Bor Plestenjak

Uvod v numerične metode (matematika)

delovna verzija

verzija: 5. oktober 2012

Kazalo

Poglavje 1

Uvod

1.1 Numerična matematika

Numerična matematika se ukvarja z razvojem in analizo metod za numerično reševanje matematičnih problemov. Za razliko od analitičnega, pri numeričnem reševanju iščemo rešitev v numerični obliki, v prav takšni obliki pa dobimo tudi začetne podatke. To npr. pomeni, da numerična metoda kot rešitev enačbe $x^2 = 3$ ne bo vrnila $\sqrt{3}$, temveč $1.73205\ldots$, pri čemer je število decimalk odvisno od natančnosti, v kateri računamo.

Numerična metoda je postopek, s katerim iz začetnih numeričnih podatkov s končnim zaporedjem elementarnih operacij izračunamo numerični približek za rešitev določenega matematičnega problema. *Elementarne operacije* so odvisne od okolja v katerem računamo, mi bomo kot elementarne operacije šteli seštevanje, odštevanje, množenje, deljenje in kvadratni koren. Za numerično metodo pravimo, da je *direktna*, kadar neodvisno od začetnih podatkov za rešitev porabimo konstantno število elementarnih operacij. Druga možnost so *iterativne metode*, kjer rešitev dobimo kot limito nekega konvergentnega zaporedja, z računanjem pa prenehamo, ko ocenimo, da je približek dovolj natančen.

Pri reševanju matematičnih problemov numerične metode uporabljamo takrat, ko do rešitve ne moremo priti analitično. Nekaj primerov je:

- Iskanje ničel polinoma pete stopnje, npr. reševanje enačbe $x^5 + 3x - 1 = 0$. Dokazano[1] za polinome stopnje pet ali več ne obstajajo končne analitične formule, s katerimi bi lahko ničle izračunali iz koeficientov polinoma, kot to npr. lahko naredimo pri kvadratni enačbi $ax^2 + bx + c = 0$. Zato za splošen polinom stopnje pet ali več nimamo druge možnosti, kot da ničle izračunamo numerično. To velja tudi za transcendentne enačbe, kot je npr. $x + \ln x = 0$, in sploh za večino nelinearnih enačb in sistemov, s katerimi se srečamo pri praktičnih problemih.
- Računanje lastnih vrednosti in vektorjev matrik. Po eni strani vemo, da so lastne vrednosti ničle karakterističnega polinoma, po drugi strani pa za vsak polinom obstaja matrika, katere karakteristični polinom se do predznaka ujema z izbranim polinomom. Zaradi

[1] To je znano kot Abel–Ruffinijev izrek, ki ga je prvi v celoti dokazal norveški matematik Niels Henrik Abel (1802–1829) leta 1824. Abelova nagrada, ki jo od leta 2002 (200-letnice Abelovega rojstva) norveški kralj vsako leto podeli izjemnim matematikom, je vredna toliko kot Nobelova nagrada, ki se sicer za področje matematike ne podeljuje.

tega se lastnih vrednosti ne da izračunati drugače kot z iterativnim postopkom, saj bi vsak končni proces pomenil protislovje, da lahko ničle poljubnega polinoma izrazimo iz njegovih koeficientov s končnimi formulami.

- Računanje določenega integrala, npr. $\int_0^1 e^{x^2} dx$. V tem primeru, kot velja tudi za večino integralov, ki nastopajo v praktičnih problemih, se nedoločenega integrala ne da zapisati analitično, zato lahko integral izračunamo le numerično. Podobno velja tudi za reševanje diferencialnih enačb.

Velikokrat pa so numerične metode udobnejše tudi za reševanje problemov, za katere sicer obstajajo tudi analitične rešitve, kar kažeta naslednja zgleda:

- Računanje inverzne matrike velikosti 100×100. Če ima npr. matrika A celoštevilske elemente, potem ima inverzna matrika A^{-1} racionalne elemente. Števci in imenovalci teh ulomkov so lahko zelo veliki in za računanje s takšnimi ulomki lahko pri eksaktnem računanju porabimo veliko časa in prostora. Če računamo numerično, vsako število zavzame enako veliko pomnilnika.
- Cardanove formule[2] za ničle kubičnega polinoma. Kljub temu, da obstajajo analitične formule za izračun ničel kubičnega polinoma, jih ni enostavno uporabljati, saj je med drugim v enem izmed korakov potrebno izračunati tretji koren kompleksnega števila. Enostavneje je, če uporabimo kar splošni numerični algoritem za računanje ničel polinomov.

Glavni problemi, ki jih bomo obravnavali, so:

- *nelinearne enačbe*: poišči eno ali več ničel funkcije $f : \mathbb{R} \to \mathbb{R}$ oziroma $F : \mathbb{R}^n \to \mathbb{R}^n$;
- *linearni sistemi*: poišči $x \in \mathbb{R}^n$, ki reši sistem $Ax = b$ za nesingularno matriko $A \in \mathbb{R}^{n\times n}$ in $b \in \mathbb{R}^n$;
- *predoločeni sistemi*: poišči $x \in \mathbb{R}^n$, ki minimizira $\|Ax - b\|_2$ za matriko $A \in \mathbb{R}^{m\times n}$ in $b \in \mathbb{R}^m$, kjer je $m > n$;
- *lastne vrednosti*: izračunaj lastne vrednosti in vektorje dane matrike A;
- *interpolacija*: poišči polinom stopnje n, ki gre skozi točke $(x_0, y_0), \ldots, (x_n, y_n)$;
- *integriranje*: izračunaj integral $\int_a^b f(t)dt$;
- *diferencialne enačbe*: npr., reši začetni problem $y' = f(x, y)$, $y(x_0) = y_0$.

Numerična matematika se deli na *numerično linearno algebro* in *numerično analizo*. Prva se ukvarja s problemi, ki so povezani z linearno algebro in imajo v glavnem opravka s končnimi matrikami, druga pa s problemi, ki so povezani z analizo, kot so interpolacija, aproksimacija, odvajanje, integriranje in reševanje diferencialnih enačb. Področji nista strogo ločeni, saj se npr. pri numerični analizi reševanja parcialnih diferencialnih enačb ne moremo lotiti brez znanja o reševanju linearnih sistemov, po drugi strani pa pri numerični linearni algebri razne algoritme

[2] Vsestranski italijanski učenjak Gerolamo Cardano (1501–1576) jih je zapisal leta 1545. Cardano je znan tudi v mehaniki po kardanski gredi, poleg tega pa se je ukvarjal še z astronomijo, fiziko, astrologijo in medicino.

za reševanje linearnih sistemov s posebno strukturo razvijamo samo zato, ker jih potrebujemo pri reševanju parcialnih diferencialnih enačb.

En izmed glavnih pristopov pri numeričnem reševanju je, da dani problem prevedemo na lažjega, ki ima ali enako rešitev ali pa se rešitvi ne razlikujeta dosti. Zgledi za ta pristop so:

- Neskončne procese nadomestimo s končnimi procesi. Tako lahko npr. pri računanju vsote konvergentne vrste seštejemo le končno mnogo členov. Podobno, če je rešitev limita nekega neskončnega procesa, naredimo le končno korakov.
- Neskončno razsežne prostore nadomestimo s končno razsežnimi. Tako npr. pri reševanju funkcijske enačbe namesto splošne funkcije iščemo rešitev v prostoru polinomov omejene stopnje.
- Diferencialne enačbe nadomestimo z algebraičnimi enačbami. Tako npr. približek za rešitev robnega problema dobimo z reševanjem linearnega sistema.
- Nelinearni problem nadomestimo z linearnim. Tako npr. nelinearne sisteme rešujemo z Newtonovo metodo iterativno tako, da v vsakem koraku za popravek vzamemo rešitev linearnega sistema, ki aproksimira nelinearni sistem.
- Zapletene funkcije nadomestimo z enostavnejšimi, npr. s polinomi. Pri numeričnem integriranju ali odvajanju tako namesto podane funkcije integriramo oziroma odvajamo interpolacijski polinom. Groba ideja je, da če se interpolacijski polinom dobro ujema s funkcijo, bo podobno veljalo tudi za odvod in integral.
- Matrike nadomestimo z matrikami, ki imajo enostavnejšo obliko. Tako npr. pri reševanju linearnega sistema matriko spravimo v zgornjo trikotno obliko, pri iskanju lastnih vrednosti pa v zgornjo Hessenbergovo obliko. Pri tem pazimo, da transformacijo na enostavnejšo obliko izvedemo na čim stabilnejši način, po možnosti z ortogonalnimi transformacijami.

Glavne zahteve za dobro numerično metodo so:

- *Zanesljivost*. Metoda naj na enostavnih problemih vedno deluje pravilno. Kot bomo videli v nadaljevanju, obstajajo t.i. zelo občutljivi problemi, ki se jih, ne glede na to, kakšno numerično metodo uporabimo, ne da rešiti natančno. Pričakujemo pa, da bo metoda znala dovolj natančno rešiti neobčutljive probleme.
- *Robustnost*. Metoda naj običajno dela tudi na težjih problemih, kadar pa ne, naj to zazna in nas obvesti. Pogosto se namreč zgodi, da smo, ko nam neka metoda vrne numerični rezultat, slepo prepričani, da je rezultat zanesljiv, saj smo prej predhodno preverili, da je metoda natančno rešila nekaj lažjih primerov. Metoda, ki nas ne opozori, da je pri reševanju zašla v težave, nas lahko hitro zavede.
- *Natančnost*. Metoda naj izračuna rešitev tako natančno, kot je to možno glede na natančnost začetnih podatkov. Seveda ne smemo pričakovati, da bomo iz začetnih meritev, ki so natančne na dve decimalki, dobili na koncu deset točnih decimalk, če pa se zgodi obratno, pa tudi ni dobro.

- *Ekonomičnost*. Tu imamo na eni strani *časovno zahtevnost*, kjer želimo, da metoda porabi čim manj računskega časa, za kar je ponavadi dobro merilo število potrebnih elementarnih operacij. Na drugi strani imamo *prostorsko zahtevnost*, kjer želimo, da metoda porabi čim manj pomnilnika. Za obe zahtevnosti naj bi veljalo, da optimalna metoda ne porabi dosti več, kot je za tovrstni problem najmanj možno.
- *Uporabnost*. Metodo naj bo čim bolj vsestranska, da jo lahko uporabimo na širokem spektru problemov. Seveda pa obstajajo tudi dovolj pomembni in pogosti tipi problemov, za katere se splača univerzalne numerične metode prilagoditi in tako pridobiti pri ekonomičnosti in natančnosti.
- *Prijaznost do uporabnika*. Metoda naj bo dobro dokumentirana in naj ima enostaven uporabniški vmesnik. V končni fazi bodo z metodami večinoma računali uporabniki, ki ne vedo nujno veliko o numerični matematiki, želijo pa dobiti numerično rešitev nekega problema.

Numerične metode se stalno razvijajo. Zelo pomembno vlogo ima tehnologija, ki je na voljo za računanje. Algoritmi, ki so bili razviti za računanje s svinčnikom in listom papirja, niso nujno najprimernejši za uporabo na računalniku in obratno. Z razvojem računalnikov so nekateri algoritmi, ki so prej veljali za potratne in neuporabne, prišli do veljave. Ko so se pojavili paralelni računalniki, so postali pomembni algoritmi, ki lahko izkoristijo paralelnost. S povečevanjem računske moči je možno reševati vedno večje probleme in tako bodo lahko prišle v ospredje metode, za katere sicer teoretično velja, da so za velike primere bolj učinkovite, a so ti problemi sedaj še nepredstavljivo veliki.

Pri numeričnem računanju vedno izračunamo numerični približek za točno rešitev problema. Razlika med približkom in točno vrednostjo je napaka približka. Ločimo absolutno in relativno napako. Naj bo število x točna vrednost, $\widehat{x}$ pa približek za x. Potem:

- če je $\widehat{x} = x + d_a$, je d_a *absolutna napaka*,
- če je $\widehat{x} = x(1 + d_r)$ oziroma $d_r = \dfrac{\widehat{x} - x}{x}$, je d_r *relativna napaka*.

Majhna absolutna napaka pomeni, da moramo točni vrednosti prišteti število blizu 0, da dobimo približek, majhna relativna napaka pa pomeni, da približek dobimo tako, da točni rezultat pomnožimo s skalarjem, ki je blizu 1. Če je točna vrednost x enaka 0, potem je relativna napaka približka $\widehat{x} \neq 0$ neskončna.

1.2 Plavajoča vejica

Napake pri numeričnem računanju so med drugim tudi posledica tega, da v računalniku ne moremo predstaviti vseh realnih števil, temveč le končno mnogo. Zaradi tega vsa nepredstavljiva števila aproksimiramo z bližnjimi predstavljivimi števili, pri vsaki taki aproksimaciji pa pride do napake.

Števila v računalniku so zapisana v obliki *plavajoče vejice* kot

$$x = \pm m \cdot b^e, \tag{1.1}$$

kjer je $m = 0.c_1c_2, \ldots c_t$ *mantisa* in:

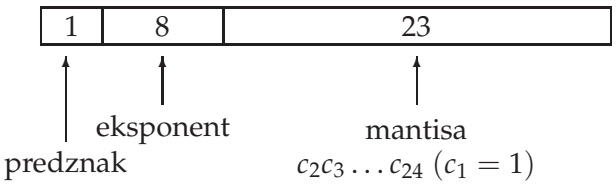
- b: *baza*, ponavadi je binarna ($b = 2$), lahko pa tudi desetiška ($b = 10$),
- t: *dolžina mantise*,
- e: *eksponent* v mejah $L \leq e \leq U$, kjer sta L in U spodnja in zgornja meja,
- c_i: *števke* v mejah od 0 do $b - 1$.

Števila so *normalizirana*, kar pomeni $c_1 \neq 0$. Na ta način po eni strani dosežemo enoličnost zapisov, saj bi sicer $0.1 \cdot b^e$ in $0.01 \cdot b^{e+1}$ bila različna zapisa istega števila; po drugi strani pa ohranjamo natančnost, saj poskrbimo, da ves čas uporabljamo celotno dolžino mantise, ki nam je na voljo. V nekaterih primerih pri najmanjšem eksponentu dopuščamo tudi t.i. *denormalizirana števila*, kjer je $c_1 = 0$.

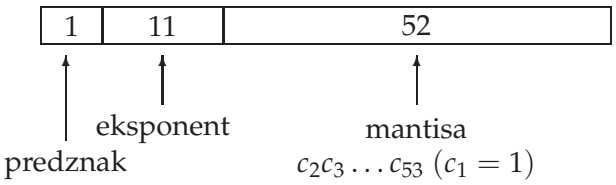
Zapis, ki je enolično določen s parametri b, t, L, U, označimo s $P(b, t, L, U)$.

Da bi poenotili računanje na različnih računalnikih, so leta 1985 predpisali pravila (gre za t.i. standard IEEE 754), ki se jih držijo vsi novejši računalniki. Najbolj znana zapisa po *standardu IEEE*[3] sta:

- *enojna natančnost* (single): $P(2, 24, -125, 128)$, število je v računalniškem spominu shranjeno v 32 bitih,



- *dvojna natančnost* (double): $P(2, 53, -1021, 1024)$, število je shranjeno v 64 bitih.



Če pogledamo razpon eksponentov pri enojni natančnosti, vidimo, da nismo porabili vseh 256 možnosti, ki jih dobimo iz zapisa eksponenta z osmimi biti, temveč le 254. Preostali vrednosti sta rezervirani za števila 0, ∞, $-\infty$ in nedoločeno vrednost NaN (Not-a-Number), ki jih ni mogoče prikazati v obliki (1.1).

Tako pri računanju v aritmetiki, ki podpira standard IEEE, dobimo npr. $1/0 = \infty$, $3/\infty = 0$ in $0/0 = \text{NaN}$. Medtem ko je rezultat vseh operacij, v katerih nastopa NaN, spet NaN, lahko

[3] IEEE oziroma Institute for Electrical and Electronics Engineers je mednarodno združenje inženirjev elektrotehnike in elektronike, ustanovljeno leta 1963, ki skrbi za številne standarde s teh področij.

iz vrednosti $\pm\infty$ z nadaljnjim računanjem spet pridemo do končnih vrednosti. V sistemih, ki podpirajo standard IEEE, se tako programi zaradi deljenja z 0 ali zaradi prekoračitve ne zaustavijo.

Standard IEEE dopušča tudi denormalizirana števila, kjer je $c_1 = 0$, eksponent pa je fiksen (v primeru enojne natančnosti je enak -125).

Če je s predznak, $0 \leq e \leq 255$ eksponent in $0 \leq f < 1$ število $f = 0.c_2 \ldots c_{24}$, potem so števila v enojni natančnosti zapisana v naslednji obliki.

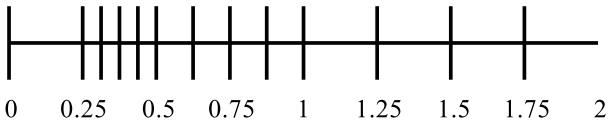
$0 < e < 255$	poljuben f	$x = (-1)^s(1+f) \cdot 2^{e-127}$
$e = 255$	$f = 0$	$x = (-1)^s\infty$
$e = 255$	$f \neq 0$	$x = \text{NaN}$
$e = 0$	$f = 0$	$x = (-1)^s 0$
$e = 0$	$f \neq 0$	$x = (-1)^s(0+f) \cdot 2^{-126}$

Podrobnejši zgledi zapisov števil v enojni natančnosti so v spodnji tabeli. Za predznak s velja, da je enak 1.

e	f	število
10000010	01100000000000000000000	$x = (1 + 2^{-2} + 2^{-3}) \cdot 2^{130-127} = 11$
11111111	00000000000000000000000	$x = \infty$
11111111	01011010100000000000000	$x = \text{NaN}$
00000000	00000000000000000000000	$x = 0$
00000000	00000100000000000000000	$x = 2^{-6} \cdot 2^{-126} = 2^{-132}$

Zgled 1.1 *Vsa pozitivna predstavljiva normalizirana števila iz množice $P(2,3,-1,1)$ so:*

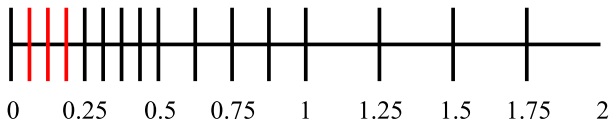
$$\begin{array}{lll} 0.100_2 \cdot 2^{-1} = 0.2500 & 0.100_2 \cdot 2^{0} = 0.500 & 0.100_2 \cdot 2^{1} = 1.00 \\ 0.101_2 \cdot 2^{-1} = 0.3125 & 0.101_2 \cdot 2^{0} = 0.625 & 0.101_2 \cdot 2^{1} = 1.25 \\ 0.110_2 \cdot 2^{-1} = 0.3750 & 0.110_2 \cdot 2^{0} = 0.750 & 0.110_2 \cdot 2^{1} = 1.50 \\ 0.111_2 \cdot 2^{-1} = 0.4375 & 0.111_2 \cdot 2^{0} = 0.875 & 0.111_2 \cdot 2^{1} = 1.75 \end{array}$$



Slika 1.1: Pozitivna predstavljiva normalizirana števila iz množice $P(2,3,-1,1)$.

Na sliki 1.1 lahko opazimo, da predstavljiva števila niso enakomerno razporejena po realni osi in da je med 0 in najmanjšim predstavljivim številom večja praznina. To praznino lahko malo zmanjšamo, če dopustimo še denormalizirana števila. V množici $P(2,3,-1,1)$ tako dobimo še naslednja pozitivna števila, ki so na sliki 1.2 označena rdeče:

$$\begin{array}{l} 0.011_2 \cdot 2^{-1} = 0.1875 \\ 0.010_2 \cdot 2^{-1} = 0.1250 \\ 0.001_2 \cdot 2^{-1} = 0.0625. \end{array}$$



Slika 1.2: Vsa pozitivna predstavljiva števila iz množice $P(2,3,-1,1)$.

□

Števila, ki niso predstavljiva, predstavimo s približki, ki jih dobimo z zaokrožanjem. To pomeni, da število x zapišemo v neskončnem zapisu kot $x = \pm 0.d_1 d_2 \ldots d_t d_{t+1} \ldots \cdot b^e$, potem pa se na podlagi d_{t+1} odločimo, ali zaokrožimo navzgor ali navzdol. Če je $d_{t+1} < b/2$, potem odrežemo števke od $(t+1)$-ve naprej in je $fl(x) = \pm 0.d_1 d_2 \ldots d_t \cdot b^e$, če pa je $d_{t+1} \geq b/2$, potem zaokrožimo navzgor in je $fl(x) = \pm(0.d_1 d_2 \ldots d_t + b^{-t}) \cdot b^e$.

Naj bo x tako število, da leži $|x|$ na intervalu med največjim in najmanjšim pozitivnim predstavljivim normaliziranim številom in naj bo $fl(x)$ njegovo najbližje predstavljivo število, dobljeno z zaokrožanjem. Z naslednjim izrekom bomo dokazali, da velja

$$fl(x) = x(1+\delta) \text{ za } |\delta| \leq u,$$

kjer je

$$u = \frac{1}{2}\, b^{1-t}$$

osnovna zaokrožitvena napaka, za katero velja:

- enojna natančnost: $u = 2^{-24} \approx 6 \cdot 10^{-8}$,
- dvojna natančnost: $u = 2^{-53} \approx 1 \cdot 10^{-16}$.

Izrek 1.1 *Če število x leži znotraj območja predstavljivih normaliziranih števil, potem velja*

$$\frac{|fl(x) - x|}{|x|} \leq \frac{u}{1+u}.$$

Dokaz. Naj bo $x = (y+z)b^e$, kjer je $y = 0.d_1 \ldots d_t$, $z = 0.0 \ldots 0 d_{t+1} d_{t+2} \ldots$ in naj bo $fl(x) = mb^e$, kjer je $m = 0.c_1 \ldots c_t$. Predpostavimo lahko, da je x pozitiven.

a) Če je $d_{t+1} < b/2$, odrežemo preostale števke in vzamemo $m = y$. Velja

$$\frac{|fl(x) - x|}{|x|} = \frac{z}{y+z} \leq \frac{\frac{1}{2}b^{-t}}{b^{-1} + \frac{1}{2}b^{-t}} = \frac{u}{1+u},$$

kjer smo pri oceni izbrali največji možni z in najmanjši možni y.

b) Če je $d_{t+1} \geq b/2$, zaokrožimo navzgor in vzamemo $m = y + b^{-t}$. Dobimo

$$\frac{|fl(x) - x|}{|x|} = \frac{b^{-t} - z}{y+z} \leq \frac{b^{-t} - \frac{1}{2}b^{-t}}{b^{-1} + \frac{1}{2}b^{-t}} = \frac{u}{1+u},$$

kjer smo pri oceni izbrali najmanjša možna z in y. ■

Standard IEEE zagotavlja, da za predstavljivi števili x in y velja:

- $fl(x \oplus y) = (x \oplus y)(1+\delta)$, kjer je $|\delta| \leq u$, za $\oplus = +, -, /, *$,
- za $x > 0$ je $fl(\sqrt{x}) = \sqrt{x}(1+\delta)$, kjer je $|\delta| \leq u$.

Če uporabimo osnovno operacijo na dveh predstavljivih številih, potem po standardu IEEE dobimo kot rezultat isto predstavljivo število, kot bi ga dobili, če bi operacijo izračunali eksaktno in zaokrožili rezultat. Izjema je, če pride do *prekoračitve* (overflow) ali *podkoračitve* (underflow) obsega predstavljivih števil. V tem primeru dobimo po standardu IEEE v primeru prekoračitve $\pm\infty$, v primeru podkoračitve pa 0.

1.3 Napake pri numeričnem računanju

Denimo, da želimo izračunati vrednost neke funkcije $f : \mathbb{R} \to \mathbb{R}$ pri danem x. Numerična metoda vrne približek $\widehat{y}$ za y, razlika $D = y - \widehat{y}$ pa je *celotna napaka* približka.

Izvor napake je lahko nenatančnost začetnih podatkov, napaka numerične metode ali pa zaokrožitvene napake med računanjem. Zato celotno napako razdelimo na tri dele.

Neodstranljiva napaka D_n

Ta napaka se pojavi zaradi nenatančnosti začetnih podatkov. Nastane zaradi napake meritev ali pa zaradi aproksimacije nepredstavljivih števil s predstavljivimi. Ker ne moremo zahtevati točnih podatkov, se neodstranljivi napaki ne moremo izogniti, od tod tudi izvira njeno ime. Prav tako je neodstranljiva napaka neodvisna od izbire numerične metode.

Namesto z x računamo s približkom $\overline{x}$, kar pomeni, da lahko, če v nadaljevanju ne pride do novih napak, namesto $y = f(x)$ v najboljšem primeru izračunamo $\overline{y} = f(\overline{x})$. Neodstranljiva napaka je

$$D_n = y - \overline{y}.$$

Če je funkcija f odvedljiva v točki x, potem je $|D_n| \approx |f'(x)||x - \overline{x}|$. Če poznamo oceno za absolutno vrednost odvoda, lahko tako ocenimo velikost neodstranljive napake.

Napaka metode D_m

Pogosto ni možno s končno mnogo osnovnimi operacijami numerično izračunati vrednosti funkcije f. Zaradi tega funkcijo aproksimiramo z drugo funkcijo g, katere vrednost znamo izračunati s končno mnogo osnovnimi operacijami. Primer je npr. numerično integriranje, kjer namesto podane funkcije integriramo interpolacijski polinom. Pogosto do napake metode pride tudi zato, ker pri računanju neskončen proces nadomestimo s končnim, npr. seštejemo le končno členov neskončne vrste ali pa prekinemo iterativno metodo po končnem številu korakov.

Vse skupaj pomeni, da namesto f računamo vrednost funkcije g, ki jo lahko izračunamo s končnim številom operacij. Namesto $\overline{y} = f(\overline{x})$ tako izračunamo $\widetilde{y} = g(\overline{x})$. Napaka metode je

$$D_m = \overline{y} - \widetilde{y}.$$

Aproksimacijsko funkcijo g po navadi izberemo tako, da poznamo tudi oceno za napako metode oziroma razliko $f(\overline{x}) - g(\overline{x})$. Običajno tudi v tej oceni nastopajo odvodi funkcije f.

Zaokrožitvena napaka D_z

Pri dejanskem računanju $\widetilde{y} = g(\overline{x})$ se pri vsaki računski operaciji pojavi zaokrožitvena napaka, zato v resnici namesto $\widetilde{y}$ izračunamo $\widehat{y}$. Sama vrednost $\widehat{y}$ je odvisna od vrstnega reda operacij in načina izračuna $g(\overline{x})$. Zaokrožitvena napaka je

$$D_z = \widetilde{y} - \widehat{y}.$$

Pri tem je potrebno poudariti, da ne velja nujno $\widehat{y} = fl(\widetilde{y})$, saj ne zaokrožujemo samo na koncu, temveč pri vsakem vmesnem koraku, ki je potreben za izračun $\widetilde{y} = g(\overline{x})$.

Celotna napaka

Končna napaka je $D = D_n + D_m + D_z$. Velja ocena

$$|D| \leq |D_n| + |D_m| + |D_z|,$$

zato nima smisla poskušati zmanjšati samo en člen, če druga dva ostaneta velika. Npr., če je neodstranljiva napaka velika, potem z metodo, ki ima majhno napako, ne pridobimo ničesar. Idealna situacija je, kadar so vse tri napake približno enakega velikostnega reda.

Zgled 1.2 *Izračunati želimo* $\sin(\pi/10)$*, na voljo pa imamo le kalkulator z osnovnimi štirimi operacijami, ki računa v plavajoči vejici z desetiško bazo in dolžino mantise 4.*

a) Neodstranljiva napaka. Namesto z $x = \pi/10$*, ki ni predstavljivo število, računamo z najbližjim predstavljivim številom* $\overline{x} = 0.3142 \cdot 10^0$*. Tako dobimo*

$$D_n = y - \overline{y} = \sin(\pi/10) - \sin(0.3142) = -3.9 \cdot 10^{-5}.$$

V praksi seveda ne poznamo točne vrednosti y. Če poznamo oceno velikosti odvoda funkcije f*, lahko uporabimo oceno* $|D_n| \approx |f'(x)||x - \overline{x}|$*. V našem primeru tako iz* $|f'(x)| \leq 1$ *in* $|x - \overline{x}| \leq 5 \cdot 10^{-5}$ *dobimo oceno* $|D_n| \leq 5 \cdot 10^{-5}$*, ki je zelo blizu dejanski napaki.*

b) Napaka metode. Ker ne znamo izračunati $f(x) = \sin x$*, za aproksimacijsko funkcijo vzamemo prva dva neničelna člena razvoja funkcije* f *v Taylorjevo vrsto, torej* $g(x) = x - x^3/6$*. Namesto* $\overline{y} = \sin(\overline{x})$ *tako izračunamo* $\widetilde{y} = g(\overline{x})$*. Napaka je*

$$D_m = \overline{y} - \widetilde{y} = 2.5 \cdot 10^{-5}.$$

Tudi napako metode v praksi lahko le ocenimo, saj ne poznamo točne vrednosti $f(\overline{x})$*. Ker Taylorjeva vrsta sinusne funkcije alternira, je v našem primeru napaka manjša od absolutne vrednosti naslednjega neničelnega člena. Tako dobimo oceno* $|D_m| \leq 2.6 \cdot 10^{-5}$ *ki se skoraj popolnoma ujema z dejansko napako.*

c) Zaokrožitvena napaka. Odvisna je od vrstnega reda in načina računanja $g(\overline{x})$. Denimo, da $\overline{x} - \overline{x}^3/6$ izračunamo po naslednjem postopku:

$$\begin{aligned} a_1 &= fl(\overline{x} * \overline{x}) = fl(0.09872164) = 0.9872 \cdot 10^{-1} \\ a_2 &= fl(a_1 * \overline{x}) = fl(0.03101154) = 0.3101 \cdot 10^{-1} \\ a_3 &= fl(a_2/6) = fl(0.0051683\ldots) = 0.5168 \cdot 10^{-2} \\ \widehat{y} &= fl(\overline{x} - a_3) = fl(0.309032) = 0.3090 \cdot 10^{0}. \end{aligned}$$

Ker je $\widetilde{y} = g(\overline{x}) = 0.3090302767\ldots$, je napaka

$$D_z = \widetilde{y} - \widehat{y} = 3.0 \cdot 10^{-5}.$$

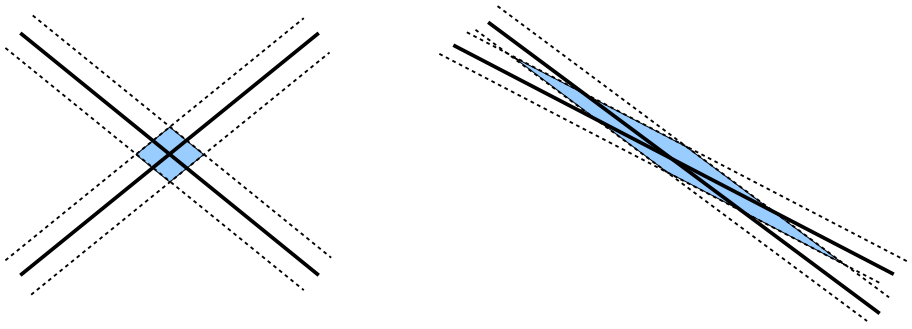
Tudi velikost zaokrožitvene napake se da oceniti, če po korakih analiziramo sam postopek izračuna $g(\overline{x})$. Nekaj primerov analize zaokrožitvenih vrednosti lahko najdete v razdelku 1.6.

Celotna napaka je $D = D_n + D_m + D_z = 1.6 \cdot 10^{-5}$. Vidimo, da se izračunana vrednost ujema s točno vrednostjo, zaokroženo na 4 decimalke. To pomeni, da smo dosegli optimalen rezultat, saj smo v dani množici predstavljivih števil dobili najboljši možen približek za vrednost $\sin(\pi/10)$. □

1.4 Občutljivost problema in stabilnost metode

Z analizo in ocenjevanjem velikosti neodstranljive napake se ukvarja *teorija motenj* (*perturbacij*). Pri teoriji motenj nas zanima, za koliko se spremeni rezultat, če malo zmotimo (perturbiramo) začetne podatke. Problem je *občutljiv* (slabo pogojen), če lahko pride do velike spremembe, in *neobčutljiv* (dobro pogojen), če so spremembe majhne.

Zgled 1.3 *Denimo, da iščemo presečišče dveh premic. Zanima nas občutljivost presečišča na vodoravne premike premic, ki so npr. lahko posledica nenatančnih podatkov. Iz slike 1.3 vidimo, da je v primeru, ko sta premici skoraj pravokotni, sprememba presečišča sorazmerna premiku premic, torej ne pride do velike spremembe in je sistem neobčutljiv. Če pa sta premici skoraj vzporedni, se lahko presečišče pri majhnem premiku premic močno premakne in sistem je zelo občutljiv.*



neobčutljiv sistem zelo občutljiv sistem

Slika 1.3: Neobčutljivo presečišče premic (levo) in občutljivo presečišče premic (desno).

Poučen je naslednji zgled dveh linearnih sistemov iz [4].

a) Če vzamemo linearni sistem

$$\begin{aligned} x+y &= 2 \\ x-y &= 0, \end{aligned}$$

katerega točna rešitev je $x = y = 1$ in malo zmotimo desno stran v

$$\begin{aligned} x+y &= 1.9999 \\ x-y &= 0.0002, \end{aligned}$$

je rešitev zmotenega sistema enaka $x = 1.00005$ in $y = 0.99985$. Ker se je rezultat spremenil za isti velikostni razred kot podatki, je ta sistem neobčutljiv.

b) Za linearni sistem

$$\begin{aligned} x+0.99y &= 1.99 \\ 0.99x+0.98y &= 1.97 \end{aligned}$$

prav tako velja, da je točna rešitev $x = y = 1$. Če sedaj zmotimo desno stran s podobno veliko motnjo kot v primeru a), je točna rešitev sistema

$$\begin{aligned} x+0.99y &= 1.9889 \\ 0.99x+0.98y &= 1.9701 \end{aligned}$$

enaka $x = 2.97$ in $y = -0.99$. Tu očitno ne gre za majhno spremembo rešitve in ta sistem je zelo občutljiv. □

Zgled 1.4 *Zelo znan je Wilkinsonov*[4] *zgled, kjer vzamemo polinom*

$$p(x) = (x-1)(x-2)\cdots(x-20) = x^{20} - 210x^{19} + \cdots + 20!,$$

ki ima ničle $1, 2, \ldots, 20$. Če malo zmotimo koeficient pri x^{19} in vzamemo polinom

$$g(x) = p(x) - 2^{-23}x^{19},$$

se izkaže, da so točne ničle polinoma g enake

$$\begin{aligned} x_9 &= 8.91752 \\ x_{10,11} &= 10.0953 \pm 0.64310i \\ &\vdots \\ x_{16,17} &= 16.7307 \pm 2.81263i \\ x_{18,19} &= 19.5024 \pm 1.94033i \\ x_{20} &= 20.8469. \end{aligned}$$

Čeprav so vse ničle polinoma p enostavne in lepo separirane, majhna motnja povzroči velike spremembe.

Wilkinson je s tem primerom pokazal, da računanje lastnih vrednosti preko karakterističnega polinoma ni stabilno. Dotedanji postopek je namreč bil, da se iz matrike najprej s kakšnim numeričnim postopkom

[4] Angleški matematik James H. Wilkinson (1919–1986) spada med pionirje analize zaokrožitvenih napak. Napisal je vplivni knjigi *Rounding Errors in Algebraic Processes* (1963) in *The Algebraic Eigenvalue Problem* (1965). Obe sta dolgo časa veljali za glavni referenci o zaokrožitvenih napakah in algoritmih za računanje lastnih vrednosti matrik. V zadnjem obdobju sta njuni vlogi prevzeli knjigi [15] in [9].

(znana je npr. metoda Danilevskega) izračuna koeficiente karakterističnega polinoma, v drugem delu pa se uporabi kakšno numerično metodo za iskanje ničel polinoma.

Izračunani koeficienti karakterističnega polinoma se zaradi zaokrožitvenih napak razlikujejo od točnih koeficientov, kot kaže Wilkinsonov primer, pa lahko majhne spremembe koeficientov povzročijo velike spremembe ničel in posledično na ta način izračunanih lastnih vrednosti. Bolje je uporabiti metode za računanje lastnih vrednosti, ki ne računajo eksplicitno koeficientov karakterističnega polinoma.

Kot zanimivost omenimo še, da lahko obratno algoritme za računanje lastnih vrednosti matrik uporabimo za računanje ničel polinomov. Vsakemu polinomu $p(x) = a_0x^n + a_1x^{n-1} + \cdots + a_n$, $a_0 \neq 0$, namreč ustreza t.i. pridružena matrika

$$C_p = \begin{bmatrix} 0 & 1 & & \\ & \ddots & \ddots & \\ & & & 1 \\ -a_n/a_0 & -a_{n-1}/a_0 & \cdots & -a_1/a_0 \end{bmatrix},$$

za katero lahko hitro preverimo, da se njen karakteristični polinom do množenja s konstanto natančno ujema s polinomom p. Lastne vrednosti matrike C_p so potem ravno ničle polinoma p. Ta postopek med drugim uporablja tudi Matlabov ukaz `roots` *za računanje ničel polinoma.* □

Stopnjo občutljivosti merimo z razmerjem med velikostjo neodstranljive napake in velikostjo napake v podatkih.

Zgled 1.5 *Naj bo $f : \mathbb{R} \to \mathbb{R}$ zvezna in odvedljiva funkcija. Potem za razliko med $f(x)$ in $f(x + \delta x)$, kjer je δx majhna motnja, velja*

$$|f(x + \delta x) - f(x)| \approx |f'(x)| \cdot |\delta x|,$$

torej je $|f'(x)|$ absolutna občutljivost funkcije f v točki x. Če pomislimo na graf funkcije, hitro vidimo, da se pri isti spremembi argumenta x za δx vrednost funkcije spremeni močneje takrat, ko je graf strm.

Za oceno relativne napake dobimo

$$\frac{|f(x + \delta x) - f(x)|}{|f(x)|} \approx \frac{|f'(x)| \cdot |x|}{|f(x)|} \cdot \frac{|\delta x|}{|x|},$$

torej je $\frac{|f'(x)| \cdot |x|}{|f(x)|}$ relativna občutljivost funkcije f v točki x. Vidimo, da je funkcija relativno občutljivejša, če je njena vrednost blizu 0, v primeru, ko je $f(x) = 0$, pa je občutljivost funkcije celo neskončna. Večje relativne spremembe lahko pričakujemo tudi kadar je graf funkcije strm ali kadar je absolutna vrednost argumenta x velika. □

1.5 Direktna in obratna stabilnost

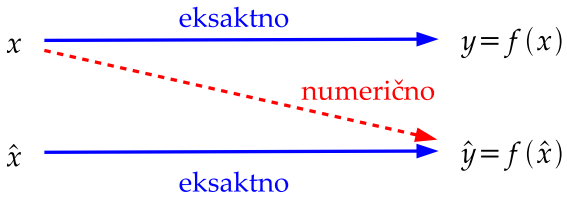
Medtem, ko je pojem občutljivosti povezan s samim problemom in je neodvisen od numerične metode, s katero problem rešujemo, se pojem stabilnosti navezuje na numerično metodo. Ločimo direktno in obratno stabilnost, glavno orodje za preverjanje pa je *analiza zaokrožitvenih napak*.

Denimo, da z neko numerično metodo želimo izračunati vrednost funkcije f pri argumentu x, metoda pa nam namesto eksaktne vrednosti $y = f(x)$ vrne približek $\widehat{y}$.

Razliko med izračunano vrednostjo $\widehat{y}$ in eksaktno vrednostjo y imenujemo *direktna napaka*. Če za vsak x velja, da je (absolutna oz. relativna) direktna napaka majhna, je proces (absolutno oz. relativno) *direktno stabilen*, sicer pa direktno nestabilen. Če je proces direktno stabilen, potem je izračunani rezultat (absolutno oz. relativno) blizu eksaktnega rezultata.

Pri obratni analizi pa nas zanima, za koliko je potrebno zmotiti argument x v $\widehat{x}$, da velja $f(\widehat{x}) = \widehat{y}$. To pomeni, da je $\widehat{y}$ eksaktni rezultat pri zmotenem argumentu $\widehat{x}$. Razliko med $\widehat{x}$ in x imenujemo *obratna napaka*. Če je za vsak x obratna napaka majhna (absolutno oz. relativno), je proces (absolutno oz. relativno) *obratno stabilen*, sicer pa obratno nestabilen. Če je proces obratno stabilen, potem je izračunani rezultat točen rezultat za malo spremenjene začetne podatke (absolutno oz. relativno).

V praksi bomo za večino algoritmov lahko pokazali da so obratno stabilni, direktno stabilnih pa je bolj malo. Tako bomo npr. pokazali, da nam stabilne metode za reševanje linearnega sistema vrnejo točno rešitev linearnega sistema z malo zmoteno matriko.



Pri obratni napaki je izračunana vrednost enaka $f(\widehat{x})$. Če je funkcija f zvezno odvedljiva v x, potem velja

$$|f(\widehat{x}) - f(x)| \approx |f'(x)| \cdot |\widehat{x} - x|.$$

Če f ni absolutno občutljiva, bo pri majhnih vrednostih $|\widehat{x} - x|$ direktna napaka absolutno obratno stabilne metode majhna in metoda bo absolutno direktno stabilna. Podobno velja za relativno stabilnost in relativno napako. Velja naslednja ocena

$$|\text{direktna napaka}| \leq \text{občutljivost} * |\text{obratna napaka}|,$$

ki nam pove, da v primeru obratno stabilne metode lahko pričakujemo, da bo direktna napaka majhna pri tistih vrednostih argumentov, pri katerih je problem neobčutljiv.

Denimo, da je proces obratno stabilen, kar pomeni, da smo izračunali točno rešitev malo zmotenega problema. Vendar, če je problem občutljiv, se točna rešitev bližnjega problema lahko zelo razlikuje od točne rešitve začetnega problema in izračunani rezultat je nenatančen. Nenatančnost je tako lahko posledica ali uporabe stabilnega algoritma na občutljivem problemu ali pa uporabe nestabilnega algoritma na neobčutljivem problemu. Natančnost je zagotovljena, kadar neobčutljiv problem rešimo z obratno stabilno numerično metodo.

1.6 Analiza zaokrožitvenih napak

V tem razdelku bomo na nekaj zgledih prikazali, kako izvajamo analizo zaokrožitvenih napak in kaj lahko sklepamo iz dobljenih ocen za zaokrožitveno napako.

1.6.1 Računanje produkta $n+1$ predstavljivih števil

Dana so predstavljiva števila $x_0, x_1, \ldots, x_n$, računamo pa produkt $p = x_0 x_1 \cdots x_n$, pri čemer bomo, tako kot tudi v nadaljnjih zgledih, predpostavili, da pri računanju ne pride do prekoračitve ali podkoračitve.

Eksaktni algoritem je:
$p_0 = x_0$
$i = 1, \ldots, n$
  $p_i = p_{i-1} x_i$
$p = p_n$

Dejanski algoritem je:
$\widehat{p}_0 = x_0$
$i = 1, \ldots, n$
  $\widehat{p}_i = \widehat{p}_{i-1} x_i (1+\delta_i), \quad |\delta_i| \le u$
$\widehat{p} = \widehat{p}_n$

Na levi strani imamo algoritem, s katerim z eksaktnim računanjem točno izračunamo želeni produkt. Na desni strani je zapis istega algoritma, kjer upoštevamo računanje v plavajoči vejici. Ker je množica vseh predstavljivih števil končna, lahko pri vsakem množenju pride do zaokrožitvene napake, ki je omejena z osnovno zaokrožitveno napako u.

Dobimo

$$\widehat{p} = p(1+\delta_1)\cdots(1+\delta_n) =: p(1+\gamma).$$

Velja

$$(1-u)^n \le 1+\gamma \le (1+u)^n,$$

to pa ocenimo navzgor z

$$(1+u)^n = 1 + \binom{n}{1}u + \binom{n}{2}u^2 + \cdots = 1 + nu + \mathcal{O}(u^2)$$

in navzdol z

$$(1-u)^n \ge 1 - nu.$$

Zadnjo oceno dokažemo z indukcijo, saj iz $(1-u)^n \ge 1-nu$ sledi

$$(1-u)^{n+1} \ge (1-nu)(1-u) = 1-(n+1)u+nu^2 > 1-(n+1)u.$$

Tako lahko pri predpostavki $nu \ll 1$ zanemarimo člene velikosti $\mathcal{O}(u^2)$ in ocenimo $|\gamma| \le nu$, To pomeni, da je relativna napaka odvisna od števila množenj in se z vsakim množenjem lahko poveča za osnovno zaokrožitveno napako u.

Pokazali smo, da je računanje produkta $n+1$ števil direktno stabilno, saj je relativna napaka izračunanega produkta majhna in ni odvisna od začetnih podatkov.

Hitro lahko preverimo, da je algoritem tudi obratno stabilen. Če namreč vzamemo $\widehat{x}_0 = x_0$ in $\widehat{x}_i = x_i(1+\delta_i)$ za $i = 1, \ldots, n$, potem je $\widehat{p} = \widehat{x}_0 \widehat{x}_1 \cdots \widehat{x}_n$ in izračunani rezultat je očitno točen produkt malo spremenjenih začetnih podatkov.

Iz zgornje izpeljave napake za računanje produkta $n+1$ števil lahko izpeljemo tudi naslednjo oceno, ki jo bomo pogosto uporabljali pri različnih ocenah. Če je

$$1+\gamma = \frac{(1+\alpha_1)\cdots(1+\alpha_n)}{(1+\beta_1)\cdots(1+\beta_m)},$$

kjer je $|\alpha_i| \le a_i u \ll 1$ za $i = 1, \ldots, n$ in $|\beta_i| \le b_i u \ll 1$ za $i = 1, \ldots, m$, potem γ lahko ocenimo z

$$|\gamma| \le \left(\sum_{i=1}^{n} a_i + \sum_{i=1}^{m} b_i \right) u.$$

1.6.2 Računanje skalarnega produkta

Imamo dva vektorja $x = [x_1 \ \cdots \ x_n]^T$ in $y = [y_1 \ \cdots \ y_n]^T$ predstavljivih števil, radi pa bi izračunali skalarni produkt $s = y^T x = \sum_{i=1}^{n} x_i y_i$.

Eksaktni algoritem je:
$s_0 = 0$
$i = 0, \ldots, n$
$\quad p_i = x_i y_i$
$\quad s_i = s_{i-1} + p_i$
$s = s_n$

Dejanski algoritem je:
$\widehat{s}_0 = 0$
$i = 0, \ldots, n$
$\quad \widehat{p}_i = x_i y_i (1 + \alpha_i), \quad |\alpha_i| \leq u$
$\quad \widehat{s}_i = (\widehat{s}_{i-1} + \widehat{p}_i)(1 + \beta_i), \quad |\beta_i| \leq u$
$\widehat{s} = \widehat{s}_n$

Za napako β_1 velja $\beta_1 = 0$, saj pri prištevanju k 0 ne pride do zaokrožitvene napake. Z analizo zaokrožitvenih napak dobimo

$$\widehat{s} = \sum_{i=1}^{n} x_i y_i (1 + \gamma_i),$$

kjer je

$$1 + \gamma_1 = (1 + \alpha_1)(1 + \beta_2) \cdots (1 + \beta_n)$$

in

$$1 + \gamma_i = (1 + \alpha_i)(1 + \beta_i) \cdots (1 + \beta_n), \quad i = 2, \ldots, n.$$

Tako lahko ocenimo $|\gamma_1| \leq nu$ in $|\gamma_i| \leq (n - i + 2)u$ za $i = 2, \ldots, n$. To pomeni, da je $\widehat{s}$ točni skalarni produkt relativno malo zmotenih vektorjev x in y. Računanje skalarnega produkta je zato obratno stabilno.

Pri analizi direktne stabilnosti najprej ocenimo absolutno direktno napako. Iz

$$\widehat{s} - s = \sum_{i=1}^{n} x_i y_i \gamma_i,$$

sledi

$$|\widehat{s} - s| \leq \sum_{i=1}^{n} |x_i| \cdot |y_i| \cdot |\gamma_i| \leq nu \sum_{i=1}^{n} |x_i| \cdot |y_i| = nu |y|^T |x|,$$

kjer je $|x| = [|x_1| \ \cdots \ |x_n|]^T$ in $|y| = [|y_1| \ \cdots \ |y_n|]^T$. Od tod za relativno napako dobimo oceno

$$\left| \frac{\widehat{s} - s}{s} \right| \leq \frac{|y|^T |x|}{|y^T x|} nu.$$

Če so vsi produkti $x_i y_i$ enakega predznaka, dobimo $\left| \frac{\widehat{s} - s}{s} \right| \leq nu$ in računanje je direktno stabilno, sicer pa imamo v primeru, ko sta vektorja x in y skoraj ali celo pravokotna, lahko veliko relativno napako.

Pri verjetnosti in statistiki nastopajo večinoma vektorji s samimi nenegativnimi elementi. Za takšne vektorje je računanje skalarnega produkta direktno stabilno. Prav tako lahko hitro vidimo, da je direktno stabilno tudi računanje norme $\|x\|_2$, saj je $\|x\|_2^2 = x^T x$.

Posledica je, da tudi seštevanje n števil v splošnem ni direktno stabilno, saj si ga lahko predstavljamo kot skalarni produkt z vektorjem samih enic. Če seštevamo števila različnih predznakov in je končna točna vsota blizu 0, lahko pričakujemo veliko relativno napako.

Iz tega primera sledi tudi ugotovitev, da kadarkoli je lahko eksaktna vrednost izračuna enaka 0, pri numeričnem računanju pa ne dobimo nujno točno 0, potem taka metoda ne more biti direktno relativno stabilna. Pri množenju $n+1$ števil te težave ni, saj je produkt enak 0 samo takrat, ko je vsaj en izmed faktorjev enak 0, v tem primeru pa tudi pri numeričnem računanju dobimo 0.

1.6.3 Računanje vrednosti polinoma po Hornerjevem algoritmu

Dan je polinom

$$p(x) = a_0x^n + a_1x^{n-1} + \cdots + a_n,$$

računamo pa njegovo vrednost v točki x po Hornerjevem[5] algoritmu. Predpostavka je, da so tako koeficienti $a_0, \ldots, a_n$ kot tudi argument x predstavljiva števila.

Eksaktni algoritem je:
$p_0 = a_0$
$i = 1, \ldots, n$
  $p_i = p_{i-1}x + a_i$
$p = p_n$

Dejanski algoritem je:
$\widehat{p}_0 = a_0$
$i = 1, \ldots, n$
  $\widehat{p}_i = (\widehat{p}_{i-1}x(1+\alpha_i) + a_i)(1+\beta_i) \quad |\alpha_i|, |\beta_i| \leq u$
$\widehat{p} = \widehat{p}_n$

Z analizo zaokrožitvenih napak dobimo

$$\widehat{p} = a_0x^n(1+\gamma_0) + a_1x^{n-1}(1+\gamma_1) + \cdots + a_n(1+\gamma_n),$$

kjer je

$$\begin{array}{rcl} 1+\gamma_0 & = & (1+\alpha_1)\cdots(1+\alpha_n)(1+\beta_1)\cdots(1+\beta_n), \\ 1+\gamma_i & = & (1+\alpha_{i+1})\cdots(1+\alpha_n)(1+\beta_i)\cdots(1+\beta_n), \quad i = 1, \ldots, n-1, \\ 1+\gamma_n & = & (1+\alpha_n), \end{array}$$

torej lahko ocenimo $|\gamma_0| \leq 2nu$ in $|\gamma_i| \leq (2(n-i)+1)u$ za $i = 1, \ldots, n$.

Računanje vrednosti polinoma je obratno stabilno, saj se izračunana vrednost ujema z eksaktno vrednostjo bližnjega polinoma, ki ima koeficiente $a_i(1+\gamma_i)$ namesto a_i, pri nespremenjenem argumentu x.

Iz absolutne napake $\widehat{p} - p = a_0x^n\gamma_0 + a_1x^{n-1}\gamma_1 + \cdots + a_n\gamma_n$ sledi ocena

$$|\widehat{p} - p| \leq 2nu(|a_0||x^n| + |a_1||x^{n-1}| + \cdots + |a_n|),$$

od tod pa

$$\frac{|\widehat{p} - p|}{|p|} \leq \frac{2nu(|a_0||x^n| + |a_1||x^{n-1}| + \cdots + |a_n|)}{|a_0x^n + \cdots + a_n|}. \tag{1.2}$$

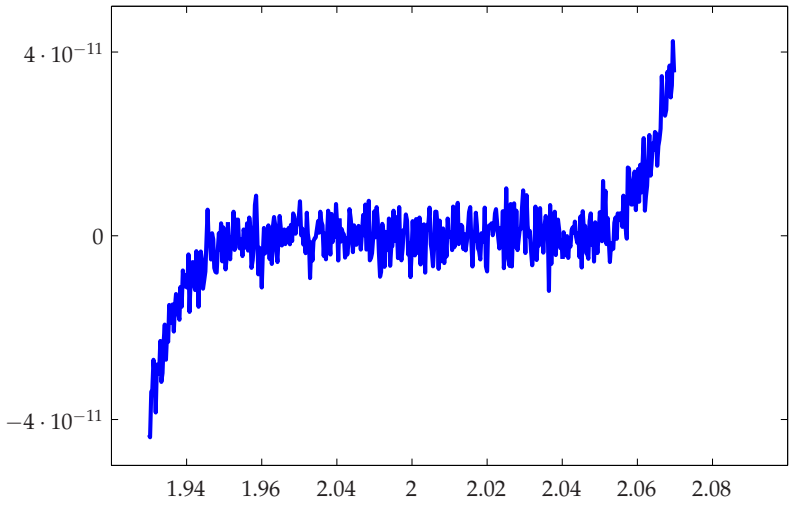
Računanje vrednosti polinoma po Hornerjevem algoritmu ni direktno stabilno, težave pa podobno kot pri računanju skalarnega produkta, lahko pričakujemo, če je vsota blizu 0, členi v vsoti pa niso enako predznačeni.

[5] Angleški matematik William George Horner (1786–1837) je postopek zapisal leta 1819, že 15 let prej pa ga je objavil italijanski matematik Paolo Ruffini (1765–1822). Metoda je bila znana že prej, uporabljal jo je tako Newton kot tudi kitajski matematiki v srednjem veku.

Velikokrat se zgodi, da so ocene za numerično napako zelo pesimistične in v praksi ne dobimo tako velikih napak. Za predstavo o kvaliteti dobljene ocene (1.2) si poglejmo računanje polinoma

$$(x-2)^9 = x^9 - 18x^8 + 144x^7 - 672x^6 + 2016x^5 - 4032x^4 + 5376x^3 - 4608x^2 + 2304x - 512 \quad (1.3)$$

v okolici točke 2. Na sliki 1.4 je graf polinoma na intervalu $[1.93, 2.07]$, ki ga dobimo v Matlabu, če po Hornerjevem algoritmu v dvojni natančnosti izračunamo vrednosti v 500 ekvidistantnih točkah.



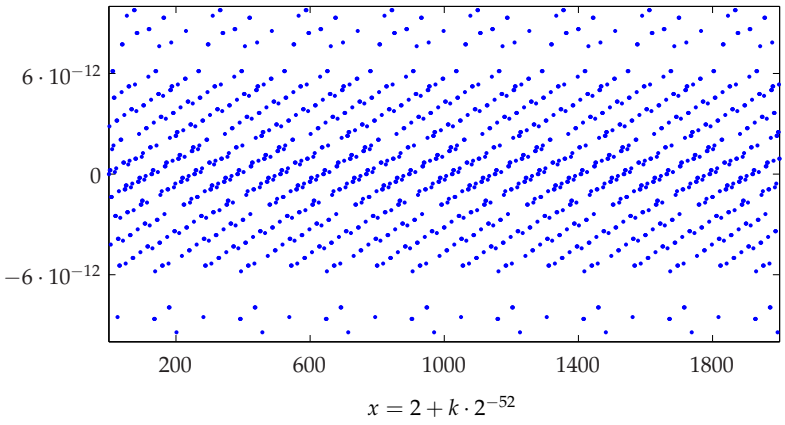
Slika 1.4: Numerično izračunana vrednost polinoma (1.3) v okolici točke 2.

Opazimo, da se numerično izračunane vrednosti ne obnašajo tako lepo, kot bi mogoče pričakovali. Če bi s postopkom bisekcije, ki ga bomo natančneje opisali v razdelku 2.2, računali ničlo tega polinoma, bi za ničlo lahko dobili katerokoli vrednost z intervala $[1.95, 2.05]$, na katerem so numerično izračunane vrednosti tako pozitivne kot negativne. To na prvi pogled zgleda zelo nenatančno, a bomo v naslednjem poglavju spoznali, da se večkratnih ničel ne da izračunati s polno natančnostjo (če seveda njihovih večkratnosti ne poznamo že vnaprej). Za devetkratno ničlo tako velja, da bo točno izračunanih le devetina decimalk, kar se ujema s situacijo na sliki.

Da napake niso povsem naključne, lahko vidimo iz vzorca na sliki 1.5, ki ga dobimo, če narišemo po Hornerjevem algoritmu izračunane vrednosti polinoma za prvih 2000 predstavljivih števil v dvojni natančnosti, ki so večje od 2.

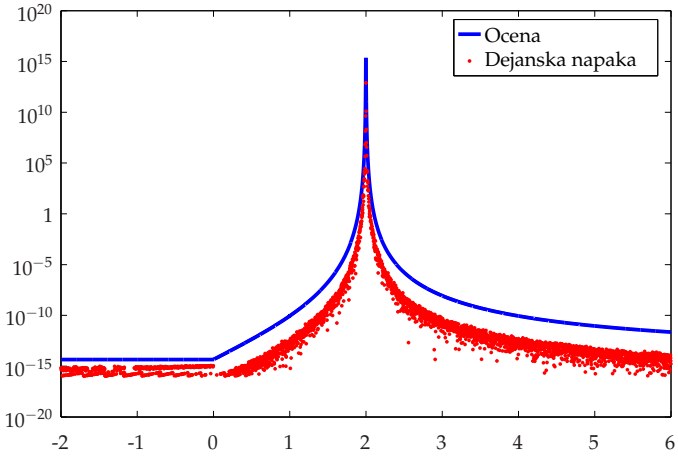
Formulo (1.2) lahko uporabimo za sprotno računanje ocene. Če algoritem razširimo v

$$\begin{aligned}
&p_0 = a_0 \\
&q_0 = |a_0| \\
&i = 1, \ldots, n \\
&\quad p_i = p_{i-1}x + a_i \\
&\quad q_i = q_{i-1}|x| + |a_i| \\
&p = p_n \\
&q = 2nu\tfrac{q_n}{|p|}
\end{aligned}$$



Slika 1.5: Po Hornerjevem algoritmu izračunane vrednosti polinoma (1.3) v najmanjših 2000 predstavljivih številih, večjih od 2.

je q ocena za relativno napako. Tako sicer porabimo dvakrat več dela, a imamo na voljo oceno, s katero lahko preverimo zanesljivost izračunane vrednosti. Vrednost q je izračunana natančno, saj seštevamo samo pozitivne člene, računanje takšne vsote pa je direktno stabilno.



Slika 1.6: Izračunana relativna napaka pri računanju vrednosti polinoma (1.3) po Hornerjevem algoritmu in primerjava z oceno za napako.

Graf na sliki 1.6 prikazuje razmerje med resnično napako in oceno za napako iz formule (1.2). Vrednosti v logaritemski skali predstavljajo absolutne vrednosti relativnih napak. Vidimo, da je zgornja meja dobra ocena za dejansko napako. V bližini točke 2 so napake res večje in se obnašajo tako, kot napoveduje ocena.

Zanimiva je tudi situacija levo od izhodišča, saj so na tem delu tako ocene kot tudi konkretne napake zelo majhne. To se da hitro razložiti, če opazimo, da ima polinom alternirajoče koeficiente. Tako se pri negativnih argumentih v algoritmu seštevajo enako predznačena števila in

napaka je zato dosti manjša.

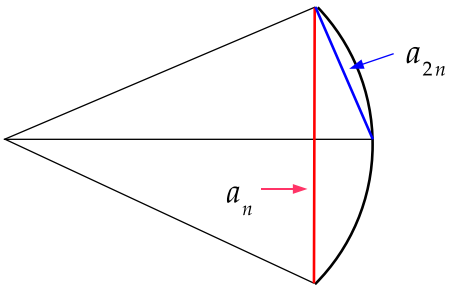
1.7 Poučni primeri

Poglejmo si nekaj zgledov, ki kažejo, da ni dovolj le izpeljati formulo in jo pognati v računalniku. Nekaj teh zgledov se skupaj z dodatnimi zanimivimi primeri nahaja tudi v [15] in [34].

1.7.1 Računanje števila π

Ideja za izračun števila π, ki se pripisuje Arhimedu[6] sledi iz dejstva, da je π limita obsega pravilnega mnogokotnika, včrtanega v krog s polmerom $r = \frac{1}{2}$, ko gre število stranic proti neskončnosti.

Naj bo a_n stranica pravilnega n-kotnika, S_n pa njegov obseg. Poiščimo zvezo med a_n in a_{2n} s slike 1.7.



Slika 1.7: Stranici pravilnega n-kotnika in pravilnega $2n$-kotnika.

S pomočjo Pitagorovega izreka lahko izrazimo

$$a_{2n} = \sqrt{\left(\frac{a_n}{2}\right)^2 + \left(\frac{1}{2} - \sqrt{\frac{1}{4} - \left(\frac{a_n}{2}\right)^2}\right)^2} = \sqrt{\frac{1 - \sqrt{1 - a_n^2}}{2}},$$

od tod pa sledi

$$S_{2n} = 2na_{2n} = 2n\sqrt{\frac{1 - \sqrt{1 - \left(\frac{S_n}{n}\right)^2}}{2}}. \tag{1.4}$$

Ideja je, da računanje začnemo pri S_6, za katerega vemo, da je $S_6 = 3$, potem pa z uporabo formule (1.4) izračunamo π kot limito S_n, ko gre n proti neskončnosti. Izkaže se, da tako v enojni kot v dvojni natančnosti formula (1.4) odpove, saj pride do odštevanja skoraj enako velikih števil, napaka pa se množi z $2n$. V spodnji tabeli so napačne decimalke označene z

[6]Slavni starogrški učenjak Arhimed (287–212 pr.n.št.) je omejil število π med obsega včrtanega in očrtanega pravilnega n-kotnika. Ker je računal z ulomki in največ za $n = 96$, ni imel numeričnih težav, ki jih opisujemo v tem razdelku.

rdečo barvo.

n	Sn	n	Sn
6	3.0000000	768	3.1430726
12	3.1058285	1536	3.1486607
24	3.1326292	3072	3.1819804
48	3.1393456	6144	3.1819804
96	3.1410184	12288	3.0000000
192	3.1413245	24576	4.2426405
384	3.1416743	49152	0.0000000

Da je s formulo (1.4) nekaj narobe, pokaže tudi naslednji kratek razmislek. V limiti naj bi šla vrednost S_n proti π. Toda, pri dovolj velikem n je vrednost S_n/n tako majhna, da je

$$fl\left(1-\left(\frac{S_n}{n}\right)^2\right)=1$$

in izračunamo $S_{2n}=0$, kar dejansko tudi vidimo v zgornji tabeli.

Za stabilno računanje je potrebno formulo preurediti. Stabilna oblika je

$$S_{2n}=2n\sqrt{\frac{\left(1-\sqrt{1-\left(\frac{S_n}{n}\right)^2}\right)\left(1+\sqrt{1-\left(\frac{S_n}{n}\right)^2}\right)}{2\left(1+\sqrt{1-\left(\frac{S_n}{n}\right)^2}\right)}}=S_n\sqrt{\frac{2}{1+\sqrt{1-\left(\frac{S_n}{n}\right)^2}}}. \tag{1.5}$$

Sedaj dobimo pravilne rezultate:

n	Sn	n	Sn
6	3.0000000	768	3.1415839
12	3.1058285	1536	3.1375906
24	3.1326287	3072	3.1415923
48	3.1393502	6144	3.1415927
96	3.1410320	12288	3.1415927
192	3.1414526	24576	3.1415927
384	3.1415577	49152	3.1415927

Ko gre sedaj v limiti n proti neskončnosti, gre spet $\frac{S_n}{n}$ proti 0, a sedaj iz formule (1.5) sledi, da bo S_{2n} kar enak S_n, torej ostane dober približek za π.

Iz tega primera vidimo, da kadar imamo nestabilen postopek, nam ne pomaga niti računanje z večjo natančnostjo. Včasih je možno, kot npr. v tem primeru, preurediti postopek tako, da se med računanjem ne izgublja natančnost.

1.7.2 Seštevanje Taylorjeve vrste za e^{-x}

Vemo, da ima Taylorjeva[7] vrsta za eksponentno funkcijo obliko

$$e^{-x}=\sum_{n=0}^{\infty}(-1)^n\frac{x^n}{n!}$$

[7] Angleški matematik Brook Taylor (1685–1731) jo je zapisal leta 1715.

in konvergira za vsak $x \in \mathbb{C}$. Če pa vrsto seštevamo numerično s prištevanjem členov po vrsti, potem za $x > 0$ ne dobimo najboljših rezultatov. Pri $x = 10$ v enojni natančnosti tako dobimo vsoto $-7.265709 \cdot 10^{-5}$, kar je očitni nesmisel. V naslednji tabeli, ki prikazuje numerično seštete vsote vrste za $x = 1, 2, \ldots, 10$, so napačne decimalke označene z rdečo barvo.

x	e^{-x}	vrsta	relativna napaka
1	$3.678795 \cdot 10^{-1}$	$3.678794 \cdot 10^{-1}$	$1.4 \cdot 10^{-7}$
2	$1.353353 \cdot 10^{-1}$	$1.353353 \cdot 10^{-1}$	$2.3 \cdot 10^{-7}$
3	$4.978707 \cdot 10^{-2}$	$4.978702 \cdot 10^{-2}$	$9.3 \cdot 10^{-7}$
4	$1.831564 \cdot 10^{-2}$	$1.831531 \cdot 10^{-2}$	$1.8 \cdot 10^{-5}$
5	$6.737947 \cdot 10^{-3}$	$6.737477 \cdot 10^{-3}$	$7.0 \cdot 10^{-5}$
6	$2.478752 \cdot 10^{-3}$	$2.477039 \cdot 10^{-3}$	$6.9 \cdot 10^{-4}$
7	$9.118820 \cdot 10^{-4}$	$9.139248 \cdot 10^{-4}$	$2.2 \cdot 10^{-3}$
8	$3.354626 \cdot 10^{-4}$	$3.485951 \cdot 10^{-4}$	$3.9 \cdot 10^{-2}$
9	$1.234098 \cdot 10^{-4}$	$1.799276 \cdot 10^{-4}$	$4.6 \cdot 10^{-1}$
10	$4.539993 \cdot 10^{-5}$	$-7.265709 \cdot 10^{-5}$	$2.6 \cdot 10^{-0}$

Razlog za velike napake je, da zaporedje členov vrste alternira, poleg tega pa členi po absolutni vrednosti nekaj časa naraščajo, preden začnejo padati proti 0. Vse skupaj se računa v enojni natančnosti, za katero vemo, da ne more prikazati več kot sedem točnih decimalk. Ker so rezultati izpisani tako, da vedno izpišemo 6 števk za decimalno vejico, zadnje decimalke števil, ki so velikostnega razreda 10^2 ali več, nimajo nobenega pravega pomena. Absolutne vrednosti delnih vsot na začetku naraščajo, potem pa začnejo padati. Prej ali slej se zgodi, da pridejo v ospredje napačne decimalke, ki pri velikih absolutnih vrednostih niso bile nevarne, končni rezultat pa povsem pokvarijo.

n	a_n	s_n	n	a_n	s_n
0	1.000000	1.000000	20	41.103188	13.396751
1	−10.000000	−9.000000	21	−19.572947	−6.176195
2	50.000000	41.000000	22	8.896794	2.720599
3	−166.666672	−125.666672	23	−3.868171	−1.147572
4	416.666687	291.000000	24	1.611738	0.464166
5	−833.333374	−542.333374	25	−0.644695	−0.180529
6	1388.888916	846.555542	26	0.247960	0.067430
7	−1984.127075	−1137.571533	27	−0.091837	−0.024407
8	2480.158936	1342.587402	28	0.032799	0.008392
9	−2755.732178	−1413.144775	29	−0.011310	−0.002918
10	2755.732178	1342.587402	30	0.000380	0.000852
11	−2505.211182	−1162.623779	31	−0.001216	−0.000364
12	2087.676025	925.052246	32	0.000380	0.000016
13	−1605.904663	−680.852417	33	−0.000115	−0.000099
14	1147.074707	466.222290	34	0.000034	−0.000065
15	−764.716492	−298.494202	35	−0.000010	−0.000075
16	477.947815	179.453613	36	0.000003	−0.000072
17	−281.145782	−101.692169	37	−0.000001	−0.000073
18	156.192108	54.499939	38	0.000000	−0.000073
19	−82.206375	−27.706436	39	−0.000000	−0.000073

Pri računanju e^{10} teh težav ni, saj so vsi členi pozitivni, rezultat pa velik in je relativna napaka potem majhna. Zaradi tega lahko natančno izračunamo e^{-10} tako, da najprej izračunamo e^{10} in nato delimo $e^{-10} = 1/e^{10}$.

1.7.3 Reševanje kvadratne enačbe

Vemo, da sta rešitvi kvadratne enačbe

$$ax^2 + bx + c = 0$$

podani s formulo

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}. \tag{1.6}$$

Direktna uporaba zgornje formule lahko v nekaterih primerih pripelje do hudih napak. Npr., če so koeficienti zelo veliki ali zelo majhni, potem lahko pride do prekoračitve ali podkoračitve obsega pri računanju $b^2 - 4ac$. Temu se lahko izognemo, če npr. predhodno celo enačbo delimo s tistim izmed koeficientov a, b, c, ki ima največjo absolutno vrednost.

Druga težava je odštevanje približno enako velikih števil, ki se lahko pojavi v eni izmed rešitev (1.6). Temu se izognemo tako, da eno rešitev (odvisno od predznaka b) izračunamo po formuli (1.6), drugo pa po Vietovi[8] formuli $x_2 = c/(ax_1)$.

Če vzamemo npr. $a = 1.2345678$, $b = 76543210.5$, $c = 0.1122334455$, potem v dvojni natančnosti po formuli (1.6) dobimo

$$x_1 = -6.200000558900046 \cdot 10^7, \quad x_2 = -6.034970778375905 \cdot 10^{-9}$$

točni ničli pa sta

$$\tilde{x}_1 = -6.200000558900046 \cdot 10^7, \quad \tilde{x}_2 = -1.466275647008561 \cdot 10^{-9}.$$

Prvi približek je natančen, pri drugem pa smo zaradi odštevanja enako velikih količin v bistvu izgubili vse točne decimalke. Če drugo ničlo izračunamo preko $x_2 = c/(ax_1)$, dobimo pravilno $x_2 = -1.466275647008561 \cdot 10^{-9}$.

1.7.4 Rekurzivna formula za I_{10}

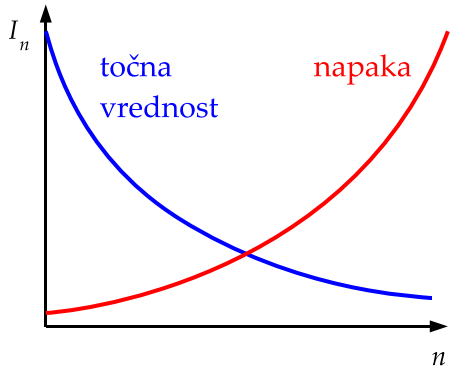
Za integrale

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

za $n = 0, 1, \ldots$ lahko z integriranjem po delih izpeljemo rekurzivno formulo

$$I_n = x^n e^{x-1}\Big|_0^1 - n \int_0^1 x^{n-1} e^{x-1} dx = 1 - nI_{n-1}.$$

[8] Fomule, ki povezujejo koeficiente polinoma z vsotami in produkti njegovih ničel, je zapisal francoski matematik Franciscus Vieta (1540–1603).



Slika 1.8: Z naraščajočim n napaka raste, točna vrednost pa pada.

To rekurzivno formulo želimo uporabiti za izračun I_{10}, saj poznamo začetno vrednost $I_0 = 1 - e^{-1}$. Ko to izračunamo v enojni natančnosti, rezultati niso najboljši:

n	I_n	n	I_n
0	0.6321205	7	0.1124296
1	0.3678795	8	0.1005630
2	0.2642411	9	0.0949326
3	0.2072767	10	0.0506744
4	0.1708932	11	0.4425812
5	0.1455340	12	−4.3109741
6	0.1267958	13	57.0426636

Razlog za velike napake, ki naraščajo z n, se skriva v formuli $I_n = 1 - nI_{n-1}$. Napaka pri členu I_{n-1} se pomnoži z n in torej po absolutni vrednosti hitro narašča, točne vrednosti I_n pa padajo, kot kaže slika 1.8.

Ker delež napake prej ali slej prevlada nad deležem točne vrednosti, dobimo z naraščajočim n vedno slabše rezultate.

Rešitev je, da vrednosti računamo v obratni smeri: $I_{n-1} = (1 - I_n)/n$. Tako se napaka v vsakem koraku deli z n in če začnemo pri nekem dovolj poznem členu, lahko z začetnim $I_n = 0$ izračunamo vse začetne člene dovolj natančno. Če tako začnemo s približkom $I_{26} = 0$, izračunamo (v enojni natančnosti) vse člene od I_{13} do I_0 na vse decimalke točno. Kljub temu, da imamo na začetku veliko napako, ker smo vzeli $I_{26} = 0$, se sedaj napaka vedno manjša, prispevek točnega rezultata pa narašča. Pri $n = 13$ postane napaka dovolj majhna, da so vse decimalke približka za I_{13} točne.

n	I_n	n	I_n
0	0.6321205	8	0.1009320
1	0.3678795	9	0.0916123
2	0.2642411	10	0.0838771
3	0.2072766	11	0.0773522
4	0.1708934	12	0.0717733
5	0.1455329	13	0.0669477
6	0.1268024	⋮	⋮
7	0.1123835	26	0.0000000

1.7.5 Zaporedno korenjenje in kvadriranje

Izberimo predstavljivo število $x > 0$ in ga numerično najprej 80-krat korenimo in nato 80-krat kvadriramo. Kaj dobimo? Izkaže se, da za $x \geq 1$ dobimo 1, za $0 < x < 1$ pa dobimo 0! Za razumljivejšo razlago si poglejmo računanje na modelu kalkulatorja HP 48G, kjer je baza desetiška, dolžina mantise pa je 12.

Poglejmo najprej primer $0 < x < 1$. Za takšne x velja $\sqrt{x} > x$. Največje predstavljivo število, ki je še manjše od 1, je $1 - 10^{-12}$ oziroma $0.\underbrace{9\ldots9}_{12}$. Zaradi $\sqrt{1+x} = 1 + \frac{1}{2}x - \frac{1}{8}x^2 + \cdots$ velja

$$\sqrt{1-10^{-12}} = 1 - \frac{1}{2}10^{-12} - \frac{1}{8}10^{-24} + \cdots = 0.\underbrace{9\ldots9}_{12}4\underbrace{9\ldots9}_{11}87\ldots$$

in število se zaokroži na $0.\underbrace{9\ldots9}_{12}$. Tako s korenjenjem nikoli ne pridemo do 1, ko pa kvadriramo, je število vedno manjše, dokler ne pride do podkoračitve in dobimo 0.

Če je $x > 1$, potem je $\sqrt{x} < x$. Najmanjše predstavljivo število, ki je še večje od 1 je $1 + 10^{-11}$. Tu dobimo

$$\sqrt{1+10^{-11}} = 1 + \frac{1}{2}10^{-11} - \frac{1}{8}10^{-22} + \cdots = 1.\underbrace{0\ldots0}_{11}4\underbrace{9\ldots9}_{10}87\ldots,$$

kar se zaokroži na 1. Tako s korenjenjem pridemo do 1, kar se s kvadriranjem več ne spremeni.

1.7.6 Tričlenska rekurzivna formula

Dana je tričlenska rekurzivna formula

$$x_{k+1} = \frac{9}{4}x_k - \frac{1}{2}x_{k-1}, \quad k \geq 2, \tag{1.7}$$

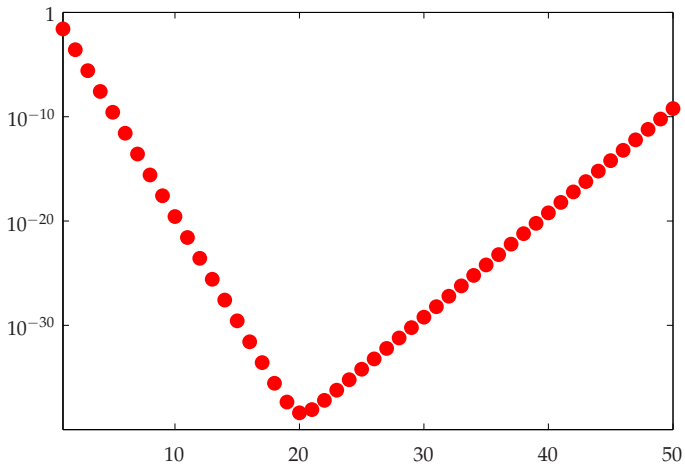
in začetni dve vrednosti $x_1 = 1/3$ in $x_2 = 1/12$. Pri teh pogojih je eksaktna rešitev $x_k = (4/3) \cdot 4^{-k}$.

V dvojni natančnosti izračunani členi po formuli (1.7) iz $x_1 = 1/3$ in $x_2 = 1/12$ so prikazani na sliki 1.9 v logaritemski skali. Namesto, da bi členi ves čas padali, začnejo pri $n = 20$ naraščati, kar je očiten znak, da je z računanjem nekaj narobe. Odgovor se skriva v dejstvu, da je splošna rešitev diferenčne enačbe (1.7) enaka

$$x_k = \alpha \left(\frac{1}{4}\right)^k + \beta 2^k.$$

Ker ne $1/3$ niti $1/12$ nista v dvojni natančnosti predstavljivi števili, pri numeričnem računanju dobimo $\beta \neq 0$ in prej ali slej drugi člen prevlada. Če smo še bolj natančni, lahko ugotovimo, da imamo pri računanju v dvojni natančnosti zaradi zaokrožitvenih napak na začetku

$$\begin{aligned} x_1 &= \frac{1}{3}(4^0 \quad + 2^{-56}), \\ x_2 &= \frac{1}{3}(4^{-1} + 2^{-55}). \end{aligned}$$



Slika 1.9: Členi zaporedja (1.7) pri začetnih pogojih $x_1 = 1/3$ in $x_2 = 1/12$, izračunani v dvojni natančnosti, v logaritemski skali.

Eksaktna rešitev za zgornja x_1 in x_2 je

$$x_k = \frac{1}{3}(4^{1-k} + 2^{k-57}),$$

do preobrata pa pride ravno ko je $4^{1-k} = 2^{k-57}$ oziroma $k \approx 20$. Vse to se zelo dobro ujema z numerično izračunanimi vrednostmi, prikazanimi na sliki 1.9.

1.7.7 Seštevanje številske vrste

Znano je, da velja

$$\sum_{k=1}^{\infty} \frac{1}{k^2} = \frac{\pi^2}{6} = 1.644934066848\ldots. \tag{1.8}$$

Kako bi to sešteli, če tega ne bi vedeli?

Prva ideja, ki nam pride na pamet je, da prištevamo člene, dokler se vsota ne spreminja več. V enojni natančnosti tako dobimo 1.64472532..., to vrednost pa dosežemo pri $k = 4096$. V tem koraku delni vsoti ≈ 1.6 prištevamo 2^{-24}, kar je premajhno, da bi se delna vsota spremenila.

Omenjeni pristop seveda ni dober, saj je napaka izračunanega približka prevelika. Mimogrede omenimo še, da če prištevamo člene, dokler se delna vsota ne spremeni več, potem tudi za harmonično vrsto $\sum_{k=1}^{\infty} \frac{1}{k}$, za katero vemo, da je divergentna, izračunamo neko končno vrednost.

Druga možnost je, da ocenimo, koliko členov bi bilo potrebno sešteti, da bo napaka dovolj majhna, potem pa jih seštevamo v obratnem vrstnem redu od majhnih vrednosti proti velikim. Tako nimamo težav s tem, da se vsota ne bi spremenila, ko ji prištejemo nov člen.

Za našo vrsto se na žalost izkaže, da konvergira zelo počasi in bi morali, da bi izračunali približek 1.64493406 z 8 točnimi decimalkami, sešteti kar 10^9 členov, kar je seveda občutno preveč.

Nobeden izmed zgornjih preprostih pristopov torej ni uporaben za izračun vsote vrste (1.8). Na srečo obstajajo tudi numerične metode za računanje vsot vrst, s katerimi lahko dobimo dovolj natančno vsoto zgornje vrste že z uporabo relativno malo začetnih členov.

1.7.8 Računanje $(e^x - 1)/x$

Računamo vrednosti funkcije $f(x) = (e^x - 1)/x$ v bližini $x = 0$, kjer je $\lim_{x\to 0} f(x) = 1$.

Z algoritmom

if $x = 0$
    $y = 1$
else
    $y = (e^x - 1)/x$
end

dobimo pri majhnih x velike napake, algoritem

$z = e^x$
if $z = 1$
    $y = 1$
else
    $y = (z - 1)/\ln z$
end

pa vrača pravilne vrednosti, kar je razvidno iz spodnje tabele.

x	Algoritem 1	Algoritem 2
10^{-5}	1.000005000006965	1.000005000016667
10^{-6}	1.000000499962184	1.000000500000167
10^{-7}	1.000000049433680	1.000000050000002
10^{-8}	$9.999999939225290 \cdot 10^{-1}$	1.000000005000000
10^{-9}	1.000000082740371	1.000000000500000
10^{-10}	1.000000082740371	1.000000000050000
10^{-11}	1.000000082740371	1.000000000005000
10^{-12}	1.000088900582341	1.000000000000500
10^{-13}	$9.992007221626408 \cdot 10^{-1}$	1.000000000000050
10^{-14}	$9.992007221626408 \cdot 10^{-1}$	1.000000000000005
10^{-15}	1.110223024625156	1.000000000000000
10^{-16}	0.000000000000000	1.000000000000000

Poglejmo računanje pri $x = 7 \cdot 10^{-14}$. Pri prvem algoritmu dobimo

$$y = fl\left(\frac{e^x - 1}{x}\right) = fl\left(\frac{6.994405055138486 \cdot 10^{-14}}{7.000000000000001 \cdot 10^{-14}}\right) = 0.99920072216264,$$

pri drugem algoritmu pa

$$y = fl\left(\frac{e^x - 1}{\ln(e^x)}\right) = fl\left(\frac{6.994405055138486 \cdot 10^{-14}}{6.994405055138241 \cdot 10^{-14}}\right) = 1.00000000000004.$$

To je vse lepo in prav, vendar! Točne vrednosti, ki bi morale nastopati v drugem algoritmu, so

$$y = fl\left(\frac{e^x - 1}{\ln(e^x)}\right) = fl\left(\frac{7.000000000000245 \cdot 10^{-14}}{7.000000000000001 \cdot 10^{-14}}\right) = 1.00000000000004.$$

Kako je možno, da iz dveh netočnih vrednosti spet dobimo točno? Pokažimo, da je drugi algoritem stabilen.

Prvi korak v algoritmu je izračun $z = e^x$. Dobimo $\widehat{z} = e^x(1+\delta)$, kjer je $|\delta| \leq u$, saj v aritmetiki, ki se drži IEEE standardov, tudi za računanje elementarnih funkcij velja, da je zaokrožitvena napaka omejena z osnovno zaokrožitveno napako.

Denimo, da smo dobili $\widehat{z} = 1$. To pomeni, da je $e^x(1+\delta) = 1$, torej je

$$x = -\ln(1+\delta) = -\delta + \delta^2/2 - \delta^3/3 + \cdots.$$

Za tak x je pravilno, da algoritem za $f(x)$ vrne 1, saj iz $f(x) = 1 + x/2 + x^2/6 + \cdots$ sledi, da je $fl(f(x)) = 1$.

Če je $\widehat{z} \neq 1$, potem računamo $y = (z-1)/\ln z$. Numerično velja

$$\widehat{y} = \frac{(\widehat{z}-1)(1+\epsilon_1)}{\ln \widehat{z}(1+\epsilon_2)}(1+\epsilon_3),$$

kjer je $|\epsilon_i| \leq u$ za $i = 1, 2, 3$. Pri tem so $\epsilon_1, \epsilon_2, \epsilon_3$ po vrsti zaokrožitvene napake pri odštevanju, računanju logaritma in deljenju. Če definiramo

$$g(z) := \frac{z-1}{\ln z},$$

potem lahko ocenimo, da je $\widehat{y} = g(\widehat{z})(1+\gamma)$, kjer je $|\gamma| \leq 3u$. Oceniti moramo še, za koliko se $g(\widehat{z})$ razlikuje od $g(z)$.

Če vpeljemo $w = z - 1$, dobimo

$$g(z) = \frac{z-1}{\ln z} = \frac{w}{\ln(1+w)} = 1 + \frac{w}{2} + \mathcal{O}(w^2).$$

To pomeni, da za majhne x, ko je $z \approx 1$, velja

$$|g(\widehat{z}) - g(z)| \approx \frac{|\widehat{w} - w|}{2} = \frac{|\widehat{z} - z|}{2} \approx \frac{e^x|\delta|}{2} \approx \frac{|\delta|}{2} \leq \frac{u}{2},$$

kjer je $\widehat{w} = \widehat{z} - 1$. Ker je $g(z) \approx 1$, to pomeni, da je $g(\widehat{z}) = g(z)(1+\eta)$, kjer je $|\eta| \leq 0.5u$. Skupna ocena je potem $\widehat{y} = y(1+\alpha)$, kjer je $|\alpha| \leq 3.5u$.

Čeprav sta $\widehat{z} - 1$ in $\ln \widehat{z}$ po vrsti slaba približka za $z - 1$ in $\ln z$, smo pokazali, da je $(\widehat{z}-1)/\ln \widehat{z}$ zelo dober približek za $y = (z-1)/\ln z$. Računanje vrednosti funkcije po tej formuli je celo direktno stabilno.

Matlab

Matlab računa po standardu IEEE v dvojni natančnosti. Osnovni ukazi, povezani s predstavljivimi števili, so:

- `eps` vrne razdaljo med 1 in najbližjim večjim predstavljivim številom, kar je ravno dvakratnik osnovne zaokrožitvene napake u oziroma 2^{-52} v dvojni natančnosti.
- `realmax` je največje še predstavljivo število.
- `realmin` je najmanjše pozitivno predstavljivo število.
- `Inf` je neskončno število.
- `NaN` je nedoločeno število.
- `single(x)` vrne število v enojni natančnosti, ki je najbližje x.
- `double(x)` vrne število v dvojni natančnosti, ki je najbližje x.

