambda_n-\sigma I)^{-1}$, lastni vektorji $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n$ pa so isti kot pri matriki A. Če lastni vektorji sestavljajo bazo prostora $\mathbb{R}^n$, lahko začetni vektor $\mathbf{y}^{(0)}$ razvijemo po tej bazi

$$\mathbf{y}^{(0)} = \sum_{j=1}^{n} \alpha_j \mathbf{x}_j \,.$$

Za vektor $\mathbf{y}^{(k)}$ potem velja

$$\mathbf{y}^{(k)} = (A-\sigma I)^{-k}\mathbf{y}^{(0)} = \sum_{j=1}^{n} \frac{\alpha_j}{(\lambda_j-\sigma)^k}\mathbf{x}_j = \frac{1}{(\lambda_i-\sigma)^k}\sum_{j=1}^{n}\left(\frac{\lambda_i-\sigma}{\lambda_j-\sigma}\right)^k \alpha_j \mathbf{x}_j \,.$$

Ker je σ bliže λ_i kot katerikoli drugi lastni vrednosti λ_j, zaporedje vektorjev $(\mathbf{y}^{(k)})$ po smeri kovergira proti $\mathbf{x}_i$, zato tudi $\left(\mathbf{y}^{(k)}\right)^T A\,\mathbf{y}^{(k)}$ konvergira proti λ_i.

Na prvi pogled inverzna iteracija izgleda zelo sumljivo. Čim bliže je parameter σ lastni vrednosti matrike A, tem večje je število občutljivosti matrike $A-\sigma I$, zato je lastni vektor $\mathbf{y}^{(k)}$ lahko izračunan s precejšnjo napako. Dejansko pa nam velika občutljivost matrike $A-\sigma I$ pri praktičnem računanju še kako prav pride. Napaka v vsaki iteraciji narašča v smer lastnega vektorja, kar nam še dodatno pospeši konvergenco.

Glavna prednost inverzne iteracije je, da lahko izberemo, proti kateremu lastnemu vektorju naj konvergira (proti tistemu, ki pripada lastni vrednosti, ki je najbliže pomiku σ). Če izberemo pomik σ zelo blizu izbrani lastni vrednosti, bo inverzna iteracija konvergirala zelo hitro. Inverzna iteracija je torej zelo učinkovita, kadar že poznamo dober približek lastne vrednosti in nas zanima le še ustrezni lastni vektor.

Slaba stran te metode pa je, da je ni primerna za poljubno velike matrike, saj moramo na vsakem koraku rešiti sistem linearnih enačb, torej potrebujemo LU razcep matrike $A-\sigma I$.

Primer 3.3. Matrika

$$A = \begin{bmatrix} 3 & 3 & 0 \\ 3 & 2 & -7 \\ 0 & -8 & 6 \end{bmatrix}$$

ima lastne vrednosti $\lambda_1 = 12.1266$, $\lambda_2 = -4.5200$ in $\lambda_3 = 3.3934$. Zanima nas lastna vrednost λ_2, ki ji pripada lastni vektor $x = [-0.30266,\ 0.75866,\ 0.57692]^T$ (normiran v $||.||_2$ normi), zato izberemo pomik $\sigma = -5$.
Če izberemo začetni približek k lastnemu vektorju $\mathbf{y}^{(0)} = [1,\ 1,\ 1]^T$, zaporedoma dobimo
$k = 1$

$$\begin{aligned} \mathbf{y}^{(1)} &= (A+5I)^{-1}\mathbf{y}^{(0)}/||(A+5I)^{-1}\mathbf{y}^{(0)}||_2 = [-0.2278,\ 0.7664,\ 0.6007] \\ \lambda &= \left(\mathbf{y}^{(1)}\right)^T A\mathbf{y}^{(1)} = -4.4573 \end{aligned}$$

$k = 2$

$$\begin{aligned} \mathbf{y}^{(2)} &= (A+5I)^{-1}\mathbf{y}^{(1)}/||(A+5I)^{-1}\mathbf{y}^{(1)}||_2 = [-0.2984,\ 0.7592,\ 0.5785] \\ \lambda &= \left(\mathbf{y}^{(2)}\right)^T A\mathbf{y}^{(2)} = -4.5190 \end{aligned}$$

$k = 3$

$$\begin{aligned} \mathbf{y}^{(3)} &= (A+5I)^{-1}\mathbf{y}^{(2)}/||(A+5I)^{-1}\mathbf{y}^{(2)}||_2 = [-0.3024, 0.7587,\ 0.5770] \\ \lambda &= \left(\mathbf{y}^{(3)}\right)^T A\mathbf{y}^{(3)} = -4.5199 \end{aligned}$$

$k = 4$

$$\begin{aligned} \mathbf{y}^{(4)} &= (A+5I)^{-1}\mathbf{y}^{(3)}/||(A+5I)^{-1}\mathbf{y}^{(3)}||_2 = [-0.3026, 0.7587,\ 0.5769] \\ \lambda &= \left(\mathbf{y}^{(4)}\right)^T A\mathbf{y}^{(4)} = -4.5200 \end{aligned}$$

Vidimo, da λ konvergira proti lastni vrednosti λ_2, medtem ko $\mathbf{y}^{(k)}$ konvergira proti pripadajočemu lastnemu vektorju.

3.4 Računanje vseh lastnih vrednosti

Vse lastne vrednosti hkrati lahko izračunamo s podobnostjo transformacijo matrike. Če je T obrnljiva matrika, potem imata matriki A in $T^{-1}AT$ iste lastne vrednosti (3.7). Pri tem je pametno transformacijsko matriko T izbrati tako, da je

1. transformirana matrika $T^{-1}AT$ čim enostavnejše oblike, tako da njene lastne vrednosti enostavno določimo (npr. trikotna ali diagonalna) in

2. transformacijska matrika T naj ima število občutljivosti čim manjše (npr. ortogonalna, ki ima število občutljivosti enako 1).

3.4.1 Osnovna QR iteracija

Opisali bomo metodo, ki je znana pod imenom *QR iteracija* in ki kvadratno matriko A preoblikuje v Schurovo obliko, o kateri govori izrek 3.12. Za odkritelja metode velja J. G. Francis[5], čeprav je istega leta metodo neodvisno opisala tudi V. N. Kublanovskaja[6].

Ideja QR iteracije je preprosta: Naj bo $A_0 = A$. Najprej izračunamo QR razcep

$$A_0 = Q_0 R_0 \,,$$

potem pa matriki Q_0 in R_0 spet zmnožimo, vendar v obratnem vrstnem redu

$$A_1 = R_0 Q_0$$

in tako naprej. V k-tem koraku najpre izračunamo QR razcep matrike A_{k-1}

$$A_{k-1} = Q_{k-1} R_{k-1}$$

in nato

$$A_k = R_{k-1} Q_{k-1} \,.$$

[5]John G. F. Francis (1934, London (Velika Britanija)), angleški matematik in računalniški znanstvenik. Po objavi članka o QR iteraciji se je ukvarjal predvsem z umetno inteligenco in računlniškimi jeziki.

[6]Vera Nikolajevna Kublanovskaja, roj Totubalina (1920, Krohino (Rusija) – 2012), ruska matematičarka. Poznana predvsem po svojem delu pri reševanju problemov numerične linearne algebre.

Ker je Q_0 ortogonalna, je $R_0 = Q_0^T A_0$ in zato

$$A_1 = R_0 Q_0 = Q_0^T A_0 Q_0 \,,$$

sta matriki A_0 in A_1 podobni (definicija 3.2) in imata (po izreku 3.7) iste lastne vrednosti. Prav tako je

$$A_k = Q_{k-1}^T A_{k-1} Q_{k-1} \,,$$

zato so vse matrike A_i podobne in imajo torej vse iste lastne vrednosti.

Ker zaporedje matrik A_i konvergira[7] proti diagonalni matriki ali proti Schurovi obliki matrike (kadar so lastne vrednosti konjugirno kompleksne), se realne lastne vrednosti pojavijo na diagonali kompleksne, lastne vrednosti (ki so paroma konjugirano kompleksne) pa so lastne vrednosti diagonalnih 2×2 blokov.

Algoritem 3.3. Osnovna QR iteracija Naj bo A realna kvadratna matrika reda n. Naslednji algoritem generira zaporedje matrik, ki imajo vse iste lastne vrednosti kot A, ki konvergira proti zgornjetrikotni matriki ali Schurovi formi:

$[Q, R] = qr(A)$
repeat $[Q, R] = qr(A)$
$A = RQ$
until poddiagonalni element dovolj majhni

[7] Konvergenca QR iteracije je sicer dokazana le v primeru, ko so vse lastne vrednosti med seboj različne po absolutnoi vrednosti, praktično pa konvergira vedno.

Primer 3.4. S QR iteracijo izračunajmo vse lastne vrednosti matrike iz primera 3.2. Zaporedoma dobimo

$$A_0 = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 5 & 2 \\ 3 & 2 & 5 \end{bmatrix} = Q_0 R_0;$$

$$A_1 = R_0 Q_0 = \begin{bmatrix} 8.28571 & 1.09447 & 1.36505 \\ 1.09447 & 2.85921 & -1.00707 \\ 1.36505 & -1.00707 & -0.14493 \end{bmatrix} = Q_1 R_1;$$

$$A_2 = R_1 Q_1 = \begin{bmatrix} 8.64143 & 0.23255 & 0.12867 \\ 0.23255 & 3.07516 & -0.32728 \\ 0.12867 & -0.32728 & -0.71659 \end{bmatrix} = Q_2 R_2;$$

$$A_3 = R_2 Q_2 = \begin{bmatrix} 8.651405 & 0.079257 & 0.011158 \\ 0.079257 & 3.093859 & -0.080638 \\ 0.011158 & -0.080638 & -0.745264 \end{bmatrix} = Q_3 R_3;$$

$$A_4 = R_3 Q_3 = \begin{bmatrix} 8.65240141 & 0.02826492 & 0.00096352 \\ 0.02826492 & 3.09447361 & -0.01950543 \\ 0.00096352 & -0.01950543 & -0.74687502 \end{bmatrix} = Q_4 R_4$$

in tako dalje. Vidimo lahko, da zaporedje A_0, A_1, A_2, A_3 konvergira proti diagonalni matriki, diagonalni elementi pa proti lastnim vrednostim.

Osnovna varianta QR iteracije ni dovolj učinkovita, kadar ima matrika A večjo dimenzijo in le malo ničel, saj za posamezni korak iteracije potrebujemo $\mathcal{O}(n^3)$ operacij, kar pomeni $\mathcal{O}(n^4)$ operacij za n iteracij. Na srečo obstaja enastavna rešitev za zmanjšanje števila operacij.

3.4.2 Izboljšave QR iteracije

Redukcija na Hessenbergovo obliko Da bi zmanjšali število operacij, potrebnih za QR iteracijo, matriko najprej reduciramo na preprostejšo obliko. Tukaj običajno uporabimo Hessenbergovo obliko matrike, saj za QR razcep Hessenbergove matrike $n \times n$ potrebujemo le $\mathcal{O}(n^2)$ operacij, prav tako pa velja, da so vse matrike v zaporedju $A_1, A_2, \ldots$ Hessenbergove, če je začetna matrika A_0 v Hessenbergovi obliki.

Vsako realno $n \times n$ matrio A lahko preoblikujemo v Hessenbergovo ma-

triko H s podobnostno transformacijo

$$PAP^T = H,$$

kjer je P ortogonalna matrika. Za izračun matrike P lahko priredimo idejo ortogonalnih transformacij, ki smo jo uporabili pri QR razcepu matrike (algoritem 2.8).

Če ima Hessenbergova matrika na poddiagonali kakšno ničlo, potem je

$$H = \begin{bmatrix} H_{11} & H_{12} \\ 0 & H_{22} \end{bmatrix} \begin{matrix} p \\ n-p \end{matrix}$$

$$\begin{matrix} p & n-p \end{matrix}$$

kjer je $1 \leq p < n$ n problem rapzde v dva manjša problema: določiti je treba lastne vrednosti matrik H_{11} in H_{22}.

V praksi lahko smatramo, da problem razpade, kadar je poddiagonalni element dovolj majhen. Navadno poddiagonalni element $h_{p+1,p}$ smatramo za 0, kadar je

$$h_{p+1,p} < \epsilon(|h_{pp})| + |h_{p+1,p+1}|.$$

Konvergenca QR iteracije in pomik izhodišča Z začetno redukcijo na Hessenbergovo obliko smo zmanjšali število operacij, ki so potrebne za vsak korak QR iteracije. Naslednje vprašanje, ki ga moramo rešiti, je hitrost konvergence, to je, kako hitro poddiagonalni elementi konvergirajo proti nič.

Naj bo A matrika, katere lastne vrednosti iščemo in jo na začetku reduciramo v Hessenbergovo matriko H. S QR iteracijo iz matrike H dobimo zaporedje Hessenbergovih matrik H_k z elementi $h_{ij}^{(k)}$. Pokazati se da (glej [11]), da poddiagonalni element $h_{i,i-1}^{(k)}$ konvergira proti nič kot

$$\left| \frac{\lambda_i}{\lambda i - 1} \right|.$$

Zato je konvergenco zelo počasna, kadar sta absolutni vrednosti dveh lastnih vrednosti blizu skupaj. Na srečo lahko konvergenco znatno pohitrimo s *pomikom izhodišča*.

Če je $\hat{\lambda}_i$ približek za lastno vrednost λ_i matrike H, lahko QR iteracijo naredimo na matriki

$$\hat{H} = H - \hat{\lambda}_i I.$$

Ker so lastne vrednosti matrike $\hat{H}$

$$\lambda_1 - \hat{\lambda}_i, \lambda_2 - \hat{\lambda}_i, \ldots, \lambda_n - \hat{\lambda}_i,$$

je hitrost, s katero $h_{i,i-1}^{(k)}$ konvergira proti nič določena z razmerjem

$$\left| \frac{\lambda_i - \hat{\lambda}_i}{\lambda_{i-1} - \hat{\lambda}_i} \right|,$$

kar je lahko znatno manj kot $|\lambda_i/\lambda_{i-1}|$.

QR iteracija z enojnim pomikom Da bi uporabili QR iteracijo s pomikom, moramo poznati približek $\hat{\lambda}_i$ lastne vrednosti λ_i. Ker tega ne poznamo, uporabimo za pomik kar zadnji element h_{nn}^k vsakokratne matrike H_k.

Algoritem 3.4. QR iteracija z enojnim pomikom Naj bo A realna kvadratna matrika reda n. Naslednji algoritem generira zaporedje matrik H_k, ki imajo vse iste lastne vrednosti kot A, ki konvergira proti zgornjetrikotni matriki ali Schurovi formi:

$H_0 = \mathbf{hess}(A)$ %matriko transformiramo na Hessenbergovo obliko
for $k = 0, 1, 2, \ldots$ dokler $h_{n,n-1}^{(k)}$ ni dovolj blizu 0
  $[Q_k, R_k] = \mathbf{qr}(H_k - h_{nn}^{(k)} * \mathbf{eye}(\mathbf{n}))$
  $H_{k+1} = R_k Q_k + h_{nn}^{(k)} * \mathbf{eye}(\mathbf{n})$
end

Kadar QR iteracija z enojnim pomikom konvergira, je konvergenca kvadratična.

QR iteracija z dvojnim pomikom Kadar so lastne vrednosti matrike realne, QR iteracija z enojnim pomikom deluje brez težav. Vendar ima realna matrika lahko tudi kompleksne lastne vrednosti. V takem primeru na neki stopmji QR iteracije naletimo na 2×2 matriko v desnem spodnjem kotu, ki ima kompleksno kojugiran par lastnih vrednosti λ_i in $\lambda_{i+1} = \bar{\lambda}_i$. v tem primeru element h_{nn} v desnem spodnjem kotu ne bo dober približek za lastno vrednost. V tem primeru bi lahko uporabili dvojni pomik, da bi uporabili obe lastne vrednosti:

$$\begin{aligned} H_k - \lambda_i I &= Q_k R_k & H_{k+1} &= R_k Q_k + \lambda_i I \\ H_{k+1} - \bar{\lambda}_i I &= Q_{k+1} R_{k+1} & H_{k+2} &= R_{k+1} Q_{k+1} + \bar{\lambda}_i I, \end{aligned}$$

vendar bi morali uporabiti kompleksno aritmetiko, čeprav je matrika H_k realna. Kompleksni aritmetiki pa se lahko izognemo, če oba koraka združimo. Pri tem ima ključno vlogo matrika

$$N = (H_k - \bar{\lambda}_i I)(H_k - \lambda_i I) = H_k^2 - (\lambda_i + \bar{\lambda}_i) H_k + \lambda_i \bar{\lambda}_i I.$$

Matrika N je realna, ker sta $\lambda_i + \bar{\lambda}_i$ in $\lambda_i \bar{\lambda}_i$ realni števili. Poleg tega sta $Q_k Q_{k+1}$ in $R_{k+1} R_K$ faktorja v QR razcepu matrike N, saj je

$$\begin{aligned} N &= (H_k - \bar{\lambda}_i I)(H_k - \lambda_i I) \\ &= (H_k - \bar{\lambda}_i I) Q_k R_k \\ &= Q_k^H Q_k (H_k - \bar{\lambda}_i I) Q_k R_k \\ &= Q_k (H_{k+1} - \bar{\lambda}_i I) R_k = Q_k Q_{k+1} R_{k+1} R_k. \end{aligned}$$

Tako lahko izračunamo H_{k+2} direktno iz matrike H_k, za λ_i in $\bar{\lambda}_i$ pa vzamemo lastni vrednosti matrike 2×2 v spodnjem desnem vogalu matrike H_k. Pri tem nam lastnih vrednosti sploh ni potrebno eksplicitno izra'cunati. Naj bo

$$\begin{bmatrix} h_{n-1,n-1} & h_{n-1,n} \\ h_{n,n-1} & h_{n,n} \end{bmatrix}$$

matrika v spodnjem desnem vogalu matrike H_k. Potem je

$$s = \lambda_i + \bar{\lambda}_i = h_{n-1,n-1} + h_{n,n}$$

sled matrike (realno število) in

$$d = \lambda_i \bar{\lambda}_i = h_{n-1,n-1} h_{n,n} - h_{n,n-1} h_{n-1,n}$$

determinanta matrike (tudi realno število). Algoritem za QR iteracijo z dvojnim pomikom lahko torej zapišemo kot

Algoritem 3.5. QR iteracija z eksplicitnim dvojnim pomikom Naj bo A realna kvadratna matrika reda n. Naslednji algoritem generira zaporedje matrik H_k, ki imajo vse iste lastne vrednosti kot A, ki konvergira proti zgornjetrikotni matriki ali Schurovi formi:

$H_0 = \mathbf{hess}(A)$ %matriko transformiramo na Hessenbergovo obliko
for $k = 0, 1, 2, \ldots$ dokler $h^{(k)}_{n,n-1}$ ni dovolj blizu 0
  $s = H_k(n-1, n-1) + H_k(n, n)$ % sled matrike
  $d = H_k(n-1, n-1)H_k(n, n) - H_k(n, n-1)H_k(n-1, n)$
    % determinanta matrike
  $N = H_k^2 - s * H_k + d * \mathbf{eye}(n)$
  $[Q_k, R_k] = \mathbf{qr}(N)$
  $H_{k+2} = Q_k^T H_k Q_k$
end

Po vsem tem lahko le z obžalovanjem pripomnimo, da zgoraj opisani QR algoritem z eksplicitnimdvojnim pomikom še ni primeren za praktično uporabo, saj že samo računanje matrike N zahteva $\mathcal{O}(n^3)$ operacij. Pomagamo si lahko z implicitnim dvojnim pomikom. Podrobnosti in celoten algoritem si bralec lahko ogleda v [8], [9], [11] ali [17].

3.5 Povzetek

V tem poglavju smo spoznali definicijo in osnovne lastnosti lastnih vrednosti in lastnih vektorjev kvadratne matrike. Pogledali smo tudi, kako lahko določimo približno lokacijo lastnih vrednosti s pomočjo prvega in drugega Geršgorinovega izreka.

Metode za računanje lastnih vrednosti lahko v grobem razdelimo v dve skupini: metode, ki računajo le eno lastno vrednost (in ustrezen lastni vektor) hkrati in metode, ki s podobnostnimi transformacijami matriko preoblikujejo v tako obliko, da brez težav razberemo vse lastne vrednosti hkrati. V prvo skupino sodita potenčna metoda, s katero lahko izračunamo tisto lastno vrednost, ki ima največjo absolutno vrednost, skupaj z usttreznim lastnim vektorjem, in inverzna iteracija, s katero lahko izračunamo poljubno lastno vrednost in ustrezni lastni vektor, če lae prav izberemo pomik.

V drugo skupino lahko uvrstimo metode, pri katerih s podobnostnimi transformacijami matrio preoblikujemo v tako obliko (trikotno, bločno triko-

tno, diagonalno ali bločno diagonalno), da lastne vrednosti lahko izračunamo brez večjih težav. Glavni predstavnik te skupine metod je QR iteracija.

3.6 Problemi

1. Naj bo
$$A = \begin{bmatrix} 3 & 0.3 & 0.2 \\ 0.1 & 5 & 0.4 \\ 0.3 & 0.2 & 7 \end{bmatrix}.$$
   (a) S pomočjo Geršgorinovega izreka ugotovi približne lastne vrednosti matrike A.
   (b) Ker imata A in A^T iste lastne vrednosti, nariši Geršgorinove diske še za matriko A^T.

2. Za Laplaceove matrika $A_n = [a_{ij}]$ z elementi
$$a_{ij} = \begin{cases} -2 & \text{za} \quad i = j \\ 1 & \text{za} \quad |i-j| = 1 \\ 0 & \text{sicer} \end{cases}$$
velikosti $n \times n$ za $n = 10, 20, \ldots, 100$ s potenčno metodo izračunaj po absolutni vrednosti največje lastne vrednosti in pripadajoče lastne vektorje.

3. Z inverzno iteracijo izračunaj vse lastne vrednosti in lastne vektorje matrike
$$\begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}!$$
Kako je potrebno izbrati primerne pomike za vsako lastno vrednost?

4. S pomočjo QR iteracije z eksplicitnim dvojnim pomikom izračunaj lastne vrednosti matrike
$$\begin{bmatrix} 1 & 2 & 3 \\ 1 & 0 & 1 \\ 0 & -2 & 2 \end{bmatrix}.$$

Poglavje 4

Reševanje nelinearnih enačb

Na iskanje rešitve enačbe oblike

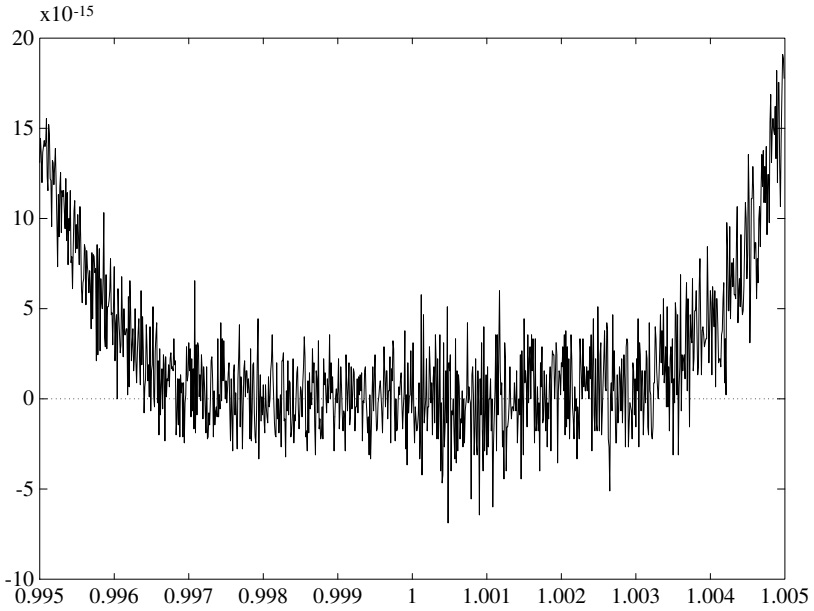
$$f(x) = 0 \tag{4.1}$$

zelo pogosto naletimo pri reševanju tehničnih problemov. Pri tem je lahko nelinearna funkcija f podana eksplicitno, pogosto pa je znan le postopek, kako pri določeni vrednosti neodvisne spremenljivke izračunamo njeno vrednost. Tako je $f(x)$ lahko vredost rešitve diferencialne enačbe v določeni točki ali vrednost nekega integrala. Nelinearno enačbo (4.1) lahko eksaktno rešimo le v zelo posebnih primerih, navadno se je moramo lotiti s kakšno od numeričnih metod. To pomeni, da ne bomo dobili točne vrednosti rešitve, ampak le njen približek. Kaj pomeni 'približek x^* k rešitvi enačbe' (4.1), pa je seveda odvisno od problema. Včasih je x^* taka točka, za katero je enačba (4.1) 'približno' izpolnjena, kar pomeni, da je $|f(x^*)|$ majhno število. Drugič pa je x^* 'blizu' točne rešitve enačbe (4.1). Poleg tega ima lahko enačba (4.1) več rešitev, večina numeričnih metod pa nam izračuna le eno do njih. Zgodi pa se tudi, da enačba (4.1) nima nobene rešitve

Pri numeričnem računanju ničle nelinearne funkcije se navadno ne moremo izogniti zaokrožitvenim napakam zaradi končne aritmetike, ki jo uporabljajo računalniki. Če izračunamo vrednosti polinoma

$$P_6(x) = 1 - 6x + 15x^2 - 20x^3 + 15x^4 - 6x^5 + x^6 \tag{4.2}$$

(slika 4.1) v bližini ničle $x = 1$ dobimo vtis, da ima ta polinom veliko število ničel na intevalu $(0.996, 1.004)$. Ta primer nam kaže, da ničle polinoma P_6 ne moremo določiti na več kot 2 mesti natančno, kljub temu, da smo vrednost



Slika 4.1: Izračunane vrednosti polinoma $(x-1)^6$ v bližini ničle

polinoma računali z natančnostjo 15 mest. Zaokrožitvena napaka je mnogo manjša, če ta polinom zapišemo v obliki

$$P_6(x) = (1-x)^6,$$

vendar moramo za tak zapis poznati vse ničle polinoma, torej vse ničle enačbe (4.1).

V tem poglavju si bomo ogledali metode za numerično reševanje nelinearnih enačb. Začeli bomo s preprostim problemom, ki nas privede do nelinearne enačbe, ob kateri si bomo ogledali nekaj splošnih lastnosti nelinearnih enačb. V naslednjih razdelkih bomo po vrsti opisali

1. Metodo bisekcije
2. Metodo regula falsi
3. Sekantno metodo
4. Newtonovo metodo

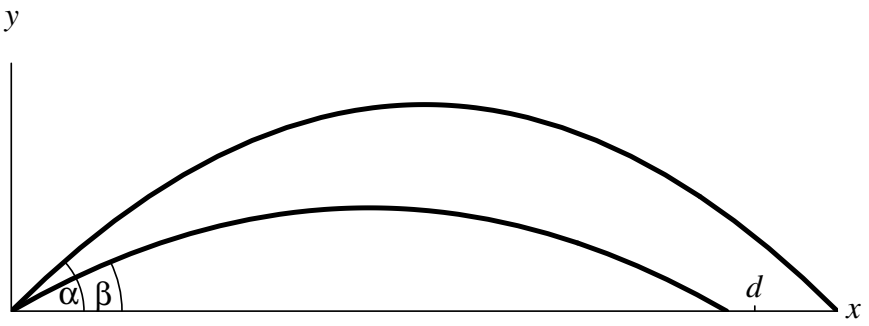
5. Kvazi-Newtonovo metodo

6. Metodo fiksne točke

Poglavje bomo končali s povzetkom najznačilnejših lastnosti opisanih metod.

4.1 Predstavitev problema

Za začetek si oglejmo preprost problem , pri katerem bomo spoznali nekaj najpogostejših problemov, s katerimi se lahko srečamo pri iskanju ničel nelinearnih enačb.



Slika 4.2: Streljanje s topom v tarčo

S topovsko kroglo želimo zadeti cilj na razdalji d od topa. Začetna hitrost izstrelka naj bo v_0, naklon cevi naj bo ϑ (glej sliko 4.2).

Začetna vertikalna hitrost granate je $v_0 \sin \vartheta$, iz fizike pa vemo, da je višina granate po času t enaka

$$y(t) = v_0 t \sin \vartheta - \frac{gt^2}{2},$$

kjer smo z g označili težnostni pospešek. Granata pade na tla po času

$$T = \frac{2v_0 \sin \vartheta}{g}.$$

Ker je vodoravna hitrost granate enaka $v_0 \cos \vartheta$ (zračni upor bomo zanemarili), bo domet granate enak $Tv_0 \cos \vartheta$. Da lahko določimo pravilen naklon

cevi, moramo torej rešiti enačbo

$$\frac{2v_0^2 \cos\vartheta \sin\vartheta}{g} = d,$$

oziroma

$$f(\vartheta) \equiv \frac{2v_0^2 \cos\vartheta \sin\vartheta}{g} - d = 0. \tag{4.3}$$

Ob tej enačbi si lahko ogledamo nekaj značilnosti, povezanih z nelinearnimi enačbami.

1. Enačba je poenostavljen model resničnega stanja. Pri izpeljavi smo upoštevali, da sta ustje topovske cevi in cilj v isti višini, kar velja le redko, zanemarili smo upor zraka in veter. Ko dobimo abstrakten numeričen problem, kot je enačba (4.3), se je koristno vprašati po njenem pomenu, še preden začnemo z numeričnim reševanjem.

2. Enačba morebiti sploh nima rešitve. Ker je produkt $\cos\vartheta \sin\vartheta$ lahko največ $\frac{1}{2}$ (pri $\vartheta = \frac{\pi}{4}$), enačba (4.3) za

   $$d > \frac{v_0^2}{g}$$

   nima nobene rešitve.

3. Rešitev, kadar obstaja, ni enolična, saj sta funkciji sin in cos periodični. Vsaka rotacija topovske cevi za poln krog spet pomeni rešitev. Pri reševanju se moramo zavedati tudi teh 'čudnih' rešitev.

4. Če je $d < v_0^2/g$ in $\vartheta_1 < \frac{\pi}{2}$, potem je tudi $\frac{\pi}{2} - \vartheta_1$ rešitev. Obe rešitvi sta smiselni, vendar je lahko za topničarja ena ugodnejša od druge. Ugotoviti moramo, katera.

5. Funkcija f je dovolj enostavna, da lahko izračunamo njen odvod, torej lahko uporabimo metodo, ki zahteva poleg poznavanja funkcije tudi poznavanje njenega odvoda (Newtonova metoda).

6. Enačbo (4.3) pravzaprav lahko rešimo tudi eksaktno, saj velja enakost $2\cos\vartheta \sin\vartheta = \sin 2\vartheta$. Le redko se zgodi, da nelinearno enačbo lahko rešimo eksaktno, vendar se splača poskusiti vsaj poenostaviti enačbo, preden jo začnemo reševati numerično.

7. Če naš model izpopolnimo, da bo bolje ustrezal dejanskemu problemu, npr. upoštevamo zračni upor, lahko dobimo sistem enačb, ki ga lahko rešimo le numerično. Funkcije postanejo tako zapletene, da ne moremo več dobiti analitičnega odvoda, torej moramo uporabiti takšno metodo, ki ne potrebuje vrednosti odvoda.

V praksi lahko topničar reši svoj problem z metodo poskusov in napak, tako da topovsko cev dviga ali spušča, dokler cilja ne zadane. Podobno delujejo tudi nekatere numerične metode.

4.2 Metoda bisekcije

Od sedaj naprej bomo reševali nelinearno enačbo (4.1). Metoda bisekcije sloni na naslednji lastnosti zvezne funkcije, ki jo že poznamo iz matematike:

Izrek 4.1. *Funkcija f naj bo zvezna na $[a, b]$ in naj bo g število med $f(a)$ in $f(b)$. Potem obstaja število $x \in [a, b]$, da je $g = f(x)$.*

Če lahko najdemo tak interval $[a, b]$, da je $f(a) \cdot f(b) < 0$, potem iz izreka 4.1 lahko sklepamo, da ima enačba (4.1) vsaj eno rešitev na intervalu $[a, b]$. Izračunamo jo s pomočjo naslednjega algoritma:

Algoritem 4.1. Bisekcija Naj bo f zvezna funkcija na intervalu $[a, b]$ in naj bo $f(a)f(b) < 0$, potem naslednji algoritem izračuna število c, ki se od ničle funkcije f razlikuje za manj kot ε_x ali pa je vrednost $|f(c)| < \varepsilon_y$.

$c = (a + b)/2$; $z = f(a)$; $y = f(c)$
while (**abs**$(y) > \varepsilon_y$)&$(b - a > \varepsilon_x)$
    if $y * z < 0$
        $b = c$
    else
        $a = c$; $z = y$
    end
    $c = (a + b)/2$
    $y = f(c)$
end

Pri vsakem prehodu skozi zanko algoritma se interval, na katerem iščemo ničlo funkcije f razpolovi, zato se po k prehodih dolžina intervala zmanjša na $(b-a)/2^k$. Ker zahtevamo, da se približek, izračunan z algoritmom Bisekcija razlikuje od točne ničle enačbe (4.1) za manj kot je vnaprej predpisana toleranca ε_x, se bo algoritem uspešno končal najmanj po

$$\log_2 \frac{b-a}{\varepsilon_x} + 1$$

korakih, če že prej nismo našli točke, v kateri je absolutna vrednost funkcije f manjša kot ε_y.

Ker je navadno časovno najzahtevnejši del opisanega algoritma računanje vrednosti funkcije f, velja omeniti, da moramo na vsakem koraku algoritma enkrat izračunati vrednost funkcije f.

Primer 4.1. Izračunajmo rešitev enačbe

$$f(x) \equiv x^3 + 2x^2 + 10x - 20 = 0 \tag{4.4}$$

s pomočjo bisekcije. Izberimo si $\varepsilon_x = 10^{-6}, \varepsilon_y = 10^{-5}$. Ker sta vrednosti $f(0) = -20 < 0$ in $f(2) = 16 > 0$, lahko vzamemo za začetni interval $[0, 2]$. Rezultati računanja so zbrani v tabeli 4.1.

4.3 Metoda regula falsi

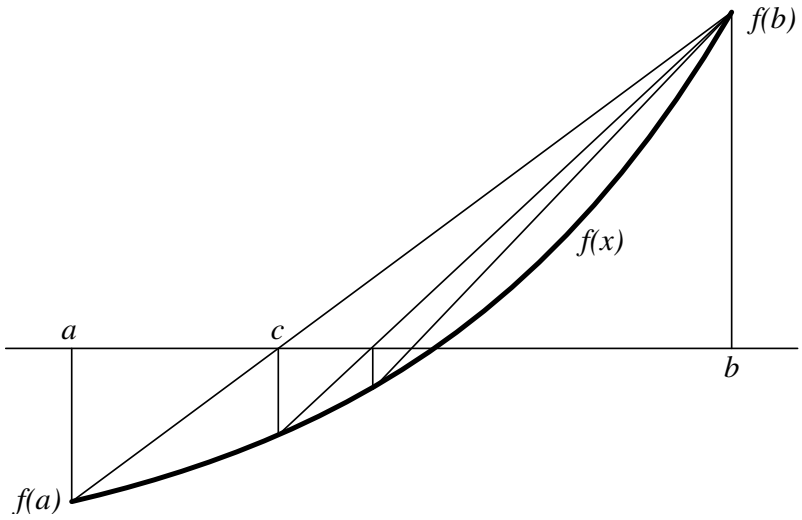
Pri metodi bisekcije interval na vsakem koraku razpolovimo, ne glede na to, kakšni sta vrednosti funkcije f na robovih intervala. Upamo lahko, da bomo ničlo funkcije hitreje našli, če bomo srednjo točko izbirali glede na vrednosti $f(a)$ in $f(b)$. Naj bo zopet $[a, b]$ tak interval, da je $f(a)f(b) < 0$, točko c pa si izberimo tam, kjer daljica skozi točki $T_0 = (a, f(a))$ in $T_1 = (b, f(b))$ seka abscisno os (glej sliko 4.3):

$$c = a - f(a)\frac{b-a}{f(b)-f(a)}. \tag{4.5}$$

Na ta način pridemo do algoritma, ki je znan pod imenom *Regula falsi*.

a	c	b	$f(c)$
0.000000	1.000000	2.000000	$-7.00 \cdot 10^{0}$
1.000000	1.500000	2.000000	$2.87 \cdot 10^{0}$
1.000000	1.250000	1.500000	$-2.42 \cdot 10^{0}$
1.250000	1.375000	1.500000	$1.30 \cdot 10^{-1}$
1.250000	1.312500	1.375000	$-1.16 \cdot 10+0$
1.312500	1.343750	1.375000	$-5.24 \cdot 10^{-1}$
1.343750	1.359375	1.375000	$-1.98 \cdot 10^{-1}$
1.359375	1.367187	1.375000	$-3.41 \cdot 10^{-2}$
1.367187	1.371093	1.375000	$4.82 \cdot 10^{-2}$
1.367187	1.369140	1.371093	$7.01 \cdot 10^{-3}$
1.367187	1.368164	1.369140	$-1.35 \cdot 10^{-2}$
1.368164	1.368652	1.369140	$-3.28 \cdot 10^{-3}$
1.368652	1.368896	1.369140	$1.86 \cdot 10^{-3}$
1.368652	1.368774	1.368896	$-7.10 \cdot 10^{-4}$
1.368774	1.368835	1.368896	$5.76 \cdot 10^{-4}$
1.368774	1.368804	1.368835	$-6.70 \cdot 10^{-5}$
1.368804	1.368820	1.368835	$2.54 \cdot 10^{-4}$
1.368804	1.368812	1.368820	$9.39 \cdot 10^{-5}$
1.368804	1.368808	1.368812	$1.34 \cdot 10^{-5}$
1.368804	1.368806	1.368808	$-2.67 \cdot 10^{-5}$
1.368806	1.368807	1.368808	$-6.64 \cdot 10^{-6}$

Tabela 4.1: Metoda bisekcije za enačbo $x^3 + 2x^2 + 10x - 20 = 0$



Slika 4.3: Metoda Regula falsi

Algoritem 4.2. Regula falsi Naj bo f zvezna funkcija na intervalu $[a, b]$ in naj bo $f(a)f(b) < 0$. Naslednji algoritem izračuna število c, ki se od ničle funkcije f razlikuje manj od $\varepsilon_x > 0$ ali pa je $|f(c)| < \varepsilon_y$.

$f_a = f(a)$; $f_b = f(b)$
$c = a - f_a * (b - a)/(f_b - f_a)$
$f_c = f(c)$
while (**abs**$(f_c) > \varepsilon_y$)&$(b - a > \varepsilon_x)$
    if $f_a * f_c < 0$
        $b = c$; $f_b = f_c$
    else
        $a = c$; $f_a = f_c$
    end
    $c = a - f_a * (b - a)/(f_b - f_a)$
    $f_c = f(c)$
end

Primer 4.2. Izračunajmo rešitev enačbe (4.4) še z metodo Regula falsi. Spet izberimo $\varepsilon_x = 10^{-6}, \varepsilon_y = 10^{-5}$ in začetni interval $[0, 2]$. Rezultati računanja so zbrani v tabeli 4.3.

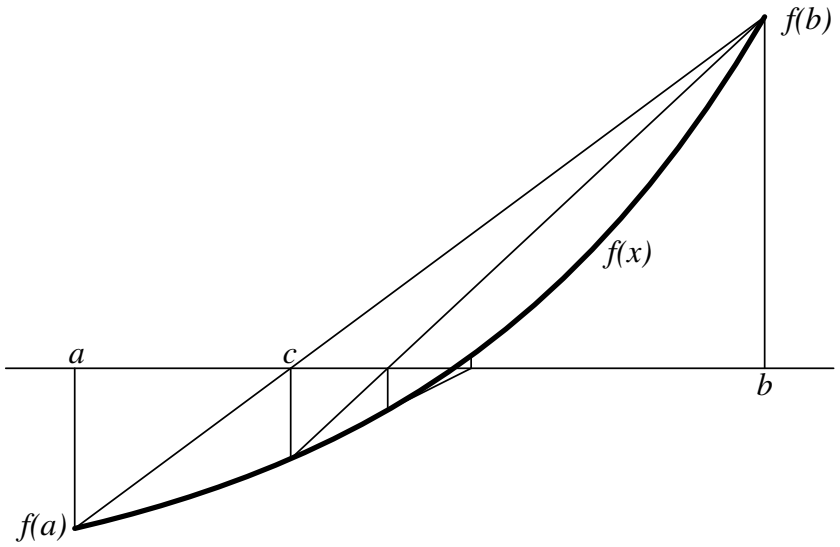
a	c	b	$f(c)$
1.111111	1.324296	2.000000	$-9.27 \cdot 10^{-1}$
1.111111	1.324296	2.000000	$-9.27 \cdot 10^{-1}$
1.324296	1.361301	2.000000	$-1.58 \cdot 10^{-1}$
1.361301	1.367547	2.000000	$-2.65 \cdot 10^{-2}$
1.367547	1.368596	2.000000	$-4.46 \cdot 10^{-3}$
1.368596	1.368772	2.000000	$-7.48 \cdot 10^{-4}$
1.368772	1.368802	2.000000	$-1.25 \cdot 10^{-4}$
1.368802	1.368807	2.000000	$-2.10 \cdot 10^{-5}$
1.368807	1.368807	2.000000	$-3.53 \cdot 10^{-6}$
1.368807	1.368808	2.000000	$-5.93 \cdot 10^{-7}$

Tabela 4.2: Metoda regula falsi za enačbo $x^3 + 2x^2 + 10x - 20 = 0$

V primerjavi z bisekcijo (tabela 4.1) lahko vidimo, da približki, dobljeni z metodo regula falsi hitreje konvergirajo proti ničli funkcije f. Opazimo lahko tudi, da se spreminja le levo krajišče intervala, desno pa ostaja ves čas pri $x = 2$, kar je posledica konveksnosti funkcije v bližini iskane rešitve.

Metoda regula falsi deluje podobno kot bisekcija: interval, na katerem leži ničla funkcije f sistematično zmanjšujemo, dokler ga ne zmanjšamo pod vnaprej predpisano dolžino ε_x. Ker se dolžina intervala pri bisekciji vedno zmanjša na polovico, lahko vnaprej ocenimo potrebno število korakov iteracije, pri reguli falsi pa tega ne moremo. Vseeno pa približki, ki jih dobimo s metodo regula falsi, večinoma konvergirajo proti ničli funkcije f hitreje kot približki, dobljeni z bisekcijo. Pri obeh metodah pa ves čas računanja poznamo interval, na katerem leži iskana ničla.

Hitrost konvergence regule falsi lahko povečamo, če se odpovemo kontroli predznaka funkcijske vrednosti v izračunanih točkah. S tem izgubimo interval, na katerem leži ničla, kar pomeni, da konvergenca zaporedja približkov ni več zagotovljena vnaprej. Tako dobimo *Sekantno metodo* (slika 4.4):



Slika 4.4: Sekantna metoda

Algoritem 4.3. Sekantna metoda Naj bo f zvezna funkcija. Če sta dani točki a in b, naslednji algoritem izračuna število c, za katerega je $|f(c)|$ manj od $\varepsilon_y > 0$, ali pa se po N korakih konča brez rezultata: $c = NaN$.

$x_s = a$; $f_s = f(x_s)$
$x_n = b$; $f_n = f(x_n)$
$n = 0$
while (**abs**$(f_n) > \varepsilon_y$)&$(n < N)$
    $c = x_s - f_s * (x_n - x_s)/(f_n - f_s)$
    $x_s = x_n$; $f_s = f_n$
    $x_n = c$; $f_n = f(c)$
    $n = n + 1$
end
if abs$(f_n) > \varepsilon_y$
    $c = NaN$
end

Primer 4.3. Izračunajmo rešitev enačbe (4.4) še s sekantno metodo. Spet izberimo $\varepsilon_y = 10^{-5}$ in začetni interval $[0, 2]$. Rezultati računanja so zbrani v tabeli **??**.

n	x_n	c	$f(c)$
1	2.000000	1.111111	$-5.04 \cdot 10^{1}$
2	1.111111	1.324296	$-9.27 \cdot 10^{-1}$
3	1.324296	1.372252	$7.27 \cdot 10^{-2}$
4	1.372252	1.368763	$-9.40 \cdot 10^{-4}$
5	1.368763	1.368808	$-9.37 \cdot 10^{-7}$

Tabela 4.3: Približki rešitve enačbe $x^3 + 2x^2 + 10x - 20 = 0$, dobljeni s sekantno metodo

V primerjavi z bisekcijo (tabela 4.1) in regulo falsi (tabela 4.3) lahko vidimo, da približki, dobljeni s sekantno metodo še hitreje konvergirajo proti ničli funkcije f.

Pri uporabi sekantne metode moramo biti previdni, saj se, posebej kadar smo še daleč od rešitve enačbe, prav lahko zgodi, da zaporedje približkov sploh ne konvergira.