Dodatna literatura

Skoraj vsak učbenik iz področja numerične analize ima na začetku poglavje o računanju v plavajoči vejici in zaokrožitvenih napakah. Na voljo je tako zelo obsežna literatura. V slovenščini lahko to snov najdete med drugim v knjigah [5], [8] in [34]. Zelo natančna predstavitev numeričnega računanja in zaokrožitvenih napak je v [15]. Omenimo lahko tudi učbenike [6], [13] in [17].

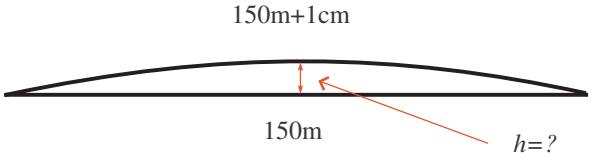
Poglavje 2

Nelinearne enačbe

2.1 Uvod

V tem poglavju bomo obravnavali numerične metode za reševanje nelinearnih enačb oblike $f(x) = 0$, kjer je f dana realna oz. kompleksna funkcija.

Zgled 2.1 *Denimo, da je* 150m *dolga tračnica trdno vpeta na obeh koncih. Zaradi segrevanja se raztegne za* 1cm *in usloči v obliki krožnega loka (glej sliko 2.1). Za koliko se bo sredina tračnice dvignila od tal?*



Slika 2.1: Na obeh koncih trdno vpeta tračnica se usloči v obliki krožnega loka.

Poskusite sami rešiti nalogo in izračunati rešitev na 10 točnih mest. Čeprav najprej pomislimo na manjše vrednosti, se izkaže, da se na sredini tračnica dvigne kar za 75.00074999cm. □

Na splošno je o obstoju in številu rešitev enačbe $f(x) = 0$, kjer je $f : \mathbb{R} \to \mathbb{R}$, težko karkoli povedati. Tako lahko enačba

- ima eno samo rešitev, primer je $e^x + x = 0$,
- ima neskončno število rešitev, primer je $x + \tan x = 0$,
- ima končno število rešitev, primer je $x^3 - 5x + 1 = 0$,
- nima nobene rešitve, primer je $x^2 + 1 = 0$.

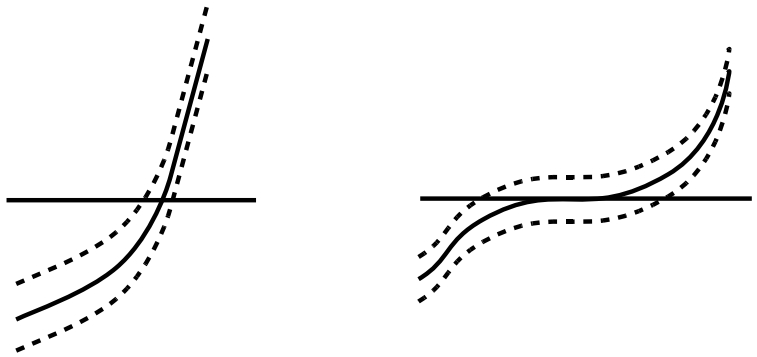
Seveda ima zadnja enačba rešitev v prostoru kompleksnih števil, a je potem od izvora same enačbe odvisno, ali ima kompleksna rešitev tudi pravi pomen.

Znanih je nekaj zadostnih pogojev za obstoj rešitve. Tako Lagrangeev izrek v primeru, ko je f realna zvezna funkcija na intervalu $[a, b]$ in velja $f(a) \cdot f(b) < 0$, zagotavlja obstoj takega števila $\xi \in (a, b)$, da je $f(\xi) = 0$. Dokaz tega izreka je konstruktiven in vodi do postopka bisekcije, ki jo bomo kot prvo numerično metodo za nelinearne enačbe predstavili v razdelku 2.2.

Naj bo α ničla funkcije f, ki je zvezno odvedljiva v α. Če je $f'(\alpha) \neq 0$, potem nam izrek o inverzni funkciji zagotavlja, da obstajata okolica U točke α in okolica V točke 0, da f bijektivno preslika U na V. To nam zagotavlja, da v U razen α ni nobene druge ničle funkcije f. V takem primeru pravimo, da je α *enostavna ničla* funkcije f. Če je $f'(\alpha) = 0$, potem je α *večkratna ničla* funkcije f. Če, ob predpostavki, da je f dovoljkrat zvezno odvedljiva, velja $f(\alpha) = f'(\alpha) = \cdots = f^{(m-1)}(\alpha) = 0$ in $f^{(m)}(\alpha) \neq 0$, potem je α m-kratna ničla funkcije f.

Naj bo α ničla funkcije f. Poglejmo, kako je z občutljivostjo računanja ničle. Pri tem ne moremo govoriti o relativni občutljivosti, saj je $f(\alpha) = 0$.

Denimo, da je α enostavna ničla funkcije f, ki je v točki α zvezno odvedljiva. Potem je odvod inverzne funkcije v točki 0 enak $1/f'(\alpha)$ in po zgledu 1.5 je $1/|f'(\alpha)|$ absolutna občutljivost ničle α. Torej, če je x približek za α in velja $|f(x)| \leq \epsilon$, potem lahko ocenimo, da je $|x - \alpha| \leq \epsilon/|f'(\alpha)|$. To pomeni, da lahko pričakujemo natančno numerično izračunane približke za ničlo, kadar tangenta funkcije f v točki α ni zelo položna. To je razvidno tudi iz slike 2.2. Če predpostavimo, da je funkcija f izračunana z določeno nenatančnostjo, potem bomo ničlo v primeru, ko je zelo občutljiva, težko izračunali s polno natančnostjo.



neobčutljiva ničla zelo občutljiva ničla

Slika 2.2: Neobčutljiva ničla (levo) in občutljiva ničla (desno).

Če je ničla večkratna, je $f'(\alpha) = 0$, kar pomeni, da je neskončno občutljiva. Če je ničla npr. dvojna, potem iz razvoja v Taylorjevo vrsto sledi, da v primeru $|f(x)| \leq \epsilon$ v bližini ničle velja

$$|x - \alpha| \leq \sqrt{\frac{2\epsilon}{|f''(\alpha)|}}.$$

To pomeni, da lahko dvojno ničlo izračunamo le z natančnostjo $\mathcal{O}(\sqrt{\epsilon})$. Pri m-kratni ničli lahko podobno pokažemo, da jo lahko izračunamo le z natančnostjo $\mathcal{O}(\epsilon^{1/m})$. Kot bomo kasneje praktično videli tudi v numeričnih zgledih, ne glede na izbrano numerično metodo, večkratnih ničel ne moremo izračunati z isto natančnostjo kot enostavne ničle.

2.2 Bisekcija

Bisekcija je ena izmed najbolj enostavnih in najbolj zanesljivih numeričnih metod za iskanje ničel realnih funkcij. Denimo, da poznamo tak interval $[a, b]$, da je zvezna funkcija $f : \mathbb{R} \to \mathbb{R}$ v krajiščih različno predznačena, torej $f(a) \cdot f(b) < 0$. Potem iz zveznosti sledi, da ima f na intervalu (a, b) vsaj eno ničlo. Če vzamemo sredinsko točko $c = (a + b)/2$, potem bo (razen, če je $f(c) = 0$, kar pomeni, da smo imeli srečo in zadeli ničlo) na enem izmed intervalov $[a, c]$ ali $[c, b]$ funkcija spet različno predznačena v krajiščih in to vzamemo za nov interval $[a, b]$. Postopek rekurzivno ponavljamo in v vsakem koraku nadaljujemo z razpolovljenim intervalom, ki zagotovo vsebuje vsaj eno ničlo. Ko je interval dovolj majhen, končamo in vrnemo točko s sredine intervala kot približek za ničlo funkcije f.

Algoritem 2.1 Bisekcija za iskanje ničle zvezne funkcije f. Začetni podatki sta točki a in b, v katerih je f različno predznačena in toleranca ϵ.

```
while |b − a| > ε
      c = a + (b − a)/2
      if sign(f(a)) = sign(f(c))
            a = c
      else
            b = c
      end
end
```

Čeprav je algoritem zelo enostaven, vseeno omenimo nekaj pomembnih podrobnosti. Kot prvo in to velja ne le za bisekcijo, temveč za vse numerične metode, je zelo malo verjetno, da obstaja tako predstavljivo število α, da je $f(\alpha)$ eksaktno ali numerično izračunano enako 0. Zaradi tega v algoritmu tudi nikoli ne preverjamo, če je slučajno $f(c) = 0$. Namesto tega pri bisekciji poskušamo ničlo omejiti med dve čim bližji točki, v katerih je funkcija različno predznačena.

Čeprav bi bilo enostavneje sredinsko točko c računati po formuli $c = (a + b)/2$, jo računamo kot $c = a + (b - a)/2$. Pri eksaktnem računanju sta izraza enakovredna, pri numeričnem računanju pa ima drugi izraz nekaj prednosti:

- Pri računanju vsote $a + b$ lahko pri prvi formuli pride v sicer malo verjetnem primeru, ko sta a in b po absolutni vrednosti zelo velika, do prekoračitve, pa čeprav je $(a + b)/2$ spet v območju predstavljivih števil.
- Še hujša težava je, da se pri prvi formuli lahko zgodi, da izračunani c ne bo ležal na intervalu $[a, b]$. Če npr. računamo v $P(10, 4, -10, 10)$, potem za $a = 0.6666$ in $b = 0.6667$ dobimo $fl((a + b)/2) = 0.6665$.

Paziti moramo, da je toleranca ϵ smiselno izbrana. Če npr. izberemo premajhen ϵ, se lahko zgodi, da je $|b - a| > \epsilon$, pa čeprav sta a in b dve zaporedni predstavljivi števili. Tako npr. iz $a = 1000$ in $b = 1001$ pri računanju v enojni natančnosti ne moremo dobiti predstavljivega intervala, za katerega bi veljalo $|b - a| \leq 10^{-10}$.

Namesto pogoja $\text{sign}(f(a)) = \text{sign}(f(c))$ bi lahko preverili, če velja $f(a) \cdot f(c) > 0$. Bolje je preverjati le ujemanje predznakov, saj pri množenju lahko pride do težav zaradi prekoračitve ali podkoračitve.

V vsakem koraku metode moramo izračunati eno vrednost funkcije f (v točki c). Vrednost v točki a poznamo še iz prejšnjega koraka (čeprav v algoritmu 2.2 to ni eksplicitno navedeno). V bistvu nas sama vrednost funkcije ne zanima, pomemben je le predznak.

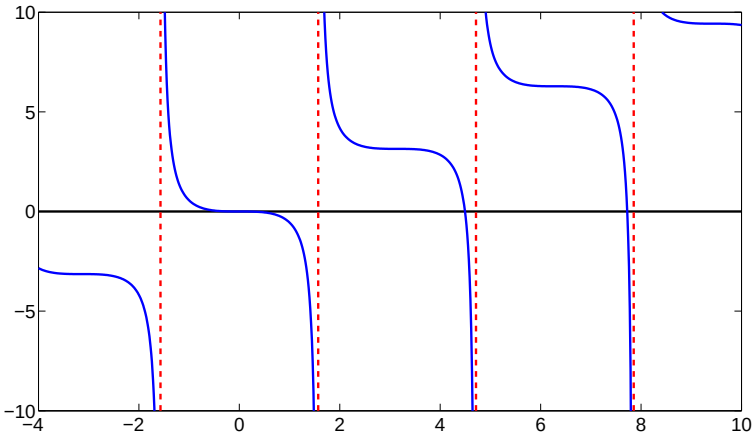
Metoda odpove pri sodih ničlah, saj je tam funkcija na obeh straneh ničle enako predznačena. Prav tako z njo ne moremo računati ničel kompleksnih funkcij. Lahko pa bisekcijo uporabimo za iskanje lihih polov, saj tam tako kot pri lihih ničlah pride do spremembe predznaka funkcije.

Možno je, da ima funkcija f na intervalu (a, b) več kot le eno ničlo (vemo le, da jih je liho mnogo). Bisekcija bo vedno našla le eno izmed teh ničel, ni pa nobenega pravila, katero izmed ničel dobimo.

Ker se v vsakem koraku interval razpolovi, smo v vsakem koraku za faktor $1/2$ bližje rešitvi. Kot bomo videli v naslednjem razdelku, to pomeni, da ima bisekcija linearno konvergenco. Sama hitrost konvergence je fiksna in ni odvisna od funkcije f. Ne glede na obnašanje f bomo pri enaki izbiri začetnega intervala $[a, b]$ in parametra ϵ vedno potrebovali enako število korakov. Narediti moramo pač toliko korakov, da bo $|b - a|2^{-k} \leq \epsilon$, torej za število potrebnih korakov k velja

$$k \geq \log_2 \left(\frac{|b - a|}{\epsilon} \right).$$

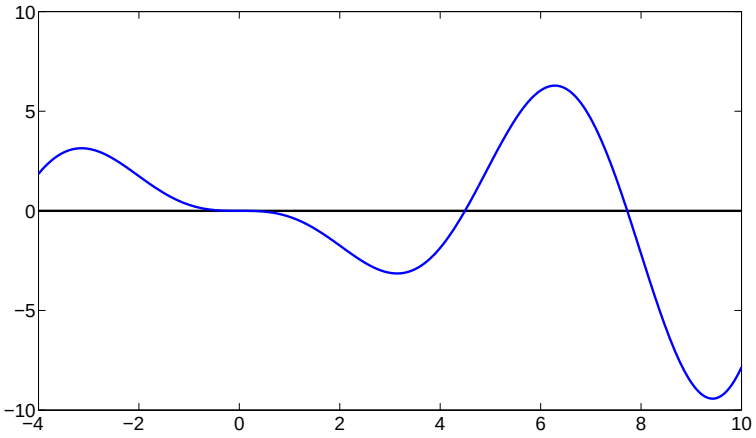
Zgled 2.2 *Z bisekcijo poiščimo ničle funkcije* $f(x) = x - \tan x$.



Slika 2.3: Graf funkcije $f(x) = x - \tan x$.

Razen za trivialno ničlo $x = 0$ *je težko poiskati dober začetni interval, saj imamo težave s poli. Ničle so blizu polov, razen na zelo majhnem intervalu med ničlo in polom pa zavzame funkcija same pozitivne vrednosti. Rešitev je, da iščemo ničle funkcije* $g(x) = x \cos x - \sin x$. *Velja namreč* $f(x) = g(x)/\cos x$ *in funkciji* f *in* g *imata iste ničle.*

Sedaj nimamo težav s poli. Če izberemo $a = 1000$, $b = 1001$ *in* $\epsilon = 10^{-10}$, *potem se bisekcija konča po* 34 *korakih, ko se začetni interval skrči na* $[1000.59626076458,\ 1000.59626076464]$. *Tako lahko sklepamo, da je ničla na 13 mest natančno enaka* $\alpha = 1000.596260764$. *Pol funkcije* f *pri* α *je enak* $318\pi + \pi/2 = 1000.597260168$. *Vidimo, da se pol in ničla ujemata v prvih* 6 *mestih, kar kaže, kako težko je poiskati dober začetni interval, če delamo direktno s funkcijo* f. □



Slika 2.4: Graf funkcije $f(x) = x\cos x - \sin x$.

2.3 Navadna iteracija

Enačbo $f(x) = 0$ ponavadi rešujemo *iterativno*, kar pomeni, da s ponavljanjem istega postopka dobimo zaporedje približkov, ki ima za limito rešitev enačbe α. Postopku pravimo *iteracija*.

Poseben primer je *navadna iteracija*, kjer enačbo $f(x) = 0$ rešujemo tako, da poiščemo ekvivalentno enačbo oblike $x = g(x)$, kjer je g *iteracijska funkcija*, izberemo začetni približek x_0 in računamo približke s predpisom

$$x_{r+1} = g(x_r)$$

za $r = 0, 1, \ldots$. Da bodo približki res konvergirali proti rešitvi, moramo pravilno izbrati iteracijsko funkcijo in začetni približek. Za iteracijsko funkcijo g mora očitno veljati, da ima negibne točke v ničlah funkcije f, oziroma

$$f(x) = 0 \iff g(x) = x.$$

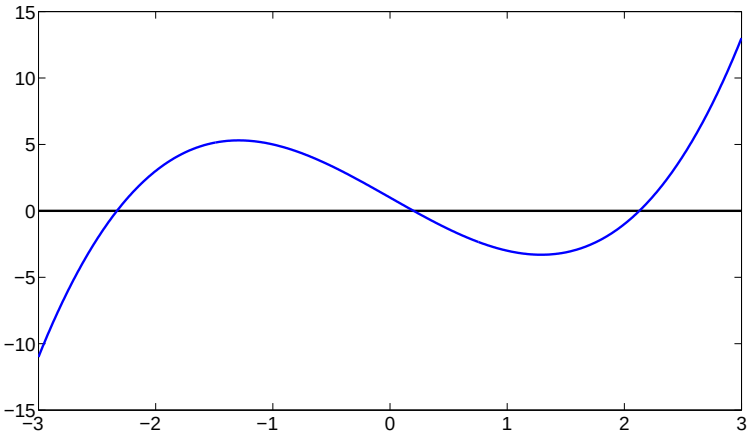
Zgledi iteracijskih funkcij, ki ustrezajo zgornjemu pogoju so npr.:

- $g(x) = x - f(x)$,
- $g(x) = x - Cf(x)$, kjer je $C \neq 0$,
- $g(x) = x - h(x)f(x)$, kjer je $h(x) \neq 0$.

Kot bomo videli v nadaljevanju, samo to še ni zadosten pogoj za konvergenco zaporedja (x_r) proti ničli funkcije f.

Zgled 2.3 *Polinom $p(x) = x^3 - 5x + 1$ ima tri realne ničle: $\alpha_1 = -2.33005874$, $\alpha_2 = 0.20163968$ in $\alpha_3 = 2.12841906$. Poskusimo poiskati različne iteracijske funkcije in preveriti, ali lahko z njimi izračunamo katero izmed ničel polinoma p.*

Enačbo $p(x) = 0$ lahko zapišemo v obliki $5x = x^3 + 1$. Tako dobimo prvo iteracijsko funkcijo $g_1(x) = (x^3 + 1)/5$. Podobno, če $p(x) = 0$ zapišemo v obliki $x^3 = 5x - 1$, dobimo drugo iteracijsko funkcijo



Slika 2.5: Graf polinoma $p(x) = x^3 - 5x + 1$.

$g_2(x) = \sqrt[3]{5x-1}$. *Tretjo dobimo, če* $p(x) = 0$ *predelamo v obliko* $x(x^2 - 5) = -1$, *od koder dobimo* $g_3(x) = 1/(5 - x^2)$.

Vse tri iteracijske funkcije imajo negibne točke v ničlah polinoma p. Ko preizkusimo konvergenco zaporedja približkov, ki jih dobimo z različnimi izbirami x_0 *in zgoraj izpeljanimi iteracijskimi funkcijami, opazimo naslednje:*

- *Če za* g_1 *izberemo* $\alpha_1 < x_0 < \alpha_3$, *potem zaporedje konvergira k* α_2, *pri* $x_0 < \alpha_1$ *in* $x_0 > \alpha_3$ *pa divergira. Tudi če je* x_0 *poljubno blizu* α_1 *oziroma* α_3, *zaporedje nikoli ne konvergira proti* α_1 *oziroma* α_3. *Tu smo namenoma izpustili zelo malo verjeten primer, da že na začetku izberemo* $x_0 = \alpha_1$ *ali* $x_0 = \alpha_3$.
- *Pri* g_2 *je situacija ravno obratna. Če je* $x_0 < \alpha_2$, *zaporedje konvergira k* α_1, *v primeru* $x_0 > \alpha_2$ *pa k* α_3.
- *Pri* g_3 *zaporedje ne glede na začetni približek konvergira k* α_2. *Tudi za* $x_0 = \pm\sqrt{5}$ *pri računanju v aritmetiki, ki podpira standard IEEE, nimamo težav, saj dobimo* $x_1 = \infty$ *in potem* $x_2 = 0$.

Vidimo, da nobena izmed treh iteracijskih funkcij ni dobra za vse tri ničle. V nadaljevanju bomo videli, kaj je vzrok za to. □

V zgledu 2.3 smo videli, da tudi če izberemo začetni približek x_0 poljubno blizu negibne točke, to še ni zagotovilo, da bo zaporedje res konvergiralo proti tej negibni točki. Naslednji izrek pove, da je konvergenca zagotovljena, če je iteracijska funkcija v okolici negibne točke skrčitev.

Izrek 2.1 *Naj bo* $\alpha = g(\alpha)$ *in naj iteracijska funkcija g na intervalu* $I = [\alpha - \delta, \alpha + \delta]$ *zadošča Lipschitzovemu*[1] *pogoju*

$$|g(x) - g(y)| \leq m|x - y|$$

za poljubna $x, y \in I$ *in konstanto* $0 \leq m < 1$. *Potem za vsak* $x_0 \in I$ *zaporedje*

$$x_{r+1} = g(x_r), \quad r = 0, 1, \ldots$$

[1] Nemški matematik Rudolf Otto Sigismund Lipschitz (1832–1903).

konvergira k α. Veljata tudi oceni

$$|x_r - \alpha| \leq m^r |x_0 - \alpha| \tag{2.1}$$

in

$$|x_{r+1} - \alpha| \leq \frac{m}{1-m} |x_r - x_{r-1}|. \tag{2.2}$$

Dokaz. Naj bo $\epsilon_r = x_r - \alpha$ napaka približka x_r. Velja

$$|\epsilon_r| = |x_r - \alpha| = |g(x_{r-1}) - g(\alpha)| \leq m|x_{r-1} - \alpha| = m|\epsilon_{r-1}|. \tag{2.3}$$

To lahko nadaljujemo in dobimo

$$|\epsilon_r| \leq m|\epsilon_{r-1}| \leq m^2|\epsilon_{r-2}| \leq \cdots \leq m^r|\epsilon_0|.$$

Tako smo izpeljali oceno (2.1), iz katere je očitno, da zaporedje (x_r) konvergira proti α.

Razliko $x_{r+1} - \alpha$ ocenimo z

$$|x_{r+1} - \alpha| \leq |x_{r+1} - x_{r+2}| + |x_{r+2} - x_{r+3}| + \cdots.$$

Sedaj upoštevamo, da je

$$|x_{r+k} - x_{r+k+1}| = |g(x_{r+k-1}) - g(x_{r+k})| \leq m|x_{r+k-1} - x_{r+k}| \leq \cdots \leq m^k|x_{r-1} - x_r|$$

in dobimo

$$|x_{r+1} - \alpha| \leq (m + m^2 + \cdots)|x_{r-1} - x_r| = \frac{m}{1-m}|x_{r-1} - x_r|. \qquad \blacksquare$$

Opomba 2.1 *Pomen ocene (2.2) je, da povezuje napako tekočega približka z razliko med zadnjima dvema približkoma, le to pa v praksi lahko izračunamo. Pove nam, da bo v primeru, ko se zadnja dva približka ne razlikujeta dosti, zadnji približek blizu točni rešitvi, če le Lipschitzova konstanta m ni zelo blizu* 1.

Posledica 2.2 *Naj bo iteracijska funkcija g zvezno odvedljiva v negibni točki α in naj velja $|g'(\alpha)| < 1$. Potem obstaja taka okolica I za α, da za vsak $x_0 \in I$ zaporedje*

$$x_{r+1} = g(x_r), \quad r = 0, 1, \ldots$$

konvergira k α.

Dokaz. Zaradi $|g'(\alpha)| < 1$ in zveznosti odvoda obstajata tak $\delta > 0$ in taka konstanta $m < 1$, da je $|g'(x)| \leq m < 1$ za $x \in I$, kjer je interval $I = [\alpha - \delta, \alpha + \delta]$. Za poljubna $x, y \in I$ potem po Lagrangeevem izreku velja $|g(x) - g(y)| \leq |g'(\xi)| \cdot |x - y|$, kjer je ξ točka med x in y in je $|g'(\xi)| \leq m$. Torej je funkcija g Lipschitzova na intervalu I in konvergenca sledi iz izreka 2.1. $\blacksquare$

Podobno lahko pokažemo, da v primeru $|g'(\alpha)| > 1$ ne moremo dobiti nobene okolice, znotraj katere bi za vsak začetni približek x_0 imeli konvergenco. V tem primeru za x_r blizu α velja $|\epsilon_{r+1}| = |g'(\xi_r)| \cdot |\epsilon_r| > |\epsilon_r|$ in nimamo konvergence.

Zgled 2.4 *Kaj se dogaja, če je* $|g'(\alpha)| = 1$*? Poglejmo si dve iteracijski funkciji za reševanje enačbe* $x - \tan x = 0$*, kjer želimo izračunati rešitev* $\alpha = 0$*. Za prvo iteracijsko funkcijo vzamemo* $g_1(x) = \tan x$*, za drugo pa* $g_2(x) = \arctan x$*.*

Odvoda iteracijskih funkcij sta $g_1'(x) = 1/(\cos^2 x)$ *in* $g_2'(x) = 1/(1+x^2)$*. Oba odvoda sta pri* $x = 0$ *enaka* 1*, opazimo pa tudi, da za* $x \neq 0$ *v majhni okolici točke* 0 *velja* $|g_1'(x)| > 1$ *in* $|g_2'(x)| < 1$*. Zaradi tega, če vzamemo* $x_0 \neq 0$ *blizu* 0*, dobimo po (2.3)* $|g_1(x_0)| > |x_0|$ *in* $|g_2(x_0)| < |x_0|$*. Zaporedje generirano z* g_1 *divergira, zaporedje generirano z* g_2 *pa konvergira k* 0*, a je njegova konvergenca vedno počasnejša, bližje ko smo* 0*, tako da je zaporedje v praksi povsem neuporabno. Če vzamemo* $x_0 = 0.1$*, potem pri iteraciji z* g_2 *po* 10^3 *korakih dobimo* $3.61 \cdot 10^{-2}$*, po* 10^6 *korakih* $1.22 \cdot 10^{-4}$*, po* 10^9 *korakih pa* $3.87 \cdot 10^{-5}$*.* □

Če je iteracijska funkcija zvezno odvedljiva, potem o konvergenci v bližini korena α odloča število $g'(\alpha)$. Če je $|g'(\alpha)| < 1$, potem je α *privlačna točka* in izrek zagotavlja konvergenco v neki okolici okrog α. V primeru $|g'(\alpha)| > 1$ je α *odbojna točka*.

Zgled 2.5 *Poglejmo odvode iteracijskih funkcij iz zgleda 2.3 v negibnih točkah. Vrednosti so:*

$$\begin{array}{lll} g_1'(\alpha_1) = 3.26, & g_1'(\alpha_2) = 0.024, & g_1'(\alpha_3) = 2.72, \\ g_2'(\alpha_1) = 0.31, & g_2'(\alpha_2) = 40.99, & g_2'(\alpha_3) = 0.37, \\ g_3'(\alpha_1) = -25.30, & g_3'(\alpha_2) = 0.016, & g_3'(\alpha_3) = 19.28. \end{array}$$

To se ujema z ugotovitvami iz zgleda 2.3, ko smo z iteracijskimi funkcijami lahko izračunali ravno tiste negibne točke, za katere je absolutna vrednost odvoda iteracijske funkcije manjša od 1*.* □

Različne iteracijske funkcije pri enakih začetnih približkih različno hitro konvergirajo proti rešitvi. Pri računanju ničel funkcij je glavno delo izračun vrednosti funkcije (oziroma, kot bomo videli pri nekaterih metodah, še izračun vrednosti odvodov), zato želimo s čim manj izračuni vrednosti funkcije priti do čim boljšega približka. Zato je pomembno vedeti, kako hitro numerična metoda konvergira proti rešitvi.

Definicija 2.3 *Denimo, da zaporedje* (x_r) *konvergira proti* α*. Pravimo, da je red konvergence enak* p*, če obstajata taki konstanti* $C_1, C_2 > 0$*, da za vse dovolj pozne člene velja*

$$C_1|x_r - \alpha|^p \leq |x_{r+1} - \alpha| \leq C_2|x_r - \alpha|^p. \tag{2.4}$$

Ekvivalentna definicija je, da je red konvergence enak p*, če obstaja taka konstanta* $C > 0$*, da velja*

$$\lim_{r\to\infty} \frac{|x_{r+1} - \alpha|}{|x_r - \alpha|^p} = C. \tag{2.5}$$

Red konvergence p pomeni, da je napaka približka x_{r+1} velikostnega razreda $|x_r - \alpha|^p$. To tudi pomeni, da se število točnih decimalk obnaša kot funkcija števila korakov na potenco p.

Očitno je najmanjši možni red konvergence enak 1, saj se v primeru, ko velja (2.4) za $0 < p < 1$, da hitro poiskati tak $\delta > 0$, da iz $0 < |x_r - \alpha| < \delta$ sledi $|x_{r+1} - \alpha| > |x_r - \alpha|$, to pa pomeni, da (x_r) ni konvergentno zaporedje.

V primeru navadne iteracije lahko, če je iteracijska funkcija dovoljkrat zvezno odvedljiva, red konvergence določimo z odvajanjem. O tem govori naslednja lema.

Lema 2.4 *Naj bo iteracijska funkcija g v okolici negibne točke α p-krat zvezno odvedljiva in naj velja $|g'(\alpha)| < 1$, $g^{(k)}(\alpha) = 0$ za $k = 1, \ldots, p-1$ in $g^{(p)}(\alpha) \neq 0$. Potem ima iterativna metoda $x_{r+1} = g(x_r)$, $r = 0, 1, \ldots$, v bližini rešitve α red konvergence p.*

Dokaz. Uporabimo razvoj funkcije g v Taylorjevo vrsto okrog točke α. Tako dobimo

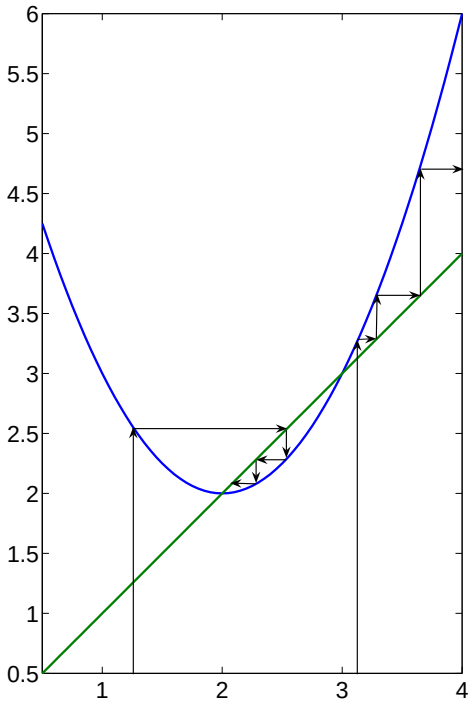
$$x_{r+1} = g(x_r) = \alpha + \frac{1}{p!}(x_r - \alpha)^p g^{(p)}(\xi),$$

kjer je ξ med x_r in α. Če ocenimo absolutno vrednost p-tega odvoda funkcije g v okolici točke α navzdol in navzgor, dobimo konstanti C_1 in C_2 iz ocene (2.4), medtem ko za konstanto C iz (2.5) velja $C = \frac{1}{p!}g^{(p)}(\alpha)$. ∎

Posebni primeri konvergence so:

- $p = 1$: *linearna* (v vsakem koraku pridobimo konstantno število novih točnih decimalk),
- $p = 2$: *kvadratična* (število točnih decimalk se v vsakem koraku podvoji),
- $p = 3$: *kubična* (število točnih decimalk se v vsakem koraku potroji),
- $1 < p < 2$: *superlinearna* (hitrejša od linearne in počasnejša od kvadratične).

Zgled 2.6 *Poglejmo, kaj se dogaja pri navadni iteraciji $g(x) = x^2 - 4x + 6$.*



Slika 2.6: Graf iteracijske funkcije $x^2 - 4x + 6$. Pri začetnem približku $x_0 = 1.25$ zaporedje konvergira k negibni točki 2, pri začetnem približku $x_0 = 3.15$ pa divergira.

Iz slike 2.6 opazimo, da imamo v primeru $x_0 \in (1, 3)$ konvergenco proti 2, sicer pa za $x_0 < 1$ in $x_0 > 3$ zaporedje divergira. To lahko pokažemo tudi računsko, saj velja

$$|x_{r+1} - 2| = (x_r - 2)^2.$$

Konvergenca je v bližini točke 2 *kvadratična, saj je* $g'(2) = 0$ *in* $g''(2) \neq 0$. *Posledica 2.2 zagotavlja konvergenco le za* $x_0 \in (\frac{3}{2}, \frac{5}{2})$, *saj je tu* $|g'(x)| < 1$, *v praksi pa imamo konvergenco na širšem območju.* □

Zgled 2.7 *Za enačbo* $x^2 + \ln x = 0$, *ki ima rešitev* $\alpha = 0.6529186$, *primerjajmo naslednje tri iteracijske funkcije:*

- $g_1(x) = e^{-x^2}$, $g_1'(\alpha) = -0.853$, *red konvergence je* 1,
- $g_2(x) = \frac{1}{2}(x + e^{-x^2})$, $g_2'(\alpha) = 0.074$, *red konvergence je* 1,
- $g_3(x) = \frac{2x^3 + e^{-x^2}}{1+2x^2}$, $g_3'(\alpha) = 0$, $g_3''(\alpha) = 2.716$, *red konvergence je* 2.

Za različne začetne približke potrebujemo različno število korakov dokler se zaporedna približka ne razlikujeta za manj kot 10^{-12}. *Rezultati so v spodnji tabeli.*

x_0	0.6	0.7	1	10	100
g_1	163	162	173	175	175
g_2	14	14	15	18	21
g_3	5	5	7	109	10013

Red konvergence vedno velja le v bližini ničle, daleč stran pa se metoda lahko obnaša povsem drugače. Tako lahko opazimo, da metoda, ki ima v bližini rešitve kvadratično konvergenco, daleč stran od rešitve konvergira zelo počasi, celo počasneje od metod, ki imajo v okolici ničle linearno konvergenco. Ponavadi metode z visokim redom konvergence konvergirajo zelo hitro le v majhni okolici. Če nimamo dobrega približka, je bolje najprej uporabiti metodo z linearnim redom konvergence, potem pa, ko smo dovolj blizu, preiti na metodo z višjim redom konvergence. □

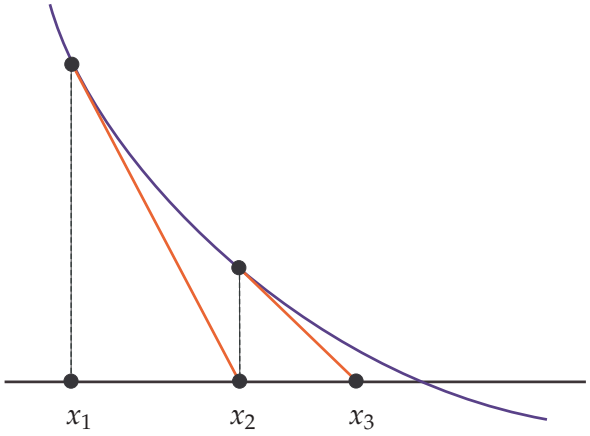
2.4 Tangentna metoda

Geometrijska ideja *tangentne* oz. *Newtonove metode*[2] je predstavljena na sliki 2.7. Funkcijo f aproksimiramo s tangento v točki $(x_r, f(x_r))$ in potem kot približek za ničlo funkcije f vzamemo ničlo tangente, torej presečišče tangente z osjo x. Ker smo funkcijo samo aproksimirali s tangento, je ničla tangente le približek za ničlo funkcije, zato postopek ponavljamo.

Enačba tangente skozi $(x_r, f(x_r))$ je $y = f(x_r) + f'(x_r)(x - x_r)$. Od tod izpeljemo, da je presečišče tangente z x-osjo v točki $x_r - f(x_r)/f'(x_r)$. To vzamemo za naslednjo točko x_{r+1} in postopek ponavljamo: za $r = 0, 1, \ldots$

$$x_{r+1} = x_r - \frac{f(x_r)}{f'(x_r)}.$$

[2]Metoda je znana tudi kot Newton-Raphsonova metoda. V resnici je sir Isaac Newton (1643–1727) metodo leta 1669 opisal precej drugače kot jo poznamo sedaj. Tako npr. sploh ni uporabljal odvodov in jo je uporabljal le za iskanje ničel polinomov. Newtonovo metodo je poenostavil Raphson leta 1690, prvič pa jo je v obliki z odvodom, kot jo poznamo sedaj, zapisal Simpson leta 1740, ki je tudi prvi posplošil metodo na nelinearne sisteme (glej razdelek 4.3). Več o zgodovini Newtonove metode lahko poiščete npr. v [33].



Slika 2.7: Geometrijska izpeljava tangentne metode.

Analitična izpeljava tangentne metode poteka preko razvoja v Taylorjevo vrsto. Naj bo x_r približek za ničlo funkcije f. Če upoštevamo le prva dva člena v razvoju $f(x_r + h)$ v Taylorjevo vrsto

$$f(x_r + h) = f(x_r) + f'(x_r)h + \frac{f''(x_r)h^2}{2} + \cdots,$$

dobimo

$$f(x_r + h) \approx f(x_r) + f'(x_r)h.$$

Če predpostavimo, da je $f'(x_r) \neq 0$, potem iz enačbe $f(x_r + h) = 0$ dobimo $h \approx -f(x_r)/f'(x_r)$.

Tangentna metoda je poseben primer navadne iteracije, kjer za iteracijsko funkcijo vzamemo

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

torej lahko red konvergence ugotovimo iz odvodov iteracijske funkcije. Prvi odvod je

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

Sedaj ločimo dva primera:

a) Če je α enostavna ničla, potem je $g'(\alpha) = 0$ in konvergenca tangentne metode je vsaj kvadratična. Velja

$$g''(\alpha) = \frac{f''(\alpha)}{f'(\alpha)},$$

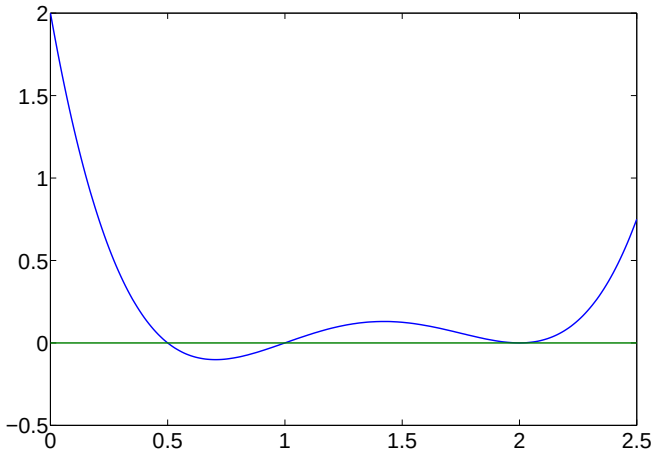
torej je pri $f''(\alpha) \neq 0$ konvergenca kvadratična, sicer pa vsaj kubična.

b) Če je α m-kratna ničla, potem se da pokazati, da je

$$\lim_{x \to \alpha} g'(x) = 1 - \frac{1}{m}.$$

Od tod sledi, da je konvergenca linearna.

Konvergenca tangentne metode je torej v standardnem primeru kvadratična (ničla je enostavna in $f''(\alpha) \neq 0$), v primeru večkratne ničle je linearna, lahko pa je tudi kubična.



Slika 2.8: Graf polinoma $p(x) = (x - \frac{1}{2})(x-1)(x-2)^2$.

	$x_0 = 0.6$	$x_0 = 1.1$	$x_0 = 2.1$
x_1	0.43529411764706	0.99411764705883	2.05356200527705
x_2	0.48933773025139	1.00000122881825	2.02787668535119
x_3	0.49963948674988	1.00000000000000	2.01424749381787
x_4	0.49999956751570	1.00000000000000	2.00720632682115
x_5	0.49999999999938		2.00362453954375
x_6	0.50000000000000		2.00181771012143
x_7	0.50000000000000		2.00091022751549
$\vdots$			$\vdots$
x_{20}			2.00000011182657
x_{21}			2.00000006946678
x_{22}			2.00000001832416
x_{23}			2.00000001832416

Tabela 2.1: Približki, izračunani s tangentno metodo za polinom $p(x) = (x - \frac{1}{2})(x-1)(x-2)^2$ z začetnimi približki 0.6, 1.1 in 2.1.

Zgled 2.8 *Poglejmo polinom* $p(x) = (x - \frac{1}{2})(x-1)(x-2)^2$. *Če vzamemo začetne približke* 0.6, 1.1 *in* 2.1, *lahko iz tabele 2.1 razberemo, da imamo pri* 0.6 *kvadratično konvergenco, pri* 1.1 *kubično konvergenco, pri* 2.1 *pa linearno konvergenco. To se ujema s teorijo, saj je iz grafa na sliki 2.8 razvidno, da ima polinom v točki* 0.5 *enostavno ničlo, pri* 1 *enostavno ničlo in prevoj,* 2 *pa je dvojna ničla. Opazimo tudi, da dvojna ničla ni izračunana z enako natančnostjo kot ostali dve, čeprav se pri računanju v dvojni natančnosti zadnja dva približka ne razlikujeta več. To ni nič nenavadnega, saj smo že v razdelku 2.1 omenili, da večkratnih ničel ni moč izračunati s polno natančnostjo. Pri dvojni ničli lahko tako praviloma natančno izračunamo le polovico razpoložljivih decimalk, kar se ujema z rezultati v tabeli 2.1.* □

Konvergenčni izrek za tangentno metodo pravi, da so vse enostavne ničle privlačne točke. Če imamo dovolj dober približek, bodo približki dobljeni s tangentno metodo konvergirali k ničli funkcije.

Izrek 2.5 *Naj bo* α *enostavna ničla dvakrat zvezno odvedljive funkcije* f. *Potem obstajata okolica* I *točke*

α in konstanta C, da tangentna metoda konvergira za vsak $x_0 \in I$ in da približki x_r zadoščajo oceni

$$|x_{r+1} - \alpha| \leq C(x_r - \alpha)^2.$$

Dokaz. Označimo napako r-tega približka z $e_r = x_r - \alpha$. Iz razvoja

$$0 = f(\alpha) = f(x_r) + f'(x_r)(\alpha - x_r) + \frac{f''(\eta_r)}{2}(\alpha - x_r)^2,$$

kjer je η_r med α in x_r, dobimo

$$x_{r+1} - \alpha = \frac{f''(\eta_r)}{2f'(x_r)}(x_r - \alpha)^2.$$

To pomeni

$$e_{r+1} = \frac{f''(\eta_r)}{2f'(x_r)}e_r^2. \tag{2.6}$$

Naj bo

$$C(\delta) = \frac{\max_{|x-\alpha|\leq\delta} |f''(x)|}{2\min_{|x-\alpha|\leq\delta} |f'(x)|}.$$

Ker je $C(0) = \frac{|f''(\alpha)|}{2|f'(\alpha)|}$, obstaja tak δ_0, da za vsak $0 \leq \delta \leq \delta_0$ velja $\delta C(\delta) < 1$. Vzamemo $I = [\alpha - \delta_0, \alpha + \delta_0]$. Za vsak $x_r \in I$ dobimo $|e_{r+1}| \leq C(\delta_0)e_r^2 \leq C(\delta_0)\delta_0|e_r| < |e_r|$. Torej po eni strani vsi približki ostanejo v I, po drugi strani pa velja $\lim_{r\to\infty} x_r = \alpha$. ■

Pri določenih pogojih lahko dokažemo tudi globalno konvergenco iz poljubne začetne točke.

Izrek 2.6 *Naj bo f na intervalu $I = [a, \infty)$ dvakrat zvezno odvedljiva, naraščajoča in konveksna funkcija, ki ima ničlo $\alpha \in I$. Potem je α edina ničla f na I in za vsak $x_0 \in I$ tangentna metoda konvergira k α.*

Dokaz. Ker je f naraščajoča, je očitno $f' > 0$ in α je edina ničla na I. Konveksnost pomeni $f'' > 0$ na I. Iz (2.6) sledi $e_{r+1} > 0$, kar pomeni, da je $\alpha \leq x_r$ za $r \geq 1$. Od tod za $r \geq 1$ sledi $\alpha \leq x_{r+1} \leq x_r$, saj je $f(x_r) \geq 0$. ■

V uporabni oceni za napako približka nastopa razlika zadnjih dveh približkov. To je količina, ki jo poznamo, in če vemo kaj o prvem in drugem odvodu funkcije f, potem lahko ocenimo napako zadnjega približka.

Naj bo $d_{r+1} = x_{r+1} - x_r$ razlika zadnjih dveh približkov. Na intervalu I naj velja $|f'(x)| \geq m_1 > 0$ in $|f''(x)| \leq M_2$. Iz razvoja

$$f(x_r) = f(\alpha) + (x_r - \alpha)f'(\xi_r),$$

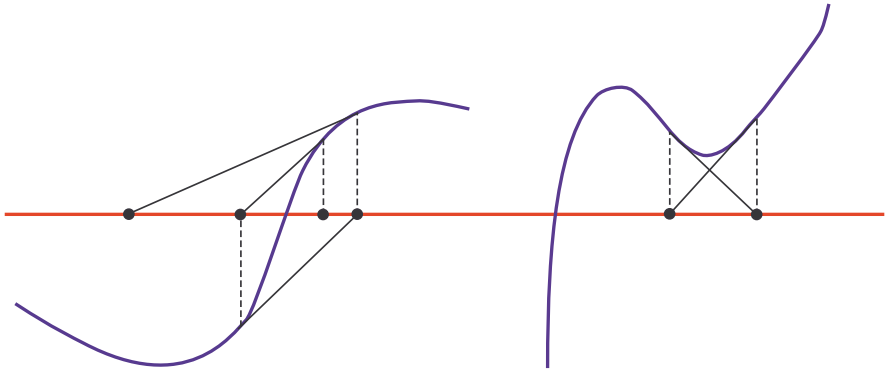
kjer leži ξ_r med α in x_r, dobimo $f(x_r) = e_r f'(\xi_r)$, odtod pa

$$d_{r+1} = x_{r+1} - x_r = -\frac{f(x_r)}{f'(x_r)} = -\frac{f'(\xi_r)}{f'(x_r)}e_r.$$

To pomeni, da lahko ocenimo $|e_r| \leq \frac{|f'(x_r)|}{m_1}|d_{r+1}|$. Iz ocene (2.6) sedaj dobimo

$$|e_{r+1}| \leq \frac{M_2|f'(x_r)|}{2m_1^2}d_{r+1}^2.$$

Izreki nam zagotavljajo konvergenco tangentne metode dovolj blizu ničle ali pa za funkcijo prave oblike. Sicer pa imamo lahko zelo različno obnašanje in slepa uporaba tangentne metode ni priporočljiva. Primera na sliki 2.9 kažeta divergenco, ko nismo dovolj blizu ničle in zaciklanje približkov.



Slika 2.9: Divergenca tangentne metode ko začetni približek ni dovolj blizu (levo) in zaciklanje tangentne metode (desno).

Tangentno metodo lahko uporabljamo tudi za računanje kompleksnih ničel analitičnih funkcij, saj tudi zanje velja izpeljava preko razvoja v Taylorjevo vrsto. Izkaže se, da tudi v tem primeru za naslednji približek obstaja geometrijska razlaga.

Naj bo f analitična funkcija in naj velja $f'(z_0) \neq 0$. Če označimo ploskev $w(z) = |f(z)|$ nad kompleksno ravnino, potem je kompleksno število

$$z_1 = z_0 - \frac{f(z_0)}{f'(z_0)},$$

ki ga dobimo po enem koraku tangentne metode, presečišče dveh premic:

- presečišča tangentne ravnine na ploskev $w = |w(z)|$ v točki $(z_0, w(z_0))$ z ravnino $w = 0$,
- projekcije gradienta na ploskev $w = |w(z)|$ v točki $(z_0, w(z_0))$ na ravnino $w = 0$.

Podroben dokaz te zveze lahko najdete v [4].

Pri iskanju kompleksnih rešitev moramo seveda uporabiti kompleksne začetne približke. Če imamo npr. polinom z realnimi koeficienti in iščemo kompleksno ničlo, potem moramo uporabiti kompleksni začetni približek, sicer bodo vsi približki realni.

2.5 Ostale metode

Obstaja še polno drugih numeričnih metod za reševanje nelinearnih enačb. Nekatere imajo prednost pred tangentno metodo v tem, da ne potrebujejo odvodov, imajo pa po drugi strani

zaradi tega lahko počasnejšo konvergenco. Spet druge metode uporabljajo še višje odvode in imajo zato lahko višji red konvergence od kvadratičnega. Opisali bomo le nekaj najbolj znanih metod.

2.5.1 Sekantna metoda

Težava s tangentno metodo je, da za izračun x_{r+1} poleg vrednosti funkcije $f(x_r)$ potrebujemo še vrednost odvoda $f'(x_r)$. Ker ta ni vedno na voljo ali pa je izračun odvoda lahko zelo zahteven, ga lahko poskusimo aproksimirati z diferenčnim kvocientom

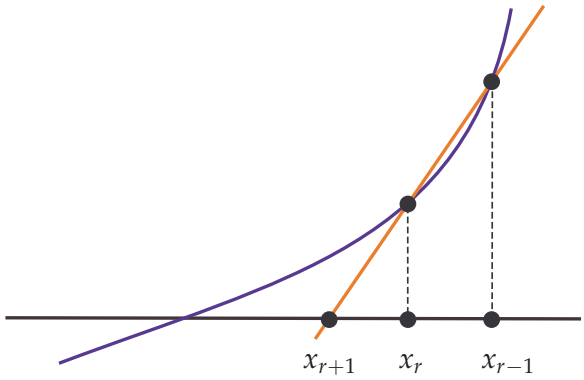
$$\frac{f(x_r) - f(x_{r-1})}{x_r - x_{r-1}}$$

skozi zadnja dva približka.

Tako dobimo *sekantno metodo*. Tako kot smo pri tangentni metodi funkcijo v bližini ničle aproksimirali s tangento, jo sedaj aproksimiramo s sekanto skozi točki $(x_{r-1}, f(x_{r-1}))$ in $(x_r, f(x_r))$, kot je predstavljeno na sliki 2.10. Na začetku potrebujemo dva začetna približka x_0 in x_1, za katera pa ni tako kot pri bisekciji nujno, da je funkcija v njiju različno predznačena. Naslednje približke izračunamo po rekurzivni formuli

$$x_{r+1} = x_r - \frac{f(x_r)(x_r - x_{r-1})}{f(x_r) - f(x_{r-1})}$$

za $r = 1, 2, \ldots$. V vsakem koraku (razen na začetku) moramo izračunati le eno vrednost funkcije f, saj vrednost $f(x_{r-1})$ poznamo še iz prejšnjega koraka.



Slika 2.10: Sekantna metoda.

En korak sekantne metode je torej cenejši od enega koraka tangentne metode, saj ne potrebujemo vrednosti odvoda. Kako pa je z redom konvergence? Sekantna metoda ne spada med navadno iteracijo, saj naslednji približek ni odvisen le od zadnjega temveč od zadnjih dveh približkov. Zaradi tega je postopek za določitev reda konvergence malce bolj zapleten in bomo navedli le glavne korake. Podrobno izpeljavo lahko najdete npr. v [4].

Naj bo α enostavna ničla in funkcija f dvakrat zvezno odvedljiva . Če označimo z $e_r = x_r - \alpha$ napako r-tega približka, potem se da pokazati, da za napake velja

$$\lim_{r\to\infty} \frac{|e_{r+1}|}{|e_r| \cdot |e_{r-1}|} = \frac{|f''(\alpha)|}{2|f'(\alpha)|}. \tag{2.7}$$

Za primerjavo, pri tangentni metodi velja $\lim_{r\to\infty} \frac{|e_{r+1}|}{|e_r|^2} = \frac{|f''(\alpha)|}{2|f'(\alpha)|}$. Denimo sedaj, da ima sekantna metoda red konvergence p, torej obstaja taka neničelna konstanta C, da je

$$\lim_{r\to\infty} \frac{|e_{r+1}|}{|e_r|^p} = C.$$

Ko to vstavimo v (2.7), dobimo

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

To je možno le, če je $p^2 - p - 1 = 0$, torej $p = (1 \pm \sqrt{5})/2$. V poštev pride le možnost $p = (1 + \sqrt{5})/2 = 1.62$, saj mora biti red konvergence vsaj 1.

Sekantna metoda ima tako superlinearno konvergenco. Obnašanje sekantne metode je zelo podobno obnašanju tangentne metode. Tudi tu velja, da bomo imeli konvergenco, če bomo imeli dva zadosti dobra približka.

Če je računanje vrednosti odvoda približno enake zahtevnosti kot računanje vrednosti funkcije, potem sta dva koraka sekantne metode toliko zahtevna kot en korak tangentne metode. Pri tangentni metodi se v enem koraku število točnih decimalk podvoji, pri sekantni metodi pa se v dveh korakih pomnoži s faktorjem 2.6. Zaradi tega s sekantno metodo praviloma lahko izračunamo ničlo hitreje kot s tangentno metodo.

Denimo, da pri tangentni metodi število točnih decimalk narašča v zaporedju

$$1, \quad 2, \quad 4, \quad 8, \quad 16.$$

Potem lahko pričakujemo, da bo pri sekantni metodi število točnih decimalk naraščalo v zaporedju

$$1, \quad 1.6, \quad 2.6, \quad 4.2, \quad 6.9, \quad 11.1, \quad 17.9.$$

Za maksimalno natančnost v dvojni natančnosti tako pri tangentni metodi potrebujemo 4 korake in izračun $4f$ in $4f'$. Pri sekantni metodi porabimo 6 korakov in izračun $7f$. Glede na porabljeno delo je sekantna metoda v tem primeru boljša od tangentne.

2.5.2 Mullerjeva metoda

Pri sekantni metodi funkcijo aproksimiramo s premico, ki jo določata vrednosti funkcije f v zadnjih dveh približkih x_r in x_{r-1}, potem pa za naslednji približek x_{r+1} vzamemo presečišče te premice z osjo x. Posplošitev te metode je *Mullerjeva metoda*[3] kjer funkcijo aproksimiramo s parabolo, ki gre skozi zadnje tri točke $(x_r, f(x_r))$, $(x_{r-1}, f(x_{r-1}))$, $(x_{r-2}, f(x_{r-2}))$, potem pa za naslednji približek x_{r+1} vzamemo ničlo te parabole. Hitro lahko razmislimo, da bomo razen v prvem koraku tudi pri Mullerjevi metodi za en korak metode potrebovali le en izračun vrednosti funkcije f.

Če ima parabola dve ničli, vzamemo tisto, ki je bližja zadnjemu približku x_r. Lahko se zgodi, da ima parabola kompleksni ničli, čeprav so vse točke x_{r-2}, x_{r-1}, x_r realne, in v tem primeru

[3] Američan David E. Muller je metodo objavil leta 1956. Muller je znan tudi po asinhronem logičnem elementu, ki se uporablja v asinhronih procesorjih.

bo približek x_{r+1} kompleksen. Zaradi tega je Mullerjeva metoda primerna za računanje kompleksnih ničel analitičnih funkcij, saj metoda lahko sama iz začetnih realnih približkov preide v kompleksne približke in tako skonvergira do kompleksne ničle. To tudi pomeni, da moramo računati s kompleksno aritmetiko.

Kaj podobnega se npr. pri tangentni ali sekantni metodi ne more zgoditi. Če imamo npr. polinom z realnimi koeficienti, potem iz začetnih realnih približkov pri tangentni ali sekantni metodi ne moremo dobiti kompleksnega približka, pri Mullerjevi metodi pa lahko.

Če interpolacijski polinom skozi zadnje tri točke zapišemo kot $p(x) = a(x - x_r)^2 + b(x - x_r) + c$, potem se da za njegove koeficiente izpeljati naslednje formule

$$\begin{aligned} a &= \frac{(x_{r-1} - x_r)(f(x_{r-2}) - f(x_r)) - (x_{r-2} - x_r)(f(x_{r-1}) - f(x_r))}{(x_{r-2} - x_r)(x_{r-1} - x_r)(x_{r-2} - x_{r-1})}, \\ b &= \frac{(x_{r-2} - x_r)^2(f(x_{r-1}) - f(x_r)) - (x_{r-1} - x_r)^2(f(x_{r-2}) - f(x_r))}{(x_{r-2} - x_r)(x_{r-1} - x_r)(x_{r-2} - x_{r-1})}, \\ c &= f(x_r). \end{aligned}$$

Sledi

$$x_{r+1} = x_r - \frac{2c}{b \pm \sqrt{b^2 - 4ac}},$$

rešitev bližjo x_r pa dobimo, če pred kvadratnim korenom izberemo predznak tako, da bo imenovalec po absolutni vrednosti čim večji.

Red konvergence Mullerjeve metode je 1.84. Podobno kot pri sekantni metodi lahko pokažemo, da pri Mullerjevi metodi za napake $\epsilon_r = x_r - \alpha$ v bližini ničle velja $|\epsilon_{r+1}| \approx C|\epsilon_r| \cdot |\epsilon_{r-1}| \cdot |\epsilon_{r-2}|$. Zaradi tega se število točnih decimalk d_r širi po diferenčni formuli $d_{r+1} = d_r + d_{r-1} + d_{r-2}$, absolutna vrednost največje ničle karakterističnega polinoma te diferenčne enačbe pa je 1.84.

Zgled 2.9 *Če za funkcijo vzamemo $f(x) = e^x + 1$ in za začetne približke $x_0 = 0$, $x_1 = 0.1$, $x_2 = 0.2$, potem Mullerjeva metoda vrne*

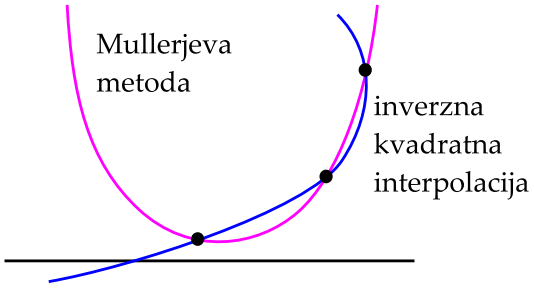
$$\begin{aligned} x_3 &= -0.9008331945 + 1.6747640462i \\ x_4 &= -0.9829710651 + 2.5610505377i \\ x_5 &= -0.5275637519 + 3.3507004256i \\ x_6 &= 0.0694667782 + 3.2426055442i \\ x_7 &= -0.0048737648 + 3.1328657324i \\ x_8 &= 0.0000151703 + 3.1414908145i \\ x_9 &= 0.0000000173 + 3.1415926662i \\ x_{10} &= 0.0000000000 + 3.1415926536i \end{aligned}$$

in tako pravilno izračuna, da je ničla pri $x = \pi i$. □

Idejo bi lahko posplošili še naprej. Skozi zadnjih $(k+1)$-točk $(x_r, f(x_r)),\ldots,(x_{r-k}, f(x_{r-k}))$ napeljemo polinom p stopnje k, potem pa za x_{r+1} vzamemo tisto ničlo polinoma p, ki je najbližja x_r. Vse takšne posplošitve imajo superlinearen red konvergence, večji ko je k, hitrejša je konvergenca. Bolj natančno je red enak največji rešitvi enačbe $\lambda^k = \lambda^{k-1} + \lambda^{k-2} + \cdots + 1$, ta pa je vedno manjša od 2. V praksi se ta princip v glavnem uporablja le za $k = 1$ (sekantna metoda) in $k = 2$ (Mullerjeva metoda), saj sicer računanje ničel interpolacijskega polinoma ni trivialno.

2.5.3 Inverzna interpolacija

Pri Mullerjevi metodi funkcijo f aproksimiramo s parabolo skozi zadnje tri točke $(x_r, f(x_r))$, $(x_{r-1}, f(x_{r-1}))$ in $(x_{r-2}, f(x_{r-2}))$. Lahko pa zamenjamo vlogi spremenljivk x in y ter poiščemo kvadratni polinom p, za katerega velja $p(f(x_r)) = x_r$, $p(f(x_{r-1})) = x_{r-1}$ in $p(f(x_{r-2})) = x_{r-2}$. Na ta način interpoliramo inverzno funkcijo s parabolo in za naslednji približek vzamemo $x_{r+1} = p(0)$. Opisana metoda se imenuje *inverzna kvadratna interpolacija*, red konvergence pa je tako kot pri Mullerjevi metodi enak 1.84. Razlika med Mullerjeva metodo in inverzno kvadratno interpolacijo je prikazana na sliki 2.11.



Slika 2.11: Primerjava inverzne kvadratne interpolacije in Mullerjeve metode.

Za razliko od Mullerjeve metode imamo pri inverzni interpolaciji vedno natanko eno presečišče z x-osjo, ki ga vzamemo za naslednjo točko. Pri inverzni interpolaciji tudi ni težav s povečevanjem stopnje polinoma, saj potrebujemo le vrednost interpolacijskega polinoma v točki 0 in ni potrebno izračunati njegovih ničel. Se pa večinoma uporablja le kvadratna interpolacija, saj pri uporabi več kot treh zadnjih točk ne pridobimo veliko pri redu konvergence. Če bi vzeli samo zadnji dve točki, potem je očitno, da v tem primeru inverzna interpolacija ni nič drugega kot sekantna metoda.

Razlika med Mullerjevo metodo in inverzno kvadratno interpolacijo je tudi ta, da pri inverzni kvadratni interpolaciji v primeru realne funkcije iz realnih začetnih približkov vedno dobimo realen približek. Zaradi tega ni tako primerna za računanje kompleksnih ničel kot je Mullerjeva metoda.

2.5.4 Kombinirane metode

S kombiniranjem različnih metod lahko dosežemo robustnost in hitro konvergenco. Če imamo tak začetni interval $[a, b]$, da je $f(a) \cdot f(b) < 0$, potem lahko npr. kombiniramo bisekcijo s hitrejšimi metodami. Glavna ideja je, da najprej izračunamo nov približek z metodo, ki konvergira hitreje kot bisekcija. Če novi približek leži na intervalu (a, b), ga obdržimo in ustrezno zmanjšamo interval, sicer pa uporabimo bisekcijo. Ta postopek nam zagotavlja, da bomo na koncu res izračunali ničlo.

Obstaja mnogo kombiniranih metod, podrobneje pa bomo zapisali algoritem za Brentovo metodo[4] (glej Algoritem 2.2), ki kombinira bisekcijo, sekantno metodo in inverzno kvadratno in-

[4]Metodo je avstralski matematik Richard P. Brent (r. 1946) objavil leta 1973. Uporablja se v številnih numeričnih paketih, med drugim tudi v funkciji `fzero` v Matlabu. Brent je znan tudi po Brent-Salaminovem algoritmu za računanje števila π, s katerim so v 80-tih in 90-tih letih prejšnjega stoletja postavljali rekorde v številu izračunanih decimalk števila π.

terpolacijo. Metoda ima v bližini ničle superlinearno konvergenco, bisekcija pa zagotavlja, da konvergenca ni prepočasna in da ne pademo iz intervala $[a, b]$.

Algoritem 2.2 Brentova metoda. Začetni podatki sta točki a in b, v katerih je f različno predznačena in toleranci $\epsilon > 0$ in $\delta > 0$.

Po potrebi medsebojno zamenjaj a in b tako, da je $|f(b)| \leq |f(a)|$ ter vzemi $c = a$.
Dokler ni $|a - b| < \epsilon$ ponavljaj

a) Izračunaj novo točko s: če je $c \neq a$ naredi en korak inverzne kvadratne interpolacije, sicer pa en korak sekantne metode.

b) Če je izpolnjen kateri izmed naslednjih pogojev, potem naredi korak bisekcije in za s vzemi $(a + b)/2$:

- s ne leži med $(3a + b)/4$ in b,
- v prejšnjem koraku je bila uporabljena bisekcija in $|s - b| \geq |b - c|/2$ ali $|b - c| < \delta$,
- v prejšnjem koraku ni bila uporabljena bisekcija in $|s - b| \geq |c - d|/2$ ali $|c - d| < \delta$.

c) $d = c$, $c = b$, glede na predznak $f(s)$ zamenjaj a oziroma b z s.

d) Po potrebi medsebojno zamenjaj a in b, da je $|f(b)| \leq |f(a)|$.

2.5.5 Metoda (f, f', f'')

To je posplošitev tangentne metode, kjer upoštevamo tudi drugi odvod in s tem pridobimo kubično konvergenco za enostavne ničle. Metodo izpeljemo z razvijanjem inverzne funkcije v Taylorjevo vrsto, kjer upoštevamo le prve tri člene (če upoštevamo le prva dva člena, dobimo tangentno metodo). Formula za izračun naslednjega približka je

$$x_{r+1} = x_r - \frac{f(x_r)}{f'(x_r)} - \frac{f''(x_r)f^2(x_r)}{2f'^3(x_r)}$$

za $r = 0, 1, \ldots$, podrobno izpeljavo pa lahko najdete npr. v [4].

2.6 Ničle polinomov

Pri polinomih pogosto potrebujemo vse ničle polinoma stopnje n. Tudi če so koeficienti polinoma realni, so ničle lahko kompleksne. Za računanje je na voljo več pristopov.

Prvi pristop je, da uporabimo metode za reševanje nelinearnih enačb in izločamo ničle po vrsti. Z eno izmed navedenih metod (npr. Newtonovo ali Mullerjevo) poiščemo eno ničlo x_1, potem pa nadaljujemo s polinomom nižje stopnje $q(x) = p(x)/(x - x_1)$. To ponavljamo, dokler ne izračunamo vseh ničel. Izkaže se, da je za stabilnost potrebno ničle izločati v pravilnem vrstnem redu.

Druga možnost je, da iz polinoma $p(x) = a_0x^n + a_1x^{n-1} + \cdots + a_n$ sestavimo t.i. *pridruženo*

matriko

$$C_p = \begin{bmatrix} 0 & 1 & & \\ & \ddots & \ddots & \\ & & & 1 \\ -a_n/a_0 & -a_{n-1}/a_0 & \cdots & -a_1/a_0 \end{bmatrix}$$

velikosti $n \times n$, katere karakteristični polinom se do množenja z neničelno konstanto ujema s polinomom p. Ničle polinoma p so potem kar lastne vrednosti matrike C_p. Tako npr. ničle polinoma izračuna Matlab z ukazom `roots`.

Tretja varianta je, da uporabimo posebne metode, ki so razvite samo za iskanje ničel polinomov, kot so npr. Laguerrova, Bairstow–Hitchcockova, Jenkins–Traubova, Ehrlich–Aberthova, Durand–Kernerjeva metoda, itd. Te posebne metode imajo ponavadi hitro konvergenco in konvergirajo skoraj za vsak začetni približek.

Najprej pa poglejmo, kaj lahko povemo o občutljivosti ničel polinomov.

2.6.1 Občutljivost ničel polinoma

Ničle polinomov so lahko zelo občutljive. Spoznali smo že Wilkinsonov primer, kjer polinomu $p(x) = (x-1)(x-2)\cdots(x-20)$ prištejemo polinom $-2^{-23}x^{19}$, kar ničle $9, \ldots, 20$ spremeni v

$$\begin{aligned} x_9 &= 8.91752 \\ x_{10,11} &= 10.0953 \pm 0.64310i \\ &\vdots \\ x_{16,17} &= 16.7307 \pm 2.81263i \\ x_{18,19} &= 19.5024 \pm 1.94033i \\ x_{20} &= 20.8469. \end{aligned}$$

Čeprav so vse ničle enostavne in lepo separirane, majhna motnja (relativna sprememba koeficienta pri x^{19} je $5 \cdot 10^{-10}$) povzroči velike spremembe. Po drugi strani pa za najmanjše ničle velja, da se ne spremenijo veliko. Tako se prve tri ničle zmotijo v

$$\begin{aligned} x_1 &= 1 - 9.8 \cdot 10^{-25} \\ x_2 &= 2 + 9.8 \cdot 10^{-18} \\ x_3 &= 3 - 1.9 \cdot 10^{-13}. \end{aligned}$$

Očitno so ene ničle bolj občutljive od drugih in na nekatere motnja $-2^{-23}x^{19}$ bolj vpliva kot na ostale. Kaj lahko povemo o tem?

Denimo, da imamo polinom $p(x) = a_0x^n + a_1x^{n-1} + \cdots + a_n$, ki ima ničle $x_1, \ldots, x_n$. Ničle zmotenega polinoma $F(x, \epsilon) = p(x) + \epsilon q(x)$, kjer je $q(x) = b_0x^n + b_1x^{n-1} + \cdots + b_n$, naj bodo $x_1(\epsilon), \ldots, x_n(\epsilon)$. Vemo, da so $x_i(\epsilon)$ zvezne funkcije ϵ, saj so ničle polinoma zvezne funkcije njegovih koeficientov. Zanima nas, kako se obnaša $x_i(\epsilon) - x_i$, ko je ϵ majhen.

Izrek 2.7 *Naj bo x_1 enostavna ničla polinoma p. Potem obstaja tak $\delta > 0$, da ima polinom $F(x, \epsilon)$ za vsak $|\epsilon| < \delta$ enostavno ničlo $x_1(\epsilon)$, ki se izraža v obliki konvergentne potenčne vrste*

$$x_1(\epsilon) = x_1 + k_1\epsilon + k_2\epsilon^2 + \cdots, \tag{2.8}$$

pri čemer za koeficient k_1 velja

$$k_1 = -\frac{q(x_1)}{p'(x_1)}.$$

Dokaz. Ker je x_1 enostavna ničla, je $F(x_1, 0) = 0$ in $\frac{\partial F}{\partial x}(x_1, 0) = p'(x_1) \neq 0$, torej po izreku o implicitni funkciji obstaja v neki okolici 0 natanko ena odvedljiva funkcija $x_1(\epsilon)$, da je

$$F(x_1(\epsilon), \epsilon) = 0$$

za vsak $|\epsilon| \leq \delta$ in $x_1(0) = x_1$. Ker je F polinom dveh spremenljivk, je implicitna funkcija analitična, torej se izraža s konvergentno potenčno vrsto (2.8).

V $F(x_1(\epsilon), \epsilon) = 0$ vstavimo (2.8) in izraz razvijemo v Taylorjevo vrsto okrog $(x_1, 0)$. Dobimo

$$0 = F(x_1, 0) + (k_1\epsilon + k_2\epsilon^2 + \cdots)F_x(x_1, 0) + \epsilon F_\epsilon(x_1, 0) + \mathcal{O}(\epsilon^2).$$

Iz tega, da morajo biti vsi koeficienti dobljene potenčne vrste enaki 0, dobimo

$$F_\epsilon(x_1, 0) + k_1 F_x(x_1, 0) = 0 \quad \Longrightarrow \quad k_1 = -\frac{F_\epsilon(x_1, 0)}{F_x(x_1, 0)} = -\frac{q(x_1)}{p'(x_1)}.$$ ■

Večkratne ničle so lahko veliko bolj občutljive. V primeru, ko je x_1 m-kratna ničla, lahko pričakujemo spremembe reda $\mathcal{O}(\sqrt[k]{\epsilon})$, kjer je $k \leq m$.

Posledica 2.8 *Denimo, da polinom $p(x) = a_0x^n + \cdots + a_n$ z enostavno ničlo x_1 zmotimo v $\widetilde{p}(x) = \widetilde{a}_0x^n + \cdots + \widetilde{a}_n$, kjer je $\widetilde{a}_i = a_i(1 + \delta_i)$ in $|\delta_i| \leq \epsilon$. Potem ima polinom $\widetilde{p}$ ničlo $\widetilde{x}_1$, ki zadošča oceni*

$$|\widetilde{x}_1 - x_1| \leq \frac{\sum_{i=0}^{n} |a_i||x_1|^{n-i}}{|p'(x_1)|}\epsilon + \mathcal{O}(\epsilon^2).$$

Zgled 2.10 *Poglejmo, kakšne ocene za motnje ničel dobimo pri Wilkinsonovem primeru. Iz izreka 2.7 dobimo ocene $|x_k - k| \leq b_k$ za $k = 1, \ldots, 20$, kjer so $b_1 = 9.8 \cdot 10^{-25}$, $b_2 = 9.8 \cdot 10^{-18}$, $b_3 = 1.9 \cdot 10^{-13}$,..., $b_9 = 1.0 \cdot 10^{-1}$, $b_{10} = 9.1 \cdot 10^{-1}$, $b_{11} = 5.5 \cdot 10^{0}$,..., $b_{16} = 2.9 \cdot 10^{2}$,..., $b_{20} = 5.1 \cdot 10^{0}$.*

Vidimo, da so ocene pri začetnih ničlah zelo dobre, pri večjih ničlah pa postanejo pregrobe. Kljub temu pa dobro napovejo, da bo pri velikih ničlah prišlo do velikih motenj, pri majhnih ničlah pa do majhnih motenj. □

2.6.2 Laguerreova metoda

Za računanje ničel polinomov imamo na voljo nekaj posebnih metod. Kot prvo si bomo pogledali *Laguerreovo metodo*.[5] Imamo polinom $f(z) = a_0z^n + a_1z^{n-1} + \cdots + a_n$. Njegove ničle naj bodo $\alpha_1, \ldots, \alpha_n$, kar pomeni

$$f(z) = a_0(z - \alpha_1) \cdots (z - \alpha_n).$$

Najprej definiramo

$$S_1(z) = \sum_{i=1}^{n} \frac{1}{z - \alpha_i} = \frac{f'(z)}{f(z)}$$

[5] Francoski matematik Edmond Nicolas Laguerre (1834–1886) se je ukvarjal z geometrijo in kompleksno analizo.

in

$$S_2(z) = \sum_{i=1}^{n} \frac{1}{(z-\alpha_i)^2} = -S_1'(z) = \frac{f'^2(z) - f(z)f''(z)}{f^2(z)}.$$

Denimo, da želimo izračunati α_n. Izbrani recipročni faktor označimo z

$$a(z) = \frac{1}{z-\alpha_n},$$

$b(z)$ pa naj bo aritmetična sredina preostalih recipročnih faktorjev

$$b(z) = \frac{1}{n-1} \sum_{i=1}^{n-1} \frac{1}{z-\alpha_i}.$$

Definirajmo še odstop i-tega faktorja od aritmetične sredine

$$d_i(z) = \frac{1}{z-\alpha_i} - b(z)$$

in vsoto kvadratov odstopov

$$d(z) = \sum_{i=1}^{n-1} d_i^2(z).$$

Sedaj bomo izpustili argument z in izrazili α_n. Če iz enačb

$$\begin{aligned} S_1 &= a + (n-1)b, \\ S_2 &= a^2 + \sum_{i=1}^{n-1} (b+d_i)^2 = a^2 + (n-1)b^2 + d \end{aligned}$$

izrazimo a, dobimo

$$a_{1,2} = \frac{1}{n}\left[S_1 \pm \sqrt{(n-1)(nS_2 - S_1^2 - nd)}\right].$$

in

$$\alpha_n = z - \frac{n}{S_1 \pm \sqrt{(n-1)(nS_2 - S_1^2 - nd)}}.$$

S_1 in S_2 lahko izračunamo za vsak z, ne moremo pa izračunati d. Vendar pa za z blizu α_n velja, da je $|S_1|, |S_2| \gg 0$ in člen nd lahko zanemarimo.

Tako dobimo Laguerreovo metodo:

$$\begin{aligned} S_1 &= \frac{f'(z_r)}{f(z_r)}, \quad S_2 = \frac{f'^2(z_r) - f(z_r)f''(z_r)}{f^2(z_r)} \\ z_{r+1} &= z_r - \frac{n}{S_1 \pm \sqrt{(n-1)(nS_2 - S_1^2)}}, \end{aligned}$$

oziroma

$$z_{r+1} = z_r - \frac{nf(z_r)}{f'(z_r) \pm \sqrt{(n-1)((n-1)f'^2(z_r) - nf(z_r)f''(z_r))}}.$$

Za stabilno računanje izberemo tisti predznak, ki da imenovalcu večjo absolutno vrednost.

Izkaže se, da ima Laguerreova metoda zelo lepe lastnosti za polinome, ki imajo same realne ničle. V takem primeru namreč metoda za poljuben realni začetni približek skonvergira k eni izmed ničel polinoma.

Izrek 2.9 *Če ima polinom f same realne ničle, potem za poljuben realen začetni približek Laguerreova metoda skonvergira k levemu ali desnemu najbližjemu korenu, pri čemer si mislimo, da sta pozitivni in negativni krak realne osi povezana v neskončnosti. V primeru enostavne ničle je red konvergence v bližini ničle kubičen.*

Velja celo več. Če ima polinom same realne ničle, potem Laguerreova metoda za vsak realen začetni približek z_r zaradi izbire predznaka vrne dva možna približka. Izkaže se, da je eden izmed približkov, ki ga bomo označili z $L_-(z_r)$, vedno levo od začetnega približka z_r, drugi približek $L_+(z_r)$ pa je vedno desno od z_r. Pri tem si spet predstavljamo, da sta pozitivni in negativni krak realne osi povezana v neskončnosti. Če vzamemo poljuben začetni približek z_0, potem bo Laguerreova metoda, če vedno izberemo približek L_+, skonvergirala proti tisti ničli polinoma, ki je najbližja z_0 na desni strani. Podobno bo, če vedno izberemo L_-, metoda skonvergirala proti najbližji levi ničli.

Zgled 2.11 *Vzemimo polinom $p(z) = (z-1)(z-2)(z-3)(z-4)(z-5)$. Če vzamemo začetni približek $z_0 = 2.2$, potem s stabilno različico Laguerreove metode dobimo zaporedje*

$$\begin{aligned} z_1 &= 2.0079184501 \\ z_2 &= 2.0000005409 \\ z_3 &= 2.0000000000. \end{aligned}$$

Isto zaporedje približkov dobimo, če v vsakem koraku izberemo približek L_-, medtem ko pri izbiranju približkov L_+ zaporedje skonvergira do najbližje desne ničle z zaporedjem

$$\begin{aligned} z_1 &= 2.4675321780 \\ z_2 &= 2.8343761018 \\ z_3 &= 2.9944283165 \\ z_4 &= 2.9999997838 \\ z_5 &= 3.0000000000. \end{aligned}$$

V obeh primerih je konvergenca kubična. Če vzamemo začetni približek $z_0 = 6$, potem stabilna in L_- različica skonvergirata k 5, različica L_+ pa skonvergira k najbližji desni ničli 1 z zaporedjem

$$\begin{aligned} z_1 &= 14.0905186262 \\ z_2 &= -0.6400718922 \\ z_3 &= 0.9195616027 \\ z_4 &= 0.9999250309 \\ z_5 &= 1.0000000000. \end{aligned}$$

□

Laguerreovo metodo lahko uporabimo tudi za računanje kompleksnih ničel. Ker v formuli nastopa kvadratni koren, ima to lepo lastnost, da tudi iz realnih začetnih približkov lahko dobimo kompleksne ničle. V primeru, ko ima polinom tudi kompleksne ničle, se da poiskati take začetne točke, za katere Laguerreova metoda ne skonvergira k eni izmed ničel. Je pa tovrstnih točk tako malo, da v praksi za naključno izbrano začetno točko Laguerreova metoda vedno skonvergira.

2.6.3 Redukcija

Imamo polinom $p(x) = a_0x^n + a_1x^{n-1} + \cdots + a_n$, za katerega smo izračunali približek α za ničlo. Sedaj lahko polinom reduciramo in nadaljujemo z iskanjem ničel polinoma nižje stopnje. V teoriji to naredimo tako, da $p(x)$ delimo z linearnim polinomom $x - \alpha$ in dobimo polinom nižje stopnje. Pri numeričnem računanju α ni točna ničla, zato se pri deljenju pojavi še ostanek in ni vseeno, na kakšen način opravimo redukcijo in v kakšnem vrstnem redu izločamo ničle.

Standardna izbira za deljenje polinoma je uporaba Hornerjevega algoritma. Če zapišemo

$$p(x) = (x - \alpha)(b_0x^{n-1} + \cdots + b_{n-1}) + b_n,$$

potem koeficiente $b_0, \ldots, b_n$ dobimo s Hornerjevim algoritmom:

$b_0 = a_0$
$r = 1, \ldots, n$
$\quad b_r = \alpha b_{r-1} + a_r$

Ta postopek se imenuje *direktna redukcija*. Na koncu dobimo $b_n = p(\alpha)$. Če je β ničla polinoma $g(x) = b_0x^{n-1} + \ldots + b_{n-1}$, potem je β ničla polinoma $p(x) - p(\alpha)$, ki ima zmoten prosti člen. V primeru, ko ima α veliko absolutno vrednost, lahko pričakujemo veliko vrednost $|p(\alpha)|$ in veliko motnjo p. To pomeni, da je direktna redukcija stabilna, če korene izločamo po naraščajoči absolutni vrednosti.

Kaj pa če najprej izločimo ničlo z veliko absolutno vrednostjo? V tem primeru lahko uporabimo *obratno redukcijo*, kjer polinom delimo tako, da bomo imeli ostanek pri vodilnem koeficientu. Če zapišemo

$$p(x) = (-\alpha + x)(c_n + c_{n-1}x + \cdots + c_1x^{n-1}) + c_0x^n,$$

potem iz enačb za c_i dobimo algoritem:

$c_n = -a_n/\alpha$
$r = n-1, \ldots, 2, 1$
$\quad c_r = (c_{r+1} - a_r)/\alpha$
$c_0 = a_0 - c_1$

Sedaj je $c_0 = \frac{p(\alpha)}{\alpha^n}$. Če je β ničla $g(x) = c_n + \cdots + c_1x^{n-1}$, je β ničla polinoma $p(x) - \frac{p(\alpha)}{\alpha^n}x^n$. Sedaj bomo imeli majhne motnje polinoma p pri ničlah α z veliko absolutno vrednostjo. Torej je obratna redukcija stabilna, če izločamo ničle po padajoči absolutni vrednosti.

Namesto ostanka pri vodilnem ali prostem členu bi lahko deljenje z $x - \alpha$ zapisali tudi tako, da bi imeli ostanek pri x^r, kjer je $r = 1, \ldots, n-1$. V tem primeru gre za *kombinirano redukcijo*, kjer sestavimo začetni del iz direktne redukcije s končnim delom iz obratne redukcije. Velja

$$p(x) = (x - \alpha)(b_0x^{n-1} + \cdots + b_{n-r-1}x^r + c_{n-r-1}x^{r-1} + \cdots + c_n) + d_rx^r,$$

kjer je $d_r = \frac{p(\alpha)}{\alpha^r}$. Če bo motnja d_rx^r majhna, bo postopek stabilen.

Pri danem α optimalni indeks r izberemo tako, da bo relativna sprememba istoležnega koeficienta v polinomu p najmanjša. Ko izračunamo približek α za ničlo polinoma p, poiščemo indeks

$r \in \{0, 1, \ldots, n\}$, kjer je dosežen minimum

$$\left| \frac{p(\alpha)}{\alpha^r a_{n-r}} \right|$$

kar je enako indeksu, kjer je dosežen maksimum

$$|\alpha^r a_{n-r}| \, .$$

Če je $r = 0$ izvedemo direktno redukcijo, če je $r = n$ izvedemo obratno redukcijo, sicer pa kombinirano redukcijo. To nam zagotavlja, da bodo ničle reduciranega polinoma dobri približki za ničle originalnega polinoma.

Zgled 2.12 *Vzemimo polinom* $p(x) = (x-1)(x-10)(x-100)\cdots(x-10^5)$. *Če v Matlabu izračunamo koeficiente tega polinoma z ukazom* `p=poly([1 10 100 1000 10000 100000])` *in potem uporabimo v Matlab vgrajeno metodo za računanje ničel* `roots(p)`, *dobimo naslednje vrednosti za ničle:*

$$\begin{aligned} x_1 &= 1.000000000000001 \cdot 10^5 \\ x_2 &= 1.000000000000001 \cdot 10^4 \\ x_3 &= 1.000000000000001 \cdot 10^3 \\ x_4 &= 1.000000000000001 \cdot 10^2 \\ x_5 &= 1.000000000000000 \cdot 10^1 \\ x_6 &= 9.999999999999996 \cdot 10^{-1}. \end{aligned}$$

To očitno pomeni, da je možno iz koeficientov polinoma p natančno izračunati njegove ničle.

Denimo, da sedaj najprej izračunamo po absolutni vrednosti največjo ničlo, potem pa uporabimo direktno redukcijo in isti postopek nadaljujemo na polinomu nižje stopnje. S tem postopkom pridemo do naslednjih vrednosti za ničle:

$$\begin{aligned} x_1 &= 1.000000000000001 \cdot 10^5 \\ x_2 &= 9.999999999016454 \cdot 10^3 \\ x_3 &= 1.000000993379452 \cdot 10^3 \\ x_4 &= 9.990046379627648 \cdot 10^1 \\ x_5 &= 1.090377418287503 \cdot 10^1 \\ x_6 &= 1.947696248554038 \cdot 10^{-1}. \end{aligned}$$

Kot smo že napovedali pri izpeljavi direktne redukcije, lahko pričakujemo velike napake, če ničle izločamo od največje po absolutni vrednosti proti najmanjši, in to se lepo kaže v tem zgledu. Napaka se veča iz koraka v korak in končna ničla je izračunana povsem narobe.

Če izločamo ničle po padajoči absolutni vrednosti, moramo uporabiti obratno redukcijo. Ta nam za isti polinom stabilno izračuna naslednje približke za ničle:

$$\begin{aligned} x_1 &= 1.000000000000001 \cdot 10^5 \\ x_2 &= 9.999999999999991 \cdot 10^3 \\ x_3 &= 9.999999999999996 \cdot 10^2 \\ x_4 &= 9.999999999999992 \cdot 10^1 \\ x_5 &= 1.000000000000000 \cdot 10^1 \\ x_6 &= 1.000000000000000 \cdot 10^0. \end{aligned}$$

□

2.6.4 Durand–Kernerjeva metoda

Pri tej metodi, ki je znana tudi pod imenom Weierstrassova metoda,[6] računamo vse ničle polinoma hkrati. Zaradi tega nimamo težav z izbiro pravilne redukcije, pride pa seveda v poštev le takrat, kadar potrebujem vse ničle polinoma.

Naj bo

$$p(z) = (z-\alpha_1)(z-\alpha_2)\cdots(z-\alpha_n),$$

kar pomeni, da je vodilni koeficient polinoma enak 1.

Naj bodo $z_1,\ldots,z_n$ po vrsti približki za ničle $\alpha_1,\ldots,\alpha_n$. Iščemo takšne popravke $\Delta z_1,\ldots,\Delta z_n$, da bodo $z_1+\Delta z_1,\ldots,z_n+\Delta z_n$ ničle polinoma p.

Veljati mora

$$(z-(z_1+\Delta z_1))(z-(z_2+\Delta z_2))\cdots(z-(z_n+\Delta z_n)) = p(z).$$

Če to uredimo po členih Δz_i, dobimo

$$p(z) = \prod_{j=1}^{n}(z-z_j) - \sum_{j=1}^{n}\Delta z_j \prod_{\substack{k=1\\k\neq j}}^{n}(z-z_k) + \sum_{\substack{j,k=1\\j<k}}^{n}\Delta z_j\Delta z_k \prod_{\substack{l=1\\l\neq j,k}}^{n}(z-z_l) + \cdots.$$

Če zanemarimo kvadratne in višje člene, potem bodo Δz_i le približni popravki. Ko v zgornjo enakost vstavimo $z = z_i$, dobimo

$$\Delta z_i = \frac{-p(z_i)}{\prod_{\substack{k=1\\k\neq i}}^{n}(z_i-z_k)}.$$

Tako dobimo Durand–Kernerjevo metodo:

$$z_i^{(r+1)} = z_i^{(r)} - \frac{p(z_i^{(r)})}{\prod_{\substack{k=1\\k\neq i}}^{n}(z_i^{(r)}-z_k^{(r)})}, \quad i=1,\ldots,n.$$

Metoda ima v primeru, ko so vse ničle enostavne, v bližini rešitve kvadratično konvergenco, konvergira pa skoraj za vsak začetni vektor $z^{(0)} = [z_1^{(0)} \ \cdots \ z_n^{(0)}]^T \in \mathbb{C}^n$. Paziti moramo le na to, da so komponente vektorja vse paroma različne, sicer se pri računanju naslednjega približka v imenovalcu pojavi 0. Za začetni vektor lahko izberemo kar naključnih n kompleksnih števil.

Če pri izračunu uporabljamo že nove vrednosti, dobimo superkvadratično konvergenco, algoritem pa je:

$$z_i^{(r+1)} = z_i^{(r)} - \frac{p(z_i^{(r)})}{\prod_{k=1}^{i-1}(z_i^{(r)}-z_k^{(r+1)})\prod_{k=i+1}^{n}(z_i^{(r)}-z_k^{(r)})}, \quad i=1,\ldots,n. \tag{2.9}$$

Pri računanju iteriramo le tiste približke z_i, ki še niso skonvergirali.

[6] Metodo so večkrat na novo odkrili. Tako sta jo ločeno objavila francoski fizik Émile Durand leta 1960 in nemški matematik Immo O. Kerner leta 1966, osnove metode pa je znani nemški matematik Karl Weierstrass (1815–1897) objavil že leta 1891. Weierstrass velja za očeta moderne analize, med drugim je prispeval tudi sodobne definicije zveznosti, limite in odvoda.

Zgled 2.13 *Če vzamemo polinom $p(x) = (x-1)(x-10)(x-100)\cdots(x-10^5)$ iz zgleda 2.12, kjer smo imeli pri nepravilni izbiri redukcije težave, potem za začetni vektor $z^{(0)} = [0 \quad 2 \quad 4 \quad 6 \quad 8 \quad 20]^T$, ki je daleč stran od vektorja ničel polinoma, z uporabo Durand–Kernerjeve metode (2.9) dobimo konvergentno zaporedje vektorjev*

$$
\begin{aligned}
z^{(1)} &= \begin{bmatrix} 1.302083333 \cdot 10^{1} \\ -4.953421050 \cdot 10^{0} \\ 1.552853690 \cdot 10^{1} \\ 1.090777571 \cdot 10^{1} \\ 1.088117223 \cdot 10^{1} \\ 3.233916176 \cdot 10^{1} \end{bmatrix} \\
z^{(2)} &= \begin{bmatrix} 1.1104628821 \cdot 10^{5} \\ 4.4098232316 \cdot 10^{5} \\ 3.9279616791 \cdot 10^{2} \\ -6.3345043826 \cdot 10^{1} \\ 1.0855433794 \cdot 10^{1} \\ 3.3598369297 \cdot 10^{1} \end{bmatrix} \\
&\vdots \\
z^{(9)} &= \begin{bmatrix} 1.0000000000 \cdot 10^{5} \\ 1.0000000000 \cdot 10^{4} \\ 1.0000164931 \cdot 10^{0} \\ 1.0000000000 \cdot 10^{3} \\ 1.0000000016 \cdot 10^{1} \\ 1.0000000000 \cdot 10^{2} \end{bmatrix} \\
z^{(10)} &= \begin{bmatrix} 1.0000000000 \cdot 10^{5} \\ 1.0000000000 \cdot 10^{4} \\ 1.0000000000 \cdot 10^{0} \\ 1.0000000000 \cdot 10^{3} \\ 1.0000000000 \cdot 10^{1} \\ 1.0000000000 \cdot 10^{2} \end{bmatrix}
\end{aligned}
$$

in tako naenkrat natančno izračunamo vse ničle polinoma. □

Obstaja več sorodnih metod, ki hkrati računajo vse ničle polinoma. Ena najbolj znanih in uporabnih je Aberth–Ehrlichova metoda, ki poleg vrednosti polinoma uporablja tudi vrednosti odvodov.

Matlab

V standardni verziji je za iskanje ničel realne funkcije ene spremenljivke na voljo funkcija `fzero`, ki uporablja Brentovo kombinirano metodo. Oblika je `fzero(fun,x0,options,p1,p2,...)`, pomen argumentov pa je:

- S prvim argumentom `fun` podamo funkcijo, katere ničlo iščemo. Možnosti so:
  - v obliki niza z definicijo funkcije, npr. `fzero('cos(x)-x',0)`,

  - v obliki t.i. inline funkcije, npr. `f=inline('cos(x)-x'); fzero(f,[-1,1])`,
  - s kazalcem na funkcijo, npr. `fzero(@humps,1)`.

- Drugi argument `x0` je začetni približek, ki je vektor z eno ali dvema komponentama.
  - `fzero('cos(x)-x',[-1,1])`: podamo točki, kjer je funkcija različno predznačena,
  - `fzero('cos(x)-x',0)`: Matlab sam z iskanjem levo in desno poišče začetni interval.
- S tretjim argumentom lahko nastavimo parametre s pomočjo funkcije `optimset`, npr `fzero('cos(x)-x',0,optimset('Display','iter','TolX',1e-4))`. Parametra, ki se jih da nastaviti, sta:
  - `'Display'`: ali se izpisujejo tekoči približki, možnosti so `'iter'`, `'off'`, `'final'`.
  - `'TolX'`: pogoj za konec (ko bo razlika zadnjih dveh približkov pod mejo).
- Z naslednjimi argumenti lahko nastavljamo vrednosti dodatnih spremenljivk funkcije `fun`, če je to funkcija več spremenljivk. Zgled je `f=inline('besselj(0,x)-y','x','y'); fzero(f,[1 2],[],0.5)`.

Dodatna literatura

Z reševanjem nelinearnih enačb se ukvarja knjiga [4]. V tuji literaturi lahko omenjene metode najdemo npr. v [6], [13] in [17].

Poglavje 3

Linearni sistemi

3.1 Osnovne oznake in definicije

V tem poglavju bomo obravnavali numerično reševanje linearnih sistemov. Sistem n linearnih enačb z n neznankami

$$\begin{aligned} a_{11}x_1 + \cdots + a_{1n}x_n &= b_1 \\ &\vdots \\ a_{n1}x_1 + \cdots + a_{nn}x_n &= b_n \end{aligned}$$

zapišemo v obliki

$$Ax = b,$$

kjer je $A \in \mathbb{R}^{n\times n}$ ($\mathbb{C}^{n\times n}$) realna (kompleksna) matrika, $x, b \in \mathbb{R}^n$ ($\mathbb{C}^n$) pa sta realna (kompleksna) vektorja. Pri tem (i,j)-ti element matrike A označimo z a_{ij}, x_i pa je i-ti element vektorja x. Vektor x zapišemo kot

$$x = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} = [x_1 \ \cdots \ x_n]^T.$$

Matriko A lahko zapišemo po stolpcih ali vrsticah v obliki

$$A = [a_1 \ \cdots \ a_n] = \begin{bmatrix} \alpha_1^T \\ \vdots \\ \alpha_n^T \end{bmatrix},$$

kjer sta $a_i, \alpha_i \in \mathbb{R}^n$ ($\mathbb{C}^n$) za $i = 1, \ldots, n$. Oznaka A^T pomeni transponirano matriko A, $A^H = \overline{A^T}$ pa je transponirana in konjugirana matrika A.

Skalarni produkt realnih vektorjev x in y je enak $y^T x = \sum_{i=1}^n x_i y_i$. Množenje vektorja z matriko $y = Ax$ si lahko predstavljamo na dva načina:

- $y_i = \alpha_i^T x$: i-ti element vektorja y je produkt i-te vrstice matrike A in vektorja x,
- $y = \sum_{i=1}^n x_i a_i$: produkt je linearna kombinacija stolpcev matrike A.

Podobno si lahko množenje matrik $C = AB$ predstavljamo na naslednje tri načine:

- $c_{ij} = a_i^T b_j$: (i,j)-ti element matrike C je produkt i-te vrstice A in j-tega stolpca B,
- $c_i = Ab_i$: i-ti stolpec C je produkt A in i-tega stolpca B,
- $C = \sum_{i=1}^n a_i \beta_i^T$: C je vsota produktov i-tega stolpca A in i-te vrstice B za $i = 1, \ldots, n$.

Kvadratna $n \times n$ matrika A je *nesingularna*, če obstaja taka inverzna matrika A^{-1}, da je $AA^{-1} = A^{-1}A = I$. Ekvivalentno je, da je $\det(A) \neq 0$, da je $\text{rang}(A) = n$ ali, da ne obstaja tak neničelni vektor x, za katerega je $Ax = 0$.

Matrika A je *simetrična*, če velja $A = A^T$, v primeru $A = A^H$ pa pravimo, da je matrika *hermitska*[1]. Simetrična matrika je *pozitivno definitna*, če za vsak $x \neq 0$ velja $x^T Ax > 0$. Podobno je hermitska matrika pozitivno definitna, če za vsak $x \neq 0$ velja $x^H Ax > 0$.

Če za skalar λ in neničelni vektor x velja $Ax = \lambda x$, potem je λ *lastna vrednost*, x pa pripadajoči *lastni vektor*. Vsaka matrika ima n lastnih vrednosti, ki so ničle karakterističnega polinoma $p(\lambda) = \det(A - \lambda I)$. Če je matrika simetrična, potem so vse lastne vrednosti $\lambda_1, \ldots, \lambda_n$ realne, lastne vektorje $x_1, \ldots, x_n$ pa lahko izberemo tako, da tvorijo ortonormirano bazo. Če je simetrična matrika dodatno še pozitivno definitna, potem so vse njene lastne vrednosti pozitivne.

3.2 Vektorske in matrične norme

Definicija 3.1 *Vektorska norma je preslikava* $\|.\| : \mathbb{C}^n \to \mathbb{R}$, *za katero velja*

1. $\|x\| \geq 0$, $\|x\| = 0 \iff x = 0$,
2. $\|\alpha x\| = |\alpha| \cdot \|x\|$,
3. $\|x + y\| \leq \|x\| + \|y\|$ *(trikotniška neenakost)*,

za poljubna vektorja $x, y \in \mathbb{C}^n$ *in poljuben skalar* $\alpha \in \mathbb{C}$.

Najbolj znane vektorske norme so:

- $\|x\|_1 = \sum_{i=1}^n |x_i|$ (1-norma),
- $\|x\|_2 = \left(\sum_{i=1}^n |x_i|^2\right)^{1/2}$ (2-norma oziroma evklidska norma),
- $\|x\|_\infty = \max_{i=1,\ldots,n} |x_i|$ (∞-norma).

Vse tri zgornje norme so posebni primeri Hölderjeve[2] p-norme $\|x\|_p = \left(\sum_{i=1}^n |x_i|^p\right)^{1/p}$, kjer je $1 \leq p \leq \infty$. Hölderjeva neenakost pravi, da za poljubna vektorja x, y velja

$$|x^H y| \leq \|x\|_p \|y\|_q \quad \text{za poljubna } 1 \leq p, q \leq \infty, \text{ ki zadoščata } \frac{1}{p} + \frac{1}{q} = 1.$$

[1] Francoski matematiki Charles Hermite (1822–1901) je prvi dokazal, da je baza za naravni logaritem e transcendentno število. Med drugim je znan tudi po interpolacijskih polinomih in družini ortogonalnih polinomih.

[2] Nemški matematik Otto Hölder (1859–1937).

Poseben primer je znana Cauchy–Schwarzeva[3] neenakost $|x^H y| \leq \|x\|_2 \|y\|_2$.

Poljubni vektorski normi $\|.\|_a$ in $\|.\|_b$ sta ekvivalentni, kar pomeni, da obstajata taki pozitivni konstanti C_1 in C_2, da za vsak $x \in \mathbb{C}^n$ velja

$$C_1\|x\|_a \leq \|x\|_b \leq C_2\|x\|_a.$$

Za najpogosteje uporabljene norme veljajo ocene:

$$\begin{array}{rcccl} \|x\|_2 & \leq & \|x\|_1 & \leq & \sqrt{n}\|x\|_2, \\ \|x\|_\infty & \leq & \|x\|_2 & \leq & \sqrt{n}\|x\|_\infty, \\ \|x\|_\infty & \leq & \|x\|_1 & \leq & n\|x\|_\infty. \end{array}$$

Definicija 3.2 *Matrična norma je preslikava* $\|.\| : \mathbb{C}^{n\times n} \to \mathbb{R}$, *za katero velja*

1. $\|A\| \geq 0$, $\|A\| = 0 \iff A = 0$,
2. $\|\alpha A\| = |\alpha| \cdot \|A\|$,
3. $\|A + B\| \leq \|A\| + \|B\|$ *(trikotniška neenakost)*,
4. $\|AB\| \leq \|A\| \cdot \|B\|$ *(submultiplikativnost)*,

za poljubni matriki $A, B \in \mathbb{C}^{n\times n}$ *in poljuben skalar* $\alpha \in \mathbb{C}$.

Vse matrične norme, ki jih bomo spoznali, so take, da lahko definicijo razširimo tudi na pravokotne matrike. Tedaj zahtevo po submultiplikativnosti izpustimo oziroma smiselno zahtevamo, da velja le, kadar sta dimenziji matrik taki, da ju lahko množimo. V tem primeru imamo v bistvu različne prostore in različne norme, ki so tako usklajene, da neenakost velja.

Elemente matrike velikosti $n \times n$ lahko shranimo v vektor dolžine n^2. Za ta vektor lahko izračunamo poljubno vektorsko normo. Ta potem avtomatično zadošča prvim trem pogojem za matrično normo, vprašanje je le, ali zadošča tudi zadnjemu pogoju o submultiplikativnosti. Če za najbolj znane vektorske norme definiramo:

$$\begin{array}{rcl} N_1(A) & := & \displaystyle\sum_{i,j=1}^{n} |a_{ij}|, \\ N_2(A) & := & \left(\displaystyle\sum_{i,j=1}^{n} |a_{ij}|^2\right)^{\frac{1}{2}}, \\ N_\infty(A) & := & \displaystyle\max_{i,j=1,\ldots,n} |a_{ij}|, \end{array}$$

se izkaže, da N_∞ ni matrična norma, saj ne izpolnjuje pogoja submultiplikativnosti. Če vzamemo npr.

$$A = B = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix},$$

[3]Neenakost je prvi objavil znani francoski matematik Augustin-Louis Cauchy (1789–1857), ponovno pa jo je odkril nemški matematik Karl Hermann Amandus Schwarz (1842–1921). Cauchy je bil izredno dejaven matematik, objavil je kar 789 člankov, zato ni čudno, da njegovo ime nastopa v mnogih matematičnih pojmih.

potem je $N_\infty(AB) > N_\infty(A)N_\infty(B)$. Ostali dve preslikavi sta matrični normi, pri čemer se N_1 ne uporablja, N_2 pa je *Frobeniusova norma*[4] $\|.\|_F$.

Pomembnejše so *operatorske oz. inducirane norme*. Definiramo jih lahko iz poljubne vektorske norme s predpisom

$$\|A\| := \max_{\substack{x\neq 0\\ x\in\mathbb{C}^n}} \frac{\|Ax\|}{\|x\|} = \max_{\|x\|=1} \|Ax\|. \tag{3.1}$$

Lema 3.3 *Operatorska norma je matrična norma.*

Dokaz. Po vrsti preverimo, da res izpolnjuje vse štiri točke iz definicije matrične norme. Očitno je $\|A\| \geq 0$. Izraz (3.1) je enak 0 le v primeru, ko je $Ax = 0$ za vsak x, to pa je natanko tedaj, ko je $A = 0$. Lastnost množenja s skalarjem in trikotniška neenakost sta očitni, preveriti moramo le še submultiplikativnost.

Iz (3.1) sledi, da za vsak x velja ocena $\|Ax\| \leq \|A\|\|x\|$. To pa pomeni

$$\|AB\| = \max_{x\neq 0} \frac{\|ABx\|}{\|x\|} \leq \max_{x\neq 0} \frac{\|A\|\|Bx\|}{\|x\|} = \|A\| \max_{x\neq 0} \frac{\|Bx\|}{\|x\|} = \|A\|\|B\|$$

in operatorska norma je submultiplikativna. ∎

Iz treh najbolj znanih vektorskih norm dobimo naslednje operatorske matrične norme: $\|A\|_1 := \max_{\substack{x\neq 0\\ x\in\mathbb{C}^n}} \frac{\|Ax\|_1}{\|x\|_1}$ (1-norma), $\|A\|_2 := \max_{\substack{x\neq 0\\ x\in\mathbb{C}^n}} \frac{\|Ax\|_2}{\|x\|_2}$ (2-norma), $\|A\|_\infty := \max_{\substack{x\neq 0\\ x\in\mathbb{C}^n}} \frac{\|Ax\|_\infty}{\|x\|_\infty}$ (∞-norma). Kot kažejo naslednje tri leme, obstajajo za vse tri norme direktne formule.

Lema 3.4 $\|A\|_1 = \max_{j=1,\ldots,n} \left(\sum_{i=1,\ldots,n} |a_{ij}| \right)$.

Dokaz. Če matriko A zapišemo po stolpcih kot $A = [a_1 \ \cdots \ a_n]$, potem velja $Ax = \sum_{i=1}^n x_i a_i$. Ocenimo lahko

$$\|Ax\|_1 = \left\| \sum_{i=1}^n x_i a_i \right\|_1 \leq \sum_{i=1}^n |x_i| \|a_i\|_1 \leq \max_{k=1,\ldots,n} \|a_k\|_1 \sum_{i=1}^n |x_i| = \max_{k=1,\ldots,n} \|a_k\|_1 \|x\|_1.$$

To pomeni $\|A\|_1 \leq \max_{k=1,\ldots,n} \|a_k\|_1$, enakost pa je dosežena, če vzamemo $x = e_j$, kjer je $\|a_j\|_1 \geq \|a_k\|_1$ za $k = 1, \ldots, n$. ∎

Za poljubno matriko $A \in \mathbb{C}^{n\times n}$ je matrika $B = A^H A$ hermitska in nenegativno definitna, saj velja $B^H = B$ in $x^H Bx = \|Ax\|_2^2 \geq 0$ za vsak $x \in \mathbb{C}^n$. Od tod sledi, da so vse lastne vrednosti matrike B nenegativne in jih lahko zapišemo urejene po velikosti kot $\sigma_1^2 \geq \sigma_2^2 \geq \cdots \geq \sigma_n^2 \geq 0$. Pozitivne kvadratne korene $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n \geq 0$ lastnih vrednosti $A^H A$ imenujemo *singularne vrednosti* matrike A.

Lema 3.5 $\|A\|_2 = \sigma_1(A) = \max_{i=1,\ldots,n} \sqrt{\lambda_i(A^H A)}$.

[4] Nemški matematik Ferdinand Georg Frobenius (1849–1912) je bil eden prvih, ki se je ukvarjal s teorijo matrik, znan pa je predvsem po svojih rezultatih iz teorije grup in teorije števil.

Dokaz. Obstaja ortonormirana baza za $\mathbb{C}^n$, ki jo sestavljajo lastni vektorji $u_1, \ldots, u_n$ matrike A^HA, za katere velja $A^HAu_i = \sigma_i^2 u_i$ za $i = 1, \ldots, n$. Če vektor x razvijemo po tej bazi kot $x = \sum_{i=1}^n \alpha_i u_i$, dobimo

$$\|Ax\|_2^2 = x^H(A^HAx) = \left(\sum_{i=1}^n \alpha_i u_i\right)^H \left(\sum_{i=1}^n \alpha_i \sigma_i^2 u_i\right) = \sum_{i=1}^n |\alpha_i|^2 \sigma_i^2 \leq \sigma_1^2 \sum_{i=1}^n |\alpha_i|^2 = \sigma_1^2 \|x\|_2^2.$$

To pomeni $\|A\|_2 \leq \sigma_1(A)$, enakost pa je dosežena pri izbiri $x = u_1$. ■

Zaradi zveze $\|A\|_2 = \sigma_1(A)$ imenujemo matrično 2-normo tudi *spektralna norma*. V primeru $A = A^H$ je $\|A\|_2 = \max|\lambda(A)|$, saj lahko hitro vidimo, da so singularne vrednosti hermitske matrike enake absolutnim vrednostim lastnih vrednosti.

Lema 3.6 $\|A\|_\infty = \max\limits_{i=1,\ldots,n} \left(\sum\limits_{j=1,\ldots,n} |a_{ij}|\right)$.

Dokaz. Za $A = \begin{bmatrix} \alpha_1^T \\ \vdots \\ \alpha_n^T \end{bmatrix}$ dobimo

$$\|Ax\|_\infty = \max_{i=1,\ldots,n} |\alpha_i^T x| \leq \max_{i=1,\ldots,n} \|x\|_\infty \|\alpha_i\|_1.$$

To pomeni $\|A\|_\infty \leq \max_{i=1,\ldots,n} \|\alpha_i\|_1$, enakost pa je dosežena pri vektorju x, ki ima k-to komponento enako

$$x_k = \begin{cases} \frac{\overline{a_{jk}}}{|a_{jk}|} & \text{za } a_{jk} \neq 0 \\ 0 & \text{sicer,} \end{cases}$$

kjer je indeks j izbran tako, da velja $\|\alpha_j\|_1 \geq \|\alpha_k\|_1$ za $k = 1, \ldots, n$. ■

Za vsak operatorsko normo očitno velja, da je $\|I\| = 1$. Zaradi tega Frobeniusova norma ni operatorska, saj je $\|I\|_F = \sqrt{n}$.

Opazimo lahko, da je spektralno normo $\|.\|_2$ veliko težje izračunati kot pa norme $\|.\|_1$, $\|.\|_\infty$ in $\|.\|_F$, Tako kot vektorske norme so tudi matrične norme medsebojno ekvivalentne. Pomembne so naslednje ocene, s katerimi lahko ocenimo $\|A\|_2$ z lažje izračunljivimi normami in količinami:

$$\begin{array}{lcll} \frac{1}{\sqrt{n}}\|A\|_F & \leq & \|A\|_2 & \leq \|A\|_F \\ \frac{1}{\sqrt{n}}\|A\|_1 & \leq & \|A\|_2 & \leq \sqrt{n}\|A\|_1 \\ \frac{1}{\sqrt{n}}\|A\|_\infty & \leq & \|A\|_2 & \leq \sqrt{n}\|A\|_\infty \\ N_\infty(A) & \leq & \|A\|_2 & \leq nN_\infty(A) \\ & & \|A\|_2 & \leq \sqrt{\|A\|_1\|A\|_\infty} \\ \|a_i\|_2, \|\alpha_i\|_2 & \leq & \|A\|_2 & \end{array}$$

Zgled 3.1 *Za matriko*

$$A = \begin{bmatrix} 3 & -1 & 2 \\ 4 & 1 & -8 \\ 1 & -5 & 0 \end{bmatrix}$$

dobimo $\|A\|_1 = 10$, $\|A\|_\infty = 13$ *in* $\|A\|_F = 11$. *Iz zgornjih neenakosti ocenimo* $9 \leq \|A\|_2 \leq 11$, *prava vrednost pa je* $\|A\|_2 = 9.02316$. □

Uporabljali bomo le t.i. usklajene pare vektorskih in matričnih norm, kjer za vsak par matrike A in vektorja x velja

$$\|Ax\| \leq \|A\|\|x\|.$$

Operatorska norma je očitno usklajena z vektorsko normo, ki nastopa v njeni definiciji. Frobeniusova norma, ki ni operatorska, pa je usklajena z evklidsko normo $\|.\|_2$, saj velja

$$\|Ax\|_2 \leq \|A\|_2\|x\|_2 \leq \|A\|_F\|x\|_2.$$

Pokazati je možno, da za vsako matrično normo obstaja z njo usklajena vektorska norma, zato lahko od zdaj naprej predpostavimo, da sta matrična in vektorska norma usklajeni.

Lema 3.7 *Za vsako matrično normo in poljubno lastno vrednost λ matrike A velja*

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

Dokaz. Naj bo $x \neq 0$ lastni vektor, ki pripada lastni vrednosti λ. Iz $Ax = \lambda x$ dobimo

$$|\lambda| \cdot \|x\| = \|\lambda x\| = \|Ax\| \leq \|A\| \cdot \|x\|.$$

■

Realna matrika Q je *ortogonalna*, če je $Q^{-1} = Q^T$. Kompleksna matrika U je *unitarna*, če je $U^{-1} = U^H$. Vemo, da se pri množenju z unitarno matriko ohranja evklidska norma vektorja, oziroma velja $\|Ux\|_2 = \|x\|_2$. Spoznali bomo, da je pri raznih transformacijah, s katerimi matriko množimo, da bi jo spravili v ugodnejšo obliko, za ohranjanje stabilnosti zelo pomembno, da matriko množimo z unitarnimi matrikami. Na ta način se natančnost ne izgublja, glavni razlog pa se skriva v naslednji lemi.

Lema 3.8 *Frobeniusova in spektralna norma sta invariantni na množenje z unitarno matriko, oziroma*

$$\begin{aligned} \|A\|_F &= \|UA\|_F = \|AU\|_F, \\ \|A\|_2 &= \|UA\|_2 = \|AU\|_2 \end{aligned}$$

za poljubno matriko A in unitarno matriko U.

Dokaz. Naj bo $A = [a_1 \ \cdots \ a_n]$. Velja

$$\|UA\|_F^2 = \sum_{i=1}^{n} \|Ua_i\|_2^2 = \sum_{i=1}^{n} \|a_i\|_2^2 = \|A\|_F^2.$$

Za spektralno normo dobimo

$$\|UA\|_2 = \max_{\|x\|_2=1} \|UAx\|_2 = \max_{\|x\|_2=1} \|Ax\|_2 = \|A\|_2.$$

Dokaz za množenje z unitarno matriko z leve zadošča, saj zaradi $\|A\|_F = \|A^H\|_F$ in $\|A\|_2 = \|A^H\|_2$ množenje z unitarno matriko z desne lahko prevedemo na množenje z leve. ■

3.3 Občutljivost linearnih sistemov

Zanima nas, kako je rešitev linearnega sistema $Ax = b$ občutljiva na spremembe matrike A in desne strani b. V nadaljevanju bomo potrebovali naslednjo lemo.

Lema 3.9 *Če za matriko X velja $\|X\| < 1$ v matrični normi, v kateri je $\|I\| = 1$, potem je matrika $I - X$ nesingularna in velja*

$$\|(I - X)^{-1}\| \leq \frac{1}{1 - \|X\|}.$$

Dokaz. Denimo, da je matrika $I - X$ singularna. Potem obstaja tak vektor z, da je $(I - X)z = 0$. To pa pomeni $z = Xz$ in $\|z\| = \|Xz\| \leq \|X\| \cdot \|z\|$, torej $\|X\| \geq 1$ in pridemo do protislovja.

Inverz matrike $I - X$ je enak vsoti $\sum_{i=0}^{\infty} X^i$, kar lahko preverimo tako, da vsoto, ki je zaradi $\|X\| < 1$ konvergentna, pomnožimo z $I - X$. Sedaj lahko ocenimo

$$\|(I - X)^{-1}\| = \left\| \sum_{i=0}^{\infty} X^i \right\| \leq \sum_{i=0}^{\infty} \|X\|^i = \frac{1}{1 - \|X\|}. \qquad \blacksquare$$

Naj bo $Ax = b$ in $(A + \delta A)(x + \delta x) = b + \delta b$. Od tod dobimo

$$\delta x = (A + \delta A)^{-1}(-\delta A x + \delta b) = (I + A^{-1}\delta A)^{-1} A^{-1}(-\delta A x + \delta b).$$

Če predpostavimo, da je $\|A^{-1}\| \cdot \|\delta A\| < 1$, potem je $I + A^{-1}\delta A$ nesingularna in lahko ocenimo

$$\|(I + A^{-1}\delta A)^{-1}\| \leq \frac{1}{1 - \|A^{-1}\delta A\|} \leq \frac{1}{1 - \|A^{-1}\| \cdot \|\delta A\|}.$$

Dobimo

$$\|\delta x\| \leq \frac{\|A^{-1}\|}{1 - \|A^{-1}\| \cdot \|\delta A\|} (\|\delta A\| \cdot \|x\| + \|\delta b\|),$$

od tod pa

$$\begin{aligned} \frac{\|\delta x\|}{\|x\|} &\leq \frac{\|A^{-1}\|}{1 - \|A^{-1}\| \cdot \|\delta A\|} \left(\frac{\|\delta A\|}{\|A\|} \cdot \|A\| + \frac{\|\delta b\| \cdot \|A\|}{\|x\| \cdot \|A\|} \right) \\ &\leq \frac{\|A^{-1}\| \cdot \|A\|}{1 - \|A^{-1}\| \cdot \|A\| \cdot \frac{\|\delta A\|}{\|A\|}} \left(\frac{\|\delta A\|}{\|A\|} + \frac{\|\delta b\|}{\|b\|} \right), \end{aligned}$$

kjer smo v zadnjem koraku upoštevali, da iz $Ax = b$ sledi $\|A\| \cdot \|x\| \geq \|b\|$.

Iz zadnje ocene je razvidno, da na velikost spremembe rešitve najbolj vpliva produkt norm $\|A^{-1}\| \cdot \|A\|$. Število $\kappa(A) := \|A^{-1}\| \cdot \|A\|$ imenujemo *občutljivost* oz. *pogojenostno število* matrike A. Tako smo izpeljali naslednji izrek.

Izrek 3.10 *Naj bo A nesingularna matrika in δA taka motnja, da je $\|A^{-1}\| \cdot \|\delta A\| < 1$. Če je $Ax = b$ in $(A + \delta A)(x + \delta x) = b + \delta b$, potem velja ocena*

$$\frac{\|\delta x\|}{\|x\|} \leq \frac{\kappa(A)}{1 - \kappa(A)\frac{\|\delta A\|}{\|A\|}} \left(\frac{\|\delta A\|}{\|A\|} + \frac{\|\delta b\|}{\|b\|} \right).$$

Za občutljivost velja $1 \leq \kappa(A)$, saj je $1 \leq \|I\| = \|AA^{-1}\| \leq \|A\| \cdot \|A^{-1}\|$.

Zgled 3.2 *V zgledu 1.3 smo imeli linearna sistema velikosti* 2×2 *z matrikama*

$$A_1 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \quad in \quad A_2 = \begin{bmatrix} 1 & 0.99 \\ 0.99 & 0.98 \end{bmatrix}.$$

Ko smo zmotili desno stran, se je rešitev sistema z matriko A_1 *spremenila za velikostni razred motnje, rešitev sistema* A_2 *pa za občutno več. Če izračunamo občutljivosti obeh matrik, dobimo* $\kappa_2(A_1) = 1$ *in* $\kappa_2(A_2) = 3.9 \cdot 10^4$*, kar potrjuje ugotovitve iz zgleda 1.3, da je sistem z matriko* A_2 *veliko občutljivejši od sistema z matriko* A_1*.* □

Za spektralno občutljivost velja

$$\kappa_2(A) = \frac{\sigma_1(A)}{\sigma_n(A)}.$$

Edine matrike, ki imajo spektralno občutljivost enako 1, so z neničelnim skalarjem pomnožene unitarne matrike.

Naslednji izrek pravi, da je občutljivost recipročna oddaljenosti od singularnega problema. Izrek bomo v splošnejši obliki dokazali v nadaljevanju, ko bomo obravnavali singularni razcep, zato bomo dokaz izpustili.

Izrek 3.11 *Če je matrika* A *nesingularna, potem je*

$$\min \left\{ \frac{\|\delta A\|_2}{\|A\|_2};\ A + \delta A \text{ singularna} \right\} = \frac{1}{\|A^{-1}\|_2 \|A\|_2} = \frac{1}{\kappa_2(A)}.$$

Ko rešujemo linearni sistem $Ax = b$, lahko pričakujemo, da bo za numerično izračunani približek $\widetilde{x}$ veljalo vsaj $\|\widetilde{x} - x\|_2 / \|x\|_2 = \mathcal{O}(\kappa_2(A)u)$, kjer je u osnovna zaokrožitvena napaka. V praksi je napaka lahko še večja, saj moramo upoštevali še stabilnost numerične metode, s katero rešujemo sistem.

Zgled 3.3 *Matrika, ki je znana kot zelo občutljiva, je Hilbertova*[5] *matrika* H_n*, kjer je* (i, j)*-ti element enak* $1/(i + j - 1)$ *za* $i, j = 1, \ldots, n$*. Tako npr. velja* $\kappa_2(H_4) = 1.6 \cdot 10^4$*,* $\kappa_2(H_7) = 4.8 \cdot 10^8$ *in* $\kappa_2(H_{10}) = 1.6 \cdot 10^{13}$*.*

Če v Matlabu, ki računa v dvojni natančnosti, vzamemo `A=hilb(7), x0=ones(7,1), b=A*x0` *in nato numerično rešimo sistem z ukazom* `x=A\b`*, potem za izračunani približek velja* $\|x - x_0\|_2 = 3.2 \cdot 10^{-8}$*, kar je primerljivo s* $\kappa_2(H_7)u = 1.1 \cdot 10^{-7}$*. Naslednji zgled kaže, da se s Hilbertovimi matrikami lahko srečamo tudi v praksi.* □

Zgled 3.4 *Iščemo koeficiente polinoma* $p(x) = a_1 + a_2 x + \cdots + a_n x^{n-1}$*, ki na intervalu* $[0, 1]$ *aproksimira zvezno funkcijo* f *tako, da je napaka* $E = \int_0^1 (f(x) - p(x))^2 dx$ *minimalna.*

[5] Nemški matematik David Hilbert (1862–1943) velja za enega najpomembnejših matematikov 19. in 20. stoletja. Njegova zbirka 23 odprtih problemov, ki jo je predstavil leta 1900, je imela velik vpliv na nadaljnji razvoj matematike.

Vemo, da so pri lokalnem minimumu vsi parcialni odvodi enaki 0*, torej mora veljati* $\frac{\partial E}{\partial a_i} = 0$ *za* $i = 1, \ldots, n$*, kjer je*

$$\frac{\partial E}{\partial a_i} = 2 \int_0^1 (p(x) - f(x)) x^{i-1} dx.$$

Od tod dobimo

$$\int_0^1 f(x) x^{i-1} dx = \int_0^1 \left(\sum_{j=1}^n a_j x^{j-1} x^{i-1} \right) dx = \sum_{j=1}^n a_j \int_0^1 x^{i+j-2} dx = \sum_{j=1}^n a_j \frac{1}{i+j-1}.$$

Opazimo, da so koeficienti optimalnega polinoma rešitev linearnega sistema s Hilbertovo matriko H_n*, kjer je* $h_{ij} = 1/(i+j-1)$*. Hilbertove matrike spadajo med zelo občutljive matrike, saj velja npr.* $\kappa(H_4) = 1.6 \cdot 10^4$*,* $\kappa(H_7) = 4.8 \cdot 10^8$ *in* $\kappa(H_{10}) = 1.6 \cdot 10^{13}$*.*

Izkaže se, da problem ni v sami nalogi, ki je dobro definirana, temveč v izbrani polinomski bazi. Če bi namesto standardne baze, ki je zelo občutljiva, izbrali bazo ortogonalnih polinomov, ne bi imeli nobenih težav. □

Zgled 3.5 *Zmotno je prepričanje, da je občutljivost povezana z velikostjo determinante matrike. To ni res. Tako ima npr. matrika* αI *lahko zelo veliko ali zelo majhno determinanto* α^n*, njeno pogojenostno število pa je vedno* 1*. Po drugi strani pa ima npr. matrika*

$$B_n = \begin{bmatrix} 1 & -1 & \cdots & -1 \\ 0 & 1 & \cdots & -1 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix}$$

determinanto 1*, občutljivost pa* $n2^{n-1}$*.* □

3.4 Elementarne eliminacije

Denimo, da imamo tak vektor $x \in \mathbb{R}^n$, da je $x_k \neq 0$. Iščemo čim preprostejšo nesingularno matriko L_k, da bo

$$L_k \begin{bmatrix} x_1 \\ \vdots \\ x_k \\ x_{k+1} \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} x_1 \\ \vdots \\ x_k \\ 0 \\ \vdots \\ 0 \end{bmatrix}.$$

Če definiramo $l_{jk} = \frac{x_j}{x_k}$, $j = k+1, \ldots, n$, potem je

$$L_k = \begin{bmatrix} 1 & & & & \\ & \ddots & & & \\ & & 1 & & \\ & & -l_{k+1,k} & 1 & \\ & & \vdots & & \ddots \\ & & -l_{n,k} & & 1 \end{bmatrix}$$

ustrezna matrika. Matriko L_k imenujemo *elementarna eliminacija*, zapišemo pa jo lahko tudi kot $L_k = I - l_k e_k^T$, kjer je

$$l_k = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ l_{k+1,k} \\ \vdots \\ l_{n,k} \end{bmatrix}.$$

Inverz matrike L_k je

$$L_k^{-1} = I + l_k e_k^T = \begin{bmatrix} 1 & & & & \\ & \ddots & & & \\ & & 1 & & \\ & & l_{k+1,k} & 1 & \\ & & \vdots & & \ddots \\ & & l_{n,k} & & 1 \end{bmatrix}.$$

Ker za $i < j$ velja $(I + l_i e_i^T)(I + l_j e_j^T) = I + l_i e_i^T + l_j e_j^T$, je produkt matrik $L_1^{-1} L_2^{-1} \cdots L_{n-1}^{-1}$ enak

$$\begin{bmatrix} 1 & & & & \\ l_{21} & 1 & & & \\ l_{31} & l_{32} & 1 & & \\ \vdots & \vdots & \ddots & \ddots & \\ l_{n1} & l_{n2} & \cdots & l_{n,n-1} & 1 \end{bmatrix}.$$

3.5 LU razcep

Večina algoritmov za reševanje matričnih problemov deluje na principu, da problem najprej prevede na enostavnejši problem, ki ima lepšo obliko. Potem ekonomično reši enostavnejši problem, pri čemer si pomaga s posebno obliko, na koncu pa rešitev enostavnejšega problema še pretvori v rešitev originalnega problema.

Pri reševanju linearnega sistema je enostavnejša oblika, v katero spravimo matriko, zgornja (ali spodnja) trikotna. V tem razdelku bomo videli, kako lahko to redukcijo izvedemo s pomočjo elementarnih eliminacij, v poglavju 5 pa bomo spoznali še, kako lahko to naredimo s pomočjo ortogonalnih transformacij.

Poglejmo, kako lahko z elementarnimi eliminacijami matriko A zapišemo v obliki $A = LU$, kjer je L spodnja trikotna matrika z enicami na diagonali, U pa zgornja trikotna matrika.

Naj bo

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix}$$

in $a_{11} \neq 0$. Za eliminacijsko matriko

$$L_1 = \begin{bmatrix} 1 & & & \\ -l_{21} & 1 & & \\ \vdots & & \ddots & \\ -l_{n1} & & & 1 \end{bmatrix},$$

kjer je $l_{21} = \frac{a_{21}}{a_{11}}, \ldots, l_{n1} = \frac{a_{n1}}{a_{11}}$ velja

$$L_1 \begin{bmatrix} a_{11} \\ a_{21} \\ \vdots \\ a_{n1} \end{bmatrix} = \begin{bmatrix} a_{11} \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \quad \text{torej} \quad A^{(1)} := L_1 A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ 0 & a_{22}^{(1)} & \cdots & a_{2n}^{(1)} \\ \vdots & \vdots & & \vdots \\ 0 & a_{n2}^{(1)} & \cdots & a_{nn}^{(1)} \end{bmatrix}.$$

Če je $a_{22}^{(1)} \neq 0$, lahko nadaljujemo in izračunamo

$$A^{(2)} := L_2 A^{(1)} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ 0 & a_{22}^{(1)} & a_{23}^{(1)} & \cdots & a_{2n}^{(1)} \\ 0 & 0 & a_{33}^{(2)} & \cdots & a_{3n}^{(2)} \\ \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & a_{n3}^{(2)} & \cdots & a_{nn}^{(2)} \end{bmatrix},$$

kjer je

$$L_2 = \begin{bmatrix} 1 & & & & \\ & 1 & & & \\ & -l_{32} & 1 & & \\ & \vdots & & \ddots & \\ & -l_{n2} & & & 1 \end{bmatrix},$$

in $l_{32} = \frac{a_{32}^{(1)}}{a_{22}^{(1)}}, \ldots, l_{n2} = \frac{a_{n2}^{(1)}}{a_{22}^{(1)}}$. Na koncu dobimo

$$U := \underbrace{L_{n-1} \cdots L_2 L_1}_{=L^{-1}} A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ & a_{22}^{(1)} & \cdots & a_{2n}^{(1)} \\ & & \ddots & \vdots \\ & & & a_{nn}^{(n-1)} \end{bmatrix},$$

in

$$L = L_1^{-1} L_2^{-1} \cdots L_{n-1}^{-1} = \begin{bmatrix} 1 & & & & \\ l_{21} & 1 & & & \\ l_{31} & l_{32} & 1 & & \\ \vdots & \vdots & \ddots & \ddots & \\ l_{n1} & l_{n2} & \cdots & l_{n,n-1} & 1 \end{bmatrix}.$$

Tako smo dobili $A = LU$. Dobljeni razcep imenujemo *LU razcep brez pivotiranja* ali *Gaussov*[6] *razcep brez pivotiranja*.

[6]Imenuje se po slavnem nemškem matematiku, astronomu in fiziku Johannu Carlu Friedrichu Gaussu (1777–1855), ki pa je metodo zapisal v povsem drugačni obliki, kot jo poznamo sedaj. Moderna matrična oblika, ki jo sedaj uporabljamo, izvira šele iz sredine 20. stoletja. Več o razvoju metode, ki so jo na Kitajskem poznali že pred našim štetjem, pred Gaussom pa jo že zapisal tudi Newton, lahko najdete npr. v [11].

Diagonalni elementi $a_{11}, a_{22}^{(1)}, \ldots, a_{n-1,n-1}^{(n-2)}$, s katerimi med postopkom delimo, se imenujejo *pivoti*, l_{ij} pa *kvocienti*. Opazimo lahko, da morajo biti pivoti neničelni, sicer metoda odpove. Naslednji izrek pove, kdaj lahko razcep izračunamo brez težav.

Izrek 3.12 *Za $n \times n$ matriko A je ekvivalentno:*

1) *Obstaja enolični razcep $A = LU$, kjer je L spodnja trikotna matrika z enicami na diagonali in U nesingularna zgornja trikotna matrika.*
2) *Vse vodilne podmatrike $A(1:k, 1:k)$ za $k = 1, \ldots, n$ so nesingularne.*

Dokaz. Pokažimo, da iz 1) sledi 2). Razcep $A = LU$ za poljubno vodilno podmatriko A_{11} bločno zapišemo kot

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} = \begin{bmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{bmatrix} \begin{bmatrix} U_{11} & U_{12} \\ 0 & U_{22} \end{bmatrix} = \begin{bmatrix} L_{11}U_{11} & L_{11}U_{12} \\ L_{21}U_{11} & L_{21}U_{12} + L_{22}U_{22} \end{bmatrix},$$

od koder sledi $A_{11} = L_{11}U_{11}$. Dobimo $\det(A_{11}) = \det(U_{11}) \neq 0$, saj je U nesingularna.

Za dokaz v obratni smeri uporabimo indukcijo. Pri matrikah 1×1 ni težav, saj je $a_{11} = 1 \cdot a_{11}$ iskani LU razcep. Naj bo $A = LU$ in $\widetilde{A} = \begin{bmatrix} A & b \\ c^T & \delta \end{bmatrix}$. Razcep za $\widetilde{A}$ mora imeti obliko

$$\begin{bmatrix} A & b \\ c^T & \delta \end{bmatrix} = \begin{bmatrix} L & 0 \\ l^T & 1 \end{bmatrix} \begin{bmatrix} U & u \\ 0 & \eta \end{bmatrix} = \begin{bmatrix} LU & Lu \\ l^T U & l^T u + \eta \end{bmatrix}.$$

Ker sta L in U nesingularni, dobimo $u = L^{-1}b$, $l = U^{-T}c$ in $\eta = \delta - l^T u$. Pri tem mora biti $\eta \neq 0$, saj je $0 \neq \det A = \eta \cdot \det U$. ∎

Zgled 3.6 *Izračunajmo LU razcep za matriko* $A = \begin{bmatrix} 2 & 2 & 3 \\ 4 & 5 & 6 \\ 1 & 2 & 4 \end{bmatrix}$.

$$A^{(1)} = \underbrace{\begin{bmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ -\frac{1}{2} & 0 & 1 \end{bmatrix}}_{L_1} \cdot A = \begin{bmatrix} 2 & 2 & 3 \\ 0 & 1 & 0 \\ 0 & 1 & \frac{5}{2} \end{bmatrix}$$

$$A^{(2)} = \underbrace{\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -1 & 1 \end{bmatrix}}_{L_2} \cdot A^{(1)} = \begin{bmatrix} 2 & 2 & 3 \\ 0 & 1 & 0 \\ 0 & 0 & \frac{5}{2} \end{bmatrix} = U.$$

$$L = L_1^{-1}L_2^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ \frac{1}{2} & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ \frac{1}{2} & 1 & 1 \end{bmatrix}. \quad \square$$

Elemente matrike L lahko shranjujemo v spodnji trikotnik matrike A in tako ne potrebujemo dodatnega prostora. Na koncu spodnjemu trikotniku matrike A dodamo še identiteto in dobimo matriko L, iz zgornjega trikotnika A pa na koncu preberemo U. Algoritem lahko zapišemo v obliki Algoritma 3.1.

Algoritem 3.1 LU razcep brez pivotiranja. Začetni podatek je $n \times n$ realna matrika A. Na koncu imamo v zgornjem trikotniku matrike A matriko U, v matriki L pa vse poddiagonalne elemente matrike L iz LU razcepa $A = LU$.

```
j = 1,...,n-1
    i = j+1,...,n
        l_ij = a_ij / a_jj
        k = j+1,...,n
            a_ik = a_ik - l_ij a_jk
```

Zahtevnosti algoritmov bomo ocenjevali s številom potrebnih osnovnih operacij. To naredimo tako, da vse zanke v algoritmu zamenjamo z vsotami in seštejemo število operacij. Če to izvedemo za algoritem 3.1, dobimo

$$\begin{aligned}\sum_{j=1}^{n-1}\sum_{i=j+1}^{n}\left(1+\sum_{k=j+1}^{n}2\right) &= \sum_{j=1}^{n-1}(n-j)(1+2(n-j)) = \sum_{j=1}^{n-1}\left(2(n-j)^2+n-j\right)\\ &= \sum_{l=1}^{n-1}(2l^2+l) = 2\frac{(n-1)n(2n-1)}{6}+\frac{(n-1)n}{2}\\ &= \frac{2}{3}n^3-\frac{1}{2}n^2-\frac{1}{6}n = \frac{2}{3}n^3+\mathcal{O}(n^2).\end{aligned}$$

Računanje LU razcepa ima torej kubično časovno zahtevnost. Če velikost matrike povečamo za faktor 2, lahko pričakujemo, da se bo čas računanja LU razcepa povečal za faktor 8. Videli pa bomo, da je to le približno res, saj na čas računanja ne vpliva le število osnovnih operacij.

Zgled 3.7 *Algoritem 3.1 za LU razcep brez pivotiranja odpove, če je pivot enak* 0*, numerično pa odpove tudi, če je pivot blizu* 0*. Če na tri decimalke točno računamo LU razcep* $A = \begin{bmatrix} 0.0001 & 1 \\ 1 & 1 \end{bmatrix}$*, dobimo* $L = \begin{bmatrix} 1 & 0 \\ 10000 & 1 \end{bmatrix}$ *in*

$$U = \begin{bmatrix} 0.0001 & 1 \\ 0 & fl(1-10000) \end{bmatrix} = \begin{bmatrix} 0.0001 & 1 \\ 0 & -10000 \end{bmatrix}.$$

Velja $LU = \begin{bmatrix} 0.0001 & 1 \\ 1 & 0 \end{bmatrix} \neq A$*, napaka pa je ogromna.* □

Rešitev obeh težav je pivotiranje, kjer med algoritmom dopuščamo zamenjavo vrstic (*delno pivotiranje*), lahko pa tudi stolpcev (*kompletno pivotiranje*).

Pri delnem pivotiranju pred eliminacijo v j-tem stolpcu primerjamo elemente $a_{jj}, a_{j+1,j}, \ldots, a_{nj}$ in zamenjamo j-to vrstico s tisto, ki vsebuje element z največjo absolutno vrednostjo. Tako je pri nesingularni matriki v vsakem koraku pivot neničelen.

Kot rezultat dobimo $PA = LU$, kjer je P permutacijska matrika. Zaradi pivotiranja so v matriki L vsi elementi po absolutni vrednosti omejeni z 1.

Za razliko od LU razcepa brez pivotiranja, kjer morajo biti vse vodilne podmatrike nesingularne, za LU razcep z delnim pivotiranjem zadošča že, da je matrika nesingularna.

Izrek 3.13 *Če je A nesingularna, potem obstaja taka permutacijska matrika P, da obstaja LU razcep $PA = LU$, kjer je L spodnja trikotna matrika z enicami na diagonali in U zgornja trikotna matrika.*

Dokaz. Pred uničevanjem elementov v j-tem stolpcu je situacija naslednja:

$$A^{(j-1)} = \begin{bmatrix} U_{11} & U_{12} \\ 0 & A_{22}^{(j-1)} \end{bmatrix}.$$

Ker je matrika A nesingularna, je nesingularna tudi $A^{(j-1)}$, to pa pomeni, da mora biti $A_{22}^{(j-1)}$ nesingularna in ne morejo biti vsi elementi v prvem stolpcu $A_{22}^{(j-1)}$ hkrati enaki 0. ■

Algoritem za izračun LU razcepa z delnim pivotiranjem je predstavljen v algoritmu 3.2.

Algoritem 3.2 LU razcep z delnim pivotiranjem. Začetni podatek je $n \times n$ realna matrika A. Na koncu imamo v zgornjem trikotniku matrike A matriko U, v matriki L pa vse poddiagonalne elemente matrike L iz LU razcepa $PA = LU$.

```
j = 1,...,n-1
    poišči |a_qj| = max_{j≤p≤n} |a_pj|
    zamenjaj vrstici q in j
    i = j+1,...,n
        l_ij = a_ij / a_jj
        k = j+1,...,n
            a_ik = a_ik - l_ij a_jk
```

Dodatno delo, ki ga potrebuje algoritem 3.2 je $\mathcal{O}(n^2)$ primerjanj, kar se pri velikem n sploh ne opazi v primerjavi z $2/3n^3 + \mathcal{O}(n^2)$ osnovnih operacij. V praksi je tako LU razcep z delnim pivotiranjem enako hiter kot LU razcep brez pivotiranja.

Zgled 3.8 *Izračunajmo LU razcep z delnim pivotiranjem za matriko $A = \begin{bmatrix} 0 & 1 & 2 \\ 1 & 2 & 3 \\ 1 & 0 & 1 \end{bmatrix}$. Na začetku definiramo $P = I$, potem pa vsakič, ko zamenjamo vrstici, zamenjavo naredimo tudi v P. Matriko L hranimo v spodnjem trikotniku matrike A.*

$$A^{(1)} = \begin{bmatrix} 1 & 2 & 3 \\ \boxed{0} & 1 & 2 \\ \boxed{1} & -2 & -2 \end{bmatrix}, \quad P = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix},$$

$$A^{(2)} = \begin{bmatrix} 1 & 2 & 3 \\ \boxed{1} & -2 & -2 \\ \boxed{0} & \boxed{-\frac{1}{2}} & 1 \end{bmatrix}, \quad P = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}.$$

Uokvirjeni elementi spadajo v matriko L, na njihovih mestih v matriki A pa so ničle. Dobili smo

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

in $PA = LU$. □

Linearni sistem $Ax = b$ rešimo preko LU razcepa z delnim pivotiranjem po naslednjih korakih:

1) $PA = LU$,
2) $Ly = Pb =: b'$,
3) $Ux = y$.

Sistem $Ly = b'$ rešujemo s *premo substitucijo*. Iz

$$\begin{bmatrix} 1 & & & \\ l_{21} & 1 & & \\ \vdots & \ddots & \ddots & \\ l_{n1} & \cdots & l_{n,n-1} & 1 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} b'_1 \\ b'_2 \\ \vdots \\ b'_n \end{bmatrix}$$

dobimo

$$l_{i1}y_1 + \cdots + l_{i,i-1}y_{i-1} + y_i = b'_i, \quad i = 1, \ldots, n,$$

od tod pa algoritem

$i = 1, \ldots, n$
  $y_i = b'_i - \sum_{j=1}^{i-1} l_{ij}y_j$

Število operacij je $\sum_{i=1}^{n}(1 + 2(i-1)) = n^2$.

Sistem $Ux = y$ rešujemo z *obratno substitucijo*. Iz

$$\begin{bmatrix} u_{11} & \cdots & u_{1n} \\ & \ddots & \vdots \\ & & u_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix}$$

dobimo

$$u_{ii}x_i + u_{i,i+1}x_{i+1} + \cdots + u_{in}x_n = y_i, \quad i = 1, \ldots, n,$$

od tod pa algoritem

$i = n, n-1, \ldots, 1$
  $x_i = \frac{1}{u_{ii}}\left(y_i - \sum_{j=i+1}^{n} u_{ij}x_j\right)$

Število operacij je $\sum_{i=1}^{n}(2 + 2(i-1)) = n^2 + n$. Dodatnih n operacij v primerjavi z reševanjem sistema $Ly = b$ je n deljenj, saj diagonalni elementi matrike U niso enaki 1.

Za LU razcep z delnim pivotiranjem torej porabimo $\frac{2}{3}n^3 + \mathcal{O}(n^2)$ operacij, ko pa L in U že poznamo, za reševanje $Ax = b$ porabimo še dodatnih $2n^2 + \mathcal{O}(n)$ operacij.

Za reševanje sistema $Ax = b$ nikoli ne uporabljamo inverzne matrike A^{-1}, saj:

- za množenje $A^{-1}b$ porabimo $2n^2$ operacij, kar ni ceneje od reševanja trikotnih L in U;
- za računanje A^{-1} potrebujemo $2n^3$ operacij, kar je trikrat toliko kot LU razcep;
- numerične napake so kvečjemu večje.

Tudi kadar je potrebno izračunati $A^{-1}B$, to naredimo tako, da rešimo sistem $AX = B$.

Poleg LU razcepa z delnim pivotiranjem poznamo še *LU razcep s kompletnim pivotiranjem*, kjer v j-tem stolpcu pivotni element izbiramo iz cele podmatrike $A(j:n, j:n)$, nato pa izvedemo zamenjavo vrstic in stolpcev. Na koncu dobimo razcep $PAQ = LU$, kjer sta P in Q permutacijski matriki za vrstice oziroma stolpce. Število operacij je enako kot pri osnovnem LU razcepu, število primerjanj pa je $\mathcal{O}(n^3)$. Ker vsako primerjanje tudi porabi nekaj časa, se toliko primerjanj pozna pri celotnem algoritmu in ga upočasni za toliko, da se v praksi zelo redko uporablja.

Pri kompletnem pivotiranju sistem $Ax = b$ rešimo po naslednjih korakih:

1) $PAQ = LU$,
2) $Ly = Pb =: b'$,
3) $Ux' = y$,
4) $x = Qx'$.

Zgled 3.9 *Sistem $Ax = b$, kjer sta $A = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 2 & 3 \\ 1 & 1 & 1 \end{bmatrix}$ in $b = \begin{bmatrix} 2 \\ 7 \\ 3 \end{bmatrix}$, bomo rešili preko LU razcepa s kompletnim pivotiranjem.*

$$A^{(1)} = \begin{bmatrix} 3 & 2 & 1 \\ \boxed{\frac{1}{3}} & \frac{1}{3} & -\frac{1}{3} \\ \boxed{\frac{1}{3}} & \frac{1}{3} & \frac{2}{3} \end{bmatrix}, \quad P = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad Q = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix},$$

$$A^{(2)} = \begin{bmatrix} 3 & 1 & 2 \\ \boxed{\frac{1}{3}} & \frac{2}{3} & \frac{1}{3} \\ \boxed{\frac{1}{3}} & \boxed{-\frac{1}{2}} & \frac{1}{2} \end{bmatrix}, \quad P = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}, \quad Q = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}.$$

Uokvirjeni elementi so iz L, v A pa so na njihovih mestih ničle. Razcep je $PAQ = LU$, kjer sta $L = \begin{bmatrix} 1 & & \\ \frac{1}{3} & 1 & \\ \frac{1}{3} & -\frac{1}{2} & 1 \end{bmatrix}$ in $U = \begin{bmatrix} 3 & 1 & 2 \\ & \frac{2}{3} & \frac{1}{3} \\ & & \frac{1}{2} \end{bmatrix}$.

Dobimo $b' = Pb = \begin{bmatrix} 7 \\ 3 \\ 2 \end{bmatrix}$, $y = \begin{bmatrix} 7 \\ \frac{2}{3} \\ 0 \end{bmatrix}$, $x' = \begin{bmatrix} 2 \\ 1 \\ 0 \end{bmatrix}$ in $x = \begin{bmatrix} 1 \\ 0 \\ 2 \end{bmatrix}$. □

3.6 Analiza zaokrožitvenih napak pri LU razcepu

Denimo, da smo pri reševanju linearnega sistema $Ax = b$ preko LU razcepa in računanja v plavajoči vejici dobili numerično rešitev $\widehat{x}$. Pri analizi obratne stabilnosti iščemo oceno za normo motnje δA, da je $(A + \delta A)\widehat{x} = b$. Pri izpeljavi ocene predpostavimo, da med računanjem ne

pride do prekoračitve oz. podkoračitve. Prav tako lahko predpostavimo, da smo uporabili LU razcep brez pivotiranja. V primeru pivotiranja si lahko namreč predstavljamo, da smo že predhodno ustrezno preuredili stolpce in vrstice matrike A in desne strani b.

V nadaljevanju oznaka $|A|$ pomeni matriko absolutnih vrednosti z elementi $|a_{ij}|$, zapis $A \leq B$ pa pomeni, da je $a_{ij} \leq b_{ij}$ za $i, j = 1, \ldots, n$. Sistem $Ax = b$ preko LU razcepa rešimo v treh korakih: najprej izračunamo razcep $A = LU$, nato rešimo spodnji trikotni sistem $Ly = b$, na koncu pa dobimo rešitev iz zgornjega trikotnega sistema $Ux = y$. Naslednje tri leme ocenjujejo obratne napake teh treh korakov. Dokaze zanje lahko najdete npr. v [8].

Lema 3.14 *Naj bo L nesingularna spodnja trikotna matrika velikosti $n \times n$. Če sistem $Ly = b$ rešimo s premo substitucijo, potem izračunana rešitev $\widehat{y}$ zadošča enačbi $(L + \delta L)\widehat{y} = b$, kjer je $|\delta L| \leq nu|L|$.*

Lema 3.15 *Naj bo U nesingularna zgornja trikotna matrika velikosti $n \times n$. Če sistem $Ux = y$ rešimo z obratno substitucijo, potem izračunana rešitev $\widehat{x}$ zadošča enačbi $(U + \delta U)\widehat{x} = y$, kjer je $|\delta U| \leq nu|U|$.*

Iz lem 3.14 in 3.15 sledi, da je reševanje trikotnega sistema (zgornjega ali spodnjega) vedno obratno stabilno, saj je norma motnje relativno majhna v primerjavi z normo matrike.

Lema 3.16 *Naj bo A nesingularna matrika velikosti $n \times n$, pri kateri se izvede LU razcep brez pivotiranja. Za izračunani matriki L in U velja $A = LU + E$, kjer je $|E| \leq nu|L| \cdot |U|$.*

Sam LU razcep ni nujno obratno stabilen. Iz leme 3.16 sledi ocena $\|E\|_\infty \leq nu\|L\|_\infty\|U\|_\infty$ in kot bomo videli v nadaljevanju, je $\|L\|_\infty\|U\|_\infty$ lahko mnogo več od $\|A\|_\infty$.

Izrek 3.17 *Za izračunano rešitev $\widehat{x}$ sistema $Ax = b$ preko LU razcepa velja $(A + \delta A)\widehat{x} = b$, kjer je $|\delta A| \leq 3nu|L| \cdot |U|$ oziroma $\|\delta A\|_\infty \leq 3nu\|L\|_\infty\|U\|_\infty$.*

Dokaz. Uporabimo leme 3.14, 3.15 in 3.16. Tako dobimo:

$$\begin{aligned} b &= (L + \delta L)\widehat{y} = (L + \delta L)(U + \delta U)\widehat{x} \\ &= (LU + \delta LU + L\delta U + \delta L\delta U)\widehat{x} \\ &= (A - E + \delta LU + L\delta U + \delta L\delta U)\widehat{x}, \end{aligned}$$

torej je $\delta A = -E + L\delta U + U\delta L + \delta L\delta U$. Ocena je:

$$\begin{aligned} |\delta A| &\leq |E| + |L| \cdot |\delta U| + |\delta L| \cdot |U| + |\delta L| \cdot |\delta U| \\ &\leq nu|L| \cdot |U| + nu|L| \cdot |U| + nu|L| \cdot |U| + n^2u^2|L| \cdot |U| \approx 3nu|L| \cdot |U|, \end{aligned}$$

od tod pa sledi $\|\delta A\|_\infty \leq 3nu\|L\|_\infty\|U\|_\infty$. ∎

Zanima nas, kdaj je $3nu\|L\|_\infty\|U\|_\infty = \mathcal{O}(u)\|A\|_\infty$, saj bo v tem primeru metoda obratno stabilna. Če uporabimo LU razcep brez pivotiranja, potem je tudi iz zgleda 3.7 razvidno, da je norma $\|L\|_\infty$ lahko poljubno velika, zato reševanje preko LU razcepa brez pivotiranja ni obratno stabilno, razen če ima matrika kakšno posebno lastnost, kot sta npr. diagonalna dominantnost po stolpcih in simetrična pozitivna definitnost.

Pri delnem in kompletnem pivotiranju velja $|l_{ij}| \leq 1$, posledica pa je $\|L\|_\infty \leq n$. Sedaj moramo oceniti še normo matrike U. Vpeljemo količino

$$g := \frac{\max |u_{ij}|}{\max |a_{ij}|},$$

ki jo imenujemo *pivotna rast*. Sedaj lahko ocenimo $\|U\|_\infty \leq ng\|A\|_\infty$ in tako za delno in kompletno pivotiranje dobimo oceno

$$\|\delta A\|_\infty \leq 3gn^3u\|A\|_\infty. \tag{3.2}$$

Tudi če je pivotna rast g majhna je ocena lahko zelo velika in je ponavadi dosti slabša od dejanskih rezultatov. Vseeno pa nam ocena (3.2) pove, da lahko pričakujemo majhno obratno napako, če bo pivotna rast majhna.

Lema 3.18 *Pri delnem pivotiranju je pivotna rast omejena z* 2^{n-1}.

Dokaz. Zaradi $a_{jk} = a_{jk} - l_{ji}a_{ik}$ in $|l_{ij}| \leq 1$ se lahko absolutna vrednost največjega elementa v matriki v vsakem koraku LU razcepa podvoji. Ker se vsak element v matriki spremeni največ $(n-1)$-krat, za pivotno rast velja $g \leq 2^{n-1}$. ∎

Ker matrike s pivotno rastjo 2^{n-1} tudi v resnici obstajajo za poljuben n, primer so npr. matrike oblike

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

zaradi tega LU razcep z delnim pivotiranjem teoretično ni obratno stabilen.

S statističnimi testi so ugotovili, da je pri delnem pivotiranju običajna pivotna rast velikostnega razreda $\mathcal{O}(n^{2/3})$, kar pomeni, da je LU razcep z delnim pivotiranjem v praksi obratno stabilen. Ker je verjetnost, da bomo v praksi naleteli na matriko z veliko pivotno rastjo, zelo majhna, se v številnih numeričnih paketih, vključno z Matlabom, za reševanje linearnih sistemov uporablja ravno LU razcep z delnim pivotiranjem.

Lema 3.19 *Pri kompletnem pivotiranju je pivotna rast omejena z*

$$g \leq \left(n \cdot 2 \cdot 3^{1/2} \cdot 4^{1/3} \cdots n^{1/(n-1)}\right)^{\frac{1}{2}} \approx n^{\frac{1}{2}+\frac{\ln n}{4}}.$$

Zaradi zgornje ocene je reševanje linearnega sistema preko LU razcepa s kompletnim pivotiranjem obratno stabilno. Omenimo še, da je točna zgornja meja za pivotno rast pri kompletnem pivotiranju še vedno odprt problem. Dolgo časa so domnevali, da je pivotna rast pri kompletnem pivotiranju navzgor omejena z n, a so domnevo ovrgli leta 1991, ko so našli matriko velikosti 13×13 s pivotno rastjo 13.02.

S statističnimi testi so ugotovili, da je pri kompletnem pivotiranju običajna pivotna rast velikostnega razreda $\mathcal{O}(n^{1/2})$, kar pomeni, da je kompletno pivotiranje tudi v praksi boljše od delnega pivotiranja.

Kljub temu, da je LU razcep s kompletnim pivotiranjem obratno stabilen, se ga le redko uporablja, saj vsebuje preveč primerjanj. Če želimo linearni sistem rešiti z obratno stabilno metodo, imamo na voljo še stabilnejše metode, kot je LU razcep. V 5. poglavju bomo tako spoznali QR razcep, ki je obratno stabilen in porabi dvakrat toliko operacij kot LU razcep z delnim pivotiranjem.

3.7 Ostanki

Denimo, da smo numerično rešili linearni sistem $Ax = b$ in dobili približek $\widetilde{x}$ za točno rešitev. Kako ugotovimo, ali je dobljena rešitev natančna?

Izračunamo lahko ostanek $r = b - A\widetilde{x}$ in pogledamo njegovo normo $\|r\|$. Ker se ostanek spremeni, če sistem $Ax = b$ pomnožimo s poljubnim skalarjem, rešitev x pa ostane nespremenjena, je smiselno gledati *relativni ostanek*

$$\frac{\|r\|}{\|A\| \cdot \|\widetilde{x}\|}.$$

Zgled 3.10 *V Matlabu rešimo sistema z matrikama* `A1=hilb(10)` *in* `A2=A1+eye(10)`*. Za desni strani vzamemo* `b1=A1*x0` *in* `b2=A2*x0`*, kjer je* `x0=ones(10,1)`*. Napaki numerično izračunanih rešitev* `x1=A1\b1` *in* `x2=A2\b2` *sta* $\|x_1 - x_0\|_2 = 8.7 \cdot 10^{-4}$ *in* $\|x_2 - x_0\|_2 = 8.9 \cdot 10^{-16}$*, kar se ujema z občutljivostima* $\kappa_2(A_1) = 1.6 \cdot 10^{13}$ *in* $\kappa_2(A_2) = 2.8 \cdot 10^{0}$*. Vendar, če izračunamo relativna ostanka, dobimo*

$$\frac{\|b_1 - A_1x_1\|_2}{\|A_1\|_2 \cdot \|x_1\|_2} = 9.4 \cdot 10^{-17} \quad in \quad \frac{\|b_2 - A_2x_2\|_2}{\|A_2\|_2 \cdot \|x_2\|_2} = 9.2 \cdot 10^{-17}.$$

Oba relativna ostanka sta majhna in očitno ne povesta ničesar o tem, ali je rešitev točna. □

Kot kaže zgornji zgled, lahko pričakujemo majhen relativni ostanek tudi kadar je numerični približek daleč od točne rešitve. To lahko razložimo z analizo obratne stabilnosti. Če je $\widetilde{x}$ točna rešitev sistema $(A + \delta A)\widetilde{x} = b$, potem sledi $\|r\| \leq \|\delta A\| \cdot \|\widetilde{x}\|$ oziroma

$$\frac{\|r\|}{\|A\| \cdot \|\widetilde{x}\|} \leq \frac{\|\delta A\|}{\|A\|}.$$

Ker so algoritmi, ki jih v praksi uporabljamo za reševanje linearnih sistemov, vsi obratno stabilni, je relativni ostanek vedno majhen. Velik relativni ostanek bi namreč pomenil veliko obratno napako in algoritem, s katerim smo dobili rešitev, potem ne bi bil obratno stabilen. Zaradi tega sam relativni ostanek ni pravo merilo za točnost dobljene rešitve.

Iz $x - \widetilde{x} = A^{-1}r$ sledi, da je napaka izračunane rešitve $\widetilde{x}$ z relativnim ostankom povezana z oceno

$$\frac{\|\widetilde{x} - x\|}{\|\widetilde{x}\|} \leq \kappa(A) \frac{\|r\|}{\|A\| \cdot \|\widetilde{x}\|}. \tag{3.3}$$

To pomeni, da iz majhnega relativnega ostanka lahko sklepamo na majhno relativno napako rešitve samo tedaj, ko je matrika A dobro pogojena. Merilo za to je občutljivost $\kappa(A)$, za katero pa potrebujemo $\|A^{-1}\|$. Na srečo obstajajo algoritmi, s katerimi lahko dobimo dovolj dobro oceno za $\|A^{-1}\|$ brez eksplicitnega izračuna A^{-1}. Če bi namreč izračunali A^{-1}, bi za to potrebovali več operacij, kot pa jih sicer potrebujemo, da izračunamo rešitev linearnega sistema $Ax = b$.

Iz $x - \widetilde{x} = A^{-1}r$ sledi tudi zelo uporabna aposteriorna ocena

$$\frac{\|\widetilde{x} - x\|}{\|\widetilde{x}\|} \leq \frac{\|A^{-1}\| \cdot \|r\|}{\|\widetilde{x}\|}, \tag{3.4}$$

ki jo uporabljamo v kombinaciji z algoritmi, ki ocenijo normo $\|A^{-1}\|$ brez računanja A^{-1}. Trenutno najboljši algoritem za ocenjevanje $\|A^{-1}\|$, ki je podrobno opisan npr. v [8], je *Hagerjeva cenilka* [7]. Če že poznamo faktorja L in U iz LU razcepa matrike A, potem Hagerjeva cenilka v času $\mathcal{O}(n^2)$ vrne spodnjo mejo za $\|A^{-1}\|_\infty$, ki ponavadi od točne vrednosti ne odstopa za več kot faktor 10. To je povsem dovolj, saj nam za oceno napake zadošča le velikostni razred.

Ocena po normi (3.4) je dostikrat preveč pesimistična. Izpeljati se da t.i. oceno po komponentah

$$\frac{\|\widetilde{x} - x\|_\infty}{\|\widetilde{x}\|_\infty} \leq \frac{\||A^{-1}| \cdot |r|\|_\infty}{\|\widetilde{x}\|_\infty}, \tag{3.5}$$

ki je kvečjemu boljša od ocene (3.4). Za oceno $\||A^{-1}| \cdot |r|\|_\infty$ lahko spet uporabimo Hagerjevo cenilko, saj za $r = [\, r_1 \quad \cdots \quad r_n \,]^T$ in $R = \mathrm{diag}(r_1, \ldots, r_n)$ velja $\||A^{-1}|r\|_\infty = \|A^{-1}R\|_\infty$.

Zgled 3.11 *V Matlabu vzamemo* `A0=invhilb(8), x0=ones(8,1)` *in* `b0=A0*x0`. *A_0 in b_0 imata celoštevilske predstavljive elemente in x_0 je eksaktna rešitev. Sedaj A_0 in b_0 pomnožimo z diagonalno matriko* `D=diag(5.^(1:8))` *v* `A=D*A0` *in* `b=D*b0`. *Sistem numerično rešimo z* `x=A\b` *in dobimo* $\frac{\|x-x_0\|_\infty}{\|x\|_\infty} = 9.3 \cdot 10^{-9}$. *Glede na to, da za občutljivost matrike A velja $\kappa_\infty(A) = 2.2 \cdot 10^{14}$, je napaka celo manjša od pričakovane. Kako dobre so aposteriorne ocene za $\delta = \frac{\|x-x_0\|_\infty}{\|x\|_\infty}$:*

- *relativni ostanek $\|r\|_\infty / (\|A\|_\infty \cdot \|x\|_\infty)$ je enak $1.6 \cdot 10^{-17}$, kar nam ne pove ničesar,*
- *ocena po normi (3.4) vrne $\delta \leq 3.5 \cdot 10^{-3}$ in je preveč pesimistična,*
- *ocena po komponentah (3.5) vrne $\delta \leq 2.6 \cdot 10^{-8}$ in se dobro ujema s pravo napako.* □

3.8 Sistemi s posebno obliko

Kadar ima matrika A kakšne dodatne lastnosti, kot je npr. posebna oblika, lahko to izkoristimo in prihranimo tako pri številu operacij kot pri porabi pomnilnika.

3.8.1 Simetrične matrike

Če je matrika $A \in \mathbb{R}^{n \times n}$ simetrična, potem za zapis matrike porabimo polovico manj prostora kot za splošno matriko velikosti $n \times n$. Kot bomo videli v nadaljevanju, pa lahko tudi pri reševanju linearnega sistema prihranimo polovico operacij v primerjavi z LU razcepom.

Najprej poglejmo, kako lahko učinkovito rešimo linearni sistem $Ax = b$, če je matrika A poleg simetričnosti še pozitivno definitna, kar pomeni, da je $x^T Ax > 0$ za vsak $x \neq 0$.

[7] Ameriški matematik William W. Hager je metodo objavil leta 1984.

Izrek 3.20 *Velja:*

1) *Vsaka vodilna podmatrika simetrične pozitivno definitne matrike je simetrična pozitivno definitna.*
2) *Če je A simetrična pozitivno definitna matrika, potem se izvede LU razcep brez pivotiranja in diagonalni elementi matrike U so strogo pozitivni.*
3) *Matrika A je simetrična pozitivno definitna natanko tedaj, ko obstaja taka nesingularna spodnja trikotna matrika V s pozitivnimi elementi na diagonali, da je $A = VV^T$.*

Razcep $A = VV^T$ imenujemo *razcep Choleskega*, V pa *faktor Choleskega*[8].

Dokaz.

1) Naj bo H vodilna $k \times k$ podmatrika matrike A. Matrika H je očitno simetrična. Poljuben neničelen vektor $x \in \mathbb{R}^k$ lahko razširimo v vektor $y \in \mathbb{R}^n$ oblike $y = \begin{bmatrix} x \\ 0 \end{bmatrix}$. Iz $x^T Hx = y^T Ay > 0$ sledi, da je H pozitivno definitna.
2) Po točki 1) so vse vodilne podmatrike matrike A nesingularne, zato po izreku 3.12 za A obstaja LU razcep brez pivotiranja. Ker ima simetrična pozitivno definitna matrika pozitivno determinanto, mora veljati $u_{11}u_{22}\cdots u_{kk} = \det(A(1:k,1:k)) > 0$ za vsak $k = 1,\ldots,n$, od tod pa sledi, da so vsi diagonalni elementi matrike U strogo pozitivni.
3) Naj bo A simetrična pozitivno definitna matrika. Iz točke 2) sledi, da obstaja LU razcep brez pivotiranja $A = LU$. To zapišemo kot $A = LDM$, kjer je $D = \operatorname{diag}(u_{11},\ldots,u_{nn})$ in M zgornja trikotna matrika z enicami na diagonali. Ker je A simetrična, sta $L(DM)$ in $M^T(DL^T)$ dva LU razcepa, ki pa je za nesingularno matriko enoličen. To pomeni $M = L^T$ in $A = LDL^T$. Če vzamemo $V = LD^{1/2}$, kjer je $D^{1/2} = \operatorname{diag}(u_{11}^{1/2},\ldots,u_{nn}^{1/2})$, dobimo iskani razcep $A = VV^T$.

   Dokaz v drugo smer je trivialen, saj je za nesingularno matriko V matrika VV^T očitno simetrična pozitivno definitna. ∎

Algoritem za razcep Choleskega dobimo tako, da iz zveze $A = VV^T$ zapišemo enačbe za a_{jk}, $j \geq k$, kjer dobimo

$$a_{jk} = \sum_{i=1}^{k} v_{ji}v_{ki} = \sum_{i-1}^{k-1} v_{ji}v_{ki} + v_{jk}v_{kk}.$$

Če uporabimo pravilni vrstni red (npr. po vrsticah ali po stolpcih), potem lahko iz zgornjih enačb po vrsti izračunamo vse elemente matrike V. Različica, kjer računamo po stolpcih, je zapisana v algoritmu 3.3.

Število osnovnih operacij, ki jih porabi algoritem 3.3 je

$$\sum_{k=1}^{n}(2k + 2(n-k)k) = \frac{1}{3}n^3 + \mathcal{O}(n^2).$$

[8] André-Louis Cholesky se je rodil leta 1875 v Franciji. Delal je kot geodet v francoski vojski. Pri svojem delu je reševal linearne probleme najmanjših kvadratov, kjer nastopajo simetrični pozitivno definitni sistemi (glej 5. poglavje). Umrl je na fronti leta 1918, njegove zapiske z razcepom pa je po smrti v objavo poslal njegov sodelavec.

Ker njegova družina izvira iz Ukrajine, so prisotne nejasnosti glede pravilne izgovorjave njegovega priimka. Po francoskih biografih je pravilna izgovorjava "Šoleski".

Algoritem 3.3 Razcep Choleskega. Začetni podatek je $n \times n$ simetrična pozitivno definitna matrika A. Algoritem vrne spodnjo trikotno matriko V, da je $A = VV^T$.

$k = 1, \ldots, n$
  $v_{kk} = \left(a_{kk} - \sum_{i=1}^{k-1} v_{ki}^2\right)^{1/2}$
  $j = k+1, \ldots, n$
    $v_{jk} = \frac{1}{v_{kk}} \left(a_{jk} - \sum_{i=1}^{k-1} v_{ji} v_{ki}\right)$

Torej res porabimo za polovico manj operacij kot pri LU razcepu.

Če matrika A ni pozitivno definitna, se v algoritmu 3.3 pod kvadratnim korenom pojavi nepozitivna vrednost. Računanje razcepa Choleskega je tako najcenejša numerična metoda za ugotavljanje pozitivne definitnosti simetrične matrike.

Zgled 3.12 *Razcep Choleskega za matriko*

$$A = \begin{bmatrix} 4 & -2 & 4 & -2 & 4 \\ -2 & 10 & 1 & -5 & -5 \\ 4 & 1 & 9 & -2 & 1 \\ -2 & -5 & -2 & 22 & 7 \\ 4 & -5 & 1 & 7 & 14 \end{bmatrix}$$

je $A = VV^T$, *kjer je*

$$V = \begin{bmatrix} 2 & & & & \\ -1 & 3 & & & \\ 2 & 1 & 2 & & \\ -1 & -2 & 1 & 4 & \\ 2 & -1 & -1 & 2 & 2 \end{bmatrix}.$$

□

Podobno kot pri LU razcepu, tudi reševanje simetričnega pozitivno definitnega sistema $Ax = b$ izvedemo po naslednjih korakih:

1) $A = VV^T$,
2) $Vy = b$,
3) $V^T x = y$.

Naj bo $\widehat{x}$ numerično izračunana rešitev simetričnega pozitivno definitnega sistema $Ax = b$, ki smo ga rešili preko razcepa Choleskega. Izpeljemo lahko podobno analizo, kot smo jo naredili za reševanje linearnega sistema preko LU razcepa. Tako ugotovimo, da $\widehat{x}$ zadošča sistemu $(A + \delta A)\widehat{x} = b$, kjer je $|\delta A| \leq 3nu|V| \cdot |V^T|$. Ker pa je

$$(|V| \cdot |V^T|)_{ij} = \sum_{k=1}^{\min(i,j)} |v_{ik}||v_{jk}| \leq \left(\sum_{k=1}^{i} |v_{ik}|^2\right)^{1/2} \left(\sum_{k=1}^{j} |v_{jk}|^2\right)^{1/2} = \sqrt{a_{ii}}\sqrt{a_{jj}} \leq \max_{i,j} |a_{ij}|,$$

velja $\||V| \cdot |V^T|\|_\infty \leq n\|A\|_\infty$ in

$$\|\delta A\|_\infty \leq 3n^2 u \|A\|_\infty.$$

To pomeni, da je reševanje simetričnega pozitivno definitnega sistema preko razcepa Choleskega obratno stabilno, ocena pa je ugodnejša od ocene pri LU razcepu. Torej je razcep Choleskega cenejši in tudi stabilnejši od LU razcepa.

Če je matrika A simetrična, ne pa tudi definitna, ne moremo uporabiti razcepa Choleskega, LU pa razcepa ne želimo uporabiti, saj ne ohranja simetrije. Za nesingularno simetrično matriko A obstaja razcep $PAP^T = LDL^T$, kjer je P permutacijska matrika, L spodnja trikotna matrika z enicami na diagonali, D pa bločna diagonalna matrika z bloki 1×1 ali 2×2. Zgled za to, da res potrebujemo 2×2 bloke v matriki D, je npr. matrika

$$A = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix},$$

ki se je ne da zapisati kot $A = LDL^T$, kjer je D diagonalna, L pa spodnja trikotna matrika.

Obstaja več algoritmov za izračun razcepa $PAP^T = LDL^T$, ki se ločijo v tem, kako izbiramo pivotne elemente in koliko primerjanj porabimo za to. Tako poznamo Bunch–Kaufmannovo in Bunch–Parlettovo metodo. Še ena možnost je Aasenova metoda, kjer matriko A razcepimo v obliko $PAP^T = LTL^T$, kjer je T tridiagonalna matrika. Vse zgornje metode potrebujejo $\frac{n^3}{3} + \mathcal{O}(n^2)$ osnovnih operacij za izračun razcepa, podrobne algoritme in ocene za obratno stabilnost pa lahko najdete npr. v [9].

Tako tudi v primeru, ko je matrika simetrična, ne pa tudi definitna, lahko prihranimo polovico časa in prostora v primerjavi z navadnim LU razcepom.

3.8.2 Tridiagonalne matrike

Denimo, da rešujemo linearni sistem s tridiagonalno matriko A. To pomeni, da ima A obliko

$$A = \begin{bmatrix} a_1 & b_1 & & & \\ c_2 & a_2 & b_2 & & \\ & \ddots & \ddots & \ddots & \\ & & c_{n-1} & a_{n-1} & b_{n-1} \\ & & & c_n & a_n \end{bmatrix}$$

in za predstavitev potrebujemo le tri vektorje dolžine n za diagonalne in obdiagonalne elemente. Pri LU razcepu brez pivotiranja dobimo

$$L = \begin{bmatrix} 1 & & & \\ l_2 & 1 & & \\ & \ddots & \ddots & \\ & & l_n & 1 \end{bmatrix} \quad \text{in} \quad U = \begin{bmatrix} u_1 & b_1 & & \\ & \ddots & \ddots & \\ & & u_{n-1} & b_{n-1} \\ & & & u_n \end{bmatrix}.$$

Za razcep in nadaljnje reševanje sistema $Ax = b$ potrebujemo le $\mathcal{O}(n)$ operacij in $\mathcal{O}(n)$ prostora, saj shranimo le neničelne elemente matrik A, L in U.

Pri delnem pivotiranju dobimo

$$U = \begin{bmatrix} u_1 & v_1 & w_1 & & \\ & \ddots & \ddots & \ddots & \\ & & u_{n-2} & v_{n-2} & w_{n-2} \\ & & & u_{n-1} & v_{n-1} \\ & & & & u_n \end{bmatrix},$$

pivotna rast pa je omejena z 2. To pomeni, da je reševanje tridiagonalnega sistema preko LU razcepa z delnim pivotiranjem obratno stabilno.

Podobno velja za pasovne matrike, ki imajo poleg glavne še p diagonal nad in q diagonal pod glavno diagonalo. Če uporabimo LU razcep brez pivotiranja, bosta imeli matriki L in U enako obliko v spodnjem oz. zgornjem trikotniku kot jo ima matrika A. Če sta $p, q \ll n$, je reševanje takih sistemov preko LU razcepa z delnim pivotiranjem obratno stabilno.

3.8.3 Kompleksni sistemi

Do sedaj smo predpostavili, da sta matrika A in desna stran b realni. Kaj pa, če moramo rešiti linearni sistem $Ax = b$, kjer je $A \in \mathbb{C}^{n\times n}$ in $x, b \in \mathbb{C}^n$?

Če nimamo na voljo kompleksne aritmetike, lahko kompleksni sistem prevedemo na dvakrat večji realni sistem

$$\begin{bmatrix} A_1 & -A_2 \\ A_2 & A_1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix},$$

kjer je $A = A_1 + iA_2$, $x = x_1 + ix_2$ in $b = b_1 + ib_2$. Sedaj lahko uporabimo podprogram za reševanje realnega linearnega sistema. Ker je sistem dvakrat večji, za reševanje preko LU razcepa z delnim pivotiranjem porabimo $\frac{16}{3}n^3 + \mathcal{O}(n^2)$ realnih operacij.

Če pa je možno računati v kompleksni aritmetiki, potem lahko uporabimo standardni algoritem z LU razcepom z delnim pivotiranjem. Pri primerjavi računske zahtevnosti moramo upoštevati, da se kompleksna aritmetika v resnici izvaja z realno aritmetiko na parih realnih števil. Če pogledamo, kako izračunamo osnovne štiri računske operacije za kompleksna števila, lahko opazimo, da imajo različne zahtevnosti. Tako za seštevanje in odštevanje porabimo 2 osnovni realni operaciji, za množenje porabimo 6, za deljenje pa kar 11 osnovnih realnih operacij.

Iz algoritma 3.1 lahko razberemo, da vsebuje $n^3/3 + \mathcal{O}(n^2)$ kompleksnih odštevanj in prav toliko množenj, medtem ko je deljenj le $\mathcal{O}(n^2)$. Zaradi tega skupno porabimo $\frac{8}{3}n^3 + \mathcal{O}(n^2)$ realnih operacij. Ta način je torej za polovico cenejši od prevedbe na večji realni sistem, hkrati pa porabi tudi za polovico manj pomnilnika.

3.8.4 Razpršene matrike

Matrika je *razpršena*, če je večina njenih elementov enakih 0, ostali pa nimajo kakšne posebne strukture. Pri taki matriki shranimo le indekse in vrednosti neničelnih elementov. To nam omogoča, da v pomnilnik shranimo razpršene matrike zelo velikih dimenzij.

Vendar, ko želimo tak sistem rešiti, se izkaže, da so pri LU razcepu razpršene matrike oziroma pri razcepu Choleskega za simetrično pozitivno definitno razpršeno matriko faktorji L, U oz. V lahko daleč od razpršenosti.

V nekaterih primerih pomaga, če stolpce in vrstice predhodno tako preuredimo, da pri razcepu nastane čim manj novih neničelnih elementov. Obstajajo različni algoritmi in pristopi, kako preurediti elemente, ki delujejo za določene tipe matrik, ponavadi pa vseeno sisteme z razpršenimi matrikami rešujemo z iterativnimi in ne z direktnimi metodami. Pri iterativnih metodah, ki jih obravnavamo v **??**. poglavju, dobimo zaporedje vektorjev, ki konvergira proti rešitvi linearnega sistema.

Matlab

Za računanje norme je na voljo ukaz `norm`. Uporaba:

- `norm(A)` ali `norm(A,2)`: spektralna ali 2-norma $\|A\|_2$,
- `norm(A,1)`: 1-norma $\|A\|_1$,
- `norm(A,'inf')`: ∞-norma $\|A\|_\infty$,
- `norm(A,'fro')`: Frobeniusova norma $\|A\|_F$.