4.4 Newtonova (tangentna) metoda

Enačbo (4.5), s katero izračunamo nov člen v zaporedju približkov z metodo regula falsi ali sekantno metodo, lahko zapišemo tudi kot

$$x_{n+1} = x_n + a_n,$$

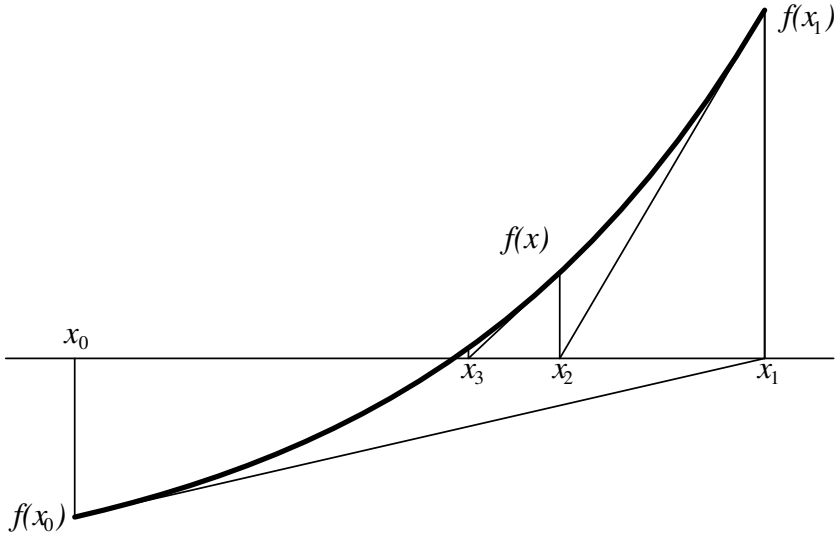
kjer je *popravek* a_n približka x_n določen kot

$$a_n = \frac{-f(x_n)}{c_n}.$$

Diferenčni kvocient $c_n = (f(x_n) - f(x_{n-1}))/(x_n - x_{n-1})$ v imenovalcu pomeni naklonski kot sekante skozi točki $T_{n-1} = (x_{n-1}, f(x_{n-1}))$ in $T_n = (x_n, f(x_n))$. Vemo tudi, da je v primeru, ko je funkcija f odvedljiva, diferenčni kvocient c_n enak odvodu $f'(x')$ v neki točki x' med x_n in x_{n-1}. Če diferenčni kvocient $(f(b) - f(a))/(b - a)$ v formuli (4.5) zamenjamo z vrednostjo odvoda v točki x_n, dobimo formulo

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}, \tag{4.6}$$

ki predstavlja osnovo *Newtonove*[1] ali *tangentne* metode. (slika 4.5)



Slika 4.5: Newtonova metoda

Algoritem 4.4. Newtonova metoda Naj bo f zvezno odvedljiva funkcija. Če je dana točka a, naslednji algoritem izračuna število c, za katerega je $|f(c)|$ manj od $\varepsilon_y > 0$, ali pa se po N korakih konča brez rezultata: $c = NaN$.

$c = a$; $z = f(c)$
$n = 0$
while (**abs**$(z) > \varepsilon_y$)&$(n < N)$
    $c = c - z/f'(c)$
    z = f(c)
    $n = n + 1$
end
if abs$(z) > \varepsilon_y$
    $c = NaN$
end

[1]Sir Isaac Newton (1643 Woolsthorpe – 1727 London), angleški matematik, fizik in astronom, eden najpomembnejših matematikov. Skupaj z Nemcem Gottfriedom Wilhelmom von Leibnizem velja za začetnika infinitezimalnega računa. Opis metode, ki je danes znana kot Newtonova je objavil leta 1987 v znameniti knjigi *Principia Mathematica*.

Primer 4.4. Izračunajmo rešitev enačbe (4.4) z Newtonovo metodo. Ponovno naj bo $\varepsilon_y = 10^{-5}$, začetna točka pa naj bo $x = 0$. Rezultati računanja so zbrani v Tabeli 4.4.

n	x_n	z
1	2.000000	$1.60 \cdot 10^{1}$
2	1.466666	2.12
3	1.371512	$5.70 \cdot 10^{-2}$
4	1.368810	$4.46 \cdot 10^{-5}$
5	1.368808	$2.73 \cdot 10^{-11}$

Tabela 4.4: Približki rešitve enačbe $x^3 + 2x^2 + 10x - 20 = 0$, dobljeni z Newtonovo metodo

Če to primerjamo s sekantno metodo (Tabela 4.3) lahko vidimo, da približki, dobljeni z Newtonovo metodo še hitreje konvergirajo proti ničli funkcije f.

V primerjavi s prej omenjenimi metodami, moramo na vsakem koraku Newtonove metode izračunati dve funkcijski vrednosti: $f(x_k)$ in $f'(x_k)$, kar lahko predstavlja hud problem, predvsem kadar

- je odvod f' težko izračunljiva funkcija ali
- je funkcija f podana z zapleteno formulo, tako da je velika verjetnost napake pri računanju odvoda in pri zapisu formule za izračun odvoda ali
- je funkcijska vrednost $f(x_k)$ rezultat daljšega numeričnega računanja. V tem primeru navadno odvoda sploh ne moremo izraziti v obliki formule.

Tem problemom se lahko deloma izognemo, če namesto formule (4.6) uporabimo formulo

$$x_{n+1} = x_n - \frac{f(x_n)}{d_n}, \tag{4.7}$$

kjer je d_k primeren, lahko izračunljiv približek za $f'(x_n)$. Enačba (4.7) predstavlja osnovo kvazi-Newtonove metode. Glede na to, kako izberemo približek d_n, je poznanih veliko kvazi-Newtonovih metod. Često uporabljana

je metoda, pri kateri za d_n vzamemo vrednost odvoda $d = f'(x_0)$ v začetni točki iteracije in računamo vse iteracije z isto vrednostjo d. Če po določenem številu iteracij ničle nismo našli, izračunamo novo vrednost odvoda in postopek ponovimo. Tako pridemo do naslednjega algoritma:

Algoritem 4.5. Kvazi-Newtonova metoda Funkcija f naj bo zvezno odvedljiva. Če je dano število a, naslednji algoritem izračuna število c, za katerega je $|f(c)|$ manj od $\varepsilon_y > 0$, ali pa se po N korakih konča brez rezultata: $c = NaN$.

$c = a$; $z = f(a)$; $d = f'(a)$
$n = 0$; $k = 0$
while (**abs**$(z) > \varepsilon_y$)&$(n < N)$
    $c = c - z/d$
    $z = f(c)$
    if $k == K$
        $d = f'(c)$; $k = 0$
    end
    $n = n + 1$; $k = k + 1$
end
if abs$(z) > \varepsilon_y$
    $c = NaN$
end

4.5 Metoda fiksne točke

Newtonovo metodo imamo lahko za poseben primer *metode fiksne točke*. Če je $f'(x) \neq 0$ na nekem intervalu in definiramo zvezno funkcijo

$$g(x) = x - \frac{f(x)}{f'(x)}, \tag{4.8}$$

potem lahko enačbo (4.6) zapišemo enostavneje kot

$$x_{n+1} = g(x_n). \tag{4.9}$$

Če zaporedje, definirano z rekurzijsko formulo (4.9), konvergira proti vrednosti ξ, potem je

$$\xi = \lim_{n\to\infty} x_{n+1} = \lim_{n\to\infty} g(x_n) = g(\xi),$$

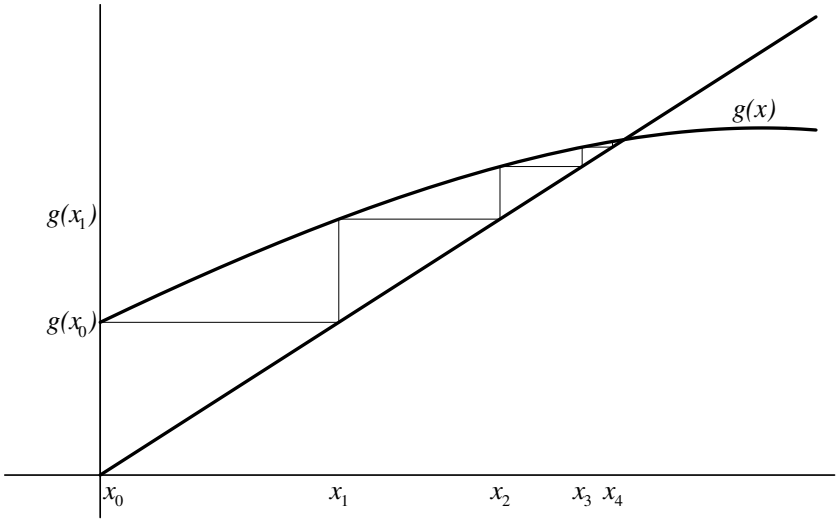
kar pomeni, da je ξ fixedpoint]*negibna (fiksna) točka* funkcije g. Očitno velja, da je vsaka negibna točka iteracijske funkcije Newtonove metode (4.8) tudi rešitev enačbe $f(x) = 0$.

Metodo fiksne točke dobimo tako, da dano enačbo $f(x) = 0$ najprej zapišemo zapišemo v ekvivalentno enačbo oblike

$$x = g(x).$$

To lahko storimo na mnogo načinov. Enačbo (4.4) lahko, poleg drugih ekvivalentnih oblik, zapišemo tudi kot

$$\begin{aligned}
&\text{a)} \quad g(x) = \frac{20 - 2x^2 - x^3}{10} \\
&\text{b)} \quad g(x) = \frac{20 + 10x - 2x^2 - x^3}{20} \\
&\text{c)} \quad g(x) = \frac{1}{2}\sqrt{20 - 10x - x^3} \\
&\text{d)} \quad g(x) = \sqrt[3]{20 - 10x - 2x^2} \\
&\text{e)} \quad g(x) = x - \frac{x^3 + 2x^2 + 10x - 20}{c} \qquad \text{za poljuben } c \neq 0.
\end{aligned} \tag{4.10}$$



Slika 4.6: Metoda fiksne točke

Ko imamo iteracijsko funkcijo g in začetni približek, lahko tvorimo zaporedje približkov po pravilu $x_{n+1} = g(x_n)$ (slika 4.6). Vprašati se moramo, kdaj tako dobljeno zaporedje (x_n) konvergira proti rešitvi enačbe (4.1). Brez dokaza (dokaz lahko radovedni bralec poišče na primer v [2]) navedimo *konvergenčni izrek*:

Izrek 4.2. *Naj bo funkcija g na intervalu $[a, b]$ zvezno odvedljiva, njene vrednosti naj zadoščajo pogoju*

$$a \leq x \leq b \quad \Rightarrow \quad a \leq g(x) \leq b,$$

za odvode pa naj velja

$$\sup_{x \in [a,b]} |g'(x)| = \lambda < 1.$$

Potem velja:

1. *Enačba $x = g(x)$ ima natanko eno rešitev ξ na intervalu $[a, b]$;*
2. *Za vsak začetni približek $x_0 \in [a, b]$ bo zaporedje x_n konvergiralo proti ξ;*
3. $$|\xi - x_n| \leq \frac{\lambda}{1 - \lambda} |x_n - x_{n-1}|;$$
4. $$|\xi - x_n| \leq \frac{\lambda^n}{1 - \lambda} |x_0 - x_1|;$$
5. $$\lim_{n \to \infty} \frac{\xi - x_{n+1}}{\xi - x_n} = g'(\xi),$$

   kar pomeni, da je v bližini rešitve

   $$\xi - x_{n+1} \approx g'(\xi)(\xi - x_n).$$

Ta izrek nam zagotavlja, da za vsak interval, na katerem je $|g'|$ manj od 1, zaporedje približkov, dobljeno s pravilom (4.9) vedno konvergira proti rešitvi

enačbe $x = g(x)$.

Algoritem 4.6. Metoda fiksne točke Dana naj bo zvezna funkcija g. Če je dana točka x, naslednji algoritem izračuna število c, za katerega je $|c - g(c)|$ manj od $\varepsilon_y > 0$, ali pa se po N korakih konča brez rezultata: $c = NaN$.

$n = 0$; $c = x$; $z = g(x)$
while (**abs**$(c - z) > \varepsilon_y$)&$(n < N)$
    $c = z$
    z = g(c)
    $n = n + 1$
end
if abs$(c - z) > \varepsilon_y$
    $c = NaN$
end

Primer 4.5. Izračunajmo rešitev enačbe (4.4) z metodo fiksne točke. Ponovno naj bo $\varepsilon_y = 10^{-5}$, začetna točka pa naj bo $x = 0$.
Najprej moramo izbrati primerno obliko iteracije. Od iteracijskih funkcij (4.10) so neprimerne a), c) in d), ker imajo v bližini ničle odvod, ki je po absolutni vrednosti večji kot 1 (glej problem 2). Iteracijska funkcija e) ima odvod, ki je po absolutni vrednosti manjši od 1 za $c < -0.6$, odvod iteracijske funkcije b) pa je blizu 0, zato izberemo iteracijo

$$x_{n+1} = \frac{20 + 10x_n - 2x_n^2 - x_n^3}{20}; \quad x_0 = 0. \tag{4.11}$$

Rezultati računanja so zbrani v tabeli 4.5.

Približki, dobljeni z iteracijo (4.11) konvergirajo proti ničli funkcije f nekoliko počasneje kot pri Newtonovi metodi. Pripomniti moramo, da je hitrost konvergence pri metodi fiksne točke tem večja, čim manjša je absolutna vrednost odvoda iteracijske funkcije v bližini ničle, kot napoveduje tudi izrek 4.2.

n	x_n	$f(x_n)$
0	0.000000	$-2.00 \cdot 10^{1}$
1	1.000000	$-7.00 \cdot 10^{0}$
2	1.350000	$-3.94 \cdot 10^{-1}$
3	1.369731	$+1.94 \cdot 10^{-2}$
4	1.368757	$-1.07 \cdot 10^{-3}$
5	1.368811	$+5.87 \cdot 10^{-5}$
6	1.368808	$-3.22 \cdot 10^{-6}$

Tabela 4.5: Približki rešitve enačbe $x^3 + 2x^2 + 10x - 20 = 0$, dobljeni z iteracijo (4.11)

4.6 Sistemi nelinearnih enačb

Splošen sistem n nelinearnih enačb z neznankami $x_1, x_2, \ldots, x_n$ lahko zapišemo kot

$$f_i(x_1, x_2, \ldots, x_n) = 0; \quad i = 1, 2, \ldots, n, \tag{4.12}$$

kjer so $f_1, f_2 \ldots, f_n$ funkcije n spremenljivk.

Podobno kot v razdelku 4.1 definiramo vektor neznank $\mathbf{x} = [x_1, \ldots, x_n]^T$, poleg tega pa vektorsko funkcijo n spremenjivk

$$\mathbf{f}(\mathbf{x}) = [f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_n(\mathbf{x})]^T,$$

pa lahko sistem enačb 4.12 zapišemo enostavno v obliki, ki je podobni enačbi (4.1)

$$\mathbf{f}(\mathbf{x}) = \mathbf{0}. \tag{4.13}$$

Za reševanje sistema nelinearnih enačb (4.13) lahko uporabimo nekatere metode za reševanje ene same nelinearne enačbe (4.1), na premer Newtonovo metodo in metodo fiksne točke.

4.6.1 Newtonova metoda za sisteme

Pri reševanju ene nelinearne enačbe (4.1) lahko do Newtonove metode pridemo tako, da $f(x + h)$ razvijemo v Tayloryevo vrsto po potencah h

$$f(x + h) = f(x) + f'(x)h + \mathcal{O}(h^2)$$

in zanemarimo člene višjega reda, skrite v $\mathcal{O}(h^2)$, nato pa rešimo *linearizirano enačbo*

$$0 = f(x) + f'(x)h$$

glede na h, od koder dobimo $h = -f(x)/f'(x)$, kar nam da naslednji, boljši približek

$$x + h = x - f(x)/f'(x).$$

Ko rešujemo sistem enačb (4.1), postopamo podobno: $\mathbf{f}(\mathbf{x}+\mathbf{h})$ razvijemo v Taylorjevo vrsto po potencah h

$$\mathbf{f}(\mathbf{x}+\mathbf{h}) = \mathbf{f}(\mathbf{x}) + \mathbf{f}'(\mathbf{x})\mathbf{h} + \mathcal{O}(||\mathbf{h}||^2),$$

kjer je $\mathbf{f}'$ Jacobijeva matrika, katere elementi so

$$[\mathbf{f}'(\mathbf{x})]_{ij} = \frac{\partial f_i}{\partial x_j}.$$

Tudi tokrat zanemarimo člene višjih redov $\mathcal{O}(||\mathbf{h}||^2)$ in rešio linearizirano enačbo

$$\mathbf{0} = \mathbf{f}(\mathbf{x}) + \mathbf{f}'(\mathbf{x})\mathbf{h}.$$

Zapišimo še algoritem:

Algoritem 4.7. Newtonova metoda za sisteme Naj bo $f : \mathbb{R}^n \to \mathbb{R}^n$ zvezno odvedljiva funkcija. Če je dana točka $a \in \mathbb{R}^n$, naslednji algoritem izračuna $c \in \mathbb{R}^n$, za katerega je $||f(c)||$ manj od $\varepsilon_y > 0$, ali pa se po N korakih konča brez rezultata: $c = NaN$.

$c = a$; $z = f(c)$
$n = 0$
while (**norm**$(z) > \varepsilon_y$)&$(n < N)$
    $J = f'(c)$
    $d = J \backslash z$
    $c = c - d$
    z = f(c)
    $n = n + 1$
end
if norm$(z) > \varepsilon_y$
    $c = NaN$
end

Delovanje Newtonove metode si oglejmo na primeru.

Primer 4.6. Izračunajmo presečiče krožnice $x^2 + y^2 = 2$ in parabole $y = x^2 + 1$.
Poiskati moramo torej ničlo funkcije

$$\mathbf{f}(x, y) = \begin{bmatrix} x^2 + y^2 - 2 \\ x^2 - y + 1 \end{bmatrix}.$$

Jacobijeva matrike v tem primeru je

$$J = \begin{bmatrix} 2x & 2y \\ 2x & -1 \end{bmatrix}.$$

Če si izberemo začetni približek $x^{(0)} = y^{(0)} = 1$ in predpišemo natančnost $\varepsilon_y = 10^{-14}$, dobimo zaporedje približkov, ki je v tabeli 4.6.
Opazimo lahko, da približki kvadrtično konvergirajo proti rešitvi sistema enačb.

n	$x^{(n)}$	$y^{(n)}$	$\|\|f(x^{(n)}, y^{(n)})\|\|$
0	1.00000	1.00000	1.0000
1	0.66667	1.33333	$2.4845 \cdot 10^{-1}$
2	0.56061	1.30303	$1.6570 \cdot 10^{-2}$
3	0.55035	1.30278	$1.4891 \cdot 10^{-4}$
4	0.55025	1.30278	$1.2939 \cdot 10^{-8}$
5	0.55025	1.30278	$2.2204 \cdot 10^{-16}$

Tabela 4.6: Zaporedje približkov in norma vrednosti funkcije $\mathbf{f}$ v primeru 4.6

.

4.6.2 Metoda fiksne točke za sisteme

Tudi pri reševanju sistemov nelinearnih enačb z metodo fiksne točke moramo enačbo (4.13) zapisati v ekvivalentni obliki

$$\mathbf{x} = \mathbf{g}(\mathbf{x}). \tag{4.14}$$

Izberemo si še začetni približek $\mathbf{x}^{(0)}$ k rešitvi, palahko generiramo zaporedje

$$\mathbf{x}^{(k+1)} = \mathbf{g}(\mathbf{x}^{(k)}), \qquad k = 0, 1, \ldots \tag{4.15}$$

in upamo, da zaporedje konvergira k fiksni točki funkcije $\mathbf{g}$.

Analiza metode fiksne točke za sisteme nelinearnih enačb je podobna kot v primeru ene same nelinearne enačbe (razdelek 4.5), glavna razlika je v tem, da napako $\mathbf{x}^{(k)} - \mathbf{x}$ merimo z normo namesto z absolutno vrednostjo [7].

Izrek 4.3. Naj bo $\mathbf{g}$ funkcija, ki zaprto množico S preslika vase (če je $\mathbf{x} \in S$, potem je tudi $\mathbf{g}(\mathbf{x}) \in S$). Poleg tega naj bo $\mathbf{g}$ na S skrčitev (to pomeni, da je za neko konstanto $C < 1$

$$||\mathbf{g}(\mathbf{x}) - \mathbf{g}(\mathbf{y})|| \leq C||\mathbf{x} - \mathbf{y}||$$

za vse $\mathbf{x}$ in $\mathbf{y}$ iz S). Potem velja

1. enačba $\mathbf{x} = \mathbf{g}(\mathbf{x})$ ima na S natanko eno rešitev;
2. Za vsak $\mathbf{x}^{(0)}$ zaporedje definirano z (4.15) konvergira proti rešitvi enačbe (4.14) tako, da je

$$||\mathbf{x} - \mathbf{x}^{(k)}|| \leq \frac{C}{1-C}||\mathbf{x}^{(k)} - \mathbf{x}^{(k-1)}|| \tag{4.16}$$

in

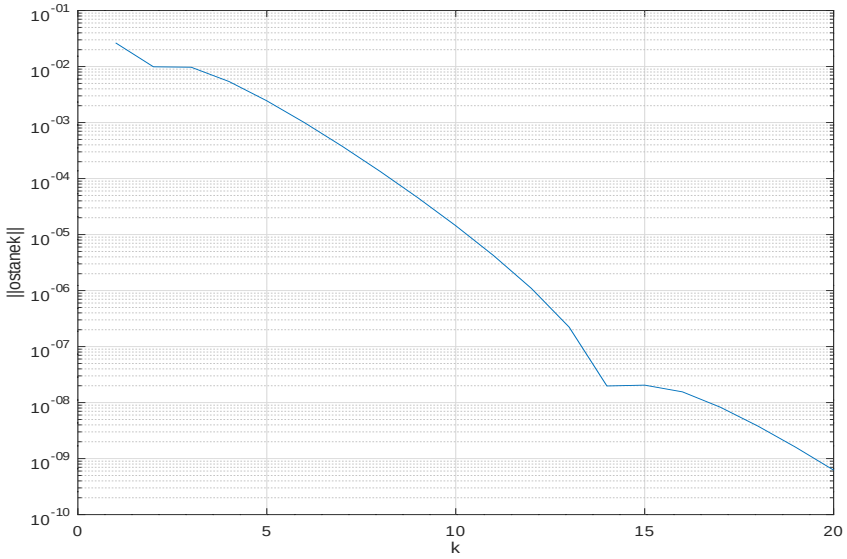
$$||\mathbf{x} - \mathbf{x}^{(k)}|| \leq \frac{C^k}{1-C}||\mathbf{x}^{(1)} - \mathbf{x}^{(0)}||. \tag{4.17}$$

Poglejmo, kako metodo fiksne točke lahko uporabimo na konkretnem primeru.

Primer 4.7. Sistem enačb iz priemra 4.6 rešimo še z metodo fiksne točke. Najprej sistem enačb zapišimo v obliki (4.14), na prmer kot

$$\begin{aligned} x &= x + c_1(x^2 + y^2 - 2) \\ y &= y + c_2(x^2 - y + 1). \end{aligned}$$

Če izberemo začetno vrednost $\mathbf{x}^{(0)} = [0.5, 1^T]$ ter konstanti $c_1 = 0.09$ in $c_2 = 1.2$,je za natančnost $\varepsilon = 10^{-9}$ potrebno 20 iteracij (glej sliko 4.7).



Slika 4.7: Ostanek ($||\mathbf{f}(\mathbf{x})||$) iz primera 4.7.

4.7 Povzetek

Povzemimo glavne značilnosti opisanih metod za reševanje nelinearnih enačb:

1. Za razliko od bisekcije, regule falsi in sekantne metode, pri katerih vedno potrebujemo dve točki, da izračunamo novi približek (zato jim pravimo dvočlenske metode), pri Newtonovi metodi in metodi fiksne točke potrebujemo le eno točko za izračun naslednjega približka. Zato pravimo, da sta Newtonova metoda in metoda fiksne točke enočlenski.

2. Pri bisekciji in pri reguli falsi imamo rešitev ves čas na omejenem intervalu, katerega dolžina se postopoma zmanjšuje, pri ostalih metodah med iteracijo nimamo zanesljive informacije o lokaciji rešitve, navadno je konvergenca približkov zagotovljena le v dovolj majhni okolici rešitve:

če je začetni približek daleč od tiste rešitve enačbe, ki jo iščemo, zaporedje približkov lahko divergira ali pa konvergira proti kakšni drugi rešitvi.

3. Od vseh omenjenih metod najpočasneje konvergira zaporedje približkov, dobljenih z bisekcijo, nekoliko hitreje zaporedje, dobljeno z regulo falsi, še hitreje zaporedje, dobljeno s sekantno metodo, najhitreje pa zaporedje, dobljeno z Newtonovo metodo. Pri metodi fiksne točke je konvergenca približkov tem hitrejša, čim manjša je absolutna vrednost odvoda iteracijske funkcije.

4. Kot splošna metoda za reševanje nelinearnih enačb se najbolje obnese Newtonova metoda. Kadar je težko ali časovno zahtevno izračunati odvod funkcije f, je namesto Newtonove metode bolje uporabiti sekantno metodo.

5. Newtonova in sekantna metoda zanesljivo in hitro konvergirata le v neposredni bližini rešitve, zato je velikokrat ugodno začeti z zanesljivejšo, čeprav počasnejšo metodo (bisekcijo ali regulo falsi), ko pa se ničli dovolj približamo, računamo dalje s hitrejšo metodo (Newtonova ali sekantna).

6. Z Newtonovo in s sekantno metodo lahko izračunamo tudi kompleksne rešitve enačb. Potrebujemo le kompleksni začetni približek in (seveda) računati moramo s kompleksnimi števili.

7. Sistem nelinearnih enačb lahko rešimo s pomočjo Newtonove metode ali metode fiksne točke.

4.8 Problemi

1. Število $\sqrt[n]{N}$ lahko izračunamo tako, da rešimo enačbo $x^n - N = 0$.

   (a) Izračunaj $\sqrt[3]{161}$ z metodo bisekcije. Kako izbrati začetni interval?

   (b) Izračunaj $\sqrt[4]{21.75}$ z metodo regula falsi. Kako izbrati začetni interval?

   (c) Izračunaj $\sqrt[5]{238.56}$ z Newtonovo metodo. Kaj bi bil primeren začetni približek?

(d) *Dodatek* Primerjaj učinkovitost vseh treh metod.

2. Izračunaj vrednost odvodov iteracijskih funkcij (4.10) v bližini ničle $x \approx 1.369$. Katera od iteracijskih funkcij ima v bližini ničle odvod z najmanjšo absolutno vrednostjo?

3. Rešujemo enačbo
$$x^3 - 5 = 0.$$
   (a) Kolikokrat moramo razpoloviti interval pri uporabi bisekcije, da dobimo $\sqrt[3]{5}$ z absolutno natančnostjo 10^{-9}, če je začetni interval $[0, 5]$?
   (b) Izračunaj naslednje tri približke s sekantno metodo, če sta prva dva 0 in 5!
   (c) Kako moramo izbrati parameter c, da bo iteracija $x_{n+1} = x_n + c(x_n^3 - 5)$ konvergirala proti $\sqrt[3]{5}$?
   (d) Na kakšnem intervalu moramo izbrati začetno vrednost, da bo Newtonova iteracija zanesljivo konvergirala? (Namig: Newtonovo iteracijo zapiši v obliki metode fiksne točke.)

4. Dan je polinom
$$p(x) = x^8 - 170x^6 + 7392x^4 - 39712x^2 + 51200.$$
   (a) Izračunaj ničle polinoma p.
   (b) Opiši, kako bi največjo ničlo izračunal z metodo fiksne točke.
   (c) Izračunaj po velikosti tretjo ničlo polinoma p_1, ki ga dobiš iz polinoma p tako, da koeficient pri x^2 spremeniš na -39710.
   (d) Kolikšna je pri spremembi polinoma iz p v p_1 relativna sprememba druge ničle?

5. Poišči tisti koren enačbe
$$y = \operatorname{tg} x,$$
   ki je najbližji 100.
   (a) Kako določiš začetni interval za bisekcijo? Izračunaj koren na dve decimalni mesti.

(b) Izberi iteracijsko funkcijo za metodo fiksne točke. Izračunaj koren na tri decimalna mesta.

(c) Kako izbrati začetno točko za Newtonovo metodo? Izračunaj koren še z Newtonovo metodo (na štiri decimalna mesta).

6. Rešujemo enačbo
$$x - \sqrt{1+x} = 0.$$

(a) Izračunaj vse njene ničle!

(b) Kakšen mora biti začetni interval za bisekcijo ali regulo falsi?

(c) Ali iteracija $x_{n+1} = \sqrt{1+x_n}$ konvergira proti rešitvi, če izberemo x_0 dovolj blizu rešitve?

(d) Za katere vrednosti c metoda fiksne točke
$$x_{n+1} = x_n - c(x_n^2 - x_n - 1)$$
konvergira proti rešitvi, če je x_0 dovolj blizu rešitve?

7. Naj bo funkcija f definirana na intervalu $(0, \frac{\pi}{2})$ s predpisom
$$f(x) = \begin{cases} \frac{12x}{\pi} - 2 & \text{za} \quad \frac{\pi}{6} \leq x < \frac{\pi}{4} \\ \frac{-12x}{\pi} + 4 & \text{za} \quad \frac{\pi}{4} \leq x < \frac{\pi}{3} \\ 0 & \text{sicer.} \end{cases}$$

Rešujemo enačbo $f(x) = \sin x$.

(a) Skiciraj graf. Koliko ničel ima enačba?

(b) Zapiši algoritem za izračun najmanjše ničle z Newtonovo metodo. Izračunaj ničlo.

(c) Kako bi največjo ničlo izračunal z metodo fiksne točke? Zapiši algoritem in izračunaj ničlo.

8. Nelinearno enačbo $f(x) = 0$ rešujemo z metodo trisekcije. To je varianta metode bisekcije, pri kateri interval $[a, b]$, na katerem leži ničla, vsakič razdelimo na tri enake podintervale.

(a) Zapiši algoritem za metodo trisekcije za primer, ko je na intervalu $[a, b]$ natanko ena ničla funkcije f.

(b) Koliko je največje potrebno število korakov, če želimo izračunati približek, ki se od prave vrednosti razlikuje manj kot ε_x?

(c) Primerjaj bisekcijo in trisekcijo glede na potrebno število izračunov vrednosti funkcije f.

(d) Metodo trisekcije uporabi pri reševanju enačbe $2x - 3 = 0$ z začetnim intervalom $[-1.2, 10/3]$ in natančnostjo $\varepsilon_x = 0.1$.

Poglavje 5

Interpolacija in aproksimacija funkcij

Na *interpolacijo* naletimo, kadar moramo vrednost funkcije, ki ima vrednosti znane le v posameznih točkah (pravimo jim *interpolacijske točke*), izračunati v kakšni točki, različni od interpolacijskih točk. Ker funkcije zunaj interpolacijskih točk ne poznamo, jo nadomestimo z *interpolacijsko funkcijo*. Da bi bila interpolacijska funkcija dober nadomestek za originalno funkcijo, (ki je lahko poznana analitično ali le v posameznih točkah) mora biti enostavno določljiva, lahko izračunljiva in njene vrednosti se morajo v predpisanih točkah ujemati z vrednostjo originalne funkcije. Navadno uporabljamo za aproksimacijo polinome, lahko pa tudi kakšne druge funkcije, na primer trigonometrične ali eksponentne funkcije.

Interpolacija je uporabna predvsem, kadar imamo funkcijo podano v obliki tabele, zanima pa nas vrednost funkcije v točki, ki leži med tabeliranimi vrednostmi. Nekoč je bil to zelo pomemben problem, danes pa so elektronski kalkulatorji in računalniki skoraj popolnoma izpodrinili uporabo tabeliranih funkcij. Kljub temu pa interpolacijo še vedno pogosto uporabljajo računalniki pri izračunu vrednosti funkcij, kot so trigonometrične, eksponentne, logaritemske in podobne.

V numerični matematiki pa je interpolacija pomembna tudi za numerično odvajanje in integriranje funkcij, kjer funkcije, tudi če so podane z analitično formulo, navadno nadomestimo z ustreznim interpolacijskim polinomom, ki ga laže integriramo ali odvajamo.

Tudi pri *aproksimaciji* skušamo dano funkcijo, od katere poznamo le vrednosti v nekaj točkah, nadomestiti z drugo, preprostejšo. Razlika je le v

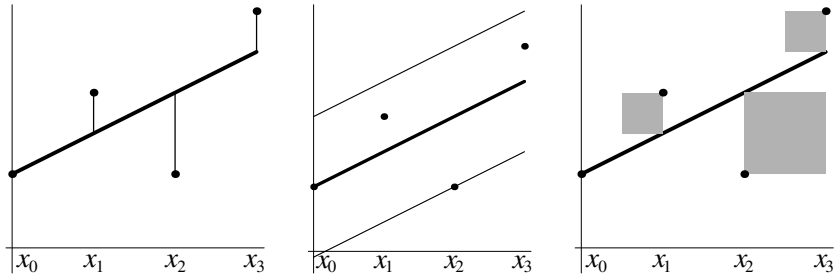
tem, da imamo navadno pri aproksimaciji znane vrednosti v večjem številu točk, vendar te vrednosti običajno niso brez napak, zato ne zahtevamo, da se mora aproksimacijska funkcija točno ujemati z danimi vrednostmi. Pri tem se moramo vprašati, kako meriti napako take aproksimacije. Odgovor na to vprašanje je seveda odvisen od tega, kaj želimo z aproksimacijo doseči. Poglejmo si nekaj tipičnih možnosti:

1. Funkcijo f imamo podano le v posameznih točkah x_i, $i = 1, \ldots, n$, kjer je število točk n lahko razmeroma veliko. Te vrednosti so pogosto le približne (rezultat meritev ali približno izračunane vrednosti). Ker je točk veliko, funkcijske vrednosti pa le približne, bi z interpolacijo dobili le slab približek. Aproksimacijska napaka e_i v posamezni točki je razlika med dano vrednostjo f_i in vrednostjo aproksimacijske funkcije v tej točki $g(x_i)$:
$$e_i = f_i - g(x_i),$$
mero za celotno napako E pa lahko izberemo na več načinov:

   (a) Celotna napaka je vsota absolutnih vrednosti napak v posameznih točkah
   $$E_s = \sum_{i=1}^{n} |e_i|.$$
   (glej sliko 5.1 (a).)

   (b) Celotna napaka je enaka največji izmed absolutnih vrednosti napak v posameznih točkah
   $$E_e = \max_i |e_i|.$$
   Aproksimaciji, pri kateri izbiramo aproksimacijsko funkcijo tako, da je E_e najmanjša, pravimo *(diskretna) enakomerna aproksimacija.* (glej sliko 5.1 (b).)

   (c) Celotna napaka je kvadratni koren vsote kvadratov napak v posameznih točkah
   $$E_{LSQ} = \sqrt{\sum_{i=1}^{n} e_i^2}.$$
   Aproksimaciji, pri kateri izbiramo aproksimacijsko funkcijo tako, da je E_{LSQ} najmanjša, pravimo *aproksimacija z metodo najmanjših kvadratov.* (glej sliko 5.1 (c).)



Slika 5.1: Različni kriteriji za aproksimacijo funkcij — (a) vsota absolutnih vrednosti napak; (b) enakomerna aproksimacija; (c) aproksimacija z metodo najmanjših kvadratov

2. Funkcija f je zvezna na intervalu $[a, b]$. Aproksimacijska napaka $e(x)$ v posamezni točki je razlika med vrednostjo funkcije $f(x)$ in vrednostjo aproksimacijske funkcije $g(x)$:

$$e(x) = f(x) - g(x),$$

mero za celotno napako E pa lahko podobno kot prej izberemo na več načinov:

(a) Celotna napaka je integral absolutne vrednosti napake

$$E_s = \int_a^b |e(x)|\, dx.$$

(b) Celotna napaka je enaka maksimumu absolutne vrednosti napake

$$E_e = \max_{x\in[a,b]} |e(x)|.$$

Aproksimaciji, pri kateri izbiramo aproksimacijsko funkcijo tako, da je E_e najmanjša, pravimo *najboljša* ali *enakomerna aproksimacija*.

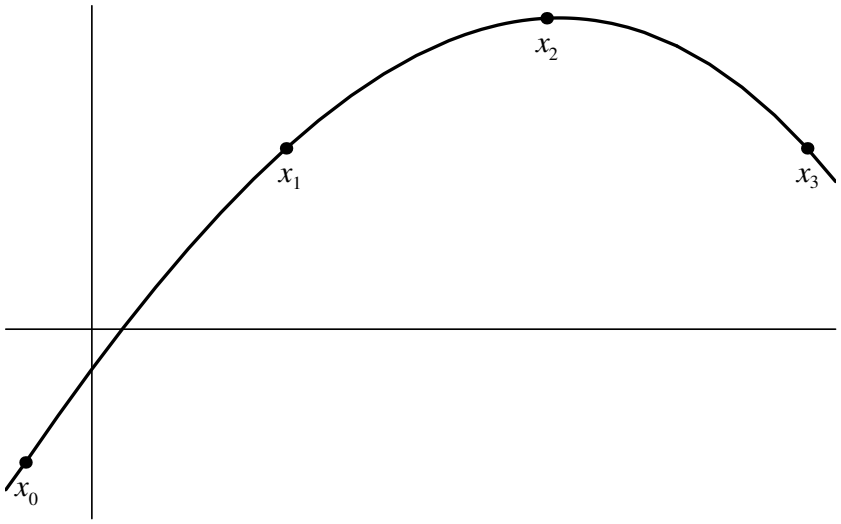
(c) Celotna napaka je kvadratni koren integrala kvadrata napake

$$E_{LSQ} = \sqrt{\int_a^b e^2(x)\, dx}.$$

Aproksimaciji, pri kateri izbiramo aproksimacijsko funkcijo tako, da je E_{LSQ} najmanjša, pravimo *(zvezna) aproksimacija z metodo najmanjših kvadratov*

V tem poglavju se bomo najprej ukvarjali z interpolacijo: formulirali bomo problem, s katerim se bomo ukvarjali v nadaljevanju in dokazali obstoj in enoličnost interpolacijskega polinoma. Posebej si bomo ogledali najprej Lagrangeovo, nato pa še Newtonovo obliko interpolacijskega polinoma, spotoma pa se bomo seznanili tudi z deljenimi diferencami ter s poenostavitvami interpolacijskih formul v primeru, ko so interpolacijske točke ekvidistantne. Spoznali bomo tudi, kakšna je napaka interpolacijskega polinoma. Od aproksimacijskih tehnik bomo spoznali polinomsko aproksimacijo z metodo najmanjših kvadratov in osnovne lastnosti ortogonalnih polinomov. Na koncu pa si bomo ogledali še aproksimacijo periodičnih funkcij z metodo najmanjših kvadratov.

5.1 Polinomska interpolacija



Slika 5.2: Interpolacijski polinom skozi 4 točke

Tipičen problem polinomske interpolacije lahko opišemo takole: Danih naj bo $n+1$ različnih točk $x_0, x_1, \ldots, x_n$ in funkcija f, definirana na intervalu, ki vsebuje te točke. Vrednost funkcije f v posamezni točki naj bo $f(x_i) = f_i$; $i = 0, \ldots, n$. Iščemo polinom $p(x)$ stopnje $\leq n$, da bo

$$p(x_i) = f_i; \quad i = 0, 1, \ldots, n. \tag{5.1}$$

Na sliki 5.2 je interpolacijski polinom, katerega graf poteka skozi štiri predpisane točke.

V načelu lahko koeficiente polinoma $p(x) = a_0 + xa_1 + \cdots + a_n x^n$ izračunamo iz sistema linearnih enačb

$$\begin{array}{ccccccccc} a_0 & + & x_0 a_1 & + & \cdots & x_0^n a_n & = & f_0 \\ a_0 & + & x_1 a_1 & + & \cdots & x_1^n a_n & = & f_1 \\ \vdots & & \vdots & & & \vdots & & \vdots \\ a_0 & + & x_n a_1 & + & \cdots & x_n^n a_n & = & f_n, \end{array} \tag{5.2}$$

z Gaussovo eliminacijo ali LU-razcepom, vendar se pokaže, da ima ta pristop nekaj bistvenih pomankljivosti in da obstajajo tudi boljše in ekonomičnejše metode:

- Reševanje sistema (5.2) zahteva $O(n^3)$ operacij. Kot bomo videli, obstajajo metode, ki zahtevajo le $O(n^2)$ operacij.
- Sistem (5.2) je lahko slabo pogojen, čemur se s primernejšo metodo lahko izognemo.
- Ko rešimo sistem (5.2), nimamo še nobene informacije o napaki interpolacijskega polinoma, medtem ko pri nekaterih drugih metodah dobimo obenem z interpolacijskim polinomom tudi oceno ali približek za njegovo napako.

Matrika sistema (5.2) ima posebno obliko, ki je poznana kot *Vandermondova matrika*. Njena determinanta je enaka produktu vseh možnih razlik

$$\det \begin{vmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^n \\ 1 & x_1 & x_1^2 & \cdots & x_1^n \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^n \end{vmatrix} = \prod_{\substack{i,j=0 \\ i>j}}^{n} (x_i - x_j),$$

torej je nesingularna natanko tedaj, kadar so vsi x_i; $i = 0, 1, \ldots, n$ med seboj različni. To ugotovitev lahko zapišemo kot

Izrek 5.1. Kadar so točke $x_0, x_1, \ldots, x_n$ vse med seboj različne, obstaja polinom p stopnje $\leq n$, ki zadošča pogojem (5.1) za vse vrednosti $f_0, f_1, \ldots, f_n$.

Pokažimo še, da je interpolacijski polinom s pogoji (5.1) enolično določen. Spomnimo se znanega izreka iz algebre:

Izrek 5.2. Če ima polinom stopnje n več kot n ničel, potem je identično enak 0.

Dokaz: Denimo, da imamo dva polinoma stopnje ne več kot n, ki oba zadoščata pogojem (5.1). Njuna razlika, ki je tudi polinom stopnje ne več kot n, ima vrednost 0 v $n+1$ točki x_i; $i = 0, 1, \ldots, n$, torej je identično enaka 0 in oba interpolacijska polinoma sta nujno enaka. S tem smo pokazali, da je interpolacijski polinom, ki zadošča pogojem (5.1) en sam.

5.2 Lagrangeova interpolacijska formula

Da bi lahko konstruirali interpolacijski polinom, ki ustreza pogojem (5.1), v Lagrangeovi obliki, rešimo najprej naslednjo pomožno nalogo: za dane, med seboj različne točke $x_0, x_1, \ldots, x_n$ konstruirajmo $n+1$ polinomov l_i; $i = 0, 1, \ldots, n$ (pravimo jim *Lagrangeovi*[1] *polinomi*), katerih stopnja naj ne bo več kot n, da bo i-ti polinom zadoščal pogojem

$$l_i(x_j) = \begin{cases} 0 & \text{za} \quad i \neq j, \\ 1 & \text{za} \quad i = j. \end{cases} \tag{5.3}$$

Ker mora imeti polinom l_i ničle v n točkah x_j; $j \neq i$ in njegova stopnja ne sme biti večja od n, ga lahko zapišemo kot produkt

$$l_i(x) = C_i \prod_{\substack{j=0 \\ i \neq j}}^{n} (x - x_j), \quad i = 0, \ldots, n,$$

kjer konstanto C_i določimo tako, da bo $l_i(x_i) = 1$, kar nam da

$$C_i = \frac{1}{\prod_{\substack{j=0 \\ i \neq j}}^{n} (x_i - x_j)}, \quad i = 0, \ldots, n.$$

[1] Joseph-Louis Lagrange (1736 Torino – 1813 Paris), francoski matematik. Sodeloval je pri ustanovitvi dveh znamenitih francoskih visokih šol, *École Polytechnique* in *École Normale*. S teorijo interpolacije se je ukvarjal med svojim bivanjem v Berlinu (1766–1787), ko je bil direktor matematičnega oddelka Berlinske akademije.

Tako vidimo, da polinome l_i, ki zadoščajo pogojem (5.3) lahko zapišemo kot

$$l_i(x) = \prod_{\substack{j=0 \\ i \neq j}}^{n} \frac{x - x_j}{x_i - x_j}, \quad i = 0, \ldots, n. \tag{5.4}$$

S tem smo našli polinome, ki zadoščajo pogojem (5.3).

S pomočjo Lagrangeovih polinomov l_i lahko zapišemo polinom stopnje ne več kot n, ki zadošča pogojem (5.1) kot linearno kombinacijo Lagrangeovih polinomov

$$p(x) = \sum_{i=0}^{n} f_i l_i(x). \tag{5.5}$$

Lahko se prepričamo, da je ravno p tisti polinom, ki zadošča pogojem (5.1).

Če pišemo $\Pi(x) = (x-x_0)(x-x_1)\cdots(x-x_n)$, lahko Lagrangeovo formulo (5.5, 5.4) zapišemo preprosto kot

$$p(x) = \Pi(x) \sum_{i=0}^{n} \frac{f_i}{(x - x_i)\Pi'(x_i)}.$$

Primer 5.1. Izračunajmo približek za $\cos 0.15$, če imamo funkcijo cos tabelirano v spodnji tabeli (prava vrednost $\cos 0.15 \approx 0.988771$).

x	$\cos x$
0.0	1.000000
0.1	0.995004
0.2	0.980066
0.3	0.955336

1. Začnimo z linearno interpolacijo. Vzemimo dve sosednji točki $x_1 = 0.1$ in $x_2 = 0.2$ ter izračunajmo oba Lagrangeva polinoma

$$l_1(x) = \frac{x - x_2}{x_1 - x_2} \quad \text{in} \quad l_2(x) = \frac{x - x_1}{x_2 - x_1},$$

katerih vrednosti pri $x = 0.15$ sta

$$l_1(0.15) = l_2(0.15) = 0.5,$$

tako da je linearni približek za $\cos 0.15$ enak

$$p_1(0.15) = \cos 0.1 \cdot l_1(0.15) + \cos 0.2 \cdot l_2(0.15) = 0.987535.$$

Absolutna napaka linearnega približka je torej enaka

$$e_1 = p_1(0.15) - \cos 0.15 \approx 0.001236.$$

2. Izračunajmo približek za $\cos 0.15$ s kvadratno interpolacijo skozi točke $x_1 = 0.1$, $x_2 = 0.2$ in $x_3 = 0.3$. Vrednosti ustreznih Lagrangeovih polinomov pri $x = 0.15$ so

$$l_1(0.15) = 0.375; \quad l_2(0.15) = 0.75; \quad l_3(0.15) = -0.125,$$

torej je kvadratni približek za $\cos 0.15$ enak

$$p_2(0.15) = \sum_{i=0}^{2} \cos x_i l_i(0.15) = 0.988759,$$

in njegova absolutna napaka

$$e_2 = p_2(0.15) - \cos 0.15 \approx -0.000012.$$

3. Izračunajmo še približek za $\cos 0.15$ s kubično interpolacijo skozi vse štiri točke $x_0 = 0$, $x_1 = 0.1$, $x_2 = 0.2$ in $x_3 = 0.3$ tabele. Najprej vrednosti Lagrangeovih polinomov pri $x = 0.15$

$$\begin{aligned} l_0(0.15) &= -0.0625; \quad l_1(0.15) = 0.5625, \\ l_2(0.15) &= 0.5625; \quad l_3(0.15) = -0.0625. \end{aligned}$$

Kubični približek za $\cos 0.15$ je torej

$$p_3(0.15) = \sum_{i=0}^{3} \cos x_i l_i(0.15) = 0.988768,$$

in njegova absolutna napaka

$$e_2 = p_3(0.15) - \cos 0.15 \approx -0.000003.$$

Iz primerjave rezultatov vidimo, da je napaka približka pada s stopnjo interpolacijskega polinoma.