Za reševanje linearnega sistema $Ax = b$ uporabimo v Matlabu operator \ v obliki `x=A\b`. Glede na lastnosti matrike A Matlab izbere optimalno metodo. Za splošno matriko uporabi LU razcep z delnim pivotiranjem, sicer pa, če je matrika trikotna, uporabi premo ali obratno substitucijo, če je matrika simetrična in pozitivno definitna, uporabi razcep Choleskega, če pa je samo simetrična in ni definitna, uporabi Bunch–Kaufmannovo[9] metodo.

LU razcep z delnim pivotiranjem dobimo z ukazom `lu`. Uporaba:

- `[L,U,P]=lu(A)`: L je spodnja trikotna matrika z enicami na diagonali, U je zgornja trikotna matrika in P permutacijska matrika, da je $LU = PA$.
- `[L,U]=lu(A)`: U je zgornja trikotna, L pa po vrsticah premešana spodnja trikotna matrika z enicami na diagonali, da je $LU = A$.

Razcep Choleskega za simetrično pozitivno definitno matriko dobimo z ukazom `chol`. Uporaba:

- `V=chol(A)`: V je taka zgornja trikotna matrika, da je $V^TV = A$. Če matrika A ni simetrična in pozitivno definitna, potem Matlab javi napako.

Za izračun oziroma oceno občutljivosti imamo na voljo naslednje ukaze:

- `cond(A)` ali `cond(A,2)` izračuna $\kappa_2(A)$ preko singularnih vrednosti.
- `cond(A,1)` izračuna $\kappa_1(A)$, uporablja `inv(A)` in porabi manj dela kot `cond(A,2)`.
- `cond(A,'inf')` izračuna $\kappa_\infty(A)$, ukaz je ekvivalenten `cond(A',1)`.
- `condest(A)` vrne oceno za $\kappa_1(A)$, ki jo izračuna po Highamovi[10] izboljšavi Hagerjevega algoritma.

[9] Metoda za reševanje simetričnih nedefinitnih sistemov, ki sta jo James Raymond Bunch in Linda Carol Kaufmann objavila leta 1977, porabi $n^3/3$ operacij za simetrični sistem velikosti $n \times n$.

[10] Nicholas John Higham (r. 1961) je znan angleški numerični matematik.

Dodatna literatura

Za reševanje linearnih sistemov je na voljo obsežna literatura. V slovenščini sta na voljo knjigi [5] in [8]. Kar se tiče tuje literature, lahko skoraj vse algoritme, ki jih potrebujemo pri reševanju linearnih sistemov, skupaj s potrebno analizo, najdete v [9]. Podrobna analiza zaokrožitvenih napak je za vse algoritme narejena v [15]. Zelo lepo napisana učbenika s številnimi algoritmi in primeri sta tudi [7] in [13].

Poglavje 4

Nelinearni sistemi

4.1 Uvod

Rešiti želimo sistem nelinearnih enačb, ki ima obliko

$$\begin{aligned} f_1(x_1, x_2, \ldots, x_n) &= 0 \\ f_2(x_1, x_2, \ldots, x_n) &= 0 \\ &\vdots \\ f_n(x_1, x_2, \ldots, x_n) &= 0. \end{aligned}$$

Krajše pišemo $F(x) = 0$, kjer je $x \in \mathbb{R}^n$ in $F : \mathbb{R}^n \to \mathbb{R}^n$.

Zgled 4.1 *Za testni nelinearni sistem bomo vzeli*

$$\begin{aligned} 3x_1 - \cos(x_1 x_2) - 0.6 &= 0 \\ x_1^2 - 81(x_2 + 0.1)^2 + \sin(x_3) + 1.1 &= 0 \\ e^{-x_1 x_2} + 20x_3 + 9.1 &= 0, \end{aligned} \tag{4.1}$$

ki ima rešitev $\alpha = (0.533332, 0.0054539, -0.504855)$. □

Več kot imamo enačb, težje pridemo do začetnih približkov, če ne poznamo še ozadja sistema, ki ga rešujemo.

V primeru ene same enačbe $f(x) = 0$ lahko iz grafa določimo začetni približek.

V primeru dveh enačb $f_1(x_1, x_2) = 0$ in $f_2(x_1, x_2) = 0$ iščemo presečišče dveh implicitno podanih krivulj. Uporabimo lahko metode za implicitno risanje krivulj in spet grafično določimo začetni približek. Če se da, lahko tudi iz ene enačbe izrazimo eno spremenljivko in jo vstavimo v drugo enačbo, da dobimo eno samo nelinearno enačbo.

Če imamo več enačb, je situacija bolj zapletena. Nekaj možnosti:

- redukcija na manjši sistem,
- aproksimacija z linearnim modelom,

- uporaba variacijskih metod (glej razdelek 4.5),
- metoda zveznega nadaljevanja (glej razdelek 9.10).

4.2 Jacobijeva iteracija

Prva metoda je posplošitev navadne iteracije. Sistem $F(x) = 0$ zapišemo v ekvivalentni obliki $x = G(x)$, kjer je $G : \mathbb{R}^n \to \mathbb{R}^n$, izberemo $x^{(0)} \in \mathbb{R}^n$ in tvorimo zaporedje približkov:

$$x^{(r+1)} = G(x^{(r)}), \quad r = 0, 1, \ldots.$$

Dobljeni metodi pravimo *Jacobijeva*[1] *iteracija*.

Zgled 4.2 *Če v našem testnem primeru (4.1) iz i-te enačbe izrazimo x_i, dobimo*

$$\begin{aligned} x_1^{(r+1)} &= \frac{1}{3}\left(\cos(x_1^{(r)} x_2^{(r)}) + 0.6\right) \\ x_2^{(r+1)} &= \frac{1}{9}\sqrt{(x_1^{(r)})^2 + \sin(x_3^{(r)}) + 1.1} - 0.1 \\ x_3^{(r+1)} &= -\frac{1}{20}\left(e^{-x_1^{(r)} x_2^{(r)}} + 9.1\right), \end{aligned} \tag{4.2}$$

od koder lahko razberemo $G(x)$. Če vzamemo začetni približek $x^{(0)} = (0.4, 0.1, -0.4)$, potem dobimo zaporedje

r	$x_1^{(r)}$	$x_2^{(r)}$	$x_3^{(r)}$	$\|\boldsymbol{x}^{(r)} - \boldsymbol{x}^{(r-1)}\|_\infty$
1	0.533066702220	0.003672183829	−0.503039471957	$1.3 \cdot 10^{-1}$
2	0.533332694686	0.005530371351	−0.504902219788	$1.9 \cdot 10^{-3}$
3	0.533331883381	0.005451521829	−0.504852740886	$7.9 \cdot 10^{-5}$
4	0.533331924436	0.005454006133	−0.504854837609	$2.5 \cdot 10^{-6}$
5	0.533331923152	0.005453901274	−0.504854771543	$1.0 \cdot 10^{-7}$
6	0.533331923206	0.005453904579	−0.504854774331	$3.3 \cdot 10^{-9}$
7	0.533331923204	0.005453904439	−0.504854774243	$1.4 \cdot 10^{-10}$
8	0.533331923204	0.005453904444	−0.504854774247	$4.4 \cdot 10^{-12}$

□

Izrek 4.1 *Če obstaja območje $\Omega \subset \mathbb{R}^n$ z lastnostmi:*

a) $x \in \Omega \Rightarrow G(x) \in \Omega$,

b) $x \in \Omega \Rightarrow \rho(JG(x)) \leq q < 1$, *kjer je $JG(x)$ Jacobijeva matrika*

$$JG(x) = \begin{bmatrix} \frac{\partial g_1(x)}{\partial x_1} & \cdots & \frac{\partial g_1(x)}{\partial x_n} \\ \vdots & & \vdots \\ \frac{\partial g_n(x)}{\partial x_1} & \cdots & \frac{\partial g_n(x)}{\partial x_n} \end{bmatrix}$$

in ρ spektralni radij matrike (največja izmed absolutnih vrednosti lastnih vrednosti matrike),

[1] Znani nemški matematik Carl Gustav Jacob Jacobi (1804-1851) je metodo objavil za reševanje linearnih sistemov (glej **??**. poglavje) leta 1845.

potem ima enačba $G(x) = x$ na Ω natanko eno rešitev α, zaporedje $x^{(r+1)} = G(x^{(r)})$, $r = 0, 1, \ldots$, pa za vsak $x^{(0)} \in \Omega$ konvergira k α.

Podobno kot pri eni nelinearni enačbi je red konvergence odvisen od Jacobijeve matrike. V primeru $JG(\alpha) = 0$ dobimo vsaj kvadratično konvergenco, ta pogoj pa je izpolnjen npr. pri Newtonovi metodi, ki je posplošitev tangentne metode.

Ker za vsako lastno vrednost λ matrike A in poljubno matrično normo velja $|\lambda| \leq \|A\|$, je zadostni pogoj za konvergenco, da je $\|JG(x)\| < 1$. Če vzamemo $\|.\|_\infty$, dobimo naslednjo posledico.

Posledica 4.2 *Če obstaja območje $\Omega \subset \mathbb{R}^n$ z lastnostima:*

a) $x \in \Omega \Rightarrow G(x) \in \Omega$,

b) $\displaystyle\sum_{k=1}^{n} \left| \frac{\partial g_j(x)}{\partial x_k} \right| \leq m < 1, \quad j = 1, \ldots, n,$

potem ima enačba $G(x) = x$ na Ω natanko eno rešitev α, zaporedje $x^{(r+1)} = G(x^{(r)})$, $r = 0, 1, \ldots$, za vsak $x^{(0)} \in \Omega$ konvergira k α in velja ocena

$$\|x^{(r)} - \alpha\|_\infty \leq \frac{m^r}{1-m} \|x^{(1)} - x^{(0)}\|_\infty.$$

Zgled 4.3 *Za testni primer (4.1) in iteracijo (4.2) dobimo*

$$JG(\boldsymbol{x}) = \begin{bmatrix} -\frac{x_2}{3}\sin(x_1x_2) & -\frac{x_1}{3}\sin(x_1x_2) & 0 \\ \frac{x_1}{9\sqrt{x_1^2+\sin x_3+1.1}} & 0 & \frac{\cos x_3}{9\sqrt{x_1^2+\sin x_3+1.1}} \\ -\frac{x_2}{20}e^{-x_1x_2} & -\frac{x_1}{20}e^{-x_1x_2} & 0 \end{bmatrix}.$$

Če vzamemo $\Omega = [-1, 1] \times [-1, 1] \times [-1, 1]$, potem za $x \in \Omega$ velja $G(x) \in \Omega$. Če ocenimo absolutne vrednosti parcialnih odvodov G na Ω, dobimo $\|G(x)\|_\infty \leq m$ za $m = 0.561$ in konvergenca je zagotovljena.

Iz ocene $\|x^{(8)} - \alpha\|_\infty \leq \frac{m^8}{1-m}\|x^{(1)} - x^{(0)}\|_\infty$ sledi $\|x^{(8)} - \alpha\|_\infty \leq 2.9 \cdot 10^{-3}$, kar ni najboljša ocena, a ponavadi so tovrstne ocene dokaj pesimistične. □

Pri Jacobijevi iteraciji lahko računamo vse komponente $x_i^{(r+1)}$, $i = 1, \ldots, n$, neodvisno drugo od druge, torej lahko z uporabo paralelnega računalnika mnogo pridobimo. Po drugi strani pa bi lahko komponente računali po vrsti in ko bi računali $x_i^{(r+1)}$, bi lahko na desni strani namesto starih vrednosti $x_1^{(r)}, \ldots, x_{i-1}^{(r)}$ upoštevali že nove $x_1^{(r+1)}, \ldots, x_{i-1}^{(r+1)}$. Če naredimo tako, potem temu postopku pravimo *Seidlova*[2] *iteracija*. Ponavadi konvergira hitreje kot Jacobijeva iteracija, obstajajo pa tudi protiprimeri.

[2]Nemški matematik Philip Ludwig von Seidel (1821–1898) je metodo za linearne sisteme (glej **??**. poglavje) objavil leta 1874.

Zgled 4.4 *Če pri iteracijski funkciji (4.2) za (4.1) računamo komponente $x^{(r+1)}$ po vrsti in izračunane komponente upoštevamo pri izračunu naslednjih, računamo po formulah*

$$\begin{aligned} x_1^{(r+1)} &= \frac{1}{3}\left(\cos(x_1^{(r)}x_2^{(r)}) + 0.6\right) \\ x_2^{(r+1)} &= \frac{1}{9}\sqrt{(x_1^{(r+1)})^2 + \sin(x_3^{(r)}) + 1.1} - 0.1 \\ x_3^{(r+1)} &= -\frac{1}{20}\left(e^{-x_1^{(r+1)}x_2^{(r+1)}} + 9.1\right). \end{aligned}$$

Če vzamemo začetni približek $x^{(0)} = (0.4, 0.1, -0.4)$, dobimo zaporedje

r	$x_1^{(r)}$	$x_2^{(r)}$	$x_3^{(r)}$	$\|\boldsymbol{x}^{(r)} - \boldsymbol{x}^{(r-1)}\|_\infty$
1	0.533066702220	0.010818602012	−0.504712478046	$1.3 \cdot 10^{-1}$
2	0.533327790230	0.005460936737	−0.504854588390	$5.4 \cdot 10^{-3}$
3	0.533331919588	0.005453913740	−0.504854774000	$7.0 \cdot 10^{-6}$
4	0.533331923200	0.005453904456	−0.504854774246	$9.3 \cdot 10^{-9}$
5	0.533331923204	0.005453904443	−0.504854774247	$1.2 \cdot 10^{-11}$

V tem primeru Seidlova metoda konvergira hitreje od Jacobijeve metode. □

4.3 Newtonova metoda

Pri Newtonovi metodi tvorimo zaporedje

$$x^{(r+1)} = x^{(r)} - JF(x^{(r)})^{-1}F(x^{(r)}), \quad r = 0, 1, \ldots.$$

V praksi ne računamo inverza Jacobijeve matrike, temveč rešujemo sistem:

$$\begin{aligned} JF(x^{(r)})\Delta x^{(r)} &= -F(x^{(r)}), \\ x^{(r+1)} &= x^{(r)} + \Delta x^{(r)}, \quad r = 0, 1, \ldots. \end{aligned}$$

Izpeljava Newtonove metode poteka preko razvoja v Taylorjevo vrsto. Denimo, da so vse funkcije f_i dvakrat zvezno odvedljive v okolici rešitve. Tedaj lahko razvijemo:

$$f_i(x + \Delta x) = f_i(x) + \sum_{k=1}^{n} \frac{\partial f_i(x)}{\partial x_k}\Delta x_k + \cdots, \quad i = 1, \ldots, n.$$

Če zanemarimo kvadratne in višje člene in želimo, da bo $F(x + \Delta x) = 0$, dobimo linearni sistem za popravke

$$\begin{bmatrix} \frac{\partial f_1(x)}{\partial x_1} & \cdots & \frac{\partial f_1(x)}{\partial x_n} \\ \vdots & & \vdots \\ \frac{\partial f_n(x)}{\partial x_1} & \cdots & \frac{\partial f_n(x)}{\partial x_n} \end{bmatrix} \begin{bmatrix} \Delta x_1 \\ \vdots \\ \Delta x_n \end{bmatrix} = - \begin{bmatrix} f_1(x) \\ \vdots \\ f_n(x) \end{bmatrix},$$

nato pa popravimo približke v

$$x_k + \Delta x_k, \quad k = 1, \ldots, n.$$

Konvergenca Newtonove metode je v bližini enostavne ničle kvadratična, težava pa je v tem, da moramo za konvergenco ponavadi poznati dovolj dober začetni približek.

Zgled 4.5 *Z Newtonovo metodo in z začetnim približkom $x_0 = 2$, $y_0 = 4$, izračunaj prva dva približka sistema*

$$\begin{aligned} x^2 + y^2 - 10x + y &= 1 \\ x^2 - y^2 - x + 10y &= 25. \end{aligned}$$

Dobimo

$$\begin{aligned} f_1(x,y) &= x^2 + y^2 - 10x + y - 1 \\ f_2(x,y) &= x^2 - y^2 - x + 10y - 25 \end{aligned}$$

in

$$JF(x,y) = \begin{bmatrix} 2x - 10 & 2y + 1 \\ 2x - 1 & -2y + 10 \end{bmatrix}.$$

V prvem koraku rešimo sistem

$$\begin{bmatrix} -6 & 9 \\ 3 & 2 \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} = \begin{bmatrix} -3 \\ -1 \end{bmatrix} \quad \Rightarrow \quad \Delta x = -\frac{1}{13},\ \Delta y = -\frac{15}{39}.$$

Torej je novi približek $x_1 = 1.9231$, $y_1 = 3.6154$. Če postopek nadaljujemo, dobimo $x_2 = 1.9625$, $y_2 = 3.6262$, točen rezultat pa je $x = 1.9623$, $y = 3.6258$. □

Zgled 4.6 *Za testni primer (4.1) velja*

$$JF(x) = \begin{bmatrix} 3 + x_2 \sin(x_1 x_2) & x_1 \sin(x_1 x_2) & 0 \\ 2x_1 & -162(x_2 + 0.1) & \cos(x_3) \\ -x_2 e^{-x_1 x_2} & -x_1 e^{-x_1 x_2} & 20 \end{bmatrix}.$$

Če vzamemo začetni približek $x^{(0)} = (0.4, 0.1, -0.4)$, potem Newtonova metoda vrne

r	$x_1^{(r)}$	$x_2^{(r)}$	$x_3^{(r)}$	$\|x^{(r)} - x^{(r-1)}\|_\infty$
1	0.533277157343	0.027209850424	−0.503797935671	$1.8 \cdot 10^{-1}$
2	0.533349540310	0.007317424279	−0.504802389730	$2.0 \cdot 10^{-2}$
3	0.533332076670	0.005470116245	−0.504854320569	$1.8 \cdot 10^{-3}$
4	0.533331923216	0.005453905692	−0.504854774212	$1.6 \cdot 10^{-5}$
5	0.533331923204	0.005453904443	−0.504854774247	$1.2 \cdot 10^{-9}$
6	0.533331923204	0.005453904443	−0.504854774247	$1.2 \cdot 10^{-16}$

Iz tabele je lepo razvidno, da je konvergenca kvadratična. Sicer res porabimo en korak več pri Seidlovi metodi z istim začetnim približkom, a v bližini rešitve je red konvergence Newtonove metode kvadratičen, pri Seidlovi metodi pa linearen. □

Za konvergenco Newtonove metode imamo na voljo Kantorovičev[3] izrek, ki nam ob izpolnjenih predpostavkah zagotavlja konvergenco.

[3]Ruski matematik in ekonomist Leonid Kantorovič (1912–1986) je izrek objavil leta 1940. Leta 1975 je prejel Nobelovo nagrado za ekonomijo.

Izrek 4.3 (Kantorovič) *Denimo, da obstajajo taka števila a, b, c, da je $h = abc < \frac{1}{2}$ in da velja:*

a) F je v $x^{(0)}$ odvedljiva in $\|JF^{-1}(x^{(0)})\|_\infty \leq a$,

b) $b = \|x^{(1)} - \boldsymbol{x}^{(0)}\|_\infty$,

c) v okolici $K_\infty(x^{(0)}, 2b) = \{x : \|x - x^{(0)}\|_\infty \leq 2b\}$ je funkcija f_i dvakrat zvezno odvedljiva in velja

$$\sum_{k=1}^{n} \left| \frac{\partial^2 f_i(x)}{\partial x_j \partial x_k} \right| \leq \frac{c}{n} \quad za \quad i, j = 1, \ldots, n.$$

Potem ima sistem $F(x) = 0$ v $K_\infty(x^{(0)}, 2b)$ natanko eno rešitev α h kateri konvergira zaporedje $\{x^{(r)}\}$ in velja ocena

$$\|x^{(r)} - \alpha\|_\infty \leq \frac{(2h)^{2^r - 1}}{2^{r-1}}.$$

Ponavadi uporabljamo Kantorovičev izrek za to, da že vnaprej pokažemo, da imamo tako dober začetni približek, da bo Newtonova metoda skonvergirala. Iz izreka je razvidno, da bomo dovolj blizu ničle vedno imeli konvergenco, saj gre tam b proti 0.

Zgled 4.7 *Za testni nelinearni sistem (4.1) velja*

$$\begin{array}{lll}
f_{1x_1x_1}(\boldsymbol{x}) = x_2^2 \cos(x_1x_2) & f_{1x_1x_2}(\boldsymbol{x}) = x_1x_2 \cos(x_1x_2) & f_{1x_1x_3}(\boldsymbol{x}) = 0 \\
f_{1x_2x_2}(\boldsymbol{x}) = x_1^2 \cos(x_1x_2) & f_{1x_2x_3}(\boldsymbol{x}) = 0 & f_{1x_3x_3}(\boldsymbol{x}) = 0 \\
f_{2x_1x_1}(\boldsymbol{x}) = 2 & f_{2x_1x_2}(\boldsymbol{x}) = 0 & f_{2x_1x_3}(\boldsymbol{x}) = 0 \\
f_{2x_2x_2}(\boldsymbol{x}) = -162 & f_{2x_2x_3}(\boldsymbol{x}) = 0 & f_{2x_3x_3}(\boldsymbol{x}) = -\sin x_3 \\
f_{3x_1x_1}(\boldsymbol{x}) = x_2^2 e^{-x_1x_2} & f_{3x_1x_2}(\boldsymbol{x}) = x_1x_2 e^{-x_1x_2} & f_{3x_1x_3}(\boldsymbol{x}) = 0 \\
f_{3x_2x_2}(\boldsymbol{x}) = x_1^2 e^{-x_1x_2} & f_{3x_2x_3}(\boldsymbol{x}) = 0 & f_{3x_3x_3}(\boldsymbol{x}) = 0.
\end{array}$$

Vzamemo lahko $c = 486$. Za a dobimo $a = 0.33$. Če začnemo pri $x^{(0)}$, izreka še ne moremo uporabiti, saj je potem $h = 28.9$. Če pa začnemo pri $x^{(2)}$, dobimo $h = 0.29$. Po izreku potem dobimo oceno $\|\boldsymbol{x}^{(5)} - \boldsymbol{\alpha}\|_\infty \leq 8.6 \cdot 10^{-5}$. □

4.4 Kvazi-Newtonove metode

Posebno pri velikem n imamo pri Newtonovi metodi veliko dela. Če je Jacobijeva matrika polna, potem moramo v vsakem koraku najprej izračunati n^2 parcialnih odvodov, nato pa še rešiti sistem z matriko $n \times n$, za kar potrebujemo $\mathcal{O}(n^3)$ operacij.

Zaradi tega in pa tudi zato, ker ne poznamo nujno parcialnih odvodov, bi tudi tukaj radi žrtvovali hitrejšo konvergence za računanje brez parcialnih odvodov, podobno kot pri sekantni metodi. V ta namen obstaja mnogo kvazi-Newtonovih metod, ki ne uporabljajo parcialnih odvodov. Najbolj znana je Broydenova[4] metoda.

[4] Angleški matematik Charles George Broyden (1933–2011) je metodo objavil leta 1965.

Kadar je n velik in je Jacobijeva matrika razpršena, potem se za reševanje linearnega sistema s to matriko ne splača uporabljati direktnih metod. Uporabljamo iterativne metode, kar pomeni, da v vsakem koraku namesto točnega popravka izračunamo le približek za popravek. Tovrstne metode so t.i. netočne Newtonove metode, primer je kombinacija Newtonove metode in katere izmed iterativnih metod iz **??**. poglavja, npr. metode GMRES.

Naj bo B_r približek za $JF(x^{(r)})$. En korak kvazi–Newtonove metode je

a) reši $B_r \Delta x^{(r)} = -F(x^{(r)})$,

b) $x^{(r+1)} = x^{(r)} + \Delta x^{(r)}$,

c) določi B_{r+1} za naslednji korak.

Pri Broydenovi metodi za B_{r+1} vzamemo najbližjo matriko B_r, ki zadošča t.i. sekantnemu pogoju

$$B_{r+1}(\boldsymbol{x}^{(r+1)} - \boldsymbol{x}^{(r)}) = F(\boldsymbol{x}^{(r+1)}) - F(\boldsymbol{x}^{(r)}).$$

Rešitev je

$$B_{r+1} = B_r + \frac{F(\boldsymbol{x}^{(r)})(\Delta \boldsymbol{x}^{(r)})^T}{(\Delta \boldsymbol{x}^{(r)})^T \Delta \boldsymbol{x}^{(r)}}.$$

Na začetku za B_0 vzamemo čim boljšo aproksimacijo za $JF(x^{(0)})$, v najslabšem primeru pa kar I. Število operacij lahko zmanjšamo, če namesto direktnega posodabljanja B_r posodobimo razcep, ki ga uporabljamo za reševanje sistema v točki a).

Teoretično se kot najcenejša metoda ponuja posodabljanje inverza matrike B_r preko Sherman–Morissonove[5] formule. Če namreč poznamo A^{-1}, potem je

$$(A + uv^T)^{-1} = A^{-1} - \frac{A^{-1}uv^TA^{-1}}{1 + v^TA^{-1}u}$$

zelo učinkovita formula za izračun inverza posodobljene matrike $A + uv^T$. Na žalost se izkaže, da uporaba te formule ni numerično stabilna, zato je bolje posodabljati LU razcep matrike B_r, kar se da tudi narediti na učinkovit način.

4.5 Variacijske metode

Iščemo ekstrem dvakrat zvezno odvedljive funkcije $G\colon \mathbb{R}^n \to \mathbb{R}$. Potreben pogoj za ekstrem je $\mathrm{grad}G(x) = 0$ oziroma $\frac{\partial G(x)}{\partial x_k} = 0$ za $k = 1, \ldots, n$.

Če je x stacionarna točka, potem o vrsti in obstoju ekstrema odloča Hessejeva[6] matrika

$$HG(x) = \begin{bmatrix} \frac{\partial^2 G(x)}{\partial x_1^2} & \cdots & \frac{\partial^2 G(x)}{\partial x_1 \partial x_n} \\ \vdots & & \vdots \\ \frac{\partial^2 G(x)}{\partial x_1 \partial x_n} & \cdots & \frac{\partial^2 G(x)}{\partial x_n^2} \end{bmatrix}.$$

Velja:

[5] Formulo sta Jack Sherman in Winifred J. Morrison objavila leta 1949.

[6] Nemški matematik Otto Hesse (1811–1874).

- $HG(x)$ pozitivno definitna: lokalni minimum,
- $HG(x)$ negativno definitna: lokalni maksimum,
- $HG(x)$ semidefinitna: odločajo višji odvodi,
- $HG(x)$ nedefinitna: ni ekstrema.

Torej lahko iskanje ekstrema funkcije več spremenljivk prevedemo na reševanje sistema nelinearnih enačb. Gre pa tudi obratno.

Denimo, da iščemo rešitev sistema $F(x) = 0$, kjer je $F\colon \mathbb{R}^n \to \mathbb{R}^n$. Funkcija

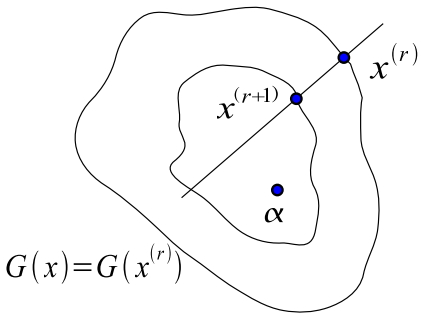
$$G(x) = \sum_{i=1}^{n} f_i^2(x)$$

ima globalni minimum ravno v točkah, kjer je $F(x) = 0$, zato lahko ničlo funkcije F poiščemo tako, da poiščemo globalni minimum funkcije G.

Poglejmo si nekaj metod za iskanje minimuma gladke funkcije n spremenljivk. Splošni nastavek je, da se iterativno monotono približujemo minimumu. Naj bo $x^{(r)}$ tekoči približek. Izberemo vektor (smer) $v_r \in \mathbb{R}^n$ in v tej smeri poiščemo naslednji približek oblike

$$x^{(r+1)} = x^{(r)} + \lambda_r v_r,$$

da je $G(x^{(r+1)}) < G(x^{(r)})$.



Pogledati moramo:

- kako izberemo smer v_r,
- kako določimo premik λ_r.

Za izbiro smeri imamo naslednje možnosti:

a) *splošna metoda spusta*: izberemo poljubno smer v_r, le da ni pravokotna na grad$G(x^{(r)})$, saj v tej smeri ne moremo vedno dobiti manjše vrednosti.

b) *metoda najhitrejšega spusta* oz. *gradientna metoda*: za smer izberemo negativni gradient $v_r = -\text{grad}G(x^{(r)})$. Pri tem pristopu moramo poznati parcialne odvode funkcije G.

c) *metoda koordinatnega spusta*: za smeri po vrsti ciklično izbiramo t.i. koordinatne smeri $e_1, e_2, \ldots, e_n$.

Pri določanju premika λ_r imamo opravka s funkcijo ene spremenljivke

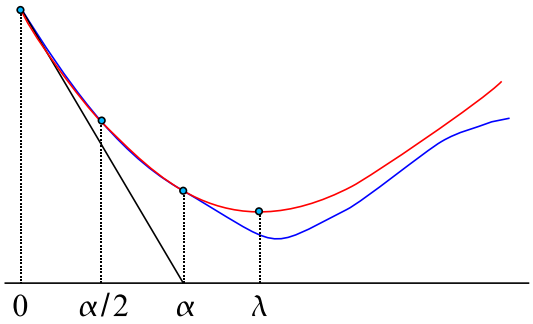
$$g_r(\lambda) := G(x^{(r)} + \lambda v_r).$$

Iščemo tak λ_r, da je $g_r(\lambda_r) < g_r(0)$. Možne metode so:

a) *Metoda največjega spusta*: poiščemo λ_r, kjer funkcija g_r doseže svoj minimum. Za to rešimo nelinearno enačbo $g_r'(\lambda_r) = 0$ ali pa uporabimo kakšno metodo za računanje minimuma funkcije ene spremenljivke, npr. metodo zlatega reza.

b) *Metoda tangentnega spusta*: za λ_r vzamemo presečišče tangente na $y = g_r(\lambda)$ v točki $\lambda = 0$ z osjo x, oziroma $\lambda_r = -g_r(0)/g_r'(0)$. Če je $g(\lambda_r) \geq g(0)$, potem λ_r toliko časa razpolavljamo, dokler ne dobimo manjše vrednosti.

c) *Metoda paraboličnega spusta*: najprej s tangentno metodo določimo α, potem pa skozi točke $(0, g_r(0))$, $(\alpha/2, g(\alpha/2))$, $(\alpha, g(\alpha))$ potegnemo parabolo in za λ_r vzamemo točko, kjer parabola doseže minimum.



d) *Metoda diskretnega spusta*: Izberemo h_r. Če je $g(h_r) < g(0)$, potem se premikamo naprej s korakom h_r in za λ_r vzamemo kh_r, kjer je prvič $g((k+1)h_r) \geq g(kh_r)$. Sicer pa h_r razpolavljamo toliko časa, da je $g(h_r) < g(0)$ in za λ_r vzamemo h_r.

Če iščemo minimum nenegativne funkcije, kar delamo, ko uporabimo variacijsko metodo za reševanje nelinearnega sistema, imamo zagotovljeno konvergenco ne glede na začetni približek, saj dobimo zaporedje približkov $\{x^{(r)}\}$, za katere velja $G(x^{(r+1)}) < G(x^{(r)})$. Kar se tiče reda konvergence, je ta ponavadi linearna.

Tako vedno dobimo nek lokalni minimum, nič pa nam ne zagotavlja, da bomo našli tudi globalni minimum. Ponavadi uporabimo kombinacijo variacijske metode in Newtonove metode ali kvazi–Newtonove metode. Variacijska metoda nas ne glede na kvaliteto začetnega približka pripelje v bližino lokalnega minimuma, potem pa uporabimo hitrejšo metodo, ki potrebuje dobre začetne približke.

Zgled 4.8 *Če rešujemo nelinearni sistem $F(x) = 0$, kjer je $F : \mathbb{R}^n \to \mathbb{R}^n$, potem za*

$$G(x) = f_1(x)^2 + \cdots + f_n(x)^2$$

velja

$$G(x) = F(x)^T F(x).$$

Za gradient velja

$$\text{grad}G(x) = 2 \begin{bmatrix} f_1(x)f_{1x_1}(x) + f_2(x)f_{2x_1}(x) + \cdots + f_n(x)f_{nx_1}(x) \\ f_1(x)f_{1x_2}(x) + f_2(x)f_{2x_2}(x) + \cdots + f_n(x)f_{nx_2}(x) \\ \vdots \\ f_1(x)f_{1x_n}(x) + f_2(x)f_{2x_n}(x) + \cdots + f_n(x)f_{nx_n}(x) \end{bmatrix}$$

oziroma enostavneje

$$\text{grad}G(x) = JF(x)^T F(x).$$

Če definiramo $g(\lambda) = G(x + \lambda z)$, potem dobimo

$$g'(0) = 2F(x)^T JF(x)z.$$

□

Zgled 4.9 *Reševanje nelinearnega sistema (4.1) lahko prevedemo na iskanje minimuma funkcije*

$$G(x) = f_1(x)^2 + f_2(x)^2 + f_3(x)^2,$$

kjer so

$$\begin{aligned} f_1(x) &= 3x_1 - \cos(x_1 x_2) - 0.6 \\ f_2(x) &= x_1^2 - 81(x_2 + 0.1)^2 + \sin(x_3) + 1.1 \\ f_3(x) &= e^{-x_1 x_2} + 20x_3 + 9.1. \end{aligned}$$

Če vzamemo začetni približek $x^{(0)} = (0, 0, 0)$, potem po 20 korakih gradientne metode s paraboličnim spustom dobimo $x^{(20)} = (0.65079172132, -0.21305107190, -0.511385522215)$.

Če to uporabimo za začetni približek za Newtonovo metodo, potem po 5 korakih dobimo rešitev $\alpha = (0.531357185580, -0.205028183876, -0.510754950476)$. Če pa bi namesto Newtonove metode še naprej uporabljali gradientno metodo, bi za enako natančnost potrebovali še 104 korake gradientne metode. □

Matlab

V standardni verziji so na voljo naslednje funkcije:

- `fminsearch`: iskanje minimuma realne funkcije iz $\mathbb{R}^n$ v $\mathbb{R}$. Uporablja simpleksni algoritem.
- `fminbnd`: iskanje minimuma funkcije ene spremenljivke. Uporablja kombinacijo metode zlatega reza in parabolične interpolacije.

Če ga imamo, potem je v dodatnem paketu za optimizacijo na voljo še:

- `fsolve`: reševanje sistema $F(\boldsymbol{x}) = 0$ preko iskanja minimuma $\|F(\boldsymbol{x})\|_2$.

Primeri uporabe:

- `f=inline('x(1)^2+x(2)^2'); fminsearch(f,[0.3;0.2])`
- `f=inline('(x(1)^2+x(2)^2-10*x(1)-1)^2+(x(1)^2-x(2)^2+10*x(2)-25)^2'); fminsearch(f,[2;4])`

Dodatna literatura

Z reševanjem nelinearnih sistemov se ukvarja knjiga [4]. Pri tuji literaturi omenimo npr. učbenike [6], [13] in [17]. Knjiga [16] se ukvarja z reševanjem velikih nelinearnih sistemov s kvazi Newtonovimi metodami.

Poglavje 5

Linearni problemi najmanjših kvadratov

5.1 Uvod

Denimo, da so dane točke $(y_1, b_1), \ldots, (y_m, b_m)$, kjer je $m > 4$, iščemo pa kubični polinom oblike $p(y) = x_1 + x_2 y + x_3 y^2 + x_4 y^3$, ki gre skozi dane točke. Polinom bi moral zadoščati m enačbam

$$\begin{aligned} x_1 + x_2 y_1 + x_3 y_1^2 + x_4 y_1^3 &= b_1 \\ &\vdots \\ x_1 + x_2 y_m + x_3 y_m^2 + x_4 y_m^3 &= b_m \end{aligned}$$

za 4 neznanke. V matrični obliki dobimo sistem $Ax = b$, kjer je

$$A = \begin{bmatrix} 1 & y_1 & y_1^2 & y_1^3 \\ 1 & y_2 & y_2^2 & y_2^3 \\ \vdots & \vdots & \vdots & \vdots \\ 1 & y_m & y_m^2 & y_m^3 \end{bmatrix}, \quad x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}, \quad b = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix}.$$

Ker imamo več enačb kot neznank, je to *predoločen sistem*. Sistem v splošnem nima rešitve, zato nalogo preoblikujemo. Iščemo tak kubični polinom, ki se najbolje prilega danim točkam, torej tak x, da bo imel ostanek $Ax - b$ minimalno normo. Različnim normam ostanka ustrezajo različne rešitve. Nalogo najlažje rešimo, če izberemo evklidsko normo $\|.\|_2$. V tem primeru rešujemo *linearni problem najmanjših kvadratov* in za rešitev x pravimo, da je *rešitev po metodi najmanjših kvadratov*. V konkretnem primeru to pomeni, da iščemo tak kubični polinom p, da bo vsota kvadratov razlik $\sum_{i=1}^{m} (p(y_i) - b_i)^2$ minimalna.

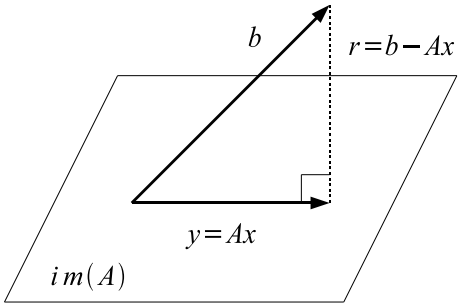
V splošnem linearnem problemu najmanjših kvadratov sta dani matrika $A \in \mathbb{R}^{m \times n}$, kjer je $m > n$, in vektor $b \in \mathbb{R}^m$. Rešitev po metodi najmanjših kvadratov je vektor $x \in \mathbb{R}^n$, ki minimizira $\|Ax - b\|_2$. Predpostavimo še, da je A polnega ranga, torej $\text{rang}(A) = n$, sicer rešitev ni enolična. Če A ni polnega ranga, potem namreč obstaja neničelni vektor $z \in \ker(A)$ in iz $Ax - b = A(x + z) - b$ sledi, da rešitev ne more biti enolična.

5.2 Normalni sistem

Imamo matriko polnega ranga $A \in \mathbb{R}^{m\times n}$, kjer je $m \geq n$, in vektor $b \in \mathbb{R}^m$, iščemo pa vektor $x \in \mathbb{R}^n$, ki minimizira $\|Ax - b\|_2$. Ker vsi vektorji oblike Ax ležijo v linearnem podprostoru $\mathrm{im}(A)$, v bistvu iščemo vektor Ax iz podprostora $\mathrm{im}(A)$, ki je v normi $\|.\|_2$ najboljša aproksimacija vektorja b. Rešitev je pravokotna projekcija vektorja b na $\mathrm{im}(A)$ in zato mora biti ostanek $b - Ax$ pravokoten na podprostor $\mathrm{im}(A)$. Ker stolpci matrike A tvorijo bazo za $\mathrm{im}(A)$, od tod sledi, da mora veljati $A^T(b - Ax) = 0$ oziroma

$$A^T Ax = A^T b. \tag{5.1}$$

Dobili smo t.i. *normalni sistem*, ki je nesingularen, saj je matrika A polnega ranga. Situacija je prikazana na sliki 5.1.



Slika 5.1: Najboljša aproksimacija v evklidski normi je pravokotna projekcija.

Z zgornjo geometrijsko izpeljavo smo dokazali naslednji izrek, ki ga bomo dokazali še na drug način.

Izrek 5.1 *Naj bo $A \in \mathbb{R}^{m\times n}$, $m \geq n$, $\mathrm{rang}(A) = n$ in $b \in \mathbb{R}^m$. Rešitev normalnega sistema (5.1) je rešitev po metodi najmanjših kvadratov.*

Dokaz. Naj bo $B = A^T A$ in $c = A^T b$. Matrika B je simetrična in pozitivno definitna, saj je A polnega ranga. Velja

$$\|Ax - b\|_2^2 = (Bx - c)^T B^{-1}(Bx - c) - c^T B^{-1} c + b^T b,$$

to pa je, ker je B^{-1} simetrična pozitivno definitna matrika, minimalno natanko takrat, ko je $Bx = c$. ∎

Matrika $A^T A$ je simetrična pozitivno definitna, zato za reševanje normalnega sistema uporabimo razcep Choleskega. Število operacij za izračun $A^T A$, razcep Choleskega in reševanje sistema je $n^2 m + \frac{1}{3}n^3 + \mathcal{O}(n^2)$, ker pa je ponavadi $m \gg n$, je najpomembnejši člen $n^2 m$.

Normalni sistem je najpreprostejši način reševanja predoločenega sistema, ni pa najstabilnejši. Težave se pojavijo, kadar stolpci matrike A niso dovolj linearno neodvisni. Takrat je bolje namesto stolpcev matrike uporabiti kakšno boljšo bazo za $\mathrm{im}(A)$, po možnosti ortonormirano.

Zgled 5.1 *Denimo, da iščemo polinom $p(x) = a_0 + a_1 x + \cdots + a_n x^n$ stopnje n, ki se najbolje prilega točkam (x_i, y_i), $i = 1, \ldots, m$. Matrika $B = A^T A$ ima elemente $b_{ij} = \sum_{k=1}^m x_k^{i+j-2}$. Če so točke x_i*

enakomerno porazdeljene po intervalu $(0,1)$, *torej* $x_i = i/(m+1)$, *velja*

$$b_{ij} = \sum_{k=1}^{m} \left(\frac{k}{m+1}\right)^{i+j-2} \approx (m+1) \int_0^1 x^{i+j-2} dx = \frac{m+1}{i+j-1},$$

to pa pomeni, da je $B \approx (m+1)H_{n+1}$. *Ker pa so Hilbertove matrike zgled za zelo občutljive matrike, računanje aproksimacijskega polinoma visoke stopnje preko normalnega sistema ni stabilno.* □

5.3 QR razcep

Naj bo $A \in \mathbb{R}^{m\times n}$, $m \geq n$ in $\text{rang}(A) = n$. Denimo, da poznamo razcep $A = QR$, kjer je Q pravokotna matrika velikosti $m \times n$ z ortonormiranimi stolpci in R zgornja trikotna matrika velikosti $n \times n$. Tak razcep imenujemo *QR razcep*. Potem iz normalnega sistema dobimo

$$x = (A^TA)^{-1}A^Tb = (R^TQ^TQR)^{-1}R^TQ^Tb = R^{-1}R^{-T}R^TQ^Tb = R^{-1}Q^Tb.$$

Rešitev po metodi najmanjših kvadratov torej dobimo, če rešimo zgornji trikotni sistem

$$Rx = Q^Tb.$$

Do iste formule pridemo tudi z geometrijskim sklepanjem. Stolpci matrike Q prav tako kot stolpci matrike A razpenjajo podprostor $\text{im}(A)$, a je zaradi ortonormiranosti baza iz stolpcev matrike Q manj občutljiva. Pogoj, da mora biti pri rešitvi po metodi najmanjših kvadratov ostanek $b - Ax$ pravokoten na podprostor $\text{im}(A)$ lahko zapišemo kot $Q^T(Ax - b) = 0$ in tako spet pridemo do enačbe $Rx = Q^Tb$. Reševanje preko QR razcepa je zato stabilnejše od reševanja normalnega sistema.

Izrek 5.2 *Naj bo* $A \in \mathbb{R}^{m\times n}$, $m \geq n$ *in* $\text{rang}(A) = n$. *Potem obstaja enolični QR razcep* $A = QR$, *kjer je* Q *pravokotna matrika velikosti* $m \times n$ *z ortonormiranimi stolpci,* R *pa zgornja trikotna matrika velikosti* $n \times n$ *s pozitivnimi diagonalnimi elementi.*

Dokaz. Za dokaz obstoja bomo QR razcep kar skonstruirali. Denimo, da je $A = [a_1 \ \cdots \ a_n]$ in $Q = [q_1 \ \cdots \ q_n]$. Potem iz $A = QR$ sledi

$$a_k = \sum_{i=1}^{k} r_{ik}q_i.$$

Vektorji $q_1, \ldots, q_k$ so ortonormirani in razpenjajo isti podprostor kot vektorji $a_1, \ldots, a_k$ za $k = 1, \ldots, n$. To pomeni, da lahko Q in R dobimo z Gram-Schmidtovo[1] ortogonalizacijo stolpcev matrike A. Postopek je zapisan v algoritmu 5.1.

CGS je *klasična Gram–Schmidtova metoda*, MGS pa *modificirana Gram–Schmidtova metoda*. Pri eksaktnem računanju vrneta CGS in MGS identične vrednosti, numerično pa je MGS stabilnejši od CGS.

[1]Danski matematik Jørgen Pedersen Gram (1850–1916) je postopek predstavil leta 1883, nemški matematik Erhard Schmidt (1876–1959) pa leta 1907, pri čemer je Gram razvil varianto MGS, Schmidt pa CGS. Oba je prehitel znani francoski matematik in astronom Pierre-Simon Laplace (1749–1827), ki je metodo MGS objavil že leta 1816.

Algoritem 5.1 Gram–Schmidtova ortogonalizacija. Začetni podatek je matrika A z n stolci. Algoritem vrne matriko Q z ortonormiranimi stolpci in zgornjo trikotno matriko R, da je $A = QR$.

$k = 1, \ldots, n$
  $q_k = a_k$
  $i = 1, \ldots, k-1$
    $r_{ik} = q_i^T a_k$ (CGS) ali $r_{ik} = q_i^T q_k$ (MGS)
    $q_k = q_k - r_{ik} q_i$
  $r_{kk} = \|q_k\|_2$
  $q_k = \frac{q_k}{r_{kk}}$

Pokažimo še enoličnost. Naj bo $A = QR$. Potem velja

$$A^T A = R^T R,$$

kar je razcep Choleskega za simetrično pozitivno definitno matriko A. Ker je razcep Choleskega enoličen, je matrika R enolična, prav tako pa potem tudi matrika $Q = AR^{-1}$. ■

Zgled 5.2 *Če vzamemo $\epsilon = 10^{-10}$ in preko CGS in MGS v Matlabu ortogonaliziramo vektorje*

$$x_1 = \begin{bmatrix} 1+\epsilon \\ 1 \\ 1 \end{bmatrix}, \quad x_2 = \begin{bmatrix} 1 \\ 1+\epsilon \\ 1 \end{bmatrix}, \quad x_3 = \begin{bmatrix} 1 \\ 1 \\ 1+\epsilon \end{bmatrix},$$

dobimo pri CGS $q_2^T q_3 \approx 0.5$, kar je narobe, pri MGS pa $q_2^T q_3 = -1.1 \cdot 10^{-16}$. □

Število operacij za QR razcep preko Gram–Schmidtove ortogonalizacije je

$$\sum_{k=1}^{n} \left(3m + \sum_{i=1}^{k-1} 4m \right) \approx 4m \sum_{k=1}^{n} k \approx 2mn^2,$$

kar je približno dvakrat toliko operacij kot pri normalnem sistemu (za $m \gg n$).

Pri reševanju predoločenega sistema z MGS moramo paziti na zadnji korak. Nepravilno je uporabiti MGS za izračun matrik Q in R, potem pa reševati sistem $Rx = Q^T b$, saj lahko pri računanju $Q^T b$ izgubimo vso natančnost, ki smo jo pridobili, ko smo namesto CGS izvajali MGS.

Pravilno je, da najprej z MGS izračunamo QR razcep za z vektorjem b razširjeno matriko A:

$$[\, A \quad b \,] = [\, Q \quad q_{n+1} \,] \begin{bmatrix} R & z \\ & \rho \end{bmatrix}.$$

Sedaj dobimo

$$Ax - b = [\, A \quad b \,] \begin{bmatrix} x \\ -1 \end{bmatrix} = [\, Q \quad q_{n+1} \,] \begin{bmatrix} R & z \\ & \rho \end{bmatrix} \begin{bmatrix} x \\ -1 \end{bmatrix} = Q(Rx - z) - \rho q_{n+1}.$$

Ker je $q_{n+1} \perp Q$, bo minimum dosežen pri $Rx = z$.

Poleg QR razcepa poznamo še *razširjeni QR razcep* $A = \widetilde{Q}\widetilde{R}$, kjer je $\widetilde{Q}$ ortogonalna matrika $m \times m$, $\widetilde{R}$ pa zgornja trapezna matrika $m \times n$. Prvih n stolpcev matrike $\widetilde{Q}$ in zgornji kvadrat matrike $\widetilde{R}$ tvori QR razcep matrike A. Tudi razširjeni QR razcep bomo ponavadi imenovali kar QR razcep, saj je vse razvidno iz dimenzij matrik.

Naj bo $A = \widetilde{Q}\widetilde{R}$, kjer je $\widetilde{Q} = [Q\ Q_1]$ in $\widetilde{R} = \begin{bmatrix} R \\ 0 \end{bmatrix}$. Potem velja

$$\|Ax - b\|_2 = \|\widetilde{Q}^T(Ax - b)\|_2 = \left\| \begin{bmatrix} R \\ 0 \end{bmatrix} x - \begin{bmatrix} Q^T b \\ Q_1^T b \end{bmatrix} \right\|_2 = \left\| \begin{bmatrix} Rx - Q^T b \\ -Q_1^T b \end{bmatrix} \right\|_2.$$

Zgornji del lahko uničimo, spodnjega pa ne. Velja torej

$$\min_{x \in \mathbb{R}^n} \|Ax - b\|_2 = \|Q_1^T b\|_2,$$

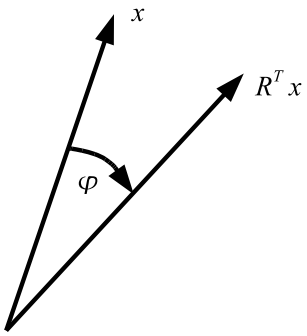
minimum pa je dosežen pri $Rx = Q^T b$.

5.4 Givensove rotacije

V ravnini vektor $x = [x_1\ x_2]^T$ zarotiramo za kot φ v negativni smeri tako, da ga pomnožimo z ortogonalno matriko

$$R^T = \begin{bmatrix} c & s \\ -s & c \end{bmatrix},$$

kjer je $c = \cos\varphi$ in $s = \sin\varphi$. Rotacija je prikazana na sliki 5.2.



Slika 5.2: Givensova rotacija.

Če to posplošimo na rotacijo v ravnini (i,k) v $\mathbb{R}^n$, dobimo ortogonalno matriko R_{ik}^T, ki je enaka identiteti povsod razen v i-ti in k-ti vrstici, kjer je

$$R_{ik}^T([i,k],[i,k]) = \begin{bmatrix} c & s \\ -s & c \end{bmatrix}.$$

Za vektor $y = R_{ik}^T x$ velja

$$\begin{aligned} y_j &= x_j, \quad j \neq i,k \\ y_i &= cx_i + sx_k \\ y_k &= -sx_i + cx_k. \end{aligned}$$

Sedaj c in s lahko izberemo tako, da bo $y_k = 0$. Rešitev je

$$\begin{aligned} r &= \sqrt{x_i^2 + x_k^2} \\ c &= \frac{x_i}{r} \\ s &= \frac{x_k}{r}. \end{aligned}$$

Matriko R_{ik} imenujemo *Givensova rotacija*.[2] Pri množenju matrike z R_{ik}^T se spremenita le i-ta in k-ta vrstica. Na podoben način kot pri LU razcepu uporabimo elementarne eliminacije, lahko elemente v matriki uničujemo tudi z Givensovimi rotacijami. Ker pa množimo z ortogonalnimi matrikami, je to bolj stabilno od uporabe elementarnih eliminacij.

Z ustreznimi rotacijami, ki jih uporabimo v pravilnem vrstnem redu, lahko v matriki uničimo vse elemente pod diagonalo in tako izračunamo QR razcep.

Pri matriki velikosti 4×3 tako dobimo

$$A = \begin{bmatrix} \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \end{bmatrix} \xrightarrow{R_{12}^T \cdot} \begin{bmatrix} \times & \times & \times \\ 0 & \times & \times \\ \times & \times & \times \\ \times & \times & \times \end{bmatrix} \xrightarrow{R_{13}^T \cdot} \begin{bmatrix} \times & \times & \times \\ 0 & \times & \times \\ 0 & \times & \times \\ \times & \times & \times \end{bmatrix} \xrightarrow{R_{14}^T \cdot} \begin{bmatrix} \times & \times & \times \\ 0 & \times & \times \\ 0 & \times & \times \\ 0 & \times & \times \end{bmatrix}$$

$$\xrightarrow{R_{23}^T \cdot} \begin{bmatrix} \times & \times & \times \\ 0 & \times & \times \\ 0 & 0 & \times \\ 0 & \times & \times \end{bmatrix} \xrightarrow{R_{24}^T \cdot} \begin{bmatrix} \times & \times & \times \\ 0 & \times & \times \\ 0 & 0 & \times \\ 0 & 0 & \times \end{bmatrix} \xrightarrow{R_{34}^T \cdot} \begin{bmatrix} \times & \times & \times \\ 0 & \times & \times \\ 0 & 0 & \times \\ 0 & 0 & 0 \end{bmatrix} = \widetilde{R}.$$

Matrika $\widetilde{R}$ je zgornja trapezna, v zgornjem kvadratu pa vsebuje matriko R. Produkt $\widetilde{Q} = R_{12}R_{13}R_{14}R_{23}R_{24}R_{34}$ je ortogonalna matrika, ki v prvih n stolpcih vsebuje matriko Q. Tako dobimo razcep $A = QR$.

Pri množenju z R_{jk}^T se j-ta in k-ta vrstica spremenita v linearni kombinaciji j-te in k-te vrstice. Če je v nekem stolpcu v obeh vrsticah 0, se to ne more pokvariti z množenjem z R_{jk}^T.

Število operacij za reševanje predoločenega sistema $Ax = b$ preko Givensovih rotacij je

$$\sum_{j=1}^{n} \left(\sum_{k=j+1}^{m} (6 + 6(n - j + 1) + 6) \right) \approx 6 \sum_{j=1}^{n} (m - j)(n - j) \approx 3mn^2 - n^3.$$

Če potrebujemo matriko Q, porabimo še dodatnih $6m^2n - 3mn^2$ operacij. Linearni sistem velikosti $n \times n$ tako s pomočjo Givensovih rotacij rešimo z uporabo $2n^3$ operacij, za matriko Q pa potrebujemo še dodatnih $3n^3$ operacij.

[2] Imenujejo se po ameriškem matematiku Wallaceu Givensu (1910–1993), ki jih je vpeljal leta 1958 za numerično računanje QR razcepa. Same rotacije so bile znane že prej, npr. že dobrih sto let pred Givensom jih je Jacobi uporabljal v svoji metodi za računanje lastnih vrednosti, kjer jih zato zdaj imenujemo Jacobijeve rotacije.

Algoritem 5.2 Ortogonalni razcep z Givensovimi rotacijami. Dana je $m \times n$ matrika A, kjer je $m \geq n$. Algoritem matriko A prepiše z matriko $\widetilde{R}$ iz razširjenega QR razcepa $A = \widetilde{Q}\widetilde{R}$. Po potrebi vrne tudi matriko $\widetilde{Q}$ oziroma produkt $\widetilde{Q}^T b$.

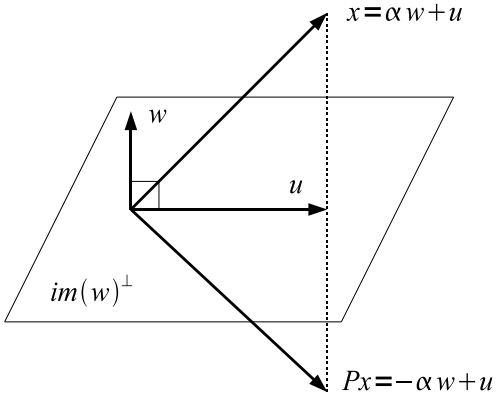
$Q = I_m$
$j = 1, \ldots, n$
  $k = j+1, \ldots, m$
    $r = \sqrt{a_{jj}^2 + a_{kj}^2}$
    $c = a_{jj}/r$
    $s = a_{kj}/r$
    $A([j\ k], j:n) = \begin{bmatrix} c & s \\ -s & c \end{bmatrix} A([j\ k], j:n)$
    $b([j\ k]) = \begin{bmatrix} c & s \\ -s & c \end{bmatrix} b([j\ k])$ (če rešujemo predoločeni sistem $Ax = b$)
    $Q([j\ k], 1:m) = \begin{bmatrix} c & s \\ -s & c \end{bmatrix} Q([j\ k], 1:m)$ (če potrebujemo matriko Q)
$Q = Q^T$.

5.5 Householderjeva zrcaljenja

Za neničelen vektor $w \in \mathbb{R}^n$ definiramo matriko

$$P = I - \frac{2}{w^T w} w w^T.$$

Hitro lahko preverimo, da velja $P = P^T$ in $P^2 = I$, kar pomeni, da je matrika P simetrična in ortogonalna.v Poljuben vektor $x \in \mathbb{R}^n$ lahko zapišemo kot $x = \alpha w + u$, kjer je $u \perp w$. Dobimo $Px = -\alpha w + u$, kar pomeni, da P predstavlja zrcaljenje preko hiperravnine, ki je ortogonalna na vektor w. Situacija je predstavljena na sliki 5.3. Matriko P imenujemo *Householderjevo zrcaljenje*.[3]



Slika 5.3: Householderjevo zrcaljenje.

Množenje z matriko P izvedemo tako, da izračunamo $Px = x - \frac{1}{m}(x^T w)w$, kjer je $m = \frac{1}{2}w^T w$. Skalar m moramo tako izračunati le enkrat, ne glede na to, koliko različnih vektorjev x bomo

[3]Zrcaljenja se imenujejo po ameriškem matematiku Alstonu S. Householderju (1904–1993), ki jih je leta 1958 uporabil za numerično računanje QR razcepa, sama zrcaljenja pa so se v literaturi pojavila že prej.

množili z matriko P. Vidimo, da matrike P ni potrebno eksplicitno izračunati, saj zadošča, da poznamo le vektor w.

Če imamo taka neničelna vektorja x in y, da je $\|x\|_2 = \|y\|_2$, potem se lahko hitro prepričamo, da velja $Px = y$, če izberemo $w = x - y$. Naj bo sedaj x neničelen vektor, ki bi ga radi prezrcalili tako, da bo vektor Px imel vse komponente razen prve enake 0. Ker to pomeni $Px = \pm\|x\|_2 e_1$, mora veljati $w = x \mp \|x\|_2 e_1$, vprašanje je le, kateri predznak izbrati. Za m dobimo

$$m = \frac{1}{2}w^T w = \frac{1}{2}(\|x\|_2^2 \mp 2\|x\|_2 x_1 + \|x\|_2^2) = \|x\|_2(\|x\|_2 \mp x_1).$$

Zaradi numerične stabilnosti se želimo pri računanju m izogniti odštevanju približno enako velikih števil, zato izberemo $m = \|x\|_2(\|x\|_2 + |x_1|)$. Izbira je torej

$$w = \begin{bmatrix} x_1 + \operatorname{sign}(x_1)\|x\|_2 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}.$$

Paziti moramo, da je funkcija predznaka sign definirana tako, da je

$$\operatorname{sign}(x) = \begin{cases} 1 & \text{za} \quad x \geq 0 \\ -1 & \text{za} \quad x < 0. \end{cases}$$

Številni programi, vključno z Matlabom, imajo namreč funkcijo sign že vgrajeno, a zanjo velja $\operatorname{sign}(0) = 0$.

Število operacij za izračun produkta $Pz = z - \frac{1}{m}(z^T w)w$ je $4n + \mathcal{O}(1)$, pri čemer m izračunamo vnaprej. Za izračun w in m pa potrebujemo $2n + \mathcal{O}(1)$ operacij.

Z uporabo ustreznih Houseolderjevih zrcaljenj lahko matriko pretvorimo v zgornjo trapezno obliko in izračunamo QR razcep. Pri matriki 4×3 tako dobimo

$$A = \begin{bmatrix} \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \end{bmatrix} \xrightarrow{\widetilde{P}_1 \cdot} \begin{bmatrix} \times & \times & \times \\ 0 & \times & \times \\ 0 & \times & \times \\ 0 & \times & \times \end{bmatrix} \xrightarrow{\widetilde{P}_2 \cdot} \begin{bmatrix} \times & \times & \times \\ 0 & \times & \times \\ 0 & 0 & \times \\ 0 & 0 & \times \end{bmatrix} \xrightarrow{\widetilde{P}_3 \cdot} \begin{bmatrix} \times & \times & \times \\ 0 & \times & \times \\ 0 & 0 & \times \\ 0 & 0 & 0 \end{bmatrix},$$

pri čemer je

$$\widetilde{P}_{i+1} = \begin{array}{c} \\ i \\ m-i \end{array} \begin{array}{c} \begin{array}{cc} i & m-i \end{array} \\ \begin{bmatrix} I_i & 0 \\ 0 & P_{i+1} \end{bmatrix} \end{array}$$

za $i = 1, 2$. Na koncu iz $Q^T = \widetilde{P}_3\widetilde{P}_2\widetilde{P}_1$ dobimo $Q = \widetilde{P}_1\widetilde{P}_2\widetilde{P}_3$. Namesto računanja matrike Q raje shranimo vektorje w_i.

Število operacij za reševanje predoločenega sistema $Ax = b$ preko Householderjevih zrcaljenj je

$$\sum_{i=1}^{n} [2(m-i+1) + 4(n-i+1)(m-i+1) + 4(m-i+1)] \approx 4\sum_{i=1}^{n}(m-i)(n-i)$$

$$\approx 2mn^2 - \frac{2}{3}n^3.$$

Algoritem 5.3 Ortogonalni razcep s Householderjevimi zrcaljenji. Dana je $m \times n$ matrika A, kjer je $m \geq n$. Algoritem matriko A prepiše z matriko $\widetilde{R}$ iz razširjenega QR razcepa $A = \widetilde{Q}\widetilde{R}$. Po potrebi vrne tudi matriko $\widetilde{Q}$ oziroma produkt $\widetilde{Q}^T b$.

$Q = I_m$
$i = 1, \ldots, n$
  določi $w_i \in \mathbb{R}^{m-i+1}$, ki prezrcali $A(i:m, i)$ v $\pm k e_1$.
  $A(i:m, i:n) = P_i \cdot A(i:m, i:n)$
  $b(i:m) = P_i \cdot b(i:m)$ (če rešujemo predoločeni sistem $Ax = b$)
  $Q(i:m, 1:n) = P_i \cdot Q(i:m, 1:n)$ (če potrebujemo matriko Q)
$Q = Q^T$.

Za matriko Q potrebujemo še dodatnih $4m^2n - 2mn^2$ operacij.

Za polno matriko so Householderjeva zrcaljenja učinkovitejša od Givensovih rotacij. Zadnje uporabljamo kadar imajo matrike malo neničelnih elementov pod glavno diagonalo, saj v takem primeru z rotacijami porabimo manj operacij kot z zrcaljenji. Prednost rotacij je tudi ta, da jih lahko na paralelnem računalniku izvajamo več sočasno, kar pri zrcaljenjih ni možno.

Seveda lahko QR razcep uporabimo tudi za reševanje kvadratnih nesingularnih linearnih sistemov. Uporaba QR razcepa je stabilnejša od uporabe LU razcepa z delnim ali kompletnim pivotiranjem, a porabimo več operacij. Tako linearni sistem velikosti $n \times n$ s pomočjo Householderjevih zrcaljenj rešimo z uporabo $\frac{4}{3}n^3$ operacij.

5.6 Singularni razcep

Izrek 5.3 *Za vsako matriko $A \in \mathbb{R}^{m\times n}$, $m \geq n$, obstaja* singularni razcep

$$A = U\Sigma V^T,$$

kjer sta $U \in \mathbb{R}^{m\times m}$ in $V \in \mathbb{R}^{n\times n}$ ortogonalni matriki in je $\Sigma \in \mathbb{R}^{m\times n}$ oblike

$$\Sigma = \begin{bmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_n \\ & & \end{bmatrix},$$

kjer so $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n \geq 0$ singularne vrednosti matrike A.

Stolpci matrike $U = [u_1 \ \cdots \ u_m]$ so *levi*, stolpci matrike $V = [v_1 \ \cdots \ v_n]$ pa *desni singularni vektorji*.

Dokaz. Ker je $A^T A$ simetrična pozitivno semidefinitna matrika, so vse njene lastne vrednosti nenegativne:

$$\sigma_1^2 \geq \sigma_2^2 \geq \cdots \geq \sigma_n^2 \geq 0.$$

Ustrezni ortonormirani lastni vektorji naj bodo $A^T A v_i = \sigma_i^2 v_i$, $i = 1, \ldots, n$.

Naj bo $\sigma_r > 0$ in $\sigma_{r+1} = \cdots = \sigma_n = 0$. Označimo $V_1 := [v_1 \ \cdots \ v_r]$ in $V_2 := [v_{r+1} \ \cdots \ v_n]$. Iz

$$(AV_2)^T(AV_2) = V_2^T A^T A V_2 = V_2^T[0 \ \cdots \ 0] = 0$$

sledi $AV_2 = 0$.

Sedaj definiramo $u_i = \frac{1}{\sigma_i} A v_i$ za $i = 1, \ldots, r$. Vektorji $u_1, \ldots, u_r$ so ortonormirani, saj je

$$u_i^T u_j = \frac{1}{\sigma_i \sigma_j} v_i^T A^T A v_j = \frac{\sigma_j}{\sigma_i} v_i^T v_j = \delta_{ij}, \quad i, j = 1, \ldots, r.$$

Označimo $U_1 := [u_1 \ \cdots \ u_r]$ in izberemo $U_2 := [u_{r+1} \ \cdots \ u_n]$ tako, da je $U = [U_1 \ U_2]$ ortogonalna matrika. Matrika $U^T AV$ ima obliko

$$U^T AV = \begin{array}{c} \\ r \\ m-r \end{array} \begin{array}{c} \begin{array}{cc} r & n-r \end{array} \\ \begin{bmatrix} U_1^T A V_1 & U_1^T A V_2 \\ U_2^T A V_1 & U_2^T A V_2 \end{bmatrix} \end{array}.$$

Desna dva bloka sta zaradi $AV_2 = 0$ enaka 0. Za $i = 1, \ldots, r$ in $k = 1, \ldots, m$ velja

$$u_k^T A v_i = \sigma_i u_k^T u_i = \sigma_i \delta_{ik},$$

torej $U_2^T AV_1 = 0$ in $U_1^T AV_1 = \operatorname{diag}(\sigma_1, \ldots, \sigma_r)$. Tako dobimo singularni razcep $A = U\Sigma V^T$, kjer je $S = \operatorname{diag}(\sigma_1, \ldots, \sigma_r)$ in

$$\Sigma = \begin{array}{c} \\ r \\ m-r \end{array} \begin{array}{c} \begin{array}{cc} r & n-r \end{array} \\ \begin{bmatrix} S & 0 \\ 0 & 0 \end{bmatrix} \end{array}.$$

■

V primeru $n < m$ dobimo singularni razcep matrike A tako, da transponiramo singularni razcep matrike A^T.

Matrika A predstavlja preslikavo iz $\mathbb{R}^n$ v $\mathbb{R}^m$. Geometrijski pomen singularnega razcepa je, da se A z ortogonalnima transformacijama baz U v $\mathbb{R}^m$ in V v $\mathbb{R}^n$ spremeni v diagonalno matriko, saj potem velja

$$Av_i = \sigma_i u_i, \quad i = 1, \ldots, n.$$

Poleg tega ima singularni razcep še naslednje lastnosti:

- število neničelnih singularnih vrednosti r je enako rangu matrike A,
- stolpci matrike U_1 tvorijo bazo za podprostor $\operatorname{im} A$,
- stolpci matrike V_2 tvorijo bazo za podprostor $\ker A$,
- stolpci matrike U_2 tvorijo bazo za podprostor $\ker A^T$,
- stolpci matrike V_1 tvorijo bazo za podprostor $\operatorname{im} A^T$.

Še nekaj lastnosti singularnega razcepa:

1. Če je $A = A^T$, potem se da A diagonalizirati kot $A = UDU^T$, $U^T U = I$. Singularni razcep matrike A je potem $A = U\Sigma V^T$ za $\sigma_i = |\lambda_i|$ in $v_i = \operatorname{sign}(\lambda_i) u_i$ (tu je $\operatorname{sign}(0) = 1$).

2. Lastne vrednosti matrike A^TA so $\sigma_1^2, \ldots, \sigma_n^2$. Ortonormirani lastni vektorji A^TA so desni singularni vektorji $v_1, \ldots, v_n$.
3. Lastne vrednosti matrike AA^T so $\sigma_1^2, \ldots, \sigma_n^2, \underbrace{0, \ldots, 0}_{m-n}$. Ortonormirani lastni vektorji AA^T so levi singularni vektorji $u_1, \ldots, u_m$.

Poleg omenjenega poznamo tudi singularni razcep oblike $A = \widetilde{U}\widetilde{\Sigma}V^T$, kjer je $\widetilde{U}$ matrika $m \times n$ z ortonormiranimi stolpci, $\widetilde{\Sigma} = \operatorname{diag}(\sigma_1, \ldots, \sigma_n)$, V pa je $n \times n$ ortogonalna matrika. $\widetilde{U}$ se ujema s prvimi n stolpci matrike U, $\widetilde{\Sigma}$ pa je zgornji kvadrat matrike Σ iz singularnega razcepa $A = U\Sigma V^T$.

Naslednja lema pove, kako si lahko s singularnim razcepom pomagamo pri reševanju predoločenih sistemov.

Lema 5.4 *Če je $A \in \mathbb{R}^{m\times n}$, $m \geq n$, $\operatorname{rang}(A) = n$, potem je minimum $\|Ax - b\|_2$ dosežen pri*

$$x = \sum_{i=1}^{n} \frac{u_i^T b}{\sigma_i} v_i.$$

Dokaz. Naj bo $A = U\Sigma V^T$ in

$$U = \begin{matrix} n & m-n \\ [U_1 & U_2\ ] \end{matrix}, \quad \Sigma = \begin{matrix} n \\ m-n \end{matrix} \begin{bmatrix} S \\ 0 \end{bmatrix}.$$

$$\|Ax - b\|_2 = \|U\Sigma V^T x - b\|_2 = \|\Sigma V^T x - U^T b\|_2 = \left\| \begin{bmatrix} SV^T x - U_1^T b \\ -U_2^T b \end{bmatrix} \right\|_2.$$

Minimum je dosežen pri $SV^T x = U_1^T b$ oziroma

$$x = VS^{-1}U_1^T b = \sum_{i=1}^{n} \frac{u_i^T b}{\sigma_i} v_i. \qquad \blacksquare$$

Definicija 5.5 *Za matriko $A \in \mathbb{R}^{m\times n}$, $m \geq n$, $\operatorname{rang}(A) = n$, definiramo* (Moore–Penroseov[4]) psevdoinverz $A^+ \in \mathbb{R}^{n\times m}$ *kot*

$$A^+ = (A^TA)^{-1}A^T.$$

V primeru $m < n$ in $\operatorname{rang}(A) = m$ definiramo $A^+ = A^T(AA^T)^{-1}$.

Rešitev predoločenega sistema polnega ranga $Ax = b$ lahko sedaj zapišemo kot $x = A^+b$. Psevdoinverz je definiran tudi za matrike, ki niso polnega ranga. Splošna definicija je naslednja.

Definicija 5.6 *Matrika $X \in \mathbb{R}^{n\times m}$ je psevdoinverz matrike $A \in \mathbb{R}^{m\times n}$, če izpolnjuje Moore–Penroseove pogoje:*

1) $AXA = A$,

[4]Neodvisno sta ga definirala ameriški matematik Eliakim Hastings Moore (1862–1932) leta 1920 in angleški matematični fizik Roger Penrose (r. 1931) leta 1955.

2) $XAX = X$,

3) $(AX)^T = AX$,

4) $(XA)^T = XA$.

Zgornja definicija ni preveč uporabna, saj nam ne pove, kako lahko pridemo do psevdoinverza dane matrike. To težavo odpravi naslednji izrek, ki nam pove, da psevdoinverz lahko izračunamo preko singularnega razcepa.

Izrek 5.7 *Naj bo* $A \in \mathbb{R}^{m\times n}$, $m \geq n$, $\operatorname{rang}(A) = r$ *in* $A = U\Sigma V^T$, *kjer je*

$$U = \begin{matrix} & r & m-r \\ & [U_1 & U_2\ ] \end{matrix}, \quad V = \begin{matrix} & r & n-r \\ & [V_1 & V_2\ ] \end{matrix}, \quad \Sigma = \begin{matrix} & r & n-r \\ r & \begin{bmatrix} S & 0 \\ 0 & 0 \end{bmatrix} \\ m-r & \end{matrix}$$

in $S = \operatorname{diag}(\sigma_1, \ldots, \sigma_r)$. *Psevdoinverz matrike* A *je enoličen in enak*

$$A^+ = V\Sigma^+ U^T,$$

kjer je

$$\Sigma^+ = \begin{matrix} & r & n-r \\ r & \begin{bmatrix} S^{-1} & 0 \\ 0 & 0 \end{bmatrix} \\ m-r & \end{matrix}.$$

Dokaz. Psevdoinverz matrike A ima dimenzijo $n \times m$, zato lahko pišemo $A^+ = VYU^T$, kjer je

$$Y = \begin{matrix} & r & m-r \\ r & \begin{bmatrix} Y_{11} & Y_{12} \\ Y_{21} & Y_{22} \end{bmatrix} \\ n-r & \end{matrix}.$$

Od tod dobimo

$$AA^+ = U \begin{bmatrix} SY_{11} & SY_{12} \\ 0 & 0 \end{bmatrix} U^T.$$

Sedaj upoštevamo, da mora A^+ zadoščati Moore–Penroseovim pogojem. Ker mora biti matrika AA^+ simetrična, od tod sledi $Y_{12} = 0$. Podobno iz simetričnosti matrike A^+A sledi $Y_{21} = 0$. Iz zveze $A^+AA^+ = A^+$ potem dobimo

$$\begin{bmatrix} Y_{11}SY_{11} & 0 \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} Y_{11} & \\ 0 & Y_{22} \end{bmatrix},$$

kar pomeni, da je tudi $Y_{22} = 0$. Končno, iz $AA^+A = A$ dobimo enačbo $SY_{11}S = S$. Ker je matrika S nesingularna, je rešitev enolična in enaka $Y_{11} = S^{-1}$. ∎

Če je $A = U\Sigma V^T$ singularni razcep matrike A in je $\operatorname{rang}(A) = r$, potem iz razcepa sledi

$$A = \sum_{i=1}^{r} \sigma_i u_i v_i^T.$$

Tako s pomočjo singularnega razcepa matriko lahko zapišemo kot vsoto matrik ranga 1. Naslednji izrek pove, kako lahko matriko najbolje aproksimiramo z matriko nižjega ranga.

Izrek 5.8 (Eckart–Young–Mirsky) [5] *Naj bo* $A = U\Sigma V^T$ *singularni razcep matrike* $A \in \mathbb{R}^{m\times n}$, $m \geq n$, *in* $\operatorname{rang}(A) > k$. *Naj bo*

$$A_k = \sum_{i=1}^{k} \sigma_i u_i v_i^T.$$

Potem velja

$$\min_{\operatorname{rang}(B)=k} \|B - A\|_2 = \|A_k - A\|_2 = \sigma_{k+1}$$

in

$$\min_{\operatorname{rang}(B)=k} \|B - A\|_F = \|A_k - A\|_F = \left(\sum_{i=k+1}^{n} \sigma_i^2 \right)^{1/2}.$$

Dokaz. Izrek bomo dokazali le za spektralno normo, kjer je dokaz enostavnejši. Če je $\operatorname{rang}(B) = k$, potem je $\dim \ker B = n - k$. Naj bo $V_{k+1} = [v_1 \ \cdots \ v_{k+1}]$. Ker je $\dim \operatorname{im} V_{k+1} + \dim \ker B = n + 1$, obstaja tak vektor $z \in \operatorname{im} V_{k+1} \cap \ker B$, da je $\|z\|_2 = 1$. Dobimo

$$\|A - B\|_2^2 \geq \|(A - B)z\|_2^2 = \|Az\|_2^2 = \|U\Sigma V^T z\|_2^2 = \|\Sigma V^T z\|_2^2 \geq \sigma_{k+1}^2 \|V^T z\|_2^2 = \sigma_{k+1}^2.$$

Po drugi strani je očitno, da je $\|A_{k+1} - A\|_2 = \sigma_{k+1}$, saj je maksimum $\|V^T z\|_2$ po vseh vektorjih z z normo $\|z\|_2 = 1$ dosežen pri $z = v_{k+1}$. ∎

To pomeni, da je A_k najboljša aproksimacija matrike A z matriko ranga k, σ_{k+1} pa nam pove, kako daleč je A od prostora matrik ranga k. Opazimo lahko še, da je najboljša aproksimacija ranga k enolična natanko takrat, ko je $\sigma_k > \sigma_{k+1}$.

Zgled 5.3 *Singularni razcep lahko uporabimo za aproksimacijo slik. Sliko lahko predstavimo z matriko A, katere elementi predstavljajo nivoje sivine. Če namesto A vzamemo najboljšo aproksimacijo z matriko ranga k, potem namesto mn podatkov potrebujemo le* $(m + n)k$ *podatkov za* $[u_1 \ \cdots \ u_k]$ *in* $[\sigma_1 v_1 \ \cdots \ \sigma_k v_k]$.

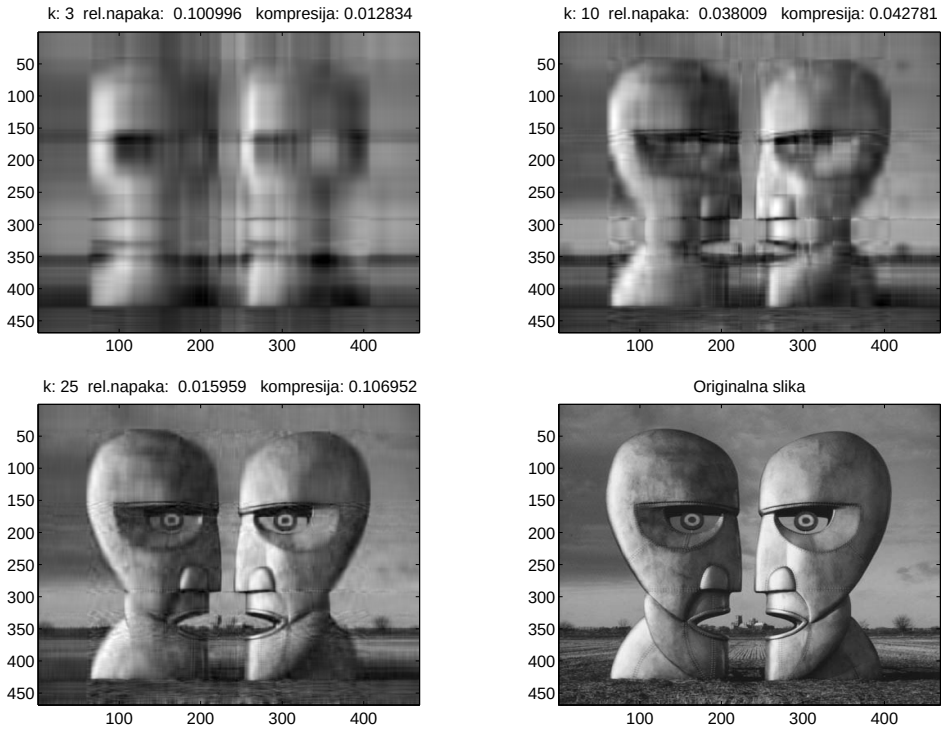
Primer je na sliki 5.4, kjer matriko velikosti 468×468, *ki predstavlja sivinsko sliko, aproksimiramo z matrikami ranga* $3, 10$ *in* 25. *Čeprav je iz slik razvidno, da pristop deluje, se v praksi za aproksimacijo slik uporabljajo še boljše metode, zelo znana je npr. metoda JPEG.* □

Zgled 5.4 *Singularni razcep se uporablja tudi v ozadju delovanja spletnih iskalnikov. Ko uporabljamo iskalnike, vpišemo nekaj ključnih besed in potem v kratkem času dobimo seznam strani, kjer se pojavljajo iskane ključne besede. Kako lahko iskalniki tako hitro poiščejo pravi seznam?*

Lahko si predstavljamo, da so vsi podatki o straneh na spletu shranjeni v $m \times n$ *matriki A, kjer vsaka vrstica predstavlja en spletni dokument, vsak stolpec pa eno izmed ključnih besed, ki se lahko pojavi v kateremkoli dokumentu. V preprostem modelu element* a_{ij} *matrike A vsebuje število pojavitev j-te ključne besede v i-tem dokumentu.*

Iz seznama ključnih besed, ki ga vnesemo v obrazec iskalnika, sestavimo vektor $y \in \mathbb{R}^n$, *ki ima na j-tem mestu 1, če iščemo j-to ključno besedo, sicer pa* 0. *Produkt* $x = Ay$ *je vektor iz* $\mathbb{R}^m$. *Element* x_i *predstavlja, kako dobro se i-ti dokument ujema z našo poizvedbo. Iskalnik kot rezultat vrne kratek seznam dokumentov, za katere so vrednosti* x_i *največje.*

[5] Prvi del sta pokazala ameriška fizika Carl Henry Eckart (1902–1973) in Gale Young (1912–1990) leta 1936, drugi del pa Leonid Mirsky leta 1960.



Slika 5.4: Uporaba singularnega razcepa za stiskanje slike. Sivinska slika velikosti 468×468 je po vrsti aproksimirana z matrikami ranga $3, 10$ in 25.

Matrika A je ogromna, saj je m lahko velikostnega razreda 10^{11}, n pa velikostnega razreda 10^6, in zelo razpršena. Kljub razpršenosti je izračun produkta $x = Ay$ zelo zahteven in je ozko grlo poizvedbe. Rešitev je, da matriko A aproksimiramo z matriko A_k majhnega ranga k (npr. $k = 20$). Če je $A_k = \sum_{i=1}^{k} \sigma_i u_i v i_i^T$, potem kot približek za x vzamemo

$$x_k = A_k y = \sum_{i=1}^{k} \sigma_i (v_i^T y) u_i,$$

ki ga lahko učinkovito izračunamo. □

5.7 Teorija motenj

Sedaj imamo na voljo dovolj rezultatov, da lahko povemo, kakšna je občutljivost linearnega problema najmanjših kvadratov, in primerjamo stabilnost numeričnih metod, ki smo jih predstavili.

Naj bo $A \in \mathbb{R}^{m \times n}$, kjer je $m \geq n$ in $\text{rang}(A) = n$, matrika iz predoločenega sistema. Za takšno matriko definiramo občutljivost kot

$$\kappa_2(A) = \|A\|_2 \|A^+\|_2 = \frac{\sigma_1(A)}{\sigma_n(A)}.$$

Podobno, kot smo naredili za linearni sistem, lahko tudi za predoločeni sistem ocenimo, za koliko se spremeni rešitev, če malo zmotimo matriko A in desno stran b.

Izrek 5.9 *Dana je $A \in \mathbb{R}^{m\times n}$, kjer je $m \geq n$ in $\operatorname{rang}(A) = n$. Naj bo $x = A^+b$ rešitev predoločenega sistema in $r = Ax - b$. Naj bo $\widetilde{x} = (A + \delta A)^+(b + \delta b)$, kjer je*

$$\epsilon := \max\left(\frac{\|\delta A\|_2}{\|A\|_2}, \frac{\|\delta b\|_2}{\|b\|_2}\right) < \frac{1}{\kappa_2(A)}.$$

Potem je matrika $A + \delta A$ ranga n in velja

$$\frac{\|\widetilde{x} - x\|_2}{\|x\|_2} \leq \frac{\epsilon\kappa_2(A)}{1 - \epsilon\kappa_2(A)}\left(2 + (\kappa_2(A) + 1)\frac{\|r\|_2}{\|A\|_2\|x\|_2}\right).$$

V primeru, ko je norma ostanka $\|r\|_2$ majhna, je občutljivost reda $\mathcal{O}(\kappa_2(A))$, če pa ostanek ni zanemarljiv, je občutljivost predoločenega sistema reda $\mathcal{O}(\kappa_2^2(A))$. V primeru $m = n$ je $r = 0$ in takrat se ocena ujema z oceno občutljivosti linearnega sistema iz izreka 3.10.

Če primerjamo število operacij in obratno stabilnost, potem za metode, ki smo jih zaenkrat navedli za reševanje predoločenega sistema $Ax = b$, $m \gg n$, velja (podrobnosti lahko najdete npr. v [15]):

a) Število operacij za normalni sistem in razcep Choleskega je $mn^2 + n^3/3$. Za numerično izračunano rešitev $\widetilde{x}$ velja $(A^TA + E)\widetilde{x} = A^Tb$, kjer je $\|E\|_2 \leq C_1 mnu\|A^TA\|_2$ in C_1 konstanta. Tu smo upoštevali tudi napake, ki se pojavijo pri računanju A^TA in A^Tb.

   Če to oceno združimo z rezultati iz izreka 3.10 o občutljivosti linearnega sistema, dobimo končno oceno, da za izračunano rešitev $\widetilde{x}$ velja

$$\frac{\|\widetilde{x} - x\|_2}{\|x\|_2} \leq \kappa_2(A^TA)\frac{\|E\|_2}{\|A^TA\|_2} = \mathcal{O}(mnu)\kappa_2^2(A).$$

   Pri reševanju preko normalnega sistema tako vedno v oceni napake vektorja $\widetilde{x}$ nastopa faktor $\kappa_2^2(A)$, ne glede na velikost norme ostanka $\|r\|_2$.

b) Če uporabimo QR razcep in modificirano Gram–Schmidtovo ortogonalizacijo, porabimo $2mn^2$ operacij. Če uporabimo Givensove rotacije, porabimo $3mn^2 - n^3$ operacij, v primeru Householderjevih zrcaljenj pa $2mn^2 - \frac{2}{3}n^3$ operacij. V vseh treh primerih za izračunani $\widetilde{x}$ velja, da je eksaktna rešitev po metodi najmanjših kvadratov za matriko $A + \delta A$ in desno stran $b + \delta b$, kjer je $\|\delta A\|_F \leq \mathcal{O}(mnu)\|A\|_F$ in $\|\delta b\|_2 \leq \mathcal{O}(mnu)\|b\|_2$.

   To oceno moramo združiti z rezultati iz izreka 5.9 o občutljivosti predoločenega sistema. Tako dobimo oceno

$$\frac{\|\widetilde{x} - x\|_2}{\|x\|_2} \leq \mathcal{O}(mnu)\kappa_2(A)\left(2 + (\kappa_2(A) + 1)\frac{\|r\|_2}{\|A\|_2\|x\|_2}\right).$$

c) Uporaba singularnega razcepa je dražja od QR razcepa, a tudi natančnejša, če uporabimo metodo, ki zna vse singularne vrednosti izračunati z visoko relativno natančnostjo. Podrobno bomo računanje singularnega razcepa obravnavali v **??**. poglavju.