Lagrangeova oblika interpolacijskega polinoma (5.5) je uporabna predvsem takrat, kadar vnaprej poznamo stopnjo interpolacijskega polinoma, vrednost interpolacijskega polinoma pa nas zanima le v kakšni posamezni točki.

Glavna pomanjkljivost Lagrangeove interpolacije je, da pri dodajanju novih interpolacijskih točk (da povečamo stopnjo interpolacijskega polinoma in s tem natančnost) ne moremo izkoristiti že izračunanih rezultatov, torej moramo vse Lagrangeove polinome (5.4) ponovno izračunati od začetka.

5.3 Newtonova interpolacijska formula

Že v 1. poglavju smo srečali polinom, zapisan v *Newtonovi obliki* (1.4). Kadar za parametre c_i; $i = 1, \ldots, n$ vzamemo interpolacijske točke x_i; $i = 0, 1, \ldots, n-1$, dobimo Newtonovo obliko interpolacijskega polinoma

$$\begin{aligned} p_n(x) &= a_0 + a_1(x - x_0) + a_2(x - x_0)(x - x_1) + \cdots + \\ &\quad a_n(x - x_0)(x - x_1) \cdots (x - x_{n-1}). \end{aligned} \tag{5.6}$$

Za vsako celo število k med 0 in n naj bo polinom $q_k(x)$ vsota prvih $k+1$ členov polinoma $p_n(x)$:

$$\begin{aligned} q_k(x) &= a_0 + a_1(x-x_0) + a_2(x-x_0)(x-x_1) + \cdots \\ &+ a_k(x-x_0)(x-x_1)\cdots(x-x_{k-1}). \end{aligned}$$

Ker imajo vsi preostali členi skupen faktor $(x-x_0)\cdots(x-x_k)$, lahko interpolacijski polinom $p_n(x)$ zapišemo v obliki

$$p_n(x) = q_k(x) + (x-x_0)\cdots(x-x_k)r(x),$$

kjer je $r(x)$ nek polinom. Opazimo, da je člen $(x-x_0)\cdots(x-x_k)r(x)$ enak nič v interpolacijskih točkah $x_0, x_1, \ldots, x_k$, kar pomeni, da mora biti $q_k(x)$ že sam interpolacijeki polinom skozi te točke, ker pa je ta en sam, mora biti $p_k(x) = q_k(x)$.

To nam omogoča, da Newtonov interpolacijski polinom (5.6) konstruiramo po členih kot zaporedje Newtonovih interpolacijskih polinomov p_0, $p_1, \ldots, p_n$, kjer polinom p_i stopnje $\leq i$ poteka skozi interpolacijske točke $x_0, x_1, \ldots, x_i$. Zaporedje gradimo tako, da dobimo polinom p_i z dodajanjem naslednjega člena polinomu p_{i-1}:

$$p_i(x) = p_{i-1}(x) + a_i(x-x_0)\cdots(x-x_{i-1}).$$

Iz tega tudi vidimo, da je koeficient a_i vodilni koeficient i-tega polinoma $p_i(x)$, torej je njegova vrednost odvisna le od interpolacijskih točk $x_0, x_1, \ldots, x_{i-1}$ in funkcijskih vrednosti v teh točkah. Imenujemo ga *i-ta deljena diferenca* v točkah $x_0, x_1, \ldots, x_i$ in ga zapišemo kot

$$a_i = f[x_0, x_1, \ldots, x_i].$$

Tako lahko zapišemo Newtonov interpolacijski polinom kot

$$p_n(x) = \sum_{i=0}^{n} f[x_0, x_1, \ldots, x_i] \prod_{j=0}^{i-1} (x - x_j). \tag{5.7}$$

Ker je

$$p_0(x) = f_0,$$

je

$$f[x_0] = f_0.$$

Prav tako zaradi

$$p_1(x) = f[x_0] + f[x_0, x_1](x - x_0)$$

velja

$$f[x_0, x_1] = \frac{f_1 - f_0}{x_1 - x_0}.$$

Naj bo $p_{k-1}(x)$ interpolacijski polinom stopnje ne več kot $k-1$ skozi točke x_j; $j = 0, 1, \ldots, k-1$ in naj bo $r_{k-1}(x)$ interpolacijski polinom stopnje ne več kot $k-1$ skozi točke x_j; $j = 1, \ldots, k$. Potem je

$$\frac{x - x_0}{x_k - x_0} r_{k-1}(x) + \frac{x_k - x}{x_k - x_0} p_{k-1}(x) \tag{5.8}$$

polinom stopnje $\leq k$, ki se v interpolacijskih točkah $x_0, x_1, \ldots, x_k$ ujema s predpisani vrednostmi, torej je zaradi enoličnosti enak interpolacijskemu polinomu $p_k(x)$. Zato je vodilni koeficient polinoma $p_k(x)$, ki smo ga pisali kot $f[x_0, x_1, \ldots, x_k]$, enak

$$\begin{aligned} &f[x_0, x_1, \ldots, x_k] = \\ &\quad \frac{\text{vodilni koeficient } r_{k-1}}{x_k - x_0} - \frac{\text{vodilni koeficient } p_{k-1}}{x_k - x_0} = \\ &\quad \frac{f[x_1, \ldots, x_k] - f[x_0, \ldots, x_{k-1}]}{x_k - x_0}. \end{aligned}$$

S tem smo dokazali tudi rekurzivno pravilo za računanje deljenih diferenc višjih redov

$$f[x_0, \ldots, x_k] = \frac{f[x_1, \ldots, x_k] - f[x_0, \ldots, x_{k-1}]}{x_k - x_0}. \tag{5.9}$$

Deljena diferenca k-tega reda je torej diferenčni kvocient deljenih diferenc reda $k-1$. S pomočjo pravila (5.9) lahko sestavimo *tabelo deljenih diferenc* (glej tabelo 5.1).

Ker so deljene diference ravno koeficienti Newtonovega interpolacijskega polinoma (5.7), zapišemo algoritem, ki nam izračuna tabelo deljenih diferenc in vrednosti interpolacijskega polinoma.

x_i	f_i	$f[\cdot,\cdot]$	$f[\cdot,\cdot,\cdot]$	$f[\cdot,\cdot,\cdot,\cdot]$	$f[\cdot,\cdot,\cdot,\cdot,\cdot]$
x_0	f_0				
		$f[x_0,x_1]$			
x_1	f_1		$f[x_0,x_1,x_2]$		
		$f[x_1,x_2]$		$f[x_0,x_1,x_2,x_3]$	
x_2	f_2		$f[x_1,x_2,x_3]$		$f[x_0,x_1,x_2,x_3,x_4]$
		$f[x_2,x_3]$		$f[x_1,x_2,x_3,x_4]$	
x_3	f_3		$f[x_2,x_3,x_4]$		
		$f[x_3,x_4]$			
x_4	f_4				

Tabela 5.1: Tabela deljenih diferenc

Algoritem 5.1. Newtonov interpolacijski polinom Pri danih vektorjih neodvisnih spremenljivk x_i; $i = 1,\ldots,n+1$ in funkcijskih vrednosti f_i; $i = 1,\ldots,n+1$ naslednji algoritem izračuna koeficiente Newtonovega interpolacijskega polinoma in njegovo vrednost v točki t.

$a = \mathbf{zeros}(n+1, n+1)$
$a(:,1) = f$
$p = a(1,1)$
$u = 1$
for $i = 2 : n+1$
    $u = u * (t - x(i-1))$
    for $j = 2 : i$
        $a(i,j) = (a(i,j-1) - a(i-1,j-1))/(x(i) - x(i-j+1))$
    end
    $p = p + a(i,i) * u$
end

Preštejmo operacije, ki so potrebne za izračun vrednosti interpolacijskega polinoma z algoritmom 5.1:

$$\sum_{i=2}^{n+1}(2 + \sum_{j=2}^{i} 1) = \frac{n^2+5n}{2}.$$

x	$\cos x$			
0.0	1.000000			
		-0.04996		
0.1	0.995004		-0.4971	
		-0.14938		0.025
0.2	0.980066		-0.4896	
		-0.24730		
0.3	0.955336			

Tabela 5.2: Tabela deljenih diferenc za funkcijo cos

Primer 5.2. Približek za $\cos 0.15$, ki smo ga v primeru 5.1 iračanal z Lagrangeovim polinomom, izračunajmo še v Newtonovi obliki.

1. Iz tabele deljenih diferenc (tabela 5.2) preberemo koeficiente linearnega interpolacijskega polinoma skozi točki x_0 in x_1:

$$p_1(t) = 1 - t \cdot 0.04996, \tag{5.10}$$

zato

$$p_1(0.15) = 1 - 0.15 \cdot 0.04996 = 0.992506.$$

Njegova napaka je

$$e_1 = p_1(0.15) - \cos 0.15 = 0.003735.$$

2. Ko dodamo točko x_2, moramo približku (5.10) dodati naslednji člen:

$$p_2(t) = p_1(t) - 0.4971(t - x_0)(t - x_1) \quad \text{in} \quad p_2(0.15) = 0.988778. \tag{5.11}$$

Napaka tega približka je

$$e_2 = p_2(0.15) - \cos 0.15 = 0.000007.$$

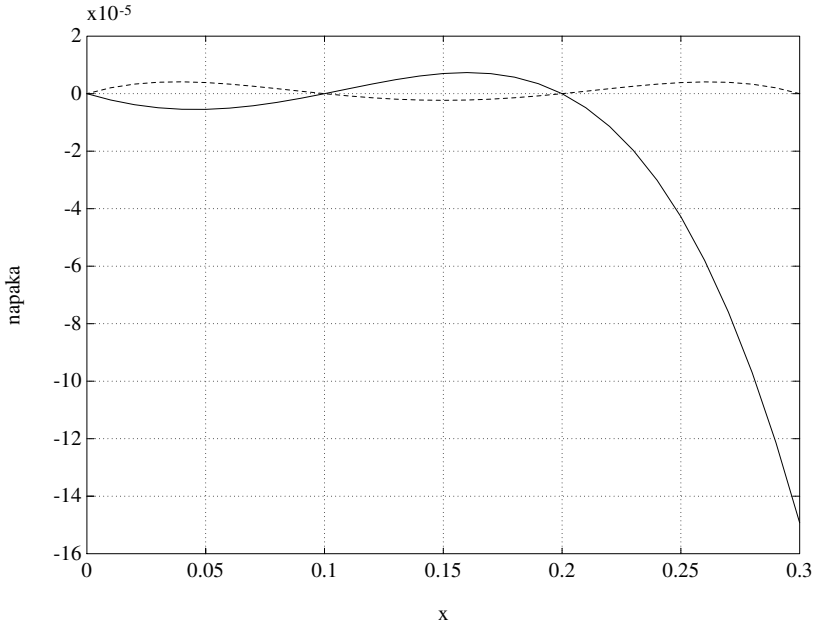
3. Za kubični približek moramo dodati še četrto točko x_3, kar pomeni, da približku (5.11) dodamo naslednji člen

$$p_3(t) = p_2(t) + 0.025(t - x_0)(t - x_1)(t - x_2) \tag{5.12}$$

$$p_3(0.15) = 0.988769.$$

Napaka kubičnega približka je

$$e_3 = p_3(0.15) - \cos 0.15 = -0.000002.$$



Slika 5.3: Napaka interpolacijskega polinoma p_2 (5.11) in p_3 (5.12)

Tudi v tem primeru lahko ugotovimo, da napaka polinomske aproksimacije pada, ko raste število interpolacijskih točk. Kako se obnaša napaka interpolacijskih polinomov p_2 (5.11) in p_3 (5.12) na celem intervalu $[0, 0.3]$, pa si lahko ogledamo na sliki 5.3.

5.4 Interpolacija ekvidistantnih tabel

Lagrangeova in Newtonova oblika interpolacijskega polinoma sta uporabni za interpolacijo funkcij, ki so podane v poljubnih točkah, ki niso nujno enako oddaljene med seboj. Pogosto pa imamo opravka z interpolacijo, pri kateri so interpolacijske točke med seboj enako oddaljene. V tem primeru lahko računanje interpolacijskih polinomov nekoliko poenostavimo, kar si bomo ogledali v tem razdelku.

Neodvisna spremenljivka naj bo na intervalu $[a, b]$ podana v točkah

$$x_i = a + ih, \quad i = 0, 1, \ldots, N, \quad N = \frac{b-a}{h}.$$

Vpeljimo novo neodvisno spremenljivko

$$s = \frac{x-a}{h}; \quad \text{da je} \quad x = a + hs. \tag{5.13}$$

S tako zamenjavo postanejo interpolacijske točke $s = 0, 1, \ldots, N$. Posebej moramo poudariti, da je zamenjava spremenljivke (5.13) linearna, zato polinom stopnje n tudi v novi spremenljivki ostane polinom iste stopnje. Zaradi enostavnosti bomo funkcijske vrednosti v interpolacijskih točkah pisali kot $f_i = f(x_i) = f(a + ih)$.

Začnimo z Lagrangeovo obliko interpolacijskega polinoma. Brez težav vidimo, da lahko Lagrangeove polinome (5.4) zapišemo kot

$$l_i(s) = \prod_{\substack{j=0 \\ j \neq i}}^{n} \frac{s-j}{i-j},$$

interpolacijski polinom pa kot njihovo linearno kombinacijo

$$p(s) = \sum_{i=0}^{n} f_i l_i(s).$$

Le malo dalj se bomo pomudili pri Newtonovi obliki interpolacijskega polinoma. Pri ekvidistantnih interpolacijskih točkah bomo namesto tabele deljenih diferenc izračunali raje *tabelo direktnih diferenc*. V ta namen definirajmo *direktne diference*

$$\Delta^i f_j = \begin{cases} f_j; & i = 0 \\ \Delta(\Delta^{i-1} f_j) = \Delta^{i-1} f_{j+1} - \Delta^{i-1} f_j; & i > 0. \end{cases} \tag{5.14}$$

Lahko se je prepričati, da velja med deljenimi in direktnimi diferencami zveza

$$f[x_k, \ldots, x_{k+i}] = \frac{\Delta^i f_k}{i! h^i}. \tag{5.15}$$

Tako lahko interpolacijski polinom skozi točke $x_0, \ldots, x_n$ v Newtonovi obliki (5.7) zapišemo kot

$$p_n(x) = \sum_{i=0}^{n} \frac{\Delta^i f_0}{i! h^i} \prod_{j=0}^{i-1} (x - x_j).$$

Če upoštevamo, da je

$$x - x_k = a + sh - (a + kh) = (s-k)h,$$

lahko zapišemo

$$p_n(x) = p_n(a + sh) = \sum_{i=0}^{n} \Delta^i f_0 \prod_{j=0}^{i-1} \frac{s-j}{j+1}. \tag{5.16}$$

Formulo (5.16) lahko še poenostavimo, če uporabimo *binomsko formulo*

$$\binom{y}{i} = \begin{cases} 1; & i = 0 \\ \prod_{j=0}^{i-1} \frac{y-j}{j+1}; & i > 0. \end{cases} \tag{5.17}$$

Končno lahko zapišemo Newtonovo obliko interpolacijskega polinoma skozi ekvidistantne točke kot

$$p_n(a + sh) = \sum_{i=0}^{n} \Delta^i f_0 \binom{s}{i}. \tag{5.18}$$

5.5 Napaka polinomske interpolacije

Do sedaj smo gledali na vrednosti $f_0, \ldots, f_n$ kot poljubna števila. Za analizo napake interpolacijskega polinoma pa je smiselno privzeti, da obstaja taka funkcija f, definirana na intervalu, ki vsebuje vse interpolacijske točke, z vrednostmi

$$f_i = f(x_i); \quad i = 0, 1, \ldots, n$$

in je dovoljkrat odvedljiva za naše potrebe.

Naj bo p_n interpolacijski polinom skozi točke $x_0, \ldots, x_n$. Ker navadno računamo interpolacijski polinom v upanju, da se bo dobro ujemal s funkcijo f, se je smiselno vprašati, kakšna je napaka interpolacijskega polinoma

$$e(t) = p_n(t) - f(t). \tag{5.19}$$

Seveda je v tej enačbi smiselno izbrati vrednost spremenljivke t različno od točk $x_0, \ldots, x_n$, saj vemo, da napake v interpolacijskih točkah ni.

Naj bo $p_n(t)$ interpolacijski polinom skozi točke $x_0, \ldots, x_n$. Da bi našli izraz za napako interpolacijskega polinoma (5.19), naj bo $q_{n+1}(u)$ interpolacijski polinom skozi točke $x_0, \ldots, x_n$ in t, ki ga v Newtonovi obliki lahko zapišemo kot

$$q_{n+1}(u) = p_n(u) + f[x_0, \ldots, x_n, t]\omega(u), \tag{5.20}$$

kjer smo z $\omega(u)$ označili produkt

$$\omega(u) = (u - x_0) \cdot \cdots \cdot (u - x_n).$$

Ker q_{n+1} interpolira f tudi v točki t, je $q_{n+1}(t) = f(t)$, zato iz (5.20) dobimo

$$f(t) = p_n(t) + f[x_0, \ldots, x_n, t]\omega(t),$$

oziroma, če preuredimo

$$e(t) = p_n(t) - f(t) = -f[x_0, \ldots, x_n, t]\omega(t), \tag{5.21}$$

to pa je že izraz za napako, ki smo ga iskali. Njegova slabost je v tem, da z njim ne moremo dobiti uporabne ocene za napako, saj ne poznamo deljene diference $f[x_0, \ldots, x_n, t]$. Da bi dobili uporabnejšo oceno, si oglejmo funkcijo

$$\varphi(u) = f(u) - p_n(u) - f[x_0, \ldots, x_n, t]\omega(u). \tag{5.22}$$

Njena vrednost v interpolacijskih točkah x_i; $i = 0, \ldots, n$ je enaka 0:

$$\varphi(x_i) = f(x_i) - p_n(x_i) - f[x_0, \ldots, x_n, t]\omega(x_i) = 0,$$

pa tudi

$$\varphi(t) = f(t) - p_n(t) - f[x_0, \ldots, x_n, t]\omega(t) = 0.$$

Če je I najmanjši interval, ki vsebuje vse točke x_i; $i = 0, \ldots, n$ in točko t, ima φ na I najmanj $n+2$ ničli. Ker smo privzeli, da je funkcija f odvedljiva tolikokrat, kot potrebujemo, p_n in ω pa sta polinoma, torej oba neskončnokrat odvedljiva, lahko sklepamo takole: po Rollejevem izreku ima odvod φ' ničlo vedno med dvema zaporednima ničlama funkcije φ, torej ima φ' na I vsaj $n+1$ ničlo. Podobno ima φ'' najmanj n ničel na I. Če tako nadaljujemo, vidimo, da mora imeti $n+1$-vi odvod $\varphi^{(n+1)}$ vsaj eno ničlo na I. Naj bo torej ξ ena izmed ničel funkcije $\varphi^{(n+1)}$ na intervalu I.

Ker je p_n polinom stopnje n, je njegov $n+1$-vi odvod enak 0, ω pa je polinom stopnje $n+1$ z vodilnim členom 1, torej $\omega(u) = u^{n+1} + \cdots$, zato je

$$\omega^{(n+1)}(\xi) = (n+1)!$$

Če to vstavimo v enačbo (5.22), dobimo

$$0 = \varphi^{(n+1)}(\xi) = f^{(n+1)}(\xi) - f[x_0, \ldots, x_n, t](n+1)!,$$

oziroma, po preureditvi

$$f[x_0, \ldots, x_n, t] = \frac{f^{(n+1)}(\xi)}{(n+1)!}. \tag{5.23}$$

Ko to vstavimo v enačbo (5.21), dobimo naslednji rezultat:

Izrek 5.3. Naj bo funkcija f vsaj $n+1$-krat zvezno odvedljiva in naj bo p_n interpolacijski polinom stopnje $\leq n$ skozi točke x_i; $i = 0, \ldots, n$. Potem je

$$f(t) - p_n(t) = \frac{f^{(n+1)}(\xi)}{(n+1)!}(t - x_0) \cdots (t - x_n), \tag{5.24}$$

kjer ξ leži v najmanjšem intervalu, ki vsebuje točke x_i; $i = 0, \ldots, n$ in t.

Za trenutek se ustavimo še ob enačbi (5.23), iz katere razberemo, da je n-ta deljena diferenca sorazmerna n-temu odvodu v neki točki na intervalu, ki vsebuje vse točke, ki smo jih uporabili za računanje deljene diference. Če točke x_i; $i = 0, \ldots, n$ izberemo dovolj blizu točke t, lahko iz n-te deljene diference dobimo dober približek za n-ti odvod funkcije v točki t.

Kako torej lahko ocenimo napako interpolacije? Oglejmo si to na primeru:

Primer 5.3. Naj bo $p_1(t)$ linearni polinom, ki interpolira funkcijo f v točkah x_0 in x_1 (naj bo $x_0 < x_1$). Funkcija f naj ima omejen drugi odvod, kar pomeni, da obstaja število M, da je

$$|f''(t)| < M$$

na nekem intervalu, na katerem nas interpolacija zanima, in ki vsebuje točki x_0 in x_1. Ocena napake interpolacije je odvisna od tega, ali točka t leži na intervalu $[x_0, x_1]$ ali zunaj njega.

1. Kadar je $t \in [x_0, x_1]$, pravimo, da gre za *interpolacijo v ožjem smislu.* Iz ocene (5.24) dobimo

$$|f(t) - p_1(t)| = \frac{|f''(\xi)|}{2}|(t - x_0)(t - x_1)| \leq \frac{M}{2}|(t - x_0)(t - x_1)|.$$

   Ker funkcija $|(t - x_0)(t - x_1)|$ doseže v točki $(x_0 + x_1)/2$ maksimalno vrednost $(x_1 - x_0)^2/4$, lahko napako interpolacije v tem primeru enakomerno omejimo z

$$|f(t) - p_1(t)| \leq \frac{M}{8}(x_1 - x_0)^2 \quad \text{za vsak} \quad x \in [x_0, x_1]. \tag{5.25}$$

   Vzemimo, da želimo iz tabeliranih vrednosti izračunati vrednost funkcije sinus z natančnostjo pod 10^{-4}. Kakšen je lahko največ *korak tabele*, da bo zadoščala že linearna interpolacija?

   Ker je absolutna vrednost drugege odvoda $|\sin'' t| = |\sin t| \leq 1$, je ocena (5.25) v tem primeru

$$|\sin t - p_1(t)| \leq \frac{h^2}{8},$$

   kar je manj od 10^{-4}, kadar je

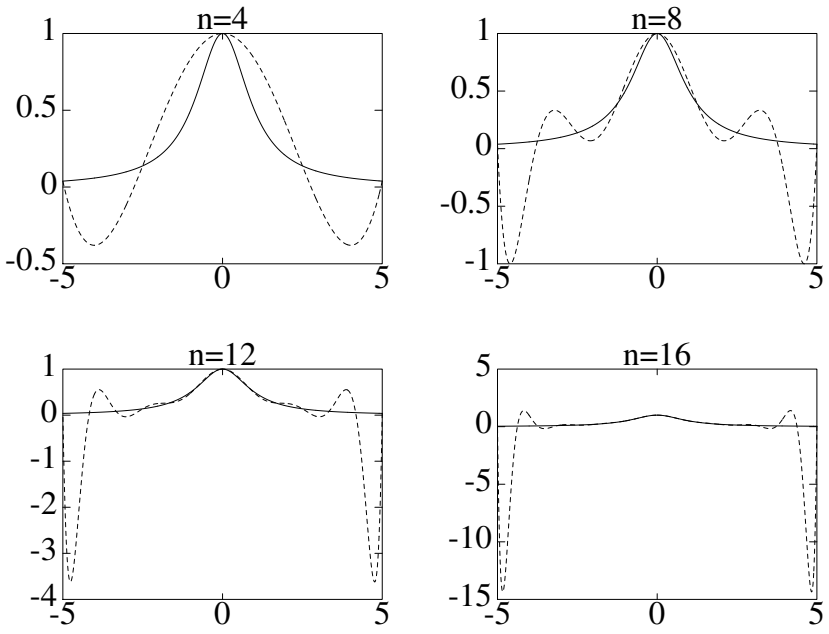
$$h \leq \sqrt{8} \cdot 10^{-2} \approx 0.0283.$$

   Da bi za navedeno natančnost zadoščala linearna interpolacija, mora biti korak tabele manjši kot 0.0283.

2. Kadar pa $t \notin [x_0, x_1]$, pravimo, da gre za *ekstrapolacijo.* Ker vrednost izraza $|(t - x_0)(t - x_1)|$ hitro in neomejeno narašča, ko se t oddaljuje od intervala $[x_0, x_1]$, je ekstrapolacija velikokrat zelo tvegana in potrebna je dodatna pazljivost pri interpretaciji rezultatov.

V prejšnjem primeru smo na primeru sinusne funkcije videli, da morajo biti vrednosti funkcije, ki jo interpoliramo, tabelirane dovolj na gosto, da za predpisano natančnost zadostuje linearna interpolacija na majnih intervalih. Druga možnost pa je, da uporabimo na celem intervalu, ki nas zanima, isti

interpolacijski polinom visoke stopnje. Na naslednjem primeru bomo videli, da je to navadno slabša rešitev.



Slika 5.4: Interpolacija v ekvidistantnih točkah

Primer 5.4. Funkcijo

$$f(t) = \frac{1}{1+x^2}$$

interpolirajmo na intervalu $[-5, 5]$ zaporedoma z interpolacijskimi polinomi stopnje 4,8,12 in 16 skozi točke, ki naj bodo enakomerno razporejene po intervalu. Iz slike 5.4 vidimo, kako napaka na robu intervala narašča z naraščanjem števila interpolacijskih točk.

5.6 Aproksimacija in metoda najmanjših kvadratov

Funkcija f naj bo podana s tabelo vrednosti v n med seboj različnih točkah $f_i = f(x_i)$, $i = 1, \ldots, n$. Iščemo tak polinom p_k stopnje ne večje od $k < n$,

za katerega ima izraz

$$E_{LSQ} = \sqrt{\sum_{i=1}^{n} (p_k(x_i) - f_i)^2}$$

najmanjšo vrednost. Ker je E_{LSQ} odvisen od koeficientov polinoma p_k

$$E_{LSQ}(a_0, \ldots, a_k) = \sqrt{\sum_{i=1}^{n} (a_0 + a_1 x_i + \cdots + a_k x_i^k - f_i)^2}, \tag{5.26}$$

imamo pred seboj problem iskanja prostega ekstrema funkcije več spremenljivk. Potreben pogoj za nastop ekstrema je, da so vsi parcialni odvodi $\partial E_{LSQ}^2 / \partial a_i$ enaki 0, od koder dobimo sistem linearnih enačb (imenujemo ga *normalni sistem* enačb)

$$\begin{array}{lcllcl}
a_0 n & + & a_1 \sum_{i=1}^n x_i & + \cdots + & a_k \sum_{i=1}^n x_i^k & = \sum_{i=1}^n f_i \\
a_0 \sum_{i=1}^n x_i & + & a_1 \sum_{i=1}^n x_i^2 & + \cdots + & a_k \sum_{i=1}^n x_i^{k+1} & = \sum_{i=1}^n f_i x_i \\
\vdots & & \vdots & & \vdots & \vdots \\
a_0 \sum_{i=1}^n x_i^k & + & a_1 \sum_{i=1}^n x_i^{k+1} & + \cdots + & a_k \sum_{i=1}^n x_i^{2k} & = \sum_{i=1}^n f_i x_i^k,
\end{array} \tag{5.27}$$

ki ga načeloma lahko rešimo z Gaussovo metodo (poglavje 2), vendar je pri velikem številu parametrov (polinom visoke stopnje) ta sistem lahko zelo slabo pogojen.

Zapišimo algoritem, s katerim lahko izračunamo aproksimacijski polinom z metodo najmanjših kvadratov:

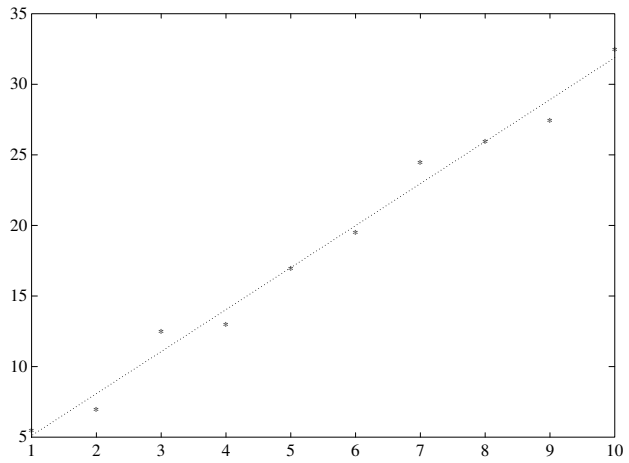
Algoritem 5.2. Metoda najmanjših kvadratov Naj bo funkcija f podana v obliki tabele

$$\begin{array}{rl} x & : x_1, \quad x_2, \quad \ldots, \quad x_n \\ f(x) & : f_1, \quad f_2, \quad \ldots, \quad f_n. \end{array}$$

Naslednji algoritem izračuna koeficiente polinoma p_k stopnje $< k$, ki aproksimira funkcijo f z metodo najmanjših kvadratov:

for $i = 1 : 2 * k + 1$
    $y(i) = \mathbf{sum}(x.\hat{}(i-1))$
end
for $i = 1 : k + 1$
    for $j = 1 : k + 1$
        $A(i,j) = y(i+j-1)$ % matrika normalnega sistema
    end
        $b(i) = \mathbf{sum}(f. * x.\hat{}(i-1))$ % desne strani
end
$p = A \backslash b$ % rešitve normalnega sistema

Uporabo tega algoritma poglejmo na primeru.



Slika 5.5: Linearna aproksimacija

Primer 5.5. Poišči linearno funkcijo, ki v smislu najmanjših kvadratov najbolje aproksimira funkcijo f, dano s tabelo

x_i	1.0	2.0	3.0	4.0	5.0	6.0	7.0	8.0	9.0	10.0
f_i	5.5	7.0	12.5	13.0	17.0	19.5	24.5	26.0	27.5	32.5

Najprej izračunamo koeficiente sistema linearnih enačb

$$\begin{aligned} a_{11} &= n &&= 10 \\ a_{12} = a_{21} &= \textstyle\sum_{i=1}^{10} x_i &&= 55 \\ a_{22} &= \textstyle\sum_{i=1}^{10} x_i^2 &&= 385, \end{aligned}$$

in desne strani

$$\begin{aligned} b_1 &= \textstyle\sum_{i=1}^{10} f_i &&= 185 \\ b_2 &= \textstyle\sum_{i=1}^{10} x_i f_i &&= 1263. \end{aligned}$$

Normalni sistem enačb

$$\begin{aligned} 10.0a + \quad 55.0b \quad &= \quad 185.0 \\ 55.0a + \quad 385.0b \quad &= 1263.0 \end{aligned}$$

ima rešitev (na tri mesta)

$$a = 2.13 \quad \text{in} \quad b = 2.98,$$

tako, da je linearna aproksimacija po metodi najmanjših kvadratov enaka

$$p(x) = 2.98x + 2.13.$$

Napaka linearne aproksimacije je v tem primeru $E_{LSQ} \approx 3.07$. (glej sliko 5.5.)

Podobno kot med polinomi, ki so linearne kombinacije potenc x^i; $i = 0, \ldots, k$, lahko poiščemo aproksimacijsko funkcijo med linearnimi kombinacijami poljubnih linearno nedvisnih funkcij $g_i(x)$; $i = 0, \ldots, k$. V tem primeru

je matrika normalnega sistema enačb enaka

$$\begin{bmatrix} \sum_{i=1}^n g_0^2(x_i) & \sum_{i=1}^n g_0(x_i)g_1(x_i) & \cdots & \sum_{i=1}^n g_0(x_i)g_k(x_i) \\ \sum_{i=1}^n g_1(x_i)g_0(x_i) & \sum_{i=1}^k g_1^2(x_i) & \cdots & \sum_{i=1}^n g_1(x_i)g_k(x_i) \\ \vdots & \vdots & \vdots & \\ \sum_{i=1}^n g_k(x_i)g_0(x_i) & \sum_{i=1}^n g_k(x_i)g_1(x_i) & \cdots & \sum_{i=1}^n g_k^2(x_i) \end{bmatrix}, \quad (5.28)$$

desne strani pa

$$\left[\sum_{i=1}^n f_i g_0(x_i), \sum_{i=1}^n f_i g_1(x_i), \ldots \sum_{i=1}^n f_i g_k(x_i)\right]^T. \quad (5.29)$$

Sistem linearnih enačb je posebno enostaven, kadar je njegova matrika diagonalna:

Definicija 5.1. Zaporedje funkcij $(g_j(x))$; $j = 0, \ldots$ je *ortogonalno na sistemu točk* x_i; $i = 1, \ldots, n$, če je

$$\sum_{i=1}^n g_j(x_i)g_k(x_i) \begin{cases} = 0 & \text{kadar} \quad j \neq k \\ \neq 0 & \text{kadar} \quad j = k, \end{cases} \quad (5.30)$$

V primeru, ko iščemo aproksimacijsko funkcijo $g(x)$ kot linearno kombinacijo funkcij $g_j(x)$; $j = 0, \ldots, k$, ki so ortogonalne na sistemu točk x_i; $i = 1, \ldots, n$, je

$$g(x) = \sum_{j=0}^k c_j g_j(x), \quad (5.31)$$

kjer so koeficienti c_j določeni z

$$c_j = \frac{\sum_{i=1}^n f_i g_j(x_i)}{\sum_{i=1}^n g_j^2(x_i)}. \quad (5.32)$$

5.7 Ortogonalni polinomi

V tem razdelku si bomo ogledali nekatere osnovne lastnosti in nekatere pomembnej'se primere zaporedij ortogonalnih polinomov. V tem poglavju se naša glavna motivacija izhaja iz polinomske aproksimacije, pa so sistemi

ortogonalnih polinomov uporabni tudi v drugih kontekstih (v naslednjem poglavju bomo spoznali uporabnost ortogonalnih polinomov pri računanju določenih integralov). Najprej poglejmo na ortogonalnost funkcij nekoliko splošneje, kot smo bili vajeni doslej.

V nadaljevanju naj bo $[a, b]$ nek dan interval in $w(x)$, ki je definirana in pozitivna na tem intervalu. Skalarni produkt $\langle f, g\rangle$ bomo definirali na dva alternativna načina: skalarni produkt na intervalu $[a, b]$

Definicija 5.2. Za poljubni funkciji f in g, definirani na intervalu $[a, b]$ in utežno funkcijo $w(x) > 0$ na $[a, b]$ je *skalarni produkt na intervalu* $[a, b]$

$$\langle f, g\rangle = \int_a^b f(x)g(x)w(x)\,dx. \tag{5.33}$$

Pri tem privzamemo, da integral obstaja (vsaj kot posplošeni integral) za vse funkcije f in g, ki nas zanimajo;

ali skalarni produkt na množici točk

Definicija 5.3. Za danih N točk $\xi_i \in [a, b]$, $i = 1, \ldots N$ je *skalarni produkt na množici točk*

$$\langle f, h\rangle = \sum_{i=1}^{N} f(\xi_i)g(\xi_i)w(\xi_i)\,dx. \tag{5.34}$$

Navadno bo že iz konteksta jasno, katero definicijo skalarnega produkta imamo v mislih. S pomočjo skalarnega produkta lahko definiramo, kdaj sta dve funkciji ortogonalni.

Definicija 5.4. *Funkciji f in g, definirani na intervalu $[a, b]$ sta* ortogonalni*, kadar je*

$$\langle f, g\rangle = 0.$$

Oglejmo si nekaj primerov.

Primer 5.6. Naj bosta $f(x) \equiv 1$ in $g(x) = x$. Z lahko se da preveriti, da sta f in g ortogonalni, če za skalarni produkt vzamemo

$$\langle f, g \rangle = \int_{-1}^{1} f(x)g(x)\,dx$$

ali za skalarni produkt

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

Prav tako sta ortogonalni za skalarni produkt

$$\langle f, g \rangle = \sum_{i=-5}^{5} f(i)g(i)$$

ali za

$$\langle f, g \rangle \sum_{i=-5}^{5} f(i)g(i)(1+i^2).$$

Nista pa ortogonalni za skalarna produkta

$$\langle f, g \rangle = \int_{-1}^{1} e^x f(x)g(x)$$

ali

$$\langle f, g \rangle \sum_{i=-5}^{5} \frac{f(i)g(i)}{(6-i)}.$$

Zelo uporabna so zaporedja ortogonalnih polinomov z naraščajočimi stopnjami.

Definicija 5.5. Zaporedje $P_0(x`), P_1(x), \ldots$ je (končno ali neskončno) *zaporedje ortogonalnih polinomov*, kadar velja

1. vsak $P_i(x)$ polinom stopnje točno i

$$P_i(x) = \alpha_i x^i + \text{ polinom stopnje } i-1, \quad \text{kjer je} \quad \alpha_i \neq 0;$$

2. polinomi P_i so paroma ortogonalni

$$\langle P_i, P_j \rangle = 0 \quad \text{kadarkoli je } i \neq j\,.$$

Poglejmo primer:

Primer 5.7. Za skalarni produkt

$$\langle f, g \rangle = \int_{-1}^{1} f(x)g(x)\,dx$$

funkcije

$$P_0(x) \equiv 1, \quad P_1(x) = x, \quad P_2(x) = 3x^2 - 1$$

sestavljajo zaporedje (treh) ortogonalnih polinomov. Že v primeru 5.6 smo ugotovili, da sta P_0 in P_1 ortogonalna, velja pa tudi

$$\int_{-1}^{1} P_0(x)P_2(x)\,dx = \int_{-1}^{1} (3x^2 - 1)\,dx = \left[x^3 - x\right]_{-1}^{1} = 0$$

in

$$\int_{-1}^{1} P_1(x)P_2(x)\,dx = \int_{-1}^{1} x(3x^2 - 1)\,dx = \left[3x^4/4 - x^2/2\right]_{-1}^{1} = 0.$$

Zaporedje ortogonalnih polinomov s skalarnim produktom še ni enolično določeno, saj lahko zsak polinomše pomnožimo s poljubno konstanto, pa bodo še vedno ortogonalni. Da bi bilo zaporedje ortogonalnih polinomov določeno enolično se moramo dogovoriti za *normalizacijo*: skatero konstanto pomnožiti posamezni polinom. V stadardi uporabi je več načinov normalizacije.

1. **Monični polinomi** so polinomi, ki imajo vodilni koeficient enak 1.

2. **Normirani polinomi** so polinomi, katerih norma $||P|| = \sqrt{\langle P, P\rangle}$ je enaka 1.

3. Predpisana je vrednost polinomov P_i v neki točki na intervalu $[a, b]$, navadno $P_i(b)$.

Če je $P_0, P_1, \ldots, P_k$ zaporedje ortogonalnih polinomov,potem veljajo naslednje lastnosti.

Izrek 5.4. Poljuben polinom $p(x) = a_k x^k + \cdots + a_0$, stopnje $\leq k$ lahko na en sam način razvijemo po ortogonalnih polinomih

$$p(x) = b_k P_k(x) + b_{k-1} P_{k-1}(x) + \cdots + b_0 P_0(x), \tag{5.35}$$

kjer so koeficienti (Fourierjevi koeficienti) b_i enaki

$$b_i = \frac{\langle P_i, p\rangle}{\langle P_i, P_i\rangle.} \tag{5.36}$$

Dokaz: Enačbo (5.36) skalarno pomnožimo z P_i. Zaradi ortogonalnosti je na desni strani le člen pri b_i različen od nič, od koder sledi (5.36).

Izrek 5.5. Polinom $P_k(x)$ je ortogonalen na vse polinome $p(x)$ s stopnjo $< k$.

Dokaz: Zaradi (5.35) in ker je $b_k = 0$ je

$$\langle P_k, p\rangle = \sum_{i=0}^{k} \langle b_i P_k(x) P_i(x)\rangle = 0.$$

Ta na videz nedolžen izrek pa ima nekaj zelo pomembnih posledic.

Izrek 5.6. Polinom $P_k(x)$ ima k enostavnih realnih ničel, ki vse ležijo znotraj intervala $[a, b]$, kar pomeni, da lahko $P_k(x)$ zapišemo kot

$$P_k(x) = \alpha_k(x - x_{i,k})(x - x_{2,k}) \cdots (x - x_{k,k})$$

za k med seboj različnih števil z intervala (a, b).

Dokaz: Naj bodo $x_{1,k}, \ldots x_{i,k}$ tiste točke, kjer $P_k(x)$ na intervalu $[a, b]$ spremeni predznak. Vemo, da je $i \leq k$, ker ima $P_k(x)$ največ k realnih ničel. Pokazali bomo, da mora biti $i = k$.
Produkt $(x - x_{1,k}) \cdots (x - x_{i,k})P_k(x)$ na intervalu $[a, b]$ nikoli ne spremeni predznaka. Ker je $P_k(x)$ ortogonalen na vse polinome nižjih stopenj, je

$$\int_a^b w(x)(x - x_{1,k}) \cdots (x - x_{i,k})P_k(x)\, dx = 0,$$

razen, ko je $i = k$. Ker pa integrand ni identično enak 0 in na $[a, b]$ ne spremeni predznaka, je lahko le $i = k$. Ker $P_k(x)$ spremeni predznak k krat na intervalu $[a, b]$, so to vse njegove ničle in vse enostavne.
Dokaz za skalarni produkt nad množico točk (5.34) je podoben, privzeti moramo le, da je $N \geq k$.

Primer 5.8. Polinoma P_1 in P_2 iz primera 5.7 lahko zapišemo kot

$$P_1(x) = x = 1 \cdot (x - 0),$$

saj je edina njegova ničla $x_{1,1} = 0$ in

$$P_2(x) = 3x^2 - 1 = 3 \cdot \left(x + \frac{\sqrt{3}}{3}\right)\left(x - \frac{\sqrt{3}}{3}\right),$$

saj sta njegovi ničli $x_{1,2} = \frac{\sqrt{3}}{3}$ in $x_{2,2} = \frac{\sqrt{3}}{3}$.

S pomočjo naslednjega rezultata lahko izračunamo vrednosti moničnih ortogonalnih polinomov za izbran skalarni produkt.

Izrek 5.7. Vsako zaporedje moničnih ortogonalnih polinomov zadošča *tričlenski rekurzivni relaciji*

$$P_{k+1}(x) = (x - A_k)P_k(x) - B_k P_{k-1}(x), \tag{5.37}$$

kjer sta koeficienta

$$A_k = \frac{\langle xP_k, P_k\rangle}{\langle P_k, P_k\rangle}$$

in

$$B_k = \frac{\langle P_k, xP_{k-1}\rangle}{\langle P_{k-1}, P_{k-1}\rangle}$$

Dokaz: Ker sta P_{k+1} in P_k monična polinoma stopnje $k+1$ oziroma k,je $P_{k+1}(x) - xP_k(x)$ polinom stopnje k, zato ga lahko (izrek 5.4) razvijemo po ortogonalnih polinomih

$$P_{k+1}(x) - xP_k(x) = -A_k P_k(x) - B_k P_{k-1}(x) - C_k P_{k-2}(x) + \cdots .$$

Ko to relacijo skalarno pomnožimo s P_k, dobimo

$$\langle xP_k, P_k\rangle = A_k\langle P_k, P_k\rangle,$$

saj so zaradi ortogonalnosti polinomov P_i vsi ostali členi enaki 0, od koder lahko izračunamo A_k. Ko pa zgornji razvoj skalarno pomnožimo s P_{k-1}, dobimo

$$\langle xP_k, P_{k-1}\rangle = B_k\langle P_{k-1}, P_{k-1}\rangle,$$

saj so zopet, zaradi ortogonalnosti polinomov P_i vsi ostali členi enaki 0. Upoštevmo, da je $\langle xP_k, P_{k-1}\rangle = \langle P_k, xP_{k-1}\rangle$, in izračunamo še B_k. Da so vsi koeficienti razvoja od C_k dalje enaki 0, pa ugotovimo, ko zgornji razvoj pomnožimo skalarno po vrsti še z P_{k-2} do P_0.

Za ortogonalno zaporedje polinomov $\hat{P}_i$, ki niso monični, velja rekurzivna zveza

$$\hat{P}_{k+1} = (\hat{A}_k x - \hat{B}_k)\hat{P}_k - \hat{C}_k\hat{P}_{k-1}.$$

Če je $\hat{P}_k = \alpha_k P_k$, kjer je P_k monični polinom, potem je

$$\hat{A}_k = \frac{\alpha_{k+1}}{\alpha_k}, \quad \hat{B}_k = \frac{\alpha_{k+1}}{\alpha_k}A_k \quad \text{in} \quad \hat{C}_k = \frac{\alpha_{k+1}}{\alpha_k}B_k.$$

S pomočjo tričlenske rekurzivne relacije lahko stabilno računamo vrednosti ortogonalnih polinomov, če le poznamo vrednosti konstant A_i in B_i (ali $\hat{A}_i$, $\hat{B}_i$ in $\hat{C}_i$) in prvih dveh polinomov P_0 in P_1.

Primer 5.9. Za skalarni produkt in polinome iz primera 5.7 lahko izračunamo

$$A_1 = \frac{\langle xP_1, P_1\rangle}{\langle P_1, P_1\rangle} = \frac{\int_{-1}^{1} x^3\,dx}{\langle P_1, P_1\rangle} = 0$$

in

$$B_1 = \frac{\langle P_1, xP_0\rangle}{\langle P_0, P_0\rangle} = \frac{\int_{-1}^{1} x^2\,dx}{\int_{-1}^{1} 1\,dx} = \frac{2/3}{2} = \frac{1}{3},$$

zato je

$$P_2(x) = (x - A_1)P_1(x) - B_1P_0(x) = x_2 - \frac{1}{3}.$$

Za naslednji polinom iz zaporedja potrebujemo

$$A_2 = \frac{\langle xP_2, P_2\rangle}{\langle P_2, P_2\rangle} = \frac{\int_{-1}^{1} x(x^2 = 1/3)^2}{\langle P_1, P_1\rangle} = 0$$

in

$$B_2 = \frac{\langle P_2, xP_1\rangle}{\langle P_1, P_1\rangle} = \frac{\int_{-1}^{1} x^2(x^2 - 1/3)\,dx}{\int_{-1}^{1} x^2\,dx} = \frac{2/5 - 2/9}{2/3} = \frac{4}{15},$$

tako da je

$$P_3(x) = (x - A_2)P_2(x) = \frac{3}{5}P_1(x) = x(x^2 - 1/3) - 4/15 = x^3 - \frac{3x}{5}.$$

5.7.1 Klasični ortogonalni polinomi

Tri družine utežnih funkcij nas pripeljejo do polinomov, ki so v matematiki znani kot "klasični":

1. Naj bo interval $[-1, 1]$ in $w(x) = (1 - x)^\alpha(1 + x)^\beta$, kjer sta parametra $\alpha, \beta > -1$. Ustrezni ortogonalni polinomi so poznani kot *Jacobijevi polinomi* $P_n^{((\alpha,\beta)}$, od katerih bomo posebej omenili Legendre-ove polinome, ki ustrezajo izbiri $\alpha = \beta = 0$, in polinome Čebiševa, ki jih dobimo, če izberemo $\alpha = \beta = -\frac{1}{2}$.

2. Polinomi, ki so na intervalu $[0, \infty)$ ortogonalni z utežjo $w(x) = x^\alpha e^{-x}$ z $\alpha > -1$ so (posplošeni) Laguerrovi polinomi $L_n^{(\alpha)}$.

3. Polinomi, ki so na intervalu $(-\infty, \infty)$ ortogonalni z utežjo $w(x) = e^{-x^2}$, so Hermitovi polinomi $H_n(x)$.

Legendreovi[2] polinomi S skalarnim produktom

$$\langle f, g \rangle = \int_a^b f(x)g(x)\,dx.$$

na intervalu $[-1, 1]$ in normalizacijo $P(1) = 1$ dobimo zaporedje $P_k(x)$ Legendreovih polinomov

$$\begin{aligned}
P_0(x) &= 1, \\
P_1(x) &= x, \\
P_2(x) &= \frac{1}{2}(3x^2 - 1), \\
P_3(x) &= \frac{1}{2}(5x^3 - 3x), \\
P_4(x) &= \frac{1}{8}(35x^4 - 30x^2 + 3), \\
P_5(x) &= \frac{1}{8}(63x^5 - 70x^3 + 15x), \quad \dots .
\end{aligned}$$

Izračunamo jih rekurzivno s pomočjo formule

$$P_0(x) = 1,\ P_1(x) = x,\ P_{n+1} = \frac{1}{n+1}\left((2n+1)xP_n(x) - nP_{n-1}(x)\right).$$

Legendreovi polinomi lihe stopnje so lihi, sode stopnje so sodi

$$P_n(-x) = (-1)^n P_n(x),$$

njihova norma je enaka

$$||P_n|| = \sqrt{\int_{-1}^{1} P_n^2(x)\,dx} = \sqrt{\frac{2}{2n+1}},$$

[2]Adrien-Marie Legendre (1752, Pariz —1833, Pariz), francoski matematik, avtor metode najmanjših kvadratov

njihov odvod pri $x = 1$ je enak

$$P_n'(1) = \frac{n(n+1)}{2}.$$

Vrednosti odvodov lahko računamo rekurzivno iz

$$(2n+1)P_n(x) = P_{n+1}'(x) - P_{n-1}'(x).$$

Zanje velja *Rodriguesova formula*

$$P_n(x) = \frac{1}{2^n n!}\frac{d^n}{dx^n}\left((x^2-1)^n\right) .$$

Legendreove polinome lahko dobimo tudi kot rešitev Legendreove diferencialne enačbe

$$\left((1-x^2)P_n'(x)\right)' + n(n+1)P_n(x) = 0\,.$$

Polinomi Čebiševa[3] Z utežno funkcijo $w(x) = \frac{1}{\sqrt{1-x^2}}$ na intervalu $[-1,1]$ dobimo zaporedje $T_k(x)$ polinomov Čebyševa (prve vrste)

$$\begin{aligned}
T_0(x) &= 1,\\
T_1(x) &= x,\\
T_2(x) &= 2x^2-1,\\
T_3(x) &= 4x^3-3x,\\
T_4(x) &= 8x^4-8x^2+1,\\
T_5(x) &= 16x^5-20x^3+5x, \quad \ldots\ .
\end{aligned}$$

Računamo jih rekurzivno s pomočjo formule

$$T_0(x) = 1,\ T_1(x) = x,\ T_{n+1} = 2xT_n(x) - T_{n-1}(x).$$

Polinomi Čebyševa lihe stopnje so lihi, sode stopnje so sodi

$$T_n(-x) = (-1)^n T_n(x)$$

zanimiva je njihova zveza s trigonometričnimi funkcijami

$$T_n(x) = \cos(n \arccos x),$$

[3]Pafnutij Lvovič Čebyšev (1821, Okatovo pri Kalugi, Rusija — 1894, St Petersburg), ruski matematik. Ukvarjal se je s teorijo verjetnosti, statistiko, mehaniko in teorijo števil.

od koder lahko ugotovimo, da so polinomi Čebyševa normirani tako, da je $T_n(1) = 1$. Njihova norma je enaka

$$||T_n|| = \sqrt{\int_{-1}^{1} \frac{T_n^2(x)}{\sqrt{1-x^2}}\, dx} = \sqrt{\pi/2} \text{ za } n \geq 1 \text{ in } ||T_0|| = \sqrt{\pi}.$$

Rodriguesova formula za polinome Čebyševa je

$$T_n(x) = \frac{\sqrt{1-x^2}}{(-1)^n(2n-1)(2n-3)\cdots(3)(1)} \frac{d^n}{dx^n}(1-x^2)^{n-1/2}\,.$$

Polinomi Čebyševa so rešitve diferencialne enačbe Čebiševa

$$(1-x^2)T_n''(x) - xT_n'(x) + n^2 P_n(x) = 0\,.$$

Ničle polinoma $T_n(x)$ so $x_k = \cos\frac{\pi(2k-1)}{2n}$.

Laguerrovi[4] **polinomi** Z utežno funkcijo $w(x) = e^{-x}$ na intervalu $[0, \infty]$ dobimo zaporedje $L_n(x)$ Laguerrovihv polinomov

$$\begin{aligned}
L_0(x) &= 1,\\
L_1(x) &= -x + 1,\\
L_2(x) &= \frac{1}{2}(x^2 - 4x + 2),\\
L_3(x) &= \frac{1}{6}(-x^3 + 9x^2 - 18x + 6),\\
L_4(x) &= \frac{1}{24}(x^4 - 16x^3 + 72x^2 - 96x + 24), \quad \ldots\ .
\end{aligned}$$

Njihove vrednosti računamo rekurzivno s pomočjo formule

$$L_0(x) = 1,\ L_1(x) = -x + 1,$$

$$L_{n+1} = \frac{1}{n+1}((2n+1-x)L_n(x) - nL_{n-1}(x))$$

[4]Edmond Nicolas Laguerre (1834, Bar-le-Duc — 1886, Bar-le-Duc, Francija), francoski matematik, ukvarjal se je predvsem z matematično analizo, teorijo aproksimacij in geometrijo.

Lagerrovi polinomi so normalizirani tako, da je $L_n(0) = 1$. Njihova norma je

$$||L_n|| = \sqrt{\int_0^\infty e^{-x} L_n^2(x)\, dx} = 1.$$

Rodriguesova formula za Laguerrove polinome je

$$L_n(x) = \frac{e^x}{n!} \frac{d^n}{dx^n}(x^n e^{-x})\,.$$

Laguerrovi polinomi so rešitev Laguerrove diferencialne enačbe

$$xL_n''(x) + (1-x)L_n'(x) + nL_n(x) = 0\,.$$

Hermitovi[5] polinomi Hermitovi polinomi so $H_n(x)$ ortogonalni na intervalu $(-\infty, \infty)$ z utežjo $w(x) = e^{-x^2}$. Prvih pet Hermitovih polinomov je

$$\begin{aligned} H_0(x) &= 1, \\ H_1(x) &= 2x, \\ H_2(x) &= 4x^2 - 2, \\ H_3(x) &= 8x^3 - 12x, \\ H_4(x) &= 16x^4 - 48x^2 + 12, \ \ldots\ . \end{aligned}$$

Vrednosti Hermitovih polinomov računamo s pomočjo rekurzivne formule

$$H_0(x) = 1,\ h_1(x) = 2x,\ H_{n+1}(x) = 2xH_n(x) - 2nH_{n-1}(x)\,.$$

Hermitovi polinomi sode stopnje so sode funkcije, lihe stopnje pa lihe funkcije

$$H_n(x) = (-1)^n H_n(-x)\,.$$

Njihova norma je enaka

$$||H_n|| = \sqrt{\int_{-\infty}^\infty e^{-x^2} H_n^2(x)\, dx} = \sqrt{\pi} 2^n!\,.$$

[5]Charles Hermite (1822 Dieuze, Francija — 1901 Pariz), francoski matematik. Ukvarjal se je predvsem z eliptičnimi funkcijami in teorijo števil. Prvi je dokazal, da je število e (osnova naravnih logaritmov) iracionalno.

Rodriguesova formula za Hermitove polinome je

$$H_n(x) = (-1)^n e^{x^2} \frac{d^n}{dx^n} e^{-x^2} .$$

Hermitovi polinomi so rešitve Hermitove diferencialne enačbe

$$y'' - 2xy' + 2ny = 0 .$$

5.7.2 Ortogonalni polinomi nad sistemom točk

Poglejmo si še polinome, ki so ortogonalni nad sistemom točk. Na intervalu $[a, b]$ si izberimo števila $x_i,\ i = 1, \ldots, N$. Preprosta posledica izreka 5.7 je

Posledica 5.8. Zaporedje moničnih ortogonalnih polinomov $t_i(x)$, ortogonalnih nad sistemom točk $x_i,\ i = 1, \ldots, N$ zadošča *tričlenski rekurzivni relaciji*

$$t_{k+1}(x) = (x - A_k)t_k(x) - B_k t_{k-1}(x), \tag{5.38}$$

kjer so koeficienti A_k in B_k določeni z

$$A_k = \frac{\sum_{j=1}^{n} x_j t_k^2(x_j)}{\sum_{j=1}^{n} t_k^2(x_j)} \tag{5.39}$$

$$B_0 = 0$$

$$B_k = \frac{\sum_{j=1}^{n} x_j t_k(x_j) t_{k-1}(x_j)}{\sum_{j=1}^{n} t_{k-1}^2(x_j)}, \tag{5.40}$$

prvi člen zaporedja pa je $t_0(x) = 1$.

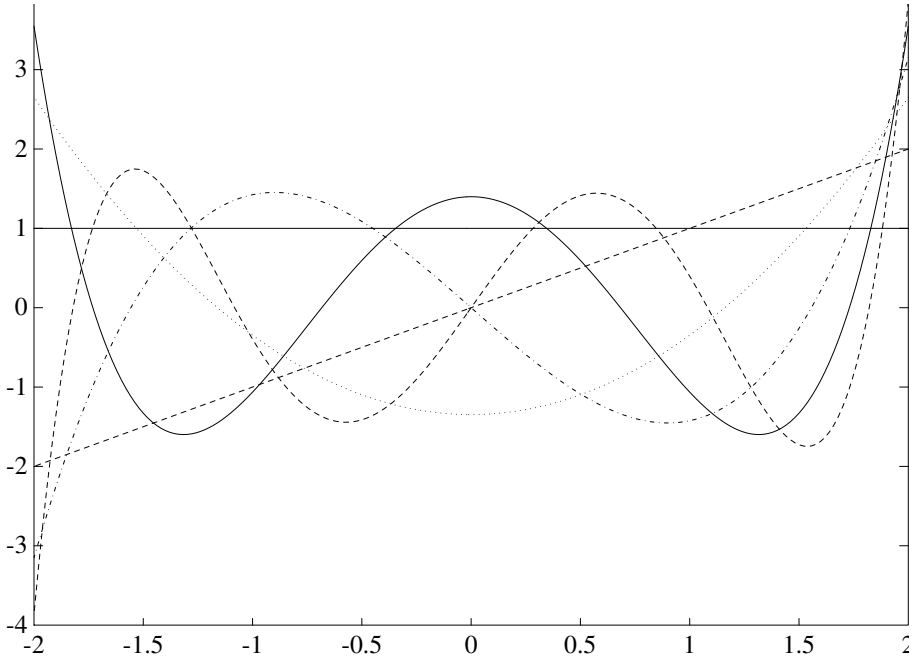
Naslednji algoritem izračuna vrednosti koeficientov rekurzivne formule in vrednosti ortogonalnih polinomov v izbrabih točkah:

i	α_i	β_i
1		1.3467
2	$0.13398 \cdot 10^{-15}$	1.0772
3	$-0.21107 \cdot 10^{-15}$	1.0387
4	$0.30531 \cdot 10^{-15}$	1.0257
5	$-0.08703 \cdot 10^{-15}$	1.0196
6	$-0.06727 \cdot 10^{-15}$	1.0162
7	$-0.09497 \cdot 10^{-15}$	1.0140
8	$-0.16334 \cdot 10^{-15}$	1.0124
9	$-0.01955 \cdot 10^{-15}$	

Tabela 5.3: Koeficienti α in β rekurzivne formule za izračun zaporedja ortogonalnih polinomov na množici točk $x_i = -2 + 0.2i$; $i = 0, \ldots, 20$

Algoritem 5.3. Ortogonalni polinomi nad sistemom točk Vektor $(x_i,\ i = 1, \ldots, n)$ naj vsebuje med seboj različne točke. Naslednji algoritem izračuna koeficiente tričlenske rekurzivne relacije (5.38), ki jih potrebujemo, da izračunamo polinome ortogonalnega zaporedja do stopnje k, in vrednosti ortogonalnih polinomov v izbranih točkah:

$[m, n] = \mathbf{size}(x)$
$t = \mathbf{zeros}(k, n)$
$a = \mathbf{zeros}(k, 1)$
$b = \mathbf{zeros}(k, 1)$
$t(1, :) = \mathbf{ones}(1, n)$
$t(2, :) = x - \mathbf{sum}(x)/n$
for $i = 2 : k$
    $a(i + 1) = (x * (t(i, :). * t(i, :))')/(t(i, :) * t(i, :)')'$
    $b(i) = t(i, :) * t(i, :)'/(t(i - 1, :) * t(i - 1, :)')$
    $t(i + 1, :) = (x(:) - a(i + 1)). * t(i, :) - b(i) * t(i - 1, :)$
end



Slika 5.6: Zaporedje ortogonalnih polinomov

Primer 5.10. Izračunajmo prvih 8 polinomov, ortogonalnih na množici točk $x_i = a + ih$; $i = 0, \ldots, 20$, kjer je $a = -2$ in $h = 0.2$. Te točke so enakomerno razporejene na intervalu $[-2, 2]$ z razmakom h.
Izračunani koeficienti α_i in β_i rekurzivne formule (5.38) so v tabeli 5.3, grafi prvih petih polinomov pa so na sliki 5.6.

Računanje aproksimacijskih polinomov, izraženih s pomočjo ortogonalnih polinomov ima prednosti pred izražavo s potencami:

- Normalni sistem, zapisan z ortogonalnimi polinomi ima diagonalno obliko in je torej enostavno rešljiv, medtem ko je normalni sistem, izražen s potencami, pogosto slabo pogojen.
- Kadar z natančnostjo izračunanega aproksimacijskega polinoma stopnje k nismo zadovoljni, mu lahko enostavno dodamo naslednji člen $d_{k+1}t_{k+1}(x)$.

V (5.26) smo aproksimacijski polinom zapisali kot linearno kombinacijo potenc x^j in smo dobili normalni sistem enačb v obliki (5.27). Če

pa aproksimacijski polinom zapišemo kot linearno kombinacijo polinomov $t_j(x);\ j = 0, \ldots, k$ stopnje j, ki so ortogonalni na sistemu med seboj različnih točk $x_i;\ i = 1, \ldots, n$, dobimo aproksimacijsko funkcijo

$$p(x) = \sum_{j=0}^{k} c_j t_j(x); \quad c_j = \frac{\sum_{i=1}^{n} f_i t_j(x_i)}{\sum_{i=1}^{n} t_j^2(x_i)}. \tag{5.41}$$

5.8 Aproksimacija periodičnih funkcij

Pri aproksimaciji funkcij, ki opisujejo periodične pojave, je najbolje uporabljati *trigonometrične polinome*

$$p(x) = a_0 + \sum_{i=1}^{m} (a_i \cos ix + b_i \sin ix). \tag{5.42}$$

Vzemimo, da je f periodična funkcija s periodo 2π. Kadar ima funkcija $g(x)$, ki jo aproksimiramo, periodo $\tau \neq 2\pi$, lahko s spremembo neodvisne spremenljivke dosežemo, da ima funkcija $f(x) = g(\frac{\tau x}{2\pi})$ periodo 2π.

Obravnavali bomo le primer, ko imamo $2n$ aproksimacijskih točk

$$x_j = \frac{j\pi}{n}, \quad j = -n, -n+1, \ldots, n-1 \tag{5.43}$$

razporejenih ekvidistantno na intervalu $[-\pi, \pi]$.

Pri aproksimaciji s trigonometričnim polinomom izkoristimo dejstvo, da tvorijo funkcije

$$1,\ \cos x,\ \sin x,\ \cos 2x,\ \sin 2x,\ \ldots,\ \cos mx,\ \sin mx \tag{5.44}$$

zaporedje ortogonalnih funkcij na množici točk (5.43), saj veljajo relacije

$$\begin{aligned}
\sum_{j=-n}^{n-1} \sin mx_j \cos kx_j &= 0, \\
\sum_{j=-n}^{n-1} \cos mx_j \cos kx_j &= \begin{cases} 0 & m \neq k \\ n & m = k \neq 0 \\ 2n & m = k = 0, \end{cases} \\
\sum_{j=-n}^{n-1} \sin mx_j \sin kx_j &= \begin{cases} n & m = k \neq 0 \\ 0 & \text{sicer} \end{cases}
\end{aligned} \tag{5.45}$$

za vse $m, k = 0, 1, \ldots, n$. O njihovi veljavnosti se lahko prepričamo, če zapišemo vsoto geometrijskega zaporedja

$$\sum_{j=-n}^{n-1} e^{ikx_j} = \sum_{j=-n}^{n-1} e^{ik\pi j/n} = e^{-ik\pi} \frac{e^{2ik\pi} - 1}{e^{ik\pi/n} - 1},$$

ki je enaka 0 za $k = 1, \ldots, 2n-1$ in enaka $2n$ za $k = 0$. Od tod dobimo

$$\sum_{j=-n}^{n-1} \cos kx_j = \sum_{j=-n}^{n-1} \sin kx_j = 0, \quad k = 1, \ldots, 2n-1,$$

relacije (5.45) pa lahko izpeljemo s pomočjo adicijskih izrekov za sinus in cosinus.

Koeficienti trigonometričnega polinoma (5.42), ki najbolje aproksimira dano funkcijo f po metodi najmanjših kvadratov so torej enaki

$$a_0 = \frac{1}{2n} \sum_{j=-n}^{n-1} f(x_j) \tag{5.46}$$

$$a_k = \frac{1}{n} \sum_{j=-n}^{n-1} f(x_j) \cos kx_j \tag{5.47}$$

$$b_k = \frac{1}{n} \sum_{j=-n}^{n-1} f(x_j) \sin kx_j \tag{5.48}$$

Formule (5.46–5.48) so podobne formulam za koeficiente Fourierove vrste. V poglavju o numerični integraciji bomo spoznali, da so (5.46–5.48) približki za Fourierove koeficiente, izračunani s trapeznim pravilom.

5.9 Povzetek

Interpolacija je postopek, pri katerem dano funkcijo nadomestimo z neko drugo, ki se s prvotno ujema v nekaj izbranih točkah. Nekoč so interpolacijo uporabljali predvsem za računanje vmesnih vrednosti tabeliranih funkcij (logaritemske tabele). Danes to nalogo opravljajo elektronski kalkulatorji in računalniki, pri tem pa si še vedno marsikdaj pomagajo z interpolacijo.

Drugo področje, kjer uporabljamo interpolacijo, in ki ga bomo srečali v naslednjih poglavjih, pa so formule za numerično odvajanje in integriranje.

V tem poglavju smo spoznali dva osnovna načina za računanje interpolacijskih polinomov, Lagrangeovo in Newtonovo interpolacijsko formulo. Lagrangeova formula je nekoliko preprostejša za uporabo, kadar želimo izračunati vrednosti interpolacijskega polinoma z vnaprej predpisano stopnjo. Kadar pa hočemo izračunati vrednost interpolacijskega polinoma v več točkah ali pa stopnje interpolacijskega polinoma ne poznamo vnaprej, je primernejša Newtonova oblika.

Poseben problem je zagotavljanje natančnosti interpolacijskega polinoma. Iz ocene (5.24) vidimo, da je napaka manjša, če so točke bliže skupaj ali pa če je stopnja interpolacije dovolj velika in pri tem višji odvodi ostajajo omejeni. Ker je v praksi le redko mogoče oceniti vrednost višjih odvodov, je varneje uporabljati interpolacijske polinome nizkih stopenj, zato pa morajo biti interpolacijske točke gostejše. Prav tako se moramo zavedati, da je napaka navadno najmanjša sredi intervala, na katerem ležijo interpolacijske točke in da narašča proti robu intervala. Kadar pa računamo vrednosti interpolacijske funkcije zunaj intervala, na katerem ležijo interpolacijske točke (ekstrapolacija), pa je napaka lahko neomejeno velika.

Večkrat (posebej, kadar imamo veliko število podatkov, ki so lahko nenatančni) je pomembno, da znamo dano funkcijo aproksimirati z neko drugo (enostavnejšo, laže izračunljivo) funkcijo. Podrobneje smo si ogledali aproksimacijo s polinomi po metodi najmanjših kvadratov. Ugotovili smo, da lahko aproksimacijski polinom, v primerjavi z običajno izražavo v obliki linearne kombinacije potenc, preprosteje in učinkoviteje izračunamo v obliki linearne kombinacije polinomov, ki so ortogonalni na množici aproksimacijskih točk ali na celem intervalu, Zato smo si ogledali osnovne lastnosti ortogonalnih polinomov ter spoznali nekaj klasičnih zaporedij ortogonalnih polinomov.

Spoznali smo tudi, kako lahko izračunamo aproksimacijo periodične funkcije v obliki trigonometrijskega polinoma. Ker so trigonometrične funkcije ortogonalne na ekvidistantni mreži točk, lahko njegove koeficiente preprosto izračunamo.

5.10 Problemi

1. Pokaži, da rešitev normalnega sistema enačb (5.28) z desnimi stranmi (5.29) določa koeficiente c_i funkcije

$$F(x) = \sum_{i=0}^{k} c_i g_i(x),$$

ki dano funkcijo f aproksimira po metodi najmanjših kvadratov,

Navodilo: Poišči ekstrem funkcije

$$E(c_0, \ldots, c_k) = \sqrt{\sum_{j=1}^{n} (F(x_j) - f(x_j))^2}.$$

2. Funkcije $g_i(x)$; $i = 0, \ldots, k$ naj bodo ortogonalne na sistemu točk x_j; $j = 1, \ldots, n$. Prepričaj se, da funkcija (5.31) s koeficienti (5.32) res aproksimira dano funkcijo f v smislu najmanjših kvadratov.

3. V tabeli

x	0	1	2	3	4
$f(x)$	6	4	20	108	370

je tabelirana vrednost nekega polinoma 4. stopnje.

   (a) Zapiši algoritem za izračun vrednosti polinoma v poljubni točki, ki je ni v tabeli.

   (b) Sestavi tabelo deljenih diferenc.

   (c) Izračunaj vrednost $f(5)$ in $f(-1)$.

   (d) Zapiši polinom v Newtonovi in v normalni obliki.

4. Funkcijo $e^{-x} \sin x$ želimo tabelirati na intervalu $[0, 2]$, da bomo tabelirane vrednosti uporabili pri računanju vmesnih vrednosti z interpolacijo.

   (a) Kolikšen naj bo korak tabele, da bo napaka kubične interpolacije pod 10^{-5}?

(b) Napiši algoritem, ki iz tabeliranih vrednosti izračuna kubični interpolacijski polinom.

(c) Kolikšna mora biti stopnja interpolacijskega polinoma, da bo napaka pod 10^{-6}, če je korak tabele 0.02?

5. Iz tabele za vrednosti funkcije sinus dobimo naslednje vrednosti:

x	$\sin x$
2.4	0.6755
2.6	0.5156
2.8	0.3350
3.0	0.1411
3.2	−0.0584
3.4	−0.2555

(a) Sestavi tabelo deljenih diferenc.

(b) Izračunaj približni vrednosti za $\sin 3.1$ in $\sin e$ s pomočjo kubičnega interpolacijskega polinoma.

(c) Oceni napako obeh aproksimacij s pomočjo formule (5.24) in jo primerjaj z dejansko napako.

(d) Kako bi s pomočjo kubičnega interpolacijskega polinoma ugotovil, za kateri x je $\sin x = 0.3$?

6. Funkcija $\sin x$ je tabelirana na intervalu $[0, 1.6]$.

(a) Kolikšna je maksimalna napaka kvadratne interpolacije, če je korak tabele enak $h = 0.01$?

(b) Kolikšen naj bo korak tabele, da bo napaka kvadratne interpolacije pod 10^{-4}?

(c) Napiši algoritem, ki iz tabeliranih vrednosti izračuna kubični interpolacijski polinom.

(d) Kolikšna mora biti stopnja interpolacijskega polinoma, da bo napaka pod 10^{-6}, če je korak tabele 0.02?

7. Iz tabele naravnih logaritmov dobimo naslednje vrednosti

x	$\log x$
1.0	0.0000
1.5	0.4055
2.0	0.6931
3.0	1.0986
3.5	1.2528
4.0	1.3863

(a) Sestavi tabelo deljenih diferenc.

(b) Izračunaj približni vrednosti za $\log 2.5$ in $\log 1.75$ s pomočjo kubične interpolacije.

(c) Oceni napako približkov iz prejšnje točke in primerjaj s točnimi vrednostmi.

8. Pri merjenju hitrosti izstrelka smo dobili naslednje rezultate:

$t[s]$	$h[m]$
1.0	197.2
1.2	393.3
1.4	519.7
1.6	568.0
1.8	528.2
2.0	391.2

(a) Napiši algoritem, ki dane podatke nadomesti s polinomom stopnje n po metodi najmanjših kvadratov.

(b) Po metodi najmanjših kvadratov izračunaj kvadratno aproksimacijo.

(c) Izračunaj največjo višino izstrelka? Kdaj jo doseže? Kdaj bo izstrelek padel na tla ($h = 0$)?

9. Funkcijo f, dano v obliki tabele

x	0.00	0.25	0.50	0.75	1.00
$f(x)$	0.5000	0.7143	1.0000	1.4000	2.0000

želimo aproksimirati s funkcijo oblike $p(x) = a + be^x$.

(a) Zapiši normalni sistem enačb za neznana koeficienta a in b.

(b) Zapiši algoritem, ki izračuna a in b.

(c) Izračunaj a in b.

Poglavje 6

Numerična integracija

6.1 Uvod

Pojma odvoda in določenega integrala smo že srečali pri matematiki. Vemo, da je odvajanje razmeroma enostavna operacija in da lahko vsaki funkciji, ki jo lahko zapišemo kot kombinacijo elementarnih funkcij, poiščemo odvod. Povsem drugače je z integralom. Določeni integral izračunamo analitično s pomočjo nedoločenega integrala po formuli

$$\int_a^b f(x)\,dx = F(b) - F(a), \tag{6.1}$$

kjer je $F(x)$ nedoločeni integral funkcije $f(x)$

$$F(x) = \int f(x)\,dx.$$

Nedoločenega integrala velikokrat ne moremo zapisati kot kombinacijo elementarnih funkcij, kot na primer integrale

$$\int e^{-x^2}\,dx, \quad \int \frac{\sin x}{x}\,dx \quad \text{in} \quad \int x \operatorname{tg} x\,dx.$$

Pri nekaterih določenih integralih, ki v uporabi zelo pogosto nastopajo, ta problem nekako 'pometemo pod preprogo' tako, da ga proglasimo za novo 'elementarno funkcijo' (to so tako imenovane *specialne funkcije*). Tako naredimo npr. s *funkcijo napake*

$$\operatorname{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2}\,dt,$$

ki je pomembna pri verjetnostnem računu in statistiki in katere vrednosti so tabelirane. Pri večini določenih integralov pa moramo ravnati drugače. Integral navadno nadomestimo s primerno končno integralsko vsoto in izkaže se, da je napaka, ki jo pri tem zagrešimo, običajno omejena in dovolj majhna.

Prav obratna situacija pa je pri računanju odvodov dane funkcije. Kadar je funkcija podana s formulo, jo je navadno lahko odvajati analitično. Teže pa je z numerično metodo natančno izračunati vrednost odvoda.

V tem poglavju bomo najprej spoznali nekaj osnovnih integracijskih formul in splošen postopek za njihovo konstrukcijo. Videli bomo, kako lahko izračunamo tudi posplošeni integral ali integral singularne funkcije. Naučili se bomo sproti prilagajati dolžino koraka integracije tako, da bomo integral izračunali z vnaprej predpisano natančnostjo in kako lahko s primerno kombinacijo enostavnih integracijskih formul dosežemo večjo natančnost (Rombergova metoda). Na koncu si bomo ogledali še nekaj metod za numerično računanje odvodov funkcije.

6.2 Trapezna formula

Da bi lahko izračunali približno vrednost določenega integrala

$$I = \int_a^{a+h} f(x)\,dx, \tag{6.2}$$

bomo funkcijo f nadomestili z linearnim interpolacijskim polinomom skozi točki a in $a+h$, torej

$$I \approx \int_a^{a+h} p(x)\,dx,$$

kjer interpolacijski polinom p izračunamo z Newtonovo interpolacijsko formulo (5.7)

$$p(x) = \frac{f(a+h)-f(a)}{h}(x-a) + f(a).$$

Če je funkcija f vsaj dvakrat odvedljiva na integracijskem intervalu $[a, a+h]$, je

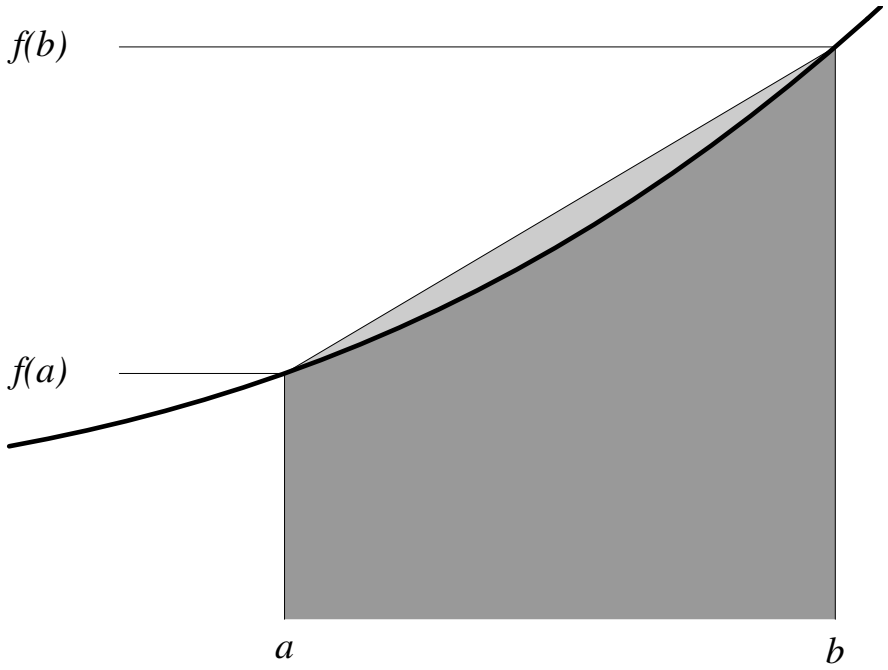
$$f(x) = p(x) + \frac{f''(\xi)}{2}(x-a)(x-a-h), \tag{6.3}$$

kjer je točka ξ med a in $a+h$. Če to vstavimo v integral (6.2) in integriramo, dobimo *trapezno formulo* za izračun vrednosti določenega integrala

(problem 1)

$$\int_a^{a+h} f(x)\,dx = \frac{h}{2}(f(a)+f(a+h)) - \frac{h^3}{12}f''(\xi_1), \tag{6.4}$$

kjer je ξ_1 zopet neka točka med a in $a+h$, navadno različna od ξ. Geometrijsko si trapezno formulo predstavljamo tako, da krivočrtni trapez med grafom funkcije f in abscisno osjo v mejah od a do $a+h$ nadomestimo s pravim trapezom (slika 6.1).



Slika 6.1: Trapezna formula

Napaka trapezne formule (6.4),ki jo predstavlja člen $\frac{h^3}{12}f''(\xi_1)$, je majhna, če je $f''(x)$ majhen na $[a, a+h]$ ali če je dolžina integracijskega intervala h majhna. Tako je npr. za $|f''(x)| \leq 1$ in $h = 1/10$ napaka trapezne formule manjša od 10^{-4}. Seveda pa pri daljših intervalih od trapezne formule (6.4) ne moremo pričakovati majhne napake. V tem primeru si pomagamo tako, da celoten integracijski interval razdelimo na dovolj majhne podintervale in uporabimo trapezno formulo na vsakem od podintervalov, potem pa dobljene

delne integrale seštejemo. Celoten interval $[a, b]$ razdelimo na n (zaradi enostavnosti enakih) podintervalov dolžine $h = (b - a)/n$ in označimo delilne točke podintervalov z

$$x_i = a + ih; \quad i = 0, \ldots, n,$$

tako da imamo

$$a = x_0 < x_1 < \cdots < x_{n-1} < x_n = b.$$

Na vsakem od podintervalov uporabimo trapezno formulo

$$\int_{x_{i-1}}^{x_i} f(x)\,dx \approx \frac{h}{2}(f(x_{i-1}) + f(x_{i-1} + h)),$$

in delne rezultate seštejemo

$$\int_a^b f(x)\,dx \approx \frac{h}{2}\sum_{i=1}^{n}[f(x_{i-1}) + f(x_i)].$$

Dobljeni formuli

$$\int_a^b f(x)\,dx \approx \frac{h}{2}[f(x_0) + 2f(x_1) + \cdots + 2f(x_{n-1}) + f(x_n)], \tag{6.5}$$

pravimo *trapezno pravilo.*

Oglejmo si še napako trapeznega pravila. Privzemimo, da je integracijska funkcija f dvakrat zvezno odvedljiva na intervalu $[a, b]$. Če s $T(h)$ označimo približek k vrednosti integrala (6.1), izračunan s trapeznim pravilom (6.5), iz (6.4) dobimo

$$\int_a^b f(x)\,dx - T(h) = -\frac{h^3}{12}\sum_{i=1}^{n} f''(\xi_i) = -\frac{h^2}{12}\frac{b-a}{n}\sum_{i=1}^{n} f''(\xi_i),$$

kjer je $\xi_i \in [x_{i-1}, x_i]$. Ker je faktor $\frac{1}{n}\sum_{i=1}^{n} f''(\xi_i)$ ravno povprečna vrednost števil $f''(\xi_i)$, leži med najmanjšim in največjim od teh števil. Iz izreka o povprečni vrednosti zvezne funkcije vemo, da obstaja tako število $\xi \in [a, b]$, da je $f''(\xi) = \frac{1}{n}\sum_{i=1}^{n} f''(\xi_i)$. Tako lahko zapišemo končni rezultat:

Izrek 6.1. *Funkcija f naj bo dvakrat zvezno odvedljiva na $[a, b]$ in naj $T(h)$ pomeni približek k vrednosti integrala* (6.1)*, izračunan s trapeznim pravilom. Potem je*

$$\int_a^b f(x)\,dx - T(h) = -\frac{h^2}{12}(b-a)f''(\xi).$$

Ta izrek pomeni, da lahko s trapeznim pravilom izračunamo približek, ki se od točne vrednosti integrala poljubno malo razlikuje, če le izračunamo vrednost funkcije, ki jo integriramo, v dovolj točkah. Ker je napaka sorazmerna h^2, se napaka zmanjša približno na četrtino, če podvojimo število točk.

Zapišimo algoritem, s katerim bomo lahko izračunali približek za določeni integral s pomočjo trapeznega pravila pri dani delitvi intervala:

Algoritem 6.1. Trapezno pravilo Naj bo f zvezna funkcija na intervalu $[a, b]$ in n neko naravno število. Naslednji algoritem izračuna približek T k vrednosti določenega integrala (6.1) s pomočjo trapeznega pravila.

$h = (b - a)/n$
$T = (f(a) + f(b))/2$
for $i = 1 : n - 1$
    $T = T + f(a + i * h)$
end
$T = T * h$

n	h	$T(h)$	$T(h) - I$
1	3	4.5000000	$-1.7 \cdot 10^{-1}$
2	1.5	4.6217082	$-4.5 \cdot 10^{-2}$
5	0.6	4.6592278	$-7.4 \cdot 10^{-3}$
10	0.3	4.6647957	$-1.9 \cdot 10^{-3}$
100	0.03	4.6666479	$-1.9 \cdot 10^{-5}$
1000	0.003	4.6666665	$-1.9 \cdot 10^{-7}$

Tabela 6.1: Približki za integral (6.6), izračunani s trapeznim pravilom

Delovanje trapeznega pravila si oglejmo na primeru.

Primer 6.1. S pomočjo trapeznega pravila izračunajmo vrednost integrala

$$I = \int_3^6 \sqrt{x-2}\,dx = \frac{14}{3} \tag{6.6}$$

za vrednosti $n = 1, 2, 5, 10, 100$ in 1000.
Vrednosti približkov in ustrezne napake so v tabeli 6.1. Opazimo lahko, da je napaka trapeznega pravila res sorazmerna kvadratu dolžine podintervalov h.

6.3 Metoda nedoločenih koeficientov

Iz napake trapeznega pravila vidimo, da s pomočjo (6.5) izračunamo natančno integral vsakega linearnega polinoma. To lahko posplošimo: konstruirajmo integracijsko formulo, ki bo natančno integrirala vse polinome, katerih stopnja ni večja od n. Ker je polinom stopnje n določen z izbiro $n+1$ prostih parametrov, lahko pričakujemo, da bomo tudi za integracijsko formulo potrebovali izračun vrednosti integranda f v $n+1$ točkah (*vozlih*). Tako bo imela iskana integracijska formula obliko

$$\int_a^{a+nh} f(x)\,dx = \sum_{i=0}^{n} a_i f(c_i) + R; \quad c_i = a + ih. \tag{6.7}$$

(Opomba: Tukaj smo vozle izbrali enakomerno na intervalu $[a, a+h]$, kar pa ni nujno. Tudi če bi abscise izbrali poljubno, bi po postopku, ki ga bomo sedaj opisali, lahko konstruirali ustrezne integracijske formule.)

Izpeljavo integracijskih formul z metodo nedoločenih koeficientov si oglejmo na konkretnem primeru. Vzemimo integracijsko formulo oblike

$$\int_0^{2h} f(x)\,dx = a_0 f(0) + a_1 f(h) + a_2 f(2h) + R \tag{6.8}$$

in določimo njene *uteži* a_0, a_1 in a_2 tako, da bo formula točna za vse kvadratne polinome.

V formulo (6.8) vstavimo namesto funkcije f zaporednoma polinome $p_0(x) = 1$, $p_1(x) = x$ in $p_2(x) = x^2$ in dobimo za uteži sistem linearnih

enačb

$$\begin{array}{ccccccc} a_0 & + & a_1 & + & a_2 & = & h \\ & & a_1/2 & + & a_2 & = & h/2 \\ & & a_1/4 & + & a_2 & = & h/3, \end{array}$$

ki ima rešitev $a_0 = a_2 = h/3$; $a_1 = 4h/3$. Tako smo dobili popularno *Simpsonovo*[1] *integracijsko formulo*

$$\int_0^{2h} f(x)\,dx = \frac{h}{3}[f(0) + 4f(h) + f(2h)] + R. \tag{6.9}$$

Če v formulo (6.9) vstavimo kubični polinom $f(x) = x^3$, in za napako predvidimo izraz $R = Cf'''(\xi)$, dobimo za konstanto napake C enačbo

$$\frac{(2h)^4}{4} = \frac{h}{3}(4h^3 + 8h^3) + 6Ch^4,$$

od koder izračunamo $C = 0$, kar pomeni, da je enostavna Simpsonova formula natančna tudi za polinome tretje stopnje. Da bi lahko izračunali konstanto napake, moramo v formulo (6.9) vstaviti polinom četrte stopnje $f(x) = x^4$, za napako pa predvidimo izraz $R = Df^{(4)}(\xi)$, od koder izračunamo

$$\frac{(2h)^5}{5} = \frac{h}{3}(4h^4 + 16h^4) + 24D.$$

Tako je $D = -h^5/90$, torej

$$\int_0^{h} f(x)\,dx = \frac{h}{6}[f(0) + 4f(h) + f(2h)] - \frac{h^5}{90}f^{(4)}(\xi). \tag{6.10}$$

Podobno kot trapezno formulo (6.4), lahko tudi Simpsonovo formulo uporabimo za računanje integralov na poljubnih intervalih tako, da interval razdelimo na $2n$ enakih podintervalov in na vsakem paru od njih uporabimo Simpsonovo formulo. Tako dobimo *Simpsonovo pravilo*

$$\begin{aligned} \int_a^b f(x)\,dx = \frac{h}{3}[&f(a) + 4f(a+h) + 2f(a+2h) + \cdots \\ + \;& 2f(b-2h) + 4f(b-h) + f(b)] + R, \end{aligned}$$

[1]Thomas Simpson (1710 Sutton Cheney – 1761 Market Bosworth, Anglija), angleški matematik, samouk. Integracijsko metodo, ki danes nosi njegovo ime, je objavil v svoji knjigi *The Doctrine and Application of Fluxions* leta 1750.

kjer je $h = (b-a)/2n$. Napaka je enaka

$$R = -h^4(b-a)f^{(4)}(\xi)/180, \tag{6.11}$$

kjer je ξ neka točka na (a, b).

Zapišimo še algoritem za izračun integrala s Simpsonovim pravilom:

Algoritem 6.2. Simpsonovo pravilo Naj bo f zvezna funkcija na intervalu $[a, b]$ in n neko naravno število. Naslednji algoritem izračuna približek S k vrednosti določenega integrala (6.1) s pomočjo Simpsonovega pravila.

$h = (b-a)/(2*n)$
$S = f(a) + f(b) + 4*f(a+h)$
for $i = 1 : n-1$
    $S = S + 2*f(a+2*i*h) + 4*f(a+2*i*h+h)$
end
$S = S*h/3$

n	h	$S(h)$	$S(h)-I$
1	1.5	4.66227766016838	$-4.39 \cdot 10^{-3}$
2	0.75	4.66622070830639	$-4.46 \cdot 10^{-4}$
5	0.3	4.66665163029280	$-1.50 \cdot 10^{-5}$
10	0.15	4.66666566830214	$-9.98 \cdot 10^{-7}$
100	0.015	4.66666666656452	$-1.02 \cdot 10^{-10}$
1000	0.0015	4.66666666666665	$-1.78 \cdot 10^{-14}$

Tabela 6.2: Približki za integral (6.6), izračunani s Simpsonovo formulo

Tudi delovanje Simpsonovega pravila si oglejmo na preimeru.

Primer 6.2. Vrednost integrala

$$I = \int_3^6 \sqrt{x-2}\,dx = \frac{14}{3} \tag{6.12}$$

za vrednosti $n = 1, 2, 5, 10, 100$ in 1000 izračunajmo še s pomočjo Simpsonovega pravila.
Vrednosti približkov in ustrezne napake so v tabeli 6.2. Napaka v zadnjem stolpcu je res približno sorazmerna četrti potenci dolžine podintervalov h, kot predvideva ocena (6.11).

6.4 Gaussove kvadraturne formule

Integracijske formule, ki smo jih spoznali v razdelku 6.2 in tista, ki jih lahko izpeljemo z metodo nedoločenih koeficientov iz razdelka 6.3, lahko zapišemo kot

$$\int_a^{a+h} f(x)\,dx \approx a_1 f(x_1) + a_2 f(x_2) + \cdots + a_n f(x_n), \tag{6.13}$$

kjer so *uteži* a_i neodvisne od izbire funkcije f. *Vozle* x_i smo določili vnaprej, pogosto kar ekvidistantno na intervalu $[a, a+h]$, nato pa izračunali uteži a_i tako, da je bila integracijska formula (6.13) reda n, kar pomeni, da je natančna za vse polinome stopnje $\leq n-1$.

Če pa v optimizacijo kvadraturne formule poleg uteži vključimo tudi izbiro vozlov, lahko red kvadraturne formule dvignemo do $2n$,kar pomeni, da bo formula točna za vse polinome, ki imajo stopnjo $\leq 2n-1$.