Če je predoločen sistem tak, da je norma ostanka $\|r\|_2$ majhna, matrika A pa občutljiva, potem bo reševanje preko ortogonalnega razcepa stabilnejše. V tem primeru namreč v oceni za napako $\widetilde{x}$ nastopa le $\kappa_2(A)$, medtem ko pri uporabi normalnega sistema v oceni vedno nastopa $\kappa_2^2(A)$. Kadar je matrika A neobčutljiva, pa dobimo dobre rezultate tudi že preko cenejšega normalnega sistema.

Podobno velja za reševanje nesingularnega kvadratnega sistema $Ax = b$, kjer je $m = n$:

a) Če uporabimo LU razcep, porabimo $\frac{2}{3}n^3$ operacij. Če uporabimo delno pivotiranje, potem metoda v teoriji ni obratno stabilna, v praksi pa je. Kompletno pivotiranje je obratno stabilno, a je zaradi dodatnih $\mathcal{O}(n^3)$ primerjanj toliko dražje, da ga redko uporabljamo.

b) Sistem lahko rešimo tudi s QR razcepom. Tako s Householderjevimi zrcaljenji porabimo $\frac{4}{3}n^3$, z Givensovimi rotacijami pa $2n^3$ operacij. Obe metodi sta obratno stabilni in za izračunani $\widetilde{x}$ velja $(A + \delta A)\widetilde{x} = b + \delta b$, kjer je $\|\delta A\|_F \leq \mathcal{O}(n^2u)\|A\|_F$ in $\|\delta b\|_2 \leq \mathcal{O}(n^2u)\|b\|_2$.

5.8 Problemi defektnega ranga in nedoločeni problemi

Če matrika A ni polnega ranga, potem pravimo, da ima *defekten rang*. V primeru defektnega ranga vektor x, ki minimizira ostanek $\|Ax - b\|_2$ ni enoličen, saj mu lahko prištejemo poljuben vektor iz jedra matrike A. V takšnem primeru pravimo, da je izmed vseh vektorjev x, ki minimizirajo $\|Ax - b\|_2$, rešitev tisti, ki ima minimalno normo $\|x\|_2$.

Trditev 5.10 *Če je $A \in \mathbb{R}^{m\times n}$, $m \geq n$, $\operatorname{rang}(A) = r < n$ in $A = U\Sigma V^T$ singularni razcep, potem ima izmed vseh vektorjev x, ki minimizirajo normo $\|Ax - b\|_2$, najmanjšo normo $x = A^+b$ oziroma*

$$x = \sum_{i=1}^{r} \frac{u_i^T b}{\sigma_i} v_i,$$

za ostanek pa velja

$$\|Ax - b\|_2^2 = \sum_{i=r+1}^{m} (u_i^T b)^2.$$

Dokaz. Za poljuben vektor $x \in \mathbb{R}^n$ velja

$$\|Ax - b\|_2^2 = \|(U^TAV)(V^Tx) - U^Tb\|_2^2 = \|\Sigma a - U^Tb\|_2^2$$

$$= \sum_{i=1}^{r}(\sigma_i a_i - u_i^T b)^2 + \sum_{i=r+1}^{m} (u_i^T b)^2,$$

kjer je $a = V^Tx$. Minimum bo očitno dosežen pri $a_i = \frac{u_i^T b}{\sigma_i}$ za $i = 1, \ldots, r$, medtem ko so komponente $a_{r+1}, \ldots, a_m$ poljubne. Vektor $x = Va$ z minimalno normo dobimo, če vzamemo $a_{r+1} = \cdots = a_m = 0$. ∎

Kaj pa, če je rang sicer n, a je minimalna singularna vrednost σ_n zelo majhna? Potem lahko pričakujemo, da bo imela rešitev veliko normo, saj velja naslednja trditev.

Trditev 5.11 *Naj bo $A \in \mathbb{R}^{m\times n}$, $A = U\Sigma V^T$ singularni razcep in $\sigma_n > 0$.*

1) Če je x rešitev predoločenega sistema $Ax = b$ po metodi najmanjših kvadratov, potem je

$$\|x\|_2 \geq \frac{|u_n^T b|}{\sigma_n}.$$

2) Če b zmotimo v $b + \delta b$, potem se x spremeni v $x + \delta x$, kjer je

$$\|\delta x\|_2 \leq \frac{\|\delta b\|_2}{\sigma_n}.$$

Dokaz. Za točko 1) iz $x = A^+ b = \sum_{i=1}^n \frac{u_i^T b}{\sigma_i} v_i$ sledi

$$\|x\|_2^2 = \sum_{i=1}^n \left(\frac{u_i^T b}{\sigma_i} \right)^2 \geq \frac{(u_n^T b)^2}{\sigma_n^2}.$$

Točka 2) sledi iz $\delta x = A^+ \delta b = \sum_{i=1}^n \frac{u_i^T \delta b}{\delta_i} v_i$, od koder dobimo

$$\|\delta x\|_2 \leq \frac{|u_n^T \delta b|}{\sigma_n}.$$

Enakost je dosežena pri $\delta b = \alpha u_n$. ■

V primeru defektnega ranga velja: če je σ_r najmanjša neničelna singularna vrednost, potem za rešitev $Ax = b$ po metodi najmanjših kvadratov velja

$$x = \sum_{i=1}^r \frac{u_i^T b}{\sigma_i} v_i,$$

od tod pa je razvidno, da je $\|x\|_2 \geq \frac{\|b\|_2}{\sigma_r}$ in da sprememba b v $b + \delta b$ spremeni rešitev kvečjemu za $\frac{\|\delta b\|_2}{\sigma_r}$.

5.9 Regularizacija

Denimo, da rešujemo linearni sistem $Ax = b$, kjer je A nesingularna, a zelo občutljiva matrika velikosti $n \times n$. Predpostavimo še, da v resnici rešujemo sistem z zmoteno desno stranjo $\widetilde{b}$, kjer so poleg b prisotne še majhne motnje, npr. zaradi meritev ali zaokrožitvenih napak. Rešitev zmotenega sistema $\widetilde{x}$ lahko s pomočjo singularnega razcepa $A = U\Sigma V^T$ izrazimo kot

$$\widetilde{x} = \sum_{i=1}^n \frac{u_i^T \widetilde{b}}{\sigma_i} v_i,$$

kjer $\widetilde{x}$ razvijemo po singularnih vektorjih $v_1, \ldots, v_n$.

Če ima matrika A najmanjše singularne vrednosti zelo blizu 0, potem iz trditve 5.11 vemo, da lahko zelo majhna motnja desne strani povsem pokvari rezultat, kar pomeni, da se lahko

$\widetilde{x}$ močno razlikuje od x. Tovrstne težave rešujemo z *regularizacijo*. Splošni nastavek je, da za regularizirano rešitev x_{reg} vzamemo

$$x_{\text{reg}} = \sum_{i=1}^{n} \phi_i \frac{u_i^T \widetilde{b}}{\sigma_i} v_i,$$

kjer so ϕ_i , $i = 1, \ldots, n$, tako imenovani *faktorji filtra*.

Z regularizacijo želimo izločiti tiste komponente v razvoju rešitve $\widetilde{x}$ po desnih singularnih vektorjih v_i, ki so preveč okužene z napako. Glavni metodi za regularizacijo sta:

a) *Odrezani singularni razcep*. Izberemo k in vzamemo

$$\phi_i = \begin{cases} 1 : & 1 \le k \le i \\ 0 : & k+1 \le i \le n. \end{cases}$$

To pomeni, da matriko A nadomestimo z matriko A_k, ki je po izreku 5.8 najboljša aproksimacija matrike A z matriko ranga k, potem pa vzamemo $x_{\text{reg}} = A_k^+ b$.

b) *Regularizacija Tihonova*[6]. Izberemo regularizacijski parameter $\alpha > 0$ in definiramo faktorje

$$\phi_i = \frac{\sigma_i^2}{\sigma_i^2 + \alpha^2}$$

za $i = 1, \ldots, n$.

Po lemi 5.12 je to ekvivalentno iskanju vektorja x, ki minimizira $\|b - Ax\|_2^2 + \alpha^2 \|x\|^2$. Vrednost parametra α moramo primerno izbrati. Če pošljemo α proti 0, potem bo x kar rešitev sistema $Ax = b$, a ker je v b tudi šum, lahko potem pričakujemo, da bo norma dobljenega vektorja zelo velika. Po drugi strani, če vzamemo velik α, potem je bolj pomembno to, da ima x majhno normo, kot to, da reši sistem $Ax = b$. Pri primerni izbiri α norma izračunanega vektorja x ne bo prevelika in hkrati tudi velikost ostanka $b - Ax$ ne bo prevelika.

Lema 5.12 *Regularizacija Tihonova s parametrom α vrne vektor x, ki reši naslednji problem:*

$$\min_{x \in \mathbb{R}^n} \left\{ \|b - Ax\|_2^2 + \alpha^2 \|x\|_2^2 \right\}.$$

Dokaz.

$$\|b - Ax\|_2^2 + \alpha^2 \|x\|_2^2 = \left\| \begin{bmatrix} b \\ 0 \end{bmatrix} - \begin{bmatrix} A \\ \alpha I \end{bmatrix} x \right\|_2^2,$$

torej je minimum dosežen pri $(A^T A + \alpha^2 I)x = A^T b$. Če je $A = U\Sigma V^T$ singularni razcep matrike A, potem dobimo

$$V(\Sigma^T \Sigma + \alpha^2 I) V^T x = V \Sigma^T U^T b.$$

Od tod sledi

$$x = V(\Sigma^T \Sigma + \alpha^2 I)^{-1} \Sigma^T U^T b = \sum_{i=1}^{n} \frac{\sigma_i}{\sigma_i^2 + \alpha^2} u_i^T b v_i = \sum_{i=1}^{n} \frac{\sigma_i^2}{\sigma_i^2 + \alpha^2} \frac{u_i^T b}{\sigma_i} v_i. \qquad \blacksquare$$

[6]Ruski matematik Andrej Nikolajevič Tihonov (1906–1993) je postopek opisal leta 1943.

5.10 Totalni najmanjši kvadrati

Ko rešujemo predoločen sistem, iščemo tak vektor x, da bo norma ostanka $\|Ax - b\|_2$ minimalna. To si lahko predstavljamo tudi drugače.

Če označimo ostanek $r = Ax - b$, potem za vsak vektor x velja $Ax = b + r$, torej $b + r$ leži v podprostoru $\mathrm{im}(A)$. Iskanje rešitve po metodi najmanjših kvadratov je tako ekvivalentno iskanju vektorja $\widetilde{b} \in \mathrm{im}(A)$, ki je v drugi normi najbližji b. Ko tak $\widetilde{b}$ imamo, je rešitev linearnega sistema $Ax = \widetilde{b}$ iskana rešitev po metodi najmanjših kvadratov. Zanima nas torej, za koliko moramo minimalno spremeniti desno stran, da bo spremenjeni sistem rešljiv.

To je smiselno, če je izvor naših podatkov tak, da so napake (ki povzročijo, da $b \notin \mathrm{im}(A)$) prisotne le v vektorju b. Če pa so napake lahko prisotne tudi v matriki A, je bolje iskati najbližji eksaktno rešljiv sistem. Pri *totalnih najmanjših kvadratih* tako iščemo matriko $\widetilde{A}$ in vektor $\widetilde{b}$, da je $\widetilde{b} \in \mathrm{im}(\widetilde{A})$ in je razlika $\| [\, \widetilde{A} \quad \widetilde{b}\,] - [\, A \quad b\,] \|_F$ minimalna.

Do rešitve pridemo s pomočjo singularnega razcepa matrike $[\, A \quad b\,]$. Predpostavimo, da je matrika A polnega ranga in $b \notin \mathrm{im}(A)$, kar pomeni, da je matrika $[\, A \quad b\,]$ ranga $n+1$. Iz $\widetilde{b} \in \mathrm{im}(\widetilde{A})$ sledi, da ima matrika $[\, \widetilde{A} \quad \widetilde{b}\,]$ rang n. Naj bo $[\, A \quad b\,] = U\Sigma V^T$ singularni razcep in naj velja $\sigma_n > \sigma_{n+1} > 0$. Potem iz Eckart–Young–Mirskyjevega izreka vemo, da je matrika ranga n, ki je v Frobeniusovi normi najbližja matriki $[\, A \quad b\,]$, enolična in enaka

$$[\, \widetilde{A} \quad \widetilde{b}\,] = [\, A \quad b\,] - \sigma_{n+1} u_{n+1} v_{n+1}^T.$$

Zgornje matrike ni potrebno eksplicitno izračunati, saj vemo, da je $[\, \widetilde{A} \quad \widetilde{b}\,]\, v_{n+1} = 0$, kar pomeni, da so vsi vektorji iz jedra $[\, \widetilde{A} \quad \widetilde{b}\,]$ oblike αv_{n+1} za $\alpha \in \mathbb{R}$. Če je $\widetilde{x}$ rešitev $\widetilde{A}\widetilde{x} = \widetilde{b}$, je to ekvivalentno $[\, \widetilde{A} \quad \widetilde{b}\,] \begin{bmatrix} \widetilde{x} \\ -1 \end{bmatrix} = 0$. Tako lahko $\widetilde{x}$ izrazimo iz vektorja v_{n+1} kot

$$\widetilde{x} = \frac{-1}{v_{n+1,n+1}} \begin{bmatrix} v_{1,n+1} \\ \vdots \\ v_{n,n+1} \end{bmatrix}.$$

Rešitev je dobro definirana, če je $v_{n+1,n+1} \neq 0$. Če so $\sigma_1' \geq \cdots \geq \sigma_n'$ singularne vrednosti matrike A, se izkaže, da je $\sigma_n' > \sigma_{n+1}$ potreben in zadosten pogoj za obstoj in enoličnost rešitve po metodi totalnih kvadratov. Velja namreč (glej npr. [31]), da sta ekvivalentna pogoja

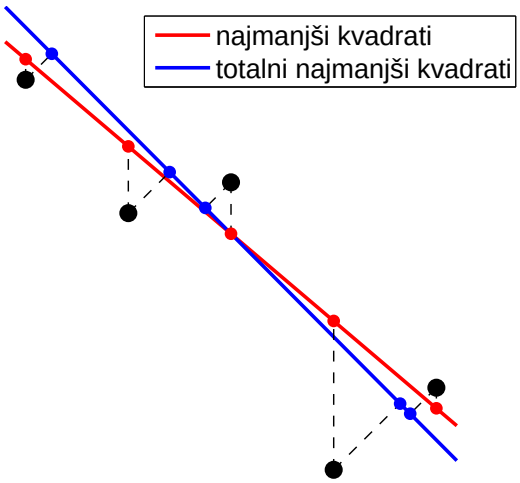
a) $\sigma_n' > \sigma_{n+1}$,

b) $\sigma_n > \sigma_{n+1}$ in $v_{n+1,n+1} \neq 0$.

Zgled 5.5 *Iščemo premico oblike $y = kx$, ki aproksimira točke, podane z vektorjema*

$$x = [\, -2 \quad -1 \quad 0 \quad 1 \quad 2\,]^T, \quad y = [\, 1.5 \quad 0.2 \quad 0.5 \quad -2.3 \quad -1.5\,]^T.$$

Če vzamemo navaden predoločen sistem, dobimo $k_1 = -0.85$, pri totalnih najmanjših kvadratih pa $k_2 = -1.0047$. Razlika med obema rešitvama je prikazana na sliki 5.5. Pri standardni metodi najmanjših kvadratov minimiziramo vsoto kvadratov razlik med y-koordinatami točk in premice, pri totalni metodi najmanjših kvadratov pa iščemo minimalno vsoto kvadratov razdalj točk od premice. □



Slika 5.5: Primerjava aproksimacije s premico po standardni metodi najmanjših kvadratov (rdeča premica) in po totalni metodi najmanjših kvadratov (modra premica).

5.11 Nelinearni problemi najmanjših kvadratov

Denimo, da iščemo krivuljo oblike $y = ae^{bx}$, ki se po metodi najmanjših kvadratov najbolje prilega točkam (x_i, y_i), $i = 1, \ldots, m$. Imamo torej *nelinearen predoločen sistem*

$$\begin{bmatrix} ae^{bx_1} \\ \vdots \\ ae^{bx_m} \end{bmatrix} = \begin{bmatrix} y_1 \\ \vdots \\ y_m \end{bmatrix} \tag{5.2}$$

in iščemo a in b, kjer je dosežen minimum funkcije

$$E(a,b) = \sum_{i=1}^{m} \left(y_i - ae^{bx_i} \right)^2.$$

Pri konkretnem primeru si lahko pomagamo tako, da model lineariziramo, kar naredimo tako, da ga logaritmiramo v

$$\ln y = \ln a + bx.$$

Tako dobimo linearen predoločen sistem

$$\begin{bmatrix} 1 & x_1 \\ \vdots & \vdots \\ 1 & x_m \end{bmatrix} \begin{bmatrix} \ln a \\ b \end{bmatrix} = \begin{bmatrix} \ln y_1 \\ \vdots \\ \ln y_m \end{bmatrix}.$$

Če nelinearni model dobro opisuje podatke, potem bo rešitev linearizirranega modela zelo dober približek za rešitev originalnega problema, ni pa to prava točna rešitev.

V splošnem nelinearnem primeru imamo nelinearno preslikavo $F : \mathbb{R}^n \to \mathbb{R}^m$, kjer je $m > n$, iščemo pa $x \in \mathbb{R}^n$, kjer je dosežen minimum

$$\phi(x) = \frac{1}{2}\|F(x)\|_2^2.$$

Denimo, da je $x^{(r)} \in \mathbb{R}^n$ tekoči približek za minimum funkcije ϕ. Sedaj iščemo popravek Δx, da bo v $x^{(r)} + \Delta x$ lokalni minimum. Razvoj v Taylorjevo vrsto nam da

$$F(x^{(r)} + \Delta x) = F(x^{(r)}) + JF(x^{(r)})\Delta x + \cdots .$$

Če zanemarimo kvadratne člene in hočemo, da bo pri $x^{(r)} + \Delta x$ minimum ϕ, dobimo linearni predoločeni sistem

$$JF(x^{(r)})\Delta x = -F(x^{(r)}).$$

Algoritem je podoben Newtonovi metodi, edina razlika je, da v vsakem koraku namesto kvadratnega rešujemo predoločeni sistem:

$$\begin{aligned} JF(\boldsymbol{x}^{(r)})\Delta\boldsymbol{x}^{(r)} &= -F(\boldsymbol{x}^{(r)}), \\ \boldsymbol{x}^{(r+1)} &= \boldsymbol{x}^{(r)} + \Delta\boldsymbol{x}^{(r)}, \quad r = 0, 1, \ldots . \end{aligned}$$

V primeru $m > n$ imamo Gauss-Newtonovo metodo, torej

$$x^{(r+1)} = x^{(r)} - JF(x^{(r)})^{+}F(x^{(r)}),$$

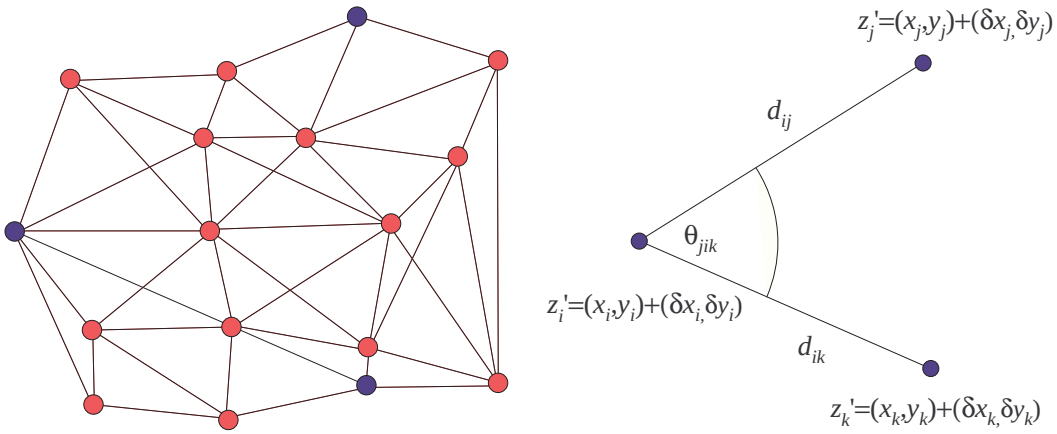
kjer je $JF(x^{(r)})^{+} \in \mathbb{R}^{m\times n}$ psevdoinverz Jacobijeve matrike.

V konkretnem primeru (5.2) dobimo

$$JF(a,b) = \begin{bmatrix} e^{bx_1} & ax_1e^{bx_1} \\ \vdots & \vdots \\ e^{bx_m} & ax_me^{bx_m} \end{bmatrix}.$$

Zgled 5.6 *Z nelinearnim problemom najmanjših kvadratov se srečamo pri izravnavi geodetskih točk. Dano imamo mrežo točk v ravnini. Poznamo razdalje med nekaterimi pari točk in pa kote med nekaterimi trojicami točk. Nekatere točke so znane (fiksne), ostale pa so znane manj natančno, na podlagi meritev pa bi radi njihovo točnost izboljšali.*

Občasno je potrebno točke iz mreže izračunati natančneje, saj so točke vedno bolj goste, premikanje tektonskih plošč premika točke, ipd.



Tako dobimo enačbe za razdalje:

$$d_{ij}^2 = \big((x_j + \delta x_j) - (x_i + \delta x_i)\big)^2 + \big((y_j + \delta y_j) - (y_i + \delta y_i)\big)^2$$

in kote

$$\cos^2\theta_{jik} \cdot d_{ij}^2 d_{ik}^2 = \Big((z_j' - z_i')^T (z_k' - z_i')\Big)^2.$$

V enačbah zanemarimo vse kvadratne δ člene in dobimo predoločen sistem za δ_i. Pri tem nekatere točke ne premikamo, npr. referenčne točke prvega reda.

V ZDA so npr. leta 1974 za potrebe izravnave točk v celotni državi reševali sistem s 700000 točkami in to je bil takrat največji linearni sistem rešen z računalnikom. □

5.12 Zvezni problem najmanjših kvadratov

Denimo, da je dana realna funkcija $f(x)$. Na intervalu $I = [0,1]$ bi jo radi čim bolje aproksimirali s kubičnim polinomom. Pri metodi najmanjših kvadratov iščemo koeficiente polinoma $p(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3$, da bo razlika

$$E = \int_0^1 (f(x) - p(x))^2 dx$$

minimalna.

Polinom p je linearna kombinacija baznih polinomov $1, x, x^2, x^3$. Tisti polinom p, ki predstavlja rešitev, je pravokotna projekcija funkcije f na prostor vseh polinomov stopnje 3, torej mora veljati

$$\langle f(x) - p(x), x^k \rangle = 0 \qquad \text{za } k = 0,1,2,3,$$

kjer je skalarni produkt dveh funkcij definiran z

$$\langle f, g \rangle = \int_0^1 f(x) g(x) dx.$$

Denimo, da na intervalu $[a,b]$ iščemo najboljšo aproksimacijo po metodi najmanjših kvadratov funkcije f z linearno kombinacijo baznih funkcij $p_1, \ldots, p_n$. Potem je rešitev $p(x) = \alpha_1 p_1(x) + \cdots + \alpha_n p_n(x)$, kjer so $\alpha_1, \ldots, \alpha_n$ rešitve linearnega sistema

$$\begin{bmatrix} \langle p_1, p_1 \rangle & \langle p_1, p_2 \rangle & \cdots & \langle p_1, p_n \rangle \\ \langle p_2, p_1 \rangle & \langle p_2, p_2 \rangle & \cdots & \langle p_2, p_n \rangle \\ \vdots & \vdots & & \vdots \\ \langle p_n, p_1 \rangle & \langle p_n, p_2 \rangle & \cdots & \langle p_n, p_n \rangle \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_n \end{bmatrix} = \begin{bmatrix} \langle p_1, f \rangle \\ \langle p_2, f \rangle \\ \vdots \\ \langle p_n, f \rangle \end{bmatrix}.$$

Matriko skalarnih produktov iz zgornjega sistema imenujemo *Gramova matrika*. Če so bazne funkcije ortogonalne, kar lahko npr. dosežemo z uporabo Gram–Schmidtove ortogonalizacije, potem je Gramova matrika diagonalna.

5.13 Podobni problemi

Če imamo sistem $Ax = b$, kjer je $A \in \mathbb{R}^{m \times n}$, $m < n$ in $\text{rang}(A) = m$, potem je to *nedoločen sistem*. Ker je $\dim \ker A = n - m$, rešitev ni enolična, saj ji lahko prištejemo poljuben vektor z iz jedra matrike A. Zaradi tega iščemo tisto rešitev x, ki ima minimalno normo $\|x\|_2$. Iskana rešitev je $x = A^+ b$ oziroma

$$x = \sum_{i=1}^{m} \frac{u_i^T b}{\sigma_i} v_i.$$

Za splošni sistem $Ax = b$ lahko rečemo, da v primeru, ko matrika A ni polnega ranga, izmed vseh rešitev x, ki minimizirajo $\|Ax - b\|_2$ vzamemo tisto z minimalno normo $\|x\|_2$. Če je $\text{rang}(A) = r$, potem je iskana rešitev kar

$$x = A^+ b = \sum_{i=1}^{r} \frac{u_i^T b}{\sigma_i} v_i.$$

Pri numeričnem računanju je težko ugotoviti točen rang matrike, zato ponavadi tiste singularne vrednosti, ki so blizu 0, proglasimo za 0 in nato preko singularnega razcepa poiščemo rešitev.

Matlab

Za reševanje linearnega sistema $Ax = b$ po metodi najmanjših kvadratov lahko uporabljamo v Matlabu operator \ v obliki `x=A\b` tako kot pri kvadratnem linearnem sistemu. Če je sistem kvadraten, Matlab uporabi LU razcep z delnim pivotiranjem, če pa gre za pravokoten sistem, ga reši po metodi najmanjših kvadratov z uporabo QR razcepa in Householderjevih zrcaljenj.

QR razcep dobimo z ukazom `qr`. Uporaba:

- `[Q,R]=qr(A)`: razširjeni QR razcep, Q je ortogonalna matrika, R pa zgornja trapezna matrika, da je $QR = A$.
- `[Q,R]=qr(A,0)`: ekonomični QR razcep, Q je matrika z ortonormiranimi stolpci, R pa zgornja trikotna matrika, da je $QR = A$.

Za singularni razcep in psevdoinverz imamo ukaza `svd` in `pinv`. Uporaba:

- `[U,S,V]=svd(A)`: singularni razcep, U in V sta ortogonalni matriki, S pa pravokotna matrika s singularnimi vrednostmi na glavni diagonali in ničlami drugje, da je $A = USV^T$.
- `[U,S,V]=svd(A,0)`: ekonomični singularni razcep, U je matrika z ortonormiranimi stolpci, S je diagonalna matrika s singularnimi vrednostmi na glavni diagonali, V pa ortogonalna matrika, da je $A = USV^T$.
- `B=pinv(A)`: B je psevdoinverz matrike A.

Dodatna literatura

Za reševanje predoločenih sistemov je na voljo obsežna literatura. V slovenščini sta na voljo knjigi [5] in [8]. Kar se tiče tuje literature, lahko skoraj vse algoritme, ki jih potrebujemo pri reševanju predoločenih sistemov, skupaj s potrebno analizo, najdete v [9]. Zelo lepo napisana učbenika s številnimi algoritmi in primeri sta tudi [7] in [13]. Več o regularizaciji lahko najdete npr. v [12].

Poglavje 6

Nesimetrični problem lastnih vrednosti

6.1 Uvod

Dana je matrika $A \in \mathbb{R}^{n\times n}$. Če neničelen vektor $x \in \mathbb{C}^n$ in skalar $\lambda \in \mathbb{C}$ zadoščata enačbi $Ax = \lambda x$, potem je λ *lastna vrednost*, x pa *(desni) lastni vektor* matrike A. Neničelen vektor $y \in \mathbb{C}^n$, za katerega velja $y^H A = \lambda y^H$, je *levi lastni vektor* matrike A. Levi in desni lastni vektorji, ki pripadajo različnim lastnim vrednostim, so med seboj ortogonalni, saj velja naslednja lema.

Lema 6.1 *Naj bosta λ in μ različni lastni vrednosti matrike A. Če je x desni lastni vektor, ki pripada λ, y pa levi lastni vektor, ki pripada μ, potem sta x in y ortogonalna.*

Dokaz. Enakost $Ax = \lambda x$ pomnožimo z leve z y^H, enakost $y^H A = \mu y^H$ pa z desne z x. Dobimo

$$y^H Ax = \lambda y^H x = \mu y^H x,$$

to pa je zaradi $\lambda \neq \mu$ lahko izpolnjeno le v primeru, ko je $y^H x = 0$. ∎

V posebnem primeru, ko je matrika A simetrična, iz zgornje leme sledi, da so lastni vektorji, ki pripadajo različnim lastnim vrednostim, paroma ortogonalni. To je posledica tega, da so za simetrično matriko desni lastni vektorji enaki levim.

Pri algoritmih za računanje lastnih vektorjev zadošča, da obravnavamo le desne lastne vektorje. Namreč, če je y levi lastni vektor matrike A za lastno vrednost λ, potem je y desni lastni vektor matrike A^H za lastno vrednost $\overline{\lambda}$.

Pravimo, da se da matriko A diagonalizirati, če obstajata nesingularna matrika $X = [x_1 \ \cdots \ x_n]$ in diagonalna matrika $\Lambda = \mathrm{diag}(\lambda_1, \ldots, \lambda_n)$, da je $A = X\Lambda X^{-1}$. V tem primeru je $Ax_i = \lambda_i x_i$ za $i = 1, \ldots, n$, kar pomeni, da so stolpci matrike X lastni vektorji, na diagonali matrike Λ pa ležijo lastne vrednosti.

Če je S nesingularna matrika, potem pravimo, da sta matriki A in $B = S^{-1}AS$ *podobni*. V tem primeru imata A in B iste lastne vrednosti. Velja, da je x desni lastni vektor za matriko A natanko tedaj, ko je $S^{-1}x$ desni lastni vektor za matriko B.

Lastne vrednosti matrike A so ničle karakterističnega polinoma $p(\lambda) = \det(A - \lambda I)$. Vsaka $n \times n$ matrika A ima tako n lastnih vrednosti $\lambda_1, \ldots, \lambda_n$, pri čemer večkratne ničle štejemo

večkrat. Če je λ lastna vrednost matrike A, potem je njena *algebraična večkratnost* enaka večkratnosti λ kot ničle karakterističnega polinoma matrike A, *geometrijska večkratnost* pa je enaka dimenziji podprostora $\ker(A - \lambda I)$ in je vedno manjša ali enaka algebraični večkratnosti. Če je algebraična večkratnost lastne vrednosti ena, potem pravimo, da je lastna vrednost enostavna.

Ker se ničel splošnega polinoma stopnje pet ali več ne da izračunati drugače kot numerično, to velja tudi za lastne vrednosti in direktne metode za računanje lastnih vrednosti ne obstajajo. Lastne vrednosti tako vedno računamo z iterativnimi postopki.

Računanje lastnih vrednosti preko ničel eksplicitno izračunanega karakterističnega polinoma v splošnem ni priporočljivo. Algoritmi za računanje koeficientov karakterističnega polinoma namreč niso stabilni, vemo pa, da so ničle polinoma lahko zelo občutljive na motnje koeficientov, kot kaže npr. Wilkinsonov primer iz zgleda 1.4.

Zgled 6.1 *Pri eksplicitni uporabi karakterističnega polinoma lahko izgubimo natančnost. Naj bo*

$$A = \begin{bmatrix} 1 & \epsilon \\ \epsilon & 1 \end{bmatrix}$$

za $\epsilon = \sqrt{u}/2$, *kjer je* u *osnovna zaokrožitvena napaka. Če izračunamo karakteristični polinom, potem zaradi zaokroževanja dobimo* $\det(A - \lambda I) = \lambda^2 - 2\lambda + 1$. *Na ta način bi izračunali dvojno lastno vrednost* 1, *pravi lastni vrednosti matrike* A *pa sta* $1 - \epsilon$ *in* $1 + \epsilon$. □

Računanje lastnih vrednosti preko eksplicitno izračunanega karakterističnega polinoma torej ni numerično stabilno. V nadaljevanju bomo spoznali več različnih stabilnih algoritmov za izračun lastnih vrednosti in vektorjev.

6.2 Schurova forma

Vemo, da za vsako $n \times n$ matriko A obstajata taka nesingularna matrika X in bločno diagonalna matrika $J = \operatorname{diag}(J_1, \ldots, J_k)$, ki ji pravimo *Jordanova*[1] *forma*, da je $X^{-1}AX = J$, kjer je

$$J_i = \begin{bmatrix} \lambda_i & 1 & & \\ & \lambda_i & \ddots & \\ & & \ddots & 1 \\ & & & \lambda_i \end{bmatrix}$$

Jordanova kletka velikosti $m_i \times m_i$ za $i = 1, \ldots, k$ in $n = m_1 + \cdots + m_k$.

Jordanova forma pove veliko o matriki A. Iz nje lahko npr. preberemo vse lastne vrednosti matrike in njihove algebraične in geometrijske večkratnosti. Zelo pomembna je tudi za računanje vrednosti funkcij matrik. Tako ima npr. rešitev sistema diferencialnih enačb s konstantnimi koeficienti $y'(t) = Ay(t)$ obliko $y(t) = y(t_0)e^{A(t-t_0)}$. Za rešitev je potrebno znati izračunati matriko e^A, ki je definirana z razvojem v vrsto

$$e^A = \sum_{j=0}^{\infty} \frac{1}{j!} A^j.$$

[1] Francoski matematik Marie Ennemond Camille Jordan (1838–1922) jo je objavil leta 1870.

Če je funkcija f definirana in dovoljkrat zvezno odvedljiva v lastnih vrednostih matrike A, potem s pomočjo Jordanove forme $A = XJX^{-1}$ velja enakost $f(A) = Xf(J)X^{-1}$, kjer je $f(J) = \operatorname{diag}(f(J_1), \ldots, f(J_k))$ in

$$f(J_i) = \begin{bmatrix} f(\lambda_i) & f'(\lambda_i) & \cdots & \frac{f^{(m_i-1)}(\lambda_i)}{(m_i-1)!} \\ & f(\lambda_i) & \ddots & \vdots \\ & & \ddots & f'(\lambda_i) \\ & & & f(\lambda_i) \end{bmatrix}$$

za $i = 1, \ldots, k$.

Na žalost je Jordanova forma zelo občutljiva in neprimerna za numerično računanje. Ker ni zvezna funkcija elementov matrike A, jo lahko v primeru večkratnih lastnih vrednosti majhne motnje popolnoma spremenijo. Zato računanje funkcij matrik preko Jordanove forme ni stabilno.

Zgled 6.2 *Matrika*

$$A = \begin{bmatrix} 0 & 1 & & \\ & \ddots & \ddots & \\ & & \ddots & 1 \\ & & & 0 \end{bmatrix}$$

je že kar v Jordanovi formi z eno samo kletko $n \times n$, za poljuben $\epsilon > 0$ pa ima Jordanova forma matrike

$$A(\epsilon) = \begin{bmatrix} 0 & 1 & & \\ & \ddots & \ddots & \\ & & \ddots & 1 \\ \epsilon & & & 0 \end{bmatrix}$$

n kletk velikosti 1×1 z lastnimi vrednostmi $\sqrt[n]{\epsilon}$. □

Računanje Jordanove forme tudi ni obratno stabilno. Denimo, da smo za matriko A numerično izračunali $\widetilde{X}$ in $\widetilde{J}$. Sedaj nas zanima, ali je $\widetilde{X}^{-1}(A + \delta A)\widetilde{X} = \widetilde{J}$ za neko matriko δA, ki je blizu 0. Pri tem predpostavimo, da je $\widetilde{X}$ točna, za $\widetilde{J}$ pa velja $\widetilde{J} = J + \delta J$. Iz $X^{-1}(A + \delta A)X = J + \delta J$ sledi $X^{-1}\delta A X = \delta J$, od tod pa lahko ocenimo

$$\|\delta A\| \leq \|X\| \cdot \|X^{-1}\| \cdot \|\delta J\| = \kappa(X)\|\delta J\|.$$

Ker je v splošnem občutljivost $\kappa(X)$ lahko poljubno velika, računanje Jordanove forme ni obratno stabilno.

Za stabilnost bi bilo bolje, če bi bila prehodna matrika X kar unitarna. Izkaže se, da z unitarno podobnostno transformacijo lahko matriko na stabilen način transformiramo v zgornjo trikotno matriko.

Izrek 6.2 (Schur) [2] *Za vsako matriko A obstajata unitarna matrika U in zgornja trikotna matrika T, da je $U^H A U = T$.*

[2] Izrek je leta 1909 zapisal nemški matematik Issai Schur (1875–1941), ki je znan tudi po svojih rezultatih iz teorije grup. S svojim mentorjem Frobeniusom sta bila med prvimi, ki so se ukvarjali s teorijo matrik.

Razcep $A = UTU^H$ iz zgornjega izreka imenujemo *Schurov razcep*, samo zgornjo trikotno matriko T pa *Schurova forma*. Vemo, da ima realna matrika lahko kompleksne lastne vrednosti in vektorje, ki nastopajo v konjugiranih parih. Zaradi tega sta matriki T in U iz Schurovega razcepa tudi za realno matriko lahko kompleksni.

Dokaz. Naredimo indukcijo po n. Za $n = 1$ izrek očitno velja, saj vzamemo $U = [1]$ in $T = A$.

Predpostavimo, da izrek velja za $n-1$ in ga dokažimo za n. Naj bo λ lastna vrednost matrike A in x njen normiran lastni vektor. Obstaja taka unitarna matrika U_1, da je $U_1 e_1 = x$ (skonstruiramo jo lahko npr. kar s kompleksnim Householderjevim zrcaljenjem). Matrika $B = U_1^H A U_1$ ima obliko

$$B = \begin{array}{c} \\ 1 \\ n-1 \end{array} \begin{array}{c} \begin{array}{cc} 1 & n-1 \end{array} \\ \begin{bmatrix} \lambda & \times \cdots \times \\ 0 & C \end{bmatrix} \end{array},$$

saj je $Be_1 = U_1^H A U_1 e_1 = U_1^H A x = U_1^H \lambda x = \lambda e_1$. Po indukcijski predpostavki obstaja Schurova forma za $(n-1) \times (n-1)$ matriko C. Torej obstaja taka unitarna matrika V_1, da je $V_1^H C V_1 = T_1$ zgornja trikotna matrika. Sedaj je

$$\underbrace{\begin{bmatrix} 1 & 0 \\ 0 & V_1^H \end{bmatrix}}_{V^H} B \underbrace{\begin{bmatrix} 1 & 0 \\ 0 & V_1 \end{bmatrix}}_{V} = \begin{bmatrix} \lambda & \times \cdots \times \\ 0 & T_1 \end{bmatrix}$$

zgornja trikotna matrika in $\underbrace{V^H U_1^H}_{U^H} A \underbrace{U_1 V}_{U}$ je Schurova forma matrike A. ∎

Tako kot Jordanova tudi Schurova forma ni enolična, saj je npr. vrstni red diagonalnih elementov λ_i lahko poljuben.

V primeru realne matrike se lahko izognemo kompleksnim matrikam, če dopustimo, da v Schurovem razcepu namesto zgornje trikotne matrike dobimo *kvazi zgornjo trikotno matriko*. Ta ima na diagonali lahko bloke 2×2, v katerih so skriti konjugirani pari kompleksnih lastnih vrednosti.

Izrek 6.3 *Za vsako realno matriko A obstajata ortogonalna matrika Q in kvazi zgornja trikotna matrika T, da je $Q^T A Q = T$ (realna Schurova forma).*

Dokaz. Spet uporabimo indukcijo. Če je $\lambda \in \mathbb{R}$, lahko nadaljujemo tako kot pri dokazu izreka 6.2, zato predpostavimo, da velja $\lambda \notin \mathbb{R}$. Potem imamo poleg lastnega para (λ, x) tudi lastni par $(\overline{\lambda}, \overline{x})$. Če definiramo realna vektorja

$$x_R = \frac{1}{2}(x + \overline{x}), \quad x_I = \frac{1}{2i}(x - \overline{x}),$$

potem obstaja taka ortogonalna matrika

$$U = \begin{array}{c} \begin{array}{cc} 2 & n-2 \end{array} \\ \begin{bmatrix} U_1 & U_2 \end{bmatrix} \end{array},$$

da je $\mathrm{Lin}(U_1) = \mathrm{Lin}(\{x_R, x_I\}) = \mathrm{Lin}(\{x, \overline{x}\})$. Ker je $\mathrm{Lin}(U_1)$ invarianten podprostor za A, velja

$$U^T A U = \begin{array}{c} \\ 2 \\ n-2 \end{array} \begin{array}{c} \begin{array}{cc} 2 & n-2 \end{array} \\ \begin{bmatrix} B & \times \cdots \times \\ 0 & C \end{bmatrix} \end{array},$$

kjer sta λ in $\overline{\lambda}$ lastni vrednosti 2×2 matrike B, preostale lastne vrednosti pa so v matriki C. Nadaljujemo podobno kot pri dokazu izreka 6.2. ∎

6.3 Teorija motenj

Kot pri ostalih problemih, bi tudi tu radi vedeli, koliko so občutljive lastne vrednosti in lastni vektorji. Zanima nas, kaj se z njimi dogaja, ko matriko zmotimo. Vemo, da so lastne vrednosti zvezne funkcije elementov matrike, saj so ničle karakterističnega polinoma, ničle polinoma pa so zvezne funkcije koeficientov polinoma. Če se da matriko diagonalizirati, potem naslednji izrek pove, da je sprememba lastnih vrednosti omejena z občutljivostjo matrike lastnih vektorjev.

Izrek 6.4 (Bauer–Fike) [3] *Predpostavimo, da se da matriko A diagonalizirati kot $A = X\Lambda X^{-1}$, kjer je $\Lambda = \mathrm{diag}(\lambda_1, \ldots, \lambda_n)$ diagonalna matrika lastnih vrednosti. Potem vse lastne vrednosti matrike $A + \epsilon E$ ležijo v uniji n krogov*

$$K_i = \{z \in \mathbb{C} : |z - \lambda_i| \leq \epsilon \kappa(X) \|E\|\}, \quad i = 1, \ldots, n,$$

kjer za normo lahko vzamemo katerokoli izmed norm $\|.\|_1$, $\|.\|_2$ ali $\|.\|_\infty$.

Dokaz. Naj bo $\lambda(\epsilon)$ lastna vrednost matrike $A + \epsilon E$. Predpostavimo lahko, da se $\lambda(\epsilon)$ razlikuje od vseh lastnih vrednosti $\lambda_1, \ldots, \lambda_n$, saj sicer izrek očitno drži. Matrika $A + \epsilon E - \lambda(\epsilon) I$ je singularna. Zapišemo lahko

$$\begin{aligned} X^{-1}(A + \epsilon E - \lambda(\epsilon) I)X &= \Lambda - \lambda(\epsilon) I + \epsilon X^{-1} E X \\ &= (\Lambda - \lambda(\epsilon) I)\Big(I + \epsilon(\Lambda - \lambda(\epsilon) I)^{-1} X^{-1} E X\Big). \end{aligned}$$

Ker je matrika $\Lambda - \lambda(\epsilon) I$ nesingularna, mora biti $I + \epsilon(\Lambda - \lambda(\epsilon) I)^{-1} X^{-1} E X$ singularna matrika. To pomeni (uporabimo lemo 3.9), da je

$$1 \leq \|\epsilon(\Lambda - \lambda(\epsilon) I)^{-1} X^{-1} E X\| \leq \epsilon \|(\Lambda - \lambda(\epsilon) I)^{-1}\| \|X^{-1}\| \|E\| \|X\|.$$

Iz

$$\|(\Lambda - \lambda(\epsilon) I)^{-1}\| = \frac{1}{\min_{i=1,\ldots,n} |\lambda_i - \lambda(\epsilon)|}$$

sledi

$$\min_{i=1,\ldots,n} |\lambda_i - \lambda(\epsilon)| \leq \epsilon \kappa(X) \|E\|.$$

∎

[3] Rezultat sta leta 1960 objavila nemški matematik Friedrich Ludwig Bauer (r. 1924) in Charles Theodore Fike. Bauer je znan predvsem po svojem delu na področju računalništva, kjer je vpeljal podatkovno strukturo sklad, pomembno pa je tudi sodeloval pri razvoju prvih programskih jezikov v 60. letih prejšnjega stoletja.

Opomba 6.1 *Če unija krogov razpade na povezane komponente, potem iz zveznosti lastnih vrednosti sledi, da vsaka komponenta vsebuje natanko toliko lastnih vrednosti, kolikor krogov jo sestavlja.*

V primeru, ko je matrika simetrična, lahko lastne vektorje izberemo tako, da tvorijo ortonormirano bazo. Spektralna občutljivost matrike lastnih vektorjev je potem 1 in iz Bauer–Fikeovega izreka dobimo naslednjo posledico.

Posledica 6.5 *Če sta matriki A in E simetrični, potem pri predpostavkah izreka 6.4 za vsako lastno vrednost $\lambda(\epsilon)$ matrike $A+\epsilon E$ velja*

$$\min_{i=1,\ldots,n} |\lambda(\epsilon)-\lambda_i| \leq \epsilon \|E\|_2.$$

Bauer–Fikeov izrek je ponavadi preveč splošen, saj za motnje vseh lastnih vrednosti uporabi isto zgornjo mejo. V resnici so lahko posamezne lastne vrednosti bolj občutljive od ostalih. Naslednji izrek pove, od česa je odvisna občutljivost enostavne lastne vrednosti.

Izrek 6.6 *Naj bo λ_i enostavna lastna vrednost matrike A z normiranima levim in desnim lastnim vektorjem y_i in x_i. Če je $\lambda_i+\delta\lambda_i$ ustrezna lastna vrednost zmotene matrike $A+\delta A$, potem velja*

$$\lambda_i+\delta\lambda_i = \lambda_i + \frac{y_i^H \delta A x_i}{y_i^H x_i} + \mathcal{O}(\|\delta A\|^2).$$

Izraz ustrezna lastna vrednost v zgornjem izreku pomeni, da si izberemo tisto lastno vrednost zmotene matrike, za katero velja $\lim_{\|\delta A\|\to 0}(\delta\lambda_i)=0$.

Dokaz. Velja $Ax_i=\lambda_i x_i$ in $(A+\delta A)(x_i+\delta x_i)=(\lambda_i+\delta\lambda_i)(x_i+\delta x_i)$, kjer smo z $x_i+\delta x_i$ označili lastni vektor zmotene matrike. Če zanemarimo člene (od tod na koncu dobimo $\mathcal{O}(\|\delta A\|^2)$), ki vsebujejo produkte dveh majhnih popravkov in pomnožimo enačbo z leve z y_i^H, dobimo

$$\delta\lambda_i = \frac{y_i^H \delta A x_i}{y_i^H x_i}.$$

■

Definicija 6.7 *Naj bo λ_i enostavna lastna vrednost, x_i in y_i pa pripadajoča desni in levi lastni vektor. Če definiramo*

$$s_i := \frac{y_i^H x_i}{\|x_i\|_2 \|y_i\|_2},$$

potem je $|s_i|^{-1}$ občutljivost *enostavne lastne vrednosti λ_i. Če je λ_i večkratna lastna vrednost, je njena občutljivost neskončna.*

Če je matrika A simetrična, potem imajo vse enostavne lastne vrednosti najmanjšo možno občutljivost 1, saj so levi lastni vektorji enaki desnim.

Zgled 6.3 *Matrika*

$$A_1 = \begin{bmatrix} -149 & -50 & -154 \\ 537 & 180 & 546 \\ -27 & -9 & -25 \end{bmatrix}$$

ima eksaktne lastne vrednosti $1, 2, 3$. *V Matlabu z ukazom* `eig(A)`, *ki uporablja obratno stabilen algoritem, ter z računanjem v dvojni natančnosti, dobimo*

$$\begin{aligned} \widehat{\lambda}_1 &= 1.000000000010722 \\ \widehat{\lambda}_2 &= 1.999999999991790 \\ \widehat{\lambda}_3 &= 2.999999999997399. \end{aligned}$$

Matrika Godunova[4]

$$A_2 = \begin{bmatrix} 289 & 2064 & 336 & 128 & 80 & 32 & 16 \\ 1152 & 30 & 1312 & 512 & 288 & 128 & 32 \\ -29 & -2000 & 756 & 384 & 1008 & 224 & 48 \\ 512 & 128 & 640 & 0 & 640 & 512 & 128 \\ 1053 & 2256 & -504 & -384 & -756 & 800 & 208 \\ -287 & -16 & 1712 & -128 & 1968 & -30 & 2032 \\ -2176 & -287 & -1565 & -512 & -541 & -1152 & -289 \end{bmatrix}$$

ima eksaktne lastne vrednosti $-4, -2, -1, 0, 1, 2, 4$. *V tem primeru Matlab izračuna naslednje približke za lastne vrednosti:*

$$\begin{aligned} \widehat{\lambda}_1 &= 5.0968 + 1.8932i \\ \widehat{\lambda}_2 &= 5.0968 - 1.8932i \\ \widehat{\lambda}_3 &= 1.2157 + 4.3784i \\ \widehat{\lambda}_4 &= 1.2157 - 4.3784i \\ \widehat{\lambda}_5 &= -5.6738 \\ \widehat{\lambda}_6 &= -3.4756 + 3.4463i \\ \widehat{\lambda}_7 &= -3.4756 - 3.4463i. \end{aligned}$$

V obeh primerih so izračunane lastne vrednosti točne lastne vrednosti malo zmotene matrike, saj Matlab uporablja obratno stabilen algoritem. Medtem, ko so lastne vrednosti matrike A_1 *izračunane zelo natančno, so med izračunanimi in točnimi lastnimi vrednostmi matrike* A_2 *zelo velike razlike. Te so posledica tega, da so občutljivosti lastnih vrednosti matrike* A_2 *velikostnega reda* 10^{13}. *Če to primerjamo z matriko* A_1, *kjer imajo lastne vrednosti občutljivosti velikostnega reda* 10^2, *je očitno, da občutljivost lastne vrednosti pomembno vpliva na natančnost izračunanih približkov.* □

V primeru, ko se da matriko diagonalizirati, lahko kaj povemo tudi o občutljivosti lastnih vektorjev.

Izrek 6.8 *Naj bo* $A = X\Lambda X^{-1} = Y^H \Lambda Y^{-H}$, *kjer je* $X = [x_1 \ \cdots \ x_n]$ *matrika normiranih desnih lastnih vektorjev,* $Y = [y_1 \ \cdots \ y_n]$ *matrika normiranih levih lastnih vektorjev in* $\Lambda = \operatorname{diag}(\lambda_1, \ldots, \lambda_n)$ *diagonalna matrika lastnih vrednosti. Če je* λ_i *enostavna lastna vrednost in* $\lambda_i + \delta\lambda_i$ *ustrezna lastna vrednost zmotene matrike* $A + \delta A$ *s pripadajočim lastnim vektorjem* $x_i + \delta x_i$, *potem velja*

$$x_i + \delta x_i = x_i + \sum_{\substack{j=1 \\ j \neq i}}^{n} \frac{y_j^H \delta A x_i}{(\lambda_i - \lambda_j) s_j} x_j + \mathcal{O}(\|\delta A\|^2).$$

[4] Po ruskem matematiku Sergeju Konstantinoviču Godunovu (r. 1929).

Dokaz. Zmoteni lastni vektor lahko izrazimo v bazi lastnih vektorjev matrike A kot

$$x_i + \delta x_i = x_i + \sum_{\substack{j=1 \\ j \neq i}}^{n} \alpha_j x_j.$$

V enakost $(A + \delta A)(x_i + \delta x_i) = (\lambda_i + \delta\lambda_i)(x_i + \delta x_i)$ vstavimo zgornji razvoj, jo pomnožimo z leve z y_k^H, kjer je $k \neq i$, nato pa tako kot prej zanemarimo člene, ki vsebujejo produkte dveh majhnih popravkov. Dobimo

$$\alpha_k = \frac{y_k^H \delta A x_i}{(\lambda_i - \lambda_k) s_k},$$

pri čemer smo upoštevali, da je levi lastni vektor y_k ortogonalen na vse desne lastne vektorje x_j, kjer je $k \neq j$. ∎

Lema 6.9 *Če je λ_i enostavna lastna vrednost, potem je $s_i \neq 0$.*

Dokaz. Brez škode za splošnost lahko privzamemo, da je $i = 1$. Naj bo sedaj $s_1 = 0$, $\|x_1\|_2 = \|y_1\|_2 = 1$ pa naj bosta ustrezni levi in desni lastni vektor. U naj bo taka unitarna matrika, da je $Ue_1 = x_1$. Potem je matrika $B = U^H A U$ oblike (glej dokaz izreka 6.2)

$$B = \begin{array}{c} \\ 1 \\ n-1 \end{array} \begin{array}{c} \begin{array}{cc} 1 & n-1 \end{array} \\ \begin{bmatrix} \lambda & \times \cdots \times \\ 0 & C \end{bmatrix} \end{array}.$$

Iz enakosti $Ax_1 = \lambda_1 x_1$, $y_1^H A = \lambda_1 y_1^H$ in $s_1 = y_1^H x_1 = 0$ dobimo

$$\begin{aligned} U^H A U e_1 &= U^H \lambda_1 x_1 = \lambda_1 e_1, \\ (y_1^H U) U^H A U &= \lambda_1 (y_1^H U), \qquad (6.1) \\ y_1^H U e_1 &= 0. \qquad (6.2) \end{aligned}$$

Iz (6.2) sledi, da je $y_1^H U$ oblike $[0\ z_1^H]$, ko pa to vstavimo v (6.1), dobimo $[0\ z_1^H]B = \lambda_1 [0\ z_1^H]$ oziroma $z_1^H C = \lambda_1 z_1^H$. Ker je λ_1 lastna vrednost matrike C, ima matrika A vsaj dvojno lastno vrednost λ_1. ∎

V primeru večkratne lastne vrednosti lahko vektorja x_i in y_i določimo tako, da bosta ortogonalna, ni pa to nujno res za poljubno izbrana lastna vektorja večkratne lastne vrednosti.

Izrek 6.10 *Naj bo $A = X\Lambda X^{-1} = Y^H \Lambda Y^{-H}$, kjer je $X = [x_1 \ \cdots \ x_n]$ matrika normiranih desnih lastnih vektorjev, $Y = [y_1 \ \cdots \ y_n]$ matrika normiranih levih lastnih vektorjev in $\Lambda = \operatorname{diag}(\lambda_1, \ldots, \lambda_n)$ diagonalna matrika lastnih vrednosti. Potem velja*

$$X^{-1} = \begin{bmatrix} \frac{1}{s_1} y_1^H \\ \vdots \\ \frac{1}{s_n} y_n^H \end{bmatrix}.$$

Dokaz. $Y^H A = \Lambda Y^H$, po drugi strani pa $X^{-1}A = \Lambda X^{-1}$. Od tod sledi, da so stolpci matrike X^{-T} levi lastni vektorji. Torej je

$$X^{-1} = \begin{bmatrix} c_1 y_1^H \\ \vdots \\ c_n y_n^H \end{bmatrix}$$

za neke konstante $c_1, \ldots, c_n$. Zaradi $XX^{-1} = I$ mora veljati $c_i = \frac{1}{s_i}$. ∎

Lema 6.11 *Matrika ne more imeti natanko ene zelo občutljive lastne vrednosti.*

Dokaz. Predpostavimo lahko, da so vse lastne vrednosti enostavne, saj v nasprotnem primeru že imamo zelo občutljiv par. Naj bo $A = X\Lambda X^{-1} = Y^H \Lambda Y^{-H}$, kjer je $X = [x_1 \ \cdots \ x_n]$ matrika normiranih desnih lastnih vektorjev, $Y = [y_1 \ \cdots \ y_n]$ matrika normiranih levih lastnih vektorjev in $\Lambda = \mathrm{diag}(\lambda_1, \ldots, \lambda_n)$ diagonalna matrika lastnih vrednosti. Po izreku 6.10 je

$$X \begin{bmatrix} \frac{1}{s_1} y_1^H \\ \vdots \\ \frac{1}{s_n} y_n^H \end{bmatrix} = I$$

oziroma

$$\sum_{i=1}^{n} \frac{x_i y_i^H}{s_i} = I.$$

Denimo, da je $|s_1|^{-1}$ največja občutljivost. Ocenimo lahko

$$\frac{1}{|s_1|} \leq 1 + \sum_{j=2}^{n} \frac{1}{|s_j|},$$

od tod pa je očitno, da mora biti vsaj ena izmed preostalih občutljivosti tudi zelo velika. ∎

Zgornjo lemo si lahko razlagamo tudi tako, da velika občutljivost lastne vrednosti pomeni, da je blizu večkratne lastne vrednosti, to pa seveda pomeni, da obstaja vsaj še ena taka lastna vrednost.

Pri občutljivosti linearnega sistema smo videli, da je recipročna vrednost občutljivosti matrike enaka oddaljenosti od najbližjega singularnega sistema. Podobno tudi sedaj velja, da je občutljivost enostavne lastne vrednosti povezana z oddaljenostjo od najbližje matrike z večkratno lastno vrednostjo. Dokaz naslednjega izreka lahko najdete npr. v [8].

Izrek 6.12 *Naj bo λ enostavna lastna vrednost matrike A z normiranima levim in desnim lastnim vektorjem y in x in naj bo $|s| < 1$, kjer je $s = y^H x$. Potem obstaja matrika $A + \delta A$ z večkratno lastno vrednostjo λ in*

$$\frac{\|\delta A\|_2}{\|A\|_2} \leq \frac{|s|}{\sqrt{1 - |s|^2}}.$$

Kaj se zgodi z občutljivostjo lastne vrednosti pri podobnostnih transformacijah? Naj bo λ enostavna lastna vrednost matrike A z normiranima levim in desnim lastnim vektorjem y in x.

Naj bo $B = S^{-1}AS$, kjer je S nesingularna matrika. Če je $\widetilde{s}$ občutljivost λ kot lastne vrednosti matrike B, potem lahko izpeljemo zvezo

$$\frac{|s|}{|\widetilde{s}|} = \frac{\|S^H y\|\|S^{-1}x\|}{\|y\|\|x\|}$$

in ocenimo

$$\frac{1}{\kappa(S)}|\widetilde{s}| \leq |s| \leq \kappa(S)|\widetilde{s}|.$$

Občutljivost lastne vrednosti se torej v najboljšem primeru zmanjša za $\kappa(S)$, v najslabšem primeru pa se za isti faktor poveča. Če je S ortogonalna matrika, potem se občutljivost ne spremeni, zato so tovrstne transformacije stabilne.

6.4 Potenčna metoda

Pri prvem algoritmu za računanje lastnih vrednosti in vektorjev ne potrebujemo drugega kot množenje z matriko A. Denimo, da izberemo normiran začetni vektor z_0 in nato za $k = 0, 1, \ldots$ generiramo vektorje po naslednjem predpisu:

$$y_{k+1} = Az_k, \qquad z_{k+1} = \frac{y_{k+1}}{\|y_{k+1}\|}. \tag{6.3}$$

Dobimo zaporedje normiranih vektorjev z_k, za katere se izkaže, da ko gre k proti neskončno, skonvergirajo proti lastnemu vektorju matrike A.

Izrek 6.13 *Naj bo λ_1 dominantna lastna vrednost matrike A, kar pomeni*

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

Potem, za splošen začetni vektor z_0, ko gre k proti neskončnosti, vektorji z_k, izračunani po predpisu (6.3), po smeri konvergirajo proti lastnemu vektorju za λ_1.

Dokaz. Izrek sicer velja za splošno matriko, dokazali pa ga bomo le za primer, ko se da matriko diagonalizirati. Naj velja $A = X\Lambda X^{-1}$, kjer je $X = [x_1 \ \cdots \ x_n]$ in $\Lambda = \mathrm{diag}(\lambda_1, \ldots, \lambda_n)$. Začetni vektor z_0 lahko razvijemo po lastnih vektorjih kot

$$z_0 = \sum_{i=1}^{n} \alpha_i x_i.$$

Potem pri pogoju $\alpha_1 \neq 0$ velja

$$z_k = \frac{A^k z_0}{\|A^k z_0\|_\infty} = \frac{\alpha_1 x_1 + \alpha_2 (\frac{\lambda_2}{\lambda_1})^k x_2 + \cdots + \alpha_n (\frac{\lambda_n}{\lambda_1})^k x_n}{\|\alpha_1 x_1 + \alpha_2 (\frac{\lambda_2}{\lambda_1})^k x_2 + \cdots + \alpha_n (\frac{\lambda_n}{\lambda_1})^k x_n\|}$$

in z_k konvergira v smeri proti x_1, ko gre k proti neskončnosti. ∎

Vidimo, da se da vektor z_k izraziti s produktom k-te potence matrike A z vektorjem z_0. Zaradi tega metodo, ki generira vektorje po predpisu (6.3), imenujemo *potenčna metoda*[5].

V zadnjem dokazu smo predpostavili:

[5] Metodo je vpeljal avstrijski matematik Richard von Mises (1883–1953) leta 1929.

a) $\alpha_1 \neq 0$,

b) A ima enostavno dominantno lastno vrednost.

Pri numeričnem računanju je predpostavka $a)$ v praksi vedno izpolnjena, saj zaokrožitvene napake povzročijo, da je $\alpha_1 \neq 0$. Če nimamo na voljo dobrega začetnega približka, je najbolj priporočljivo, da začetni vektor generiramo iz naključnih vrednosti.

Pri točki b) se da pokazati, da izrek velja tudi, ko je λ_1 večkratna lastna vrednost. Metoda se da ustrezno predelati (če si zapomnimo in hkrati gledamo zadnje tri približke) tudi za primera:

- $|\lambda_1| = |\lambda_2| > |\lambda_3| \geq \cdots$ in $\lambda_1 = -\lambda_2$,
- $|\lambda_1| = |\lambda_2| > |\lambda_3| \geq \cdots$ in $\lambda_1 = \overline{\lambda_2}$.

Vektor z_k po smeri konvergira proti lastnemu vektorju za λ_1. Zaradi tega normiranje vektorja v vsakem koraku v bistvu ni potrebno, izvajamo ga le zato, da pri numeričnem računanju ne pride do prekoračitve (v primeru $|\lambda_1| > 1$) ali podkoračitve (v primeru $|\lambda_1| < 1$).

Kako ugotovimo, kdaj je postopek že skonvergiral dovolj blizu rešitve? Ker vektor konvergira po smeri, ne pa tudi po komponentah, pogoj $\|z_{k+1} - z_k\| \leq \epsilon$ ni dober. Potrebujemo kriterij, ki nam za dani približek za lastni vektor pove, kako dober približek je to. Tega pa se ne da ugotoviti drugače kot da za približek za lastni vektor poiščemo še približek za lastno vrednost in potem skupaj pogledamo kako dober približek za lastni par imamo.

Denimo, da imamo približek x za lastni vektor in iščemo lastno vrednost. Najboljši približek je λ, ki minimizira

$$\|Ax - \lambda x\|_2,$$

rešitev pa je (uporabimo normalni sistem za predoločeni sistem $x\lambda = Ax$) *Rayleighov*[6] *kvocient*

$$\rho(x, A) = \frac{x^H Ax}{x^H x},$$

ki je definiran za $x \neq 0$.

Za Rayleighov kvocient velja $\rho(x, A) = \rho(\alpha x, A)$ za $\alpha \neq 0$. Rayleighov kvocient je torej odvisen le od smeri, ne pa tudi od norme vektorja. Očitno je tudi, da iz $Ax = \lambda x$ sledi $\rho(x, A) = \lambda$.

Pravilen zaustavitveni kriterij za potenčno metodo je, da za vsak vektor z_k izračunamo Rayleighov kvocient $\rho_k = \rho(z_k, A)$, potem pa pogledamo normo ostanka $\|Az_k - \rho_k z_k\|_2$. Če je norma dovolj majhna, je (ρ_k, z_k) dober približek za lastni par.

Vidimo, da računanje Rayleighovega kvocienta in preverjanje konvergence ne vplivata bistveno na časovno zahtevnost. Glavna operacija, ki jo v vsakem koraku algoritma izvedemo enkrat, je množenje z matriko A. Ta ima v primeru splošne matrike zahtevnost $\mathcal{O}(n^2)$, ostale operacije pa imajo zahtevnost $\mathcal{O}(n)$.

Iz dokaza izreka 6.13 sledi, da je konvergenca potenčne metode linearna. Hitrost konvergence je odvisna od razmerja $\left|\frac{\lambda_2}{\lambda_1}\right|$. Če je razmerje blizu 1, bo konvergenca počasna, blizu 0 pa hitrejša.

[6]John William Strutt (1842–1919), tretji baron Rayleigha oziroma lord Rayleigh, je bil znani angleški fizik. Leta 1904 je za raziskave plinov in odkritje argona prejel Nobelovo nagrado za fiziko. Kvocient je uporabljal pri raziskavah o termoakustiki.

Algoritem 6.1 Osnovna varianta potenčne metode. Začetni podatki so matrika A (zadošča funkcija, ki zna za dani vektor x izračunati produkt Ax), normiran vektor z_0 in toleranca ϵ.

$y_1 = Az_0, \quad \rho_0 = z_0^H y_1, \quad k = 0$
dokler $\|y_{k+1} - \rho_k z_k\|_2 \geq \epsilon$
  $k = k + 1$
  $z_k = \frac{1}{\|y_k\|_2} y_k$
  $y_{k+1} = Az_k$
  $\rho_k = z_k^H y_{k+1}$

Tako dobimo dominantno lastno vrednost λ_1. Naj bo x_1 pripadajoči normiran lastni vektor. Za ostale lastne pare lahko naredimo redukcijo:

a) *Hotellingova*[7] *redukcija* za $A = A^T$. Definiramo

$$B = A - \lambda_1 x_1 x_1^T.$$

Hitro lahko preverimo, da velja $Bx_1 = 0$ in $Bx_k = \lambda_k x_k$ za $k \neq 1$. Če uporabimo potenčno metodo na matriki B, bomo tako dobili drugo dominantno lastno vrednost matrike A

Matrike B nam ni potrebno eksplicitno izračunati. Iz enakosti $Bz = Az - \lambda_1(x_1^T z)x_1$ namreč sledi, da potrebujemo le množenje z matriko A in izračun skalarnega produkta z vektorjem x_1. Na ta način tudi za izračun naslednje lastne vrednosti še vedno zadošča, da poznamo le funkcijo, ki izračuna produkt matrike A z danim vektorjem.

b) *Householderjeva redukcija* za splošno matriko. Poiščemo unitarno matriko U, da je $Ux_1 = e_1$, pri čemer lahko seveda uporabimo Householderjeva zrcaljenja. Potem ima matrika $B = UAU^H$ obliko (glej dokaz izreka 6.2)

$$B = \begin{bmatrix} \lambda_1 & b^T \\ 0 & C \end{bmatrix}.$$

Preostale lastne vrednosti matrike A se ujemajo z lastnimi vrednostmi matrike C.

Tudi v primeru Householderjeve redukcije ekspliciten izračun matrike C ni potreben. Pri potenčni metodi moramo znati izračunati produkt Cw za vektor $w \in \mathbb{C}^{n-1}$. Pri tem si pomagamo z zvezo

$$UAU^H \begin{bmatrix} 0 \\ w \end{bmatrix} = \begin{bmatrix} \lambda_1 & b^T \\ 0 & C \end{bmatrix} \begin{bmatrix} 0 \\ w \end{bmatrix} = \begin{bmatrix} b^T w \\ Cw \end{bmatrix}.$$

Potrebujemo torej množenje z matriko A in dve množenji z ortogonalno matriko Q, ki ju lahko v primeru Householderjevega zrcaljenja izvedemo z zahtevnostjo $\mathcal{O}(n)$.

Če iščemo lastno vrednost nesingularne matrike A, ki je najmanjša po absolutni vrednosti, delamo potenčno metodo za A^{-1}, saj ima A^{-1} lastne vrednosti $\lambda^{-1}, \ldots, \lambda_n^{-1}$. V algoritmu namesto množenja $y_{k+1} = A^{-1}z_k$ rešujemo sistem $Ay_{k+1} = z_k$.

[7] Ameriški ekonomist in statistik Harold Hotelling (1895–1973) je potenčno metodo uporabljal pri kanonični korelacijski analizi.

6.5 Obratna napaka in izračunljive ocene

Denimo, da smo numerično, npr. s potenčno metodo, izračunali približek $(\widehat{\lambda}, \widehat{x})$ za lastni par matrike A. Radi bi ocenili, za koliko se $\widehat{\lambda}$ razlikuje od prave lastne vrednosti.

Definicija 6.14 Po normi relativna obratna napaka *približka* $(\widehat{\lambda}, \widehat{x})$ *za lastni par je*

$$\eta(\widehat{\lambda}, \widehat{x}) = \min\left\{\epsilon > 0 : (A + \delta A)\widehat{x} = \widehat{\lambda}\widehat{x},\ \|\delta A\|_2 \leq \epsilon \|A\|_2\right\}.$$

Če gledamo samo približek za lastno vrednost $\widehat{\lambda}$*, definiramo obratno napako kot* $\eta(\widehat{\lambda}) = \min_{\widehat{x}\neq 0} \eta(\widehat{\lambda}, \widehat{x})$*, podobno za obratno napako približka za lastni vektor vzamemo* $\eta(\widehat{x}) = \min_{\widehat{\lambda}} \eta(\widehat{\lambda}, \widehat{x})$*.*

Lema 6.15 *Velja*

1) $\eta(\widehat{\lambda}, \widehat{x}) = \dfrac{\|A\widehat{x} - \widehat{\lambda}\widehat{x}\|_2}{\|A\|_2\|\widehat{x}\|_2}$,

2) $\eta(\widehat{\lambda}) = \dfrac{\sigma_{\min}(A - \widehat{\lambda}I)}{\|A\|_2}$,

3) $\eta(\widehat{x}) = \dfrac{\|A\widehat{x} - \rho(\widehat{x}, A)\widehat{x}\|_2}{\|A\|_2\|\widehat{x}\|_2}$.

Dokaz. Za točko 1) označimo $r = A\widehat{x} - \widehat{\lambda}\widehat{x}$. Iz $(A + \delta A)\widehat{x} = \widehat{\lambda}\widehat{x}$ sledi $-\delta A\widehat{x} = r$, od tod pa $\|r\|_2 \leq \|\delta A\|_2\|\widehat{x}\|_2$.

Če vzamemo $\delta A = -r\widehat{x}^H/\|\widehat{x}\|_2^2$, potem je $(A + \delta A - \widehat{\lambda}I)\widehat{x} = 0$ in $\|\delta A\|_2 = \|r\|_2/\|\widehat{x}\|_2$, kar pomeni, da je minimum res dosežen in točka 1) drži.

Pri točki 2) upoštevamo dejstvo, da je $\min_{x\neq 0} \|Mx\|_2 = \sigma_{\min}(M)$, pri točki 3) pa lastnost Rayleighovega kvocienta, da je minimum $\|Ax - \tau x\|_2$ dosežen pri $\tau = \rho(x, A)$. ∎

Iz zgornje leme sledi, da je potenčna metoda obratno stabilna, saj vrne tak približek $(\widehat{\lambda}, \widehat{x})$ za lastni par, za katerega velja $\|A\widehat{x} - \widehat{\lambda}\widehat{x}\|_2 \leq \epsilon$.

Če bi radi ocenili, za koliko se približek $\widehat{\lambda}$ razlikuje od točne lastne vrednosti, potrebujemo poleg ocene za obratno napako še oceno za občutljivost lastne vrednosti. Za oceno potrebujemo tudi približek za levi lastni vektor. Tega bodisi izračunamo v algoritmu ali pa uporabimo npr. inverzno iteracijo, ki jo bomo spoznali v naslednjem razdelku.

Naj bosta torej $\widehat{x}$ in $\widehat{y}$ približka za desni in levi lastni vektor, ki pripadata $\widehat{\lambda}$. Potem velja ocena

$$|\widehat{\lambda} - \lambda| \leq \frac{\|r\|_2}{|\widehat{s}|},$$

kjer je

$$\widehat{s} = \frac{-\widehat{y}^H\widehat{x}}{\|\widehat{y}\|_2\|\widehat{x}\|_2}.$$

6.6 Inverzna iteracija

Rayleighov kvocient vrne najboljši približek za lastno vrednost, ki ustreza danemu vektorju. Kaj pa obratno? Denimo, da smo izračunali približek za lastno vrednost σ, sedaj pa potrebujemo še lastni vektor. Tu si lahko pomagamo z *inverzno iteracijo*[8], ki je v grobem predstavljena v algoritmu 6.2.

Algoritem 6.2 Osnovna verzija inverzne iteracije. Začetni podatki so matrika A, približek za lastno vrednost σ in normiran vektor z_0.

$k = 0, 1, \ldots$
  reši sistem $(A - \sigma I)y_{k+1} = z_k$
  $z_{k+1} = \frac{1}{\|y_{k+1}\|} y_{k+1}$

Naj za približek σ velja, da mu je najbližja lastna vrednost λ_i in velja $|\lambda_i - \sigma| \ll |\lambda_j - \sigma|$ za $j \neq i$. Inverzna iteracija v bistvu ni nič drugega kot potenčna metoda za matriko $(A - \sigma I)^{-1}$, zato jo imenujemo tudi *inverzna potenčna metoda*. Od tod vemo, da vektor z_k po smeri konvergira proti lastnemu vektorju, ki pripada dominantni lastni vrednosti matrike $(A - \sigma I)^{-1}$.

Lastne vrednosti matrike $(A - \sigma I)^{-1}$ so $(\lambda_j - \sigma)^{-1}$ za $j = 1, \ldots, n$. Ker velja $|\lambda_i - \sigma| \ll |\lambda_j - \sigma|$ za $j \neq i$, je $|(\lambda_i - \sigma)^{-1}| \gg |(\lambda_j - \sigma)^{-1}|$ za $j \neq i$. Boljši, ko je približek σ, dominantnejša je lastna vrednost $(\lambda_i - \sigma)^{-1}$ in hitrejša je konvergenca.

Inverzno iteracijo ponavadi uporabljamo zato, da dobimo lastni vektor za numerično izračunano lastno vrednost. V tem primeru je σ kar lastna vrednost matrike A, izračunana z natančnostjo $\mathcal{O}(u)$. V praksi zato potrebujemo le en do dva koraka inverzne iteracije, da iz poljubnega začetnega vektorja izračunamo pripadajoči lastni vektor.

Če je σ res lastna vrednost matrike A, potem je matrika $A - \sigma I$ singularna in v algoritmu lahko pričakujemo težave pri reševanju sistema s to matriko. Izkaže se, da nam zaokrožitvene napake pomagajo do tega, da v praksi ne pride do deljenja z nič, zato je inverzna iteracija zelo učinkovita metoda za računanje lastnih vektorjev za lastne vrednosti.

6.7 Ortogonalna iteracija

Pravimo, da je podprostor $\mathcal{N}$ *invarianten* za matriko A, če velja $A\mathcal{N} \subset \mathcal{N}$. Naj ima matrika S_1 p linearno neodvisnih stolpcev. Če jo dopolnimo do nesingularne matrike $S = [\, S_1 \quad S_2 \,]$ in je

$$B = S^{-1}AS = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix},$$

potem lahko hitro preverimo, da stolpci S_1 razpenjajo invariantni podprostor natanko takrat, ko je $B_{21} = 0$. V tem primeru so lastne vrednosti matrike A unija lastnih vrednosti matrik B_{11} in B_{22}. Če lastne vrednosti matrike A lahko uredimo po absolutni vrednosti tako, da velja $|\lambda_1| \geq \cdots \geq |\lambda_p| > |\lambda_{p+1}| \geq \cdots \geq |\lambda_n|$, potem stolpci matrike S_1 razpenjajo dominantni invariantni podprostor natanko takrat, ko so lastne vrednosti matrike B_{11} enake $\lambda_1, \ldots, \lambda_p$.

[8] Metodo je leta 1944 vpeljal nemški matematik Helmut Wielandt (1910–2001).

Če se da matriko diagonalizirati, potem bazo za dominantni invariantni podprostor dimenzije p lahko sestavimo iz p lastnih vektorjev, ki pripadajo po absolutni vrednosti p največjim lastnim vrednostim.

Za izračun baze za dominantni invariantni podprostor imamo na voljo *ortogonalno iteracijo*.

Algoritem 6.3 Osnovna verzija ortogonalne iteracije. Začetni podatki so matrika A velikosti $n \times n$ in matrika Z_0 velikosti $n \times p$, $p \leq n$, z ortonormiranimi stolpci.

$k = 0, 1, \ldots$
  $Y_{k+1} = AZ_k$
  izračunaj QR razcep $Y_{k+1} = QR$ in vzemi $Z_{k+1} = Q$

Opazimo lahko, da je pri $p = 1$ to kar potenčna metoda. Vemo, da potenčna metoda skonvergira proti dominantnemu lastnemu vektorju, linearni podprostor, ki ga razpenja ta lastni vektor, pa je očitno dominanten invarianten podprostor dimenzije 1.

Izrek 6.16 *Naj velja $A = X\Lambda X^{-1}$, kjer je $X = [x_1 \ \cdots \ x_n]$ in $\Lambda = \operatorname{diag}(\lambda_1, \ldots, \lambda_n)$. Lastne vrednosti naj bodo urejene po absolutni vrednosti in naj velja $|\lambda_p| > |\lambda_{p+1}|$. Potem, za splošno izbrano začetno matriko Z_0, matrika Z_k iz ortogonalne iteracije konvergira proti ortonormirani bazi za invariantni podprostor* $\operatorname{Lin}(\{x_1, \ldots, x_p\})$.

Dokaz. Očitno je $\operatorname{Lin}(Z_{k+1}) = \operatorname{Lin}(Y_{k+1}) = \operatorname{Lin}(AZ_k)$, od tod pa sledi $\operatorname{Lin}(Z_k) = \operatorname{Lin}(A^k Z_0)$. Ker je $A^k = X\Lambda^k X^{-1}$, velja

$$A^k Z_0 = X\Lambda^k X^{-1} Z_0 = \lambda_p^k X \begin{bmatrix} (\lambda_1/\lambda_p)^k & & & & \\ & \ddots & & & \\ & & 1 & & \\ & & & \ddots & \\ & & & & (\lambda_n/\lambda_p)^k \end{bmatrix} X^{-1} Z_0.$$

Ko gre k proti neskončno, gre $A^k Z_0$ proti $X \cdot \begin{matrix} p \\ n-p \end{matrix} \begin{pmatrix} \times \\ 0 \end{pmatrix}$, to pa pomeni, da $\operatorname{Lin}(Z_k)$ konvergira proti $\operatorname{Lin}(\{x_1, \ldots, x_p\})$. ∎

Podobno v primeru, ko je $|\lambda_r| > |\lambda_{r+1}|$, za prvih $r < p$ stolpcev velja, da $\operatorname{Lin}(Z_k(:, 1:r))$ konvergira proti $\operatorname{Lin}(\{x_1, \ldots, x_r\})$.

Vzemimo kar $p = n$ in poljubno nesingularno matriko Z_0. V tem primeru seveda ne računamo invariantnega podprostora dimenzije n, saj je to kar celotni prostor. Pri predpostavki, da so absolutne vrednosti λ_i paroma različne (to hkrati pomeni, da so vse realne), lahko pokažemo, da matrika $A_k := Z_k^T A Z_k$ konvergira proti Schurovi formi, ko gre k proti neskončno.

A_k in $Z_k^T A Z_k$ sta podobni matriki, saj je matrika Z_k ortogonalna. Naj bo $Z_k = [Z_{k1} \ Z_{k2}]$, kjer ima Z_{k1} p stolpcev. Potem je

$$Z_k^T A Z_k = \begin{bmatrix} Z_{k1}^T A Z_{k1} & Z_{k1}^T A Z_{k2} \\ Z_{k2}^T A Z_{k1} & Z_{k2}^T A Z_{k2} \end{bmatrix}.$$

Ker $\operatorname{Lin}(Z_{k1})$ konvergira proti invariantnemu podprostoru $\operatorname{Lin}(\{x_1, \ldots, x_p\})$, enako velja tudi za $\operatorname{Lin}(AZ_{k1})$, to pa pomeni, da $Z_{k2}^T A Z_{k1}$ konvergira proti 0, saj je $Z_{k2}^T Z_{k1} = 0$. Ker to velja za

vsak $p = 1, \ldots, n$, sledi, da matrika $Z_k^T A Z_k$ res konvergira proti zgornji trikotni matriki, torej proti Schurovi formi.

Poddiagonalni elementi matrike A_k konvergirajo proti 0 z linearno konvergenco, hitrost konvergence (i, j)-tega elementa, kjer je $i > j$, pa je odvisna od razmerja $|\lambda_j|/|\lambda_i|$.

6.8 QR iteracija

V prejšnjem razdelku smo videli, da z ortogonalno iteracijo lahko izračunamo Schurovo formo in s tem vse lastne vrednosti matrike. V tem razdelku pa bomo spoznali algoritem, ki zna to narediti na ekonomičnejši način. Gre za *QR iteracijo*[9], ki je trenutno najboljša numerična metoda za izračun vseh lastnih vrednosti splošne nesimetrične matrike. Osnovna verzija je zapisana v algoritmu 6.4.

Algoritem 6.4 Osnovna verzija QR iteracije. Začetni podatek je matrika A.

$A_0 = A$
$k = 0, 1, \ldots$
  $A_k = Q_k R_k$ (izračunaj QR razcep)
  $A_{k+1} = R_k Q_k$

V vsakem koraku izračunamo QR razcep matrike in faktorja v zamenjanem vrstnem redu zmnožimo v novo matriko. Iz $A_{k+1} = R_k Q_k = Q_k^T A_k Q_k$ sledi, da sta si matriki A_{k+1} in A_k ortogonalno podobni, torej je A_{k+1} ortogonalno podobna začetni matriki A in velja

$$A_{k+1} = Q_k^T \cdots Q_0^T A Q_0 \cdots Q_k.$$

Izkaže se, da je QR iteracija povezana z ortogonalno iteracijo, od tod pa sledi, da A_k konvergira proti Schurovi formi.

Lema 6.17 *Za matriko A_k iz QR iteracije velja $A_k = Z_k^T A Z_k$, kjer je Z_k matrika, ki jo dobimo pri ortogonalni iteraciji iz $Z_0 = I$. V primeru, ko imajo lastne vrednosti paroma različne absolutne vrednosti, A_k konvergira proti Schurovi formi.*

Dokaz. Uporabimo indukcijo po k. Na začetku je $A_0 = Z_0^T A Z_0$, saj je $Z_0 = I$. Denimo, da je $A_k = Z_k^T A Z_k$. Potem je

$$A_k = Z_k^T A Z_k = Z_k^T \underbrace{(\underbrace{Z_{k+1}}_{\text{ort.}} \underbrace{S_{k+1}}_{\text{zg. trik.}})}_{\text{QR razcep } AZ_k} = \underbrace{Z_k^T Z_{k+1}}_{\text{ort.}} \underbrace{S_{k+1}}_{\text{zg. trik.}} = Q_k R_k.$$

Ker je QR razcep matrike enoličen, to pomeni $S_{k+1} = R_k$ in $Z_k^T Z_{k+1} = Q_k$. Sledi

$$A_{k+1} = R_k Q_k = S_{k+1} Z_k^T Z_{k+1} = \underbrace{Z_{k+1}^T A Z_k}_{S_{k+1}} Z_k^T Z_{k+1} = Z_{k+1}^T A Z_{k+1}.$$

[9] Metodo sta neodvisno leta 1961 odkrila angleški računalnikar John G. F. Francis (r. 1934) in ruska matematičarka Vera N. Kublanovskaja (r. 1920). Francis se je leta 1962 nehal ukvarjati z numerično matematiko in se nato do leta 2007 sploh ni zavedal, kakšen vpliv ima njegov algoritem na numerično matematiko. Po oceni, ki sta jo naredila Jack Dongarra in Francis Sullivan leta 2000, spada QR iteracija med 10 algoritmov iz 20. stoletja, ki so najbolj vplivali na razvoj znanosti in tehnike [10].

V primeru, ko ima matrika A tudi kompleksne lastne vrednosti, matrika A_k skonvergira proti realni Schurovi formi.

Če pogledamo zahtevnost enega koraka QR iteracije, vidimo, da ima časovno zahtevnost $\mathcal{O}(n^3)$, saj moramo v vsakem koraku izračunati QR razcep matrike. Hitrost konvergence je odvisna od razmerja med lastnimi vrednostmi. Če se dve lastni vrednosti le malo razlikujeta, potem lahko pričakujemo, da bo metoda potrebovala veliko korakov, preden bo skonvergirala do Schurove forme.

Da pridemo do uporabne verzije QR iteracije, moramo vpeljati še nekaj izboljšav.

6.8.1 Redukcija na Hessenbergovo obliko

En korak osnovne QR iteracije porabi $\mathcal{O}(n^3)$ operacij, kar ni najbolj ekonomično. Zahtevnost enega koraka lahko močno zmanjšamo, če matriko A predhodno reduciramo na zgornjo Hessenbergovo[10] obliko.

Definicija 6.18 *Pravimo, da je matrika A* zgornja Hessenbergova, *če je $a_{ij} = 0$ za $i > j + 1$.*

Zgornja Hessenbergova matrika ima torej le zgornji trikotnik in eno poddiagonalo. Od Schurove forme jo loči le poddiagonala. Izkaže se, da se zgornja Hessenbergova oblika ohranja med QR iteracijo.

Trditev 6.19 *Če je A zgornja Hessenbergova, se oblika med QR iteracijo ohranja.*

Dokaz. Pri QR razcepu matrike $A = \begin{bmatrix} a_1 & \cdots & a_n \end{bmatrix}$ dobimo zgornjo Hessenbergovo matriko Q in zgornjo trikotno matriko R. Pri $Q = \begin{bmatrix} q_1 & \cdots & q_n \end{bmatrix}$ je oblika razvidna iz dejstva, da je q_i linearna kombinacija stolpcev $a_1, \ldots, a_i$. Hitro lahko preverimo, da je produkt zgornje trikotne in zgornje Hessenbergove matrike spet zgornja Hessenbergova matrika. ∎

Vsako realno matriko A lahko z ortogonalno podobnostno transformacijo spremenimo v zgornjo Hessenbergovo matriko. Za splošno matriko uporabimo Householderjeva zrcaljenja, če pa ima matrika A v spodnjem trikotniku že veliko ničel, so lahko Givensove rotacije še bolj učinkovite.

Zgled 6.4 *Na zgledu matrike velikosti 5×5 poglejmo, kako matriko z ortogonalnimi podobnostnimi transformacijami spremenimo v zgornjo Hessenbergovo matriko. Naj bo*

$$A = \begin{bmatrix} \times & \times & \times & \times & \times \\ (\times) & \times & \times & \times & \times \\ (\times) & \times & \times & \times & \times \\ (\times) & \times & \times & \times & \times \\ (\times) & \times & \times & \times & \times \end{bmatrix}.$$

Elementi v oklepajih označujejo elemente vektorja, ki določa Householderjevo zrcaljenje v naslednjem koraku. Zrcaljenje določimo tako, da se označeni vektor prezrcali v smer prvega enotskega vektorja.

[10] Nemški matematik Karl Adolf Hessenberg (1904–1959).