Primer 6.3. V integracijski formuli

$$\int_{-1}^{1} f(x)\,dx = a_1 f(x_1) + a_2 f(x_2) \tag{6.14}$$

določimo vozle x_i in uteži a_i $i = 1, 2$, do bo natančna za vse polinome reda ≤ 3.
Če za f, podobno kot smo storili v razdelku 6.3, vstavimo zaporedoma 1, x, x^2 in x^3, dobimo sistem nelinearnih enačb za neznane $a_1, a_2, x_1,$ in x_2

$$\begin{aligned} a_1 + \quad a_2 &= 2 \\ a_1 x_1 + \quad a_2 x_2 &= 0 \\ a_1 x_1^2 + \quad a_2 x_2^2 &= \frac{2}{3} \\ a_1 x_1^3 + \quad a_2 x_2^3 &= 0. \end{aligned}$$

Do rešitve tega sistema enačb pridemo preprosto: zaradi simetrije (v formulo (6.14) vstavimo $f(-x)$) mora biti $x_1 = -x_2$ in $a_1 = a_2$, kar že zadovolji drugo in zadnjo enačbo, tako da nam ostaneta le dve enačbi

$$\begin{aligned} 2a_1 &= 2 \\ 2a_1 x_1^2 &= \frac{2}{3}, \end{aligned}$$

kar nam da $x_1 = \frac{-1}{\sqrt{3}}$, $x_2 = \frac{1}{\sqrt{3}}$, $a_1 = a_2 = 1$, kar nam da končni razultat

$$\int_{-1}^{1} f(x)\,dx = f(-1/\sqrt{3}) + f(1/\sqrt{3}),$$

kar je formula, 4. reda (točna je za vse polinome stopnje ≤ 3).

Če vozla x_1 in x_2 primerjamo z Legendreovim polinomom $P_2(x)$ lahko vidimo, da sta x_1 in x_2 ravno ničli polinoma $P_2(x)$, kar pa ni naključje, saj velja

Izrek 6.2. *Naj bo $P_k(x), k = 1, \ldots$ zaporedje ortogonalnih polinomov za skalarni produkt $\langle f(x), g(x)\rangle = \int_a^b w(x)f(x)g(x)\,dx$. Ničle polinoma P_n naj bodo $x_1, x_2, \ldots, x_n$ in $a_1, a_2, \ldots, a_n$ rešitve sistema linearnih enačb*

$$\sum_{j=1}^{n} a_j x_j^m = \int_a^b w(u)u^m\,du\,.$$

Potem je:

1. *Integracijska formula*

$$\int_a^{a+h} w(x)f(x)\,dx \approx \sum_{j=1}^{n} a_j f(x_j) \tag{6.15}$$

   natančna za vse polinome stopnje $\leq 2n-1$ in

2. *nobena druga integracijska formula z n vozli ni natančna za vse polinome stopnje $\leq 2n$.*

Dokaz: Naj bo $p(x)$ poljuben polinom stopnje ne več kot $2n-1$. Potem obstaja natanko en polinom $q(x)$ (kvocient) in natanko en polinom $r(x)$ (ostanek), oba stopnje ne več kot $n-1$, da je $p = P_n q + r$. Zato je

$$\int_a^b p(u)w(u)\,du = \langle P_n, q\rangle + \int_a^b r(u)w(u)\,du = \int_a^b r(u)w(u)\,du,$$

saj je P_n ortogonalen na vse polinoma nižje stopnje. Poleg tega je

$$\sum_{j=1}^{n} a_j p(x_j) = \sum_{j=1}^{n} a_j P_n(x_j)q(x_j) + \sum_{j=1}^{n} a_j r(x_j) = \sum_{j=1}^{n} a_j r(x_j),$$

saj so x_j ničle polinoma $P_n(x)$. Tako nam ostane, da v formuli

$$\int_a^b r(u)w(u)\,du = \sum_{j=1}^{n} a_j r(x_j)$$

določimo uteži a_j, da bo formula točna za vse polinome stopnje $\leq n-1$, pa bo tudi

$$\int_a^b p(u)w(u)\,du = \sum_{j=1}^{n} a_j p(x_j),$$

kar pomeni, da je formula točna za vse polinome, katerih stopnja na presega $2n-1$.
Da bi dokazali tudi drugo trditev izreka, moramo pokazati, da obstaja polinom stopnje $2n$, ki ga formula 6.15 ne more integrirati natančno. Vzemimo, da je za nek nabor vozlov x_j in uteži a_j formula 6.15 natančna za vse polinome stopmje $2n$. Torej mora biti natančna tudi za polinom

$$p(x) = (x-x_1)^2(x-x_2)^2\cdots(x-x_n)^2.$$

To pa ni mogoče, saj je

$$\int_a^b p(u)w(u)\,du > 0,$$

medtem ko je

$$\sum_{j=1}^{n} a_j p(x_j) = 0.$$

n	h	$k=1$	$k=2$	$k=3$	$k=4$
1	3.0	$1.00 \cdot 10^{-2}$	$6.82 \cdot 10^{-5}$	$9.50 \cdot 10^{-7}$	$1.74 \cdot 10^{-8}$
2	1.5	$2.58 \cdot 10^{-3}$	$4.93 \cdot 10^{-6}$	$2.12 \cdot 10^{-8}$	$1.27 \cdot 10^{-10}$
4	0.75	$6.49 \cdot 10^{-4}$	$3.23 \cdot 10^{-7}$	$3.77 \cdot 10^{-10}$	$6.30 \cdot 10^{-13}$
8	0.375	$1.62 \cdot 10^{-4}$	$2.04 \cdot 10^{-8}$	$6.12 \cdot 10^{-12}$	$1.78 \cdot 10^{-15}$
16	0.1875	$4.07 \cdot 10^{-5}$	$1.28 \cdot 10^{-9}$	$9.59 \cdot 10^{-14}$	
32	0.09375	$1.02 \cdot 10^{-5}$	$8.02 \cdot 10^{-11}$	$1.78 \cdot 10^{-15}$	
64	0.046875	$2.54 \cdot 10^{-6}$	$5.01 \cdot 10^{-12}$		

Tabela 6.3: Napaka pri izračunu vrednost integrala (6.6) s sestavljeno Gauss-Legendrovo kvadraturno formulo s k točkami pri koraku h.

Ker imajo Gaussove kvadraturne formule visok red točnosti, so zelo učinkovite, predvsem pri integraciji gladkih funkcij, saj je za primerljivo natančnost potrebno precej manj izračunov vrednosti funkcije. Gaussove kvadraturne formule, ki ustrezajo utžni funkciji $w(x) = 1$ (Gauss-Legendrove kvadraturne formule) lahko uporabimo tudi kot sestavljena pravila. Za izračun integrala

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

interval $[a, b]$ razdelimo s točkami $a = x_0 < x_1 < x_2 < \cdots < x_N = b$ na N podintervalov $[x_{i-1}, x_i]$, potem pa vsakega od N integralov

$$I_i = \int_{x_{i-1}}^{x_i} f(x)\,dx \quad \text{za} \quad i = 1, \ldots, N$$

najprej s pomočjo zamenjave spremenljivke

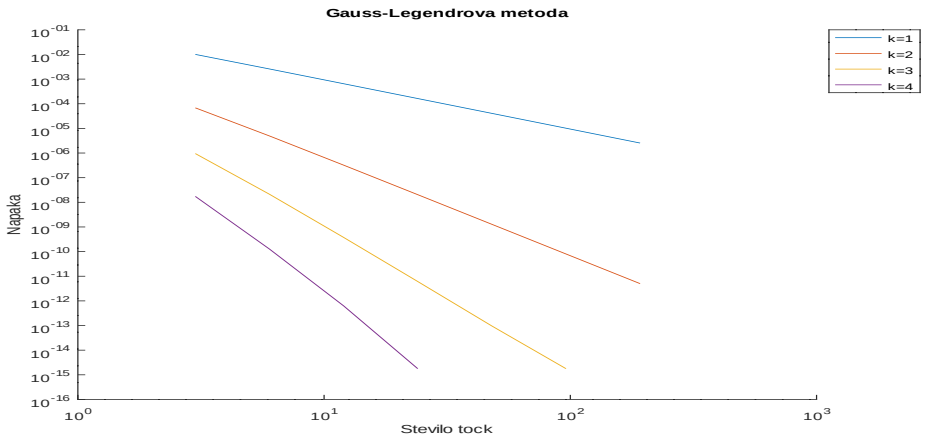
$$x = \frac{x_i - x_{i-1}}{2} t + \frac{x_{i-1} + x_i}{2}$$

prevedemo na integral

$$I_i = \frac{x_i - x_{i-1}}{2} \int_{-1}^{1} f\left(\frac{x_i - x_{i-1}}{2} t + \frac{x_{i-1} + x_i}{2}\right) dt,$$

ga izračunamo z Gauss-Legendrovo kvadraturno formulo in dobljene rezultate seštejemo

$$I = I_1 + \cdots I_N.$$



Slika 6.2: Napaka pri izračunu vrednosti integrala (6.6) z Gauss-Legendrovimi kvadraturnimi sestavljenimi pravili na $1, 2, 3$ in 4 točkah in dolžino koraka $h = 3/2^j$ za $j = 0, 1, \ldots, 7$.

Poglejmo si primer.

Primer 6.4. Izračunajmo integral $\int_3^6 \sqrt{x-2}\,dx$ še z Gauss-Legendrovimi metodami na k točkah za $k-1, 2, 3, 4$. Na sliki 6.2 vidimo, kako se obnaša napaka rezultata v odvisnosti od števila podintervalov. V tabeli 6.3 so napake teh formul.

6.5 Izbira koraka

Iz približka za napako trapeznega pravila

$$T(h) - I = (b-a)\frac{h^2}{12}f''(\xi)$$

(izrek 6.1) lahko sklepamo, kako se zmanjšuje napaka, ko $h \to 0$. Za praktično ocenjevanje napake pa ta formula ni primerna, saj zahteva poznavanje drugega odvoda funkcije f. Da bi dobili izračunljiv približek za velikost napake, izračunamo s trapezno formulo približek tudi pri polovični dolžini koraka in dobimo približno 4-krat manjšo napako

$$T(h/2) - I = \frac{h^2}{4}\frac{b-a}{12}f''(\eta),$$

kjer je $\eta \in [a,b]$ različen od ξ. Kljub temu pa lahko smatramo faktor $(b-a)f''/12$ kot 'približno' konstanto[2] (označili jo bomo s C), zato lahko iz enačb

$$\begin{aligned} I &= T(h) - Ch^2 + O(h^4) \\ I &= T(h/2) - \frac{C}{4}h^2 + O(h^4) \end{aligned}$$

izračunamo glavni del napake tako, da ti enačbi odštejemo

$$T(h) - T(h/2) = \frac{3}{4}Ch^2 + O(h^4), \tag{6.16}$$

zato je

$$Ch^2 \approx \frac{4}{3}(T(h) - T(h/2)).$$

Tako je $4(T(h)-T(h/2))/3$ uporabni približek za napako $T(h)-I$ (in $(T(h)-T(h/2))/3$ približek za napako $T(h/2) - I$). Pri računanju $T(h/2)$ pri tem ni potrebno računati vrednosti funkcije f v abscisah, ki smo jih že uporabili pri računanju $T(h)$, ker velja

$$T(h/2) = \frac{T(h)}{2} + \frac{h}{2}\left(\sum_{i=1}^{n} f(a + (i-1/2)h)\right),$$

[2]Pravzaprav lahko napako trapeznega pravila kot funkcijo dolžine koraka h razvijemo v potenčno vrsto, ki vsebuje le sode potence spremenljivke h:

$$f''(\eta) = f''(\zeta) + C_1h^2 + C_2h^4 + \cdots,$$

kar bomo uporabili kasneje pri Rombergovi metodi.

kjer smo z n označili $n = (b-a)/h$.

Če želimo izračunati približek $T(h)$ za I, da bo $|T(h)-I| < \varepsilon$, izračunamo najprej $T(b-a) = (b-a)(f(a)+f(b))/2$ in

$$T\left(\frac{b-a}{2}\right) = \frac{T(b-a)}{2} + \frac{b-a}{2} f\left(\frac{a+b}{2}\right),$$

ter približek za napako $(T(b-a)-T((b-a)/2))/3$. Če je absolutna vrednost približka za napako manjša od predpisane natančnosti ε, je $T((b-a)/2)$ dovolj dober približek za I, sicer korak h razpolovimo in izračun ponovimo.

Celoten postopek zapišimo kot algoritem:

Algoritem 6.3. Trapezno pravilo s kontrolo napake Naj bo f zvezna funkcija na intervalu $[a,b]$, N neko naravno število in $\varepsilon > 0$. Naslednji algoritem izračuna s pomočjo trapezne formule približek T, ki se od vrednosti določenega integrala (6.1) razlikuje manj kot ε ali pa se po N razpolovitvah dolžine koraka konča brez rezultata ($T = NaN$).

```
e = 2 * ε
m = 0
h = b − a
T = h * (f(a) + f(b))/2
while (m < N)&(abs(e) > ε)
    m = m + 1
    h = h/2                    % Ponovno, s polovičnim korakom
    k = 2^(m − 1)              % Nove abscise
    s = 0
    for i = 1 : k
        s = s + f(a + (2 * i − 1) * h)
    end
    e = s * h − T/2;           % Ocena za napako
    T = T + e                  % Nov približek
end
if abs(e) > ε
    T = NaN                    % Približek ni dober
end
```

ε	n	h	$T(h)$	$T(h) - I$
10^{0}	2^{1}	$1.5 \cdot 10^{-0}$	4.621708245	$-4.5 \cdot 10^{-2}$
10^{-1}	2^{2}	$7.5 \cdot 10^{-1}$	4.655092593	$-1.2 \cdot 10^{-2}$
10^{-2}	2^{3}	$3.7 \cdot 10^{-1}$	4.663746678	$-2.9 \cdot 10^{-3}$
10^{-3}	2^{5}	$9.4 \cdot 10^{-2}$	4.666483600	$-1.8 \cdot 10^{-4}$
10^{-4}	2^{7}	$2.3 \cdot 10^{-2}$	4.666655223	$-1.1 \cdot 10^{-5}$
10^{-5}	2^{8}	$1.2 \cdot 10^{-2}$	4.666663806	$-2.9 \cdot 10^{-6}$
10^{-6}	2^{10}	$2.9 \cdot 10^{-3}$	4.666666488	$-1.8 \cdot 10^{-7}$
10^{-7}	2^{12}	$7.3 \cdot 10^{-4}$	4.666666655	$-1.1 \cdot 10^{-8}$
10^{-8}	2^{13}	$3.7 \cdot 10^{-4}$	4.666666664	$-2.8 \cdot 10^{-9}$

Tabela 6.4: Približki za integral (6.6), izračunani s trapezno metodo s kontrolo napake pri različnih ε. Število n pomeni število izračunov funkcije f.

Primer 6.5. Izračunajmo približek za vrednost integrala (6.12) še z algoritmom 6.3 pri različnih vrednostih parametra ε.
Rezultati so v tabeli 6.4. Opazimo lahko, da je napaka $T(h) - I$ vedno manjša od predpisane natančnosti ε.

Adaptivna izbira koraka Sestavljena pravila, ki smo jih opisali do sedaj, so temeljila na delitvi integracijskega intervala na *enake* podintervale. Ta izbira je naravna in včasih (predvsem kadar imamo integracijsko funkcijo znano le v posameznih, enakomerno razporejenih točkah) edina možna. Kadar pa znamo vrednost funkcije f izračunati v vsaki točki integracijskega intervala, je včasih bolje razdeliti celoten interval na podintervale, katerih dolžine so odvisne od obnašanja funkcije na vsakem od podintervalov. To nam omogoča, da izračunamo približno vrednost integrala s predpisano natančnostjo z manj računanji funkcijske vrednosti, kot če bi bili vsi podintervali enake dolžine.

Vzemimo za primer *splošno* trapezno pravilo

$$I = \sum_{i=1}^{n} \frac{h_i}{2}[f(x_{i-1}) + f(x_i)] - \sum_{i=1}^{n} \frac{h_i^3}{12} f''(\xi_i),$$

kjer so delilne točke $a = x_0 < x_1 < \cdots < x_{n-1} < x_n = b$, ne nujno enakomerno razporejene in $h_i = x_i - x_{i-1}$. Pri tem je prispevek podintervala

$[x_{i-1}, x_i]$ k celotni napaki enak

$$-\frac{h_i^3}{12} f''(\xi_i) \quad \text{za neki} \quad \xi_i \in (x_{i-1}, x_i)$$

in je odvisen od dolžine h_i podintervala in od vrednosti $f''(x)$ na podintervalu (x_{i-1}, x_i). Tako lahko na tistem delu intervala $[a, b]$, kjer je $f''(x)$ majhen, vzamemo 'dolge' podintervale, kjer je $f''(x)$ velik, pa 'kratke', če hočemo, da bodo prispevki delnih napak približno sorazmerni dožini podintervalov.

Integracijske metode, ki sproti prilagajajo dolžine podintervalov glede na lokalno obnašanje integranda, imenujemo *adaptivne*. Glavna težava, s katero se srečujemo pri adaptivnih metodah je, da ne poznamo odvoda, ki nastopa v izrazu za napako, zato moramo, podobno kot v algoritmu 6.3, napako ocenjevati sproti.

Opisali bomo adaptivno integracijsko metodo na osnovi trapezne formule. Podobno bi lahko za osnovo adaptivne metode vzeli Simpsonovo ali kakšno drugo formulo.

Želimo izračunati približek, ki se od prave vrednosti integrala (6.1) ne bo razlikoval za več kot ε in pri tem čimmanjkrat izračunati vrednost funkcije f.

Vrednost integrala na vsakem od podintervalov

$$I_i = \int_{x_{i-1}}^{x_i} f(x)\, dx$$

računamo dvakrat s trapezno formulo:

$$\begin{aligned} T(h_i) &= \frac{h_i}{2}[f(x_{i-1}) + f(x_i)], \\ T(h_i/2) &= T(h_i)/2 + \frac{h_i}{2} f(x_{i-1} + h_i/2), \end{aligned} \tag{6.17}$$

da dobimo

$$\begin{aligned} I_i &= T(h_i) - Ch_i^3 + O(h_i^5) \\ I_i &= T(h_i/2) - 2\frac{C}{8} h_i^3 + O(h_i^5). \end{aligned}$$

Iz teh dveh približkov lahko ocenimo napako (pravzaprav konstanto C) podobno kot z enačbo (6.16):

$$T(h_i) - T(h_i/2) = \frac{3C}{4} h_i^3 + O(h_i^5), \tag{6.18}$$

tako da je napaka na tem intervalu približno enaka

$$I_i - T(h_i/2) \approx \frac{T(h_i) - T(h_i/2)}{3}.$$

Če naj bo napaka integrala na celotnem intervalu $[a, b]$ manjša od ε, je smiselna zahteva, naj bo napaka na podintervalu z dolžino h_i manjša od $h_i\varepsilon/(b-a)$. Delni rezultat (6.17) torej sprejmemo, če je

$$\frac{|T(h_i) - T(h_i/2)|}{3} < h_i\varepsilon/(b-a),$$

sicer pa moramo vzeti manjši podinterval. Na osnovi ocene napake tudi lahko določimo optimalno dolžino naslednjega podintervala. Ker mora napaka na naslednjem koraku zadoščati neenačbi

$$\frac{C}{4} h_{i+1}^3 < \frac{\varepsilon h_{i+1}}{b-a},$$

izberemo dolžino koraka

$$h_{i+1} = \sigma \sqrt{\frac{3\varepsilon h_i^3}{(b-a)|T(h_i) - T(h_i/2)|}},$$

kjer smo s σ označili 'varnostni koeficient', ki ga navadno izberemo malo manj kot 1 (npr. $\sigma = 0.9$).

Zapišimo celoten algoritem.

Algoritem 6.4. Adaptivno trapezno pravilo Naj bo f zvezna funkcija na intervalu $[a, b]$ in $\varepsilon > 0$. Naslednji algoritem izračuna s pomočjo trapezne formule približek I, ki se od vrednosti določenega integrala (6.1) razlikuje manj kot ε.

$\sigma = 0.9$
$x = a$
$I = 0$
$h = b - a$
$f_x = f(a)$
$f_{xh} = f(b)$
while $x < b$
    $f_{xh/2} = f(x + h/2)$
    $T_1 = h * (f_x + f_{xh})/2$ % Najprej z osnovnim korakom
    $T_2 = T_1/2 + h * f_{xh/2}/2$ % Ponovno, s polovičnim korakom
    if abs$((T_1 - T_2)/3) < h * \varepsilon/(b - a)$
        $x = x + h$ % Rezultat sprejet
        $I = I + T_2$ % Prištejemo delni rezultat
        $f_x = f_{xh}$
        $h = \sigma * \mathbf{sqrt}(3 * h^3 * \varepsilon/\mathbf{abs}(T_1 - T_2)/(b - a))$
        % novi korak
        if $x + h > b$ % Ali smo že blizu konca?
            $h = b - x$
        end
        $f_{xh} = f(x + h)$
    else % Rezultat zavrnjen
        $h = h/2$ % Razpolovimo korak
        $f_{xh} = f_{xh/2}$ % Vrednost v končni točki
    end
end

ε	M_f	N_f
10^0	4	3
10^{-1}	13	17
10^{-2}	38	65
10^{-3}	87	257
10^{-4}	211	1025
10^{-5}	578	8193
10^{-6}	1709	?
10^{-7}	5251	?

Tabela 6.5: Število izračunov funkcije f pri računanju vrednosti integrala (6.19) s trapeznim pravilom s kontrolo napake (N_f) in z adaptivnim trapeznim pravilom M_f

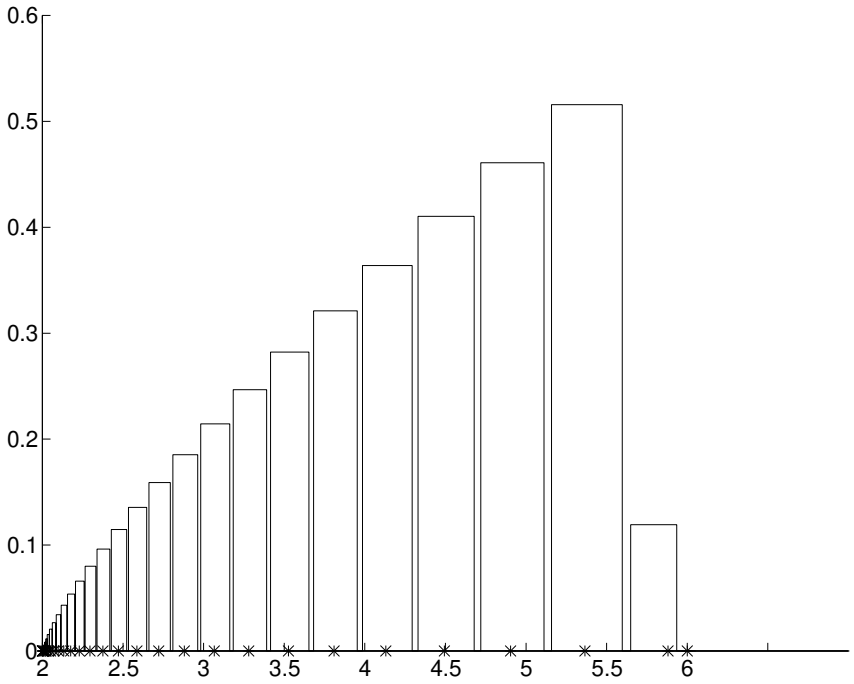
Primer 6.6. Izračunajmo približno vrednost integrala

$$\int_2^6 \sqrt{x-2}\,dx \tag{6.19}$$

z algoritmoma 6.3 in 6.4 pri različnih vrednostih parametra ε.
Rezultati so v tabeli 6.5. V stolpcu N_f je število izračunov vrednosti funkcije f z algoritmom 6.3, v koloni M_f pa število izračunov z adaptivnim algoritmom 6.4. Vsi izračunani rezultati zadoščajo zahtevani natančnosti. Prepričamo se lahko, da je adaptivni algoritem precej bolj učinkovit, saj prilagaja korak integracije lokalnemu obnašanju funkcije. Ta se v našem primeru hitro spreminja v bližini spodnje meje, kjer je potreben zelo majhen korak, nato pa je proti zgornji meji vse bolj položna in zato tam lahko uporabimo mnogo daljši korak. Algoritem s kontrolo napake je pri večjih natančnostih zahteval preveliko število funkcijskih izračunov, kar je v tabeli označeno z "?". Na sliki 6.3 je prikazano, kako adaptivni algoritem 6.4 spreminja dolžino koraka pri natančnosti $\varepsilon = 10^{-3}$.

6.6 Rombergova metoda

Kadar je funkcija, ki jo integriramo vsaj $2k+2$-krat odvedljiva, se da pokazati, da lahko napako trapezne formule razvijemo v konvergentno vrsto po sodih



Slika 6.3: Spreminjanje dolžine koraka pri adaptivni integraciji

potencah h:

$$I = \int_a^b f(x)\,dx = T(h) + C_1h^2 + C_2h^4 + C_3h^6 + \cdots + C_kh^{2k} + O(h^{2k+2}),$$

kjer konstante $C_1, \ldots, C_k$ niso odvisne od h. Že v (6.16) smo izračunali približek za napako trapezne formule tako, da smo izračunali $T(h)$ in $T(h/2)$ in dobili

$$C_1h^2 = \frac{4}{3}[T(h) - T(h/2)] + O(h^4).$$

Če za vrednost te približne napake popravimo rezultat, dobimo boljši približek za I

$$T(h, h/2) = T(h) - \frac{4}{3}[T(h) - T(h/2)] = \frac{4T(h/2) - T(h)}{3},$$

tako da je

$$I = T(h, h/2) + Dh^4 + O(h^6), \tag{6.20}$$

kjer je D spet neka konstanta, neodvisna od h. Če izračunamo še

$$T(h/2, h/4) = \frac{4T(h/4) - T(h/2)}{3},$$

za katerega velja

$$I = T(h/2, h/4) + \frac{D}{16}h^4 + O(h^6), \tag{6.21}$$

lahko enačbi (6.20) in (6.21) odštejemo:

$$0 = T(h, h/2) - T(h/2, h/4) + \frac{15}{16}Dh^4 + O(h^6).$$

Ko iz te enačbe izračunamo napako in popravimo rezultat, dobimo

$$T(h, h/2, h/4) = \frac{16T(h/2, h/4) - T(h, h/2)}{15},$$

za katerega velja

$$I = T(h, h/2, h/4) + Eh^6 + O(h^8),$$

kjer je E zopet neka konstanta, neodvisna od h. Tako lahko nadaljujemo in dobimo na k-tem koraku

$$T(h, \ldots, h/2^k) = \frac{4^k T(h/2, \ldots, h/2^k) - T(h, \ldots, h/2^{k-1})}{4^k - 1}, \tag{6.22}$$

in velja

$$I = T(h, \ldots, h/2^k) + Fh^{2k+2} + O(h^{2k+4}).$$

Da bi iz enačbe (6.22) izračunali $T(h, \ldots, h/2^k)$, moramo prej izračunati vrednosti $T(h, \ldots, h/2^{k-1})$ in $T(h/2, \ldots, h/2^k)$, da bi jih izračunali, potrebujemo tudi $T(h, \ldots, h/2^{k-2})$, $T(h/2, \ldots, h/2^{k-1})$ in $T(h/4, \ldots, h/2^k)$, ... in končno $T(h/2^k)$, ..., $T(h/2)$ in $T(h)$. Vse te vrednosti najlaže predstavimo v obliki *Rombergove*[3] *tabele*

$$\begin{array}{llll} T(h) & & & \\ T(h/2) & T_1(h/2) & & \\ T(h/4) & T_1(h/4) & T_2(h/4) & \\ \vdots & \vdots & \vdots & \ddots \\ T(h/2^k) & T_1(h/2^k) & T_2(h/2^k) & \cdots \quad T_k(h/2^k) \end{array} \tag{6.23}$$

[3]Werner Romberg (1909 Berlin – 2003 Heidelberg), nemški matematik. Pred nacizmom se je omaknil na Norveško in kasneje na Švedsko, po vojni je bil profesor v Trondheimu in kasneje v Heidelbergu. Ukvarjal se je z numerično analizo in uporabno matematiko.

Pozor! Približki v drugem stolpcu tabele T_1 so isti, kot če bi jih izračunali s Simpsonovim pravilom (glej problem 2).

Zapišimo algoritem, ki bo izračunal Rombergove približke:

Algoritem 6.5. Rombergova metoda Naj bo f zvezna funkcija na intervalu $[a, b]$ in k naravno število. Naslednji algoritem izračuna približke $T_n(h/2^m)$ iz Rombergove tabele.

```
h = b − a
T(1,1) = h ∗ (f(a) + f(b))/2
for m = 2 : k + 1
      h = h/2
      T(m,1) = T(m − 1,1)/2
      s = 0
      for i = 1 : 2^(m − 2)
            s = s + f(a + (2 ∗ i − 1) ∗ h)
      end
      T(m,1) = T(m,1) + h ∗ s
      for n = 2 : m
            T(m,n) = (4^(n − 1) ∗ T(m,n − 1) − T(m − 1,n − 1))/
                  (4^(n − 1) − 1)
      end
end
```

Rombergovo metodo uporabimo na primeru:

m	$I - T(h)$	$I - T(h, \frac{h}{2})$	$I - T(h, \frac{h}{2}, \frac{h}{4})$	$I - T(h, \ldots, \frac{h}{8})$	$I - T(h, \ldots, \frac{h}{16})$
1	$1.6 \cdot 10^{-1}$				
2	$4.5 \cdot 10^{-2}$	$4.4 \cdot 10^{-3}$			
3	$1.2 \cdot 10^{-2}$	$4.5 \cdot 10^{-4}$	$1.8 \cdot 10^{-4}$		
4	$2.9 \cdot 10^{-3}$	$3.5 \cdot 10^{-5}$	$7.9 \cdot 10^{-6}$	$5.1 \cdot 10^{-6}$	
5	$7.3 \cdot 10^{-4}$	$2.4 \cdot 10^{-6}$	$2.2 \cdot 10^{-7}$	$9.3 \cdot 10^{-8}$	$7.3 \cdot 10^{-8}$
6	$1.8 \cdot 10^{-4}$	$1.5 \cdot 10^{-7}$	$4.2 \cdot 10^{-9}$	$8.9 \cdot 10^{-10}$	$5.3 \cdot 10^{-10}$

Tabela 6.6: Napake posameznih približkov integrala (6.12), izračunanih z Rombergovo metodo

Primer 6.7. Izračunajmo približek za vrednost integrala (6.12) še z Rombergovo metodo (algoritem 6.5). Izberimo vrednost parametra $k = 5$. V tabeli 6.6 so napake posameznih približkov. Iz rezultatov vidimo, da so približki vse bolj natančni, ko gremo po posameznem stolpcu navzdol, kar pomeni vse krajši korak h trapezne formule. Prav tako pa opazimo, da se napake manjšajo, ko gremo po posamezni vrstici z leve proti desni, kar pomeni vse natančnejše integracijske formule. Vidimo tudi, da se to zmanjševanje napake ustavi, kar je posledica zaokrožitvenih napak, saj so rezultati izračunani pri osnovni zaokrožitveni napaki $\approx 2.2 \cdot 10^{-16}$.

6.7 Računanje posplošenih integralov

Večkrat se zgodi, da ima funkcija, katere določeni integral moramo izračunati, na integracijskem intervalu singularnost, npr.

$$\int_0^1 \frac{dx}{\sqrt{x}} = 2. \tag{6.24}$$

V tem primeru ne moremo uporabiti trapezne ali Simpsonove formule, ker je vrednost funkcije v točki $x = 0$ nedefinirana. Z metodo nedoločenih koeficientov iz prejšnjega razdelka bi sicer lahko konstruirali integracijsko formulo, ki ne bi vsebovala točke 0 kot abscise, vendar bi dobili še vedno dokaj slab približek, saj napaka take formule vsebuje ustrezni višji odvod funkcije, ta pa je neomejen. Boljša rešitev je, da singularnost upoštevamo že pri konstrukciji integracijske formule same. Kako to naredimo, si oglejmo kar na zgledu integrala

$$I = \int_0^1 \frac{f(x)}{\sqrt{x}}\, dx,$$

kjer je funkcija $f(x)$ na intervalu $[0, 1]$ regularna.

Z metodo nedoločenih koeficientov bomo določili uteži integracijske formule, ki ima abscisi v točkah 0 in 1:

$$I \approx a_0 f(0) + a_1 f(1)$$

tako, da bo le-ta natančna za konstante in linearne polinome. Ko za $f(x)$ vstavimo konstanto 1, dobimo prvo enačbo

$$\int_0^1 x^{-1/2}\, dx = 2 = a_0 + a_1,$$

pri $f(x) = x$ pa dobimo drugo enačbo

$$\int_0^1 x^{1/2}\,dx = \frac{2}{3} = a_1.$$

Rešitvi tega sistema sta $a_0 = 4/3$ in $a_1 = 2/3$, tako da imamo integracijsko formulo

$$\int_0^1 \frac{f(x)}{\sqrt{x}}\,dx \approx \frac{4}{3}f(0) + \frac{2}{3}f(1). \tag{6.25}$$

Kadar želimo izračunati podoben integral na drugem intervalu, npr. na $[a, a+h]$, moramo zamenjati spremenljivko: $t = a + hx$, torej

$$\int_0^1 \frac{f(x)}{\sqrt{x}}\,dx = \frac{1}{\sqrt{h}}\int_a^{a+h} \frac{f(\frac{t-a}{h})}{\sqrt{t-a}}\,dt,$$

tako da je integracijska formula v tem primeru

$$\int_a^{a+h} \frac{f(t)}{\sqrt{t-a}}\,dt \approx \sqrt{h}\left(\frac{4}{3}f(a) + \frac{2}{3}f(a+h)\right). \tag{6.26}$$

Na isti način lahko tudi pri podobnih singularnih integralih izračunamo uteži integracijske formule oblike

$$\int_a^{a+h} w(x)f(x)\,dx \approx \sum_{i=1}^{n} a_i f(a + c_i h),$$

kjer je funkcija $f(x)$ regularna, $w(x)$ singularna na $[a, a+h]$, števila c_i; $i = 1, \ldots, n$ (vozli integracijske formule) pa poljubna, med seboj različna na $[0, 1]$.

Kadar računamo vrednost singularnega integrala s sestavljeno integracijsko formulo (npr. Simpsonovim pravilom), lahko na vseh podintervalih, kjer je funkcija regularna, uporabimo Simpsonovo formulo, le na podintervalih, kjer ima funkcija singularnost moramo uporabiti posebno formulo za singularne integrale.

Primer 6.8. Z integracijsko formulo (6.26) izračunajmo približek za vrednost integrala

$$\int_0^{0.1} \left(\frac{\cos x}{2\sqrt{x}} - \sqrt{x} \sin x \right) dx \approx \sqrt{0.1}\, \frac{2 + \cos 0.1 - 0.2 \sin 0.1}{3} \approx 0.31360.$$

Točna vrednost tega integrala je

$$\sqrt{0.1} \cos 0.1 \approx 0.31465.$$

Če hočemo natančnejši rezultat, moramo interval $[0, 0.1]$ razdeliti na več podintervalov, na prvem uporabimo formulo (6.25), na ostalih pa trapezno ali Simpsonovo pravilo.

Pogosto si pri računanju posplošenih integralov lahko pomagamo z Gaussovimi kvadraturnimi formulami. Če ima integrand singularnost v enem ali obeh krajiščih integracijskega intervala, lahko integral zapišemo kot

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

kjer je $w(x)$ nenegativna integrabilna funkcija, $g(x) = f(x)/w(x)$ pa gladka funkcija. V tem primeru lahko uporabimo Gaussovo kvadraturno formulo

$$\int_a^b w(x) g(x)\, dx \approx \sum_{j=1}^{n} a_j g(x_j),$$

pri kateri za vozle x_j izberemo ničle n-tega oertogonalnega polinoma za skalarni produkt

$$\langle f, g \rangle = \int_a^b w(x) f(x) g(x)\, dx,$$

uteži a_j pa optimalno glede na izbrane vozle. Oglejmo si primer.

k	I_k	$I_k - I$
1	3.141592653589793	$-8.36 \cdot 10^{-1}$
2	3.960266052790758	$-1.72 \cdot 10^{-2}$
3	3.977321960082316	$-1.41 \cdot 10^{-4}$
4	3.977462634661957	$-6.26 \cdot 10^{-7}$
5	3.977463258776694	$-1.73 \cdot 10^{-9}$
6	3.977463260503157	$-3.26 \cdot 10^{-12}$
7	3.977463260506418	$-3.55 \cdot 10^{-15}$

Tabela 6.7: Približki in napake Gauss-Čebiševe kvadraturne formule na k točkah pri izračunu vrednosti integrala (6.27) pri različnih k

Primer 6.9. Izračunajmo vrednost integrala

$$I = \int_{-1}^{1} \frac{e^x}{\sqrt{1-x^2}}\, dx\,. \tag{6.27}$$

Uporabili bono Gauss-Čebiševo kvadraturno formulo, zato je $w(x) = 1/\sqrt{1-x^2}$ in $f(x) = e^x$. Vozli so ničle k-tega polinoma Čebiševa, uteži kvadraturne formule pa so vse enake π/k. Približke za integral (6.27) s k vozli (torej reda $2k$) računamo kot

$$I_k = \frac{\pi}{k} \sum_{j=1}^{k} e^{\cos((2j-1)\pi/2k)}.$$

Približki I_k in njihove napake $I_k - I$ za različne k so v tabeli 6.7.

6.8 Numerično odvajanje

Poglejmo si še, kako lahko numerično izračunamo vrednost odvoda f' funkcije f v določeni točki. Ker je odvod funkcije f v točki x definiran kot

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h},$$

dobimo najenostavnejšo formulo (*direktno formulo*) za izračun odvoda tako, da vzamemo dovolj majhen h in je

$$f'(x) \approx \frac{f(x+h) - f(x)}{h}. \tag{6.28}$$

Da bi ugotovili, kakšna je napaka pri uporabi formule (6.28), razvijemo $f(x+h)$ (predpostavimo, da je f vsaj dvakrat odvedljiva) po Taylorjevi[4] formuli

$$f(x+h) = f(x) + hf'(x) + \frac{h^2}{2} f''(\xi),$$

kjer je ξ neka točka med x in $x+h$. Tako dobimo

$$f'(x) = \frac{f(x+h) - f(x)}{h} - \frac{h}{2} f''(\xi), \tag{6.29}$$

torej je napaka približka, izračunanega s formulo (6.28), sorazmerna h.

Do natančnejših formul za numerično računanje odvoda lahko pridemo bodisi z odvajanjem interpolacijskega polinoma, bodisi z metodo nedoločenih koeficientov. Poglejmo si oba načina na preprostih primerih.

Odvajanje interpolacijskega polinoma. Naj bo $p_1(x)$ linearni polinom, ki interpolira vrednosti funkcije f v točkah $x_0 - h$ in x_0, torej po Newtonovi interpolacijski formuli (5.7)

$$\begin{aligned} p_1(x) &= f(x_0 - h) + f[x_0 - h, x_0](x - (x_0 - h)) \\ &= f(x_0 - h) + \frac{f(x_0) - f(x_0 - h)}{h}(x - x_0 + h), \end{aligned}$$

kar odvajamo

$$f'(x) \approx p_1'(x) = \frac{f(x_0) - f(x_0 - h)}{h}.$$

Tako smo dobili *obratno formulo*, ki je podobna, kot direktna formula (6.28), le da h zamenjamo z $-h$, zato je tudi napaka te formule enaka $hf''(\eta)/2$, le da je točka η v tem primeru med $x_0 - h$ in x_0. Na podoben način dobimo lahko tudi natančnejše formule za izračun vrednosti odvoda, če za osnovo vzamemo interpolacijski polinom skozi več točk.

[4]Brook Taylor (1685 Edmonton – 1731 London), angleški matematik, danes predvsem poznan po Taylorjevi vrsti. Objavil jo je leta 1715 v svoji knjigi *Methodus incrementorum directa et inversa*. Podoben rezultat je opisal Johan Bernoulli že leta 1694.

Metoda nedoločenih koeficientov. Odvod funkcije f aproksimiramo kot linearno kombinacijo njenih vrednosti v sosednjih točkah — vzemimo kar točke $x_0 - h$, x_0 in $x_0 + h$:

$$f'(x_0) \approx Af(x_0 - h) + Bf(x_0) + Cf(x_0 + h). \tag{6.30}$$

Da bi lahko izračunali koeficiente A, B in C, zapišimo aproksimaciji za $f(x_0 - h)$ in $f(x_0 + h)$ po Taylorjevi formuli

$$\begin{aligned} f(x_0 - h) &= f(x_0) - hf'(x_0) + \frac{h^2}{2}f''(x_0) - \frac{h^3}{6}f'''(x_0) + \cdots \\ f(x_0) &= f(x_0) \\ f(x_0 + h) &= f(x_0) + hf'(x_0) + \frac{h^2}{2}f''(x_0) + \frac{h^3}{6}f'''(x_0) + \cdots \ . \end{aligned}$$

Dobljene razvoje vstavimo v enačbo (6.30) in dobimo

$$\begin{aligned} f'(x_0) &= (A + B + C)f(x_0) + (-A + C)hf'(x_0) \\ &+ \frac{A + C}{2}h^2 f''(x_0) + \frac{-A + C}{6}h^3 f'''(x_0) + \cdots . \end{aligned} \tag{6.31}$$

S primerno izbiro konstant A, B in C lahko eliminiramo člena, ki vsebujeta f in f'', konstanta pri f' pa mora biti enaka $1/h$. Tako lahko zapišemo sistem enačb za konstante A, B in C

$$\begin{array}{rcrcrcr} A & + & B & + & C & = & 0 \\ -A & & & + & C & = & 1/h \\ A & & & + & C & = & 0, \end{array}$$

ki ima rešitev $A = -1/2h$, $B = 0$ in $C = 1/2h$, od koder dobimo *sredinsko formulo* za izračun odvoda

$$f'(x_0) \approx \frac{f(x_0 + h) - f(x_0 - h)}{2h}. \tag{6.32}$$

Napako te formule izračunamo tako, da funkcijo

$$f'(x_0) - \left(\frac{f(x_0 + h) - f(x_0 - h)}{2h}\right) \tag{6.33}$$

razvijemo po Taylorjevi formuli. Tako pridemo do rezultata

$$f'(x_0) = \frac{f(x_0 + h) - f(x_0 - h)}{2h} - \frac{h^2}{6}f'''(x_0) + O(h^3), \tag{6.34}$$

Iz primerjave direktne, obratne in sredinske formule vidimo, da je sredinska natančnejša od ostalih dveh, saj ima napaka faktor h^2 v primerjavi z direktno in obratno, ki imata obe v napaki faktor h.

Drugi odvod Podobno kot smo z nastavkom (6.31) izrazili prvi odvod, lahko z linearno kombinacijo funkcijskih vrednosti $f(x_0-h)$, $f(x_0)$ in $f(x_0+h)$ izrazimo tudi drugi odvod $f''(x_0)$:

$$\begin{aligned} f''(x_0) &\approx (A+B+C)f(x_0)+(-A+C)hf'(x_0) \\ &\quad +\frac{A+C}{2}h^2f''(x_0)+\frac{-A+C}{6}h^3f'''(x_0)\ldots, \end{aligned} \tag{6.35}$$

od koder za koeficiente A, B in C dobimo sistem linearnih enačb

$$\begin{array}{rcrcrl} A & + & B & + & C & = & 0 \\ -A & & & + & C & = & 0 \\ A & & & + & C & = & 2/h^2, \end{array}$$

katerega rešitev je $A=C=1/h^2$ in $B=-2/h^2$. Tako smo prišli do formule za izračun drugega odvoda

$$f''(x_0)\approx\frac{f(x_0-h)-2f(x_0)+f(x_0+h)}{h^2}.$$

Njeno napako izračunamo tako, da funkcijo

$$f''(x_0)-\left(\frac{f(x_0-h)-2f(x_0)+f(x_0+h)}{h^2}\right)$$

razvijemo po Taylorjevi formuli, od koder dobimo

$$f''(x_0)=\frac{f(x_0-h)-2f(x_0)+f(x_0+h)}{h^2}-\frac{h^2}{12}f^{(4)}(x_0)+O(h^5). \tag{6.36}$$

Vpliv nenatančnih funkcijskih vrednosti. Napake v funkcijskih vrednostih lahko zelo zmanjšajo uporabnost formul za numerično odvajanje.

Oglejmo si njihov vpliv pri direktni formuli (6.28). Če so funkcijske vrednosti poznane natančno, je napaka direktne formule

$$f'(x_0)-\frac{f(x_0+h)-f(x_0)}{h}=-\frac{h}{2}f''(\xi),$$

torej je napaka poljubno majhna, ko $h\to 0$.

Navadno pa funkcije f ne poznamo povsem natančno, ampak le njen približek

$$\hat{f}(x)=f(x)+e(x),$$

kjer smo s funkcijo e označili napako pri računanju vrednosti funkcije f. Predpostavimo tudi, da je ta napaka omejena

$$|e(x)| \leq \varepsilon$$

na intervalu, ki nas zanima. Dejansko tako izračunamo

$$\hat{f}'(x_0) \approx \frac{\hat{f}(x_0+h) - \hat{f}(x_0)}{h},$$

pri čemer velja

$$\frac{\hat{f}(x_0+h) - \hat{f}(x_0)}{h} = \frac{f(x_0+h) - f(x_0)}{h} + \frac{e(x_0+h) - e(x_0)}{h},$$

od koder dobimo oceno

$$\left| \frac{\hat{f}(x_0+h) - \hat{f}(x_0)}{h} - \frac{f(x_0+h) - f(x_0)}{h} \right| \leq \frac{2\varepsilon}{h}.$$

Če to vstavimo v enačbo (6.29), dobimo za dejansko napako pri izračunu vrednosti odvoda oceno

$$\left| \frac{\hat{f}(x_0+h) - \hat{f}(x_0)}{h} - f'(x_0) \right| \leq \frac{2\varepsilon}{h} + \frac{h}{2}|f''(\xi)|.$$

Ta ocena pomeni, da vrednosti odvoda funkcije ne moremo izračunati s poljubno natančnostjo. Za velike h predstavlja glavni del napake člen $\frac{h}{2}f''(\xi)$, pri majhnih h pa člen $\frac{2\varepsilon}{h}$. Celotna napaka je najmanjša, kadar sta prispevka obeh členov enaka, to je, kadar je $|f''(\xi)|h/2 = 2\varepsilon/h$, oziroma za

$$h = 2\sqrt{\frac{\varepsilon}{|f''(\xi)|}}, \tag{6.37}$$

kar pomeni, da ocena za napako ne more biti manjša kot $2\sqrt{\varepsilon|f''(\xi)|}$, ne glede na to, s kakšnim h računamo.