Najprej poiščemo tako ortogonalno matriko Q_1, da je

$$Q_1A = \begin{bmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \end{bmatrix}, \quad A_1 = Q_1AQ_1^T = \begin{bmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ 0 & (\times) & \times & \times & \times \\ 0 & (\times) & \times & \times & \times \\ 0 & (\times) & \times & \times & \times \end{bmatrix}.$$

Nato poiščemo ortogonalno matriko Q_2, da je

$$Q_2A_1 = \begin{bmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times \\ 0 & 0 & \times & \times & \times \end{bmatrix}, \quad A_2 = Q_2A_1Q_2^T = \begin{bmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & 0 & (\times) & \times & \times \\ 0 & 0 & (\times) & \times & \times \end{bmatrix},$$

na koncu pa še ortogonalno matriko Q_3, da je

$$Q_3A_2 = \begin{bmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times \\ 0 & 0 & 0 & \times & \times \end{bmatrix}, \quad H = Q_3A_2Q_3^T = \begin{bmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times \\ 0 & 0 & 0 & \times & \times \end{bmatrix}.$$

Tako dobimo zgornjo Hessenbergovo matriko $H = Q_3Q_2Q_1A(Q_3Q_2Q_1)^T$. □

Algoritem za splošno matriko ima naslednjo obliko.

Algoritem 6.5 Redukcija na zgornjo Hessenbergovo obliko preko Householderjevih zrcaljenj. Začetni podatek je $n \times n$ matrika A. Algoritem vrne zgornjo Hessenbergovo matriko H in po potrebi tudi ortogonalno matriko Q, da je $A = Q^THQ$.

$Q = I$ (*)
$i = 1, \ldots, n-2$
  določi $w_i \in \mathbb{R}^{n-i}$ za Householderjevo zrcaljenje P_i, ki prezrcali $A(i+1:n, i)$ v $\pm ke_1$
  $A(i+1:n, i:n) = P_iA(i+1:n, i:n)$
  $A(1:n, i+1:n) = A(1:n, i+1:n)P_i$
  $Q(i+1:n, i:n) = P_iQ(i+1:n, i:n)$ (*)

Korake označene z (*) izvedemo le v primeru, če potrebujemo tudi prehodno matriko Q.

Število operacij je $\frac{10}{3}n^3 + \mathcal{O}(n^2)$ oziroma $\frac{14}{3}n^3 + \mathcal{O}(n^2)$ če potrebujemo tudi matriko Q.

Če na začetku matriko A reduciramo na Hessenbergovo obliko, porabimo potem za en korak QR iteracije le še $\mathcal{O}(n^2)$ namesto $\mathcal{O}(n^3)$ operacij. Matrika Q_k iz QR razcepa zgornje Hessenbergove matrike A_k je namreč produkt $n-1$ Givensovih rotacij, za eno množenje matrike z Givensovo rotacijo pa vemo, da ima zahtevnost $\mathcal{O}(n)$.

Definicija 6.20 *Hessenbergova matrika H je* nerazcepna, *če so vsi njeni subdiagonalni elementi $h_{i+1,i}$ neničelni.*

Če je H razcepna, kot je npr.

$$H = \begin{bmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times \\ 0 & 0 & \times & \times & \times \\ 0 & 0 & 0 & \times & \times \end{bmatrix},$$

potem problem lastnih vrednosti razpade na dva ločena problema. Zaradi tega lahko vedno predpostavimo, da je H nerazcepna.

Pri numeričnem računanju subdiagonalni element $a_{i+1,i}^{(k)}$ matrike A_k postavimo na 0, kadar je dovolj majhen v primerjavi s sosednjima diagonalnima elementoma. To pomeni, da zadošča kriteriju

$$|a_{i+1,i}^{(k)}| < \epsilon(|a_{ii}^{(k)}| + |a_{i+1,i+1}^{(k)}|),$$

kjer je $\epsilon = \mathcal{O}(u)$ izbrana toleranca.

6.8.2 Premiki

Z redukcijo na Hessenbergovo obliko smo zmanjšali zahtevnost posameznega koraka QR iteracije, samo število potrebnih korakov pa se ni zmanjšalo, saj je hitrost konvergence odvisna od razmerja med lastnimi vrednostmi.

Konvergenco lahko pospešimo z vpeljavo premikov, kot je predstavljeno v algoritmu 6.6.

Algoritem 6.6 QR iteracija s premiki. Začetni podatek je matrika A.

$A_0 = A$
$k = 0, 1, \ldots$
  izberi premik σ_k
  $A_k - \sigma_k I = Q_k R_k$ (izračunaj QR razcep)
  $A_{k+1} = R_k Q_k + \sigma_k I$

Naslednja lema nam zagotavlja, da je tudi po vpeljavi premika matrika A_k še vedno ortogonalno podobna začetni matriki A.

Lema 6.21 *Matriki A_k in A_{k+1} pri QR iteraciji s premiki sta ortogonalno podobni.*

Dokaz.

$$A_{k+1} = R_k Q_k + \sigma_k I = Q_k^T (Q_k R_k + \sigma_k I) Q_k = Q_k^T A_k Q_k. \qquad \blacksquare$$

Za hitro konvergenco moramo za premik izbrati čim boljši približek za lastno vrednost. Če bi za premik izbrali kar lastno vrednost, potem iz naslednje leme sledi, da bi se v enem koraku QR iteracije iz matrike izločila ta lastna vrednost in bi računanje lahko nadaljevali na manjši matriki.

Lema 6.22 *Naj bo σ lastna vrednost nerazcepne zgornje Hessenbergove matrike A. Če je QR razcep $A - \sigma I = QR$ in $B = RQ + \sigma I$, potem je $b_{n,n-1} = 0$ in $b_{nn} = \sigma$.*

Dokaz. Ker je A nerazcepna, je prvih $n-1$ stolpcev matrike $A-\sigma I$ linearno neodvisnih. V razcepu $A-\sigma I = QR$ zato velja $r_{ii} \neq 0$ za $i = 1, \ldots, n-1$. Ker je $A-\sigma I$ singularna, mora biti $r_{nn} = 0$. To pomeni, da je zadnja vrstica v matriki RQ enaka 0, torej v matriki $B = RQ + \sigma I$ velja $b_{n,n-1} = 0$ in $b_{nn} = \sigma$.

Preostale lastne vrednosti lahko potem izračunamo iz matrike $B(1:n-1, 1:n-1)$. ■

Za premik potrebujemo čim boljši približek za lastno vrednost matrike A. Uporabljata se naslednji izbiri:

a) *Enojni premik*: za σ_k izberemo $a_{nn}^{(k)}$.

   Motivacija, da za premik izberemo element v spodnjem desnem kotu je, da naj bi bil to dober približek za po absolutni vrednosti najmanjšo lastno vrednost λ_n matrike A.

   Naj bo y_n levi lastni vektor za λ_n. Z malce računanja lahko pokažemo, da pri QR algoritmu brez premikov velja

   $$A^k = \widetilde{Q}_{k-1}\widetilde{R}_{k-1},$$

   kjer je $\widetilde{Q}_{k-1} = Q_0 \cdots Q_{k-1}$ in $\widetilde{R}_{k-1} = R_{k-1} \cdots R_0$. Od tod sledi $A^{-k} = \widetilde{R}_{k-1}^{-1}\widetilde{Q}_{k-1}^T$. Iz zadnje enakosti vidimo, da je zadnja vrstica matrike $\widetilde{Q}_{k-1}^T$ proporcionalna $e_n^T A^{-k}$, to je zadnji vrstici matrike A^{-k}. Razen v izjemnem in pri numeričnem računanju malo verjetnem primeru, ko e_n nima nobene komponente v smeri y_n, bo vrstica $e_n^T A^{-k}$ po smeri konvergirala proti y_n^T. Od tod sledi, da zadnji stolpec matrike $\widetilde{Q}_{k-1}$ konvergira proti levemu lastnemu vektorju y_n. Iz zveze $A_k = \widetilde{Q}_{k-1}^T A \widetilde{Q}_{k-1}$ je tako razvidno, da $a_{nn}^{(k)} = e_n^T A_k e_n = (\widetilde{Q}_{k-1}e_n)^T A \widetilde{Q}_{k-1} e_n$ konvergira proti lastni vrednosti λ_n.

   Pri tej izbiri imamo kvadratično konvergenco v bližini enostavne realne lastne vrednosti, za matrike, ki imajo tudi kompleksne lastne vrednosti, pa premik ni dober.

b) *Dvojni* oz. *Francisov premik*: vzamemo podmatriko

   $$A_k(n-1:n, n-1:n) = \begin{bmatrix} a_{n-1,n-1}^{(k)} & a_{n-1,n}^{(k)} \\ a_{n,n-1}^{(k)} & a_{nn}^{(k)} \end{bmatrix},$$

   ki ima lastni vrednosti $\sigma_1^{(k)}, \sigma_2^{(k)}$ (lahko sta tudi kompleksni). Sedaj naredimo dva premika v enem koraku:

   $$\begin{aligned} &A_k - \sigma_1^{(k)} I = Q_k R_k \text{ (izračunaj QR razcep)} \\ &A_k' = R_k Q_k + \sigma_1^{(k)} I \\ &A_k' - \sigma_2^{(k)} I = Q_k' R_k' \text{ (izračunaj QR razcep)} \\ &A_{k+1} = R_k' Q_k' + \sigma_2^{(k)} I. \end{aligned}$$

Izkaže se, da lahko en korak QR z dvojnim premikom izvedemo brez kompleksne aritmetike, saj velja:

$$\begin{aligned} Q_k Q_k' R_k' R_k &= Q_k (A_k' - \sigma_2^{(k)} I) R_k \\ &= Q_k Q_k^H (A_k - \sigma_2^{(k)} I) Q_k R_k = (A_k - \sigma_2^{(k)} I) Q_k R_k \\ &= (A_k - \sigma_2^{(k)} I)(A_k - \sigma_1^{(k)} I) \\ &= A_k^2 - (\sigma_1^{(k)} + \sigma_2^{(k)}) A_k + \sigma_1^{(k)} \sigma_2^{(k)} I =: N_k \end{aligned}$$

in $(Q_k Q_k')(R_k' R_k)$ je QR razcep realne matrike N_k. Ker po lemi 6.21 velja tudi

$$A_{k+1} = Q_k'^H A_k' Q_k' = Q_k'^H Q_k^H A_k Q_k Q_k',$$

potrebujemo le realni QR razcep realne matrike N_k.

Na prvi pogled nam zgornja ugotovitev ne pomaga dosti, saj v formuli za N_k nastopa matrika A_k^2, za izračun le te pa potrebujemo $\mathcal{O}(n^3)$ operacij. Kot bomo videli v naslednjem razdelku, pa se izkaže, da v resnici potrebujemo le prvi stolpec matrike N_k.

6.9 Implicitna QR metoda

Izrek 6.23 (Implicitni Q) *Če je $Q = [q_1 \ \cdots \ q_n]$ taka ortogonalna matrika, da je $Q^T A Q = H$ nerazcepna Hessenbergova matrika, potem so stolpci $q_2, \ldots, q_n$ do predznaka natančno določeni s q_1.*

Dokaz. Denimo, da je $V^T A V = G$, kjer je $V = [v_1 \ \cdots \ v_n]$ ortogonalna matrika, G nerazcepna zgornja Hessenbergova matrika in $q_1 = v_1$.

Potem je $W = V^T Q$ ortogonalna matrika. Če zapišemo $W = [w_1 \ \cdots \ w_n]$, potem je $w_1 = e_1$. Velja

$$GW = GV^T Q = V^T A Q = V^T Q H = WH.$$

Iz te zveze sledi $Gw_i = \sum_{j=1}^{i+1} h_{ji} w_j$ oziroma

$$h_{i+1,i} w_{i+1} = Gw_i - \sum_{j=1}^{i} h_{ji} w_j.$$

Ker je $w_1 = e_1$ in ima Gw_i en neničelni element več od w_i, sledi $w_i \in \mathrm{Lin}(\{e_1, \ldots, e_i\})$. To pomeni, da je W zgornja trikotna matrika. Ker pa je W hkrati ortogonalna, je edina možnost $W = \mathrm{diag}(1, \pm 1, \ldots, \pm 1)$, torej $v_i = \pm q_i$ za $i = 2, \ldots, n$. ∎

Posledica je, da če v QR algoritmu $A_k = Q_k R_k$, $A_{k+1} = R_k Q_k = Q_k^T A_k Q_k$, poznamo prvi stolpec matrike Q_k, potem lahko matriko A_{k+1} izračunamo brez računanja celotnega QR razcepa matrike A_k. Tako dobimo *implicitno QR iteracijo*.

Najprej poglejmo implicitno QR iteracijo z enojnim premikom. Vemo, da je prvi stolpec Q_k enak normiranemu prvemu stolpcu $A_k - \sigma_k I$. Če uspemo poiskati tako ortogonalno matriko Q_k, da bo njen prvi stolpec normiran prvi stolpec matrike $A_k - \sigma_k I$ in bo $Q_k^T A_k Q_k$ zgornja Hessenbergova matrika, potem je po izreku o implicitnem Q matrika $Q_k^T A_k Q_k$ enaka matriki iz naslednjega koraka QR metode.

Matriko Q_k poiščemo kot produkt Givensovih rotacij $Q_k = R_{12} R_{23} \cdots R_{n-1,n}$. Prva rotacija R_{12} je že določena s prvim stolpcem $A_k - \sigma_k I$, ostale pa določimo tako, da bo $Q_k^T A_k Q_k$ zgornja Hessenbergova matrika. Če je namreč

$$R_{12} = \begin{bmatrix} c_1 & s_1 & & & \\ -s_1 & c_1 & & & \\ & & 1 & & \\ & & & \ddots & \\ & & & & 1 \end{bmatrix},$$

potem je

$$Q_k = R_{12}R_{23}\cdots R_{n-1,n} = \begin{bmatrix} c_1 & \times & \cdots & \cdots & \times \\ -s_1 & \times & \cdots & \cdots & \times \\ & \times & & & \vdots \\ & & \ddots & & \vdots \\ & & & \times & \times \end{bmatrix}.$$

Algoritem lahko označimo kot *premikanje grbe*. Poglejmo si ga na primeru matrike velikosti 5×5. Po prvem koraku dobimo

$$R_{12}^T A_k R_{12} = \begin{bmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ + & \times & \times & \times & \times \\ & & \times & \times & \times \\ & & & \times & \times \end{bmatrix}.$$

Novi neničelni element, označen s $+$, je grba, ki jo z naslednjimi rotacijami pomikamo navzdol ob diagonali. Tako po vrsti poiščemo R_{23}, R_{34} in R_{45}, da je

$$R_{23}^T R_{12}^T A_k R_{12} R_{23} = \begin{bmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ & \times & \times & \times & \times \\ & + & \times & \times & \times \\ & & & \times & \times \end{bmatrix},$$

$$R_{34}^T R_{23}^T R_{12}^T A_k R_{12} R_{23} R_{34} = \begin{bmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ & \times & \times & \times & \times \\ & & \times & \times & \times \\ & & + & \times & \times \end{bmatrix}$$

in

$$R_{45}^T R_{34}^T R_{23}^T R_{12}^T A_k R_{12} R_{23} R_{34} R_{45} = \begin{bmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ & \times & \times & \times & \times \\ & & \times & \times & \times \\ & & & \times & \times \end{bmatrix}.$$

Dobimo zgornjo Hessenbergovo matriko, ki je po izreku o implicitnem Q enaka matriki A_{k+1}.

Še bolj kot pri enojnem premiku nam izrek o implicitnem Q pride prav pri dvojnem premiku. Vemo, da je $A_{k+1} = U_k^T A_k U_k$, kjer je U_k ortogonalna matrika iz QR razcepa matrike $N_k = A_k^2 - (\sigma_1^{(k)} + \sigma_2^{(k)})A_k + \sigma_1^{(k)}\sigma_2^{(k)} I$. Dovolj je poznati le prvi stolpec matrike N_k. Le ta ima obliko

$$\begin{bmatrix} a_{11}^2 + a_{12}a_{21} - sa_{11} + t \\ a_{21}(a_{11} + a_{22} - s) \\ a_{21}a_{32} \\ 0 \\ \vdots \\ 0 \end{bmatrix},$$

kjer sta

$$s = a_{n-1,n-1} + a_{nn}$$

$$t \;=\; a_{n-1,n-1}a_{nn} - a_{n-1,n}a_{n,n-1}.$$

Sedaj najprej poiščemo Householderjevo zrcaljenje oblike

$$P_1 = \begin{bmatrix} \times & \times & \times & & & \\ \times & \times & \times & & & \\ \times & \times & \times & & & \\ & & & 1 & & \\ & & & & \ddots & \\ & & & & & 1 \end{bmatrix},$$

ki ima prvi stolpec enak normiranemu prvemu stolpcu matrike N_k. Po množenju s P_1 dobimo grbo velikosti 2×2, ki jo premikamo navzdol s Householderjevimi zrcaljenji. Poglejmo si, kako to naredimo v primeru matrike velikosti 6×6.

$$P_1 A_k P_1 = \begin{bmatrix} \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times \\ + & \times & \times & \times & \times & \times \\ + & + & \times & \times & \times & \times \\ & & & \times & \times & \times \\ & & & & \times & \times \end{bmatrix},$$

$$P_2 = \begin{bmatrix} 1 & & & & & \\ & \times & \times & \times & & \\ & \times & \times & \times & & \\ & \times & \times & \times & & \\ & & & & 1 & \\ & & & & & 1 \end{bmatrix}, \quad P_2 P_1 A_k P_1 P_2 = \begin{bmatrix} \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times \\ & \times & \times & \times & \times & \times \\ & + & \times & \times & \times & \times \\ & + & + & \times & \times & \times \\ & & & & \times & \times \end{bmatrix},$$

$$P_3 = \begin{bmatrix} 1 & & & & & \\ & 1 & & & & \\ & & \times & \times & \times & \\ & & \times & \times & \times & \\ & & \times & \times & \times & \\ & & & & & 1 \end{bmatrix}, \quad P_3 P_2 P_1 A_k P_1 P_2 P_3 = \begin{bmatrix} \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times \\ & \times & \times & \times & \times & \times \\ & & \times & \times & \times & \times \\ & & + & \times & \times & \times \\ & & + & + & \times & \times \end{bmatrix},$$

$$P_4 = \begin{bmatrix} 1 & & & & & \\ & 1 & & & & \\ & & 1 & & & \\ & & & \times & \times & \times \\ & & & \times & \times & \times \\ & & & \times & \times & \times \end{bmatrix}, \quad P_4 P_3 P_2 P_1 A_k P_1 P_2 P_3 P_4 = \begin{bmatrix} \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times \\ & \times & \times & \times & \times & \times \\ & & \times & \times & \times & \times \\ & & & \times & \times & \times \\ & & & + & \times & \times \end{bmatrix},$$

$$P_5 = \begin{bmatrix} 1 & & & & & \\ & 1 & & & & \\ & & 1 & & & \\ & & & 1 & & \\ & & & & \times & \times \\ & & & & \times & \times \end{bmatrix}, \quad P_5 \cdots P_1 A_k P_1 \cdots P_5 = \begin{bmatrix} \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times \\ & \times & \times & \times & \times & \times \\ & & \times & \times & \times & \times \\ & & & \times & \times & \times \\ & & & & \times & \times \end{bmatrix}.$$

V vsakem koraku grbo velikosti 2×2 premaknemo za eno mesto navzdol, dokler je v zadnjem koraku ne izločimo iz matrike in nam ostane A_{k+1}. S pomočjo izreka o implicitnem Q lahko

tako en korak QR z dvojnim premikom izvedemo v $\mathcal{O}(n^2)$ operacijah. En korak stane $10n^2$ operacij, če računamo le A_{k+1}, in še dodatnih $10n^2$ operacij, če posodobimo še matriko Q.

QR iteracija je obratno stabilna. Za izračunano kvazi zgornjo trikotno matriko $\widehat{T}$ velja, da je ortogonalno podobna relativno malo zmoteni matriki A. To pomeni, da obstaja taka ortogonalna matrika Q, da je $A + E = Q\widehat{T}Q^T$, kjer je $\|E\| \approx \|A\|_2 u$. Podobno za izračunano matriko $\widehat{Q}$ velja, da je skoraj ortogonalna, oziroma $\|\widehat{Q}^T\widehat{Q} - I\|_2 \approx u$.

Matlab

Za izračun lastnih vrednosti in vektorjev imamo na voljo ukaz `eig`, ki uporabi implicitno QR iteracijo. Za nesimetrične matrike uporabi dvojne premike, za simetrične pa Wilkinsonove premike (glej razdelek **??**).

- `l = eig(A)`: vrne vektor z lastnimi vrednostmi matrike A.
- `[X,D] = eig(A)`: vrne matriko lastnih vektorjev X in diagonalno matriko lastnih vrednosti D, da je $AX = XD$.

Schurovo formo dobimo z naslednjimi ukazi:

- `[Q,R] = schur(A)`: za realno matriko A vrne ortogonalno matriko Q in kvazi zgornjo trikotno matriko R, da je $A = QRQ^T$. Če pa je matrika A kompleksna, potem vrne unitarno matriko U in zgornjo trikotno matriko R, da je $A = QRQ^H$.
- `[Q,R] = schur(A,'complex')`: za realno matriko A vrne vrne unitarno matriko U in zgornjo trikotno matriko R, da je $A = QRQ^H$.

Občutljivosti lastnih vrednosti dobimo z ukazom `condest`. Če uporabimo `c = condest(A)`, potem je $c_i = |s_i|^{-1}$ občutljivost i-te lastne vrednosti.

Dodatna literatura

Za računanje lastnih vrednosti in vektorjev je na voljo obsežna literatura. V slovenščini je na voljo knjiga [8]. Kar se tiče tuje literature, lahko vse algoritme, ki smo jih navedli v tem poglavju, skupaj s potrebno analizo, najdete v [9]. Uporabna sta tudi učbenika [7] in [13].

Poglavje 7

Interpolacija

7.1 Uvod

Podane imamo vrednosti funkcije f v $n+1$ paroma različnih točkah $x_0, \ldots, x_n$, iščemo pa preprostejšo *interpolacijsko funkcijo* g, ki se v točkah x_i ujema s funkcijo f, torej $g(x_i) = f(x_i)$ za $i = 0, \ldots, n$. Interpolacijsko funkcijo potrebujemo npr. takrat, kadar imamo funkcijo f tabelirano, zanima pa nas vrednost funkcije v točki x, ki v tabeli ni vsebovana. Kot približek za vrednost $f(x)$ potem vzamemo vrednost interpolacijske funkcije $g(x)$. Preprost primer za interpolacijsko funkcijo g je npr. kosoma linearna funkcija, kjer vrednost v točki $x_i \leq x \leq x_{i+1}$ dobimo s konveksno kombinacijo vrednosti funkcije f v x_i in x_{i+1}, oziroma

$$g(x) = \frac{1}{x_{i+1} - x_i}\Big((x_{i+1} - x)f(x_i) + (x - x_i)f(x_{i+1})\Big).$$

Ponavadi za interpolacijsko funkcijo izberemo polinom. Poazali bomo, da obstaja natanko en polinom stopnje n ali manj, ki se v $n+1$ paroma različnih točkah $x_0, \ldots, x_n$ ujema s funkcijo f. Tak polinom imenujemo *interpolacijski polinom*. Skoraj nikoli ne iščemo zapisa interpolacijskega polinoma v standardni bazi. Največkrat zadošča, da znamo učinkovito izračunati njegovo vrednost v dani točki.

Interpolacijo so v preteklosti uporabljali večinoma za določanje vrednosti tabeliranih funkcij, danes pa se v glavnem uporablja kot orodje pri numeričnem odvajanju, integriranju in reševanju diferencialnih enačb.

Namesto polinomov lahko, odvisno od funkcije in potreb, uporabljamo tudi druge funkcije, npr. trigonometrične polinome, racionalne funkcije in zlepke. Polinomi imajo zelo lepe lastnosti, saj sta konstrukcija in računanje vrednosti enostavni, prav tako pa jih je zelo preprosto odvajati in integrirati.

7.2 Interpolacijski polinom

Denimo, da imamo paroma različne točke $x_0, \ldots, x_n$ in vrednosti $y_0, \ldots, y_n$. Iščemo polinom I_n stopnje n ali manj, za katerega velja $I_n(x_i) = y_i$ za $i = 0, \ldots, n$.

Naivna varianta je, da polinom iščemo v standarni bazi in za koeficiente polinoma

$$p(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n$$

nastavimo linearni sistem

$$\begin{array}{ccccccccccc} a_0 & + & a_1 x_0 & + & a_2 x_0^2 & + & \cdots & + & a_n x_0^n & = & y_0 \\ a_0 & + & a_1 x_1 & + & a_2 x_1^2 & + & \cdots & + & a_n x_1^n & = & y_1 \\ & \vdots & & \vdots & & \vdots & & \vdots & & \vdots & \\ a_0 & + & a_1 x_n & + & a_2 x_n^2 & + & \cdots & + & a_n x_n^n & = & y_n. \end{array}$$

Matrika zgornjega sistema je t.i. *Vandermondova matrika*[1]

$$V(x_0, x_1, \ldots, x_n) = \begin{bmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^n \\ 1 & x_1 & x_1^2 & \cdots & x_1^n \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^n \end{bmatrix}.$$

Zanjo velja, da je v primeru paroma različnih točk nesingularna, saj se izkaže, da velja

$$\det(V(x_0, x_1, \ldots, x_n)) = \prod_{i>j} (x_i - x_j).$$

Izkaže se, da je računanje interpolacijskega polinoma preko reševanja zgornjega linearnega sistema z Vandermondovo matriko po eni strani zelo zamudno, po drugi strani pa je sistem lahko zelo slabo pogojen. Če npr. interpoliramo na intervalu $[0, 1]$ in vzamemo ekvidistantne točke $x_0 = 0, x_1 = 1/n, \ldots, x_n = 1$, potem ima Vandermondova matrika v primeru $n = 5$ pogojenostno število $4.9 \cdot 10^3$, pri $n = 10$ je $1.2 \cdot 10^8$, pri $n = 20$ pa že kar $9.7 \cdot 10^{16}$.

V resnici znamo interpolacijski polinom izračunati na ekonomičnejši način. To izkoriščajo tudi hitre metode za reševanje sistemov z Vandermondovimi matrikami, saj lahko reševanje sistemov te oblike prevedemo na iskanje koeficientov interpolacijskega polinoma.

Izrek 7.1 *Za paroma različne točke $x_0, \ldots, x_n$ in vrednosti $y_0, \ldots, y_n$ obstaja natanko en polinom I_n stopnje n ali manj, za katerega velja $I_n(x_i) = y_i$ za $i = 0, \ldots, n$.*

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

$$L_{n,i}(x) = \frac{(x - x_0) \cdots (x - x_{i-1})(x - x_{i+1}) \cdots (x - x_n)}{(x_i - x_0) \cdots (x_i - x_{i-1})(x_i - x_{i+1}) \cdots (x_i - x_n)} = \prod_{\substack{k=0 \\ k \neq i}}^{n} \frac{x - x_k}{x_i - x_k},$$

ki so stopnje n, imenujemo pa jih *Lagrangeevi*[2] *koeficienti*. Če v definicijo po vrsti vstavimo točke $x_0, \ldots, x_n$, lahko hitro preverimo, da velja

$$L_{n,i}(x_j) = \delta_{ij} := \begin{cases} 1 & i = j \\ 0 & i \neq j. \end{cases}$$

[1] Imenovana je v čast francoskega matematika Alexandra-Théophila Vandermonda (1735—1796).

[2] Italijansko-francoski matematik in astronom Joseph Louis Lagrange (1736–1813) spada med najpomembnejše znanstvenike 18. stoletja. Med drugim je znan po razvoju variacijskega računa, Lagrangeevi mehaniki in metodi variacije konstant za reševanje diferencialnih enačb.

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

$$I_n(x) = \sum_{k=0}^{n} y_k L_{n,k}(x),$$

je to očitno polinom stopnje n ali manj, prav tako pa velja $I_n(x_i) = y_i$ za $i = 0, \ldots, n$.

Pokažimo še enoličnost. Denimo, da imamo še en interpolacijski polinom $\widetilde{I}_n$ stopnje manjše ali enake n, za katerega prav tako velja $\widetilde{I}_n(x_i) = f(x_i)$ za $i = 0, \ldots, n$. Razlika $I_n - \widetilde{I}_n$ je potem tudi polinom stopnje manjše ali enake n, ki pa ima vsaj $n+1$ ničel v točkah $x_0, \ldots, x_n$. To pa je seveda možno le, če je polinom $I_n - \widetilde{I}_n$ identično enak 0 in prišli smo do protislovja. ■

Interpolacijski polinom I_n se s funkcijo f ujema v točkah $x_0, \ldots, x_n$, drugje pa ne nujno. Lahko pa ocenimo razliko, če je funkcija dovoljkrat zvezno odvedljiva. V oceni natopa polinom

$$\omega(x) = (x - x_0)(x - x_1) \cdots (x - x_n),$$

v katerem nastopajo vse interpolacijske točke.

Izrek 7.2 *Če je f $(n+1)$-krat zvezno odvedljiva funkcija na intervalu $[a, b]$, ki vsebuje vse paroma različne točke $x_0, \ldots, x_n$, potem za vsak $x \in [a, b]$ obstaja tak $\xi \in (a, b)$, da velja*

$$f(x) - I_n(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!}\omega(x) \tag{7.1}$$

in $\min(x, x_0, \ldots, x_n) < \xi < \max(x, x_0, \ldots, x_n)$.

Dokaz. Pokazati moramo, da za vsak $x \in [a, b]$ obstaja ustrezen ξ. V primeru $x = x_i$ lahko vzamemo kar poljuben ξ, saj je v enakosti (7.1) na obeh straneh vrednost 0. Zato lahko predpostavimo, da velja $x \neq x_i$ za $i = 0, \ldots, n$, in definiramo funkcijo

$$F(z) := f(z) - I_n(z) - R \cdot \omega(z). \tag{7.2}$$

Funkcija F je očitno $(n+1)$-krat zvezno odvedljiva in velja $F(x_i) = 0$ za $i = 0, \ldots, n$. Za našo točko $x \in [a, b]$, za katero smo predpostavili, da je različna od $x_0, \ldots, x_n$, lahko konstanto R določimo tako, da bo $F(x) = 0$.

Pri tako izbrani konstanti R ima funkcija F vsaj $n+2$ različnih ničel na intervalu $[a, b]$. Po Rolleovem izreku ima zato prvi odvod F' vsaj $n+1$ različnih ničel na (a, b), drugi odvod F'' ima vsaj n ničel, ..., in končno, $(n+1)$-vi odvod $F^{(n+1)}$ ima vsaj eno ničlo na (a, b), to pa je ravno točka ξ, ki jo iščemo.

Iz enačbe (7.2) sledi $0 = F^{(n+1)}(\xi) = f^{(n+1)}(\xi) - R(n+1)!$, torej

$$R = \frac{f^{(n+1)}(\xi)}{(n+1)!}$$

in za izbrani x smo poiskali tak ξ, da velja (7.1). ■

Lagrangeeva oblika ni najprimernejša za konstrukcijo in računanje vrednosti interpolacijskega polinoma. Prva težava je veliko število operacij, druga težava pa je, da moramo že na začetku

določiti stopnjo polinoma. Boljša je npr. zaporedna linearna interpolacija, pa tudi deljene diference.

Označimo z $I_{i,i+1,\ldots,i+k}$ interpolacijski polinom stopnje manjše ali enake k za funkcijo f na točkah $x_i, x_{i+1}, \ldots, x_{i+k}$. Denimo, da poznamo interpolacijska polinoma $I_{i,i+1,\ldots,i+k-1}$ in $I_{i+1,\ldots,i+k}$. Množici točk, na katerih interpolirata funkcijo f, se ujemata v točkah $x_{i+1}, \ldots, x_{i+k-1}$ in razlikujeta v točkah x_i in x_{i+k}. Hitro se lahko prepričamo, da lahko polinom stopnje kvečjemu $k+1$, ki interpolira f v vseh točkah $x_i, \ldots, x_{i+k}$, zapišemo v obliki

$$I_{i,i+1,\ldots,i+k}(x) = \frac{I_{i,i+1,\ldots,i+k-1}(x)(x - x_{i+k}) - I_{i+1,i+2,\ldots,i+k}(x)(x - x_i)}{x_i - x_{i+k}} \tag{7.3}$$

oziroma

$$I_{i,i+1,\ldots,i+k}(x) = \frac{1}{x_{i+k} - x_i} \begin{vmatrix} x - x_i & I_{i,i+1,\ldots,i+k-1}(x) \\ x - x_{i+k} & I_{i+1,i+2,\ldots,i+k}(x) \end{vmatrix}.$$

Pri *metodi zaporednih linearnih interpolacij* začnemo z interpolacijskimi polinomi - konstantami, ki interpolirajo funkcijo f le v eni izmed točk $x_0, \ldots, x_n$, potem pa z uporabo formule (7.3) povečujemo stopnje interpolacijskih polinomov, dokler ne pridemo na koncu do polinoma, ki interpolira f v vseh točkah $x_0, \ldots, x_n$. Obstaja več podobnih postopkov, opisali pa bomo *Nevilleovo*[3] *shemo*, ki se jo da preprosto implementirati v računalniku. Predstavimo jo z naslednjo trikotno shemo za primer $n = 3$:

x_i	$x - x_i$	y_i	$I_{..}(x)$	$I_{...}(x)$	$I_{....}(x)$
x_0	$x - x_0$	y_0			
			$I_{01}(x)$		
x_1	$x - x_1$	y_1		$I_{012}(x)$	
			$I_{12}(x)$		$I_{0123}(x)$
x_2	$x - x_2$	y_2		$I_{123}(x)$	
			$I_{23}(x)$		
x_3	$x - x_3$	y_3			

Zgled 7.1 *Poiskati je potrebno vrednost interpolacijskega polinoma za podatke* $x : 0, 2, 4$ *in* $f(x) : 2, 4, 8$ *v točki* $x = 1$. *Uporabimo Nevilleovo shemo:*

x_i	$x - x_i$	y_i	$I_{..}(x)$	$I_{...}(x)$
$x_0 = 0$	$x - x_0 = 1$	$y_0 = 2$		
			$I_{01}(x) = 3$	
$x_1 = 2$	$x - x_1 = -1$	$y_1 = 4$		$I_{012}(x) = \frac{11}{4}$
			$I_{12}(x) = 2$	
$x_2 = 4$	$x - x_2 = -3$	$y_2 = 8$		

[3] Angleški matematik Eric Harold Neville (1889–1961).

Do vrednosti smo prišli z naslednjimi računi:

$$\begin{aligned} I_{01}(x) &= \tfrac{1}{x_1-x_0}\begin{vmatrix} x-x_0 & I_0(x) \\ x-x_1 & I_1(x) \end{vmatrix} = \tfrac{1}{2}\begin{vmatrix} 1 & 2 \\ -1 & 4 \end{vmatrix} = 3, \\ I_{12}(x) &= \tfrac{1}{x_2-x_1}\begin{vmatrix} x-x_1 & I_1(x) \\ x-x_2 & I_2(x) \end{vmatrix} = \tfrac{1}{2}\begin{vmatrix} -1 & 4 \\ -3 & 8 \end{vmatrix} = 2, \\ I_{012}(x) &= \tfrac{1}{x_2-x_0}\begin{vmatrix} x-x_0 & I_{01}(x) \\ x-x_2 & I_{12}(x) \end{vmatrix} = \tfrac{1}{4}\begin{vmatrix} 1 & 3 \\ -3 & 2 \end{vmatrix} = \tfrac{11}{4}. \end{aligned}$$ □

7.3 Deljene diference

Še enostavnejši zapis interpolacijskega polinoma nam omogočajo deljene diference. Kot bomo videli v nadaljevanju, lahko z njimi posplošimo interpolacijo tudi na primere, ko interpolacijske točke niso paroma različne. Večkratna točka potem pomeni, da se polinom in funkcija ujemata ne samo v vrednosti, temveč še v ustreznem številu odvodov. Za začetek definirajmo deljeno diferenco na paroma različnih točkah.

Definicija 7.3 *Deljena diferenca $f[x_0, x_1, \ldots, x_k]$ je vodilni koeficient (pri x^k) interpolacijskega polinoma stopnje k, ki se ujema s funkcijo f v paroma različnih točkah $x_0, x_1, \ldots, x_k$.*

Izrek 7.4 *Za deljene diference velja:*

a)

$$P_n(x) = f[x_0] + f[x_0, x_1](x - x_0) + \cdots + f[x_0, x_1, \ldots, x_n](x - x_0)\cdots(x - x_{n-1}) \tag{7.4}$$

je interpolacijski polinom, ki se v točkah $x_0, \ldots, x_n$ ujema s funkcijo f.

b) $f[x_0, x_1, \ldots, x_k]$ je simetrična funkcija svojih argumentov.

c) $(\alpha f + \beta g)[x_0, \ldots, x_k] = \alpha f[x_0, \ldots, x_k] + \beta g[x_0, \ldots, x_k]$.

d) Velja rekurzivna formula

$$f[x_0, \ldots, x_k] = \frac{f[x_1, \ldots, x_k] - f[x_0, \ldots, x_{k-1}]}{x_k - x_0}. \tag{7.5}$$

Dokaz.

a) Uporabimo indukcijo. Pri $n = 0$ se očitno $f[x_0]$ ujema z $f(x_0)$. Naj bo sedaj

$$P_i(x) = f[x_0] + f[x_0, x_1](x - x_0) + \cdots + f[x_0, x_1, \ldots, x_i](x - x_0)\cdots(x - x_{i-1})$$

interpolacijski polinom na točkah $x_0, \ldots, x_i$. Hitro se lahko prepričamo, da obstaja taka konstanta c, da lahko polinom, ki interpolira funkcijo f v točkah $x_0, \ldots, x_{i+1}$ zapišemo v obliki

$$P_{i+1}(x) = P_i(x) + c(x - x_0)\cdots(x - x_i).$$

Za $k = 0, \ldots, i$ namreč neodvisno od c velja $P_{i+1}(x_k) = P_i(x_k) = f(x_k)$. Od tod sledi, da lahko c določimo tako, da bo $P_{i+1}(x_{i+1}) = f(x_{i+1})$. Ker pa je c vodilni koeficient polinoma P_{i+1}, je po definiciji deljene diference $c = f[x_0, \ldots, x_{i+1}]$.

b), c) Očitno.

d) Naj bo p_0 interpolacijski polinom, ki se ujema z f v točkah $x_0, \ldots, x_{k-1}$ in p_1 polinom, ki se ujema z f v točkah $x_1, \ldots, x_k$. Interpolacijski polinom p, ki se z f ujema v vseh točkah $x_0, \ldots, x_k$, ima po formuli (7.3) obliko

$$p(x) = \frac{x - x_k}{x_0 - x_k} p_0(x) + \frac{x - x_0}{x_k - x_0} p_1(x).$$

S primerjanjem vodilnih koeficientov polinomov p, p_0 in p_1 pridemo do zveze (7.5). ■

Obliko (7.4) imenujemo *Newtonov interpolacijski polinom*.

Za deljeno diferenco v eni točki smo že ugotovili, da velja $f[x_0] = f(x_0)$. Za dve točki dobimo po formuli (7.5) deljeno diferenco

$$f[x_0, x_1] = \frac{f(x_1) - f(x_0)}{x_1 - x_0},$$

kar se ujema s smernim koeficientom premice, ki gre skozi točki $(x_0, f(x_0))$ in $(x_, f(x_1))$.

Kaj se zgodi, če vzamemo interpolacijski polinom na dveh točkah in v limiti pošljemo eno točko proti drugi? Iz sekante

$$p(x) = f[x_0] + f[x_0, x_1](x - x_0) = f(x_0) + \frac{f(x_1) - f(x_0)}{x_1 - x_0}(x - x_0)$$

v limiti, ko gre x_1 proti x_0, dobimo tangento

$$p(x) = f(x_0) + f'(x_0)(x - x_0).$$

To pomeni, da lahko definiciji interpolacijskega polinoma in deljene diference razširimo na primere, ko se točke ponavljajo. Mislimo si, da so bile na začetku vse točke paroma različne, potem pa jih v limiti po potrebi združimo. Večkratnost interpolacijske točke pomeni, v koliko odvodih naj se interpolacijski polinom ujema s funkcijo. Tako npr. pri točkah $x_1, x_2, x_2, x_2, x_3, x_3$ iščemo polinom p stopnje manjše ali enake 5, za katerega velja $p(x_1) = f(x_1)$, $p(x_2) = f(x_2)$, $p'(x_2) = f'(x_2)$, $p''(x_2) = f''(x_2)$, $p(x_3) = f(x_3)$ in $p'(x_3) = f'(x_3)$.

Če dopuščamo tudi ponavljanje točk, potem za deljene diference velja rekurzivna zveza

$$f[x_0, x_1, \ldots, x_k] = \begin{cases} \dfrac{f^k(x_0)}{k!}, & x_0 = x_1 = \cdots = x_k, \\ \dfrac{f[x_1, \ldots, x_k] - f[x_0, \ldots, x_{k-1}]}{x_k - x_0}, & \text{sicer.} \end{cases}$$

Če je funkcija f v točki x_0 k-krat zvezno odvedljiva, potem iz interpolacijskega polinoma

$$p(x) = f[x_0] + f[x_0, x_1](x - x_0) + \cdots + f[x_0, x_1, \ldots, x_k](x - x_0) \cdots (x - x_{k-1})$$

v limiti, ko pošljemo vse točke proti x_0, dobimo Taylorjev polinom razvoja f okoli točke x_0:

$$T_k(x) = f(x_0) + \frac{f'(x_0)}{1!}(x - x_0) + \cdots + \frac{f^{(k)}(x_0)}{k!}(x - x_0)^k.$$

Deljene diference računamo v trikotni shemi iz katere na koncu lahko hitro preberemo enačbo polinoma oz. izračunamo njegovo vrednost v iskani točki. Konstrukcija je razvidna iz naslednjega zgleda.

Zgled 7.2 *Zapiši enačbo polinoma stopnje kvečjemu 5, za katerega velja:* $p(0) = 1$, $p'(0) = 2$, $p''(0) = 3$, $p(1) = -1$, $p'(1) = 3$ *in* $p(2) = 4$.

x_i	$f[.]$	$f[.,.]$	$f[.,.,.]$	$f[.,.,.,.]$	$f[.,.,.,.,.]$	$f[.,.,.,.,.,.]$
0	1					
		2				
0	1		$\frac{3}{2}$			
		2		$-\frac{11}{2}$		
0	1		-4		$\frac{29}{2}$	
		-2		9		$-\frac{79}{8}$
1	-1		5		$-\frac{21}{4}$	
		3		$-\frac{3}{2}$		
1	-1		2			
		5				
2	4					

Vrednosti v tabeli, ki so označene z rdečo, dobimo direktno iz začetnih podatkih, saj so v teh deljenih diferencah vse točke enake. Ostale vrednosti izračunamo po rekurzivni zvezi. Tako npr. vrednost $f[0,0,0] = 3/2$ *dobimo iz formule* $f[0,0,0] = f''(0)/2$, *vrednost* $f[0,0,0,1] = 29/2$ *pa izračunamo iz* $f[0,0,0,1] = (f[0,0,1] - f[0,0,0])/(1-0)$.

Koeficiente interpolacijskega polinoma preberemo iz zgornje diagonale in dobimo

$$p(x) = 1 + 2x + \frac{3}{2}x^2 - \frac{11}{2}x^3 + \frac{29}{2}x^3(x-1) - \frac{79}{8}x^3(x-1)^2.$$

Namesto tega bi lahko koeficiente vzeli tudi iz spodnje diagonale. V tem primeru interpolacijske točke nastopajo v obratnem vrstnem redu. Interpolacijski polinom je seveda enak kot prej, saj je enoličen, le zapisan je v drugačni bazi. Če vzamemo spodnjo diagonalo, dobimo

$$p(x) = 4 + 5(x-2) + 2(x-2)(x-1) - \frac{3}{2}(x-2)(x-1)^2$$
$$-\frac{21}{4}(x-2)(x-1)x - \frac{79}{8}(x-2)(x-1)^2x^2. \qquad \square$$

Izrek 7.5 (Hermite–Genocchijeva formula) [4] *Za k-krat zvezno odvedljivo funkcijo f velja*

$$f[x_0,\ldots,x_k] = \int_0^1 dt_1 \int_0^{t_1} dt_2 \int \cdots \int_0^{t_{k-1}} f^{(k)}\Big(t_k(x_k - x_{k-1}) + \cdots + t_1(x_1 - x_0) + x_0\Big)dt_k. \quad (7.6)$$

Dokaz. V primeru, ko so vse točke enake, lahko integral (7.6) direktno izračunamo in je enak

$$\int_0^1 dt_1 \int_0^{t_1} dt_2 \int \cdots \int_0^{t_{k-1}} f^{(k)}(x_0)dt_k = f^{(k)}(x_0)\int_0^1 dt_1 \int_0^{t_1} dt_2 \int \cdots \int_0^{t_{k-1}} dt_k = \frac{1}{k!}f^{(k)}(x_0).$$

V glavnem delu dokaza lahko tako predpostavimo, da se vsaj dve izmed točk $x_0,\ldots,x_k$ razlikujeta. Veljavnost formule preverimo z indukcijo. V primeru $k = 1$, pri predpostavki $x_0 \neq x_1$, res dobimo

$$\int_0^1 f'(t_1(x_1 - x_0) + x_0)dt_1 = \frac{1}{x_1 - x_0}f(t_1(x_1 - x_0) + x_0)\Big|_{t_1=0}^{t_1=1} = \frac{f(x_1) - f(x_0)}{x_1 - x_0} = f[x_0, x_1].$$

[4] Italijanski matematik Angelo Genocchi (1817–1889).

Sedaj predpostavimo, da izrek velja, če imamo k ali manj točk in ga poskusimo dokazati za primer $k+1$ točk. Če definiramo novo spremenljivko $\xi = t_k(x_k - x_{k-1}) + \cdots + t_1(x_1 - x_0) + x_0$, potem lahko zapišemo integral (7.6) kot

$$\int_0^1 dt_1 \int_0^{t_1} dt_2 \int \cdots \int_0^{t_{k-1}} f^{(k)}(\xi) dt_k.$$

S substitucijo spremenljivk dobimo

$$d\xi = (x_k - x_{k-1}) dt_k$$

in

$$\int_0^{t_{k-1}} f^{(k)}(\xi) dt_k = \frac{1}{x_k - x_{k-1}} \int_{\xi_0}^{\xi_1} f^{(k)}(\xi) d\xi = \frac{f^{(k-1)}(\xi_1) - f^{(k-1)}(\xi_0)}{x_k - x_{k-1}},$$

kjer sta meji enaki (ξ_0 ustreza $t_k = 0$, ξ_1 pa $t_k = t_{k-1}$)

$$\begin{aligned} \xi_0 &= t_{k-1}(x_{k-1} - x_{k-2}) + t_{k-2}(x_{k-2} - x_{k-3}) + \cdots + t_1(x_1 - x_0) + x_0, \\ \xi_1 &= t_{k-1}(x_k - x_{k-2}) + t_{k-2}(x_{k-2} - x_{k-3}) + \cdots + t_1(x_1 - x_0) + x_0. \end{aligned}$$

Po indukcijski predpostavki sledi

$$\int_0^1 dt_1 \int_0^{t_1} dt_2 \int \cdots \int_0^{t_{k-2}} f^{(k-1)}(\xi_1) dt_{k-1} = f[x_0, x_1, \ldots, x_{k-2}, x_k]$$

in

$$\int_0^1 dt_1 \int_0^{t_1} dt_2 \int \cdots \int_0^{t_{k-2}} f^{(k-1)}(\xi_0) dt_{k-1} = f[x_0, x_1, \ldots, x_{k-2}, x_{k-1}],$$

ker pa po rekurzivni zvezi (7.5) velja

$$f[x_0, x_1, \ldots, x_k] = \frac{f[x_0, x_1, \ldots, x_{k-2}, x_k] - f[x_0, x_1, \ldots, x_{k-2}, x_{k-1}]}{x_k - x_{k-1}},$$

je dokaz končan. ■

Naslednjo zvezo dobimo tako, da za integral (7.7) uporabimo izrek o povprečni vrednosti.

Posledica 7.6 *Za k-krat zvezno odvedljivo funkcijo f velja*

$$f[x_0, \ldots, x_k] = \frac{1}{k!} f^{(k)}(\xi),$$

kjer je

$$\min_{i=0,\ldots,k}(x_i) \leq \xi \leq \max_{i=0,\ldots,k}(x_i).$$

Izrek 7.7 *Za $(n+1)$-krat zvezno odvedljivo funkcijo f in interpolacijski polinom I_n na točkah $x_0, \ldots, x_n$ velja*

$$f(x) = I_n(x) + f[x_0, \ldots, x_n, x](x - x_0) \cdots (x - x_n). \tag{7.7}$$

Dokaz. Naj polinom q interpolira funkcijo f v točkah $x_0, \ldots, x_n, t$. V Newtonovi obliki lahko q zapišemo kot

$$q(x) = I_n(x) + f[x_0, \ldots, x_n, t](x - x_0) \cdots (x - x_n).$$

Očitno pri izbranem t potem velja $q(t) = I_n(t) + f[x_0, \ldots, x_n, t](t - x_0) \cdots (t - x_n)$. Ker to velja za vsak t, dobimo formulo (7.7), ki jo je bilo potrebno dokazati. ■

Posledica 7.8 *Za $(n+1)$-krat zvezno odvedljivo funkcijo f in interpolacijski polinom I_n na točkah $x_0, \ldots, x_n$ velja*

$$f(x) - I_n(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!}\omega(x),$$

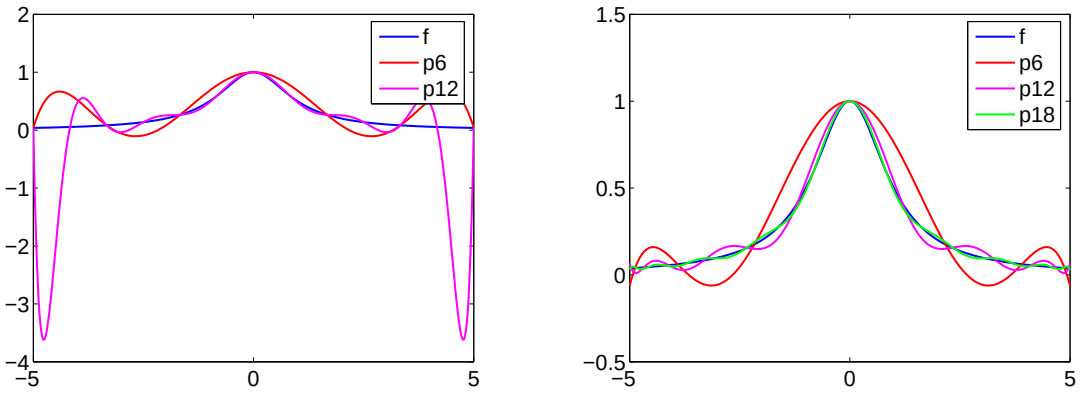
kjer je $\min(x, x_0, \ldots, x_n) \leq \xi \leq \max(x, x_0, \ldots, x_n)$. ■

Zgornja ocena se ujema z oceno, ki smo jo zapisali v izreku 7.2 za paroma različne točke. Prednost nove ocene je, da pride v poštev tudi, kadar vse točke niso medsebojno različne.

7.4 Interpolacija s kosoma polinomskimi funkcijami

Če želimo, da bi se na danem intervalu interpolacijski polinom čim bolj prilegal izbrani funkciji, je prva ideja ta, da povečamo stopnjo interpolacijskega polinoma. To naredimo tako, da povečamo število interpolacijskih točk. Kot kaže naslednji zgled, ki je znan pod imenom *Rungejev*[5] *protiprimer*, ni nujno, da večanje stopnje interpolacijskega polinoma res izboljša aproksimacijo funkcije s polinomom.

Zgled 7.3 *Denimo, da interpoliramo funkcijo $f(x) = 1/(1+x^2)$ na intervalu $[-5, 5]$. Če uporabimo ekvidistantne točke, potem z večanjem stopnje interpolacijski polinom vedno slabše aproksimira f, kar kaže slika 7.1 (leva slika).*



Slika 7.1: Rungejev protiprimer. Če funkcijo $f(x) = 1/(1+x^2)$ na intervalu $[-5, 5]$ interpoliramo v ekvidistantnih točkah (levo), razlika med funkcijo in interpolacijskim polinomom z večanjem n narašča. Če pa interpoliramo na Čebiševih točkah (desno), potem z večanjem števila točk interpolacijski polinom vedno bolje aproksimira f.

Težava se v tem primeru pojavi zaradi ekvidistantnih točk. Če za interpolacijske točke vzamemo Čebiševe točke, ki so v tem primeru definirane kot $x_j = 5\cos(j\pi/n)$ za $j = 0, \ldots, n$, potem z večanjem stopnje interpolacijski polinom vedno bolje aproksimira funkcijo f, kar se vidi na sliki 7.1 (desna slika). □

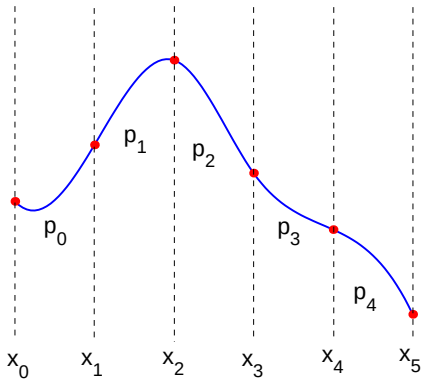
[5]Nemški matematik Carl David Tolmé Runge (1856—1927) je znan predvsem po Runge–Kutta metodi za numerično reševanje diferencialnih enačb.

Tu je potrebno še omeniti, da za razliko med interpolacijo z ekvidistantnimi in Čebiševimi točkami v zadnjem zgledu niso odgovorne zaokrožitvene napake oz. numerično računanje nasploh, saj pride do razlik pri eksaktnem računanju.

Ponavadi se Čebiševe točke sicer res obnašajo bolje kot ekvidistantne točke, a je dokazano, da za vsako zaporedje predpisanih interpolacijskih točk obstaja funkcija, pri kateri se zgodi, da z večanjem stopnje interpolacijskega polinoma prihaja do vse večjih razlik med to funkcijo in interpolacijskim polinomom.

To ni v nasprotju z Weierstrassovim izrekom iz prejšnjega poglavja. Ta pravi, da je možno zvezno funkcijo poljubno dobro aproksimirati s polinomom dovolj visoke stopnje, a pri tem ni predpisano, v katerih točkah se mora polinom ujemati s funkcijo. Če za te točke predpišemo, da morajo biti npr. ekvidistantne, potem izreka seveda ne moremo več uporabiti.

Rungejev protiprimer kaže, da večanje stopnje interpolacijskega polinoma ni nujno dober način za boljšo aproksimacijo funkcije. Rešitev je, da sicer povečamo število interpolacijskih točk, a izberemo drugačno vrsto interpolacijske funkcije. Namesto enega polinoma visoke stopnje, interpolacijsko funkcijo sestavimo (zlepimo) iz polinomov nizkih stopenj. Tako dobimo kosoma polinomsko funkcijo oziroma *zlepek*. Ker imamo še vedno opravka s polinomi, sta konstrukcija zlepka in izračun vrednosti še vedno preprosta. Primer zlepka je prikazan na sliki 7.2.



Slika 7.2: Zgled polinomskega zlepka. Na vsakem intervalu $[x_i, x_{i+1}]$ imamo definiran polinom p_i za $i = 0, \ldots, 4$, skupaj pa sestavljajo zvezno funkcijo na intervalu $[x_0, x_5]$.

Denimo, da iščemo interpolacijsko funkcijo, ki bo v točkah $x_0, \ldots, x_{n+1}$ po vrsti imela vrednosti $y_0, \ldots, y_{n+1}$. Pri polinomskem zlepku interval $[x_0, x_{n+1}]$ razdelimo na podintervale $[x_i, x_{i+1}]$ za $i = 0, \ldots, n$. Na vsakem intervalu $[x_i, x_{i+1}]$ vzamemo polinom p_i nizke stopnje, za katerega velja $p_i(x_i) = y_i$ in $p_i(x_{i+1}) = y_{i+1}$. Želimo, da bo sestavljena krivulja čim bolj gladka in da bo čim bolje opisovala obliko začetnih podatkov. Ker gre krivulja čez točke (x_i, y_i) za $i = 1, \ldots, n$, je očitno najmanj zvezna, saj v notranjih točkah velja

$$p_i(x_{i+1}) = p_{i+1}(x_{i+1}) = y_{i+1}.$$

Osnovna varianta je *kosoma linearna interpolacija*, kjer na vsakem intervalu $[x_i, x_{i+1}]$ podatke interpoliramo z linearno funkcijo

$$p_i(x) = y_i + \frac{y_{i+1} - y_i}{x_{i+1} - x_i}(x - x_i).$$

Dobljena kosoma linearna funkcija je očitno zvezna, ni pa zvezno odvedljiva.

Če želimo, da bi bil zlepek tudi zvezno odvedljiv, moramo povečati stopnjo polinomov. Naslednji na vrsti so kvadratni zlepki, kjer sicer lahko dosežemo, da bo zlepek zvezno odvedljiv, a so preobčutljivi na motnje v začetnih podatkih. Izkaže se, da so najbolj primerni naslednji na vrsti, torej *kubični zlepki*, kjer je p_i kubičen polinom na $[x_i, x_{i+1}]$. Za kubični interpolacijski zlepek ponavadi zahtevamo še, da je vsaj zvezno odvedljiv.

Posamezni kubični polinom p_i je natanko določen z vrednostmi in odvodi v krajiščih:

$$p_i(x_i) = y_i, \qquad p_i(x_{i+1}) = y_{i+1}$$

$$p_i'(x_i) = d_i, \qquad p_i'(x_{i+1}) = d_{i+1}.$$

Če označimo $h_i = x_{i+1} - x_i$, $\delta_i = (y_{i+1} - y_i)/h_i$, in uporabimo deljene diference, dobimo

$$\begin{array}{c|cccc}
x_i & y_i & & & \\
 & & d_i & & \\
x_i & y_i & & \frac{\delta_i - d_i}{h_i} & \\
 & & \delta_i & & \frac{d_i + d_{i+1} - 2\delta_i}{h_i^2} \\
 & & & \frac{d_{i+1} - \delta_i}{h_i} & \\
x_{i+1} & y_{i+1} & & & \\
 & & d_{i+1} & & \\
x_{i+1} & y_{i+1} & & &
\end{array}$$

in lahko zapišemo polinom p_i v obliki

$$p_i(x) = y_i + d_i(x - x_i) + \frac{\delta_i - d_i}{h_i}(x - x_i)^2 + \frac{d_i + d_{i+1} - 2\delta_i}{h_i^2}(x - x_i)^2(x - x_{i+1}).$$

Kubični zlepek je tako določen s parametri $d_0, \ldots, d_{n+1}$, ki so zaenkrat še prosti, določiti pa jih moramo tako, da bo imel zlepek željene lastnosti. Obstaja veliko možnih izbir, najpogostejše pa so:

- Pri *Hermiteovem kubičnem zlepku* izberemo kar $d_i = y_i'$, kjer je y_i' odvod funkcije, ki jo interpoliramo, v točki x_i za $i = 0, \ldots, n+1$. Na ta način se zlepek in funkcija v interpolacijskih točkah ujemata ne samo v vrednostih temveč tudi v prvih odvodih.
- Pri *dvakrat zvezno odvedljivem kubičnem zlepku* parametre $d_0, \ldots, d_{n+1}$ določimo tako, da je zlepek dvakrat zvezno odvedljiv. S kratkim računom lahko preverimo, da za druge odvode v interpolacijskih točkah velja

  $$p_i''(x_{i+1}) = \frac{1}{h_i}(2d_i + 4d_{i+1} - 6\delta_i)$$

  in

  $$p_{i+1}''(x_{i+1}) = \frac{1}{h_{i+1}}(-4d_{i+1} - 2d_{i+2} + 6\delta_{i+1}).$$

  Da bo zlepek dvakrat zvezno odvedljiv mora torej veljati

  $$p_i''(x_{i+1}) = p_{i+1}''(x_{i+1})$$

za $i = 0, \ldots, n-1$. Tako dobimo sistem n linearnih enačb za $n+2$ parametrov:

$$h_{i+1}d_i + 2(h_i + h_{i+1})d_{i+1} + h_i d_{i+2} = 3h_{i+1}\delta_i + 3h_i\delta_i.$$

Enačb je manj kot parametrov, kar pomeni, da imamo v splošni rešitvi še dve prostostni stopnji. Najbolj pogoste rešitve, kako izbrati še dve manjkajoči enačbi, da dobimo enolično rešitev, so:

- *Kompletni zlepek*: če poznamo vrednosti odvodov funkcije, ki jo interpoliramo, v začetni in končni točki, potem lahko vzamemo $d_0 = y'_0$ in $d_{n+1} = y'_{n+1}$.
- *Naravni zlepek*: s pogojema $p''_0(x_0) = 0$ in $p''_n(x_{n+1}) = 0$ dosežemo, da ima zlepek v začetni in končni točki prevoj.
- *Zlepek brez vozlov*: zahtevamo, da je zlepek na $[x_0, x_2]$ in $[x_{n-1}, x_{n+1}]$ kubični polinom oziroma, da velja $p'''_0(x_1) = p'''_1(x_1)$ in $p'''_{n-1}(x_n) = p'''_n(x_n)$. Tako se v bistvu znebimo vozlov x_1 in x_n.

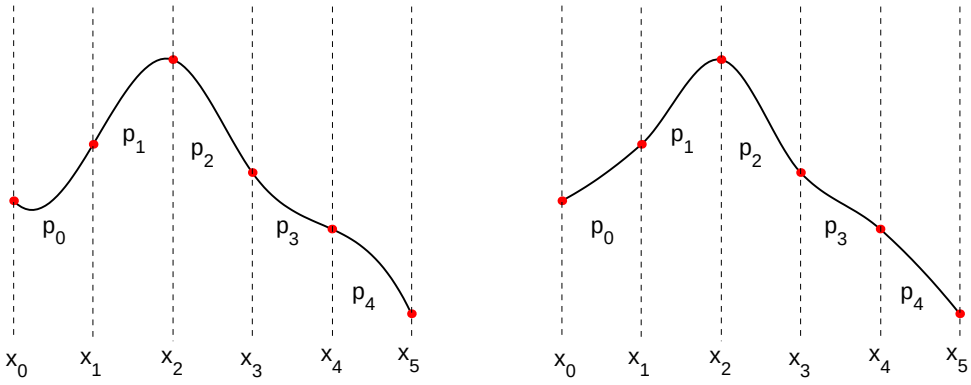
V vseh treh primerih moramo na koncu rešiti linearni sistem, iz katerega dobimo vrednosti parametrov $d_0, \ldots, d_n$. Ker je sistem tridiagonalen, lahko to naredimo zelo ekonomično v časovni zahtevnosti $\mathcal{O}(n)$.

- Pri *kubičnem zlepku, ki ohranja obliko*, naklone d_i določimo tako, da za $1/d_i$ vzamemo povprečje recipročnih vrednosti smernih koeficientov premic skozi (x_{i-1}, x_i) in (x_i, x_{i+1}). Če sta smerna koeficienta δ_i in δ_{i+1} enako predznačena, potem pri predpostavki $h_i = h_{i+1}$ dobimo

$$\frac{1}{d_i} = \frac{1}{2}\left(\frac{1}{\delta_i} + \frac{1}{\delta_{i+1}}\right).$$

Če sta širini intervalov $[x_{i-1}, x_i]$ in $[x_i, x_{i+1}]$ različni, je potrebno formulo za d_i popraviti. V primeru, ko sta smerna koeficienta δ_i in δ_{i+1} nasprotno predznačena, vzamemo $d_i = 0$.

Dobljeni kubični zlepek je le enkrat zvezno odvedljiv, preskok v drugem odvodu je lepo razviden na sliki 7.3. Na levi strani je dvakrat zvezno odvedljiv (naravni) zlepek, na desni strani pa zlepek, ki ohranja obliko. Ohranjanje oblike pomeni, da na vsakem intervalu $[x_i, x_{i+1}]$ vrednosti zlepka ležijo med $f(x_i)$ in $f(x_{i+1})$. Da to pri dvakrat zvezno odvedljivem zlepku ni nujno res, se vidi na sliki 7.3 na intervalu $[x_0, x_1]$.



Slika 7.3: Primerjava kubičnega zlepka, ki ohranja obliko (desno), z dvakrat zvezno odvedljivim naravnim kubičnim zlepkom (levo).

Pokažimo, da v primeru Hermiteovega kubičnega zlepka res velja, da, z večanjem števila interpolacijskih točk, zlepek vedno bolj aproksimira funkcijo, ki jo interpolira. Pri večanju števila točk moramo seveda paziti na to, da se manjša maksimalna širina podintervalov

$$h := \max_{i=0,\ldots,n} h_i.$$

Izrek 7.9 *Naj bo f štirikrat zvezno odvedljiva funkcija na intervalu $[a,b]$. Če interval razdelimo s točkami $a = x_0 < x_1 < \ldots < x_n < x_{n+1} = b$, potem za Hermiteov kubični zlepek p za vsak $x \in [a,b]$ velja ocena*

$$|f(x) - p(x)| \leq \frac{h^4}{384} M_4, \tag{7.8}$$

kjer je $h = \max_{i=0,\ldots,n}(x_{i+1} - x_i)$ in $M_4 = \max_{x\in[a,b]} |f^{(4)}(x)|$.

Dokaz. Na podintervalu $[x_i, x_{i+1}]$ funkcijo f interpoliramo s kubičnim polinomom p_i, za katerega velja $p_i(x_i) = f(x_i)$, $p_i'(x_i) = f(x_i)$, $p_i(x_{i+1}) = f(x_{i+1})$ in $p_i(x_{i+1}) = f(x_{i+1})$. Za razliko vemo, da velja

$$f(x) - p_i(x) = f[x_i, x_i, x_{i+1}, x_{i+1}, x](x - x_i)^2(x - x_{i+1})^2.$$

Če je $h_i = x_{i+1} - x_i$, lahko ocenimo

$$|f(x) - p_i(x)| \leq \frac{M_4}{4!} \cdot \frac{h_i^4}{16}.$$

Od tod sledi ocena (7.8). ∎

Podobne ocene se da izpeljati tudi za ostale zlepke. Tako npr. v [18] lahko najdete izpeljavo, da za kompletni kubični zlepek p in štirikrat zvezno odvedljivo funkcijo f pri predpostavkah izreka 7.9 veljajo ocene

$$\begin{aligned} |f(x) - p(x)| &\leq \frac{5h^4}{384} M_4, \\ |f'(x) - p'(x)| &\leq \frac{h^3}{24} M_4, \\ |f''(x) - p''(x)| &\leq \frac{3h^2}{8} M_4. \end{aligned}$$

Ne samo, da kompletni kubični zlepek dobro aproksimira funkcijo f, iz zadnjih dveh ocen sledi, da tudi prvi in drugi odvod zlepka dobro aproksimirata prvi in drugi odvod funkcije f.

7.5 Beziérove krivulje

Pri interpolaciji v ravnini imamo dane točke (x_i, y_i) za $i = 0, \ldots, n$, kjer x in y koordinate niso nujno paroma različne oziroma monotono urejene, iščemo pa krivuljo, ki bi šla skozi te točke v navedenem vrstnem redu. Ena izmed možnih rešitev je, da izberemo parametre $t_0 < t_1 < \ldots < t_n$, potem pa poiščemo interpolacijska polinoma p_x in p_y stopnje manjše ali enake n, za katera velja $p_x(t_i) = x_i$ in $p_y(t_i) = y_i$ za $i = 0, \ldots, n$. Polinomska krivulja $p(t) = (p_x(t), p_y(t))$ gre potem skozi vse izbrane točke. Očitno imamo tu veliko svobode, saj

lahko vrednosti parametrov $t_0 < \ldots < t_n$ izberemo poljubno, vsaka izbira pa vpliva na obliko interpolacijske krivulje.

Izkaže se, da boljše lastnosti dobimo, če tudi tu uporabimo zlepke in skupaj sestavljamo polinomske odseke nižjih stopenj med dvema zaporednima točkama.

Pri tem so pomembno orodje *Beziérove krivulje*[6]. Vsaka Beziérova krivulja je določena s kontrolnimi točkami $p_i = (x_i, y_i)$ za $i = 0, \ldots, n$. Formula za točke na krivulji je

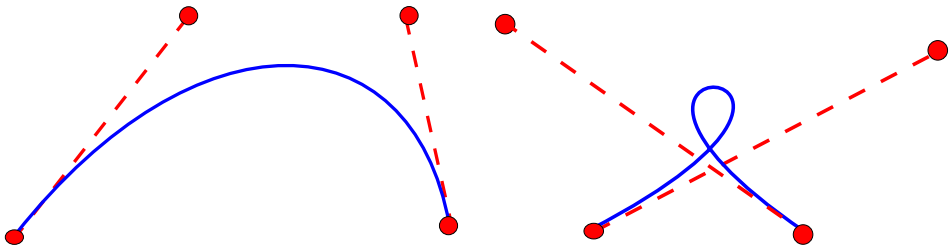
$$P(t) = \sum_{k=0}^{n} \binom{n}{k} t^k (1-t)^{n-k} p_k,$$

kjer je $t \in [0, 1]$. V formuli nastopajo *Bernsteinovi*[7] *polinomi*

$$B_{n,k}(t) = \binom{n}{k} t^k (1-t)^{n-k}$$

za $k = 0, \ldots, n$, za katere iz $(t + (1-t))^n = 1$ sledi, da velja $\sum_{k=0}^{n} B_{n,k}(t) = 1$ za vsak $t \in [0, 1]$. Izkaže se tudi (glej [18] za podrobnosti), da Bernsteinovi polinomi $B_{n,0}, \ldots, B_{n,n}$ tvorijo bazo za polinome stopnje manjše ali enake n.

Najpogosteje se uporabljajo kubične Beziérove krivulje. Dva zgleda sta predstavljena na sliki 7.4.



Slika 7.4: Kubični Beziérovi krivulji.

Nekaj lastnosti Beziérovih krivulj je:

- $P(0) = p_0$ in $P(1) = p_n$.
- Krivulja leži znotraj konveksne ogrinjače točk $p_0, \ldots, p_n$.
- Naklon pri $t = 0$ je enak naklonu premice skozi p_0 in p_1, podobno se naklon pri $t = 1$ ujema z naklonom premice skozi p_{n-1} in p_n.

Pri interpolaciji v ravini sestavljamo skupaj kubične Beziérove krivulje v (Beziérjev) zlepek. Pogoj za geometrijsko zveznost dobljenega zlepka je, da zadnji dve točki prve krivulje in prvi dve točki druge krivulje ležijo na isti premici.

[6] Neodvisno sta jih odkrila francoski matematik Paul de Casteljau (r. 1930) v avtomobilski tovarni Citroën leta 1959 in francoski inženir Pierre Étienne Bézier (1910–1999) v konkurenčni tovarni Renault leta 1962. Krivulje so glavno orodje za računalniško podprto načrtovanje in izdelavo (CAD/CAM), zato je razvoj na začetku potekal večinoma v letalski in avtomobilski industriji.

[7] Ruski matematik Sergej Natanovič Bernstein (1880–1968).

Beziérovi kubični zlepki se npr. uporabljajo za zapis računalniških pisav v vektorski obliki. Zapis ne porabi veliko prostora in omogoča, da se lahko velikost pisave brez izgube kvalitete poljubno poveča.

Omenimo še nekaj pojmov, ki se pojavijo, kadar pri interpolaciji ravninskih krivulj uporabljamo zlepke:

- *Geometrijska zveznost*: krivulji se dotikata, kar pomeni, da se zadnja točka prve krivulje ujema z začetno točko druge krivulje.
- *Geometrijska zveznost odvodov (G^1)*: tangenti prve in druge krivulje imata v skupni točki enako smer.
- *Parametrična zveznost odvodov (C^1)*: prva odvoda prve in druge krivulje v skupni točki se ujemata (tako v smeri kot v velikosti, pri čemer je velikost odvisna od parametrizacije).

Matlab

Na voljo so naslednje funkcije za delo s polinomi in za konstrukcijo interpolacijskih polinomov in zlepkov:

- `polyval(p,x)`: vrednost polinoma, podanega s koeficienti v p v točki x
- `roots(p)`: ničle polinoma, podanega s koeficienti v p
- `poly(r)`: vrne koeficiente polinoma z danimi ničlami v r
- `polyfit(x,y,n)`: vrne koeficiente polinoma p stopnje n, ki po metodi najmanjših kvadratov najbolje aproksimira točke (x_i, y_i), kar pomeni da je vsota

$$\sum_{i=0}^{n} (p(x_i) - y_i)^2$$

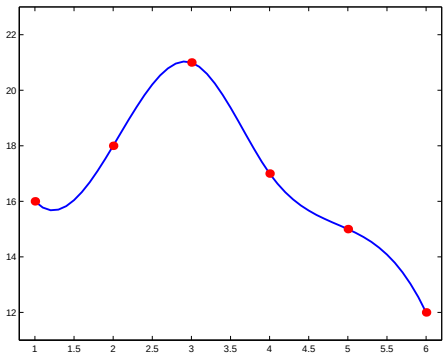
  minimalna. Če izberemo $m \geq n$, potem dobimo interpolacijski polinom.

- `yi=interp1(x,y,xi)`: vrne vrednost kosoma polinomske interpolacijske funkcije v točki `xi`. Kot četrti argument lahko podamo vrsto interpolacijske funkcije, na voljo so
  - `'nearest'`: kosoma konstantna interpolacija,
  - `'linear'`: kosoma linearna interpolacija,
  - `'spline'`: dvakrat zvezno odvedljiv kubični zlepek,
  - `'cubic'`: kubični zlepek, ki ohranja obliko.

Za interpolacijo v dveh dimenzijah sta na voljo ukaza `interp2` in `griddata`.

Zgled 7.4 *Naslednji ukazi v Matlabu zgenerirajo dvakrat zvezno odvedljiv kubični zlepek in narišejo njegov graf, ki je prikazan na desni strani.*

```
x=[1 2 3 4 5 6];
y=[16 18 21 17 15 12];
t=1:0.1:6;
y3=interp1(x,y,t,'spline');
plot(t,y3,'b-','LineWidth',2);
hold on
plot(x,y,'r.','MarkerSize',27);
axis([0.8 6.2 11 23])
hold off
```



Dodatna literatura

Za interpolacijo je na voljo obsežna literatura, saj je obravnavana skoraj v vseh učbenikih numerične analize. V slovenščini imate na voljo knjigo [18], osnove interpolacije pa najdete tudi v knjigah [3] in [34]. Od tuje literature omenimo npr. knjigi [6] in [17].

7.6 Numerično odvajanje

Iščemo odvod funkcije, ki je podana s tabelo vrednosti v točkah $x_0, \ldots, x_n$. Ideja je, da za približek vzamemo odvod interpolacijskega polinoma. Če je funkcija dovoljkrat zvezno odvedljiva, potem vemo, da je

$$f(x) = I_n(x) + \frac{\omega(x)}{(n+1)!} f^{(n+1)}(\xi),$$

kjer je $\omega(x) = (x - x_0) \cdots (x - x_n)$. Z odvajanjem zgornje zveze dobimo

$$f'(x) = I_n'(x) + \underbrace{\frac{\omega'(x)}{(n+1)!} f^{(n+1)}(\xi) + \frac{\omega(x)}{(n+1)!} \cdot \frac{df^{(n+1)}(\xi)}{dx}}_{\text{napaka}}.$$

Izraz za napako ni najlepši, saj ne poznamo odvisnosti ξ od x. To nam preprečuje, da bi lahko ocenili maksimalno napako. Če pa računamo odvod v eni izmed točk $x_0, \ldots, x_n$, zadnji člen odpade in dobimo

$$f'(x_k) = I_n'(x_k) + \frac{\omega'(x_k)}{(n+1)!} f^{(n+1)}(\xi). \tag{7.9}$$

V zgornji formuli je $I_n'(x_k)$ odvod interpolacijskega polinoma v točki x_k. Formulo za $I_n'(x_k)$ lahko izpeljemo preko Lagrangeevih koeficientov. Iz $I_n(x) = \sum_{i=0}^{n} f(x_i) L_{n,i}(x)$ sledi $I_n'(x_k) = \sum_{i=0}^{n} f(x_i) L_{n,i}'(x_k)$. Z odvajanjem Lagrangeevega koeficienta

$$L_{n,i}(x) = \frac{(x - x_0) \cdots (x - x_{i-1})(x - x_{i+1}) \cdots (x - x_n)}{(x_i - x_0) \cdots (x_i - x_{i-1})(x_i - x_{i+1}) \cdots (x_i - x_n)}$$

dobimo

a) $i \neq k$: $L'_{n,i}(x_k) = \dfrac{\omega'(x_k)}{(x_k - x_i)\omega'(x_i)}$,

b) $i = k$: $L'_{n,k}(x_k) = \sum\limits_{\substack{j=0 \\ j\neq k}}^{n} \dfrac{1}{x_k - x_j}$.

S pomočjo teh formul lahko zapišemo

$$I'_n(x_k) = \omega'(x_k) \sum_{\substack{j=0 \\ j\neq k}}^{n} \frac{f(x_j)}{(x_k - x_j)\omega'(x_j)} + f(x_k) \sum_{\substack{j=0 \\ j\neq k}}^{n} \frac{1}{x_k - x_j}.$$

V primeru, ko so točke ekvidistantne in velja $x_i = x_0 + ih$ za $i = 0, 1, \ldots, n$, se formula (8.1) še poenostavi. Z malce računanja lahko izpeljemo formulo

$$f'(x_k) = \frac{1}{h}\left(\frac{(-1)^k}{\binom{n}{k}} \sum_{\substack{j=0 \\ j\neq k}}^{n} \frac{(-1)^j \binom{n}{j} f(x_j)}{k-j} + f(x_k) \sum_{\substack{j=0 \\ j\neq k}}^{n} \frac{1}{k-j} \right) + \frac{(-1)^{n-k} h^n}{(n+1)\binom{n}{k}} f^{(n+1)}(\xi).$$

Nekaj prvih formul, ki jih dobimo z vstavljanjem n in k, je:

- $n = 1$:

$$\begin{aligned} f'(x_0) &= \frac{1}{h}(f(x_1) - f(x_0)) - \frac{1}{2}hf''(\xi_0) \\ f'(x_1) &= \frac{1}{h}(f(x_1) - f(x_0)) + \frac{1}{2}hf''(\xi_1) \end{aligned}$$

- $n = 2$:

$$\begin{aligned} f'(x_0) &= \frac{1}{2h}(-3f(x_0) + 4f(x_1) - f(x_2)) + \frac{1}{3}h^2 f'''(\xi_0) \\ f'(x_1) &= \frac{1}{2h}(-f(x_0) + f(x_2)) - \frac{1}{6}h^2 f'''(\xi_1) \quad \text{(simetrična diferenca)} \\ f'(x_2) &= \frac{1}{2h}(f(x_0) - 4f(x_1) + 3f(x_2)) + \frac{1}{3}h^2 f'''(\xi_2) \end{aligned}$$

Če primerjamo formuli pri $n = 1$ in simetrično diferenco, potem v vseh treh formulah vrednost odvoda aproksimiramo z vrednostnima funkcije v dveh točkah. Pri simetrični diferenci zaradi simetrije pridobimo red h^2 namesto h, višji red pa imamo v bistvu zaradi tega, ker smo odvajali interpolacijski polinom na treh točkah namesto na dveh.

7.7 Drugi načini izpeljave

Formule za numerično odvajanje lahko izpeljujemo tudi iz razvoja v Taylorjevo vrsto. Naj bodo točke ekvidistantne in $y_i = f(x_i)$. Iz razvojev

$$\begin{aligned} y_0 &= y_1 - hy'_1 + \frac{1}{2}h^2 y''_1 - \frac{1}{6}h^3 y'''_1 + \frac{1}{24}h^4 f^{(4)}(\xi_0) \\ y_1 &= y_1 \\ y_2 &= y_1 + hy'_1 + \frac{1}{2}h^2 y''_1 + \frac{1}{6}h^3 y'''_1 + \frac{1}{24}h^4 f^{(4)}(\xi_2) \end{aligned}$$

s seštevanjem dobimo formulo $f''(x_1) = \frac{1}{h^2}(y_0 - 2y_1 + y_2) - \frac{1}{24}h^2(f^{(4)}(\xi_0) + f^{(4)}(\xi_2))$. Namesto $f^{(4)}(\xi_0) + f^{(4)}(\xi_2)$ lahko pišemo $2f^{(4)}(\xi)$ in dobimo končno formulo

$$f''(x_1) = \frac{1}{h^2}(y_0 - 2y_1 + y_2) - \frac{1}{12}h^2 f^{(4)}(\xi). \tag{7.10}$$

Formula (8.2) je simetrična diferenca za drugi odvod. Skupaj s simetrično diferenco za prvi odvod se npr. uporabljata za numerično reševanje robnih problemov preko diferenčne metode. Formuli aproksimirata drugi oz. prvi odvod z natančnostjo $\mathcal{O}(h^2)$.

Alternativni način izpeljave je *metoda nedoločenih koeficientov*. Pri tej metodi nastavimo sistem

$$f'(x_k) = \sum_{j=0}^{n} \alpha_j f(x_j) + R(f)$$

in določimo koeficiente $\alpha_0, \ldots, \alpha_n$ tako, da je formula točna za polinome čim višje stopnje. Napako dobimo iz polinoma najnižje stopnje, za katerega formula ni točna.

Zgled 7.5 *Izpeljimo formulo $f''(x_1) = af(x_0) + bf(x_1) + cf(x_2) + R(f)$.*

Za bazo izberemo $1, x - x_1, (x - x_1)^2, \ldots$, saj tako dobimo lepši sistem:

$$\left.\begin{array}{rcccl} 1 & : & 0 & = & a + b + c \\ x - x_1 & : & 0 & = & -ha + hc \\ (x - x_1)^2 & : & 2 & = & h^2 a + h^2 c \end{array}\right\} \Rightarrow a = \frac{1}{h^2},\ b = \frac{-2}{h^2},\ c = \frac{1}{h^2}.$$

Napaka ima obliko $R(f) = Ch^p f^{(r)}(\xi)$, kjer je C konstanta, p in r pa sta stopnji, ki ju moramo določiti. Odvod r ustreza najnižji stopnji polinoma, za katerega formula ni točna.

Napako dobimo tako, da po vrsti v formulo vstavljamo $f(x) = x^r$ in poiščemo najnižjo stopnjo, za katero formula ni točna. Ker imamo v formuli tri točke, je formula točna za polinome stopnje 2 ali manj, zato najprej vstavimo $r = 3$, ker pa se izkaže, da je formula točna, nadaljujemo z $r = 4$.

$$\left.\begin{array}{rcccl} (x - x_1)^3 & : & 0 & = & -h^3 a + h^3 c \\ (x - x_1)^4 & : & 0 & \neq & h^4 a + h^4 c \end{array}\right\} \Rightarrow r = 4,\ p = 2.$$

Ker za $f(x) = (x - x_1)^4$ velja $f^{(4)}(\xi) = 4!$, sledi, da je napaka enaka $R(f) = -\frac{1}{12}h^2 f^{(4)}(\xi)$. □

7.8 Celotna napaka

Naslednji zgled prikazuje težave, ki se pojavijo pri numeričnem računanju odvodov.

Zgled 7.6 *Če po formulah*

$$\begin{aligned} f'(0) &= \frac{1}{2h}(f(h) - f(-h)) + \mathcal{O}(h^2) \\ f''(0) &= \frac{1}{h^2}(f(-h) - 2f(0) + f(h)) + \mathcal{O}(h^2) \end{aligned}$$

računamo $f'(0)$ in $f''(0)$ za $f(x) = e^x$, potem v enojni natančnosti dobimo:

h	$f'(0)$	$f''(0)$
0.4	1.0268809	1.0134048
0.04	1.0002673	1.0001659
0.004	1.0000094	1.0021030
0.0004	1.0000169	0.7450581
0.00004	1.0006130	$3.7252907 \cdot 10^1$
0.000004	1.0058284	$3.7252903 \cdot 10^3$

Napaka se z manjšanjem h nekaj časa manjša, potem pa začne naraščati, čeprav za obe formuli velja, da imata napako reda $\mathcal{O}(h^2)$. □

Do težav, predstavljenih v prejšnjem zgledu, pride zaradi neodstranljive napake. Oglejmo si situacijo na zgledu formule

$$f''(x_1) = \frac{1}{h^2}(y_0 - 2y_1 + y_2) - \frac{1}{12}h^2 f^{(4)}(\xi). \tag{7.11}$$

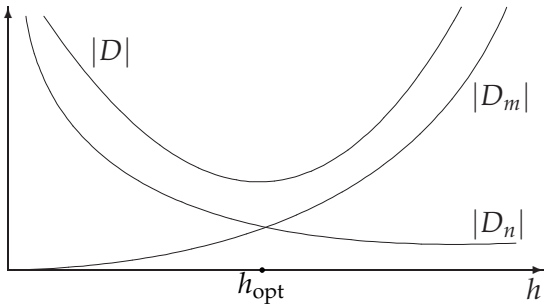
Vemo, da za napako metode D_m velja $D_m = -\frac{1}{12}h^2 f^{(4)}(\xi)$. Od tod lahko ocenimo

$$|D_m| \leq \frac{h^2}{12}\|f^{(4)}\|_\infty.$$

Poleg napake metode pa se pojavi tudi neodstranljiva napaka D_n, saj namesto s točnimi vrednostmi $f(x_i)$ računamo s približki $\widetilde{y}_i$, za katere predpostavimo $|f(x_i) - \widetilde{y}_i| \leq \epsilon$. Če nič drugega, pride do neodstranljive napake že zaradi tega, ker namesto točnih vrednosti uporabljamo najbližja predstavljiva števila. Ocenimo lahko

$$|D_n| \leq \frac{4\epsilon}{h^2}.$$

Pri samem računanju se pojavi še zaokrožitvena napaka D_z, ki pa smo jo v tokratni analizi zanemarili.



Slika 7.5: Ocene za neodstranljivo napako D_n, napako metode D_m in celotno napako D v odvisnosti od h.

Ocena za celotno napako (za formulo (8.3)) je

$$|D| \leq |D_m| + |D_n| \leq \frac{h^2}{12}\|f^{(4)}\|_\infty + \frac{4\epsilon}{h^2}.$$

Če poznamo oceni za $\|f^{(4)}\|_\infty$ in ϵ, lahko iz ocen za $|D_m|$ in $|D_n|$ določimo optimalni h, kjer bo ocena za skupno napako najmanjša.

Numerično odvajanje torej ni numerično dobro pogojen problem, saj se pri premajhnem h zgodi, da se z manjšanjem h napaka poveča.

Dodatna literatura

Za numerično odvajanje imate v slovenščini na voljo knjige [18], [3] in [34], od tuje literature pa omenimo npr. knjigi [6] in [17].