Primer 6.10. Izračunajmo vrednost odvoda eksponentne funkcije e^x v točki $x = 0$ z direktno formulo pri različnih korakih h med 1 in 10^{-13} in tabelirajmo njihove napake (tabela 6.8).
Iz rezultatov vidimo, da je najmanjša napaka pri koraku $h = 10^{-8}$, kar se ujema s teoretično optimalnim korakom, izračunanim iz enačbe (6.37), ki je pri $\varepsilon = 2.2 \cdot 10^{-16}$ enak $h = 2\sqrt{2.2 \cdot 10^{-16}/e} \approx 10^{-8}$.

h	napaka
10^{-0}	0.71828182845905
10^{-1}	0.05170918075648
10^{-2}	0.00501670841679
10^{-3}	0.00050016670838
10^{-4}	0.00005000166714
10^{-5}	0.00000500000696
10^{-6}	0.00000049996218
10^{-7}	0.00000004943368
10^{-8}	−0.00000000607747
10^{-9}	0.00000008274037
10^{-10}	0.00000008274037
10^{-11}	0.00000008274037
10^{-12}	0.00008890058234
10^{-13}	−0.00079927783736

Tabela 6.8: Napake pri računanju odvoda eksponentne funkcije e^x pri $x = 0$

6.9 Povzetek

Numerično računanje določenih integralov je praviloma enostavnejše od računanja odvodov. Spoznali smo dve metodi, ki sta uporabni tako za konstrukcijo integracijskih formul, kot tudi formul za numerično odvajanje. To sta metoda interpolacijskega polinoma, kjer funkcijo nadomestimo z interpolacijskim polinomom skozi dve ali več sosednjih točk, in metodo nedoločenih koeficientov, kjer zapišemo željeno formulo s še neznanimi koeficienti, ki jih potem določimo tako, da je dobljena formula čimbolj natančna.

Med integracijskimi formulami smo spoznali dve najpopularnejši, to sta trapezna in Simpsonova formula. Bolj natančne so Gaussove kvadraturne formule, ki jih pri različnih utežnih funkcijah $w(x)$ lahko s pridom uporabimo tudi za računanj posplošenih integralov.

Naučili smo se ugotavljati napako približkov za vrednost določenega integrala in določiti korak formule tako, da je napaka končnega rezultata znotraj vnaprej predpisane natančnosti. S pomočjo Rombergove metode smo se naučili izboljšati natančnost približkov, izračunanih z trapezno formulo.

Med formulami za numerično odvajanje smo spoznali direktno in obratno

formulo, ter natančnejšo sredinsko formulo za izračun prvega odvoda, pa tudi formulo za izračun drugega odvoda. Prav tako smo spoznali, da napaka pri računanju odvoda funkcije ne more biti poljubno majhna.

6.10 Problemi

1. Enačbo (6.3) integriraj na intervalu $[a, a+h]$, da dobiš enostavno trapezno formulo (6.4).

2. Prepričaj se, da so približki, ki jih dobimo v drugem stolpcu Rombergove tabele (6.23) isti, kot bi jih dobili pri uporabi Simpsonovega pravila.

3. (a) Izračunaj uteži integracijske formule

$$\int_a^{a+4h} f(x)\,dx \approx h[Af(a+h) + Bf(a+2h) + Cf(a+3h)]$$

   (b) Kolikšna je napaka te integracijske formule?

   (c) Oceni velikost napake te integracijske formule za $a = 0$ in $f(x) = \cos(x)$.

   (d) Primerjaj to integracijsko formulo s trapezno in s Simpsonovo formulo.

4. (a) Izračunaj uteži integracijske formule

$$\int_a^{a+3h} f(x)\,dx \approx h[Af(a+h) + Bf(a+2h) + Cf(a+3h)]$$

   (b) Kolikšna je napaka te integracijske formule?

   (c) Izračunaj približno vrednost integrala

$$\int_0^{0.6} \sqrt{x}x^3\,dx.$$

   Kolikšna je napaka?

   (d) Primerjaj to integracijsko formulo s trapezno in s Simpsonovo formulo.

5. Vrednost integrala
$$I = \int_a^b f(x)\,dx$$
lahko izračunamo z *enostavno sredinsko formulo*
$$\int_c^{c+h} f(x)\,dx \approx hf(c + \frac{h}{2}).$$
   (a) Zapiši *sestavljeno sredinsko pravilo* za izračun vrednosti integrala I pri dolžini koraka $h = \frac{b-a}{4}$
   (b) Kolikšna je napaka enostavne sredinske formule?
   (c) Izračunaj vrednost integrala $\int_0^1 \frac{dx}{1+x^2}$ s sestavljenim sredinskim pravilom pri dolžini koraka $h = 0.25$.
   (d) Za primerjavo izračunaj vrednost istega integrala še s trapeznim pravilom pri isti dolžini koraka in primerjaj obe napaki.

6. Izpelji sredinsko formulo (6.32) za izračun vrednosti odvoda v točki tako, da odvajaš interpolacijski polinom skozi točke x_0-h, x_0 in x_0+h.

7. Izpelji formulo za izračun vrednosti drugega odvoda (6.36) tako, da odvajaš interpolacijski polinom skozi točke $x_0 - h$, x_0 in $x_0 + h$.

Poglavje 7

Navadne diferencialne enačbe

7.1 Uvod

Splošna rešitev navadne diferencialne enačbe n-tega reda

$$F(x, y, y', y'', \ldots, y^{(n)}) = 0$$

je n-parametrična družina funkcij. Kadar želimo iz splošne rešitve izločiti eno izmed partikularnih rešitev, potrebujemo dodatno informacijo, ki je navadno podana kot vrednost te izbrane partikularne rešitve in/ali njenih odvodov pri eni ali več vrednostih neodvisne spremenljivke x. Pri enačbi n-tega reda je navadno dovolj n dodatnih pogojev. Če so ti pogoji dani kot vrednosti izbrane rešitve in njenih odvodov v eni sami točki, govorimo o *začetnem problemu*, Če so te vrednosti dane v dveh ali več različnih točkah, imamo *robni problem*.

V tem poglavju se bomo ukvarjali predvsem z reševanjem začetnih problemov pri diferencialnih enačbah prvega reda

$$y' = f(x, y) \quad \text{in} \quad y(x_0) = y_0. \tag{7.1}$$

Metode, ki jih bomo uporabljali za reševanje problema (7.1), bomo kasneje posplošili tudi za reševanje začetnih problemov pri sistemih navadnih diferencialnih enačb in enačb višjega reda, pa tudi za reševanje enostavnih robnih problemov. Ker enačbo (7.1) lahko analitično rešimo le v posebno enostavnih primerih, se moramo navadno zadovoljiti le z numerično rešitvijo. Na intervalu $[x_0, b]$, na katerem želimo izračunati rešitev, si izberemo zaporedje točk

$$x_1 < x_2 < \cdots < x_n < \cdots,$$

razdalje med njimi pa označimo s $h_i = x_{i+1} - x_i$. Če so vse točke x_i med seboj enako oddaljene, je $h_i = h$ konstanta. Rešitev danega začetnega problema (7.1) dobimo kot zaporedje približkov y_i za vrednosti točne rešitve $y(x_i)$ v izbranih točkah. Za odvod funkcije v točki približne rešitve bomo uporabljali skrajšano oznako $f_i = f(x_i, y_i)$.

Pri numeričnem reševanju začetnih problemov nas predvsem zanima razlika med numerično in točno rešitvijo

$$g_n = y_n - y(x_n),$$

ki ji pravimo *globalna napaka*. Na njeno velikost vplivajo različni faktorji, kot so: izbrana numerična metoda, velikost koraka h_i, obnašanje točne rešitve $y(x)$ začetnega problema, računalnik, s katerim računamo, itd. Tu si bomo ogledali le, kako na velikost globalne napake vplivata izbor metode in velikost koraka, ostale vplive bomo zanemarili. Globalna napaka je posledica napake, ki jo naredimo na vsakem posameznem koraku. *Lokalna napaka* metode v točki x_{n+1} je razlika med numeričnim približkom y_{n+1} in tisto rešitvijo diferencialne enačbe, ki poteka skozi prejšnjo točko numerične rešitve (x_n, y_n), to je rešitvijo začetnega problema

$$z' = f(x, z), \qquad z(x_n) = y_n, \tag{7.2}$$

izračunano v točki x_{n+1}:

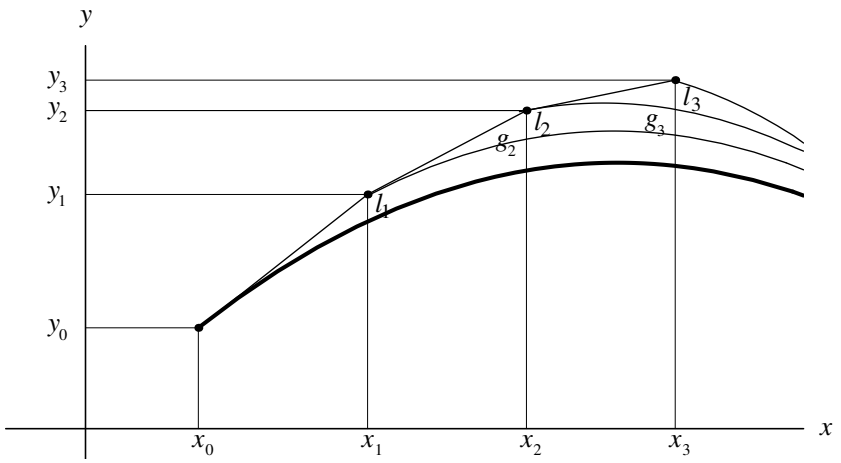
$$l_{n+1} = y_{n+1} - z(x_{n+1}).$$

Iz slike 7.1 vidimo, da je globalna napaka numerične rešitve v neki točki posledica globalne napake v prejšnji točki in lokalne napake v zadnjem koraku.

Definicija 7.1. Red metode *za reševanje diferencialnih enačb je celo število p, za katerega je*

$$l_n = Ch_n^{p+1} + O(h_n^{p+2}).$$

Konstanta C je poznana kot konstanta napake.



Slika 7.1: Globalna napaka numerične rešitve je posledica globalne napake v prejšni točki in lokalne napake v zadnjem koraku

7.2 Eulerjeva metoda

Najenostavnejša metoda za reševanje začetnih problemov (7.1) je *Eulerjeva metoda*[1]. Izpeljemo jo lahko iz Taylorjeve formule za točno rešitev začetnega problema

$$y(x_n + h_n) = y(x_n) + h_n y'(x_n) + \frac{h_n^2}{2} y''(\xi_n); \quad \xi \in (x_n, x_n + h_n). \tag{7.3}$$

Če zanemarimo člen s h_n^2, dobimo pravilo, po katerem iz znane vrednosti y_n izračunamo naslednjo vrednost v zaporedju približkov k rešitvi

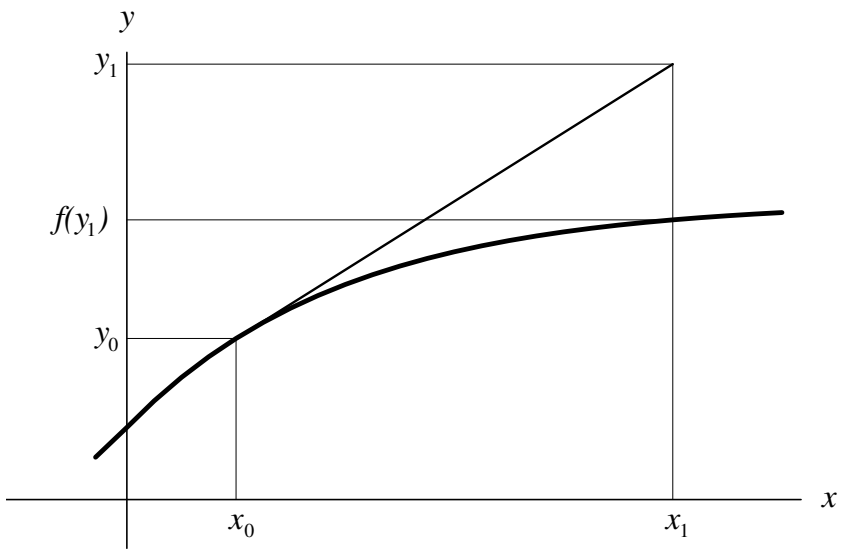
$$y_{n+1} = y_n + h_n f_n. \tag{7.4}$$

Geometrijsko Eulerjeva formula (7.4) pomeni, da rešitev enačbe med točkama x_i in x_{i+1} nadomestimo z odsekom premice iz točke x_i v smeri, ki je določena z odvodom f_i do točke x_{i+1} (slika 7.2).

Iz enačbe (7.3) vidimo, da je lokalna napaka Eulerjeve metode enaka $h^2 y''(\xi)/2$, torej je Eulerjeva metoda prvega reda.

Zapišimo postopek za reševanje začetnega problema z Eulerjevo metodo v obliki algoritma:

[1]Leonhard Euler (1707 Basel – 1783 St Petersburg) švicarski matematik, eden izmed najpomembnejših matematikov. Večino življenja je preživel v Berlinu in St. Petersburgu. Poleg matematike se je ukvarjal tudi z astronomijo, in fiziko.



Slika 7.2: Rešitev diferencialne enačbe z Eulerjevo metodo

Algoritem 7.1. Eulerjeva metoda Naj bo $y' = f(x, y)$ diferencialna enačba, $y(x_0) = y_0$ začetni pogoj, h dolžina koraka in N naravno število. Naslednji algoritem izračuna zaporedje približkov y_n k vrednostim točne rešitve $y(x_n)$ diferencialne enačbe v izbranih točkah $x_n = x_0 + nh$; $n = 1, \ldots, N$ s pomočjo Eulerjeve metode.

$y = y_0$
$x = x_0$
for $n = 1 : N$
    $y = y + h * f(x, y)$
    $x = x + h$
end

Delovanje Eulerjeve metode si oglejmo na preprostem primeru.

n	x_n	y_n	$y_n - y(x_n)$
0	0.0	2.0000	−0.0000
1	0.1	1.9000	−0.0048
2	0.2	1.8100	−0.0087
3	0.3	1.7290	−0.0118
4	0.4	1.6561	−0.0142
5	0.5	1.5905	−0.0160
6	0.6	1.5314	−0.0174
7	0.7	1.4782	−0.0183
8	0.8	1.4305	−0.0189
9	0.9	1.3874	−0.0191
10	1.0	1.3487	−0.0192

Tabela 7.1: Rešitev začetnega problema z Eulerjevo metodo

Primer 7.1. Poiščimo približno rešitev začetnega problema

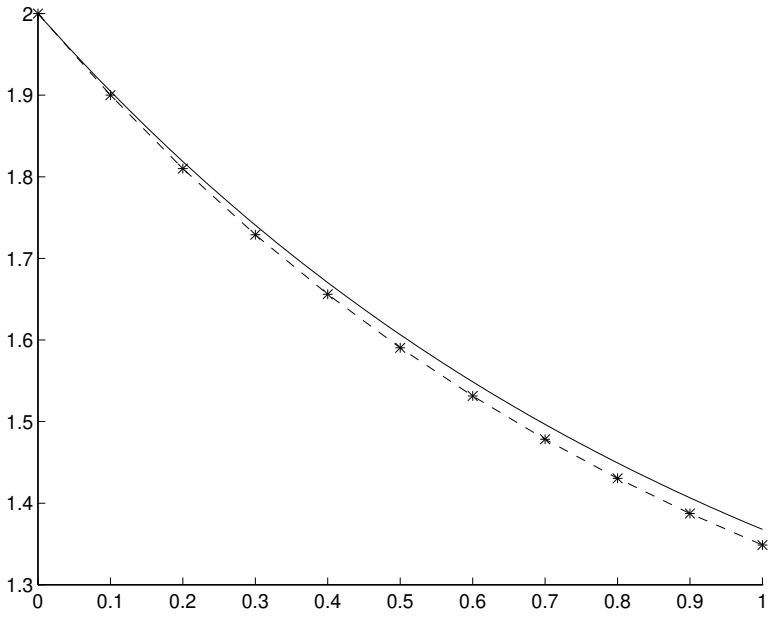
$$y' = -y + 1 \quad \text{in} \quad y(0) = 2 \tag{7.5}$$

na intervalu $[0, 1]$ z Eulerjevo metodo pri koraku $h = 0.1$.
Iz rešitve v tabeli 7.1 in na sliki 7.3 vidimo, da numerična rešitev sicer spremlja točno rešitev, vendar je globalna napaka dokaj velika.

7.3 Linearne veččlenske metode

Eulerjeva metoda je preprosta, vendar je za resno uporabo premalo natančna. Natančnejše metode lahko dobimo z različnimi posplošitvami Eulerjeve metode. V tem razdelku si bomo ogledali metode, pri katerih za izračun vrednosti v naslednji točki (x_{n+1}) poleg vrednosti rešitve y_n in odvoda f_n v točki x_n uporabimo več že prej izračunanih vrednosti in odvodov v točkah $x_{n-1}, x_{n-2}, \ldots$ Take metode imenujemo *linearne veččlenske metode*.

Adams-Bashforthove metode Vzemimo, da smo pri konstantni dolžini koraka h že izračunali približne rešitve na množici ekvidistantnih točk $x_i = x_0 + ih$; $i = 0, \ldots, n$. Diferencialno enačbo $y' = f(x, y)$ integriramo med x_n



Slika 7.3: Točna in numerična rešitev začetnega problema (7.5) z Eulerjevo metodo pri dolžini koraka $h = 0.1$.

in x_{n+1} in dobimo

$$\int_{x_n}^{x_{n+1}} y'\,dx = \int_{x_n}^{x_{n+1}} f(x, y(x))\,dx,$$

kar lahko zapišemo kot

$$y_{n+1} = y_n + \int_{x_n}^{x_{n+1}} f(x, y(x))\,dx. \tag{7.6}$$

Če integral na desni izračunamo tako, da $f(x, y(x))$ zamenjamo z interpolacijskim polinomom skozi točke $x_n, x_{n-1}, \ldots, x_{n-k+1}$, dobimo k-člensko *Adams*[2]-*Bashforthovo*[3] metodo (AB metodo). (Glej problem 1) Navadno jo zapišemo

[2]John Couch Adams (1819 Laneast (Anglija) – 1892 Cambridge), Angleški astronom, velika je njegova vloga pri odkrivanju planeta Neptuna. Metodo za numerično integracijo diferencialnih enačb je objavil skupaj z F. Bashforthom leta 1883.

[3]Francis Bashforth (1819 Doncaster – 1912 Woodhall Spa (Anglija)), angleški gradbeni inženir in balistik. Na univerzi v Cambridgu je bil sošolec Johna Adamsa. V mladih letih je deloval v gradbeništvu, pozneje je bil profesor uporabne matematike in balistike na vojaški akademiji v Woolwichu.

v obliki

$$y_{n+1} = y_n + h \sum_{i=1}^{k} \beta_i f_{n-i+1}. \tag{7.7}$$

Napaka k-členske Adams-Bashforthove metode je $C_{k+1} h^{k+1} y^{(k+1)}(\xi_n)$. Konstanti C_{k+1} pravimo *konstanta napake*. Tako ima k-členska Adams-Bashforthova metoda red k. Koeficienti prvih šestih AB metod z ustreznimi konstantami napake so zbrani v tabeli 7.2. Prvo ($k = 1$) izmed njih, $y_{n+1} = y_n + h f_n$, že poznamo, to je Eulerjeva metoda (7.4).

k	i	1	2	3	4	5	6	C_{k+1}
1	β_i	1						$\frac{1}{2}$
2	$2\beta_i$	3	-1					$\frac{5}{12}$
3	$12\beta_i$	23	-16	5				$\frac{3}{8}$
4	$24\beta_i$	55	-59	37	-9			$\frac{251}{720}$
5	$720\beta_i$	1901	-2774	2616	-1274	251		$\frac{95}{288}$
6	$1440\beta_i$	4277	-7923	9982	-7298	2877	-475	$\frac{19087}{60480}$

Tabela 7.2: Koeficienti Adams–Bashforthovih metod.

Za reševanje začetnega problema s k-člensko AB metodo ($k \geq 2$) potrebujemo poleg začetne vrednosti y_0 še $k-1$ vrednosti $y_1, \ldots, y_{k-1}$, ki jih moramo izračunati s kakšno drugo metodo (na primer z metodo Runge-Kutta iz naslednjega razdelka), ki ima isti red natančnosti kot AB metoda.

Zapišimo algoritem za reševanje začetnega problema z AB metodo:

Algoritem 7.2. [AB metoda] Naj bo $y' = f(x, y)$ diferencialna enačba, $y(x_0) = y_0$ začetni pogoj, h korak in N naravno število. Naslednji algoritem izračuna zaporedne približke y_n k vrednostim rešitve $y(x_n)$ diferencialne enačbe v točkah $x_n = x_0 + nh;\ n = 1, \ldots, N$ s pomočjo AB metode 4. reda.

$y = y_0$
$x = x_0$
$d(1) = f(x, y)$
for $n = 2 : 4$
    $x = x + h$
    Izračunaj dodatno vrednost $y \approx y(x)$
    $d(n) = f(x, y)$
end
for $n = 4 : N$
    $x = x + h$
    $y = y + h * (55 * d(4) - 59 * d(3) + 37 * d(2) - 9 * d(1))/24$
    $d(5) = f(x, y)$
    for $j = 1 : 4$
        $d(j) = d(j + 1)$
    end
end

Izračunajmo rešitev diferencialne enačbe (7.5) še z Adams-Bashforthovo metodo četrtega reda.

Primer 7.2. Poiščimo približno rešitev začetnega problema (7.5) na intervalu $[0, 1]$ še z Adams-Bashforthovo metodo 4. reda pri enakem koraku $h = 0.1$ kot v primeru 7.1. Manjkajoče začetne vrednosti določimo kar iz točne rešitve $y = e^{-x} + 1$ v točkah $-0.3, -0.2$ in -0.1.
Iz rezultatov v tabeli 7.3 lahko ugotovimo, da je napaka približkov, izračunanih z AB metodo 4. reda, dosti manjša od napake približkov, izračunanih z Eulerjevo metodo.

Adams-Moultonove metode Če integral na desni strani enačbe (7.6) izračunamo tako, da interpolacijskim točkam dodamo tudi x_{n+1}, dobimo k-

n	x_n	y_n	$y_n - y(x_n)$
0	0.0	2.0000	0
1	0.1	1.9048	$3.9 \cdot 10^{-6}$
2	0.2	1.8187	$6.5 \cdot 10^{-6}$
3	0.3	1.7408	$9.1 \cdot 10^{-6}$
4	0.4	1.6703	$1.1 \cdot 10^{-5}$
5	0.5	1.6065	$1.2 \cdot 10^{-5}$
6	0.6	1.5488	$1.3 \cdot 10^{-5}$
7	0.7	1.4966	$1.4 \cdot 10^{-5}$
8	0.8	1.4493	$1.5 \cdot 10^{-5}$
9	0.9	1.4066	$1.5 \cdot 10^{-5}$
10	1.0	1.3679	$1.5 \cdot 10^{-5}$

Tabela 7.3: Rešitev začetnega problema z Adams-Bashforthovo metodo 4. reda pri dolžini koraka $h = 0.1$. Dodatne 3 vrednosti odvoda so bile izračunane iz točne rešitve v točkah $-0.3, -0.2$ in -0.1.

člensko *Adams-Moultonovo*[4] metodo (AM metodo) (Glej problem 2). Zapišemo jo kot

$$y_{n+1} = y_n + h \sum_{i=0}^{k} \beta_i^* f_{n-i+1}. \tag{7.8}$$

Napaka k-členske Adams-Moultonove metode je $C_{k+2}^* h^{k+2} y^{(k+1)}(\xi_n)$, kjer konstanti C_{k+2}^* zopet pravimo *konstanta napake*. Tako ima k-členska Adams-Moultonova metoda red $k+1$. Koeficienti prvih šestih AM metod z ustreznimi konstantami napake so zbrani v tabeli 7.2. Metoda, ki jo dobimo za $k = 0$

$$y_{n+1} = y_n + h f_{n+1} \tag{7.9}$$

je poznana kot *implicitna Eulerjeva metoda*, metoda s $k = 1$

$$y_{n+1} = y_n + \frac{h}{2}(f_n + f_{n+1})$$

pa je poznana pod imenom *trapezna metoda*.

[4]Forest Ray Moulton (1872 LeRoy, Michigan – 1952 Chicago), ameriški astronom. Adamsovo metodo za reševanje diferencialnih enačb je izpopolnil, da bi natančneje izračunaval balistične trajektorije, objavil jo je leta 1926.

k	i	0	1	2	3	4	5	C^*_{k+1}
0	β^*_i	1						$\frac{-1}{2}$
1	$2\beta^*_i$	1	1					$\frac{-1}{12}$
2	$12\beta^*_i$	5	8	-1				$\frac{-1}{24}$
3	$24\beta^*_i$	9	19	-5	1			$\frac{-19}{720}$
4	$720\beta^*_i$	251	646	-264	106	-19		$\frac{-3}{160}$
5	$1440\beta_i$	475	1427	-798	482	-173	27	$\frac{-863}{60480}$

Tabela 7.4: Koeficienti Adams–Moultonovih metod.

Za razliko od AB metod, nastopa pri AM metodah neznana vrednost y_{n+1} na obeh straneh enačbe (7.8), tako da moramo na vsakem koraku rešiti nelinearno enačbo

$$y_{n+1} = y_n + h\beta^*_0 f(x_{n+1}, y_{n+1}) + h\sum_{i=1}^{k} \beta^*_i f_{n-i+1}. \tag{7.10}$$

Metodam, pri katerih moramo vrednost y_{n+1} določiti kot rešitev enačbe, pravimo *implicitne* metode, za razliko od *eksplicitnih* metod (kot so AB metode), pri katerih izračunamo y_{n+1} direktno.

V praksi rešujemo enačbo (7.10) običajno z navadno iteracijo. Če imamo dober začetni približek, ki ga navadno dobimo z ustrezno eksplicitno AB metodo istega reda, ki ji v tem primeru pravimo *prediktor*, in dovolj majhen korak h, je večinoma dovolj že ena iteracija (implicitni AM formuli v tem primeru pravimo *korektor*). Taka kombinacija AB in AM metode je znana pod imenom *metoda prediktor-korektor* . Zapišimo še ustrezen algoritem:

Algoritem 7.3. ABM prediktor-korektor Naj bo $y' = f(x, y)$ diferencialna enačba, $y(x_0) = y_0$ začetni pogoj, h korak in N naravno število. Naslednji algoritem izračuna zaporedje približkov y_n k vrednostim točne rešitve $y(x_n)$ diferencialne enačbe v izbranih točkah $x_n = x_0 + nh$; $n = 1, \ldots, N$ s pomočjo ABM metode prediktor-korektor 4. reda.

$x = x_0, y = y_0$
$d(1) = f(x, y)$
for $n = 2 : 4$
    $x = x + h$
    Izračunaj dodatno vrednosti $y \approx y(x)$
    $d(n) = f(x, y)$
end
for $n = 4 : N$
    $x = x + h$
    $yp = y + h * (55 * d(4) - 59 * d(3) + 37 * d(2) - 9 * d(1))/24$
    $dp = f(x, yp)$
    $y = y + h * (9 * dp + 19 * d(4) - 5 * d(3) + d(2))/24$
    $d(5) = f(x, y)$
    for $j = 1 : 4$
        $d(j) = d(j + 1)$
    end
end

Rešitev enačbe (7.5) izračunajmo še z Adams-Bashforth-Moultonovo metodo prediktor-korektor 4. reda.

Primer 7.3. Poiščimo približno rešitev začetnega problema (7.5) na intervalu $[0, 1]$ še z ABM metodo prediktor-korektor 4. reda pri enakem koraku $h = 0.1$ kot v obeh prejšnjih primerih 7.1 in 7.2. Manjkajoče začetne vrednosti določimo kar iz točne rešitve $y = e^{-x} + 1$.
Iz rezultatov v tabeli 7.5 vidimo, da je napaka približkov, izračunanih z metodo prediktor-korektor za velikostni razred manjša od napak približkov, ki smo jih izračunali samo s prediktorjem (primer 7.2).

n	x_n	y_n	$y_n - y(x_n)$
0	0.0	2.0000	0
1	0.1	1.9048	$-4.2 \cdot 10^{-7}$
2	0.2	1.8187	$-7.5 \cdot 10^{-7}$
3	0.3	1.7408	$-1.0 \cdot 10^{-6}$
4	0.4	1.6703	$-1.2 \cdot 10^{-6}$
5	0.5	1.6065	$-1.4 \cdot 10^{-6}$
6	0.6	1.5488	$-1.5 \cdot 10^{-6}$
7	0.7	1.4966	$-1.6 \cdot 10^{-6}$
8	0.8	1.4493	$-1.6 \cdot 10^{-6}$
9	0.9	1.4066	$-1.7 \cdot 10^{-6}$
10	1.0	1.3679	$-1.7 \cdot 10^{-6}$

Tabela 7.5: Rešitev začetnega problema z ABM metodo prediktor-korektor 4. reda in dolžino koraka $h = 0.1$. Dodatne 3 vrednosti odvoda so bile izračunane iz točne rešitve v točkah $-0.3, -0.2$ in -0.1.

7.4 Metode Runge-Kutta

Pri Eulerjevi metodi (7.4) vzamemo za vrednost odvoda rešitve na intervalu $[x_n, x_{n+1}]$ kar vrednost odvoda rešitve v začetni točki intervala $f'(x_n)$. Pri metodah Runge[5]-Kutta[6] dosežemo večjo natančnost tako, da vrednosti odvoda izračunamo v več točkah na intervalu x_n, x_{n+1} in pri računanju vrednosti rešitve v točki x_{n+1} upoštevamo njihovo primerno obteženo povprečje. Kot primer izračunajmo koeficiente a, b_1, b_2 in c preproste dvostopenjske metode Runge-Kutta

$$y_{n+1} = y_n + b_1 k_1 + b_2 k_2, \tag{7.11}$$

[5]Carle David Tolmé Runge (1856 Bremen – 1927 Göttingen), nemški matematik in fizik, profesor na univerzah v Hannovru in Göttingenu. Svoj prvi članek o numeričnem reševanju diferencialnih enačb je objavil leta 1895.

[6]Martin Wilhelm Kutta (1867 Pitschen (sedaj Byczyna, Poljska) – 1944 Fürstenfeldbruck), nemški matematik, profesor na univerzi v Stuttgartu. Na osnovi Rungejeve ideje je objavil izboljšano verzijo metode Runge-Kutta leta 1901.

kjer sta hk_1 in hk_2 približka za vrednost odvoda v točkah x_n in $x_n + ch$. Izračunamo ju iz

$$\begin{aligned} k_1 &= hf(x_n, y_n) \\ k_2 &= hf(x_n + ch, y_n + ak_1). \end{aligned}$$

Točno rešitev $y(x_n + h)$ razvijemo v Taylorjevo vrsto po potencah h in dobimo

$$\begin{aligned} y(x_n + h) &= y(x_n) + hy'(x_n) + \frac{h^2}{2}y''(x_n) + \frac{h^3}{6}y'''(x_n) + \cdots \\ &= y_n + hf_n + \frac{h^2}{2}(f_x + ff_y)_n \\ &+ \frac{h^3}{6}(f_{xx} + 2ff_{xy} + f_{yy}f^2 + f_xf_y + f_y^2 f)_n + \cdots , \end{aligned} \tag{7.12}$$

kjer smo z indeksom n označili, da je potrebno vse funkcije izračunati v točki (x_n, y_n).

Da bi dobili Taylorjevo vrsto za numerično rešitev (7.11), razvijemo najprej k_2:

$$k_2 = hf_n + h^2(cf_x + aff_y)_n + h^3\left(\frac{c^2}{2}f_{xx} + acff_{xy} + \frac{a^2}{2}f_{yy}\right)_n + \cdots ,$$

kar vstavimo v (7.11), da dobimo

$$\begin{aligned} y_{n+1} &= y_n + (b_1 + b_2)hf_n + b_2h^2(cf_x + aff_y)_n + \\ &\quad b_2h^3\left(\frac{c^2}{2}f_{xx} + acff_{xy} + \frac{a^2}{2}f^2f_{yy}\right)_n + \cdots \end{aligned} \tag{7.13}$$

S primerjavo vrst (7.12) in (7.13) ugotovimo, da se vrsti ujemata v členih s h in h^2, če so izpolnjene enačbe

$$\begin{aligned} b_1 + b_2 &= 1 \\ ab_2 = cb_2 &= \frac{1}{2}. \end{aligned}$$

Ta sistem ima enoparametrično družino rešitev. Najpreprostejši rešitvi dobimo, če si izberemo $b_2 = 1/2$ ali $b_2 = 1$. V prvem primeru imamo metodo

$$\begin{aligned} k_1 &= hf(x_n, y_n) \\ k_2 &= hf(x_n + h, y_n + k_1) \\ y_{n+1} &= y_n + (k_1 + k_2)/2, \end{aligned} \tag{7.14}$$

v drugem pa

$$\begin{aligned} k_1 &= hf(x_n, y_n) \\ k_2 &= hf(x_n + h/2, y_n + k_1/2) \\ y_{n+1} &= y_n + k_2, \end{aligned} \tag{7.15}$$

kar sta obe metodi Runge-Kutta drugege reda.

Splošno s-stopenjsko eksplicitno metodo Runge-Kutta lahko zapišemo kot

$$y_{n+1} = y_n + \sum_{i=1}^{s} b_i k_i,$$

kjer števila $k_1, \ldots, k_s$ izračunamo zaporedoma kot

$$\begin{aligned} k_1 &= hf(x_n, y_n) \\ k_2 &= hf(x_n + c_2 h, y_n + a_{2,1} k_1) \\ &\vdots \\ k_s &= hf(x_n + c_s h, y_n + \sum_{j=1}^{s-1} a_{s,j} k_j). \end{aligned}$$

Koeficiente $a_{i,j}, b_i$ in c_i lahko pregledno zapišemo v obliki tabele (*Butcherjeva tabela*[7])

$$\begin{array}{c|ccccc} 0 & 0 & & & & \\ c_2 & a_{2,1} & 0 & & & \\ c_3 & a_{3,1} & a_{3,2} & 0 & & \\ \vdots & \vdots & \ddots & & & \\ c_s & a_{s,1} & a_{s,2} & \cdots & a_{s,s-1} & 0 \\ \hline & b_1 & b_2 & \cdots & b_{s-1} & b_s, \end{array}$$

kjer so koeficienti c_i določeni s koeficienti a_{ij} kot

$$c_i = \sum_{j=1}^{i-1} a_{ij}.$$

[7]John Butcher (1933 Auckland (Nova Zelandija)), novozelandski matematik, profesor računalništva in matematike na univerzi v Aucklandu. Najzaslužnejši za razvoj teorije in prakse metod Runge Kutta. Leta 1963 je objavil članek, ki je omogočil teoretično analizo in nadaljni razvoj teh metod.

Pri večjem številu stopenj s postanejo enačbe, ki določajo koeficiente zelo zapletene. Najpopularnejša in najpogosteje uporabljana je metoda Runge-Kutta četrtega reda

$$\begin{aligned} k_1 &= hf_n \\ k_2 &= hf(x_n + h/2, y_n + k_1/2) \\ k_3 &= hf(x_n + h/2, y_n + k_2/2) \\ k_4 &= hf(x_n + h, y_n + k_3) \\ y_{n+1} &= y_n + (k_1 + 2k_2 + 2k_3 + k_4)/6. \end{aligned}$$

Zapišimo jo še kot algoritem:

Algoritem 7.4. [Runge-Kutta 4. reda] Naj bo $y' = f(x, y)$ diferencialna enačba, $y(x_0) = y_0$ začetni pogoj in N naravno število. Naslednji algoritem izračuna vektor približkov y_n k vrednostim rešitve $y(x_n)$ diferencialne enačbe v točkah $x_n = x_0 + nh;\ n = 1, \ldots, N$ s pomočjo metode Runge-Kutta 4. reda.

$x = x_0$
$y = y_0$
for $n = 1 : N$
    $k1 = h * f(x, y)$
    $k2 = h * f(x + h/2, y + k1/2)$
    $k3 = h * f(x + h/2, y + k2/2)$
    $k4 = h * f(x + h, y + k3)$
    $y = y + (k1 + 2 * k2 + 2 * k3 + k4)/6$
    $x = x + h$
end

Primer 7.4. Poiščimo približno rešitev začetnega problema (7.5) na intervalu $[0, 1]$ tudi z metodo Runge-Kutta 4. reda z enako dolžino koraka $h = 0.1$ kot v prejšnjih primerih.
Iz rezultatov v tabeli 7.6 vidimo, da je napaka pri računanju z metodo Runge-Kutta nekoliko manjša od napake, ki smo jo dobili pri metodi prediktor-korektor (tabela 7.5).

Rezultati, dobljeni z metodo Runge-Kutta, so praviloma nekoliko boljši, kot tisti, ki jih dobimo z ABM metodo prediktor-korektor. Cena, ki jo mo-

n	x_n	y_n	$y_n - y(x_n)$
0	0.0	1.0000	0
1	0.1	1.9048	$8.2 \cdot 10^{-8}$
2	0.2	1.8187	$1.5 \cdot 10^{-7}$
3	0.3	1.7408	$2.0 \cdot 10^{-7}$
4	0.4	1.6703	$2.4 \cdot 10^{-7}$
5	0.5	1.6065	$2.7 \cdot 10^{-7}$
6	0.6	1.5488	$3.0 \cdot 10^{-7}$
7	0.7	1.4966	$3.1 \cdot 10^{-7}$
8	0.8	1.4493	$3.3 \cdot 10^{-7}$
9	0.9	1.4066	$3.3 \cdot 10^{-7}$
10	1.0	1.3679	$3.3 \cdot 10^{-7}$

Tabela 7.6: Rešitev začetnega problema z metodo Runge-Kutta 4. reda

ramo za to plačati, pa so štiri računanja odvodov v primerjavi s samo dvema pri metodi prediktor-korektor. Po drugi strani pa je metoda Runge-Kutta enostavnejša za uporabo, saj pri njej ne potrebujemo dodatnih vrednosti odvodov v prejšnjih točkah, enostavno pa je tudi spreminjanje dolžine koraka.

7.5 Kontrola koraka

Pri računanju približka k točni rešitvi začetnega problema nas predvsem zanima velikost globalne napake. Njene velikosti med računanjem ne moremo direktno ocenjevati, lahko pa ocenjujemo velikost lokalnih napak, od katerih je odvisna tudi globalna napaka. Če bomo z izbiro dolžine koraka primerno omejili lokalne napake, bo navadno tudi globalna napaka ostala majhna.

Skoraj pri vseh metodah za reševanje začetnih problemov lahko uporabimo podoben način, ki smo ga spoznali že pri računanju integralov. Vzemimo, da smo z metodo p-tega reda izračunali vrednost $y_{n+1,h}$ z dolžino koraka h. Lokalna napaka (7.2) je torej

$$l_{n+1} = y_{n+1,h} - z(x_{n+1}) \approx Ch^{p+1},$$

kjer je $z(x_{n+1})$ rešitev začetnega problema (7.2). Izračunajmo še vrednost

$y_{n+1,h/2}$ tako, da iz točke x_n naredimo dva koraka dolžine $h/2$, torej

$$y_{n+1,h/2} - z(x_{n+1}) \approx 2Ch^{p+1}2^{-p-1}.$$

Zadnji dve enačbi odštejemo:

$$y_{n+1,h} - y_{n+1,h/2} \approx Ch^{p+1}(1 - 2^{-p}),$$

od koder dobimo približek za lokalno napako vrednosti $y_{n+1,h}$

$$l_{n+1} \approx \frac{y_{n+1,h} - y_{n+1,h/2}}{1 - 2^{-p}},$$

lokalna napaka točnejše vrednosti $y_{n+1,h/2}$ pa je še za faktor 2^p manjša.

Na podlagi vnaprej predpisane zgornje meje ε za lokalno napako, lahko sedaj sodimo o primernosti dolžine koraka h. Če je ugotovljena lokalna napaka manjša od produkta εh, potem vrednost $y_{n+1,h}$ (še bolje pa $y_{n+1,h/2}$) sprejmemo, sicer začnemo računati ponovno iz prejšnje vrednosti y_n, vendar z manjšo dolžino koraka. Če pa je ugotovljena lokalna napaka dosti manjša od εh, lahko v naslednjem koraku h povečamo.

Opisani način kontrole lokalne napake in dolžine koraka je uporaben pri vseh metodah (linearnih veččlenskih in Runge-Kutta), zahteva pa nekaj dodatnega računanja, saj moramo vrednost rešitve v vsaki točki računati dvakrat. Oba tipa metod pa omogočata tudi učinkovitejše računanje približne vrednosti lokalne napake, kar si bomo sedaj ogledali.

Metode prediktor-korektor Pri metodah prediktor-korektor lahko izračunamo približno vrednost napake iz razlike med vrednostjo rešitve, izračunano z eksplicitno metodo (prediktorjem) in z implicitno metodo (korektorjem) popravljeno vrednostjo. Vzemimo, da sta obe formuli, prediktor in korektor, istega reda p. Približni vrednosti za lokalni napaki prediktorja in korektorja

$$\begin{aligned} y^p_{n+1} - z(x_{n+1}) &\approx C_p h^{p+1} \\ l_{n+1} = y_{n+1} - z(x_{n+1}) &\approx C_k h^{p+1}, \end{aligned}$$

odštejemo in dobimo

$$y_{n+1} - y^p_{n+1} \approx (C_k - C_p)h^{p+1},$$

od koder izračunamo približno vrednost lokalne napake

$$l_{n+1} \approx \frac{C_k}{C_k - C_p}(y_{n+1} - y^p_{n+1}).$$

Kadar je tako izračunana lokalna napaka manjša od εh, sprejmemo vrednost y_{n+1} kot približek za rešitev v točki x_{n+1}. Kadar pa je ugotovljena napaka večja, moramo zmanjšati dolžino koraka h in začeti znova od vrednosti y_n, kar pa je pri veččlenskih metodah povezano z dodatnimi težavami, saj moramo posebej izračunati vrednosti odvodov rešitve v zaporednih točkah, ki so med seboj oddaljene za h. Če korak razpolovimo, lahko nekatere izmed znanih vrednosti uporabimo tudi pri novi dolžini koraka, ostale pa lahko izračunamo s pomočjo interpolacije. Kadar je ugotovljena napaka znatno manjša (za več kot faktor 2^p) od dovoljene, pa lahko dolžino koraka povečamo, vendar se tudi v tem primeru znajdemo pred podobnimi težavami kot pri zmanjšanju dolžine koraka. V tem primeru je najbolje dolžino koraka podvojiti, saj tako lahko uporabimo vrednosti odvodov, ki smo jih že izračunali, seveda pa si moramo v tem primeru zapomniti več kot samo zadnjih k vrednosti odvoda.

Metode Runge-Kutta Pri metodah Runge-Kutta se je v zadnjih letih za sprotno ocenjevanje lokalne napake uveljavila tehnika *vgnezdenih parov metod*. To sta dve metodi Runge-Kutta, ki imata isto matriko koeficientov a_{ij}, $i = 1, \ldots, j-1; j = 1, \ldots, s$ in različna vektorja uteži b_i in b_i^*; $i = 1, \ldots, s$. Koeficienti in uteži morajo biti izbrani tako, da sta metodi različnih redov, naprimer metoda z utežmi b_i je reda p, metoda z utežmi b_i^* pa reda $p+1$.

Najpreprostejši primer takega vgnezdenega para sta metodi prvega in drugega reda, ki ju lahko zapišemo kot Butcherjevo tabelo

$$\begin{array}{c|cc} 0 & 0 & \\ 1 & 1 & 0 \\ \hline & 1 & 0 \\ & [\frac{1}{2} & \frac{1}{2}], \end{array}$$

kjer smo v oglatem oklepaju zapisali uteži metode višjega reda. To pomeni, da je metoda nižjega reda (v tem primeru prvega reda) Eulerjeva metoda, metoda višjega reda (v tem primeru drugega) pa metoda (7.14). Ker je metoda višjega reda točnejša od metode nižjega reda, izračunamo približno vrednost lokalne napake kot razliko dveh približkov

$$\begin{aligned} y_{n+1} &= y_n + k_1 \\ y_{n+1}^* &= y_n + (k_1 + k_2)/2, \end{aligned}$$

torej

$$l_{n+1} \approx y_{n+1}^* - y_{n+1} = (-k_1 + k_2)/2,$$

Na ta način smo sicer izračunali približno vrednost napake metode nižjega reda, vendar običajno za rešitev y_{n+1} vzamemo točnejši približek y^*_{n+1}.

Pri praktičnem računanju se zelo obnese metoda, poznana pod imenom DOPRI5, ki sta jo konstruirala Dormand in Prince (1980). Njen red je 5(4), njeni koeficienti pa so v tabeli 7.7.

0							
$\frac{1}{5}$	$\frac{1}{5}$						
$\frac{3}{10}$	$\frac{3}{40}$	$\frac{9}{40}$					
$\frac{4}{5}$	$\frac{44}{45}$	$-\frac{56}{15}$	$\frac{32}{9}$				
$\frac{8}{9}$	$\frac{19372}{6561}$	$-\frac{25360}{2187}$	$\frac{64448}{6561}$	$-\frac{212}{729}$			
1	$\frac{9017}{3168}$	$-\frac{355}{33}$	$\frac{46732}{5247}$	$\frac{49}{176}$	$-\frac{5103}{18656}$		
1	$\frac{35}{384}$	0	$\frac{500}{1113}$	$\frac{125}{192}$	$-\frac{2187}{6784}$	$\frac{11}{84}$	
	$\frac{35}{384}$	0	$\frac{500}{1113}$	$\frac{125}{192}$	$-\frac{2187}{6784}$	$\frac{11}{84}$	0
	$\left[\frac{5179}{57600}\right.$	0	$\frac{7571}{16695}$	$\frac{393}{640}$	$-\frac{92097}{339200}$	$\frac{187}{2100}$	$\left.\frac{1}{40}\right]$

Tabela 7.7: Koeficienti metode DOPRI5 (Dormand in Prince (1980))

Pri metodah Runge-Kutta ni težav pri spreminjanju dolžine koraka, zato jo lahko sproti spreminjamo glede na to, kako se obnaša lokalna napaka. Vzemimo, da računamo z metodo reda p, torej je lokalna napaka $l_n \approx Ch_n^{p+1}$. Dolžino naslednjega koraka želimo izbrati tako, da bo lokalna napaka $Ch_{n+1}^{p+1} \approx \varepsilon h_{n+1}$. Od tod lahko eliminiramo konstanto napake C in dobimo

$$\frac{h_{n+1}^{p+1}}{h_n^{p+1}} \approx \frac{\varepsilon h_{n+1}}{|l_n|},$$

zato moramo dolžino naslednjega koraka izbrati kot

$$h_{n+1} = h_n \sqrt[p]{\frac{\varepsilon h_n}{|l_n|}},$$

raje pa, da bo napaka zanesljivo manjša od maksimalno dovoljene, nekoliko manj. Zapišimo celoten algoritem.