Poglavje 8

Numerično odvajanje in integriranje

8.1 Numerično odvajanje

Denimo, da iščemo odvod funkcije f, ki je podana s tabelo vrednosti v točkah $x_0, \ldots, x_n$. Ideja je, da za približek vzamemo odvod interpolacijskega polinoma. Če je funkcija f dovoljkrat zvezno odvedljiva in je I_n njen interpolacijski polinom na točkah $x_0, \ldots, x_n$, potem vemo, da za napako velja

$$f(x) = I_n(x) + \frac{\omega(x)}{(n+1)!} f^{(n+1)}(\xi),$$

kjer je $\omega(x) = (x - x_0) \cdots (x - x_n)$ in $\min(x, x_0, \ldots, x_n) \leq \xi \leq \max(x, x_0, \ldots, x_n)$. Z odvajanjem zgornje zveze dobimo

$$f'(x) = I_n'(x) + \underbrace{\frac{\omega'(x)}{(n+1)!} f^{(n+1)}(\xi) + \frac{\omega(x)}{(n+1)!} \cdot \frac{df^{(n+1)}(\xi)}{dx}}_{\text{napaka}}.$$

Izraz za napako ni najlepši, saj ne poznamo odvisnosti ξ od x, vemo le, da za vsako točko x obstaja ustrezna točka ξ. To nam preprečuje, da bi lahko ocenili maksimalno napako. Če pa računamo odvod v eni izmed točk $x_0, \ldots, x_n$, potem zadnji člen odpade in ostane

$$f'(x_k) = I_n'(x_k) + \frac{\omega'(x_k)}{(n+1)!} f^{(n+1)}(\xi), \tag{8.1}$$

za $k = 0, \ldots, n$. kjer je $I_n'(x_k)$ odvod interpolacijskega polinoma v točki x_k za $k = 0, \ldots, n$. Formulo za $I_n'(x_k)$ lahko izpeljemo preko Lagrangeevih koeficientov. Iz $I_n(x) = \sum_{i=0}^{n} f(x_i) L_{n,i}(x)$ sledi $I_n'(x_k) = \sum_{i=0}^{n} f(x_i) L_{n,i}'(x_k)$. Z odvajanjem Lagrangeevega koeficienta

$$L_{n,i}(x) = \frac{(x - x_0) \cdots (x - x_{i-1})(x - x_{i+1}) \cdots (x - x_n)}{(x_i - x_0) \cdots (x_i - x_{i-1})(x_i - x_{i+1}) \cdots (x_i - x_n)}$$

dobimo

a) $i \neq k$: $\quad L_{n,i}'(x_k) = \dfrac{\omega'(x_k)}{(x_k - x_i)\omega'(x_i)}$,

b) $i = k$: $\quad L'_{n,k}(x_k) = \sum\limits_{\substack{j=0 \\ j\neq k}}^{n} \dfrac{1}{x_k - x_j}$.

S pomočjo teh formul lahko zapišemo

$$I'_n(x_k) = \omega'(x_k) \sum_{\substack{j=0 \\ j\neq k}}^{n} \frac{f(x_j)}{(x_k - x_j)\omega'(x_j)} + f(x_k) \sum_{\substack{j=0 \\ j\neq k}}^{n} \frac{1}{x_k - x_j}.$$

V primeru, ko so točke ekvidistantne in velja $x_i = x_0 + ih$ za $i = 0, 1, \ldots, n$, se formula (8.1) še poenostavi. Z malce računanja lahko izpeljemo splošno formulo

$$f'(x_k) = \frac{1}{h}\left(\frac{(-1)^k}{\binom{n}{k}} \sum_{\substack{j=0 \\ j\neq k}}^{n} \frac{(-1)^j \binom{n}{j} f(x_j)}{k-j} + f(x_k) \sum_{\substack{j=0 \\ j\neq k}}^{n} \frac{1}{k-j} \right) + \frac{(-1)^{n-k} h^n}{(n+1)\binom{n}{k}} f^{(n+1)}(\xi).$$

Nekaj prvih formul, ki jih dobimo z vstavljanjem n in k, je:

- $n = 1$:

$$\begin{aligned} f'(x_0) &= \frac{1}{h}(f(x_1) - f(x_0)) - \frac{1}{2}hf''(\xi_0), \\ f'(x_1) &= \frac{1}{h}(f(x_1) - f(x_0)) + \frac{1}{2}hf''(\xi_1). \end{aligned}$$

- $n = 2$:

$$\begin{aligned} f'(x_0) &= \frac{1}{2h}(-3f(x_0) + 4f(x_1) - f(x_2)) + \frac{1}{3}h^2 f'''(\xi_0), \\ f'(x_1) &= \frac{1}{2h}(-f(x_0) + f(x_2)) - \frac{1}{6}h^2 f'''(\xi_1), \quad \text{(simetrična diferenca)} \\ f'(x_2) &= \frac{1}{2h}(f(x_0) - 4f(x_1) + 3f(x_2)) + \frac{1}{3}h^2 f'''(\xi_2). \end{aligned}$$

Če primerjamo formuli pri $n = 1$ in simetrično diferenco, potem v vseh treh formulah vrednost odvoda aproksimiramo z vrednostnima funkcije v dveh točkah. Pri simetrični diferenci zaradi simetrije pridobimo red h^2 namesto h, višji red pa imamo v bistvu zaradi tega, ker smo odvajali interpolacijski polinom na treh točkah namesto na dveh.

Formule za numerično odvajanje lahko izpeljujemo tudi iz razvoja v Taylorjevo vrsto. Naj bodo točke ekvidistantne in $y_i = f(x_i)$. Iz razvojev

$$\begin{aligned} y_0 &= y_1 - hy'_1 + \frac{1}{2}h^2 y''_1 - \frac{1}{6}h^3 y'''_1 + \frac{1}{24}h^4 f^{(4)}(\xi_0) \\ y_1 &= y_1 \\ y_2 &= y_1 + hy'_1 + \frac{1}{2}h^2 y''_1 + \frac{1}{6}h^3 y'''_1 + \frac{1}{24}h^4 f^{(4)}(\xi_2) \end{aligned}$$

s seštevanjem dobimo formulo $f''(x_1) = \dfrac{1}{h^2}(y_0 - 2y_1 + y_2) - \dfrac{1}{24}h^2(f^{(4)}(\xi_0) + f^{(4)}(\xi_2))$. Namesto $f^{(4)}(\xi_0) + f^{(4)}(\xi_2)$ lahko pišemo $2f^{(4)}(\xi)$ v neki točki ξ in tako dobimo končno formulo

$$f''(x_1) = \frac{1}{h^2}(y_0 - 2y_1 + y_2) - \frac{1}{12}h^2 f^{(4)}(\xi). \tag{8.2}$$

Formula (8.2) je simetrična diferenca za drugi odvod. Skupaj s simetrično diferenco za prvi odvod se npr. uporabljata za numerično reševanje robnih problemov preko diferenčne metode. Formuli aproksimirata drugi oz. prvi odvod z natančnostjo $\mathcal{O}(h^2)$.

Formule lahko izpeljemo tudi preko *metode nedoločenih koeficientov*. Tu nastavimo sistem

$$f'(x_k) = \sum_{j=0}^{n} \alpha_j f(x_j) + R(f)$$

in določimo koeficiente $\alpha_0, \ldots, \alpha_n$ tako, da je formula točna za polinome čim višje stopnje. Napako dobimo iz polinoma najnižje stopnje, za katerega formula ni točna.

Zgled 8.1 *Preko metode nedoločenih koeficientov izpeljimo formulo $f''(x_1) = af(x_0) + bf(x_1) + cf(x_2) + R(f)$ za ekvidistantne točke x_0, $x_1 = x_0 + h$ in $x_2 = x_0 + 2h$.*

Ker iščemo formulo, ki računa odvod v točki x_1, je najbolje za polinomsko bazo izbrati kar $1, x - x_1, (x - x_1)^2, \ldots$, saj tako dobimo lepši linearni sistem za nedoločene koeficiente a, b in c:

$$\left.\begin{array}{rcccl} 1 & : & 0 & = & a+b+c \\ x-x_1 & : & 0 & = & -ha+hc \\ (x-x_1)^2 & : & 2 & = & h^2a+h^2c \end{array}\right\} \Rightarrow a = \frac{1}{h^2},\ b = \frac{-2}{h^2},\ c = \frac{1}{h^2}.$$

Napaka ima obliko $R(f) = Ch^p f^{(r)}(\xi)$, kjer je C konstanta, p in r pa sta stopnji, ki ju moramo določiti. Odvod r ustreza najnižji stopnji polinoma, za katerega formula ni točna.

Napako dobimo tako, da po vrsti v formulo vstavljamo $f(x) = x^r$ in poiščemo najnižjo stopnjo, za katero formula ni točna. Ker imamo v formuli tri točke, že iz izpeljave sledi, da je formula točna za polinome stopnje 2 ali manj. Zaradi tega najprej vstavimo $r = 3$, ker pa se izkaže, da je formula točna, nadaljujemo z $r = 4$.

$$\left.\begin{array}{rcccl} (x-x_1)^3 & : & 0 & = & -h^3a+h^3c \\ (x-x_1)^4 & : & 0 & \neq & h^4a+h^4c \end{array}\right\} \Rightarrow r = 4,\ p = 2.$$

Ker za $f(x) = (x - x_1)^4$ velja $f^{(4)}(\xi) = 4!$, sledi, da je napaka enaka $R(f) = -\frac{1}{12}h^2 f^{(4)}(\xi)$. □

Naslednji zgled prikazuje težave, ki se pojavijo pri numeričnem računanju odvodov. Teorija pravi, da bi morali pri manjšem razmiku med točkami dobiti boljše približke za vrednost odvoda, a se to pri numeričnem računanju ne zgodi.

Zgled 8.2 *Če po formulah za simetrični diferenci*

$$\begin{aligned} f'(0) &= \frac{1}{2h}(f(h) - f(-h)) + \mathcal{O}(h^2), \\ f''(0) &= \frac{1}{h^2}(f(-h) - 2f(0) + f(h)) + \mathcal{O}(h^2) \end{aligned}$$

računamo $f'(0)$ in $f''(0)$ za $f(x) = e^x$, potem v enojni natančnosti dobimo naslednje vrednosti:

h	$f'(0)$	$f''(0)$
0.4	1.0268809	1.0134048
0.04	1.0002673	1.0001659
0.004	1.0000094	1.0021030
0.0004	1.0000169	0.7450581
0.00004	1.0006130	$3.7252907 \cdot 10^1$
0.000004	1.0058284	$3.7252903 \cdot 10^3$

Napaka se z manjšanjem h nekaj časa manjša, potem pa začne naraščati, čeprav za obe formuli velja, da imata napako reda $\mathcal{O}(h^2)$. □

Do težav, predstavljenih v zadnjem zgledu, pride zaradi neodstranljive napake. Oglejmo si situacijo na primeru simetrične diference za drugi odvod

$$f''(x_1) = \frac{1}{h^2}(y_0 - 2y_1 + y_2) - \frac{1}{12}h^2 f^{(4)}(\xi). \tag{8.3}$$

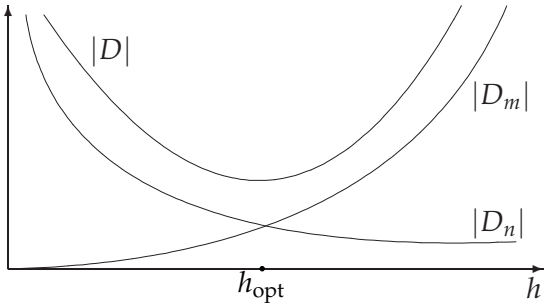
Vemo, da za napako metode D_m velja $D_m = -\frac{1}{12}h^2 f^{(4)}(\xi)$. Od tod lahko ocenimo

$$|D_m| \leq \frac{h^2}{12}\|f^{(4)}\|_\infty,$$

kar očitno pomeni, da gre v limiti, ko h pošljemo proti 0, napaka metode tudi proti 0. Poleg napake metode pa se pojavi tudi neodstranljiva napaka D_n, saj namesto s točnimi vrednostmi $f(x_i)$ računamo s približki $\widetilde{y}_i$, za katere predpostavimo $|f(x_i) - \widetilde{y}_i| \leq \epsilon$. Če nič drugega, pride do neodstranljive napake že zaradi tega, ker pri numeričnem računanju namesto s točnimi vrednostmi v resnici računamo z najbližjimi predstavljivimi števili. V našem primeru lahko neodstranljivo napako za formulo (8.3) ocenimo z

$$|D_n| \leq \frac{4\epsilon}{h^2}.$$

Ker je ϵ neodvisen od h, je neodstranljiva napaka, ko gre h proti 0, lahko poljubno velika.



Slika 8.1: Ocene za neodstranljivo napako D_n, napako metode D_m in celotno napako D v odvisnosti od h.

Pri samem računanju se pojavi še zaokrožitvena napaka D_z, ki pa smo jo v tokratni analizi zanemarili. Ocena za celotno napako (za formulo (8.3)) je tako

$$|D| \leq |D_m| + |D_n| \leq \frac{h^2}{12}\|f^{(4)}\|_\infty + \frac{4\epsilon}{h^2}.$$

Če poznamo oceni za $\|f^{(4)}\|_\infty$ in ϵ, lahko iz ocen za $|D_m|$ in $|D_n|$ določimo optimalni h, kjer bo ocena za skupno napako najmanjša.

Numerično odvajanje torej ni numerično dobro pogojen problem, saj se pri premajhnem h zgodi, da se z manjšanjem h napaka poveča.

8.2 Kvadraturne formule

Radi bi izračunali določeni integral

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

Pri tem predpostavimo, da je funkcija f taka, da integral obstaja, npr., da je kosoma zvezna.

Ideja je, da namesto funkcije f, ki se je v splošnem primeru ne da analitično integrirati, integriramo drugo funkcijo, ki se čim bolj prilega f in za katero obstaja določeni integral. Prva izbira so interpolacijski polinomi.

Če za paroma različne interpolacijske točke izberemo $x_0, \ldots, x_n$ in predpostavimo, da je funkcija f dovoljkrat zvezno odvedljiva, potem lahko s pomočjo Lagrangeevih koeficientov zapišemo

$$f(x) = \sum_{i=0}^{n} f(x_i)L_{n,i}(x) + \frac{f^{(n+1)}(\xi)}{(n+1)!}\omega(x).$$

Če to integriramo, dobimo

$$\int_a^b f(x)dx = \sum_{i=0}^{n} f(x_i) \int_a^b L_{n,i}(x)dx + \int_a^b \frac{f^{(n+1)}(\xi)}{(n+1)!}\omega(x)dx.$$

Od tod sledi, da integral $I(f)$ lahko aproksimiramo s *kvadraturno formulo* oblike

$$\int_a^b f(x)dx = \sum_{i=0}^{n} A_i f(x_i) + R(f), \tag{8.4}$$

kjer paroma različne točke $x_0, \ldots, x_n$ imenujemo *vozli*, koeficiente $A_0, \ldots, A_n$, za katere velja

$$A_i = \int_a^b L_{n,i}(x)dx$$

za $i = 0, \ldots, n$, imenujemo *uteži*, $R(f)$ pa je napaka, ki je enaka

$$R(f) = \int_a^b \frac{f^{(n+1)}(\xi)}{(n+1)!}\omega(x)dx.$$

Formuli pravimo kvadraturna zato, ker je integral $I(f)$ enak ploščini lika, ki ga določa graf funkcije f na intervalu $[a, b]$.

Alternativno bi lahko izpeljali kvadraturno formulo (8.4) tudi tako, da bi najprej izbrali vozle $x_0, \ldots, x_n$, potem pa določili uteži $A_0, \ldots, A_n$ tako, da je formula točna za polinome čim višje stopnje. Pri tem bi podobno kot pri numeričnem odvajanju lahko uporabili metodo nedoločenih koeficientov.

Za kvadraturno formulo (8.4) pravimo, da je reda k, če je točna za vse polinome stopnje manjše ali enake k, ni pa točna za polinome stopnje $k + 1$. V obeh zgornjih primerih dobimo kvadraturno formulo, ki je očitno reda vsaj n, saj je pravilna za polinome stopnje manjše ali enake n, ne glede na to, kako smo izbrali $n + 1$ vozlov. Kot bomo videli kasneje, pa lahko s primerno izbiro vozlov dosežemo, da bo formula točna tudi za polinome višjih stopenj.

8.3 Newton–Cotesova pravila

Pri *Newton-Cotesovih*[1] *pravilih* so vozli ekvidistantni, torej $a = x_0$, $b = x_n$, $h = (b-a)/n$ in $x_i = x_0 + ih$ za $i = 0, \ldots, n$. Vrednosti funkcije f v vozlih označimo z $y_i = f(x_i)$ za $i = 0, \ldots, n$. Ločimo dva tipa Newton–Cotesovih pravil:

a) *zaprti tip*: upoštevamo tudi krajišča: $\int_a^b f(x)dx = \sum_{k=0}^{n} A_k y_k + R_n(f)$;

b) *odprti tip*: brez krajišč: $\int_a^b f(x)dx = \sum_{k=1}^{n-1} B_k y_k + R_n(f)$.

Poglejmo nekaj osnovnih zaprtih Newton–Cotesovih pravil.

- Pri $n = 1$ dobimo *trapezno pravilo*

$$\int_{x_0}^{x_1} f(x)dx = \frac{h}{2}(y_0 + y_1) - \frac{h^3}{12} f''(\xi). \tag{8.5}$$

  Kot se da sklepati že iz imena, funkcijo interpoliramo s premico in površino trapeza (glej sliko 8.2) vzamemo za približek za integral funkcije.

  Koeficiente lahko izračunamo preko integralov Lagrangeevih koeficientov. Tako dobimo

$$\begin{aligned} A_0 &= \int_{x_0}^{x_1} \frac{x - x_1}{x_0 - x_1} dx = \frac{h}{2}, \\ A_1 &= \int_{x_0}^{x_1} \frac{x - x_0}{x_1 - x_0} dx = \frac{h}{2}. \end{aligned}$$

  Za napako velja

$$R_1(f) = \int_{x_0}^{x_1} \frac{f''(\xi_x)}{2}(x - x_0)(x - x_1)dx = \frac{f''(\xi)}{2} \int_{x_0}^{x_1} (x - x_0)(x - x_1)dx = -\frac{h^3}{12} f''(\xi).$$

  Pri izpeljavi napake smo uporabili izrek o povprečni vrednosti, saj ima izraz $(x - x_0)(x - x_1)$ konstanten predznaka na intervalu $[x_0, x_1]$. Trapezno pravilo je točno za polinome stopnje manjše ali enake 1, kar je očitno že iz same interpolacije s premico.

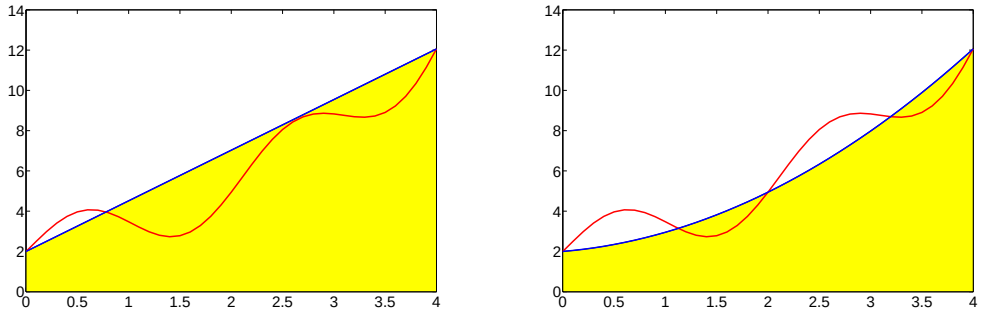
- Pri $n = 2$ dobimo *Simpsonovo*[2] *pravilo*

$$\int_{x_0}^{x_2} f(x)dx = \frac{h}{3}(y_0 + 4y_1 + y_2) - \frac{h^5}{90} f^{(4)}(\xi).$$

  Kot je razvidno tudi iz slike 8.2, funkcijo interpoliramo s parabolo in integral parabole vzamemo za približek za integral funkcije. Podobno kot pri trapezni metodi lahko tudi

[1] Angleški matematik Roger Cotes (1682–1716) je vpeljal merjenje kotov v radianih namesto v stopinjah, z Newtonom pa je sodeloval pri drugi izdaji Newtonove knjige *Principia*.

[2] Pravilo se imenuje po angleškem matematiku Thomasu Simpsonu (1710–1761), ki pa je zanj izvedel od Newtona [24]. Pravilo je že 100 let prej uporabljal nemški matematik in astronom Johannes Kepler (1571–1630).



Slika 8.2: Primerjava trapeznega pravila (levo) in Simpsonovega pravila (desno) na integralu $\int_0^4 (x^2 - x + 2 + 3\sin(2x)\cos x)dx$.

sedaj uteži določimo z integriranjem Lagrangeevih koeficientov. Tako dobimo

$$\begin{aligned} A_0 &= \int_{x_0}^{x_2} \frac{(x-x_1)(x-x_2)}{(x_0-x_1)(x_0-x_2)}dx = \frac{h}{3}, \\ A_1 &= \int_{x_0}^{x_2} \frac{(x-x_0)(x-x_2)}{(x_1-x_0)(x_1-x_2)}dx = \frac{4h}{3}, \\ A_2 &= \int_{x_0}^{x_2} \frac{(x-x_0)(x-x_1)}{(x_2-x_0)(x_2-x_1)}dx = \frac{h}{3}. \end{aligned}$$

Iz interpolacije s parabolo je očitno, da je pravilo točno za polinome stopnje 2 ali manj. Izkaže pa se, da je pravilo točno tudi za kubične polinome, saj za $f(x) = x^3$ velja

$$\int_{x_0}^{x_2} \frac{f^{(3)}(\xi)}{6}(x-x_0)(x-x_1)(x-x_2)dx = 0.$$

Podobno za vsa Newton–Cotesova pravila z lihim številom točk (sodi n) zaradi simetrije velja, da so točne tudi za polinome stopnje $n+1$.

Tu sicer pri izpeljavi napake ne moremo več uporabiti izreka o povprečni vrednosti, a na srečo za Newton–Cotesova pravila velja, da napako za dovoljkrat zvezno odvedljivo funkcijo lahko ugotovimo iz napake polinoma x^{n+1} (oziroma x^{n+2} za sodi n). To sledi iz Peanovega izreka, ki je predstavljen v razdelku 8.6.

- Pri $n = 3$ dobimo *3/8 pravilo*

$$\int_{x_0}^{x_3} f(x)dx = \frac{3h}{8}(y_0 + 3y_1 + 3y_2 + y_3) - \frac{3h^5}{80}f^{(4)}(\xi).$$

Čeprav vsebuje en vozel več, ima 3/8 pravilo enak red natančnosti kot Simpsonovo pravilo. Zaradi tega se 3/8 pravila večinoma ne uporablja.

- Pri $n = 4$ dobimo *Booleovo*[3] *pravilo*

$$\int_{x_0}^{x_4} f(x)dx = \frac{2h}{45}(7y_0 + 32y_1 + 12y_2 + 32y_3 + 7y_4) - \frac{8h^7}{945}f^{(6)}(\xi).$$

[3]Angleški matematik Goerge Boole (1815–1864) je znan predvsem po Booleovi mreži in njegovem delu na področju formalne logike.

Tako kot pri Simpsonovi metodi, tudi pri Booleovi metodi zaradi simetrije pridobimo en red natančnosti.

Nekaj osnovnih odprtih Newton–Cotesovih pravil je:

- Pri $n = 2$ dobimo *sredinsko pravilo*

$$\int_{x_0}^{x_2} f(x)dx = 2hy_1 + \frac{h^3}{3}f''(\xi).$$

Čeprav uporabimo vrednost v eni sami točki, je zaradi simetrije pravilo točno tudi za polinome stopnje 1.

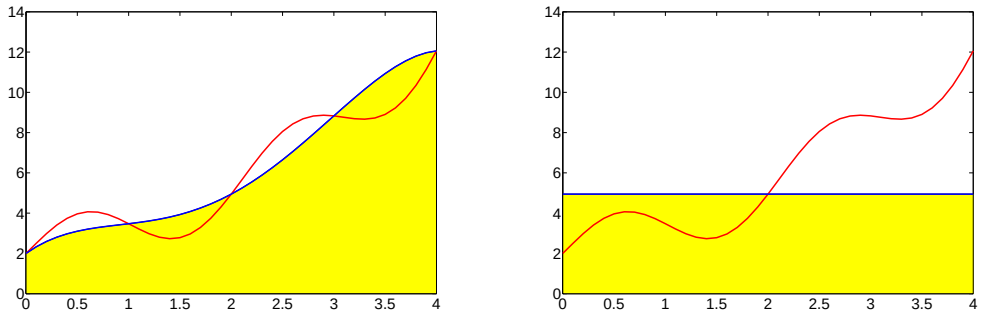
- Pri $n = 3$ dobimo pravilo

$$\int_{x_0}^{x_3} f(x)dx = \frac{3h}{2}(y_1 + y_2) + \frac{3h^3}{4}f''(\xi).$$

- Pri $n = 4$ dobimo *Milneovo*[4] *pravilo*

$$\int_{x_0}^{x_4} f(x)dx = \frac{4h}{3}(2y_1 - y_2 + 2y_3) + \frac{28h^5}{90}f^{(4)}(\xi).$$

To je Newton–Cotesovo pravilo z najmanjšim številom vozlov, ki nima samih uteži s pozitivnim predznakom. Izkaže se, da pri povečevanju števila točk ravno negativne uteži povzročijo, da Newton–Cotesove formule za velike n niso primerne za numerično računanje.



Slika 8.3: Primerjava Booleovega pravila (levo) in sredinskega pravila (desno) na integralu $\int_0^4 (x^2 - x + 2 + 3\sin(2x)\cos x)dx$.

8.4 Neodstranljiva napaka

Pri numeričnem odvajanju smo imeli pri numeričnem računanju težave, ko je šel razmik med točkami h proti 0, saj se je izkazalo, da zaradi neodstranljive napake ne moremo odvoda izračunati poljubno natančno. Glavni problem pri formulah za numerično odvajanje je, da imamo h v

[4] Ameriški matematik William Edmund Milne (1890–1971).

imenovalcu in ko gre potem h proti 0, zgoraj v števcu pa je majhna napaka, lahko neodstranljiva napaka naraste preko vseh meja.

Kako pa je s tem pri kvadraturnih formulah? Ker se sedaj h pojavi v števcu, je na prvi pogled vse v redu, a se izkaže, da imamo tudi tukaj lahko težave. Vzemimo splošno kvadraturno pravilo

$$\int_a^b f(x)dx = \sum_{i=0}^{n} A_i f(x_i) + R(f)$$

in predpostavimo, da je pri izračunu $f(x_i)$ absolutna napaka omejena z $\epsilon > 0$. Ocena za neodstranljivo napako je potem

$$|D_n| \leq \epsilon \sum_{i=0}^{n} |A_i|.$$

Ker so uteži enake integralom Lagrangeevih koeficientov, za vsoto Lagrangeevih koeficientov pa vemo, da je enaka 1, velja

$$\sum_{i=0}^{n} A_i = b - a.$$

Če so vse uteži pozitivne, od tod sledi $|D_n| \leq (b-a)\epsilon$ in dobimo lepo oceno za neodstranljivo napako, ki pada proti 0 ko gre h proti 0. Na žalost pa tako pri zaprtih (za $n > 8$) kot pri odprtih (za $n > 3$) Newton–Cotesovih pravilih dobimo negativne uteži in vsota $\sum_{i=0}^{n} |A_i|$ je lahko zelo velika. Tako se tudi tukaj pojavi napaka in višanje stopnje polinoma ni dobra odločitev.

Naslednja tabela prikazuje vsote $\sum_{i=0}^{n} |A_i|$, ki jih dobimo pri zaprtih Newton–Cotesovih pravilih, če integriramo na intervalu $[0, 1]$.

n	$\sum_{i=0}^{n} \lvert A_i \rvert$
10	3.06
20	$5.44 \cdot 10^2$
30	$2.12 \cdot 10^5$
40	$1.10 \cdot 10^8$
50	$6.70 \cdot 10^{10}$

Iz tabele je očitno, da pri Newton–Cotesovih pravilih z višanjem stopnje polinoma ne moremo natančno izračunati integrala. Namesto tega je bolje uporabljati sestavljene formule.

8.5 Sestavljene formule

Pri sestavljenih formulah interval, po katerem integriramo, razdelimo na manjše podintervale. Na vsakem podintervalu uporabimo kvadraturno pravilo nizke stopnje, nato pa rezultate seštejemo.

Denimo, da interval $[a, b]$ razdelimo z ekvidistantnimi točkami $a = x_0$, $b = x_n$, $h = (b-a)/n$ in $x_i = x_0 + ih$ za $i = 0, \ldots, n$. Osnovne sestavljene formule so:

- *Trapezna formula*:

$$\int_{x_0}^{x_n} f(x)dx = \underbrace{\frac{h}{2}\left(y_0 + 2y_1 + \cdots + 2y_{n-1} + y_n\right)}_{T_h(f)} - \frac{h^2(x_n - x_0)}{12} f''(\xi).$$

Kratka izpeljava je

$$\int_{x_0}^{x_n} f(x)dx = \sum_{k=0}^{n-1} \int_{x_k}^{x_{k+1}} f(x)dx = \sum_{k=0}^{n-1} \left(\frac{h}{2}(y_k + y_{k+1}) - \frac{h^3}{12} f''(\xi_k) \right).$$

Opazimo, da je globalna napaka za en red nižja kot napaka na vsakem intervalu $[x_i, x_{i+1}]$. Do tega pride, ker moramo sešteti n lokalnih napak, pri čemer upoštevamo, da je $nh = x_n - x_0$.

- *Simpsonova formula*:

$$\int_{x_0}^{x_n} f(x)dx = \underbrace{\frac{h}{3}\left(y_0 + 4y_1 + 2y_2 + 4y_3 + \cdots + 2y_{n-2} + 4y_{n-1} + y_n\right)}_{S_h(f)} - \frac{h^4(x_n - x_0)}{180} f^{(4)}(\xi).$$

- *Sredinska formula*:

$$\int_{x_0}^{x_n} f(x)dx = \underbrace{2h\left(y_1 + y_3 + \cdots + y_{n-1}\right)}_{U_h(f)} + \frac{h^2(x_n - x_0)}{6} f''(\xi).$$

Pri Simpsonovi in sredinski formuli mora biti $n = 2m$.

8.6 Peanov izrek

Napake kvadraturnih pravil in formul za numerično odvajanje lahko ocenimo s Peanovim[5] izrekom. Izrek pride prav tudi v primerih, ko funkcija ni dovoljkrat zvezno odvedljiva.

Izrek 8.1 (Peano) *Naj bo $\mathcal{L}$ linearen funkcional oblike*

$$\begin{aligned} \mathcal{L}(f) &= \int_a^b \left(a_0(x)f(x) + a_1(x)f'(x) + \cdots + a_n(x)f^{(n)}(x) \right) dx \\ &+ \sum_{i=0}^{j_0} b_{i0} f(x_{i0}) + \sum_{i=0}^{j_1} b_{i1} f'(x_{i1}) + \cdots + \sum_{i=0}^{j_n} b_{in} f^{(n)}(x_{in}), \end{aligned}$$

kjer so funkcije a_i odsekoma zvezne na $[a, b]$, točke x_{ij} pa ležijo na intervalu $[a, b]$. Funkcional $\mathcal{L}$ naj bo za vse polinome stopnje n ali manj enak 0. Tedaj za vsako funkcijo f, ki je $(n+1)$-krat zvezno odvedljiva na $[a, b]$, velja

$$\mathcal{L}(f) = \int_a^b f^{(n+1)}(t) K_n(t) dt,$$

kjer je K_n Peanovo jedro

$$K_n(t) = \frac{1}{n!} \mathcal{L}((x-t)_+^n)$$

in

$$(x-t)_+^n = \begin{cases} (x-t)^n, & x \geq t, \\ 0, & x < t. \end{cases}$$

[5] Italijanski matematik Giuseppe Peano (1858–1932) je znan predvsem po Peanovih aksiomih o naravnih številih.

Dokaz. Taylorjev izrek z ostankom v integralski obliki pravi

$$f(x) = f(a) + f'(a)(x-a) + \cdots + \frac{f^{(n)}(a)}{n!}(x-a)^n + \frac{1}{n!}\int_a^x f^{(n+1)}(t)(x-t)^n dt,$$

kar lahko zapišemo kot

$$f(x) = f(a) + f'(a)(x-a) + \cdots + \frac{f^{(n)}(a)}{n!}(x-a)^n + \frac{1}{n!}\int_a^b f^{(n+1)}(t)(x-t)_+^n dt. \tag{8.6}$$

Če na obeh straneh (8.6) uporabimo funkcional $\mathcal{L}$, dobimo

$$\mathcal{L}(f) = \frac{1}{n!}\mathcal{L}\left(\int_a^b f^{(n+1)}(t)(x-t)_+^n dt\right) = \frac{1}{n!}\int_a^b f^{(n+1)}(t)\mathcal{L}((x-t)_+^n)dt,$$

saj lahko zamenjamo vrstni red $\mathcal{L}$ in integriranja. ■

Posledica 8.2 *Če je Peanovo jedro K_n konstantnega predznaka, potem za $(n+1)$-krat zvezno odvedljivo funkcijo f velja*

$$\mathcal{L}(f) = \frac{f^{(n+1)}(\xi)}{(n+1)!}\mathcal{L}(x^{n+1}).$$

Dokaz. Po Peanovem izreku velja $\mathcal{L}(x^{n+1}) = \int_a^b (n+1)!K_n(t)dt$, torej je

$$\int_a^b K_n(t)dt = \frac{1}{(n+1)!}\mathcal{L}(x^{n+1}).$$

Ker je Peanovo jedro K_n konstantnega predznaka, po izreku o povprečni vrednosti sledi

$$\mathcal{L}(f) = \int_a^b f^{(n+1)}(t)K_n(t)dt = f^{(n+1)}(\xi)\int_a^b K_n(t)dt.$$

■

Zgled 8.3 *Ocenimo napako trapeznega pravila pri integriranju enkrat zvezno odvedljive funkcije f. Standardne formule za napako iz (8.5) ne moremo uporabiti, saj f ni dvakrat zvezno odvedljiva. Uporabili bomo Peanov izrek in K_0.*

$$\begin{aligned} K_0(t) &= \int_{x_0}^{x_1}(x-t)_+^0 dx - \frac{h}{2}\left((x_0-t)_+^0 + (x_1-t)_+^0\right) \\ &= \int_t^{x_1}(x-t)^0 dx - \frac{h}{2}(0+1) \\ &= x_1 - t - \frac{h}{2} = \frac{x_0+x_1}{2} - t. \end{aligned}$$

Peanovo jedro ni konstantnega predznaka, zato lahko le ocenimo

$$|R(f)| \le \int_{x_0}^{x_1}|f'(t)|\left|\frac{x_0+x_1}{2} - t\right| dt \le \frac{h^2}{4}\|f'\|_\infty.$$

□

8.7 Richardsonova ekstrapolacija

Pri numeričnem odvajanju in integriranju ponavadi poskušamo priti do čim boljšega približka za točen rezultat tako, da zmanjšamo razmik h. Pri tem si ne moremo privoščiti, da bi bil h poljubno majhen, saj potem v primeru numeričnega odvajanja podivja neodstranljiva napaka, pri numeričnem integriranju pa postane izračun predrag.

Denimo, da je $F(h)$ približek, ki ga dobimo z razmikom h, in denimo, da velja

$$F(h) = a_0 + a_1 h^p + \mathcal{O}(h^r) \tag{8.7}$$

za $r > p$, pri čemer sta konstanti a_0 in a_1 neodvisni od h. Točen rezultat, ki ga iščemo, je $F(0) = a_0$. Če izračunamo približka $F(h)$ in $F(h/2)$, potem iz (8.7) in

$$F(h/2) = a_0 + a_1 (h/2)^p + \mathcal{O}(h^r)$$

sledi

$$a_0 = F(h/2) + \frac{F(h/2) - F(h)}{2^p - 1} + \mathcal{O}(h^r). \tag{8.8}$$

Iz formule (8.8) ugotovimo dvoje. Kot prvo, pri predpostavki (8.7) lahko iz približkov $F(h)$ in $F(h/2)$ dobimo natančnejši približek

$$F(h, h/2) = \frac{2^p F(h/2) - F(h)}{2^p - 1},$$

za katerega velja $F(h, h/2) = a_0 + \mathcal{O}(h^r)$. Temu postopku pravimo Richardsonova[6] ekstrapolacija. Druga ugotovitev je, da lahko kot oceno za napako približka $F(h/2)$ vzamemo kar $(F(h/2) - F(h))/(2^p - 1)$.

Podobno bi lahko v primeru, ko za napako vemo, da ima obliko

$$F(h) = a_0 + a_1 h^{p_1} + a_2 h^{p_2} + \mathcal{O}(h^r), \tag{8.9}$$

kjer je $p_1 < p_2 < r$, iz $F(h)$, $F(h/2)$ in $F(h/4)$ izračunali približek $F(h, h/2, h/4)$, ki bi imel napako velikosti $\mathcal{O}(h^r)$. Tovrsten postopek bomo uporabili kasneje pri Rombergovi ekstrapolaciji.

Za zgled uporabimo Richardsonovo ekstrapolacijo na Simpsonovi formuli. Naj bo $S_h(f)$ Simpsonova formula s korakom h in $R_h(f)$ napaka Simpsonove formule pri koraku h. Vemo, da velja

$$I(f) = \int_a^b f(x)dx = S_h(f) + R_h(f) = S_{h/2}(f) + R_{h/2}(f).$$

Za napaki velja

$$R_h(f) = \frac{-(b-a)h^4}{180} f^{(4)}(\xi_1), \quad R_{h/2}(f) = \frac{-(b-a)h^4}{16 \cdot 180} f^{(4)}(\xi_2).$$

Če predpostavimo, da je $f^{(4)}(\xi_1) \approx f^{(4)}(\xi_2)$, dobimo

$$R_h(f) \approx 16 R_{h/2}(f).$$

[6] Lewis Fry Richardson (1881–1953) je bil angleški matematik, fizik in meteorolog. Prvi je uporabljal metodo končnih diferenc za numerično reševanje parcialnih diferencialnih enačb.

Iz $R_{h/2}(f) = I(f) - S_{h/2}(f) = S_h(f) + R_h(f) - S_{h/2}(f) \approx S_h(f) + 16R_{h/2}(f) - S_{h/2}(f)$ dobimo

$$R_{h/2}(f) \approx \frac{S_{h/2}(f) - S_h(f)}{15}.$$

To formulo lahko uporabimo za oceno napake Simpsonove formule. Ocena sicer ni preveč zanesljiva, saj smo izenačili odvode, a v večini primerov vseeno dobimo spodobne rezultate in lahko ocenimo velikostni razred napake. Poleg ocene lahko iz $S_{h/2}(f)$ in $S_h(f)$ z ekstrapolacijo dobimo še boljši približek, saj je

$$I(f) = S_{h/2}(f) + R_{h/2}(f) \approx \frac{16S_{h/2}(f) - S_h(f)}{15}.$$

Podobno lahko naredimo pri trapezni in sredinski formuli, le konstante so drugačne.

8.8 Adaptivne metode

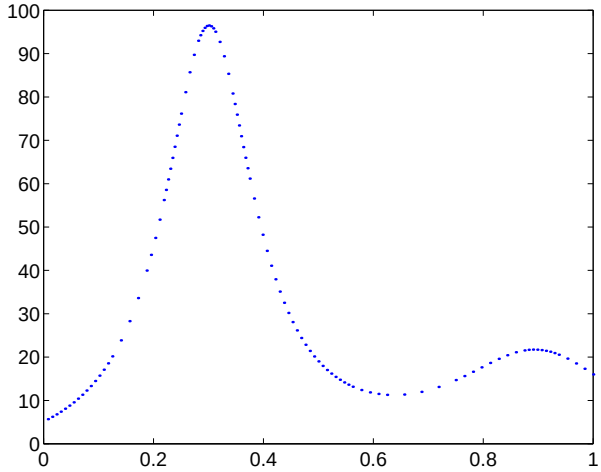
Večina metod, ki se jih v praksi uporablja za numerično integriranje, deluje na adaptivnem principu. To pomeni, da metoda sproti ocenjuje napako in temu prilagaja velikost podintervalov. Pri adaptivnih metodah tako na območjih, kjer je funkcija pohlevna, uporabimo večji razmik h, kjer je njeno obnašanje bolj divje, pa manjši h.

Rekurzivna adaptivna metoda, ki temelji na Simpsonovem pravilu in Richardsonovi ekstrapolaciji, je predstavljena v algoritmu 8.1.

Algoritem 8.1 Adaptivna Simpsonova metoda. Vhodni podatki so funkcija f, interval $[a, b]$ in toleranca δ. Metoda vrne vrednost integrala Q in oceno $\epsilon \leq \delta$ za napako tega približka.

$[Q, \epsilon]$ =AdaptivniSimpson(f, a, b, δ)
  $h = b - a$
  $c = (a + b)/2$
  $d = (a + c)/2, \quad e = (c + b)/2$
  $Q_1 = (h/6)(f(a) + 4f(c) + f(b))$
  $Q_2 = (h/12)(f(a) + 4f(d) + 2f(c) + 4f(e) + f(b))$
  $\epsilon = |Q_1 - Q_2|$
  if $\epsilon \leq \delta$
    $Q = Q_2 + (Q_2 - Q_1)/15$
  else
    $[Q_a, \epsilon_a]$ =AdaptivniSimpson$(f, a, c, \delta/2)$
    $[Q_b, \epsilon_b]$ =AdaptivniSimpson$(f, c, b, \delta/2)$
    $Q = Q_a + Q_b$
    $\epsilon = \epsilon_a + \epsilon_b$
  end

V algoritmu 8.1 prvi približek Q_1 za integral dobimo tako, da na celem intervalu uporabimo Simpsonovo pravilo, drugi približek Q_2 pa dobimo tako, da interval razdelimo na dva enaka dela in na vsakem uporabimo Simpsonovo pravilo. Potem primerjamo Q_1 in Q_2.



Slika 8.4: Primer uporabe adaptivnega kvadraturnega pravila v Matlabu. Vrednosti, označene na grafu, so bile uporabljene za izračun integrala. Vidi se, da točke niso ekvidistantne.

- Če je razlika $|Q_2 - Q_1|$ večja od zahtevane natančnosti δ, potem interval razdelimo na dva dela in na vsakem rekurzivno izračunamo integral z isto metodo, zahtevamo pa, da je sedaj napaka na vsaki polovici pod $\delta/2$.
- Če je $|Q_2 - Q_1| \leq \delta$, je Q_2 približek za integral, izračunan z zahtevano natančnostjo. Ker pa imamo na voljo še Q_1, lahko naredimo še korak Richardsonove ekstrapolacije in tako še izboljšamo rezultat.

Namesto Simpsonovega pravila bi lahko uporabili tudi kakšno drugo kvadraturno pravilo. V grobem pri adaptivnem integriranju vedno vrednost integrala ocenimo z dvema kvadraturnima praviloma. Njuna razlika nam služi kot ocena za napako. Če je razlika prevelika, interval razpolovimo in postopek rekurzivno ponovimo. Da bo postopek ekonomičen, moramo kvadraturni pravili izbrati tako, da se vozli čim bolj prekrivajo in da se vsaj del vozlov prenese v razpolovljeni interval. Tako dosežemo, da je končnih izračunov vrednosti funkcije čim manj, saj pri numeričnem integriranju zahtevnost merimo v tem, koliko vrednosti funkcije moramo izračunati, da pridemo do dovolj dobrega približka.

8.9 Rombergova metoda

Če je funkcija f na intervalu $[a, b]$ dovoljkrat zvezno odvedljiva, potem iz Euler-Maclaurinove[7] sumacijske formule sledi, da se ostanek trapezne formule izraža kot asimptotična vrsta potence h^2. To nam omogoča, da z zaporedno uporabo Richardsonove ekstrapolacije po vrsti uničimo vodilne člene in izboljšamo rezultat, ustrezen postopek pa se imenuje *Rombergova metoda*.

Izrek 8.3 *(Euler-Maclaurin) Če je funkcija f na intervalu $[a, b]$ $(2m + 2)$-krat zvezno odvedljiva, po-*

[7]Formulo sta neodvisno odkrila švicarski matematik Leonhard Euler (1707–1783) in škotski matematik Colin Maclaurin (1698–1746). Euler, eden najpomebnejših matematikov 18. stoletja, znan po številnih rezultatih iz analize, teorije števil in teorije grafov, je sumacijsko formulo uporabljal za računanje vsot počasi konvergentnih vrst, Maclaurin pa jo je uporabljal za računanje integralov.

tem velja sumacijska formula

$$I(f) = \int_a^b f(x)dx = T_h(f) - \sum_{k=1}^{m} \frac{B_{2k}}{(2k)!} h^{2k} \left(f^{(2k-1)}(b) - f^{(2k-1)}(a) \right) + R(f), \tag{8.10}$$

kjer je $T_h(f)$ trapezna formula s korakom h, B_k so Bernoullijeva števila, ostanek pa je enak

$$R(f) = -\frac{B_{2m+2}}{(2m+2)!}(b-a)h^{2m+2} f^{(2m+2)}(\xi)$$

za nek $\xi \in (a, b)$.

Dokaz in podrobno izpeljavo formule (8.10) lahko najdete npr. v [18].

Definicija 8.4 *Bernoullijeva*[8] *števila B_k so določena z razvojem*

$$\frac{x}{e^x - 1} = \sum_{k=0}^{\infty} \frac{B_k}{k!} x^k, \quad |x| < 2\pi.$$

Vsa Bernoullijeva števila so racionalna, vsa z lihimi indeksi razen B_1 pa so enaka 0. Nekaj prvih Bernoullijevih števil je

$$B_0 = 1, \quad B_1 = -\frac{1}{2}, \quad B_2 = \frac{1}{6}, \quad B_4 = -\frac{1}{30}, \quad B_6 = \frac{1}{42}, \quad B_8 = -\frac{1}{30}, \quad B_{10} = \frac{5}{66}.$$

Če zapišemo nekaj prvih členov vsote v (8.10), dobimo

$$I(f) = T_h(f) - \frac{h^2}{12}(f'(b) - f'(a)) + \frac{h^4}{720}(f'''(b) - f'''(a)) + \cdots.$$

Če poznamo nekaj začetnih lihih odvodov funkcije f v robnih točkah intervala, lahko tako z njimi izboljšamo približek $T_h(f)$, ki ga dobimo s trapezno formulo.

Za Rombergovo metodo je bistvena ugotovitev, da lahko (8.10) pišemo v obliki

$$I(f) = T_h(f) + \sum_{k=1}^{m} a_{k,0} h^{2k} + \mathcal{O}(h^{2m+2}),$$

pri čemer so koeficienti $a_{k,0}$ neodvisni od h. To je posplošitev formule (8.9) in z Richardsonovo ekstrapolacijo lahko iz približkov, ki jih dobimo za različne razmike h, dobimo boljši približek. Če uporabimo še razmika $h/2$ in $h/4$, dobimo

$$\begin{aligned}
I(f) &= T_h(f) + a_{1,0}h^2 + a_{2,0}h^4 + a_{3,0}h^6 + \cdots \\
I(f) &= T_{h/2}(f) + a_{1,0}\left(\frac{h}{2}\right)^2 + a_{2,0}\left(\frac{h}{2}\right)^4 + a_{3,0}\left(\frac{h}{2}\right)^6 + \cdots \\
I(f) &= T_{h/4}(f) + a_{1,0}\left(\frac{h}{4}\right)^2 + a_{2,0}\left(\frac{h}{4}\right)^2 + a_{3,0}\left(\frac{h}{4}\right)^6 + \cdots
\end{aligned}$$

[8] Števila je odkril švicarski matematik Jacob Bernoulli (1654–1705). Leta 1842 je Ada Lovelace zapisala algoritem, ki bi s pomočjo analitičnega stroja, ki ga je načrtoval Charles Babbage, računal Bernoullijeva števila. Čeprav stroj ni bil nikoli dokončan, velja Ada Lovelace za pionirko računalniškega programiranja.

Če enačbo s $T_{h/2}(f)$ pomnožimo s 4 in odštejemo od enačbe s $T_h(f)$, se znebimo vodilnega člena h^2 in dobimo natančnejši približek, podobno pa naredimo tudi s približkoma $T_{h/4}(f)$ in $T_{h/2}(f)$. Velja

$$\begin{aligned} I(f) &= T^{(1)}_{h/2}(f) + a_{2,1}h^4 + a_{3,1}h^6 + \cdots \\ I(f) &= T^{(1)}_{h/4}(f) + a_{2,1}\left(\frac{h}{2}\right)^4 + a_{3,1}\left(\frac{h}{2}\right)^6 + \cdots, \end{aligned}$$

kjer je

$$T^{(1)}_{h/2}(f) = \frac{4T_{h/2}(f) - T_h(f)}{3}, \quad T^1_{h/4}(f) = \frac{4T_{h/4}(f) - T_{h/2}(f)}{3}.$$

Postopek sedaj nadaljujemo tako, da se znebimo vodilnega člena h^4. Dobimo

$$I(f) = T^{(2)}_{h/4}(f) + a_{3,2}h^6 + a_{4,2}h^8 + \cdots,$$

kjer je

$$T^{(2)}_{h/4}(f) = \frac{16T^{(1)}_{h/4}(f) - T^{(1)}_{h/2}(f)}{15}.$$

V splošnem postopku tvorimo trikotno shemo

napaka	$\mathcal{O}(h^2)$	$\mathcal{O}(h^4)$	$\mathcal{O}(h^6)$	$\mathcal{O}(h^8)$	$\cdots$
	$T^{(0)}_h(f)$				
	$T^{(0)}_{h/2}(f)$	$T^{(1)}_{h/2}(f)$			
	$T^{(0)}_{h/4}(f)$	$T^{(1)}_{h/4}(f)$	$T^{(2)}_{h/4}(f)$		
	$T^{(0)}_{h/8}(f)$	$T^{(1)}_{h/8}(f)$	$T^{(2)}_{h/8}(f)$	$T^{(3)}_{h/8}(f)$	

s splošno formulo

$$T^{(j)}_{h/2^k}(f) = \frac{4^j T^{(j-1)}_{h/2^k}(f) - T^{(j-1)}_{h/2^{k-1}}(f)}{4^j - 1}$$

za $j = 0, \ldots, m$ in $k = m - j + 1, \ldots, m$.

Zgled 8.4 *Z Rombergovo metodo bomo izračunali* $\int_1^{2.2} \ln x\, dx = 0.5346062$. *Začeli bomo s* $h = 0.6$ *in nato naredili dve razpolavljanji.*

Dobimo:

$$\begin{aligned} T^{(0)}_h &= 0.6\left(\frac{1}{2}\ln 1 + \ln 1.6 + \frac{1}{2}\ln 2.2\right) = 0.5185394 \\ T^{(0)}_{h/2} &= \frac{1}{2}T^{(0)}_h + 0.3(\ln 1.3 + \ln 1.9) = 0.5305351 \\ T^{(0)}_{h/4} &= \frac{1}{2}T^{(0)}_{h/2} + 0.15(\ln 1.15 + \ln 1.45 + \ln 1.75 + \ln 2.05) = 0.5335847. \end{aligned}$$

Pomembno je, da $T_{h/2^k}$ *vedno računamo kot*

$$T_{h/2^k} = \frac{1}{2}T_{h/2^{k-1}} + \frac{h}{2^k}(y_1 + y_3 + \cdots + y_{2^k-1}).$$

Tako vsako funkcijsko vrednost izračunamo le enkrat in imamo z Rombergovo ekstrapolacijo zanemarljivo dodatnega dela v primerjavi z računanjem $T^{(0)}_{h/2^k}$, *rezultat pa je lahko mnogo natančnejši.*

Sedaj z Rombergovo ekstrapolacijo dobimo

$$\begin{aligned} T_{h/2}^{(1)} &= \frac{4T_{h/2}^{(0)} - T_h^{(0)}}{3} = 0.5345337 \\ T_{h/4}^{(1)} &= \frac{4T_{h/4}^{(0)} - T_{h/2}^{(0)}}{3} = 0.5346013 \\ T_{h/4}^{(2)} &= \frac{16T_{h/4}^{(1)} - T_{h/2}^{(1)}}{15} = 0.5346058. \end{aligned}$$

Vidimo, da je ekstrapolirana vrednost $T_{h/4}^{(2)}$ mnogo natančnejša od $T_{h/4}^{(0)}$, pri obeh pa smo uporabili vrednosti funkcije v istih točkah. □

Zgled 8.5 *Če isti postopek poskusimo na integralu $\int_0^1 \sqrt{x}\,dx = 2/3$ z začetnim $h = 0.5$, potem dobimo naslednjo tabelo*

$$\begin{array}{lll} T_h^{(0)}(f) = 0.6035534 & & \\ T_{h/2}^{(0)}(f) = 0.6432831 & T_{h/2}^{(1)}(f) = 0.6565263 & \\ T_{h/4}^{(0)}(f) = 0.6581302 & T_{h/4}^{(1)}(f) = 0.6630793 & T_{h/4}^{(2)}(f) = 0.6635162 \end{array}$$

Ekstrapolacija očitno ne deluje tako dobro kot v prejšnjem zgledu, razlog pa je, da funkcija ni odvedljiva v levi robni točki in zato pogoji izreka 8.3 niso izpolnjeni. □

8.10 Gaussove kvadraturne formule

Integral $\int_a^b f(x)\rho(x)dx$, kjer smo dodali še nenegativno utež ρ, aproksimiramo s kvadraturno formulo

$$\int_a^b f(x)\rho(x)dx = \sum_{i=0}^{n} A_i^{(n)} f(x_i^{(n)}) + R(f). \tag{8.11}$$

V primeru $\rho \equiv 1$ dobimo integral $\int_a^b f(x)dx$, tako da teorija iz tega razdelka pokrije tudi standardni primer.

Če uporabimo isto idejo kot v razdelku 8.2 in namesto f integriramo interpolacijski polinom, ugotovimo da so koeficienti prav tako določeni z vozli, saj velja

$$A_i^{(n)} = \int_a^b L_{n,i}(x)\rho(x)dx,$$

formula pa je točna za polinome stopnje vsaj n. S primerno izbiro vozlov lahko dosežemo, da bo formula (8.11) točna za polinome stopnje vsaj $2n+1$, v ozadju pa so ortogonalni polinomi.

Da je formula (8.11) lahko točna za polinome stopnje $2n+1$, je očitno že iz tega, da v njej nastopa $2n+2$ parametrov (vozlov in uteži). Po metodi nedoločenih koeficientov bi lahko te parametre nastavili tako, da bi bilo pravilo točno za polinome stopnje $2n+1$. Težava pri tem pristopu je, da dobimo nelinearen sistem, ki ga je težko numerično rešiti. Lažje je, če najprej s pomočjo ortogonalnih polinomov določimo vozle, uteži pa lahko izračunamo z integriranjem Lagrangeevih koeficientov.

Za funkciji f in g, definirani na intervalu $[a,b]$, definiramo njun skalarni produkt kot

$$\langle f,g\rangle = \int_a^b f(x)g(x)\rho(x)dx. \tag{8.12}$$

Funkciji sta si ortogonalni, če je $\langle f,g\rangle = 0$. Iz standardne baze polinomov $1, x, x^2, \ldots$ lahko z ortogonalizacijo dobimo ortonormirano bazo $P_0(x), P_1(x), P_2(x), \ldots$, kjer je P_i polinom stopnje i in velja

$$\langle P_i, P_k\rangle = \delta_{ik}.$$

Lema 8.5 *Naj bo P_{n+1} normiran polinom, ki je glede na skalarni produkt (8.12) ortogonalen na vse polinome stopnje manjše ali enake n. Potem so vse ničle polinoma P_{n+1} realne, enostavne in ležijo na intervalu (a,b).*

Dokaz. Lemo dokažemo s protislovjem. Denimo, da so $z_1, \ldots, z_k$ vse ničle polinoma P_{n+1}, ki ležijo na intervalu (a,b), pri čemer morebitne večkratne ničle štejemo le enkrat, in naj velja $k < n+1$. Sedaj definiramo polinom

$$q(x) = (x-z_1)^{j_1}\ldots(x-z_k)^{j_k},$$

kjer za potenco j_i vzamemo 1, če je z_i liha ničla polinoma P_{n+1}, oziroma 2 v primeru sode ničle za $i = 1, \ldots, k$. Za q očitno velja $\langle P_{n+1}, q\rangle \neq 0$, kar je protislovje, saj je v primeru $k < n+1$ stopnja polinoma q manjša kot $n+1$ in bi moral biti P_{n+1} ortogonalen na q. ∎

Po zgornji lemi lahko torej pišemo

$$P_{n+1}(x) = k_{n+1}(x-x_0^{(n)})\cdots(x-x_n^{(n)}),$$

kjer vse ničle $x_0^{(n)}, \ldots, x_n^{(n)}$ ležijo na intervalu (a,b). Pri *Gaussovi kvadraturni formuli* za vozle izberemo ravno točke $x_0^{(n)}, \ldots, x_n^{(n)}$. Pokažimo, da je tako dobljeno pravilo točno za vse polinome stopnje manjše ali enake $2n+1$.

Poljuben polinom p stopnje manjše ali enake $2n+1$ lahko zapišemo kot $p(x) = q(x)\omega(x) + r(x)$, kjer je je $\omega(x) = (x-x_0^{(n)})\cdots(x-x_n^{(n)})$ in sta q in r polinoma stopnje kvečjemu n. Ker sta polinoma q in ω ortogonalna, dobimo:

$$\begin{aligned}\int_a^b p(x)\rho(x)dx &= \int_a^b q(x)\omega(x)\rho(x)dx + \int_a^b r(x)\rho(x)dx \\ &= 0 + \sum_{i=0}^{n} A_i^{(n)} r(x_i^{(n)}) = \sum_{i=0}^{n} A_i^{(n)} p(x_i^{(n)}),\end{aligned}$$

saj v vozlih velja $r(x_i^{(n)}) = p(x_i^{(n)})$. Torej je pravilo res točno za vse polinome stopnje $2n+1$ ali manj.

Naslednja lema pove, da imajo Gaussove kvadraturne formule poleg tega, da imajo maksimalni možni red (glede na število uporabljenih vozlov), še to lepo lastnost, da so vse uteži pozitivne. Zaradi tega pri Gaussovih kvadraturnih formulah pri povečevanju števila točk nimamo težav z neodstranljivo napako.

Lema 8.6 *Uteži Gaussovih kvadraturnih pravil so pozitivne.*

Dokaz. Vzemimo

$$P_i(x) = \frac{\omega^2(x)}{(x - x_i^{(n)})^2}$$

za $i = 0, \ldots, n$. P_i je polinom stopnje $2n$, torej je zanj kvadraturno pravilo točno in velja

$$\int_a^b P_i(x)\rho(x)dx = \sum_{k=0}^{n} A_k P_i(x_k^{(n)}) = A_i P_i(x_i^{(n)}).$$

Ker je $P_i(x_i^{(n)}) > 0$ in je P_i nenegativna funkcija, mora biti $A_i > 0$. ■

Izrek 8.7 *Če je $f \in \mathcal{C}^{(2n+2)}[a, b]$, potem za napako Gaussovega kvadraturnega pravila na $n + 1$ vozlih velja*

$$\int_a^b f(x)\rho(x) = \sum_{i=0}^{n} A_i^{(n)} f(x_i^{(n)}) + \frac{f^{(2n+2)}(\xi)}{(2n+2)!k_{n+1}^2}.$$

Dokaz. Vzamemo Hermiteov interpolacijski polinom H stopnje $2n + 1$ v točkah $x_0^{(n)}, \ldots, x_n^{(n)}$, za katerega velja $H(x_i^{(n)}) = f(x_i^{(n)})$ in $H'(x_i^{(n)}) = f'(x_i^{(n)})$ za $i = 0, \ldots, n$. Iz

$$f(x) = H(x) + \frac{f^{(2n+2)}(\xi)}{(2n+2)!}\omega^2(x)$$

sledi

$$\int_a^b f(x)\rho(x)dx = \int_a^b H(x)\rho(x)dx + \int_a^b \frac{f^{(2n+2)}(\xi)}{(2n+2)!}\omega^2(x)dx.$$

Ker je H polinom stopnje $2n + 1$, velja

$$\int_a^b H(x)\rho(x)dx = \sum_{k=0}^{n} A_k H(x_k^{(n)}) = \sum_{k=0}^{n} A_k f(x_k^{(n)}),$$

za drugi integral pa velja

$$\int_a^b \frac{f^{(2n+2)}(\xi)}{(2n+2)!}\omega^2(x)dx = \int_a^b \frac{f^{(2n+2)}(\xi)}{(2n+2)!} \cdot \frac{P_{n+1}^2(x)}{k_{n+1}^2}dx = \frac{f^{(2n+2)}(\xi)}{(2n+2)!k_{n+1}^2}\int_a^b P_{n+1}^2(x)dx.$$ ■

Koeficiente A_i lahko izračunamo z integriranjem Lagrangeevih koeficientov ali pa uporabimo *Darboux–Cristoffelove*[9] *formule* (izpeljavo lahko najdete npr. v [18]):

$$A_i^{(n)} = \frac{1}{\sum_{j=0}^{n} P_j^2(x_i^{(n)})}, \quad i = 0, \ldots, n. \tag{8.13}$$

Eleganten numerični način računanja uteži in vozlov za Gaussovo kvadraturno formulo je, da problem prevedemo na iskanje lastnih vrednosti in lastnih vektorjev. Najprej pokažimo, da obstajajo taki koeficienti a_k in b_k, da ortonormirani polinomi zadoščajo tričlenski rekurzivni formuli

$$xP_k(x) = b_{k-1}P_{k-1}(x) + a_k P_k(x) + b_k P_{k+1}(x). \tag{8.14}$$

[9] Francoski matematik Jean Gaston Darboux (1842–1917) in nemški matematik Elwin Bruno Christoffel (1829–1900).

Omenili smo že, da za računanje baze ortonormiranih polinomov lahko uporabimo Gram–Schmidtovo ortogonalizacijo. Če poznamo polinome $P_0, \ldots, P_k$, potem lahko P_{k+1} dobimo tako, da polinom xP_k, ki je stopnje $k+1$, ortogonaliziramo glede na $P_0, \ldots, P_k$. Pred dokončnim normiranjem ima tako polinom $\widetilde{P}_{k+1}$ obliko

$$\widetilde{P}_{k+1} = xP_k - \sum_{j=1}^{k} \langle xP_k, P_j \rangle P_j.$$

V vsoti sta lahko neničelna le zadnja dva člena, saj za $j < k-1$ velja $\langle xP_k, P_j \rangle = \langle P_k, xP_j \rangle = 0$. Tako ostane

$$\widetilde{P}_{k+1} = xP_k - \langle xP_k, P_k \rangle P_k - \langle xP_k, P_{k-1} \rangle P_{k-1}.$$

Sedaj definiramo $a_j = \langle xP_j, P_j \rangle$ in $b_j = \|\widetilde{P}_{j+1}\| = \langle \widetilde{P}_{j+1}, \widetilde{P}_{j+1} \rangle^{(1/2)}$ za $j = 1, \ldots, k$. Iz izračuna

$$\langle xP_k, P_{k-1} \rangle = \langle P_k, xP_{k-1} \rangle = \langle P_k, \widetilde{P}_k \rangle = \|\widetilde{P}_k\| = b_{k-1}$$

sledi, da velja

$$b_k P_{k+1} = \widetilde{P}_{k+1} = xP_k - a_k P_k - b_{k-1} P_{k-1},$$

kar je ravno formula (8.14). Na podlagi tega lahko razvijemo postopek za izračun ortogonalnih polinomov, ki je predstavljen v algoritmu 8.2. Stranski produkt algoritma so konstante a_k in b_k iz zveze (8.14).

Algoritem 8.2 Algoritem za izračun ortogonalnih polinomov. Začetni podatki so interval $[a, b]$ in utež ρ, ki definirata skalarni produkt $\langle f, g \rangle = \int_a^b f(x)g(x)\rho(x)dx$. Algoritem vrne ortonormirane polinome $P_0, \ldots, P_{n+1}$.

$b_{-1} = \langle 1, 1 \rangle^{1/2}, \quad P_0 = 1/b_{-1}, \quad P_{-1} = 0$
$j = 0, 1, \ldots, n$
  $\widetilde{P} = xP_j$
  $a_j = \langle \widetilde{P}, P_j \rangle$
  $\widetilde{P} = \widetilde{P} - a_j P_j - b_{j-1} P_{j-1}$
  $b_j = \langle \widetilde{P}, \widetilde{P} \rangle^{1/2}$
  $P_{j+1} = (1/b_j)\widetilde{P}$

Če poznamo koeficiente a_k in b_k, potem so vozli $x_0^{(n)}, \ldots, x_n^{(n)}$ lastne vrednosti simetrične tridiagonalne matrike

$$T_n = \begin{bmatrix} a_0 & b_0 & & & \\ b_0 & a_1 & b_1 & & \\ & \ddots & \ddots & \ddots & \\ & & b_{n-2} & a_{n-1} & b_{n-1} \\ & & & b_{n-1} & a_n \end{bmatrix},$$

uteži pa dobimo iz prvih komponent ortonormiranih lastnih vektorjev

$$A_k^{(n)} = x_{1k}^2 \int_a^b \rho(x)dx,$$

kjer je X matrika ortonormiranih lastnih vektorjev matrike T_n.

Najbolj znane družine ortogonalnih polinomov so:[10]

$[-1,1]$,	$\rho(x)=1$,		(Legendre)
$[-1,1]$,	$\rho(x)=(1-x^2)^{-\frac{1}{2}}$,		(Čebišev 1. vrste)
$[-1,1]$,	$\rho(x)=(1-x^2)^{\frac{1}{2}}$,		(Čebišev 2. vrste)
$[-1,1]$,	$\rho(x)=(1-x)^{\alpha}(1+x)^{\beta}$,	$\alpha,\beta>-1$,	(Jacobi)
$[-1,1]$,	$\rho(x)=(1-x^2)^{\sigma-\frac{1}{2}}$,	$\sigma>\frac{1}{2}$,	(Gegenbauer)
$[0,\infty)$,	$\rho(x)=x^{\sigma}e^{-x}$,	$\sigma>-1$	(Laguerre)
$(-\infty,\infty)$,	$\rho(x)=e^{-x^2}$,		(Hermite).

Za te ortogonalne polinome se da poiskati tabelirane vozle in uteži za ustrezne Gaussove kvadraturne formule.

Zgled 8.6 *Gauss–Legendreovi kvadraturni formuli na dveh in treh točkah sta*

$$\int_{-1}^{1} f(x)dx = f\left(-\sqrt{\frac{1}{3}}\right) + f\left(\sqrt{\frac{1}{3}}\right) + \frac{1}{135}f^{(4)}(\xi),$$

$$\int_{-1}^{1} f(x)dx = \frac{5}{9}f\left(-\sqrt{\frac{3}{5}}\right) + \frac{8}{9}f(0) + \frac{5}{9}f\left(\sqrt{\frac{3}{5}}\right) + \frac{1}{15750}f^{(6)}(\xi).$$

Za primerjavo, pri trapeznem in Simpsonovem pravilu, kjer uporabimo isti interval in isto število vozlov, dobimo

$$\int_{-1}^{1} f(x)dx = f(-1) + f(1) - \frac{2}{3}f^{(2)}(\xi),$$

$$\int_{-1}^{1} f(x)dx = \frac{1}{3}f(-1) + \frac{4}{3}f(0) + \frac{1}{3}f(1) + \frac{1}{90}f^{(4)}(\xi). \qquad \square$$

Gaussove kvadraturne formule lahko posplošimo tako, da so nekateri vozli fiksni, ostale pa določimo tako, da bo formula točna za polinome čim višje stopnje. Tako pri Gauss–Lobattovih[11] formulah fiksiramo obe krajišči, pri Gauss–Radaujevih[12] formulah pa eno krajišče.

Gauss–Lobatto: fiksna vozla sta $x_0 = a$ in $x_n = b$, npr.

$$\int_{-1}^{1} f(x)dx = \frac{5}{6}f(-1) + \frac{1}{6}f\left(-\sqrt{\frac{1}{5}}\right) + \frac{1}{6}f\left(\sqrt{\frac{1}{5}}\right) + \frac{5}{6}f(1) + Cf^{(4)}(\xi).$$

Gauss–Radau: fiksen vozel je $x_0 = a$, npr.

$$\int_{-1}^{1} f(x)dx = \frac{2}{9}f(-1) + \frac{16+\sqrt{6}}{18}f\left(\frac{1-\sqrt{6}}{5}\right) + \frac{16-\sqrt{6}}{18}f\left(\frac{1+\sqrt{6}}{5}\right) + Cf^{(5)}(\xi).$$

[10] Francoski matematik Adrien-Marie Legendre (1752–1833) in avstrijski matematik Leopold Gegenbauer (1849–1903).

[11] Nizozemski matematik Rehuel Lobatto (1797-1866).

[12] Francoski astronom in matematik Jean Charles Rodolphe Radau (1835–1911).

8.11 Izlimitirani integrali

Poglejmo si nekaj možnih pristopov pri računanju izlimitiranih integralov, kjer imamo bodisi pol v krajišču ali pa neskončen interval. Pole in morebitne točke nezveznosti v notranjosti intervala moramo poznati že na začetku in potem interval razdeliti na take podintervale, da v nobenem ni pola ali nezveznosti v notranjosti intervala.

Denimo, da računamo

$$I = \int_a^b \frac{g(x)}{(x-a)^p} dx, \quad 0 < p < 1,$$

kjer je funkcija g zvezna na $[a, b]$. Po definiciji je

$$I = \lim_{\epsilon \to 0} \int_{a+\epsilon}^b \frac{g(x)}{(x-a)^p} dx.$$

Ta konvergenca je lahko prepočasna, da bi bila uporabna za praktično računanje. Poleg tega lahko pri formulah, kjer vozli vsebujejo tudi krajišča, pričakujemo velike napake pri računanju vrednosti v levem krajišču, ko bo ϵ blizu 0. Zato iščemo boljše možnosti.

Varianta 1: Integral I razdelimo na $I = I_1 + I_2$, kjer sta

$$I_1 = \int_a^{a+\epsilon} \frac{g(x)}{(x-a)^p} dx, \qquad I_2 = \int_{a+\epsilon}^b \frac{g(x)}{(x-a)^p} dx.$$

Integral I_2 izračunamo s standardnimi metodami, pri računanju integrala I_1 pa funkcijo g razvijemo v Taylorjevo vrsto okoli točke a:

$$g(x) = g(a) + (x-a)g'(a) + \frac{(x-a)^2}{2} g''(a) + \cdots.$$

Tako dobimo

$$I_1 = \epsilon^{1-p} \left(\frac{g(a)}{1-p} + \frac{\epsilon g'(a)}{1!(2-p)} + \frac{\epsilon^2 g''(a)}{2!(3-p)} + \cdots \right).$$

Varianta 2: Funkcijo g razvijemo v Taylorjevo vrsto okoli a:

$$g(x) = P_s(x) + \text{ostanek},$$

kjer je P_s polinom stopnje s. Sedaj I zapišemo kot $I = I_1 + I_2$, kjer sta

$$I_1 = \int_a^b \frac{P_s(x)}{(x-a)^p} dx, \qquad I_2 = \int_a^b \frac{g(x) - P_s(x)}{(x-a)^p} dx.$$

Integral I_1 lahko izračunamo eksplicitno, za računanje integrala I_2 pa uporabimo standardne numerične metode.

Varianta 3: Uporabimo substitucijo, npr.

$$(x-a) = t^m, \quad dx = m t^{m-1} dt, \quad m = \frac{k}{1-p}, \quad k \in \mathbb{N}.$$

Dobimo

$$I = m \int_0^{(b-a)^{1/m}} g(a + t^m) t^{k-1} dt$$

in lahko uporabimo standardne metode, saj smo se znebili pola.

Varianta 4: Pomagamo si z Gaussovimi kvadraturnimi formulami. Npr., če imamo

$$I = \int_{-1}^{1} \frac{g(x)}{\sqrt{1-x^2}}\, dx,$$

potem je utež $\frac{1}{\sqrt{1-x^2}}$ in lahko uporabimo Gauss–Čebiševe kvadraturne formule. Tako lahko npr. pridemo do formule

$$\int_{-1}^{1} \frac{g(x)}{\sqrt{1-x^2}}\, dx = \frac{\pi}{3}\left(g\left(-\frac{\sqrt{3}}{2}\right) + g(0) + g\left(\frac{\sqrt{3}}{2}\right)\right) + Cf^{(6)}(\xi).$$

Pri računanju integrala $I = \int_a^\infty f(x)dx$ na neskončnem intervalu lahko I razdelimo na $I = I_1 + I_2$, kjer sta

$$I_1 = \int_a^b f(x)dx, \qquad I_2 = \int_b^\infty f(x)dx.$$

Integral I_1 izračunamo normalno s standardnimi metodami. Variante za izračun I_2 so

- I_2 zanemarimo, če poznamo kakšno dobro oceno za $|I_2|$.
- Naredimo substitucijo $u = 1/x$ in dobimo integral po končnem intervalu

$$I_2 = -\int_{1/b}^{0} f(1/u)u^{-2}du.$$

Tudi tu si lahko mogoče pomagamo z Gaussovimi kvadraturnimi formulami.

8.12 Večdimenzionalni integrali

Denimo, da računamo integral funkcije f dveh spremenljivk po pravokotniku $\Omega = [a,b] \times [c,d]$. Pišemo lahko

$$\int_\Omega f(x,y)dxdy = \int_a^b dx \int_c^d f(x,y)dy.$$

Če interval $[a,b]$ razdelimo s točkami $x_i = a + ih$, $i = 0,\ldots,n$, $h = (b-a)/n$, interval $[c,d]$ razdelimo s točkami $y_j = c + jk$, $j = 0,\ldots,m$, $k = (d-c)/m$, in za integrale uporabimo trapezno pravilo, dobimo

$$\int_a^b dx \int_c^d f(x,y)dy = \frac{hk}{4}\sum_{i=0}^{n}\sum_{j=0}^{m} A_{ij} f(x_i,y_j) + \mathcal{O}(h^2+k^2),$$

kjer za koeficiente A_{ij} velja $A_{ij} = A_1(i)A_2(j)$ in $A_1(0) = A_1(n) = 1$, $A_1(i) = 2$ za $i \in \{1,\ldots,n-1\}$, ter $A_2(0) = A_2(m) = 1$, $A_2(j) = 2$ za $j \in \{1,\ldots,m-1\}$. V bistvu dobimo tenzorski produkt sestavljenega trapeznega pravila, podobno pa lahko naredimo tudi za ostala sestavljena pravila.

Zgled 8.7 *Če imamo npr. mrežo* 5×4

(x_0, y_3)	(x_1, y_3)	(x_2, y_3)	(x_3, y_3)	(x_4, y_3)
(x_0, y_2)	(x_1, y_2)	(x_2, y_2)	(x_3, y_2)	(x_4, y_2)
(x_0, y_1)	(x_1, y_1)	(x_2, y_1)	(x_3, y_1)	(x_4, y_1)
(x_0, y_0)	(x_1, y_0)	(x_2, y_0)	(x_3, y_0)	(x_4, y_0)

potem so koeficienti pri trapeznem pravilu

$$\frac{hk}{4} \cdot \begin{array}{|c|c|c|c|c|} \hline 1 & 2 & 2 & 2 & 1 \\ \hline 2 & 4 & 4 & 4 & 2 \\ \hline 2 & 4 & 4 & 4 & 2 \\ \hline 1 & 2 & 2 & 2 & 1 \\ \hline \end{array}.$$

Če pa bi npr. uporabili Simpsonovo metodo za mrežo 5×5*, bi dobili*

$$\frac{hk}{9} \cdot \begin{array}{|c|c|c|c|c|} \hline 1 & 4 & 2 & 4 & 1 \\ \hline 2 & 8 & 4 & 8 & 2 \\ \hline 4 & 16 & 8 & 16 & 4 \\ \hline 2 & 8 & 4 & 8 & 2 \\ \hline 1 & 4 & 2 & 4 & 1 \\ \hline \end{array}.$$

Če je $\Omega = [0,1]^d$ in v vsaki dimenziji porabimo n točk, uporabimo pa sestavljeno pravilo reda k, potem za dobljeno tenzorsko pravilo velja, da je napaka reda $\mathcal{O}(N^{-k/d})$, kjer je $N = n^d$ število vseh točk.

Težava je, da pri velikem d prvič potrebujemo zelo veliko točk (n^d), po drugi strani pa napaka prepočasi pada glede na število uporabljenih točk. Zaradi tega se pri velikih n bolje obnaša metoda Monte Carlo, kjer lahko z znosnim številom točk ponavadi dobimo zadovoljiv približek.

8.13 Metoda Monte Carlo

Integral $I(f) = \int_0^1 f(x)dx$ je enak povprečni vrednosti funkcije f na intervalu $[0,1]$. To je osnova za *metodo Monte Carlo*.

Če je X slučajna spremenljivka, enakomerno porazdeljena po $[0,1]$, potem za matematično upanje velja $E(f(X)) = I(f)$. Približek za matematično upanje je

$$I_N(f) = \frac{1}{N} \sum_{i=1}^{N} f(X_i),$$

kjer so X_i neodvisne naključne vrednosti z $[0,1]$.

Za napako $\epsilon_N(f) = I(f) - I_N(f)$ velja

$$\epsilon \approx \sigma(f) N^{-1/2},$$

kjer je $\sigma(f) = \sqrt{D(f)}$ standardna deviacija,

$$D(f) = \int_0^1 (f(x) - I(f))^2 dx$$

pa varianca funkcije f.

Pri računanju integrala po $\Omega = [0,1]^d$ velja enako, le na mestu X_i sedaj izbiramo vektorje iz Ω, kjer je vsak element naključno število med 0 in 1.

Za integral po območju $I^d = [0,1]^d$ dobimo

$$I[f] = E[f(\boldsymbol{X})] = \int_{I^d} f(\boldsymbol{X}) d\boldsymbol{X},$$

kjer je $\boldsymbol{X}$ vektor neodvisnih slučajnih spremenljivk $X_1, \ldots, X_d$, enakomerno porazdeljenih na $[0,1]$. Za napako še vedno velja, da pada z $\mathcal{O}(N^{-1/2})$, neodvisno od d.

Če iz kvadraturne formule reda k s tenzorskimi produkti izpeljemo mrežno kvadraturno formulo za integriranje po I^d, potem pri N točkah napaka pada z $\mathcal{O}(N^{-k/d})$, saj velja $N = n^d$, napaka pa pada kot n^{-k}.

Ostale razlike med mrežnimi kvadraturnimi formulami in metodo Monte Carlo so:

- Mreža pri mrežnih kvadraturnih pravilih je pri velikem d prevelika. Če npr. uporabimo tenzorsko trapezno pravilo, potem potrebujemo vsaj 2^d izračunov vrednosti funkcije.
- Monte Carlo metode so tudi v eni dimenziji primernejše za računanje Fourierovih koeficientov.
- Monte Carlo metoda nima težav z nezveznostmi, ki sicer zmanjšajo red kvadraturnih pravil.

Z računalnikom ne moremo generirati pravih naključnih števil, saj jih vedno generiramo z nekim procesom, ki ga lahko ponovimo. Tako v resnici delamo s psevdonaključnimi števili. Ena izmed preprostih metod za njihovo generiranje je *linearni kongruenčni model*, kjer računamo zaporedje

$$x_{r+1} = ax_r + c \pmod{m},$$

kjer sta a in c določeni pozitivni konstanti, izbiramo pa števila med 0 in $m-1$. Pri tem a, c in m izberemo tako, da je perioda (števila se najkasneje po m korakih začnejo ponavljati) čim večja.

Model ima še to težavo, da če generiramo naključne d-dimenzionalne vektorje, potem točke ležijo na $(d-1)$-dimenzionalnih ravninah, ki jih je približno $m^{1/d}$.

8.14 Numerično integriranje v Matlabu

Na voljo so naslednje funkcije:

- `quad(f,a,b,tol)`: adaptivno Simpsonovo pravilo,
- `quadl(f,a,b,tol)`: adaptivno Gauss–Lobatto–Kronrodovo pravilo na 4+7 točkah,
- `dlbquad(fun,a,b,c,d,tol)`: adaptivno Simpsonovo pravilo na pravokotniku.
- `trapz(x,y)`: sestavljeno trapezno pravilo.

Argumenti so:

- f, fun: funkcija, podana v inline obliki, delovati mora tudi, če je argument vektor,
- a,b,c,d: meje območij ($[a,b]$ oziroma $[a,b] \times [c,d]$),
- tol: relativna natančnost, ko prenehamo rekurzivno deliti podinterval.
- x,y: vektorja točk x_i in $y_i = f(x_i)$.

Primeri uporabe:

- `f=inline('cos(x.^2)'); quad(f,0,1)`
- `f=inline('cos(x.^2)+sin(y.^3)'); dblquad(f,0,1,0,1)`
- `x=linspace(0,pi,100); y=sin(x); trapz(x,y)`

8.15 Numerično seštevanje vrst

Če integral prevedemo na računanje vrste ali pa tudi kako drugače, se srečamo s problemom, ko je potrebno sešteti vrsto

$$S = \sum_{k=1}^{\infty} a_k,$$

za katero vemo, da je konvergentna. Iščemo torej limito zaporedja delnih vsot

$$S_n = \sum_{k=1}^{n} a_k.$$

V številnih primerih lahko z ustrezno transformacijo pohitrimo konvergenco delnih vsot in tako pridemo do dovolj točnega približka z računanjem relativno malo členov vrste.

Prva takšna transformacija je *Aitkenova δ^2-transformacija*. Kadar je zaporedje delnih vsot $S_n = \sum_{k=1}^{n} a_k$ počasi konvergentno in se obnaša podobno kot geometrijska vrsta, potem velikokrat dobimo hitrejše zaporedje z uporabo transformacije

$$T(S_n) = \frac{S_{n+1}S_{n-1} - S_n^2}{S_{n+1} - 2S_n + S_{n-1}}.$$

Transformacijo lahko ponovno uporabimo na novem zaporedju $T(S_n)$. Metoda se obnaša dobro pri konvergentnih alternirajočih zaporedjih.

Posplošitev Aitkenove transformacije je *Wynnova ϵ-metoda*

$$\epsilon_{r+1}(S_n) = \epsilon_{r-1}(S_n) + \frac{1}{\epsilon_r(S_{n+1}) - \epsilon_r(S_n)},$$

kjer vzamemo $\epsilon_0(S_n) = S_n$ in $\epsilon_{-1}(S_n) = 0$. Metoda se obnaša dobro pri konvergentnih alternirajočih zaporedjih.

Pri *Eulerjevi transformaciji* konvergentno alternirajočo vrsto

$$\sum_{k=0}^{\infty}(-1)^k a_k = a_0 - a_1 + a_2 - a_3 + \cdots$$

spremenimo v hitreje konvergentno vrsto z isto vsoto

$$\sum_{k=0}^{\infty}\frac{(-1)^k \Delta^k a_0}{2^{k+1}},$$

kjer je

$$\Delta^k a_0 = \sum_{j=0}^{k}(-1)^j \binom{k}{j} a_{k-j}$$

k-ta prema diferenca a_0.

Pomagamo si lahko tudi z Euler–Maclaurinovo formulo, ki smo jo uporabili pri razvoju Rombergove metode. Pri računanju določenega integrala vrednost le tega aproksimiramo s trapezno formulo. Če pa zamenjamo, lahko vrednost vsote za trapezno formulo aproksimiramo z integralom in preostalimi členi v obliki

$$\sum_{k=1}^{N} f(k) = \int_1^N f(x)dx + \frac{1}{2}(f(N+1)+f(1)) + \sum_{k=1}^{m} B_{2k}\frac{f^{(2k-1)}(N+1) - f^{(2k-1)}(1)}{(2k)!} + R_m,$$

kjer so B_m Bernoullijeva števila. Za napako velja

$$R_m = -N \cdot B_{2m+2}\frac{f^{(2m+2)}(\xi)}{(2m+2)!}.$$

Tako dobimo

$$\sum_{k=1}^{N-1} f(k) = \int_1^N f(x)dx - \frac{1}{2}(f(0)-f(N)) + \frac{1}{12}(f'(N)-f'(0))$$
$$-\frac{1}{720}\left(f^{(3)}(N)-f^{(3)}(0)\right) + \frac{1}{302240}\left(f^{(5)}(N)-f^{(5)}(0)\right) - \frac{1}{1209600}\left(f^{(7)}(N)-f^{(7)}(0)\right).$$

Tako lahko npr. izpeljemo formulo za $\sum_{k=1}^{n} k^4$. Dobimo

$$\sum_{k=1}^{n} k^4 = \frac{1}{5}n^5 + \frac{1}{2}n^4 + \frac{1}{12}(4n^3) - \frac{1}{720}(24n),$$

višji odvodi pa so enaki 0. Ostane

$$\sum_{k=1}^{n} k^4 = \frac{1}{30}n(n+1)(2n+1)(3n^2+2n-1).$$

Dodatna literatura

Numerično integriranje je obravnavano skoraj v vseh učbenikih numerične analize. V slovenščini lahko več o tem preberete v knjigi [18], osnovne informacije so tudi v knjigah [3] in [34], od tuje literature pa omenimo [6] in [17].

Poglavje 9

Diferencialne enačbe

9.1 Uvod

Rešujemo *začetni problem prvega reda* v obliki

$$\begin{aligned} y' &= f(x,y) \\ y(x_0) &= y_0, \end{aligned}$$

kjer je f dana dovolj gladka funkcija, zanima pa nas rešitev na intervalu $[x_0, b]$. Želimo, da je problem dobro definiran, kar pomeni, da rešitev obstaja in je enolična.

Zgled 9.1 *Začetni problem $y' = 1 + y^2$, $y(0) = 0$ ima analitično rešitev $y(x) = \tan x$. Rešitev desno od $x = 0$ vedno bolj strmo narašča, v levo stran pa pada, in pri končni vrednosti x (pri $\pm\pi/2$) pridemo do $\pm\infty$. Torej rešitev obstaja le na intervalu $(-\pi/2, \pi/2)$. Ta zgled kaže, da se lahko zgodi, da rešitev začetnega problema ni definirana na celotnem intervalu $[x_0, b]$.* □

Zgled 9.2 *Začetni problem $y' = y^{2/3}$, $y(0) = 0$ ima dve rešitvi, saj tako trivialna rešitev $y_1(x) = 0$ kot $y_2(x) = x^3/27$ ustrezata tako diferencialni enačbi kot začetnemu pogoju. To je zgled začetnega problema, ki nima enolične rešitve.* □

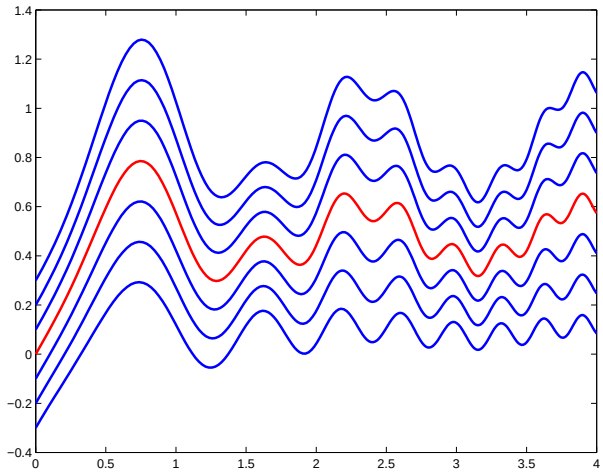
Numerična rešitev bo sestavljena iz zaporedja točk $x_0 < x_1 < x_2 < \ldots$ in pripadajočega zaporedja vrednosti $y_0, y_1, y_2, \ldots$, kjer je y_n izračunani približek za rešitev začetnega problema v x_n, oziroma

$$y_n \approx y(x_n), \quad n = 0, 1 \ldots,$$

kjer je $y(x_n)$ točna vrednost rešitve v x_n. Pri tem sodobne metode delujejo na adaptivnem principu in avtomatično določajo velikosti korakov $h_i = x_{i+1} - x_i$ tako, da napake približkov y_n ostanejo v vnaprej določenih mejah.