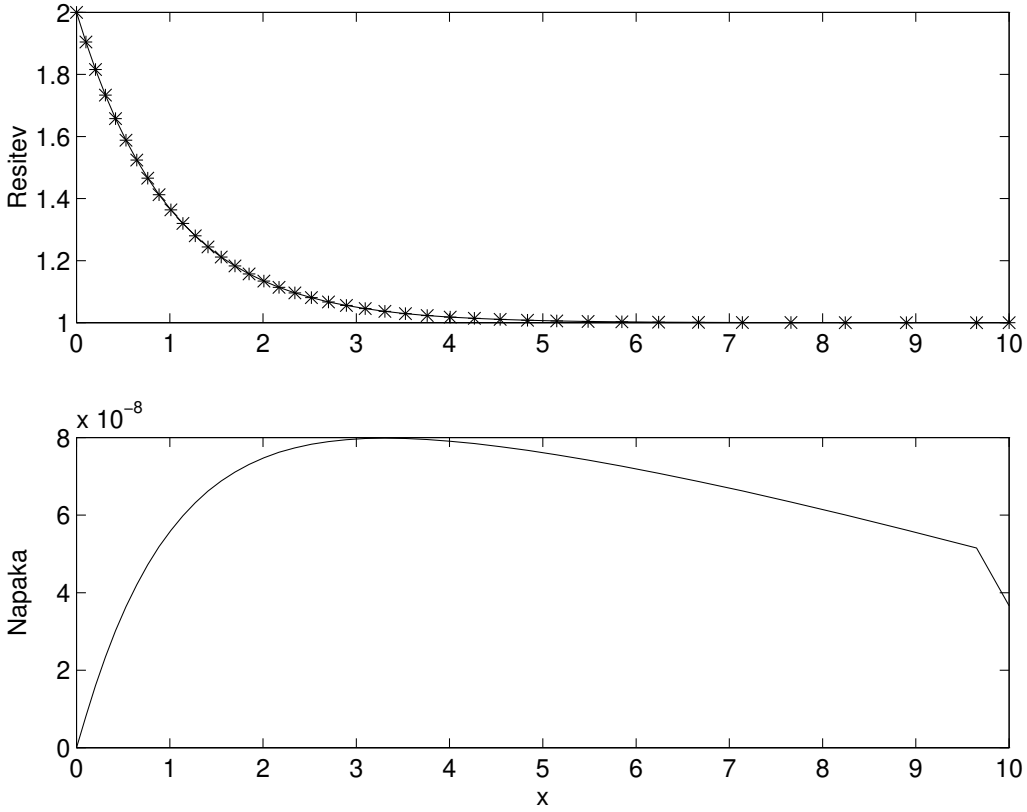
Algoritem 7.5. DOPRI5 Naj bo $y' = f(x, y)$ diferencialna enačba, $y(x_0) = y_0$ začetni pogoj, b končna točka intervala, na katerem nas zanima rešitev in ε zgornja meja za lokalno napako na enotskem intervalu (intervalu dolžine 1). Naslednji algoritem izračuna vektor približkov y_n k vrednostim rešitve $y(x_n)$ diferencialne enačbe v točkah $x_n = x_{n-1}+h_n$; $n = 1, \ldots, N$, $x_N = b$ s pomočjo metode DOPRI5.

$x = x_0$
$y = y_0$
$n = 1$
$\sigma = 0.9$
$h = (b - x)$
while $x < b$
    $k1 = h * f(x, y)$
    $k2 = h * f(x + h/5, y + k1/5)$
    $k3 = h * f(x + h * 0.3, y + k1 * 0.075 + k2 * 0.225)$
    $k4 = h * f(x + h * 0.8, y + 44 * k1/45 - 56 * k2/15 + 32 * k3/9)$
    $k5 = h * f(x + 8 * h/9, y + 19372 * k1/6561 - 25360 * k2/2187 +$
        $64448 * k3/6561 - 212 * k4/729)$
    $k6 = h * f(x + h, y + 9017 * k1/3168 - 355 * k2/33 +$
        $46732 * k3/5247 + 49 * k4/176 - 5103 * k5/18656)$
    $k7 = h * f(x + h, y + 35 * k1/384 + 500 * k3/1113 +$
        $125 * k4/192 - 2187 * k5/6784 + 11 * k6/84)$
    $l = 71 * k1/57600 - 71 * k3/16695 + 71 * k4/1920$
        $-17253 * k5/339200 + 22 * k6/525 - k7/40$
    if $|l| < \varepsilon * h$
        $y = y + (5179 * k1/57600 + 7571 * k3/16695 +$
            $393 * k4/640 - 92097 * k5/339200 + 187 * k6/2100 + k7/40)$
        $x = x + h$
        $h = \sigma * h * \sqrt[5]{\varepsilon h/|l|}$
        if $x + h > b$
            $h = b - x$
        end
    else
        $h = h/2$
    end
end

ε	$\max \lvert y_n - y(x_n)\rvert$	število korakov
10^{-0}	$2.8 \cdot 10^{-0}$	4
10^{-1}	$7.7 \cdot 10^{-2}$	5
10^{-2}	$1.9 \cdot 10^{-3}$	6
10^{-3}	$3.1 \cdot 10^{-4}$	8
10^{-4}	$4.5 \cdot 10^{-5}$	11
10^{-5}	$5.9 \cdot 10^{-6}$	16
10^{-6}	$7.0 \cdot 10^{-7}$	25
10^{-7}	$8.0 \cdot 10^{-8}$	40
10^{-8}	$8.6 \cdot 10^{-9}$	68
10^{-9}	$9.1 \cdot 10^{-10}$	118
10^{-10}	$9.4 \cdot 10^{-11}$	205
10^{-11}	$9.6 \cdot 10^{-12}$	358
10^{-12}	$9.8 \cdot 10^{-13}$	631

Tabela 7.8: Rešitev začetnega problema (7.5) z metodo DOPRI5 pri različnih tolerancah ε.

Primer 7.5. Poiščimo približno rešitev začetnega problema (7.5) na intervalu $[0, 10]$ tudi z metodo DOPRI5 in pri različnih zgornjih mejah za lokalno napako.
Iz rezultatov v tabeli 7.8 vidimo, da zgornja meja za lokalno napako dobro omejuje globalno napako. V zadnji koloni je navedeno število korakov (zavrnjenih in sprejetih), ki so bili potrebni za izračun rezultata na intervalu $[0, 10]$. Na sliki 7.4 je rešitev problema (7.5) z algoritmom 7.5 in njena globalna napaka, na sliki 7.5 pa so dolžine posameznih korakov, vse pri $\varepsilon = 10^{-7}$. □



Slika 7.4: Rešitev začetnega problema (7.5) z metodo DOPRI5 (zgoraj) in globalna napaka (spodaj) pri $\varepsilon = 10^{-7}$

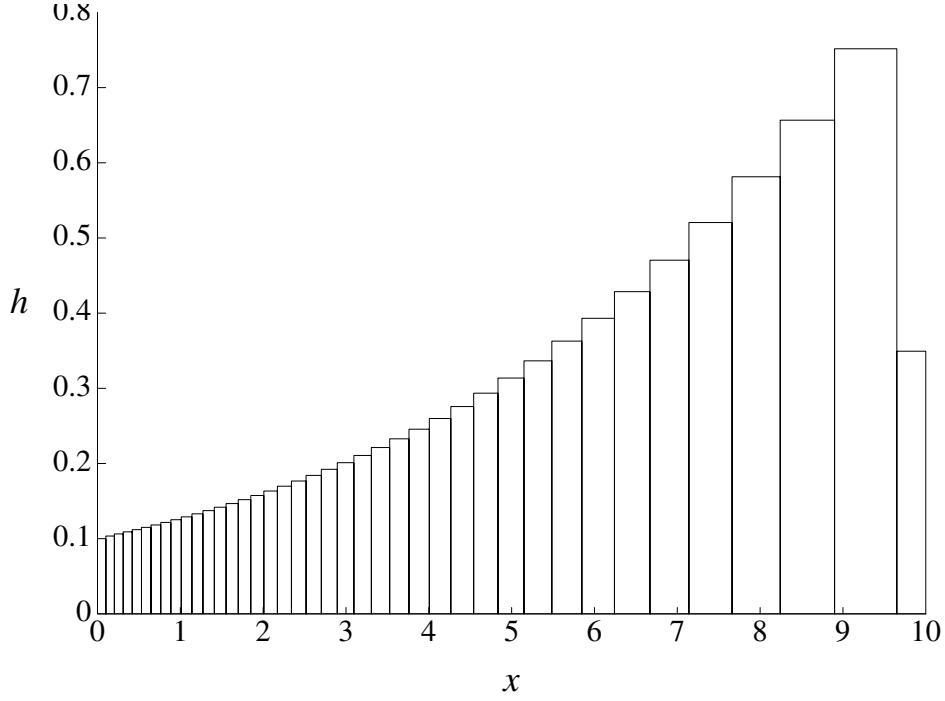
7.6 Sistemi diferencialnih enačb

Sistem m diferencialnih enačb

$$\begin{aligned} y_1' &= f_1(x, y_1, y_2, \ldots, y_m) \\ y_2' &= f_2(x, y_1, y_2, \ldots, y_m) \\ \ldots & \quad \ldots \\ y_m' &= f_m(x, y_1, y_2, \ldots, y_m) \end{aligned} \tag{7.16}$$

za m neznanih funkcij $y_1, \ldots, y_m$ ima v splošnem m-parametrično družino rešitev, zato potrebujemo še m dodatnih pogojev, ki naj bodo dani kot začetni pogoji

$$y_i(x_0) = y_{i,0} \qquad i = 1, \ldots, m. \tag{7.17}$$



Slika 7.5: Zaporedje dolžin korakov pri reševanju začetnega problema (7.5) z metodo DOPRI5 pri $\varepsilon = 10^{-7}$

Začetni problem (7.16, 7.17) lahko krajše zapišemo v vektorski obliki

$$\boldsymbol{y}' = \boldsymbol{f}(x, \boldsymbol{y}), \qquad \boldsymbol{y}(x_0) = \boldsymbol{y}_0, \tag{7.18}$$

kjer je $\boldsymbol{y} = [y_1, \ldots, y_m]^T$, $\boldsymbol{f} = [f_1, \ldots, f_m]^T$ in $\boldsymbol{y}_0 = [y_{1,0}, \ldots, y_{m,0}]^T$.

Za reševanje sistema enačb, zapisanega v vektorski obliki (7.18), lahko uporabljamo iste metode kot za eno samo enačbo, le pri zapisu algoritma moramo paziti na to, da imamo opravka z vektorji in ne skalarji. Zapišimo kot zgled varianto algoritma 7.3, prirejeno za reševanje sistema diferencialnih enačb.

Algoritem 7.6. ABM prediktor-korektor za sistem Naj bo

$$\boldsymbol{y}' = \boldsymbol{f}(x, \boldsymbol{y})$$

sistem m diferencialnih enačb, $\boldsymbol{y}(x_0) = \boldsymbol{y}_0$ začetni pogoj, h korak in N naravno število. Naslednji algoritem izračuna zaporedje vektorjev približkov $\boldsymbol{y}_n$ k vrednostim rešitve $\boldsymbol{y}(x_n)$ diferencialne enačbe v točkah $x_n = x_0 + nh;\ n = 1, \ldots, N$ s pomočjo ABM metode prediktor-korektor 4. reda.

$x = x_0$
$y = y_0$
$d = \mathbf{zeros}(m, 5)$
$d(:, 1) = f(x, y)$
for $n = 2 : 4$
    $x = x + h$
    Izračunaj dodatne vrednosti $y \approx y(x)$
    $d(:, n) = f(x, y)$
end
for $n = 4 : N$
    $x = x + h$
    $yp = y + h * (55 * d(:, 4) - 59 * d(:, 3) + 37 * d(:, 2) - 9 * d(:, 1))/24$
    $dp = f(x, yp)$
    $y = y + h * (9 * dp + 19 * d(:, 4) - 5 * d(:, 3) + d(:, 2))/24$
    $d(:, 5) = f(x, y)$
    for $j = 1 : 4$
        $d(:, j) = d(:, j + 1)$
    end
end

		RK4		ABM4	
n	x_n	$g_n(y)$	$g_n(z)$	$g_n(y)$	$g_n(z)$
0	0.0				
1	0.1	$3.9 \cdot 10^{-7}$	$5.8 \cdot 10^{-7}$	$-1.3 \cdot 10^{-7}$	$2.9 \cdot 10^{-7}$
2	0.2	$6.6 \cdot 10^{-7}$	$1.1 \cdot 10^{-6}$	$-3.2 \cdot 10^{-7}$	$4.8 \cdot 10^{-7}$
3	0.3	$8.1 \cdot 10^{-7}$	$1.6 \cdot 10^{-6}$	$-5.5 \cdot 10^{-7}$	$6.0 \cdot 10^{-7}$
4	0.4	$8.5 \cdot 10^{-7}$	$2.0 \cdot 10^{-6}$	$-8.2 \cdot 10^{-7}$	$6.3 \cdot 10^{-7}$
5	0.5	$7.8 \cdot 10^{-7}$	$2.3 \cdot 10^{-6}$	$-1.1 \cdot 10^{-6}$	$5.8 \cdot 10^{-7}$
6	0.6	$6.2 \cdot 10^{-7}$	$2.6 \cdot 10^{-6}$	$-1.4 \cdot 10^{-6}$	$4.7 \cdot 10^{-7}$
7	0.7	$3.7 \cdot 10^{-7}$	$2.7 \cdot 10^{-6}$	$-1.7 \cdot 10^{-6}$	$2.9 \cdot 10^{-7}$
8	0.8	$5.5 \cdot 10^{-8}$	$2.8 \cdot 10^{-6}$	$-1.9 \cdot 10^{-6}$	$6.3 \cdot 10^{-8}$
9	0.9	$-3.2 \cdot 10^{-7}$	$2.7 \cdot 10^{-6}$	$-2.2 \cdot 10^{-6}$	$-2.1 \cdot 10^{-7}$
10	1.0	$-7.3 \cdot 10^{-7}$	$2.6 \cdot 10^{-6}$	$-2.5 \cdot 10^{-6}$	$-8.2 \cdot 10^{-7}$

Tabela 7.9: Globalne napake komponent začetnega problema (7.19) z metodo Runge-Kutta 4. reda in z Adams-Bashforth-Moultonovo metodo prediktor-korektor 4. reda ($h = 0.1$).

Primer 7.6. Poiščimo približno rešitev začetnega problema

$$\begin{aligned} y' &= y - 2z - 2e^{-x} + 2 \\ z' &= 2y - z - 2e^{-x} + 1; \\ y(0) &= 1 \\ z(0) &= 1 \end{aligned} \tag{7.19}$$

na intervalu $[0, 1]$ z metodo Runge-Kutta in z Adams-Bashforth-Moultonovo metodo prediktor-korektor, obe z isto dolžino koraka $h = 0.1$. Začetne vrednosti za Adams-Bashforth-Moultonovo metodo so dobljene iz točne rešitve $y = e^{-x}$, $z = 1$ na intervalu $[-0.3, 0]$. Zapišimo še algoritem za izračun vrednosti funkcije f v tem primeru:

function $f = f(x, y)$
$f = [y(1) - 2 * y(2) - 2 * \mathbf{exp}(-x) + 2;$
$2 * y(1) - y(2) - 2 * \mathbf{exp}(-x) + 1];$

7.7 Enačbe višjih redov

Diferencialne enačbe višjih redov navadno rešujemo tako, da jih zapišemo kot sistem enačb prvega reda. To najlaže dosežemo tako, da za nove spremenljivke izberemo odvode neznane funkcije. Vzemimo diferencialno enačbo m-tega reda

$$y^{(m)} = f(x, y, y', \ldots, y^{(m-1)})$$

z začetnimi pogoji

$$\begin{aligned} y(x_0) &= y^{(0)} \\ y'(x_0) &= y^{(1)} \\ y''(x_0) &= y^{(2)} \\ &\vdots \\ y^{(m-1)}(x_0) &= y^{(m-1)}. \end{aligned}$$

Z uvedbo novih spremenljivk

$$\begin{aligned} y_1 &= y \\ y_2 &= y' \\ y_3 &= y'' \\ &\vdots \\ y_m &= y^{(m-1)} \end{aligned}$$

dobimo sistem enačb prvega reda

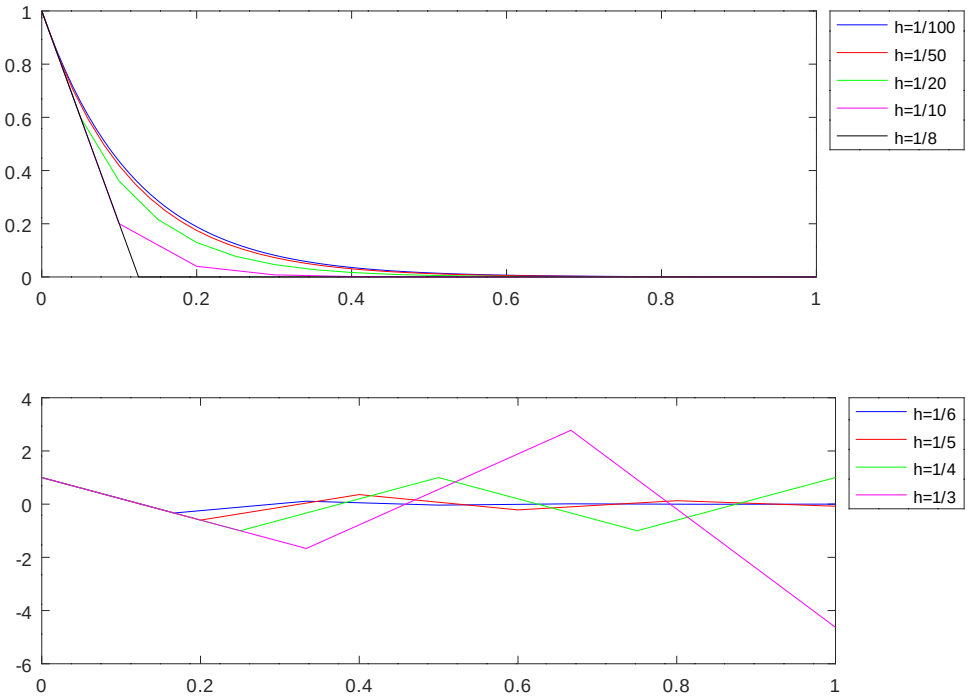
$$\begin{aligned} y_1' &= y_2 \\ y_2' &= y_3 \\ &\vdots \\ y_{m-1}' &= y_m \\ y_m' &= f(x, y_1, y_2, \ldots, y_m) \end{aligned}$$

z začetnimi pogoji

$$\begin{aligned} y_1(x_0) &= y^{(0)} \\ y_2(x_0) &= y^{(1)} \\ y_3(x_0) &= y^{(2)} \\ &\vdots \\ y_m(x_0) &= y^{(m-1)}, \end{aligned}$$

ki ga rešujemo s pomočjo že znanih metod.

7.8 Stabilnost



Slika 7.6: Rešitve testne enačbe $y' = -8y$ z Eulerjevo metodo pri različnih dolžinah koraka

Začnimo z enostavnim primerom:

Primer 7.7. Rešimo začetni problem

$$y' = -8y; \quad y(0) = 1 \tag{7.20}$$

na intervalu $x \in (0, 1)$ za različne dolžine koraka h.
Iz rezultatov na sliki 7.6 vidimo, da se numerične rešitve obnašajo zelo različno. Pri dolžinah koraka $h < 1/8$ numerične rešitve monotono padajo proti vrednosti 0, prav tako kakor točna rešitev $y = e^{-8x}$. Pri dolžinah koraka $1/8 < h < 1/4$ rešitve oscilirajo okoli točne rešitve in se ji približujejo, ko $x \to \infty$. Za dolžine korakov $h > 1/4$ pa rešitve oscilirajo okoli točne rešitve, njihove absolutne vrednosti pa neomejeno naraščajo (pravimo, da so nestabilne).

Poizkusimo pojasniti tako obnašanje rešitev. Ko rešujemo *testno diferencialno enačbo*

$$y' = \lambda y; \ \lambda < 0 \tag{7.21}$$

z Eulerjevo metodo, dobimo med dvema zaporednima približkoma relacijo

$$y_{n+1} = y_n + h\lambda y_n = (1 + h\lambda)y_n, \tag{7.22}$$

medtem ko točna rešitev $y = e^{\lambda x}$ zadošča podobni relaciji

$$y(x_{n+1}) = e^{h\lambda} y(x_n).$$

Ker je za $\lambda < 0$ faktor $|e^{h\lambda}| < 1$, točna rešitev (monotono) konvergira proti 0. Seveda želimo, da se podobno obnaša tudi numerična rešitev, zato mora tudi faktor $1 + h\lambda$ v enačbi (7.22) zadoščati podobni neenačbi kot točna rešitev, torej $|1 + h\lambda| < 1$. Od tod dobimo da mora biti $-2 < h\lambda < 0$. Interval $[-2, 0]$ zato imenujemo *interval absolutne stabilnosti* za Eulerjevo metodo.

Na podoben način je definiran interval absolutne stabilnosti tudi za druge metode: interval absolutne stabilnosti je največji interval $[-a, 0]$, da je numerična rešitev testne diferencialne enačbe (7.21) omejena za vsak $-a \leq h\lambda \leq 0$. Za metode, ki smo jih srečali v tem poglavju, so intervali absolutne stabilnosti zbrani v tabeli 7.10.

Izkaže se, da je obnašanje metode pri reševanju testne enačbe (7.21) značilno za to, kako globalna napaka iz prejšnjega koraka vpliva na globalno napako v naslednjem koraku. Kadar je $h\lambda$ zunaj intervala absolutne stabilnosti, se globalna napaka iz koraka v korak povečuje. Zato moramo pri reševanju diferencialnih enačb paziti, da je korak metode vedno v mejah, ki

Metoda	interval abs. stabilnosti
AB1 (Eulerjeva)	$[-2, 0]$
AB2	$[-1, 0]$
AB3	$[-0.65, 0]$
AB4	$[-0.4, 0]$
AM1 (Implicitna Eulerjeva)	$[-\infty, 0]$
AM2 (trapezna)	$[-\infty, 0]$
AM3	$[-6, 0]$
AM4	$[-3, 0]$
ABM4	$[-1.4, 0]$
RK2 (obe metodi)	$[-2, 0]$
RK4 (klasična Runge-Kutta)	$[-2.8, 0]$
DOPRI5	$[-3.3, 0]$

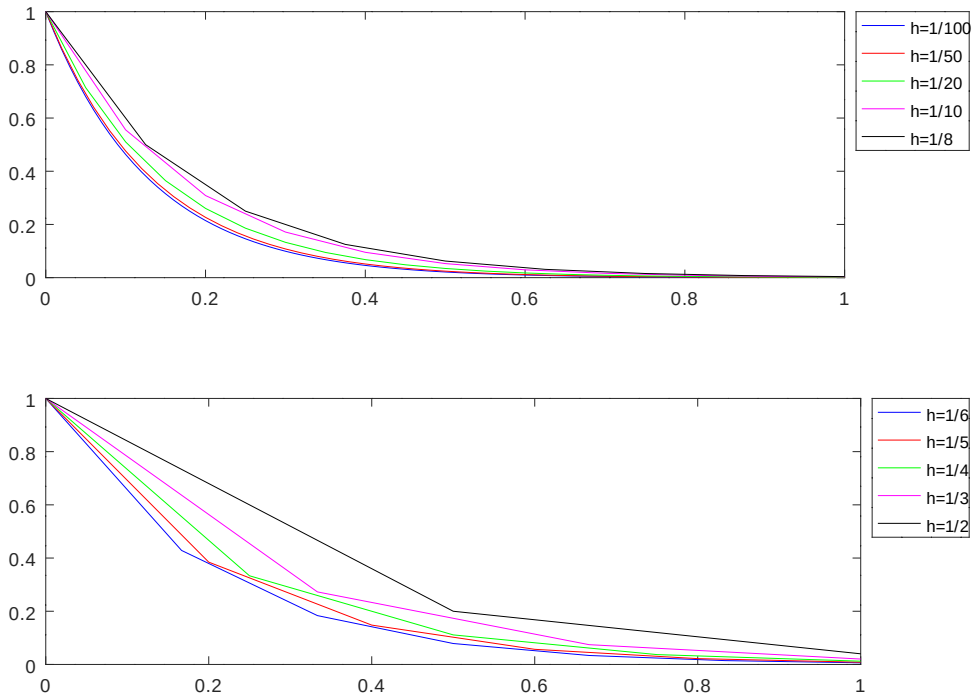
Tabela 7.10: Intervali absolutne stabilnosti

jih določa interval absolutne stabilnosti, sicer lahko napaka v rešitvi nekontrolirano narašča.

Primer 7.8. Rešimo začetni problem (7.20) še z implicitno Eulerjevo metodo (7.9).
Iz rezultatov na sliki 7.7 vidimo, da pri reševanju z implicitno Eulerjevo metodo zaradi njene stabilnosti (interval absolutne stabilnosti implicitne Eulerjeve metode je $(-\infty, 0]$, glej tabelo 7.10) izračunana rešitev dobro sledi pravi rešitvi tudi pri večjih dolžinah koraka.

7.9 Robni problemi

Dodatni pogoji, ki določajo partikularno rešitev diferencialne enačbe niso nujno podani v isti točki (začetni pogoj). Predvsem pri enačbah višjih redov in pri sistemih diferencialnih enačb se lahko zgodi, da imamo namesto začetnih pogojev dane *robne pogoje*.



Slika 7.7: Rešitve testne enačbe $y' = -8y$ z implicitno Eulerjevo metodo pri različnih dolžinah koraka

Primer 7.9. Za sistem diferencialnih enačb

$$\begin{aligned} y' &= y^2/z \\ z' &= y/2 \end{aligned} \tag{7.23}$$

naj bo robni pogoj podan kot

$$y(0) = 1/2 \quad \text{in} \quad y(1) = 2.$$

Ker pri robnih pogojih nimamo dovolj začetne informacije o rešitvi, da bi lahko izračunali odvod v začetni točki, ne moremo uporabiti že poznanih metod za reševanje začetnih problemov.

Opisali bomo tri metode za reševanje robnh problemov: *strelsko metodo*, ki je dovolj splošna, da z njo lahko rešujemo tako linearne kot tudi nelinearne robne probleme ter *metodo končnih diferenc* in *kolokacijsko metodo*, ki sta

primerni predvsem za linearne enačbe.

Strelska metoda Za primer vzemimo robni problem

$$\begin{aligned} y' &= f(x,y,z) \\ z' &= g(x,y,z); \end{aligned} \tag{7.24}$$

$$y(a) = A, \quad y(b) = B.$$

Ker želimo za reševanje problema (7.24) uporabiti kakšno izmed obravnavanih metod za reševanje začetnih problemov, moramo manjkajočo vrednost $z(a)$ nadomestiti z neznanim parametrom, npr. α in njegovo vrednost določiti tako, da bo funkcija y pri $x = b$ zavzela predpisano vrednost B.

Izberimo za α poljubno vrednost in rešimo začetni problem

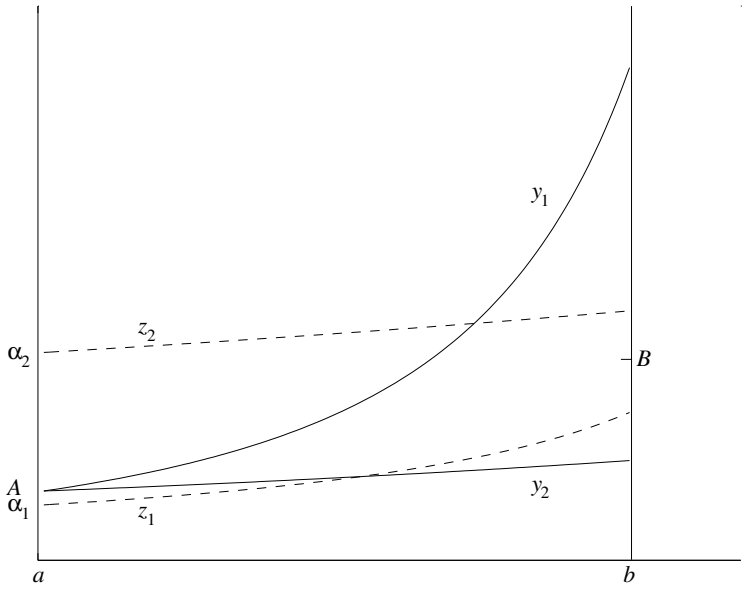
$$\begin{aligned} y' &= f(x,y,z) \\ z' &= g(x,y,z); \end{aligned} \tag{7.25}$$

$$y(a) = A, \quad z(a) = \alpha$$

na intervalu $[a, b]$. Vrednost spremenljivke y pri $x = b$ je tako odvisna od izbire parametra α, torej $y(b) = F(\alpha)$. Vrednost parametra α, za katerega je $y(b) = B$ je torej rešitev nelinearne enačbe

$$F(\alpha) - B = 0. \tag{7.26}$$

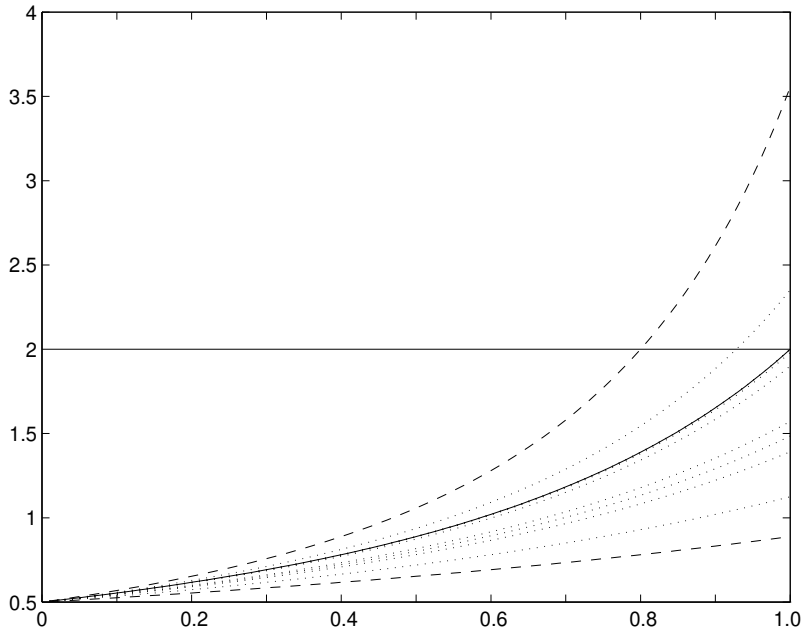
Izberimo za začetek dve različni vrednosti α_0 in α_1 in rešimo začetni problem (7.25) za obe vrednosti (slika 7.8). Rešitev nelinearne enačbe (7.26) sedaj lahko poiščemo s sekantno metodo. Zapišimo algoritem:



Slika 7.8: Dve rešitvi začetnega problema (7.25)

Algoritem 7.7. Strelska metoda Naj bo (7.24) dan robni problem, ε največje dopustno odstopanje izračunane rešitve od predpisane vrednosti B, α_0 in α_1 dva začetna približka za $z(a)$ in m naravno število. Naslednji algoritem izračuna pravilno začetno vrednost $z(a) = \alpha$ in rešitev robnega problema, ali pa se po m iteracijah konča brez rezultata ($\alpha = NaN$).

Reši začetni problem (7.25) za $\alpha = \alpha_0$ z
    eno izmed metod za reševanje začetnih problemov
$ys = y$
Reši začetni problem (7.25) za $\alpha = \alpha_1$
$yn = y$
$j = 1$
while (**abs**$(yn - B) > \varepsilon) * (j < m)$
    $j = j + 1$
    $\alpha = \alpha_0 + (\alpha_1 - \alpha_0) * (B - ys)/(yn - ys)$
    Reši začetni problem (7.25) za α
    $ys = yn$
    $yn = y$
    $\alpha_0 = \alpha_1$
    $\alpha_1 = \alpha$
end
if abs$(yn - B) > \varepsilon$
    $\alpha = NaN$
end



Slika 7.9: Reševanje robnega problema (7.27) s strelsko metodo

Poglejmo strelsko metodo na delu.

Primer 7.10. Rešimo robni robni problem

$$\begin{aligned} y' &= \frac{y^2}{z}; \qquad y(0) = \frac{1}{2} \\ z' &= \frac{y}{2}: \qquad y(1) = 2 \end{aligned} \tag{7.27}$$

s strelsko metodo.
Računamo z algoritmom 7.7, začetne probleme računamo z metodo Runge-Kutta, za manjkajoči začetni vrednosti odvoda izberemo $\alpha_0 = 1$ in $\alpha_1 = 0.4$. Rezultati računanja so povzeti v tabeli 7.11. Končna vrednost $y(1) = 2$ je bila pri natančnosti 10^{-12} dosežena po 12 korakih iteracije.

Metoda končnih diferenc Naj bo

$$y'' + f(x)y' + g(x)y = h(x) \tag{7.28}$$

n	α	$y(1)$
0	1.000	.8889
1	.4000	3.556
2	.7500	1.125
3	.6240	1.392
4	.3369	7.520
5	.5955	1.485
6	.5734	1.572
7	.4639	2.352
8	.5133	1.900
9	.5024	1.981
10	.4998	2.001
11	.5000	2.000

Tabela 7.11: Rezultati reševanja robnega problema (7.27) s strelsko metodo

linearna diferencialna enačba 2. reda ter

$$y(a) = A \quad \text{in} \quad y(b) = B \quad (a < b) \tag{7.29}$$

ustrezna robna pogoja. Interval $[a, b]$ bomo razdelili na N enakih podintervalov dolžine $h = (b-a)/N$ z notranjimi točkami $x_i = a+ih,\ i = 1, \ldots, N-1$. Posebej označimo še $x_0 = a$ in $x_N = b$. Vrednosti rešitve v notranjih točkah bomo zapisali kot

$$y_i \approx y(x_0 + ih),\ i = 1, \ldots, N-1.$$

Da bi izračunali rešitev robnega problema (7.28–7.29) z metodo končnih diferenc, moramo diferencialno enačbo (7.28) zapisati za vsako od notranjih točk, nato pa vse odvode zamenjati z ustreznimi približki s končnimi diferencami. Če je le mogoče, uporabimo sredinske formule, na primer približek (6.32) za prvi odvod in (6.36) za drugi odvod. Tako dobimo

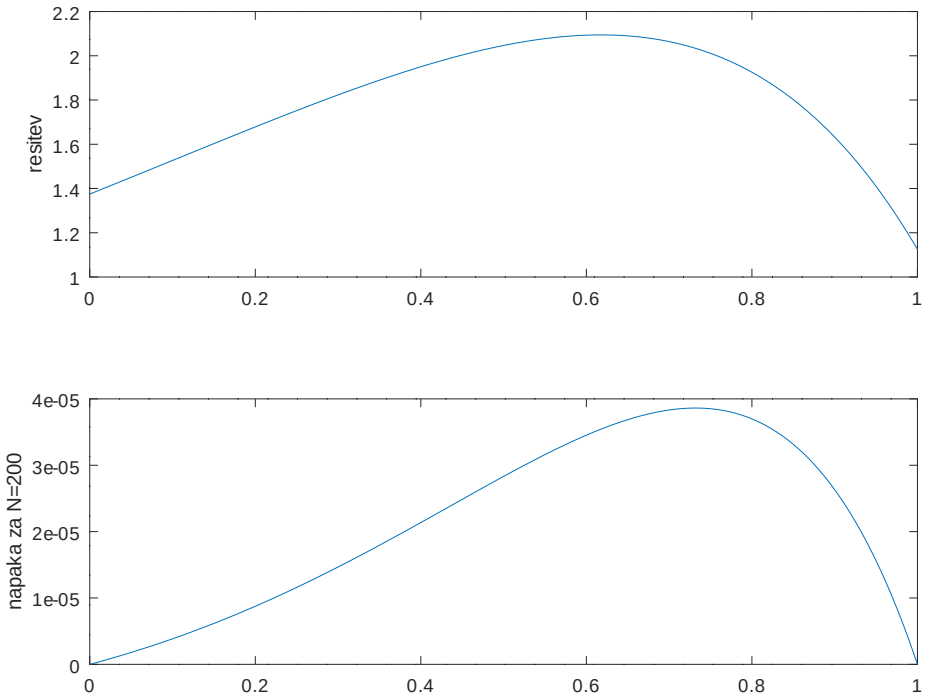
$$\frac{y_{i-1} - 2y_i + y_{i+1}}{h^2} + f(x_i)\frac{y_{i+1} - y_{i-1}}{2h} + g(x_i)y_i = h(x_i), \quad i = 1, \ldots, N-1, \tag{7.30}$$

kar je sistem linearnih enačb za neznane vrednosti rešitve $y_i,\ i = 1, \ldots, N-1$. V sistemu nastopata tudi y_0 in y_N, ki sta znani robni vrednosti. Če vse

enačbe (7.30) pomnožimo s h^2, pišemo f_i namesto $f(x_i)$, g_i namesto $g(x_i)$, h_i namesto $h(x_i)$ in preuredimo, dobimo

$$\begin{aligned}
(-2+h^2g_1)y_1 \qquad &+\left(1+h\tfrac{f_i}{2}\right)y_2 = h^2h_1 - A\left(1-h\tfrac{f_1}{2}\right), \\
\left(1-h\frac{f_i}{2}\right)y_{i-1} \qquad &+(-2+h^2g_i)y_i + \left(1+h\tfrac{f_i}{2}\right)y_{i+1} = h^2h_i, \qquad (7.31)\\
i = 2,\ldots,N-2,& \\
\left(1-h\frac{f_{N-1}}{2}\right)y_{N-2} \quad &+(-2+h^2g_{N-1})y_{N-1} = h^2h_{N-1} - B\left(1+h\tfrac{f_{N-1}}{2}\right).
\end{aligned}$$

Ta sistem enačb je tridiagonalen, zato ga lahko učinkovito rešimo v $\mathcal{O}(N)$ operacijah brez pivotiranja.



Slika 7.10: Rešitev robnega problema iz primera (7.11) z metodo končnih diferenc (zgoraj) in njena napaka pri $N = 200$ točkah (spodaj).

N	globalna napaka
2	$6.41 \cdot 10^{-1}$
10	$1.57 \cdot 10^{-2}$
100	$1.54 \cdot 10^{-4}$
1000	$1.54 \cdot 10^{-6}$
10000	$1.52 \cdot 10^{-8}$

Tabela 7.12: Globalna napaka rešitve robnega problema iz primera 7.11 pri različnih vrednostih N.

Primer 7.11. Izračunajmo približno rešitev linearnega robnega problema

$$y'' - 4y' + 4y = x^2, \quad \text{in} \quad y(0) - \frac{11}{8},\ y(1) = \frac{9}{8}$$

z metodo končnih diferenc.

Na sliki 7.10 je rešitev robnega problema (zgoraj) in globalna napaka za $N = 200$ (spodaj). V tabeli 7.12 je maksimalna globalna napaka na $[0, 1]$ pri različnih vrednostih N.

Kolokacija Naj bo $\{\phi_j,\ j = i, \ldots, N\}$ množica linearno neodvisnih funkcij, ki jih bomo imenovali *kolokacijske funkcije*. Pri metodi kolokacije rešitev robnega problema iščemo kot linearno kombinacijo

$$u_N(x) = \sum_{j=1}^{N} d_j \phi_j(x)\,. \tag{7.32}$$

Koeficiente d_j izberemo tako, da funkcija $u_N(x)$ zadošča robnima pogojema (7.29) in diferencialni enačbi (7.28) eksaktno v izbranih $N-2$ točkah (imenovali jih bomo *kolokacijske točke*) v notranjosti intervala $[a, b]$. Zadoščati mora torej enačbam

$$\begin{aligned} u_N(a) &= A \\ u_N(b) &= B \\ u_N''(x_i) + f(x_i)u_N'(x_i) + g(x_i)u_N(x_i) &= h(x_i)\,, \end{aligned} \tag{7.33}$$

kjer so $x_i,\ i = 1, \ldots, N-2$ izbrane točke znotraj intervala $[a, b]$.

Pogoje (7.33) sestavlja N linearnih enačb za N neznanih koeficientov d_j. Ko ta sistem enačb rešimo (s kakšno metodo iz poglavja 2), lahko rešitve d_j vstavimo v (7.32) in dobimo približek rešitve.

Kako izbrati množico kolokacijskih funkcij ϕ_j? Za učinkovitost metode bi bilo zaželeno, da imajo funkcije ϕ_j eno ali več izmed naslednjih lastnosti:

1. Funkcije ϕ_j so zvezno odvedljive na intervalu $[a, b]$.
2. Funkcije ϕ_j so ortogonalne na intervalu $[a, b]$, torej
$$\int_a^b \phi_j(x)\phi_k(x)\,dx = 0 \text{ kadarkoli je } j \neq k.$$
3. Funkcije ϕ_j so enostavno izračunljive, na primer polinomi ali trigonometrični polinomi.
4. Funkcije ϕ_j zadoščajo homogenim robnim pogojem na intervalu $[a, b]$, torej
$$\phi_j(a) = 0 = \phi_j(b).$$

Tem zahtevam najbolje ustrezajo ortogonalni polinomi. Največ se uporabljajo Legendreovi polinomi in polinomi Čebiševa. V primeru, ko so robni pogoji periodični, pa se raje uporabljajo trigonometrični polinomi.

Primer 7.12. Izračunajmo rešitev linearnega robnega problema iz primera 7.11 še z metodo kolokacije.
Za kolokacijske funkcije bomo izbrali polinome Čebyševa, za kolokacijske točke pa ničle polinoma Čebiševa.

Na sliki 7.11 je napaka robnega problema za $N = 5$ (zgoraj) in za $N = 10$ (spodaj). V tabeli 7.13 je maksimalna globalna napaka na $[0, 1]$ pri različnih vrednostih N.

Analiza napake kolokacijske metode je precej zapletena, zato jo bomo tukaj izpustili. V praksi dobimo informacijo o napaki, če ponovimo izračun pri več različnih N.

Če primerjamo metodo končnih diferenc in kolokacijsko metodo lahko ugotovimo, da kolokacijska metoda konvergira mnogo hitreje od metode končnih diferenc, kar pomeni, da je potrebno rešiti manjši sistem linearnih enačb, Pri metodi končnih diferenc je sistem enačb sicer precej večji, vendar je njegova matrika zelo razpršena (pasovna), kar zelo olajša reševanje.

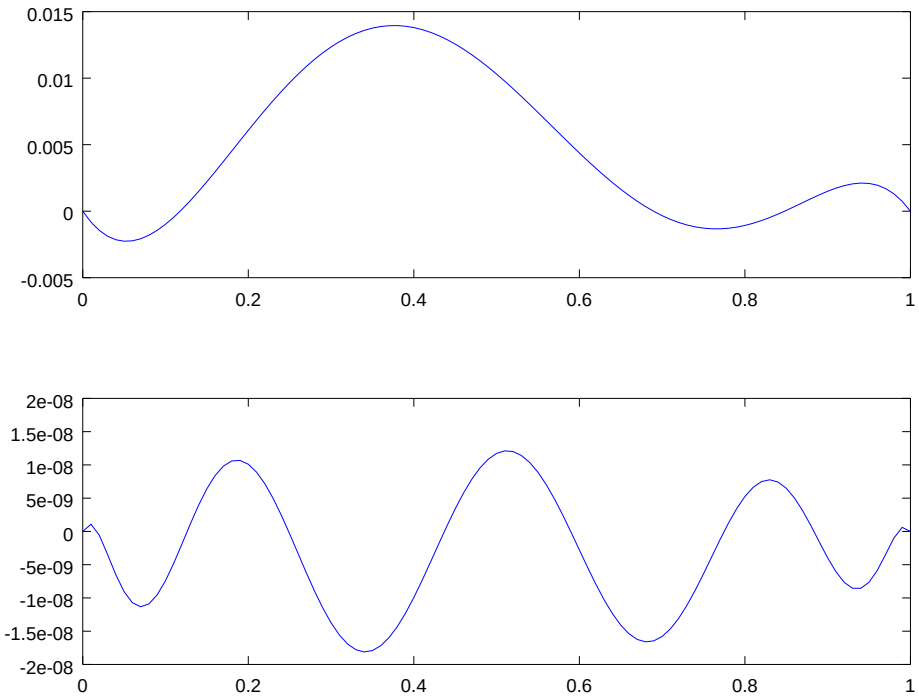
N	globalna napaka
2	$8.77 \cdot 10^{-1]}$
3	$7.08 \cdot 10^{-1}$
4	$1.03 \cdot 10^{-1}$
5	$1.39 \cdot 10^{-2}$
6	$1.17 \cdot 10^{-3}$
7	$7.67 \cdot 10^{-5}$
8	$5.73 \cdot 10^{-6}$
9	$3.38 \cdot 10^{-7}$
10	$1.81 \cdot 10^{-8}$
11	$8.37 \cdot 10^{-10}$
12	$3.97 \cdot 10^{-11}$
13	$1.60 \cdot 10^{-12}$
14	$6.17 \cdot 10^{-14}$
15	$3.11 \cdot 10^{-15}$

Tabela 7.13: Globalna napaka rešitve robnega problema iz primera 7.12 s kolokacijo pri različnem številu kolokacijskih točk N.

7.10 Povzetek

Izmed problemov, povezanih z reševanjem sistemov navadnih diferencialnih enačb, smo se ukvarjali predvsem z začetnimi problemi za diferencialno enačbo prvega reda. Najenostavnejša metoda za reševanje teh problemov je Eulerjeva metoda, ki pa žal za resno uporabo ni dovolj natančna.

Izmed linearnih veččlenskih metod smo spoznali dve družini Adamsovih metod: eksplicitne Adams-Bashforthove in implicitne Adams-Moultonove metode. Pri eksplicitnih formulah je na vsakem koraku potrebno le enkrat računati vrednost desne strani diferencialne enačbe in njihova uporaba je razmeroma enostavna. Implicitne formule so praviloma natančnejše, vendar je njihova uporaba zapletenejša, saj moramo na vsakem koraku rešiti nelinearno enačbo, zato jih uporabljamo navadno skupaj z ustrezno eksplicitno metodo kot metodo prediktor-korektor. V tem primeru moramo na vsakem koraku dvakrat računati vrednost desne strani diferencialne enačbe. Glavni pomanjkljivosti teh metod sta: zapleteno spreminjanje dolžine koraka in potreba po posebni proceduri, s katero na začetku izračunamo dodatne vrednosti (za kar



Slika 7.11: Napaka rešitve robnega problema iz primera (7.12) z metodo kolokacije za $N = 4$ (zgoraj) in za $N = 10$ (spodaj).

navadno uporabimo ustrezno metodo Runge-Kutta).

Metode tipa Runge-Kutta so enočlenske in zato enostavnejše za uporabo. Pri njih lahko enostavno menjamo dolžino koraka. So tudi dovolj natančne, njihova pomanjkljivost pa je, da moramo na vsakem koraku večkrat računati vrednost desne strani diferencialne enačbe, zato je reševanje počasnejše.

Reševanje začetnih problemov za sisteme diferencialnih enačb poteka podobno, le formule, ki smo jih spoznali za reševanje začetnih problemov z eno samo diferencialno enačbo, moramo uporabiti za vsako komponento posebej.

Diferencialne enačbe višjega reda navadno rešujemo tako, da jih zapišemo kot sistem diferencialnih enačb prvega reda.

Reševanje robnih problemov prevedemo na reševanje zaporedja začetnih problemov, kjer dodatne začetne vrednosti izbiramo tako, da zadovoljimo preostale robne pogoje. Kadar je diferencialna enačba linearna, pa je bolje uporabiti metodo končnih diferenc al kolokacijsko metodo.

Spoznali smo tudi, kako lahko sproti, med samim reševanjem ocenjujemo lokalno napako metode in obenem prilagajamo dolžino koraka metode tako, da so lokalne napake manjše od dovoljenih.

7.11 Problemi

1. V identiteti (7.6) nadomesti funkcijo $f(x, y(x))$ z interpolacijskim polinomom skozi točki x_n in x_{n-1} in integriraj. Tako dobiš dvostopenjsko Adams-Bashforthovo metodo

$$y_{n+1} = y_n + \frac{h}{2}(3f_n - f_{n-1}),$$

   katere lokalna napaka je enaka $\frac{5}{12}h^3y'''(\xi_n)$.

2. V identiteti (7.6) nadomesti funkcijo $f(x, y(x))$ z interpolacijskim polinomom skozi točki x_{n+1} in x_n ter integriraj. Tako dobiš dvostopenjsko Adams-Moultonovo (trapezno) metodo

$$y_{n+1} = y_n + \frac{h}{2}(f_{n+1} + f_n)$$

   z lokalno napako $\frac{-1}{12}h^3y'''(\xi_n)$.

3. Rešujemo začetni problem $y'(x) = y(x)$, $y(0) = 2$.

   (a) Poišči numerično rešitev enačbe z Eulerjevo metodo na intervalu $[0, 1]$ s korakom $h = 0.25$.

   (b) Poišči numerično rešitev še z metodo Runge-Kutta četrtega reda z enakim korakom.

   (c) Izračunaj točno rešitev diferencialne enačbe.

   (d) Kolikšna je globalna napaka rešitve iz točke a) pri $x = 1$? Kolikšna je globalna napaka rešitve iz točke b) pri $x = 1$?

4. Na intervalu $[0, 1]$ iščemo rešitev diferencialne enačbe

$$y'(x) = -2xy(x)$$

   pri pogoju, da je $y(0) = 1$.

(a) Prepričaj se, da je
$$y(x) = Ce^{-x^2}$$
rešitev diferencialne enačbe in določi C, da bo izpolnjen začetni pogoj.

(b) Izračunaj približno vrednost $y(1)$ z metodo Runge-Kutta reda 2 in korakom $h = 0.5$.

(c) Kolikšna je absolutna in relativna napaka približka za $y(1)$?

5. Rešujemo začetni problem
$$y' = 2\frac{y}{x} + \frac{3}{x^2}; \quad y(1) = 0.$$

(a) Poišči točno rešitev začetnega problema.

(b) Z Eulerjevo metodo izračunaj približno rešitev na intervalu $[1, 2]$ z dolžinami koraka $h = 0.5$, $h = 0.1$ in $h = 0.01$. Kolikšni sta lokalna in globalna napaka pri $x = 2$?

(c) Izračunaj približno rešitev z metodo Runge-Kutta. Kolikšni sta lokalna in globalna napaka pri $x = 2$ v tem primeru?

(d) Izračunaj približno rešitev še z metodo prediktor-korektor ABM4 (dodatne začetne vrednosti naj bodo izračunane z metodo Runge-Kutta). Kolikšni sta lokalna in globalna napaka pri $x = 2$ v tem primeru?

6. Rešujemo začetni problem $y' = y + e^x$; $y(0) = 0$.

(a) Poišči točno rešitev problema.

(b) Izračunaj približke za $y(x)$ v točkah $x = 0.1$ in $x = 0.2$ z metodo Runge Kutta drugega reda ($h = 0.1$).

(c) Izračunaj približno vrednost rešitve tega začetnega problema na intervalu $[0, 1]$ z Adams-Bashforth-Moultonovo metodo prediktor-korektor 3. reda. Kolikšni sta globalna in lokalna napaka metode pri $x = 0.7$?

7. Algoritem 7.4 priredi za reševanje sistema diferencialnih enačb in z njim reši začetni problem (7.19).

8. Dan je začetni problem

$$y'' + 4y' + 13y = 40\cos x; \quad y(0) = 3;\ y'(0) = 4$$

   (a) Diferencialno enačbo drugega reda zapiši kot sistem dveh diferencialnih enačb prvega reda.

   (b) Izračunaj točno rešitev.

   (c) Izračunaj rešitev z Eulerjevo metodo na intervalu $[0, 1]$ z dolžinami koraka $h = 0.5$, 0.05 in 0.005. Kolikšna je globalna napaka v končni točki?

   (d) Ponovi račun z metodo Runge-Kutta. Kolikšna je sedaj napaka?

   (e) Ponovi račun še z metodo prediktor-korektor ABM4 (dodatne začetne vrednosti so izračunane z metodo Runge-Kutta). Kolikšna je globalna napaka v tem primeru?

9. Iščemo rešitev začetnega problema

$$x'' - 3x' - 10x = 0; \quad x(0) = 3 \quad \text{in} \quad x'(0) = 15.$$

   (a) Poišči točno rešitev tega začetnega problema.

   (b) Diferencialno enačbo zapiši kot sistem diferencialnih enačb prvega reda. Zapiši tudi ustrezne začetne pogoje.

   (c) Izračunaj rešitev z Eulerjevo metodo s korakom $h = 0.2$ na intervalu $[0, 1]$.

   (d) Na istem intervalu izračunaj še rešitev z metodo Runge-Kutta s korakom $h = 0.5$. Kako lahko ocenimo, katera od obeh rešitev ima manjšo napako?

10. Rešujemo začetni problem

$$y'' + 102y' + 200y = 0;$$
$$y(0) = 1, \quad y'(0) = -2.$$

   (a) Poišči točno rešitev tega problema.

   (b) Zapiši algoritem za reševanje tega problema z Adams-Bashforth-Moultonovo metodo prediktor-korektor tretjega reda.

(c) Reši začetni problem z Eulerjevo metodo s korakom $h = 0.2$. Komentiraj dobljeni rezultat!

(d) Točna rešitev začetnega problema $y' = -100y; y(0) = 1$ je padajoča funkcija. Za kakšne velikosti koraka h bo tudi numerična rešitev, izračunana z Eulerjevo metodo, padajoča?

11. Začetni problem

$$y' + \frac{y}{x} = -x; \quad y(1) = 1$$

na intervalu $(1, 2)$ rešujemo z Milne-Simpsonovo metodo prediktor-korektor

$$\begin{aligned} y^p_{n+1} &= y_{n-3} + \frac{4h}{3}(2f_n - f_{n-1} + 2f_{n-2}) \\ y_{n+1} &= y_{n-1} + \frac{h}{3}(f^p_{n+1} + 4f_n + f_{n-1}). \end{aligned}$$

(a) Izračunaj red metode (Navodilo: razvoj po Taylorjevi formuli).

(b) Zapiši algoritem.

(c) Izračunaj točno rešitev začetnega problema.

(d) Izračunaj numerično rešitev s koraki $h = 0.1$, $h = 0.01$ in 0.001. Za dodatne začetne vrednosti vzemi kar vrednosti prave rešitve. Kako se obnaša globalna napaka?

Dodatek A

Programski paketi

Algoritmi, ki v tej knjigi spremljajo opise posameznih numeričnih metod, so namenjeni zgolj boljšemu razumevanju. Njihov namen nikakor ni uporaba pri reševanju zapletenih problemov, saj je danes uporabnikom na voljo že dovolj natančnejših, učinkovitejših in vsestransko uporabnih programov, ki so to nalogo sposobni opraviti mnogo bolje. V tem dodatku si bomo ogledali nekatere izmed njih.

Ker se stanje na tem področju neprestano spreminja, si lahko najnovejše informacije o numeričnih, splošno matematičnih programih in programskih knjižnicah ogledate na naslovu `https://en.wikipedia.org/wiki/List_of_numerical-analysis_software`.

A.1 Komercialni matematični programi

A.1.1 Integrirani numerični programi

V zadnjih desetih letih se je na tržišču pojavilo več programov, ki so enostavni za uporabo in namenjeni numeričnemu računanju. Pojavljajo pa se že podobni programi nekomercialnih proizvajalcev, ki so v prosti uporabi.

Nekaj naslovov:

MATLAB

The Mathworks, Inc.

el. pošta: info@mathworks.com

www: `http://www.mathworks.com`

Program MATLAB je verjetno vodilni na tem področju. Namenjen je predvsem numeričnemu računanju z vektorji in matrikami. Poleg osnovne verzije lahko dobimo (od proizvajalca ali prosto na mreži) dodatna orodja (angl. *toolbox*), ki so namenjena uporabi MATLABa pri reševanju specializiranih problemov (optimizacija, kontrola sistemov, nevronske mreže in podobno). Z dodatnim orodjem za simbolično matematiko postane MATLAB učinkovit tudi pri analitičnem reševanju matematičnih problemov.

Gauss:

APTECH SYSTEMS, inc.

el. pošta info@aptech.com

www `http://www.aptech.com/`

Program Gauss je namenjen matematični in statistični analizi podatkov in vizualizaciji. Njegovo jedro je močan matrično orientiran programski jezik. N razpolago so tudi dodatki (angl. Application Modules), ki razširjajo uporabnost paketa.

A.1.2 Integrirani simbolični matematični programi

Sredi osemdesetih let so se pojavili prvi programi za simbolično računanje. Njihova uporaba vse bolj spreminja tradicionalne pristope pri matematiki. Ti programi imejo velik vpliv na poučevanje in raziskovanje v matematiki.

Navajamo dva popularna integrirana simbolična programa, ki sta na razpolago za najrazličnejše tipe računalnikov, od žepnih kalkulatorjev preko osebnih do superračunalnikov:

MAPLE
Waterloo Maple Inc.

el. pošta: info_web@maplesoft.com

www: `http://www.maplesoft.com/`

Mathematica
Wolfram Research, Inc.

el. pošta: info@wolfram.com

www: `http://www.wri.com/`

A.2 Javno dostopni matematični paketi

Te vrste programi večinoma vsebujejo knjižnico numeričnih programov in interpretativni jezik, ki omogoča njihovo uporabo. Navajamo nekaj takih programov skupaj z naslovi in poglavitnimi značilnostimi:

Octave
www: `http://www.octave.org/`

Med podobnimi programi je še najbolj podoben MATLABu. Omogoča tudi računanje vrednosti integralov na neskončnih intervalih ter reševanje diferencialnih in diferencialno-algebrajskih enačb. Njegova distribucija vsebuje tudi več kot 200 strani priročnika.

Scilab
www: `http://scilabsoft.inria.fr/`

Podobno kot pri MATLABu ima na razpolago več dodatnih knjižnic (linearna algebra, kontrolna teorija, procesiranje signalov, simulacije, optimizacije). V povezavi s programom MAPLE omogoča tudi simbolično računanje.

Euler
www: `http://mathsrv.ku-eichstaett.de/MGF/homes/grothmann/euler/`

Podoben MATLABu, vendar uporablja svoj programski jezik.

PETSc
el. pošta: petsc-maint@mcs.anl.gov

www: `https://www.mcs.anl.gov/petsc/`

Zbirka orodij za (paralelno) računanje modelov, ki jih lahko opišemo z diferencialnimi enačbami.

A.3 Programske knjižnice

Pri razvoju lastnih programov lahko koristno uporabljamo veliko število že napisanih programov. Večina je napisana v Fortranu, vse pogosteje uporabljajo tudi C, in C++, nekateri programi pa so napisani tudi v Pascalu, Adi, ...

A.3.1 Komercialne knjižnice

Komercialne knjižnice navadno sestavljajo

- knjižnica numeričnih programov
- grafični programi
- lupina knjižnice z interaktivnim interpreterskim jezikom.

in so tako vedno bolj podobne integriranim numeričnim programom.

Jedro velikih komercialnih knjižnic sestavlja veliko število podprogramov za reševanje najrazličnejših problemov numerične matematike in statistike. Podprogrami so napisani v več programskih jezikih (Fortran, C, C++). Na voljo so za večino velikih računalnikov in delovnih postaj in tudi za nekatere mikroračunalnike. Najbolj znani sta:

NAG (The numericalAlgorithms Group)
www: `http://www.nag.co.uk/`

IMSL
www: `https://docs.roguewave.com/en/imsl-main`

A.3.2 Javno dostopne knjižnice

Veliko število uporabnih programov je uporabnikom na razpolago tudi preko javno dostopnih knjižnic. Za razliko od programov iz komercialnih knjižnic so javno dostopni programi mnogo bolj neizenačene kvalitete in zanesljivosti ter neenotno dokumentirani. Dokaj zanesljivo informacijo o javno dostopnih programih lahko dobimo preko kazala NIST, ki se nahaja na naslovu

www: `http://gams.nist.gov/`

el. pošta: gams@nist.gov

Največja zbirka javno dostopnih programov in knjižnic je NETLIB. Dostopna je preko naslovov

www: `http://www.netlib.org/`

www: `http://www.netlib.no/`

el. pošta: netlib_maintainers@netlib.org.

Najbolj znane knjižnice, ki se nahajajo v zbirki NETLIB so

BLAS (Basic Linear Algebra Subroutines) je temelj, na katerem je zgrajena večina programov v tej zbirki. Programi iz knjižnice BLAS omogočajo lažjo prenosljivost programov med različnimi tipi računalnikov.

LAPACK je knjižnica progamov v Fortranu 77 za reševanje problemov linearne algebre (reševanje sistemov linearnih enačb, računanje lastnih vrednosti in lastnih vektorjev matrik, ...

CLAPACK je varianta knjižnice LAPACK, napisana v jeziku C.

LAPACK++ je varianta knjižnice LAPACK, napisana v jeziku C++.

ScaLAPACK je varianta knjižnice LAPACK za večprocersorske računalnike z razdeljenim pomnilnikom.

Dodatek B

Kompleksni vektorji in matrike

Računanju s kompleksnimi števili se velikokrat ne moremo izogniti. Tudi kadar so podatki realni, se lahko zgodi, da so rezultati kompleksna števila, kot na primer pri preprosti kvadratni enačbi $x^2+1=0$, ki nima realnih ničel, zato pa ima dve kompleksni konjugirani rešitvi $x_1 = i$ in $x_2 = -i$. Podobno so tudi lastne vrednosti realne matrike lahko kompleksne, na primer lastni vrednosti matrike

$$A = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}$$

sta i in $-i$, prav tako sta kompleksna tudi lastna vektorja.

Ker posplošitev realnih vektorjev in matrik na kompleksne vektorje in matrike zahteva nekaj pozornosti, si bomo ogledali, kako lahko operacije nad realnimi vektorji in matrikami posplošimo na kompleksne vektorje in matrike tako, da bomo ohranili konsistentnost operacij.

B.1 Kompleksna števila

Množico kompleksnih števil $\mathbb{C}$ dobimo tako, da množici realnih števil dodamo rešitev kvadratne enačbe $x^2 + 1 = 0$, ki jo označimo z i. Če poleg tega zahtevamo, da naj bo množica $\mathbb{C}$ zaprta za seštevanje in množenje z realnim številom, so kompleksna števila vsa števila, ki jih lahko zapišemo kot $z = x + iy$, kjer sta $x, y \in \mathbb{R}$. Pri tem je $x = \mathrm{Re}(z)$ *realna komponenta* in $y = \mathrm{Im}(z)$ *imaginarna komponenta* števila z.

Operacije s kompleksnimi števili V množici kompleksnih števil imamo tako definirano seštevanje in množenje: če sta $z = x + iy$ in $w = u + iv$ dve kompleksni števili, potem je

$$z + w = (x + u) + i(y + v)$$

vsota števil z in w,

$$zw = (xu - yv) + i(xv + yu)$$

pa njun *produkt*.

Poleg dvomestnih operacij v množici kompleksnih števil definiramo tudi enomestno operacijo *konjugiranje*: številu $z = x + iy$ konjugirano število je $\bar{z} = x - iy$. Število z in njemu konjugirano število $\bar{z}$ se razlikujeta le v predznaku imaginarne komponente.

S pomočjo konjugiranja lahko med kompleksnimi števili definiramo tudi deljenje: *kvocient* kompleksnih števil z in w je

$$\frac{z}{w} = \frac{z\bar{w}}{w\bar{w}} = \frac{(x+iy)(u-iv)}{(u+iv)(u-iv)} = \frac{xu+yv}{u^2+v^2} + i\frac{-xv+yu}{u^2+v^2}\,.$$

Bralec lahko enostavno preveri, da je tako definirano deljenje nasprotna operacija množenja.

Za konjugiranje kompleksnih števil veljajo naslednja pravila:

Izrek B.1.

1. $\bar{\bar{z}} = z$;
2. $\overline{z + w} = \bar{z} + \bar{w}$
3. $\overline{zw} = \bar{z}\bar{w}$
4. $\overline{z/w} = \bar{z}/\bar{w}$

Dokazi teh lastnosti so enostavni, prepuščamo jih bralcu.

Vsota kompleksnega števila in njegove konjugirane vrednosti je vedno realno število

$$z + \bar{z} = (x + iy) + (x - iy) = 2x\,,$$

njuna razlika pa čisto imaginarno število

$$z - \bar{z} = (x + iy) - (x - iy) = 2iy\,.$$

Produkt kompleksnega števila z in njegove konjugirane vrednosti $\bar{z}$ je vedno pozitivno realno število

$$z\bar{z} = (x + iy)(x - iy) = x^2 + y^2 \,,$$

zato je smiselna naslednja definicija.

Definicija B.1. Absolutna vrednost *kompleksnega števila je* $|z| = \sqrt{z\bar{z}}$.

Če je z realno število ($y = 0$), potem se je njegova absolutna vrednost po definiciji B.1 ujema z običajno definicijo absolutne vrednosti realnega števila.

Polarna oblika Namesto para realnih števil, kjer prvo pomeni realno komponento, drugo pa imaginarno, lahko kompleksno število lahko zapišemo tudi v polarni obliki, kot par realnih števil, kjer prvo pomeni absolutno vrednost števila, drugo pa *polarni kot*. Tako je

$$z = x + iy = |z|(\cos\varphi + i\sin\varphi)\,.$$

Zveza med zapisom s komponentama $z = x + iy$ in polarnim zapisom $z = |z|(\cos\varphi + i\sin\varphi)$ je določena z

$$x = |z|\cos\varphi, \qquad y = |z|\sin\varphi\,,$$

obratna povezava pa z

$$\cos\varphi = \frac{x}{\sqrt{x^2+y^2}}, \qquad \sin\varphi = \frac{y}{\sqrt{x^2+y^2}}\,.$$

Kompleksno število v polarni obliki lahko s pomočjo *Eulerjeve identitete*

$$e^{i\varphi} = \cos\varphi + i\sin\varphi \tag{B.1}$$

zapišemo tudi enostavneje kot $z = |z|e^{i\varphi}$.

Kompleksnemu številu $z = |z|(\cos\varphi + i\sin\varphi)$ konjugirano število je $\bar{z} = |z|(\cos\varphi - i\sin\varphi) = |z|(\cos(-\varphi) + i\sin(-\varphi))$. S pomočjo o zapišemo enostavneje

$$z = |z|e^{i\varphi}, \qquad \bar{z} = |z|e^{-i\varphi}\,.$$

Produkt dveh kompleksnih števil v polarni obliki

$$z = |z|(\cos\varphi + i\sin\varphi) \quad \text{in} \quad w = |w|(\cos\psi + i\sin\psi)$$

je enak

$$\begin{aligned} zw &= |z|\cdot|w|\cdot[(\cos\varphi\cos\psi - \sin\varphi\sin\psi) \\ &\quad +i(\sin\varphi\cos\psi + \cos\varphi\sin\psi)] \\ &= |z|\cdot|w|\cdot[\cos(\varphi+\psi) + i\sin(\varphi+\psi)]. \end{aligned}$$

Enostavneje to zapišemo z uporabo Eulerjeve identitete kot

$$zw = (|z|e^{i\varphi})(|w|e^{i\psi}) = (|z|\cdot|w|)e^{i(\varphi+\psi)}\,. \tag{B.2}$$

Pri množenju se torej absolutne vrednosti množijo, polarni koti pa seštevajo.

Kvocient dveh kompleksnih števil z in $w \neq 0$ pa je

$$\frac{z}{w} = \frac{|z|}{|w|}[\cos(\varphi-\psi) + i\sin(\varphi-\psi)],$$

oziroma z uporabo Eulerjeve identitete

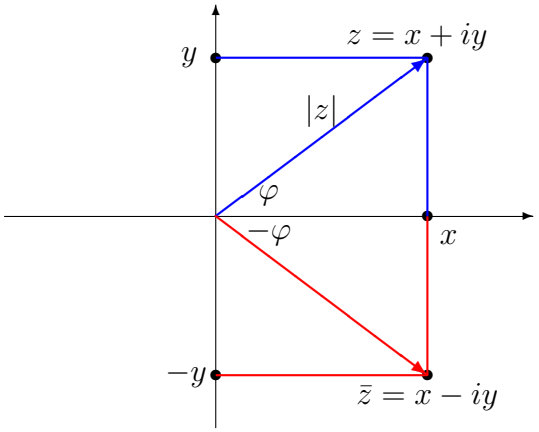
$$\frac{z}{w} = (|z|e^{i\varphi})/(|w|e^{i\psi}) = (|z|/|w|)e^{i(\varphi-\psi)}\,.$$

Kvocient kompleksnih števil ima absolutno vrednost $|z/w| = |z|/|w|$ in polarni kot $\varphi - \psi$.

Kompleksna ravnina Kompleksna števila ustrezajo točkam v koordinatni ravnini. Realno komponento narišemo na abscisni osi, imaginarno na ordinatni. Kompleksnemu številu $z = x + iy$ ustreza točka s koordinatama (x, y).

Seštevanje kompleksnih števil je podobno seštevanju vektorjev: vsoti kompleksnih števil $z = x + iy$ in $w = u + iv$ ustreza točka, katere krajevni vektor je vsota krajevnih vektorjev točk, ki pripadata številoma z in w.

Kompleksnemu številu se pri konjugiranju spremeni predznak imaginarne komponente, zato točka, ki ustreza konjugiranemu številu leži zrcalno (glede na abscisno os) točki, ki ustreza prvotnemu številu.



Slika B.1: Kompleksnemu številu $z = x + iy$ ustreza točka (x, y), številu $\bar{z} = x - iy$ pa točka $(x, -y)$.

Potence in koreni Če pravilo za produkt (B.2) uporabimo za n enakih faktorjev, dobimo

$$z^n = |z|^n(\cos(n\varphi) + i\sin(n\varphi)) = |z|^n e^{in\varphi}, \tag{B.3}$$

znano pravilo za potenciranje kompleksnih števil, imenovano tudi *de Moivreov obrazec* [1]

Če števili $p \in \mathbb{Z}$ in $q \in \mathbb{N}$ nimata skupnega delitelja potem lahko definiramo $z^{p/q}$ s predpisom

$$w = z^{p/q} \quad \Longleftrightarrow \quad w^q = z^p. \tag{B.4}$$

Števili w in z zapišimo v polarni obliki: $w = \rho(\cos\psi + i\sin\psi)$ in $z = r(\cos\varphi + i\sin\varphi)$. Iz pogoja (B.4) in de Moivrejevega obrazca sledi

$$\rho^q[\cos(q\psi) + i\sin(q\psi)] = r^p[\cos(p\varphi) + i\sin(p\varphi)].$$

Število na levi je enako številu na desni, če je $\rho^q = r^p$ in $q\psi - p\varphi = 2k\pi$, od koder dobimo

$$\rho = r^{p/q} \quad \text{in} \quad \psi = \frac{p\varphi + 2k\pi}{q}.$$

[1] Abraham de Moivre (1667 (Vitry-le-François, Francija)–1754 (London, Anglija)), Francoski matematik. Poleg formule, ki povezuje potence kompleksnih števil s trigonometričnimi funkcijami in nosi njegovo ime, se je ukvarjal predvsem s teorijo verjetnosti. Bil je prijatelj G. Galileia in I. Newtona.

Izraz $z^{p/q}$ ni enolično določen, saj za vsak $k = 0, 1, \ldots, q-1$ dobimo različno vrednost

$$w_k = z^{p/q} = r^{p/q} \left[\cos \frac{p\varphi + 2k\pi}{q} + i \sin \frac{p\varphi + 2k\pi}{q} \right] . \tag{B.5}$$

V kompleksni ravnini ležijo koreni $w_0, w_1, \ldots, w_{q-1}$ na oglišcih pravilnega q-kotnika s središčem v točki 0, središčni kot med dvema sosednjima oglišcema je $2\pi/q$.

B.2 Kompleksni vektorji

V kompleksnem vektorskem prostoru so skalarji kompleksna števila. Če so $c_1, c_2, \ldots, c_m$ kompleksna števila in $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_m$ vektorji, potem je njihova linearna kombinacija

$$c_1\mathbf{x}_1 + c_2\mathbf{x}_2 + \cdots + c_m\mathbf{x}_m \, .$$

Kompleksna verzija vektorskega prostora $\mathbb{R}^n$ je kompleksni vektorski prostor $\mathbb{C}^n$, ki ga sestavljajo urejene n-terice kompleksnih števil. Vektorji iz prostora $\mathbb{C}^n$ so torej

$$\begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix} = \begin{bmatrix} a_1 + ib_1 \\ a_2 + ib_2 \\ \vdots \\ a_n + ib_n \end{bmatrix} ,$$

kjer so $a_1, \ldots, a_n$ ter $b_1, \ldots, b_n$ realna števila in $c_j = a_j + ib_j, \ j = 1, \ldots, n$.

Osnovni operaciji, seštevanje vektorjev in množenje vektorja s skalarjem je povsem podobno podobnima operacijama v realnem vektorskem prostoru.

Standardna baza prostora $\mathbb{C}^n$ je, podobno kot v realnem primeru, sestavljena iz vektorjev

$$\mathbf{e}_1 = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} , \quad \mathbf{e}_2 = \begin{bmatrix} 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix} , \quad \cdots , \quad \mathbf{e}_n = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix} ,$$

kar pomeni, da je dimenzija prostora $\mathbb{C}^n$ enaka n.

Skalarni produkt Kompleksni produkt vektorjev $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ navadno definiramo kot $\mathbf{x}^T\mathbf{y}$. Tako definiran skalarni produkt je *komutativen* ($\mathbf{x}^T\mathbf{y} = \mathbf{y}^T\mathbf{x}$), *linearen* za prvi faktor ($(a_1\mathbf{x}_1 + a_2\mathbf{x}_2)^T\mathbf{y} = a_1(\mathbf{x}_1^T\mathbf{y}) + a_2(\mathbf{x}_2^T\mathbf{y})$, lineranost za drugi faktor je posledica komutativnosti) in *pozitivno definiten* ($\mathbf{x}^T\mathbf{x} \geq 0$ in $\mathbf{x}^T\mathbf{x} = 0$ natanko tedaj, ko je $\mathbf{x} = \mathbf{0}$).

Naivna posplošitev skalarnega produkta na kompleksni vektorski prostor sicer ohranja prvi dve lastnosti, ne pa pozitivne definitnosti, ki se ji nočemo odpovedati, ker je pomembna zaradi definicije dolžine vektorja.

V kompleksnih vektorskih prostorih moramo najprej običajno transpozicijo vektorjev nadomestiti s konjugirano transpozicijo:

Definicija B.2. Konjugirana transpozicija *vektorja* $\mathbf{x} \in \mathbb{C}^n$ *je*

$$\mathbf{x}^H = (\bar{\mathbf{x}})^T = [\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_n]\,.$$

S pomočjo konjugirane transpozicije lahko definiramo skalarni produkt v kompleksnih vektorskih prostorih:

Definicija B.3. Skalarni produkt *vektorjev* $\mathbf{x}$ *in* $\mathbf{y}$ *iz* $\mathbb{C}^n$ *je* $\mathbf{x}^H\mathbf{y}$*:*

$$\mathbf{x}^H\mathbf{y} = [\bar{x}_1 \ldots\ \bar{x}_n] \begin{bmatrix} y_1 \\ \vdots \\ v_n \end{bmatrix} = \bar{x}_1 y_1 + \cdots + \bar{x}_n y_n\,.$$

Lastnost 5. *Za kompleksne vektorje* $\mathbf{x}, \mathbf{y}$ *in* $\mathbf{z}$ *ter kompleksna skalarja* α *in* β *veljajo naslednje lastnosti:*

1. $\mathbf{x}^H\mathbf{y} = \overline{\mathbf{y}^H\mathbf{x}}$*;*
2. $(\alpha\mathbf{x} + \beta\mathbf{y})^H\mathbf{z} = \bar{\alpha}(\mathbf{x}^H\mathbf{z}) + \bar{\beta}(\mathbf{y}^H\mathbf{z})$*;*
3. $\mathbf{x}^H(\alpha\mathbf{y} + \beta\mathbf{z}) = \alpha(\mathbf{x}^H\mathbf{y}) + \beta(\mathbf{x}^H\mathbf{z})$*;*
4. $\mathbf{x}^H\mathbf{x} \geq 0$ *in* $\mathbf{x}^H\mathbf{x} = 0$ *samo kadar je* $\mathbf{x} = \mathbf{0}$*.*

Dokaz: Dokaze teh lastnosti prepuščamo bralcu.

Dva kompleksna vektorja $\mathbf{x}$ in $\mathbf{y}$ sta *ortogonalna*, kadar je njun skalarni produkt $\mathbf{x}^H\mathbf{y}$ enak nič. *Kot* φ med vektorjema lahko določimo s pomočjo formule

$$\mathrm{Re}(\mathbf{x}^H\mathbf{y}) = ||\mathbf{x}|| \cdot ||\mathbf{y}|| \cos\varphi\,.$$

Pozor! V nekaterih učbenikih je kompleksni skalarni produkt definiran kot $\bar{\mathbf{x}}\mathbf{y}^T$. V tem primeru se nekoliko spremenita druga in tretja lastnost skalarnega produkta. Uporabljamo lahko eno ali drugo definicijo, pod pogojem, da se je striktno držimo.
V primeru, da sta vektorja $\mathbf{x}$ in $\mathbf{y}$ realna, sta obe definiciji ekvivalentni običajni definiciji skalarnega produkta.

S pomočjo skalarnega produkta sedaj lahko (podobno kot v realnem vektorskem prostoru) definiramo dolžino vektorja:

Definicija B.4. Dolžina *vektorja* $\mathbf{x}$ *je*

$$||\mathbf{x}|| = \sqrt{\mathbf{x}^H\mathbf{x}} = \sqrt{|x_1|^2 + \cdots + |x_n|^2}$$

Dolžina vektorja ima naslednje lastnosti:

Lastnost 6.

1. *Pozitivna definitnost:* $||\mathbf{x}|| \geq 0$ *in* $||\mathbf{x}|| = 0$ *natanko tedaj, ko je* $\mathbf{x} = \mathbf{0}$.
2. *Homogenost:* $\lambda\mathbf{x} = |\lambda| \cdot ||\mathbf{x}||$.
3. *Trikotniška neenakost:* $||\mathbf{x} + \mathbf{y}|| \leq ||\mathbf{x}|| + ||\mathbf{y}||$.

Dokaz: Tudi ta dokaz prepuščamo bralcu.

S pomočjo skalarnega produkta lahko definiramo tudi razdaljo med dvema vektorjema:

Definicija B.5. Evklidska razdalja *med vektorjema* $\mathbf{x}$ *in* $\mathbf{y}$ *je*

$$d(\mathbf{x},\mathbf{y}) = ||\mathbf{x}-\mathbf{y}|| = \sqrt{|x_1-y_1|^2+\cdots+|x_n-y_n|^2}\,.$$

Razdalja med dvema vektorjema ima naslednje lastnosti:

Lastnost 7.

1. *Pozitivna definitnost:* $d(\mathbf{x},\mathbf{y}) \geq 0$ *in* $d(\mathbf{x},\mathbf{y}) = 0$ *natanko tedaj, ko je* $\mathbf{x} = \mathbf{y}$.
2. *Simetričnost:* $d(\mathbf{x},\mathbf{y}) = d(\mathbf{y},\mathbf{x})$.
3. *Homogenost:* $d(\lambda\mathbf{x},\lambda\mathbf{y}) = |\lambda| \cdot d(\mathbf{x},\mathbf{y})$.
4. *Trikotniška neenakost:* $d(\mathbf{x},\mathbf{y}) \leq d(\mathbf{x},\mathbf{z}) + d(\mathbf{z},\mathbf{y})$.

Dokaz: Lastnosti razdalje med vektorjema so preproste posledice lastnosti dolžine vektorja. Podrobnosti prepuščamo bralcu.

B.3 Kompleksne matrike

Tudi pri kompleksnih matrikah bomo, podobno kot pri kompleksnih vektorjih, namesto transponiranja raje uporabljali konjugirano transponiranje $A^H = \bar{A}^T$.

Konjugirano transponiranje je po lastnostih podobno običajnemu transponiranju matrik:

Lastnost 8.

1. $(A+B)^H = A^H + B^H$
2. $(\lambda A)^H = \bar{\lambda} A^H$
3. $(A^H)^H = A$
4. $(AB)^H = B^H A^H$

Dokaz: Tudi dokazi teh lastnosti so dovolj enostavni, da jih bo bralec zmogel z malo truda.

Hermitske matrike Podobno vlogo, kot jo imajo med realnimi matrikami simetrične matrike, imajo med kompleksnimi matrikami hermitske matrike:

Definicija B.6. *Matrika $A \in \mathbb{C}^{n\times n}$ je* hermitska*, če je $A^H = A$.*

Primer B.1. Vsaka realna simetrična matrika je tudi hermitska, saj konjugiranje na realna števila nima vpliva. Tudi matrika

$$A = \begin{bmatrix} 1 & 3+i \\ 3-i & -2 \end{bmatrix}$$

je hermitska, saj je $A^H = \bar{A}^T = A$.

Hermitske matrike imajo podobne lastnosti kot realne simetrične matrike.

Izrek B.2. *Če je matrika A hermitska, je število $\mathbf{x}^H A\mathbf{x}$ realno za vsak vektor $\mathbf{x} \in \mathbb{C}^n$.*

Dokaz: Izraz $\mathbf{x}^H A\mathbf{x}$ je očitno število (matrika velikosti 1×1). Če je matrika A hermitska, je zaradi lastnosti 8

$$(\mathbf{x}^H A\mathbf{x})^H = \mathbf{x}^H (A^H)^H (\mathbf{x}^H)^H = \mathbf{x}^H A\mathbf{x}$$

in zato realno.

Realne simetrične matrike imajo le realne lastne vrednosti. Za hermitske matrike velja podobno.

Izrek B.3. *Vse lastne vrednosti hermitske matrike so realne.*

Dokaz: Naj bo A hermitska matrika in $\mathbf{x}$ njen lastni vektor, ki pripada lastni vrednosti λ, torej $A\mathbf{x} = \lambda\mathbf{x}$. Obe strani te enačbe pomnožimo z leve s $\mathbf{x}^H$ in dobimo $\mathbf{x}^H A\mathbf{x} = \lambda\mathbf{x}^H\mathbf{x}$. Ker je A hermitska, je na levi strani $\mathbf{x}^H A\mathbf{x}$ realno število. Na desni je $\mathbf{x}^H\mathbf{x}$ kvadrat dolžine vektorja $\mathbf{x}$, ki je tudi realno število, zato mora biti tudi λ realen.

Pri simetričnih matrikah so lastni vektorji, ki pripadajo različnim lastnim vrednostim med seboj ortogonalni. Podobno je pri hermitskih matrikah.

Izrek B.4. *Lastni vektorji, ki pripadajo različnim lastnim vrednostim hermitske matrike, so med seboj ortogonalni. Če je*

$$A\mathbf{x} = \lambda\mathbf{x}, \quad A\mathbf{y} = \mu\mathbf{y} \quad \textit{in} \quad \lambda \neq \mu\,,$$

potem je $\mathbf{x}^H\mathbf{y} = 0$.

Dokaz: Enačbo $A\mathbf{x} = \lambda\mathbf{x}$ z leve pomnožimo z $\mathbf{y}^H$, enačbo $A\mathbf{y} = \mu\mathbf{y}$ pa najprej konjugiramo in transponiramo $\mathbf{y}^H A^H = \mu\mathbf{y}^H$, potem pa z desne pomnožimo z $\mathbf{x}$. Tako dobimo dve enačbi

$$\mathbf{y}^H A\mathbf{x} = \lambda\mathbf{y}^H\mathbf{x} \quad \text{in} \quad \mathbf{y}^H A^H\mathbf{x} = \mu\mathbf{y}^H\mathbf{x}\,.$$

Enačbi odštejemo in upoštevamo, da sta levi strani enaki (ker je A hermitska)

$$0 = (\lambda - \mu)\mathbf{y}^H\mathbf{x}\,.$$

Ker sta λ in μ različni lastni vrednosti, mora biti produkt $\mathbf{y}^H\mathbf{x}$ enak 0, kar pomeni, da sta vektorja $\mathbf{x}$ in $\mathbf{y}$ ortogonalna.

Unitarne matrike Med realnimi matrikami imajo posebno vlogo ortogonalne matrike, to so matrike Q, za katere velja $Q^TQ = I$. Podobno vlogo imajo med kompleksnimi matrikami unitarne matrike.

Definicija B.7. *Kompleksna kvadratna matrika U je* unitarna*, kadar je* $U^HU = I$.

Primer B.2. Vsaka realna ortogonalna matrika je unitarna, saj je za realne matrike $A^T = A^H$. Tudi matrika

$$A = \frac{1}{\sqrt{3}} \begin{bmatrix} 1 & 1+i \\ 1-i & -1 \end{bmatrix}$$

je unitarna, ker je

$$A^H A = \frac{1}{3} \begin{bmatrix} 1 & 1+i \\ 1-i & -1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 1+i \\ 1-i & -1 \end{bmatrix} = \frac{1}{3} \begin{bmatrix} 3 & 0 \\ 0 & 3 \end{bmatrix} .$$

Seveda so vse unitarne matrike, prav tako kot ortogonalne, obrnljive. Inverz unitarne matrike A je A^H.

Izrek B.5. *Množenje z unitarno matriko ohranja dolžine vektorjev in kote med njimi. Če je U unitarna matrika dimenzije n, potem je*

$$||U\mathbf{x}|| = ||\mathbf{x}|| \quad \textit{za vsak} \quad \mathbf{x} \in \mathbb{C}^n$$

in

$$(U\mathbf{x})^H U\mathbf{y} = \mathbf{x}^H \mathbf{y} \quad \textit{za vsak} \quad \mathbf{x}, \mathbf{y} \in \mathbb{C}^n .$$

Dokaz: Za unitarno matriko U velja

$$||U\mathbf{x}|| = \sqrt{(U\mathbf{x})^H U\mathbf{x}} = \sqrt{\mathbf{x}^H U^H U\mathbf{x}} = \sqrt{\mathbf{x}^H \mathbf{x}} = ||\mathbf{x}||$$

in

$$(U\mathbf{x})^H U\mathbf{y} = \mathbf{x}^H U^H U\mathbf{y} = \mathbf{x}^H \mathbf{y} .$$

Preprosta posledica te lastnosti je

Posledica B.6. *Vse lastne vrednosti unitarne matrike imajo absolutno vrednost enako* 1.

Dokaz: Če je matrika U unitarna, je (zaradi izreka B.5) za lastni vektor $\mathbf{x}$ in lastno vrednost λ po eni strani

$$||U\mathbf{x}|| = ||\mathbf{x}||\,,$$

po drugi strani pa

$$||U\mathbf{x}|| = ||\lambda\mathbf{x}|| = |\lambda| \cdot ||\mathbf{x}||\,,$$

torej mora biti $|\lambda| = 1$.

Še eno uporabno lastnost imajo unitarne matrike:

Izrek B.7. *Produkt dveh unitarnih matrik je unitarna matrika.*

Dokaz: Če sta U in S unitarni matriki, potem je

$$(SU)^H SU = U^H S^H SU = U^H U = I\,,$$

torej je tudi SU*unitarna.*

Zanimiv je tudi *Schurov razcep* poljubne kvadratne matrike:

Izrek B.8. Schurov izrek *Za poljubno matriko $A \in \mathbb{C}^{n\times n}$ obstajata unitarna matrika U in zgornjetrikotna matrika T, da je $U^H AU = T$. Diagonalni elementi matrike T so lastne vrednosti matrike A.*

Izrek navajamo brez dokaza. Radovedni bralec si ga lahko ogleda v [9].

Schurov izrek zagotavlja, da lahko vsako kvadratno matriko razcepimo na produkt unitarne matrike U, zgornjetrikotne matrike T in inverzne matrike U^H. Za nekatere matrike pa lahko namesto zgornjetrikotne matrike T dobimo diagonalno matriko Λ, ki ima na diagonali lastne vrednosti matrike A.

Definicija B.8. *Kvadratna matrika A je* normalna*, če je $AA^H = A^H A$.*

Normalne matrike so natančno tiste, ki se dajo diagonalizirati z unitarno matriko:

Izrek B.9. *Matriko A lahko diagonaliziramo z unitarno matriko natanko tedaj, ko je A normalna. Natančneje:*

1. *Če je A normalna, obstaja tala unitarna matrika U da je $U^H AU$ diagonalna.*
2. *Kadar obstaja tala unitarna matrika U, da je $U^H AU$ diagonalna, je A normalna.*

Tudi ta izrek navajamo brez dokaza. Bralec ga lahko poišče v [11].

Ker so hermitske matrike ($A^H = A$) in unitarne matrike ($A^H = A^{-1}$) tudi normalne, lahko vse hermitske in unitarne matrike diagonaliziramo.

Posledica B.10. *Če so λ_i, $i = 1, \ldots, n$ lastne vrednosti in $\mathbf{x}_i$, $i = 1, \ldots, n$ ortonormirani lastni vektorji hermitske matrike U, potem lahko U zapišemo kot linerano kombinacijo matrik ranga* 1

$$U = \sum_{i=1}^{n} \lambda_i \mathbf{x}_i \mathbf{x}_i^H .$$

Temu zapisu navadno pravimo *spektralna reprezentacija* ali *spektralna predstavitev* hermitske matrike.

Literatura

[1] T. J. Akai: *Numerical Methods*, John Wiley & Sons, Inc., New York 1994.

[2] K. Atkinson: *Elementary Numerical Analysis*, John Wiley & Sons, Inc., New York, 1985.

[3] Z. Bohte: *Numerične metode*, DMFA Slovenije, Ljubljana 1991.

[4] Z. Bohte: *Numerično reševanje nelinearnih enačb*, DMFA, Ljubljana 1993.

[5] Z. Bohte: *Numerično reševanje sistemov linearnih enačb*, DMFA Slovenije, Ljubljana 1994.

[6] Z. Bohte: *Uvod v numerično računanje*, DMFA Slovenije, Ljubljana 1995.

[7] S. D. Conte, C. de Boor: *Elementary Numerical Analysis - An Algoritmic Approach*, McGraw-Hill, New York 1980.

[8] B. N. Datta:*Numerical Linear Algebra and Applications*, Brooks/Cole Publishing Co., Pacific Grove 1995

[9] J. W. Demmel: *Uporabna numericna linearna algebra*, DMFA, Ljubljana 2000.

[10] J. F. Epperson:*Numerical Methods and Analysis*, John Wiley & Sons, Inc. New York 2002.

[11] G. H. Golub, C.F. van Loan: *Matrix Computations*, The John Hopkins University Press, Baltimore 1989.

[12] E. Hairer, S. P. Nørsett, G. Wanner: *Solving Ordinary Differential Equations I: Nonstiff Problems* (2nd ed.), Springer-Verlag, Berlin 1991.

[13] P. Henrici: *Discrete Variable Methods in Ordinary Differential Equations*, John Wiley & Sons, Inc, New York 1962.

[14] A. Iserles: *A First Course in the Numerical Analysis of Differential Equations*, Cambridge University Press, Cambridge 1996.

[15] J. Kozak: *Numerična analiza*, DMFA založništvo, Ljubljana 1994.

[16] G. W. Stewart: *Afternotes on Numerical Analysis*, SIAM, Philadelphia 1996.

[17] Vetterling, W.T. and Press, W.H. and Teukolsky, S.A. and Flannery, B.P.: *Numerical Recipes 3rd Edition: The Art of Scientific Computing*, Cambridge University Press, Cambridge 1996.