Metode za reševanje začetnega problema delimo na:

- *enokoračne metode*: y_{n+1} izračunamo iz y_n.
- *večkoračne metode*: y_{n+1} izračunamo iz $y_n, y_{n-1}, \ldots, y_{n-k+1}$, kjer je $k \geq 1$.



Slika 9.1: Rešitev diferencialne enačbe $y' = \cos(3x^2) + \sin(4x)y$ pri različnih začetnih pogojih $y(0) = -0.3, -0.2, \ldots, 0.3$.

Metode ločimo še na:

- *eksplicitne metode*: imamo direktno formulo za y_{n+1},
- *implicitne metode*: y_{n+1} dobimo tako, da rešimo nelinearno enačbo.

Slika 9.1 prikazuje tipično obnašanje rešitev diferencialne enačbe pri različnih začetnih pogojih. Vsakemu začetnemu pogoju pripada svoja veja. Težava pri reševanju začetnega problema je, da je veja, ki ji želimo slediti, določena z vrednostjo v začetni točki. Ko pa med računanjem zaidemo s prave veje, nimamo več nobene informacije, ki bi nas vrnila nanjo, saj se metode obnašajo tako, kot da bi problem začeli reševati znova iz zadnje točke.

Sistem diferencialnih enačb prvega reda

$$\begin{aligned} y_1' &= f_1(x, y_1, y_2, \ldots, y_k) \\ y_2' &= f_2(x, y_1, y_2, \ldots, y_k) \\ &\vdots \\ y_k' &= f_k(x, y_1, y_2, \ldots, y_k) \end{aligned}$$

z začetnimi pogoji

$$y_1(x_0) = y_{10},\ y_2(x_0) = y_{20},\ \ldots,\ y_k(x_0) = y_{k0}$$

rešujemo tako kot eno enačbo, če ga zapišemo v vektorski obliki

$$Y' = F(x, Y),$$

kjer je $Y = [y_1\ y_2\ \cdots\ y_k]^T$. Zaradi tega lahko vse metode, ki jih bomo spoznali v nadaljevanju, uporabljamo tako za eno samo enačbo kot tudi za sistem diferencialnih enačb.

9.2 Obstoj rešitve, občutljivost in stabilnost

V tem razdelku bomo na kratko ponovili teorijo s področja diferencialnih enačb, ki jo bomo potrebovali v nadaljevanju. Zanima nas, kdaj imamo zagotovljen obstoj enolične rešitve, za samo numerično reševanje pa sta pomembni tudi občutljivost problema in stabilnost rešitve.

Funkcija $f(x,y)$, kjer je $f : \mathbb{R}^{k+1} \to \mathbb{R}^k$, na območju $D = [a,b] \times \Omega \subset \mathbb{R}^{k+1}$ *zadošča Lipschitzovemu pogoju* na y s konstanto L, če za poljubna $(x,y_1),(x,y_2) \in D$ velja

$$\|f(x,y_1) - f(x,y_2)\| \leq L\|y_1 - y_2\|.$$

Zadostni pogoj, da f zadošča Lipschitzovemu pogoju, je, da je f zvezno odvedljiva po y, saj potem lahko vzamemo

$$L = \max_{(x,y)\in D} \|Jf_y(x,y)\|,$$

kjer je Jf_y $k \times k$ Jacobijeva matrika f glede na y:

$$Jf_y(x,y) = \begin{bmatrix} \frac{\partial f_1}{\partial y_1}(x,y) & \cdots & \frac{\partial f_1}{\partial y_k}(x,y) \\ \vdots & & \vdots \\ \frac{\partial f_k}{\partial y_1}(x,y) & \cdots & \frac{\partial f_k}{\partial y_k}(x,y) \end{bmatrix}.$$

V primeru navadne diferencialne enačbe, kjer je $k = 1$, je $Jf_y(x,y)$ kar $f_y(x,y)$.

Če f ustreza Lipschitzovemu pogoju, potem za vsak $(x_0,y_0) \in D$ obstaja podinterval $[a,b]$, ki vsebuje x_0, na katerem rešitev začetnega problema $y' = f(x,y)$, $y(x_0) = y_0$ obstaja in je enolična.

9.2.1 Občutljivost rešitve začetnega problema

Pri občutljivosti nas zanima, kako se spremeni rešitev, če se spremenijo začetni podatki. Tu bomo ločili dva primera.

V prvem imamo motnjo samo v začetnem pogoju, kar ustreza temu, da rešujemo isto diferencialno enačbo, a smo na drugi veji. To se dogaja med numeričnim reševanjem začetnega problema, saj se v vsakem koraku zaradi lokalne napake premaknemo na drugo vejo. Denimo, da f ustreza Lipschitzovemu pogoju. Točna rešitev začetnega problema $y' = f(x,y)$, $y(x_0) = y_0$ naj bo $y(x)$. Če je $\widetilde{y}(x)$ rešitev začetnega problema z zmotenim začetnim pogojem $y' = f(x,y)$, $y(x_0) = \widetilde{y}_0$, potem za poljuben $x \geq x_0$ velja

$$\|\widetilde{y}(x) - y(x)\| \leq e^{L(x-x_0)}\|\widetilde{y}_0 - y_0\|.$$

V drugem primeru obravnavamo tako motnjo v začetnem pogoju kot tudi motnjo v diferencialni enačbi. Tu nas zanima, kaj se zgodi, če npr. zapleteno diferencialno enačbo nadomestimo z lažje izračunljivo aproksimacijo. Če namesto začetnega problema $y' = f(x,y)$, $y(x_0) = y_0$, rešujemo bližnji problem $\widehat{y}' = \widehat{f}(x,\widehat{y})$, $\widehat{f}(x_0) = \widehat{y}_0$, potem velja

$$\|\widehat{y}(x) - y(x)\| \leq e^{L(x-x_0)}\|\widehat{y}_0 - y_0\| + \frac{e^{L(x-x_0)} - 1}{L}\|\widehat{f} - f\|,$$

kjer je $\|\widehat{f} - f\| = \max_{(x,y)\in D} \|\widehat{f}(x,y) - f(x,y)\|$,

9.2.2 Stabilnost rešitve začetnega problema

Ponavadi smo stabilnost omenjali v povezavi z numeričnimi metodami. Pri diferencialnih enačbah pa govorimo o stabilnosti rešitve problema, ki ni povezana z numeričnimi metodami.

Definicija 9.1 *Pravimo, da je rešitev začetnega problema* $y' = f(x,y)$, $y(x_0) = y_0$ stabilna, *če za vsak* $\epsilon > 0$ *obstaja tak* $\delta > 0$, *da za* $\widetilde{y}(x)$, *ki je rešitev* $\widetilde{y}' = f(x,\widetilde{y})$, $\widetilde{y}(x_0) = \widetilde{y}_0$, *kjer je* $\|y_0 - \widetilde{y}_0\| \leq \delta$, *velja* $\|\widetilde{y}(x) - y(x)\| \leq \epsilon$ *za vse* $x \geq x_0$.

Če ima začetni problem stabilno rešitev, lahko pričakujemo, da bo rešitev pri zmotenem začetnem pogoju ostala blizu točne rešitve.

Pravimo, da je stabilna rešitev *asimptotično stabilna*, če gre $\|\widetilde{y}(x) - y(x)\|$ proti 0, ko gre x proti neskončnosti. Pri asimptotični rešitvi lahko pričakujemo, da bo napaka zmotenega začetnega pogoja šla proti 0, ko gre x proti neskončnosti.

Zgled 9.3 *Rešitev diferencialne enačbe* $y'(x) = \lambda y$, $y(0) = y_0$, *kjer je* $\lambda = a + ib \in \mathbb{C}$, *je*

$$y(x) = y_0 e^{\lambda x} = y_0 e^{ax}(\cos(bx) + i\sin(bx)).$$

Stabilnost rešitve je odvisna od a. Rešitev je asimptotično stabilna pri $\mathrm{Re}(\lambda) < 0$, *stabilna pri* $\mathrm{Re}(\lambda) = 0$, *in nestabilna pri* $\mathrm{Re}(\lambda) > 0$. □

Zgled 9.4 *Imamo sistem* $y' = Ay$, $y_0 = y_0$, *kjer je A matrika* $k \times k$. *Če se da A diagonalizirati in so njene lastne vrednosti* $\lambda_1, \ldots, \lambda_k$ *z lastnimi vektorji* $v_1, \ldots, v_k$, *potem je rešitev*

$$y(x) = \sum_{i=1}^{k} \alpha_i e^{\lambda_i x} v_i,$$

kjer so koeficienti α_i *določeni z*

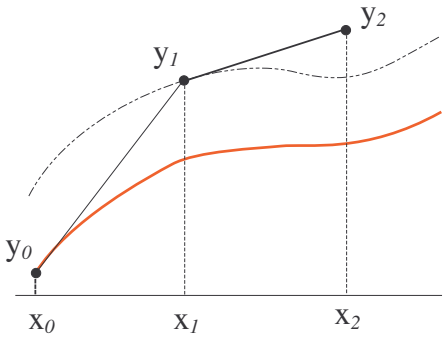
$$y_0 = \sum_{i=1}^{k} \alpha_i v_i.$$

Rešitev je asimptotično stabilna, če za vse lastne vrednosti velja $\mathrm{Re}(\lambda_i) < 0$; *stabilna, če velja* $\mathrm{Re}(\lambda_i) \leq 0$ *za vse lastne vrednosti; in nestabilna, če obstaja lastna vrednost* λ_i, *kjer je* $\mathrm{Re}(\lambda_i) > 0$. □

9.3 Enokoračne metode

9.3.1 Eulerjeva metoda

V prvem koraku reševanja začetnega problema $y' = f(x,y)$, $y(x_0) = y_0$ se moramo iz točke (x_0, y_0) premakniti v točko (x_1, y_1), kjer je y_1 približek za vrednost $y(x_1)$ in $x_1 = x_0 + h$. Poleg vrednosti (x_0, y_0) lahko v tej točki iz diferencialne enačbe izračunamo še smer tangente $y'_0 = f(x_0, y_0)$. Če je h dovolj majhen, ne bomo naredili velike napake, če bomo rešitev na intervalu $[x_0, x_1]$ aproksimirali s premico in se premaknili v smeri tangente.



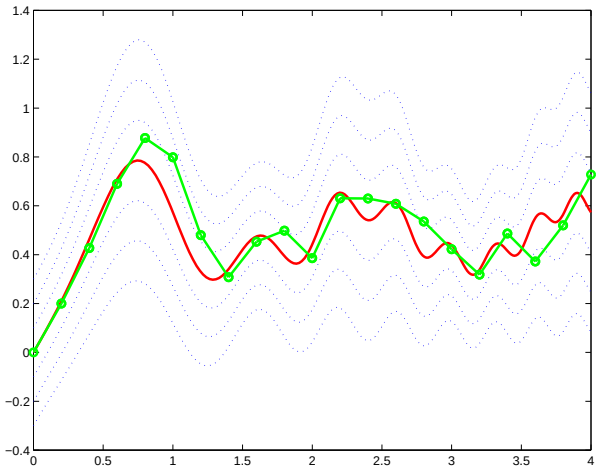
Slika 9.2: Eksplicitna Eulerjeva metoda.

Tako pridemo do najpreprostejše enokoračne metode, ki se imenuje *eksplicitna Eulerjeva metoda*. Nastavek za en korak je

$$\begin{aligned} y_{n+1} &= y_n + hf(x_n, y_n) \\ x_{n+1} &= x_n + h. \end{aligned}$$

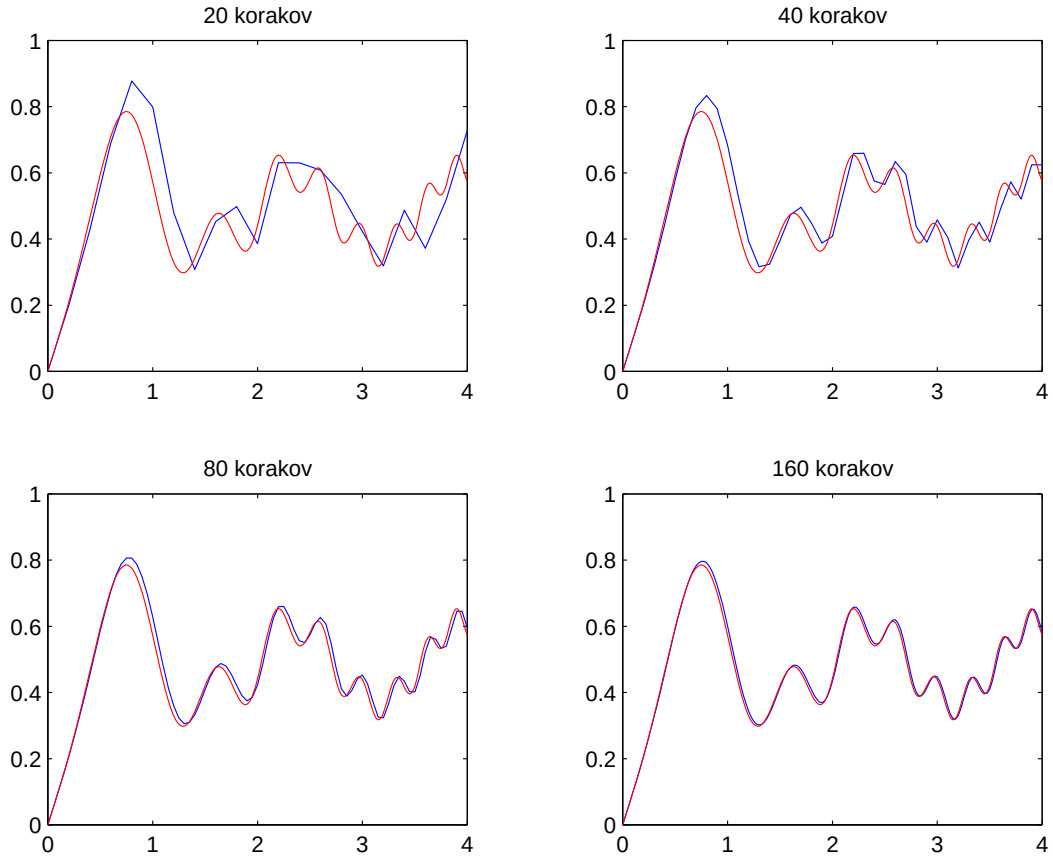
Metoda je prikazana na sliki 9.2. V vsaki točki se premaknemo v smeri tangente. Ponavadi uporabljamo fiksen h, lahko pa tudi velikost h v vsakem koraku zmanjšamo ali povečamo odvisno od obnašanja rešitve.

Zgled 9.5 *Oglejmo si delovanje eksplicitne Eulerjeve metode na primeru diferencialne enačbe* $y' = \cos(3x^2) + \sin(4x)y$, $y(0) = 1$. *Denimo, da nas zanima rešitev na intervalu* $[0, 4]$.



Slika 9.3: Delovanje eksplicitne Eulerjeve metode na enačbi $y' = \cos(3x^2) + \sin(4x)y$, $y(0) = 1$ na intervalu $[0, 4]$ s premikom $h = 0.2$.

Če vzamemo $h = 0.2$ *in uporabimo eksplicitno Eulerjevo metodo, potem je graf numerično dobljene rešitve na sliki 9.3 predstavljen z zeleno barvo, medtem, ko je točna rešitev pobarvana rdeče. Vidimo, da je premik* $h = 0.2$ *prevelik, da bi rešitev lahko sledila vseh lokalnim ekstremom, ki jih ima točna rešitev.*



Slika 9.4: Delovanje eksplicitne Eulerjeve metode na enačbi $y' = \cos(3x^2) + \sin(4x)y$, $y(0) = 1$ na intervalu $[0, 4]$ s premiki $h = 0.2, 0.1, 0.05, 0.025$.

Če zmanjšamo premik h se poveča čas računanja, a kot kaže slika 9.4, se z manjšanjem h povečuje natančnost dobljene rešitve. Ko gre h proti 0, numerična rešitev konvergira proti točni rešitvi. □

Poleg eksplicitne poznamo tudi *implicitno Eulerjevo metodo*, ki je podana z

$$\begin{aligned} y_{n+1} &= y_n + hf(x_{n+1}, y_{n+1}) \\ x_{n+1} &= x_n + h. \end{aligned}$$

Pri implicitni metodi rešitev na intervalu $[x_n, x_{n+1}]$ še vedno aproksimiramo s premico, le da tokrat za smerni koeficient vzamemo vrednost odvoda v točki (x_{n+1}, y_{n+1}). Ker y_{n+1} nastopa na obeh straneh enačbe, je potrebno pri implicitni metodi v vsakem koraku rešiti nelinearni sistem za y_{n+1}.

9.3.2 Taylorjeva vrsta

Pri eksplicitni Eulerjevi metodi smo rešitev aproksimirali s tangento. Če bi imeli na voljo še višje odvode, potem bi lahko uporabili razvoj v Taylorjevo vrsto in rešitev aproksimirali s polinomom stopnje višje od ena.

Za razvoj v Taylorjevo vrsto potrebujemo višje odvode y. Dobimo jih tako, da odvajamo diferencialno enačbo. Če predpostavimo, da je f dovoljkrat zvezno odvedljiva, potem z odvajanjem enačbe $y' = f(x, y)$ dobimo

$$\begin{aligned} y' &= f \\ y'' &= f_x + f_y y' = f_x + f_y f \\ y''' &= f_{xx} + 2f_{xy}f + f_{yy}f^2 + f_y(f_x + f_y f) \\ &\vdots \end{aligned}$$

Preko razvoja v Taylorjevo vrsto lahko potem izračunamo

$$y(x_1) = y(x_0 + h) = y_0 + hy_0' + \frac{h^2}{2}y_0'' + \cdots.$$

Zgled 9.6 *Z metodo razvoja v Taylorjevo vrsto bomo naredili en korak za začetni problem* $y' = xy + 1$, $y(0) = 0$. *Vzeli bomo* $h = 0.2$ *in uporabili prve štiri člene razvoja v Taylorjevo vrsto.*

Z odvajanjem diferencialne enačbe dobimo

$$\begin{aligned} y' = xy + 1 &\implies y'(0) = 1 \\ y'' = xy' + y &\implies y''(0) = 0 \\ y''' = xy'' + 2y' &\implies y'''(0) = 2 \end{aligned}$$

$$y(h) = h + \frac{1}{3}h^3 + \cdots.$$

Pri $h = 0.2$ *tako dobimo*

$$y_1 = 0.2 + 0.00267 = 0.20267.$$

Če iščemo $y(0.4)$, *nadaljujemo iz točke* $(0.2, 0.20267)$ *in ponovimo postopek.* □

Očitno bo napaka izračunane rešitve manjša, če uporabimo več členov razvoja v Taylorjevo vrsto. Na osnovi tega definiramo lokalno napako metode.

Definicija 9.2 *Pravimo, da ima metoda* lokalno napako *reda k, če se pri točni vrednosti* $y_n = y(x_n)$ *izračunani* y_{n+1} *ujema z razvojem* $y(x_n + h)$ *v Taylorjevo vrsto okrog* x_n *do vključno člena* h^k.

Lokalna napaka predstavlja razliko, ki nastopi v enem samem koraku. Pove nam, koliko se naslednji približek razlikuje od vrednosti na veji, kjer smo pričeli z računanjem, torej točki (x_n, y_n).

Metoda iz zadnjega zgleda ima red 3, sicer pa z razvijanjem v Taylorjevo vrsto lahko dobimo metodo poljubnega reda. Eulerjeva metoda (tako eksplicitna kot implicitna) ima red 1.

9.3.3 Runge-Kutta metode

Runge-Kutta metode so še vedno enokoračne. Da določimo čim natančnejši premik, izračunamo vrednost funkcije f v m točkah. Tako dobimo m premikov v smereh tangent, na koncu pa sestavimo premik iz uteženega povprečja dobljenih premikov.

Najprej izračunamo m koeficientov

$$k_i = hf(x_n + \alpha_i h, y_n + \sum_{j=1}^{i} \beta_{ij} k_j), \quad i = 1, \ldots, m,$$

nato pa je naslednji približek enak

$$y_{n+1} = y_n + \sum_{i=1}^{m} \gamma_i k_i.$$

Pri tem je m stopnja Runge-Kutta metode, ki je ne smemo zamenjevati z redom metode. Konstante α_i, β_{ij} in γ_i določimo tako, da se y_{n+1} čim bolje ujema z razvojem $y(x_n + h)$ v Taylorjevo vrsto. Izkaže se, da mora veljati $\alpha_i = \sum_{j=1}^{i} \beta_{ij}$ in $\sum_{i=1}^{m} \gamma_i = 1$.

V primeru, ko je $\beta_{ii} = 0$ za $i = 1, \ldots, m$, je metoda eksplicitna, sicer pa implicitna.

Zgled 9.7 *Dvostopenjska eksplicitna Runge-Kutta metoda ima obliko*

$$\begin{aligned} k_1 &= hf(x_n, y_n) \\ k_2 &= hf(x_n + \alpha h, y_n + \beta k_1) \\ y_{n+1} &= y_n + \gamma_1 k_1 + \gamma_2 k_2. \end{aligned}$$

S pomočjo razvoja funkcije f v Taylorjevo vrsto kot funkcijo dveh spremenljivk dobimo razvoja

$$\begin{aligned} k_1 &= hf \\ k_2 &= hf + \alpha h^2 f_x + \beta h k_1 f_y + \mathcal{O}(h^3). \end{aligned}$$

Od tod sledi, da za naslednji približek y_{n+1} vzamemo

$$y_{n+1} = y_n + (\gamma_1 + \gamma_2) hf + \gamma_2 \alpha h^2 f_x + \gamma_2 \beta h^2 f f_y + \mathcal{O}(h^3),$$

kar primerjamo z

$$y(x_n + h) = y(x_n) + hf + \frac{1}{2} h^2 (f_x + f f_y) + \mathcal{O}(h^3).$$

Sledi

$$\begin{aligned} \gamma_1 + \gamma_2 &= 1 \\ \alpha \gamma_2 &= \tfrac{1}{2} \\ \beta \gamma_2 &= \tfrac{1}{2}. \end{aligned}$$

Sistem ima več rešitev, saj za poljubni $\gamma_2 \neq 0$ dobimo

$$\gamma_1 = 1 - \gamma_2, \quad \alpha = \frac{1}{2\gamma_2}, \quad \beta = \frac{1}{2\gamma_2}.$$

□

Primera ekplicitne dvostopenjske Runge-Kutta metode drugega reda sta:

- *Heunova metoda*

$$\begin{aligned} k_1 &= hf(x_n, y_n) \\ k_2 &= hf(x_n + h, y_n + k_1) \\ y_{n+1} &= y_n + \tfrac{1}{2}(k_1 + k_2), \quad \text{lokalna napaka : } \mathcal{O}(h^3); \end{aligned}$$

- *modificirana Eulerjeva metoda*

$$\begin{aligned} k_1 &= hf(x_n, y_n) \\ k_2 &= hf(x_n + \tfrac{1}{2}h, y_n + \tfrac{1}{2}k_1) \\ y_{n+1} &= y_n + k_2, \quad \text{lokalna napaka : } \mathcal{O}(h^3). \end{aligned}$$

Zelo znana je tudi Runge-Kutta 4-stopenjska metoda reda 4, kjer vzamemo

$$\begin{aligned} k_1 &= hf(x_n, y_n) \\ k_2 &= hf(x_n + \tfrac{1}{2}h, y_n + \tfrac{1}{2}k_1) \\ k_3 &= hf(x_n + \tfrac{1}{2}h, y_n + \tfrac{1}{2}k_2) \\ k_4 &= hf(x_n + h, y_n + k_3) \\ y_{n+1} &= y_n + \tfrac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4), \quad \text{lokalna napaka : } \mathcal{O}(h^5). \end{aligned}$$

Zgled 9.8 *Denimo, da s 4-stopenjsko Runge-Kutta metodo rešujemo* $y' = -y - 5e^x \sin(x)$, $y(0) = 1$, *kjer vzamemo* $h = 0.1$. *Dobimo*

$$\begin{aligned} k_1 &= hf(x_0, y_0) = 0.1 * f(0, 1) = -0.1 \\ k_2 &= hf(x_0 + \tfrac{1}{2}h, y_0 + \tfrac{1}{2}k_1) = 0.1 * f(0.05, 0.95) = -0.12127 \\ k_3 &= hf(x_0 + \tfrac{1}{2}h, y_0 + \tfrac{1}{2}k_2) = 0.1 * f(0.05, 0.93936) = -0.12021 \\ k_4 &= hf(x_0 + h, y_0 + k_3) = 0.1 * f(0.1, 0.87979) = -0.14315 \end{aligned}$$

$$y_1 = y_0 + \tfrac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) = 0.87898. \qquad \square$$

Izkaže se, da je pri eksplicitnih Runge-Kutta metodah stopnje 5 ali več red vedno manjši od stopnje. Tako je npr. maksimalni red 5-stopenjske eksplicitne Runge-Kutta metoda enak 4, 6-stopenjske pa 5.

9.3.4 Adaptivna ocena koraka

Če imamo na voljo oceno lokalne napake, lahko h adaptivno prilagajamo. Denimo, da po 4-stopenjski Runge-Kutta metodi reda 4 iz $y(x)$ izračunamo $y(x + 2h)$ enkrat s korakom $2h$, drugič pa v dveh korakih s korakom h. Podobno kot pri Richardsonovi ekstrapolaciji dobimo

$$\begin{aligned} y(x + 2h) &= y^{(1)} + (2h)^5 \cdot C_1 + \mathcal{O}(h^6) \\ y(x + 2h) &= y^{(2)} + 2(h)^5 \cdot C_2 + \mathcal{O}(h^6) \end{aligned}$$

in

$$\Delta = \frac{y^{(2)} - y^{(1)}}{15}$$

je ocena za lokalno napako približka $y^{(1)}$. Ko je ocena Δ velika h zmanjšamo, če pa je ocena dovolj majhna lahko h povečamo. Za izračun Δ in $y^{(1)}$ potrebujemo 11 izračunov vrednosti funkcije f (4 izračune za vsako Runge-Kutta metodo, pri čemer se izračun $f(x_n, y_n)$ ponovi dvakrat).

Boljša je *Runge-Kutta–Fehlbergova metoda*, kjer vzamemo 6 stopenjsko Runge-Kutta metodo

$$k_i = hf(x_n + \alpha_i h, y_n + \sum_{j=1}^{i-1} \beta_{ij} k_j), \quad i = 1, \ldots, 6,$$

potem pa iz istih $k_1, \ldots, k_6$ sestavimo metodo reda 4

$$y_{n+1} = y_n + \sum_{i=1}^{6} \gamma_i k_i$$

in kontrolno metodo reda 5

$$y^*_{n+1} = y_n + \sum_{i=1}^{6} \gamma^*_i k_i.$$

Ocena za napako y_{n+1} je potem kar $y_{n+1} - y^*_{n+1} = \sum_{i=1}^{6} (\gamma_i - \gamma^*_i) k_i$.

Podrobne formule za Runge-Kutta–Fehlbergovo metodo so

$$\begin{aligned}
k_1 &= hf(x_n, y_n) \\
k_2 &= hf(x_n + \tfrac{1}{4}h, y_n + \tfrac{1}{4}k_1) \\
k_3 &= hf(x_n + \tfrac{3}{8}h, y_n + \tfrac{3}{32}k_1 + \tfrac{9}{32}k_2) \\
k_4 &= hf(x_n + \tfrac{12}{13}h, y_n + \tfrac{1932}{2197}k_1 - \tfrac{7200}{2197}k_2 + \tfrac{7296}{2197}k_3) \\
k_5 &= hf(x_n + h, y_n + \tfrac{439}{216}k_1 - 8k_2 + \tfrac{3680}{513}k_3 - \tfrac{845}{4104}k_4) \\
k_6 &= hf(x_n + \tfrac{1}{2}h, y_n - \tfrac{8}{27}k_1 + 2k_2 - \tfrac{3544}{2565}k_3 + \tfrac{1859}{4104}k_4 - \tfrac{11}{40}k_5) \\
y_{n+1} &= y_n + \tfrac{25}{216}k_1 + \tfrac{1408}{2565}k_3 + \tfrac{2197}{4104}k_4 - \tfrac{1}{5}k_5, \quad \text{lokalna napaka : } \mathcal{O}(h^5) \\
y^*_{n+1} &= y_n + \tfrac{16}{135}k_1 + \tfrac{6656}{12825}k_3 + \tfrac{28561}{56430}k_4 - \tfrac{9}{50} + \tfrac{2}{55}k_6, \quad \text{lokalna napaka : } \mathcal{O}(h^6).
\end{aligned}$$

Opazimo lahko, da za izračun y_{n+1} potrebujemo le koeficiente $k_1, \ldots, k_5$, torej je vrednost y_{n+1} v bistvu dobljena s 5-stopenjsko metodo. Koeficient k_6 potrebujemo le za izračun kontrolne vrednosti y^*_{n+1}.

V naslednjem koraku vzamemo razmik qh, kjer je ϵ željena natančnost in

$$q = \left(\frac{\epsilon h}{2|y^*_{n+1} - y_{n+1}|} \right)^{1/4}.$$

Potrebno je poudariti, da pri Runge-Kutta–Fehlbergovi metodi za naslednji približek vzamemo y_{n+1}, kljub temu, da imamo na voljo po vsej verjetnosti natančnejši približek y^*_{n+1}. To naredimo zaradi tega, ker iz razlike $y_{n+1} - y^*_{n+1} = \mathcal{O}(h^5)$ lahko ocenimo vodilni člen lokalne napake y_{n+1}, tega pa ne moremo narediti za y^*_{n+1}. Tako ima Runge-Kutta–Fehlbergova metoda red 4.

9.4 Stabilnost in konvergenca enokoračnih metod

Numerične metode so primerne za uporabo le, če so stabilne in konvergentne. V tem razdelku bomo pokazali, kdaj lahko ti lastnosti predpostavimo za enokoračne metode. Navedli bomo le definicije in glavne izreke. Podrobnosti skupaj z dokazi izrekov lahko najdete v [18].

Vsako enokoračno metodo lahko zapišemo v obliki

$$y_{n+1} = y_n + h\phi(x_n, y_n, h),$$

kjer je ϕ *funkcija prirastka*.

Definicija 9.3 *Enokoračna metoda za numerično reševanje začetnega problema je* konsistentna, *če velja*

$$\lim_{h\to 0} \phi(x, y, h) = f(x, y).$$

Zgled 9.9 *Funkcija prirastka za modificirano Eulerjevo metodo*

$$y_{n+1} = y_n + hf(x_n + \frac{h}{2}, y_n + \frac{1}{2}hf(x_n, y_n))$$

je $\phi(x_n, y_n, h) = f(x_n + \frac{h}{2}, y_n + \frac{1}{2}hf(x_n, y_n))$. *Metoda je očitno konsistentna.* □

Lokalno napako pri izračunu y_{n+1} lahko zapišemo kot

$$T(x_{n+1}) = y(x_{n+1}) - y(x_n) - h\phi(x_n, y_n, h).$$

Če velja $T(x_{n+1}) = \mathcal{O}(h^{p+1})$, potem je metoda reda p.

Numerična metoda za reševanje začetnega problema je stabilna, kadar majhne motnje ne povzročijo divergence numerične rešitve.

Definicija 9.4 *Denimo, da iščemo rešitev začetnega problema na intervalu* $[a, b]$, *ki ga razdelimo z ekvidistantnimi točkami* $x_i = a + ih$, *kjer je* $h = (b - a)/n$ *in* $i = 0, \ldots, n$. *Pravimo, da je enokoračna metoda* stabilna, *če za vsako diferencialno enačbo, ki zadošča Lipschitzovemu pogoju, obstajata taki konstanti* $h_0 > 0$ *in* $K > 0$, *da za poljubni dve rešitvi* y_r, $\widetilde{y}_r$, *ki ju dobimo z* $0 < h \leq h_0$, *velja* $\|y_r - \widetilde{y}_r\| \leq K\|y_0 - \widetilde{y}_0\|$ *za* $x_r \in [a, b]$.

Stabilnost numerične metode ni povezana s stabilnostjo rešitve diferencialne enačbe, ki smo jo definirali v podrazdelku 9.2.2.

Izrek 9.5 *Če funkcija prirastka* ϕ *zadošča Lipschitzovemu pogoju na* y, *potem je metoda stabilna.*

Definicija 9.6 *Enokoračna metoda je* konvergentna, *če za vsako diferencialno enačbo, ki zadošča Lipschitzovemu pogoju, za vsak r velja*

$$\lim_{h\to 0} \|y_r - y(x_r)\| = 0.$$

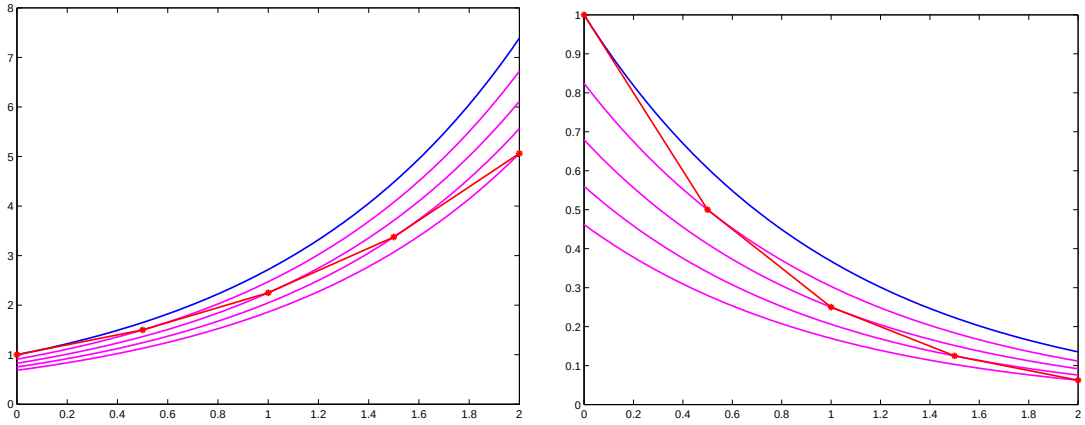
Pri tem predpostavimo, da v limiti upoštevamo le tako majhne h, da je $rh \leq |b - a|$.

Izrek 9.7 *Če je funkcija prirastka ϕ zvezna v x, y, h in zadošča Lipschitzovemu pogoju na y, potem je metoda konvergentna natanko tedaj, ko je metoda konsistentna.*

Izrek 9.8 (O globalni napaki) *Če funkcija prirastka ϕ zadošča pogojem za konvergenco in za lokalno napako velja $\|T(x)\| \leq Dh^{p+1}$, potem za globalno napako velja*

$$\|y_n - y(x_n)\| \leq Dh^p \frac{e^{L(x_n - x_0)} - 1}{L} + e^{L(x_n - x_0)} \|y_0 - y(x_0)\|.$$

Globalna napaka ni preprosto kar vsota lokalnih napak, v grobem pa velja, da ima metoda z lokalno napako reda k globalno napako reda $k-1$. Primera na sliki 9.5 prikazujeta dva primera povezave globalne in lokalne napake. V prvem primeru (levo) je globalna napaka večja od vsote absolutnih vrednosti lokalnih napak. V desnem primeru pa je globalna napaka manjša od posameznih lokalnih napak.



Slika 9.5: Povezava med globalno in lokalno napako.

Zgled 9.10 *Stabilnost diferencialne enačbe lahko razberemo iz obnašanja pri reševanju* modelne enačbe *$y' = \lambda y$, $y(0) = y_0$. Ta diferencialna enačba ima rešitev $y(x) = y_0 e^{\lambda x}$. V primeru, ko je $\mathrm{Re}(\lambda) < 0$, je rešitev asimptotično stabilna.*

Če vzamemo eksplicitno Eulerjevo metodo, potem dobimo $y_1 = (1 + \lambda h) y_0$ in

$$y_k = (1 + \lambda h)^k y_0.$$

Če naj bo pri $\mathrm{Re}(\lambda) < 0$ Eulerjeva metoda stabilna, mora veljati $|1 + \lambda h| < 1$, torej mora biti h omejen ($h\lambda$ mora ležati v krogu z radijem 1 okrog -1).

Če vzamemo implicitno Eulerjevo metodo, potem dobimo $(1 - \lambda h) y_1 = y_0$ in

$$y_k = \left(\frac{1}{1 - \lambda h} \right)^k y_0.$$

Sedaj mora pri $\mathrm{Re}(\lambda) < 0$ veljati $\left| \frac{1}{1-\lambda h} \right| \leq 1$, torej h ni omejen *in za ta začetni problem je implicitna Eulerjeva metoda vedno stabilna.* □

9.5 Večkoračne metode

Pri večkoračnih metodah poleg zadnjega uporabimo tudi še nekaj prejšnjih približkov. Splošni nastavek je

$$\sum_{i=0}^{k} \alpha_i y_{n+1-i} + h \sum_{i=0}^{k} \beta_i f_{n+1-i} = 0,$$

kjer je $f_i = f(x_i, y_i)$, privzamemo pa še $\alpha_0 = 1$. Če je $\beta_0 = 0$, je metoda eksplicitna, sicer pa implicitna. Metodo določata *rodovna polinoma*

$$\begin{aligned} \rho(z) &= \sum_{i=0}^{k} \alpha_{k-i} z^i, \\ \sigma(z) &= \sum_{i=0}^{k} \beta_{k-i} z^i. \end{aligned}$$

Ker na začetku potrebujemo več kot eno začetno vrednost, moramo te približke pridobiti s kakšno drugo metodo. Lahko uporabimo npr. Runge-Kutta metodo ali razvijanje v Taylorjevo vrsto, paziti pa moramo na to, da ne uporabimo metode, ki ima manjši red lokalne napake kot večkoračna metoda. Pri reševanju začetnega problema je namreč tako, da ko so napake enkrat prisotne, se tudi z natančnejšim računanjem v nadaljevanju ne moremo več približati točni rešitvi.

Adamsove metode dobimo tako, da enačbo $y' = f(x, y)$ integriramo na $[x_n, x_{n+1}]$

$$y_{n+1} - y_n = \int_{x_n}^{x_{n+1}} y' dx = \int_{x_n}^{x_{n+1}} f(x, y) dx,$$

f pa nadomestimo z interpolacijskim polinomom na točkah:

a) $x_n, x_{n-1}, \ldots, x_{n-k+1}$: eksplicitne *Adams–Bashworthove formule*

$$\begin{aligned} y_{n+1} &= y_n + h f_n, \quad \text{lokalna napaka } \mathcal{O}(h^2), \\ y_{n+1} &= y_n + \frac{h}{2}(3f_n - f_{n-1}), \quad \text{lokalna napaka } \mathcal{O}(h^3), \\ y_{n+1} &= y_n + \frac{h}{12}(23f_n - 16f_{n-1} + 5f_{n-2}), \quad \text{lokalna napaka } \mathcal{O}(h^4). \end{aligned}$$

b) $x_{n+1}, x_n, \ldots, x_{n-k+2}$: implicitne *Adams–Moultonove formule*

$$\begin{aligned} y_{n+1} &= y_n + h f_{n+1}, \quad \text{lokalna napaka } \mathcal{O}(h^2), \\ y_{n+1} &= y_n + \frac{h}{2}(f_{n+1} + f_n), \quad \text{lokalna napaka } \mathcal{O}(h^3), \\ y_{n+1} &= y_n + \frac{h}{12}(5f_{n+1} + 8f_n - f_{n-1}), \quad \text{lokalna napaka } \mathcal{O}(h^4). \end{aligned}$$

Pri predpostavki, da je $k = n$ in $x_0 = 0$, definiramo linearni funkcional

$$L(y) = \sum_{i=0}^{k} \Big(\alpha_i y((k-i)h) + h\beta_i y'((k-i)h) \Big) = \sum_{j=0}^{k} \Big(\alpha_{k-j} y(jh) + h\beta_{k-j} y'(jh) \Big).$$

Če y in y' razvijemo v Taylorjevo vrsto okrog 0, dobimo

$$L(y) = d_0 y(0) + d_1 h y'(0) + d_2 h^2 y''(0) + \cdots .$$

Metoda je reda p, če je $L(y) = \mathcal{O}(h^{p+1})$ za vsako $(p+1)$-krat zvezno odvedljivo funkcijo y.

Izrek 9.9 *Za večkoračno metodo je ekvivalentno:*

a) $d_0 = d_1 = \cdots = d_m = 0$,

b) $L(q) = 0$ *za poljuben polinom* q *stopnje kvečjemu* m,

c) $L(y) = \mathcal{O}(h^{m+1})$ *za vsako* $(m+1)$*-krat zvezno odvedljivo funkcijo* y.

Red metode je p, če velja $d_0 = d_1 = \cdots = d_p = 0 \neq d_{p+1}$.

Če v Taylorjevo vrsto razvijemo y in y', dobimo

$$\begin{aligned} y(jh) &= \sum_{r=0}^{\infty} \frac{(jh)^r}{r!} y^{(r)}(0), \\ y'(jh) &= \sum_{r=0}^{\infty} \frac{(jh)^r}{r!} y^{(r+1)}(0). \end{aligned}$$

Sedaj dobimo naslednje formule za $d_0, d_1, d_2, \ldots$:

$$\begin{aligned} d_0 &= \sum_{j=0}^{k} \alpha_{k-j}, \\ d_1 &= \sum_{j=0}^{k} (j\alpha_{k-j} + \beta_{k-j}), \\ d_2 &= \sum_{j=0}^{k} \left(\frac{j^2}{2} \alpha_{k-j} + j\beta_{k-j} \right), \\ &\vdots \\ d_q &= \sum_{j=0}^{k} \left(\frac{j^q}{q!} \alpha_{k-j} + \frac{j^{q-1}}{(q-1)!} \beta_{k-j} \right). \end{aligned}$$

Zgled 9.11 *Določi red večkoračne metode* $y_n - y_{n-2} = \frac{h}{3}(f_n + 4f_{n-1} + f_{n-2})$.

Koeficienti so $\alpha_0 = -1$, $\alpha_1 = 0$, $\alpha_2 = 1$, $\beta_0 = \frac{1}{3}$, $\beta_1 = \frac{4}{3}$, $\beta_2 = \frac{1}{3}$. *Po zgornjih formulah lahko izračunamo* $d_0 = d_1 = \cdots = d_4 = 0$ *in* $d_5 = -\frac{1}{90}$, *kar pomeni, da je red metode enak* 4. □

Ponavadi pri večkoračnih metodah uporabljamo *prediktor-korektor metode*, kjer za izračun prediktorja y^P_{n+1} vzamemo eksplicitno metodo, za korektor pa vzamemo implicitno metodo, katere enačbo rešujemo iterativno tako, da na desno stran vstavimo približek y^P_{n+1}, na levi pa dobimo y^K_{n+1}. Tako se izognemo reševanju nelinearne enačbe oziroma je to kar en korak navadne iteracije za reševanje nelinearne enačbe. Postopek lahko ponovimo tako da dobljeni približek vstavimo na desno stran in izračunamo nov približek. Teorija nam zagotavlja, da bo ta postopek skonvergiral, če je h dovolj majhen.

Primer so *Milneove metode*, kjer $y' = f(x,y)$ integriramo na $[x_{n-k}, x_{n+1}]$:

$$y_{n+1} - y_{n-k} = \int_{x_{n-k}}^{x_{n+1}} f(x,y)dx,$$

integral pa računamo preko Newton–Cotesovih formul. Pri odprti dobimo eksplicitno, pri zaprti pa implicitno metodo. Primer je Milneov par

$$\begin{aligned} y_{n+1} &= y_{n-3} + \frac{4h}{3}(2f_n - f_{n-1} + 2f_{n-2}), \quad \text{napaka } \mathcal{O}(h^5) \quad : \text{ prediktor} \\ y_{n+1} &= y_{n-1} + \frac{h}{3}(f_{n+1} + 4f_n + f_{n-1}), \quad \text{napaka } \mathcal{O}(h^5) \quad : \text{ korektor} \end{aligned}$$

Zgled 9.12 *Diferencialno enačbo* $y' = -y - 5e^{-x}\sin(5x)$ *z začetnim pogojem* $y(0) = 1$ *rešujemo z Milneovim parom prediktor-korektor*

$$\begin{aligned} y_{n+1}^{(P)} &= y_{n-3} + \frac{4h}{3}(2f_n - f_{n-1} + 2f_{n-2}), \\ y_{n+1}^{(K)} &= y_{n-1} + \frac{h}{3}(f(x_{n+1}, y_{n+1}^{(P)}) + 4f_n + f_{n-1}). \end{aligned}$$

Denimo, da smo pri $h = 0.1$ *z drugimi metodami že prišli do približkov* $y_1 = 0.79407$, $y_2 = 0.44235$, $y_3 = 0.05239$, *sedaj pa bi z Milneovim parom radi izračunali* y_4.

$y_4^{(P)} = 1 + \frac{0.4}{3}(2 \cdot (-3.7472) + 3.8870 + 2 \cdot (-2.9631)) = -0.27115$

$y_4^{(K)} = 0.44235 + \frac{0.1}{3}(-2.7765 + 4 \cdot (-3.7472) - 3.8870) = -0.27939$

Postopek lahko nadaljujemo, $y_4^{(K)}$ *vnesemo na desno stran in dobimo nov približek.*

$y_4^{(K2)} = 0.44235 + \frac{0.1}{3}(-2.7682 + 4 \cdot (-3.7472) - 3.8870) = -0.27912$

$y_4^{(K3)} = 0.44235 + \frac{0.1}{3}(-2.7685 + 4 \cdot (-3.7472) - 3.8870) = -0.27913$

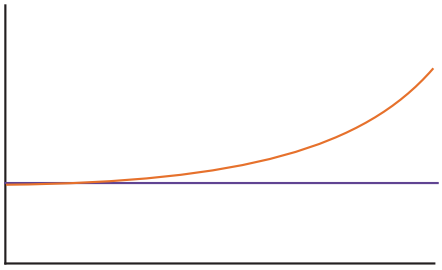
Sedaj nadaljujemo s točko $y_4 = -0.27913$ *in gremo na računanje* y_5. □

9.6 Inherentna in inducirana nestabilnost

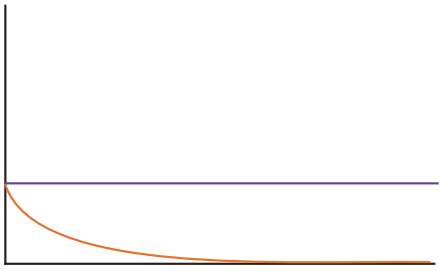
Inherentna nestabilnost je odvisna od problema in neodvisna od numerične metode.

Zgledi so:

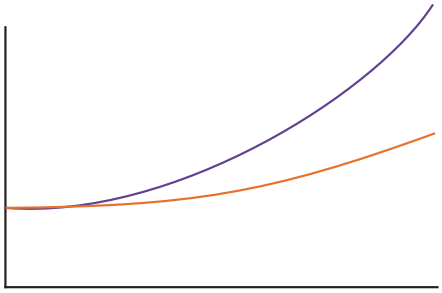
a) $y' = y - 1$, $y(0) = 1$, splošna rešitev je $y(x) = Ce^x + 1$ in $C = 0$, numerično pa dobimo $C \neq 0$. Problem je inherentno absolutno in relativno nestabilen.



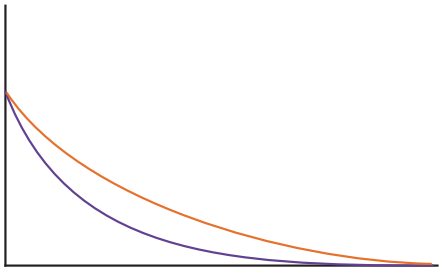
b) $y' = -y + 1$, $y(0) = 1$, splošna rešitev je $y(x) = Ce^{-x} + 1$ in $C = 0$, numerično pa dobimo $C \neq 0$. Problem je inherentno absolutno in relativno stabilen.



c) $y' = y + e^{2x}$, $y(0) = 1$, splošna rešitev je $y(x) = Ce^{x} + e^{2x}$ in $C = 0$, numerično pa dobimo $C \neq 0$. Problem je inherentno absolutno nestabilen in relativno stabilen.



d) $y' = -y - e^{2x}$, $y(0) = 1$, splošna rešitev je $y(x) = Ce^{-x} + e^{-2x}$ in $C = 0$, numerično pa dobimo $C \neq 0$. Problem je inherentno absolutno stabilen in relativno nestabilen.



Pri relativni nestabilnosti včasih pomaga, če obrnemo interval:

a) $y' = y - 1$, $y(x_0) = y_0$. Vzamemo $y(x_n) = \gamma$ in računamo nazaj,

b) $y' = -y - e^{2x}$, $y(x_0) = y_0$. Vzamemo $y(x_n) = \gamma$ in računamo nazaj.

Parameter γ moramo izbrati tako, da bo nazaj izračunana rešitev imela v točki x_0 pravo začetno vrednost y_0. Za pravo izbiro γ lahko uporabimo katerokoli metodo za reševanje nelinearnih enačb, npr. bisekcijo ali sekantno metodo. Če z $y(x;\gamma)$ označimo numerično rešitev, ki jo dobimo pri začetnem pogoju $y(x_n) = \gamma$ in računanju nazaj, potem moramo rešiti enačbo $y(x_0;\gamma) = y_0$. Podoben postopek uporabimo pri strelski metodi za reševanje robnega problema.

Inducirana nestabilnost je odvisna od izbire numerične metode. Pojavi se lahko le pri večkoračnih metodah. Za večkoračno metodo oblike

$$\sum_{i=0}^{k} \alpha_i y_{n-i} + h \sum_{i=0}^{k} \beta_i f_{n-i} = 0$$

pričakujemo, da je stabilna za enačbo $y' = 0$ (temu pravimo *ničelna stabilnost*), katere rešitev je konstanta. Ničelna stabilnost pomeni, da tudi če začnemo z nekaj začetnimi vrednostmi, kjer je prisotna napaka, mora potem rešitev, dobljena z večkoračno metodo, ostati omejena.

V primeru $y' = 0$ se večkoračna metoda spremeni v diferenčno enačbo

$$\sum_{i=0}^{k} \alpha_i y_{n-i} = 0.$$

Njen karakteristični polinom $\rho(\xi)$ ima k ničel $\xi_1, \ldots, \xi_k$. Splošna rešitev ima pri enostavnih ničlah obliko

$$y_n = \sum_{j=1}^{k} A_j \xi_j^n.$$

Za ničelno stabilnost mora tako veljati (Dahlquistov izrek)

a) $|\xi_i| \leq 1$ za $i = 1, \ldots, k$,

b) če je $|\xi_j| = 1$, mora biti ξ_j enostavna ničla.

Večkoračna metoda je konsistentna natanko tedaj, ko velja $\rho(1) = 0$ in $\rho'(1) + \sigma(1) = 0$, ta dva pogoja pa sta ekvivalentna temu, da je red vsaj 1. O pogojih za konvergentno večkoračno metodo govori naslednji izrek. Tako kot pri enokoračnih metodah lahko dokaz in ostale podrobnosti najdete v [18].

Izrek 9.10 *Večkoračna metoda je konvergentna natanko tedaj, ko je ničelno stabilna in konsistentna.* ■

9.7 Začetni problemi drugega reda

Rešujemo diferencialno enačbo drugega reda

$$\begin{aligned} y'' &= f(x, y, y') \\ y(x_0) &= y_0 \end{aligned}$$

$$y'(x_0) = y'_0.$$

To lahko prevedemo na sistem enačb prvega reda

$$\begin{array}{rclrcl} y' & = & p, & y(x_0) & = & y_0, \\ p' & = & f(x,y,p), & p(x_0) & = & y'_0. \end{array}$$

Zgled 9.13 *Vzemimo začetni problem drugega reda*

$$y'' = x + y^2$$

in $y(0) = 1$, $y'(0) = 0$. *To prevedemo na sistem dveh enačb prvega reda*

$$\begin{array}{rclrcl} y' & = & p, & y(0) & = & 1, \\ p' & = & x + y^2, & p(0) & = & 0. \end{array}$$

□

Zgled 9.14 *Problem dveh teles opisuje gibanje telesa zaradi sile težnosti telesa z veliko večjo maso. Če je težje telo v izhodišču, dobimo v kartezičnem koordinatnem sistemu enačbi*

$$\begin{aligned} x''(t) &= -x(t)/r(t)^3 \\ y''(t) &= -y(t)/r(t)^3, \end{aligned}$$

kjer je $r(t) = \sqrt{x(t)^2 + y(t)^2}$.

To prevedemo na sistem

$$z(t) = \begin{bmatrix} x(t) \\ y(t) \\ x'(t) \\ y'(t) \end{bmatrix}, \qquad z'(t) = \begin{bmatrix} z_3(t) \\ z_4(t) \\ -z_1(t)/r(t)^3 \\ -z_2(t)/r(t)^3 \end{bmatrix},$$

kjer je $r(t) = \sqrt{z_1(t)^2 + z_2(t)^2}$. □

V posebnem primeru, ko je $y'' = f(x,y)$, obstajajo posebne Runge-Kutta ali večkoračne metode, kot je npr. *metoda Numerova*

$$y_{n+1} - 2y_n + y_{n-1} = \frac{h^2}{12}(f_{n+1} + 10f_n + f_{n-1}) + \mathcal{O}(h^6).$$

9.8 Reševanje začetnih diferencialnih enačb v Matlabu

Na voljo imamo več metod. Izmed njjih je verjetno najbolj pogosto uporabljena `ode45`. Kličemo jo v obliki

```
[x,y]=ode45(fun,span,y0),
```

kjer je `fun` ime funkcije `yodv=fun(x,y)`, ki iz argumentov x in y, kjer je zadnji v primeru sistema vektor, izračuna vrednosti odvoda iz diferencialne enačbe. Seveda je v primeru sistema diferencialnih enačb tudi odvod, ki ga vrne `fun`, v obliki vektorja. Argument `span` poda interval $[a,b]$, na katerem iščemo rešitev, `y0` pa je začetni pogoj.

Zgled uporabe:

```
funcion yprime=fun(t,x)
yprime=-y-5*exp(-t)*sin(5*t);

tspan=[0 3]; yzero=1;
[x,y]=ode45('fun',tspan,y0)
plot(x,y,'*--');
```

Vse metode, ki jih uporablja Matlab, so adaptivne. Na voljo so naslednje metode:

- `ode45`: Temelji na Dormand–Princeovi metodi, ki je eksplicitna Runge-Kutta metoda reda 4 in 5 (podobno kot Fehlbergova metod). To je metoda, ki najpogosteje daje dobre rezultate, zato je priporočljivo problem najprej poskusiti rešiti s to metodo.
- `ode23`: Temelji na eksplicitnem Runge-Kutta pravilu reda 2 in 3. Kadar ne zahtevamo velike natančnosti in v primeru majhne togosti je lahko metoda bolj učinkovita kot pa `ode45`.
- `ode113`: Gre za Adams–Bashworth–Moultonovo večkoračno prediktor-korektor metodo, kjer vedno naredimo en korak korektorja. Lahko je bolj učinkovita od `ode45` v primerih, ko zahtevamo visoko natančnost in pa kadar je funkcija f iz diferencialne enačbe še posebej zapletena in želimo imeti čim manj izračunov vrednosti f.

Prejšnje tri metode so namenjene za netoge sisteme. V primeru, ko ne dajejo dobrih rezultatov imamo lahko opravka s togim sistemom in takrat lahko uporabimo naslednje metode:

- `ode15s`: Večkoračna metoda, ki temelji na metodah za numerično odvajanje. Je sicer manj učinkovita kot `ode45`, a deluje na zmerno togih primerih, ko `ode45` odpove.
- `ode23s`: Uporablja modificirano Rosenbrockovo metodo reda 2. Kadar ne zahtevamo velike natančnosti je lahko bolj učinkovita kot `ode15s` in deluje na nekaterih zelo togih primerih, kjer `ode15s` odpove.
- `ode23t`: Varianta trapezne metode. Uporabna za zmerno toge primere kadar ne zahtevamo velike natančnosti.
- `ode23tb`: Varianta implicitne dvostopenjske Runge-Kutta metode, primerna za zelo toge sisteme.

Vse metode uporabljamo na enak način, zato je dovolj pogledati le uporabo `ode45`.

- Osnovna uporaba je `[x,y] = ode45('f',xab,y0)`, kjer metoda `f` določa diferencialno enačbo $y' = f(x,y)$, $xab = [a\ b]$ je interval, na katerem rešujemo enačbo in y_0 je začetni pogoj $y(a) = y_0$. Kot rezultat dobimo vektorja izračunanih x in y.
- Dodatne opcije, s katerimi nastavimo npr. željeno natančnost, podamo v obliki `[x,y] = ode45('f',xab,y0,options)`. Pri tem moramo opcije `options` podati ali prej definirati s pomočjo funkcije `odeset`.

  Tako npr. z `options=odeset('RelTol',1e-4,'AbsTol',1e-8)` povečamo relativno in absolutno natančnost (privzeti vrednosti sta `1e-3` in `1e-6`).

- Če ima diferencialna enačba f poleg x in y še kakšne dodatne parametre, jih podamo v obliki `[x,y] = ode45('f',xab,y0,options,p1,p2,...)`. Pri tem lahko kot `options` podamo prazem seznam `[]`, kadar ne želimo spreminjati nastavitev.
- Z `[x,y,s] = ode45('f',xab,y0,options)` dobimo še vektor s s šestimi komponentami, ki nam povedo, kaj se je dogajalo med računanjem. Tako je s_1 število uspešnih korakov, s_2 število neuspešnih poskusov, s_3 število izračunov f, s_4 število izračunov Jacobijeve matrike parcialnih odvodov f po y, s_5 število LU razcepov in s_6 število reševanj linearnih sistemov.

Diferencialne enačbe, kjer poleg x in y nastopajo še dodatni parametri, uporabljamo v obliki `[x,y] = ode45('f',xab,y0,options,p1,p2,...)`. Pri tem moramo paziti na to, da ima funkcija f po vrsti parametre x, y, $flag$, p_1, p_2, ... Parameter $flag$ se uporablja za to, da diferencialna enačba po potrebi avtomatično poda začetne pogoje in interval, če tega ne uporabljamo pa moramo le pustiti prostor za nek parameter.

Tako npr. s kombinacijo

```
f=inline('x^2+y+p','x','y','flag','p')
[x1,y1]=ode45(f,[0 1],0,[],1)
[x2,y2]=ode45(f,[0 1],0,[],2)
```

rešimo diferencialni enačbi $y' = x^2 + x + 1$ in $y' = x^2 + y + 2$.

Velikokrat rešujemo diferencialno enačbo $y' = f(x,y)$, $y(x_0) = y_0$, kjer točen interval $[x_0, x_n]$ ni znan, saj je izračun točke x_n, kjer pride do kakšnega dogodka, del same naloge. Tako lahko npr. rešujemo nalogo telesa, ki prosto pada z določene višine, zanima pa nas, v katerem trenutku zadane tla.

Matematično formulirano imamo dano funkcijo $g(x,y)$ in iščemo rešitev začetnega problema $y' = f(x,y)$, $y(x_0) = y_0$ in točko x^*, kjer je $g(x^*, y(x^*)) = 0$. V Matlabu to lahko rešimo tako, da v nastavitvah določimo kontrolno funkcijo g, ki pove, kdaj se je dogodek zgodil.

```
function ydot = f(t,y)
ydot = [y(2); -1+y(2)^2];

function [gstop,isterminal,direction] = g(t,y)
gstop = y(1);      % Dogodek je, ko funkcija g vrne 0
isterminal = 1;    % Metoda se konča, ko pride do dogodka
direction = [];    % Do nič lahko pridemo z obeh strani

opts = odeset('events',@g);
y0 = [1; 0];
[t,y,tfinal] = ode45(@f,[0 Inf],y0,opts);
plot(t,y(:,1),'-',[0 tfinal],[1 0],'o')
```

9.9 Implicitno podane diferencialne enačbe

Diferencialna enačba je lahko namesto v eksplicitni obliki $y' = f(x,y)$ podana v implicitni obliki

$$f(x,y,y') = 0. \tag{9.1}$$

Če se iz enačbe (9.1) ne da direktno izraziti y' kot funkcija x in y, potem lahko sistem numerično rešujemo tako, da v vsakem koraku pri danih vrednostih za y in x dobimo y' iz (9.1) z numeričnim reševanjem nelinearne enačbe.

Reševanje začetnega problema si lahko predstavljamo kot sledenje krivulji, ki se začne pri $y(x_0) = y_0$. V primeru implicitno podane diferencialne enačbe moramo poznati tudi vrednost $y'(x_0)$, saj ima enačba (9.1) lahko več kot le eno rešitev. Ko poznamo x_0, y_0 in y'_0 lahko naredimo npr. en korak Eulerjeve metode in pridemo do $x_1 = x_0 + h$, $y_1 = y_0 + hy'_0$. V novi točki moramo spet rešiti nelinearno enačbo za y'_1, pri čemer pa lahko, če h ni prevelik, za začetni približek vzamemo kar y'_0.

Podobno velja za nelinearne implicitne sisteme nelinearnih enačb. Princip je enak, le da moramo sedaj namesto nelinearne enačbe reševati nelinearne sisteme, da pridemo do vektorja odvodov.

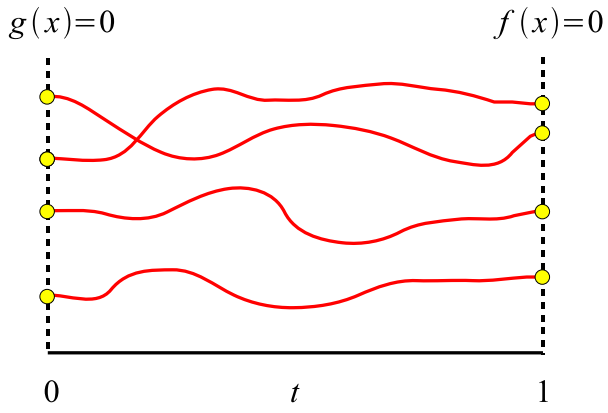
9.10 Metoda zveznega nadaljevanja

To je metoda za reševanje nelinearne enačbe $f(x) = 0$. Če je težko poiskati začetni približek (posebno, kadar je $f : \mathbb{R}^n \to \mathbb{R}^n$), si lahko pomagamo z uvedbo dodatnega parametra. Ideja je, da izberemo sorodni enostavnejši problem $g(x) = 0$, ki ga znamo rešiti, potem pa z dodatnim parametrom t, ki teče od 0 do 1, enostavnejši problem zvezno prevedemo na $f(x) = 0$. Pri tem pričakujemo, da se bodo z zveznim spreminjanjem problema zvezno premikale tudi rešitve in bomo tako s sledenjem iz znanih rešitev enačbe $g(x) = 0$ prišli do iskanih rešitev $f(x) = 0$.

Opazujemo npr. *konveksno homotopijo*

$$F(t,x) = t \cdot f(x) + (1-t) \cdot g(x),$$

kjer je $0 \le t \le 1$ in poznamo rešitve $g(x) = 0$. S sledenjem krivulji od $t = 0$ do $t = 1$ dobimo rešitve $f(x) = 0$.



Sledenje krivulje je povezano z reševanjem diferencialne enačbe. Z odvajanjem po t dobimo

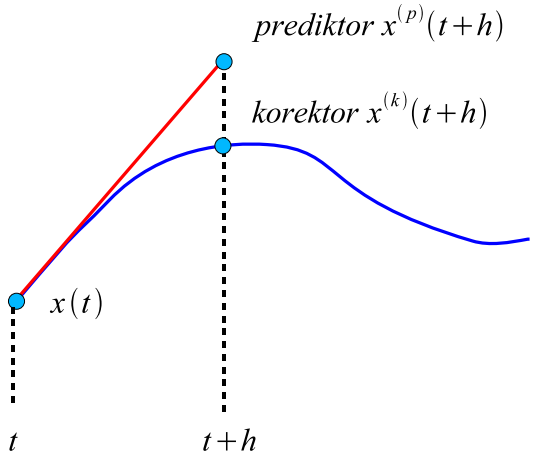
$$F_t(t,x) + F_x(t,x) \cdot \dot{x} = 0,$$

kar nam da navadno diferencialno enačbo

$$\dot{x} = -F_x(t,x)^{-1} \cdot F_t(t,x), \quad x(0) = x_0,$$

kjer je $g(x_0) = 0$.

Z metodami za reševanje navadnih diferencialnih enačb lahko sledimo rešitvi, poleg tega pa pri vsaki vrednosti t lahko upoštevamo še, da mora za $x(t)$ veljati $F(t, x(t)) = 0$. Ponavadi uporabljamo kombinacijo prediktor–korektor, ko najprej uporabimo metodo za reševanje začetnega problema, potem pa iz dobljenega približka izračunamo novo točko na krivulji z kakšno metodo za reševanje nelinearnega sistema.



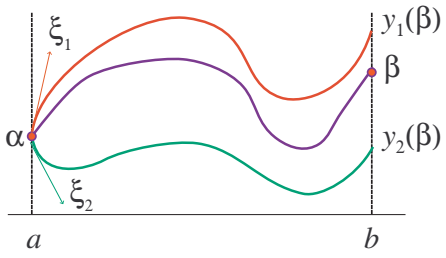
a) prediktor: Iz točke $(t, x(t))$ z eno izmed metod za reševanje začetnega problema (npr. z Eulerjevo metodo) izračunamo prediktor $x^{(P)}(t+h)$, ki je približek za rešitev v točki $t+h$.

b) korektor: Z eno izmed metod za reševanje nelinearnega sistema (npr. z Newtonovo metodo) rešimo sistem $F(t+h, x(t+h)) = 0$, za začetni približek pa vzamemo $(t+h, x^{(P)}(t+h))$.

9.11 Robni problemi drugega reda

Pri robnih problemih imamo opravka z diferencialno enačbo višjega reda, kjer dodatni pogoji, ki določajo pravo vejo, niso podani v isti točki. Tako pri robnem problemu drugega reda rešujemo diferencialno enačbo drugega reda na intervalu $[a, b]$ z robnima pogojema:

$$\begin{aligned} y'' &= f(x, y, y') \\ y(a) &= \alpha \\ y(b) &= \beta. \end{aligned}$$



Slika 9.6: Rešitev linearnega robnega problema lahko sestavimo kot linearno kombinacijo dveh začetnih problemov.

9.11.1 Linearni robni problem

Najpreprostejši je *linearni robni problem drugega reda*, ki ima obliko

$$-y''(x) + p(x)y'(x) + q(x)y(x) = r(x), \tag{9.2}$$
$$y(a) = \alpha, \quad y(b) = \beta.$$

Numerični načini reševanja linearnega robnega problema so:

a) *Kombinacija dveh začetnih problemov*. Izberemo različna ξ_1 in ξ_2 in rešimo začetna problema, ki ustrezata enačbi (9.2) in

$$\begin{aligned} y_1(a) &= \alpha, \quad y_1'(a) = \xi_1, \\ y_2(a) &= \alpha, \quad y_2'(a) = \xi_2. \end{aligned}$$

Za poljuben λ je $y = \lambda y_1 + (1-\lambda)y_2$ tudi rešitev enačbe (9.2) in ustreza pogoju $y(a) = \alpha$, Sedaj λ določimo tako, da bo $y(b) = \beta$, situacija je predstavljena na sliki 9.6. Veljati mora

$$\lambda y_1(b) + (1-\lambda)y_2(b) = \beta,$$

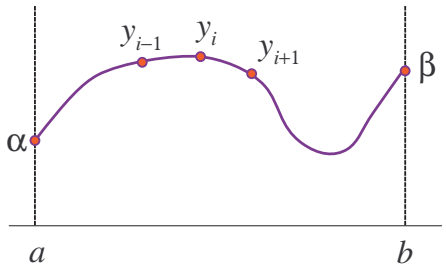
torej

$$\lambda = \frac{\beta - y_2(b)}{y_1(b) - y_2(b)}.$$

Dva začetna problema moramo reševati kot sistem, pa čeprav bi lahko enačbi rešili ločeno drugo od druge. Če jih rešujemo ločeno, se pri adaptivnih metodah lahko namreč zgodi, da rešitvi ne bosta tabelirani v istih točkah, potem pa ne moremo izračunati linearne kombinacije rešitev.

b) *Diferenčna metoda*. Interval $[a, b]$ ekvidistantno razdelimo na $n+1$ delov s točkami $x_0 = a, x_1, \ldots, x_n, x_{n+1} = b$, kjer je $x_i = x_0 + ih$ in $h = (b-a)/(n+1)$. Odvode aproksimiramo s simetričnimi diferencami

$$\begin{aligned} y_i' &= \frac{y_{i+1} - y_{i-1}}{2h} + \mathcal{O}(h^2), \\ y_i'' &= \frac{y_{i+1} - 2y_i + y_{i-1}}{h^2} + \mathcal{O}(h^2). \end{aligned}$$



Slika 9.7: Približki pri diferenčni metodi.

Dobimo sistem enačb

$$\frac{-y_{i+1}+2y_i-y_{i-1}}{h^2}+p_i\frac{y_{i+1}-y_{i-1}}{2h}+q_iy_i=r_i,\quad i=1,\ldots,n,$$

pri čemer je $y_0=\alpha$ in $y_{n+1}=\beta$. Sistem je linearen in tridiagonalen, zato ga lahko rešimo na preprost način. To nam tudi omogoča, da vzamemo velik n, kar izboljša natančnost rešitve.

Če v diferencialni enačbi ne nastopa y', lahko uporabimo metodo Numerova, saj je potem napaka reda $\mathcal{O}(h^6)$ namesto $\mathcal{O}(h^2)$. V primeru diferencialne enačbe

$$-y''(x)+q(x)y(x)=r(x)$$

tako za $i=1,\ldots,n$ dobimo enačbo

$$y_{i+1}-2y_i+y_{i-1}=\frac{h^2}{12}(q_{n+1}y_{n+1}-r_{n+1}+10q_ny_n-r_n-q_{n-1}y_{n-1}+r_{n-1}).$$

Zgled 9.15 *Z diferenčno metodo in $h=1/2$ rešimo robni problem*

$$\begin{aligned}(1+x^2)y''&+2xy'-x^2y=1\\ y(-1)&=0\\ y(1)&=0.\end{aligned}$$

Dobimo sistem enačb za y_1,y_2,y_3:

$$(1+(-0.5)^2)\frac{y_2-2y_1+0}{(0.5)^2}+2(-0.5)\frac{y_2-0}{2(0.5)}-(-0.5)^2y_1=1$$

$$(1+(0.0)^2)\frac{y_3-2y_2+y_1}{(0.5)^2}+2(0.0)\frac{y_3-y_1}{2(0.5)}-(0.0)^2y_2=1$$

$$(1+(0.5)^2)\frac{0-2y_3+y_2}{(0.5)^2}+2(0.5)\frac{0-y_3}{2(0.5)}-(0.5)^2y_3=1,$$

ki ima rešitev

$$y_1=-0.24,\ y_2=-0.365,\ y_3=-0.24.$$ □

Diferenčno metodo lahko uporabljamo tudi pri robnih pogojih, v katerih nastopajo odvodi. Če imamo splošne robne pogoje oblike

$$\alpha_1 f(a) + \beta_1 f'(a) = 0, \quad \text{in} \quad \alpha_2 f(b) + \beta_2 f'(b) = 0,$$

si pomagamo s t.i. navideznimi točkami.

Npr., če imamo v robnem pogoju odvod v točki a, potem ta odvod aproksimiramo s simetrično diferenco

$$y_0' = \frac{y_1 - y_{-1}}{2h}.$$

Nova neznanka y_{-1} je le navidezna. Ker mora tudi v točki a rešitev zadoščati diferencialni enačbi, dobimo še enačbo

$$\frac{-y_1 + 2y_0 - y_{-1}}{h^2} - p_0 \frac{y_1 - y_{-1}}{2h} + q_0 y_0 = r_0,$$

iz teh dveh enačb pa lahko izločimo y_{-1}.

Če podvojimo število točk, pričakujemo, da bomo dobili boljše približke. Ker pa za napako velja, da je reda h^2, lahko z uporabo ekstrapolacije, podobne kot pri Rombergovi metodi, dobimo še boljše približke v tistih točkah, ki nastopajo tako v grobi kot v fini delitvi.

9.11.2 Nelinearni robni problem

Pri *nelinearnem robnem problemu drugega reda* je f nelinearna funkcija y in y'. Zaradi nelinearnosti ne moremo uporabiti kombinacije dveh začetnih problemov. Uporabimo lahko diferenčno metodo, a dobimo nelinearni sistem z veliko neznankami.

Nelinearne enačbe imajo obliko

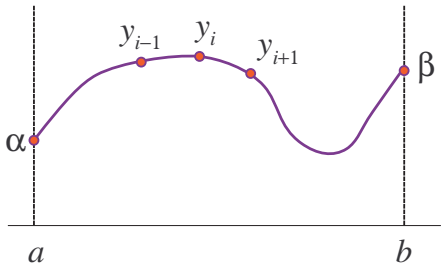
$$\frac{y_{i-1} - 2y_i + y_{i+1}}{h^2} = f\left(x, y_i, \frac{y_{i+1} - y_{i-1}}{2h}\right), \quad i = 1, \ldots, n.$$

Če nelinearni sistem rešujemo z Newtonovo metodo, je Jacobijeva matrika tridiagonalna, tako da se da sistem reševati dokaj ekonomično. Za začetni približek lahko npr. vzamemo kar točke na daljici med (a, α) in (b, β), torej $y_i = \alpha + i(\beta - \alpha)/n$.

Na voljo imamo tudi *strelsko metodo*. Pri strelski metodi rešujemo začetni problem

$$\begin{aligned} y'' &= f(x, y, y'), \\ y(a) &= \alpha, \\ y'(a) &= \xi. \end{aligned} \tag{9.3}$$

Rešitev je odvisna od parametra ξ, ki ga je potrebno izbrati tako, da bo $y(b; \xi) = \beta$.



Če definiramo $E(\xi) := y(b;\xi) - \beta$, potem iščemo tak ξ, da bo $E(\xi) = 0$. Uporabimo lahko katerokoli metodo za reševanje nelinearne enačbe. Če npr. želimo uporabiti tangentno metodo, potem potrebujemo tudi $E'(\xi) = \frac{\partial y(b;\xi)}{\partial \xi}$. Definiramo $z := \frac{\partial y}{\partial \xi}$ in z odvajanjem

$$y'' = f(x, y, y'), \quad y(a) = \alpha,\ y'(a) = \xi$$

dobimo

$$z'' = f_y(x, y, y')z + f_{y'}(x, y, y')z', \quad z(a) = 0,\ z'(a) = 1. \tag{9.4}$$

Če rešujemo sistem (9.3) in (9.4), potem dobimo tudi vrednost odvoda funkcije E.

9.11.3 Prevedba na variacijski problem

Reševanje robnega problema

$$-\frac{d}{dx}\left(p(x)\frac{dy}{dx}\right) + q(x)y = f(x), \quad 0 \le x \le b,$$

z robnima pogojema $y(0) = 0$ in $y'(b) + \sigma y(b) = 0$, kjer je $p(x) \ge \delta > 0$, $q(x) \ge 0$ in $\sigma > 0$, je ekvivalentno iskanju funkcije $y \in C^2[0, b]$, ki zadošča robnima pogojema in minimizira funkcijo

$$F(y) = \int_0^b \left(p(x)y'^2(x) + q(x)y^2(x) - 2f(x)y(x)\right) dx + p(b)\sigma y^2(b).$$

Ideja *Rayleigh–Ritzove metode* je, da namesto $y \in C^2[0, b]$ iščemo približek oblike

$$y(x) = \sum_{i=1}^{n} c_i \phi_i(x),$$

kjer so $\phi_1, \ldots, \phi_n$ t.i. *bazne funkcije*.

V točki, kjer je dosežen minimum, morajo biti vsi parcialni odvodi F po c_i enaki 0. Tako dobimo linearni sistem $Ac = b$ za koeficiente $c = [\, c_1 \quad \cdots \quad c_n \,]^T$, kjer je

$$a_{ij} = \int_0^b \left(p(x)\phi_i'(x)\phi_j'(x) + q(x)\phi_i(x)\phi_j(x)\right) dx$$

in

$$b_i = \int_0^b \frac{\phi_i(x)}{f(x)} dx$$

za $i, j = 1, \ldots, n$. Od izbire baznih funkcij je odvisno, ali bo sistem rešljiv in kako dober bo dobljen približek.

Metoda končnih elementov je poseben primer Rayleigh-Ritzove metoda, kjer za bazne funkcije izberemo t.i. *piramidne funkcije*

$$\phi_i(x) = \begin{cases} 0, & 0 \le x \le x_{i-1}, \\ (x - x_{i-1})/h_{i-1}, & x_{i-1} < x \le x_i, \\ (x_{i+1} - x)/h_i, & x_i < x \le x_{i+1}, \\ 0, & x_{i+1} < x \le b, \end{cases}$$

kjer je $0 = x_0 < x_1 < x_2 < \cdots < x_n = b$ in $x_i - x_{i-1} = h_{i-1}$.

Funkcija ϕ_i, kjer je $i = 1, \ldots, n-1$, ima nosilec na intervalu (x_{i-1}, x_{i+1}), izjema je le funkcija ϕ_n ki ima neničelni del samo na intervalu $(x_{n-1}, x_n]$. Zaradi tega je matrika A sedaj tridiagonalna. Elementi sistema $Ax = b$ so

$$\begin{aligned}
a_{ii} &= \int_{x_{i-1}}^{x_i} \frac{1}{h_{i-1}^2} p(x)dx + \int_{x_i}^{x_{i+1}} \frac{1}{h_i^2} p(x)dx \\
&\quad + \int_{x_{i-1}}^{x_i} \frac{1}{h_{i-1}^2}(x - x_{i-1})^2 q(x)dx + \int_{x_i}^{x_{i+1}} \frac{1}{h_i^2}(x_{i+1} - x)^2 q(x)dx, \quad i = 1, \ldots, n-1, \\
a_{i,i+1} &= \int_{x_i}^{x_{i+1}} \frac{-1}{h_i^2} p(x)dx + \int_{x_i}^{x_{i+1}} \frac{1}{h_i^2}(x_{i+1} - x)(x - x_i) q(x)dx, \quad i = 1, \ldots, n-1, \\
a_{i-1,i} &= \int_{x_{i-1}}^{x_i} \frac{-1}{h_{i-1}^2} p(x)dx + \int_{x_i}^{x_{i+1}} \frac{1}{h_i^2}(x_{i+1} - x)(x - x_{i-1}) q(x)dx, \quad i = 2, \ldots, n, \\
b_i &= \int_{x_{i-1}}^{x_i} \frac{1}{h_{i-1}}(x - x_{i-1}) f(x)dx + \int_{x_i}^{x_{i+1}} \frac{1}{h_i}(x_{i+1} - x) f(x)dx, \quad i = 1, \ldots, n-1, \\
a_{nn} &= \int_{x_{n-1}}^{x_n} \frac{1}{h_{n-1}^2} p(x)dx + \int_{x_{n-1}}^{x_n} \frac{1}{h_{n-1}^2}(x - x_{n-1})^2 q(x)dx + p(b)\sigma, \\
b_n &= \int_{x_{n-1}}^{x_n} \frac{1}{h_{n-1}}(x - x_{n-1}) f(x)dx.
\end{aligned}$$

9.12 Reševanje robnih problemov v Matlabu

V Matlabu imamo za reševanje robnih problemov na voljo metodo `bvp4c`, ki uporablja kolokacijsko metodo. S to metodo lahko rešujemo robne probleme, ki jih zapišemo v obliki

$$\frac{d}{dx} y(x) = f(x, y(x), p), \quad g(y(a), y(b), p) = 0.$$

Tu je $y(x)$ vektor, f je podana funkcija x in y, ki opisuje diferencialno enačbo. Vektor p je neobvezen in opisuje neznane parametre, ki jih iščemo. Rešitev iščemo na intervalu $[a, b]$, s funkcijo g pa podamo robne pogoje. Metoda potrebuje tudi začetni približek za rešitev $y(x)$.

Oblika za klic `bvp4c` je `sol=bvp4c(@odefun,@bcfun,solinit,options)`, pri čemer:

- funkcija `odefun` poda diferencialno enačbo, oblika je `yodv = odefun(x,y);`
- funkcija `bcfun` poda robne pogoje (vrne ostanek, ki mora biti pri izpolnjenih pogojih enak 0), oblika je `res = bcfun(ya,yb);`
- `solinit` je struktura, s katero podamo začetni približek, strukturo zgradimo s pomožno funkcijo `bvpinit` kot `solinit = bvpinit(linspace(a,b,n),@initf)`, kjer je `initf` funkcija oblike `y = initf(x)`, s katero podamo robne pogoje;
- `options` je neobvezen argument, kjer lahko nastavimo delovanje metode, za nastavitve uporabimo metodo `bvpset`.

Zgled 9.16 *Presek oblike kapljice vode na ravni podlagi je podan z rešitvijo robnega problema*

$$u''(x) + (1 - u(x))(1 + u'(x)^2)^{3/2} = 0, \quad u(-1) = 0, \quad u(1) = 0.$$

Za uporabo `bvp4c` *to spremenimo v sistem enačb prvega reda:*

$$\begin{aligned} y_1'(x) &= y_2(x), \\ y_2'(x) &= (y_1(x)-1)(1+y_2(x)^2)^{3/2}, \end{aligned}$$

Za začetni približek vzamemo $y_1(x) = \sqrt{1-x^2}$ *in* $y_2(x) = -x/(0.1+\sqrt{1-x^2})$.

Ustrezna koda za rešitev problema v Matlabu je:

```
function yodv = kaplja(x,y)
yodv = [ y(2); (y(1)-1)*((1+y(2)^2)^(3/2)) ];

function res = kapljarp(ya,yb)
res = [ ya(1); yb(1) ];

function y = kapljainit(x)
y = [ sqrt(1-x^2); -x/(0.1+sqrt(1+x^2)) ];

resinit = bvpinit(linspace(-1,1,20), @kapljainit);
res = bvp4c(@kaplja, @kapljarp, resinit);
plot(res.x, res.y(1,:))
```

□

Zgled 9.17 *Iščemo lastno vrednost* λ*, pri kateri ima robni problem*

$$y''(x) + (\lambda - 10\cos(2x))y(x) = 0, \quad y'(\pi) = 0, \quad y'(0) = 0$$

neničelno rešitev. Ker lahko vsako tako rešitev pomnožimo s poljubnim skalarjem, potrebujemo še en pogoj, npr. $y(0) = 1$*. Podobno kot v prejšnjem zgledu enačbo spremenimo v sistem enačb prvega reda:*

$$y_1'(x) = y_2(x), \quad y_2'(x) = (10\cos(2x) - \lambda)y_1(x).$$

Za začetni približek vzamemo $y(x) = \cos(4x)$ *in* $\lambda = 15$*.*

Ustrezna koda za rešitev problema v Matlabu je:

```
function yodv = mth(x,y,lambda)
yodv = [  y(2);  (10*cos(2*x)- lambda)*y(1) ];

function res = mthrp(ya,yb,lambda)
res = [ ya(2);  yb(2); ya(1)-1 ];

function y = mthinit(x)
y = [ cos(4*x);  -4*sin(4*x) ];

resinit = bvpinit(linspace(0,pi,10), @mthinit, 15);
res = bvp4c(@mth, @mthrp, resinit);
fprintf('Lastna vrednost je %7.3f.\n', res.parameters)
plot(res.x, res.y(1,:))
```

□

Zgled 9.18 *Naslednje enačbe opisujejo pretok v dolgem navpičnem kanalu, kjer je tekočina vbrizgana skozi eno izmed stranic:*

$$\begin{aligned} f''' - R\left[(f')^2 - ff''\right] + RA &= 0, \\ h'' + Rfh' + 1 &= 0, \\ \theta'' + Pf\theta' &= 0, \end{aligned}$$

pri čemer je R Raynoldsovo število in $P = 0.7R$. Ker parameter A spada med neznanke, imamo 8 robnih pogojev:

$$\begin{aligned} f(0) &= f'(0) = 0, \quad f(1) = 1, \quad f'(1) = 0, \\ h(0) &= h(1) = 0, \\ \theta(0) &= 0, \quad \theta(1) = 1. \end{aligned}$$

Problem bi radi rešili pri $R = 10000$, vendar nimamo nobenih dobrih začetnih približkov. Zato uporabimo metodo zveznega nadaljevanja. Najprej problem rešimo za $R = 100$, potem to uporabimo kot začetni približek za $R = 1000$, tega pa za $R = 10000$.

Ustrezna koda za rešitev problema v Matlabu je:

```
R = 100; resinit = bvpinit(linspace(0,1,10), ones(7,1), 1);
res1 = bvp4c(@pretok, @pretokbc, resinit);
fprintf('Parameter A pri R=100  : %7.4f\n',res1.parameters)

R = 1000; res2 = bvp4c(@pretok, @pretokbc, res1);
fprintf('Parameter A pri R=1000 : %7.4f\n',res2.parameters)

R = 10000; res3 = bvp4c(@pretok, @pretokbc, res2);
fprintf('Parameter A pri R=10000  : %7.4f\n',res3.parameters)
plot(res1.x, res1.y(2,:), res2.x, res2.y(2,:), res3.x, res3.y(2,:));

function yodv = pretok(x,y,A)
  P = 0.7*R;
  yodv = [ y(2); y(3); R*(y(2)^2 - y(1)*y(3) - A);
           y(5); -R*y(1)*y(5) - 1; y(7); -P*y(1)*y(7) ];
end

function res = pretokbc(ya,yb,A)
  res = [ya(1); ya(2); yb(1) - 1; yb(2); ya(4); yb(4); ya(6); yb(6) - 1];
end
```

□

Dodatna literatura

Več o metodi zveznega nadaljevanja lahko najdete v [1].

Literatura

[1] E. L. Allgower, K. Georg, *Numerical Continuation Methods: An Introduction*, Springer-Verlag, New York, 1990.

[2] R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, H. van der Vorst, *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*, SIAM, Philadelphia, 1994.

[3] Z. Bohte, *Numerične metode, 4. ponatis*, DMFA Slovenije, Ljubljana, 1991.

[4] Z. Bohte, *Numerično reševanje nelinearnih enačb*, DMFA Slovenije, Ljubljana, 1993.

[5] Z. Bohte, *Numerično reševanje sistemov linearnih enačb*, DMFA Slovenije, Ljubljana, 1994.

[6] R. L. Burden, J. D. Faires: *Numerical Analysis*, Brooks/Cole, Pacific Grove, 1997.

[7] B. N. Datta, *Numerical Linear Algebra and Applications*, Brooks/Cole, Pacific Grove, 1995.

[8] J. W. Demmel, *Uporabna numerična linearna algebra*, DMFA Slovenije, Ljubljana, 2000.

[9] G. H. Golub, C. F. van Loan, *Matrix Computations, 3rd edition*, The Johns Hopkins University Press, Baltimore, 1996.

[10] G. H. Golub, F. Uhlig, *The QR algorithm: 50 years later its genesis by John Francis and Vera Kublanovskaya and subsequent developments*, IMA Journal of Numerical Analysis **29**, 2009, str. 467—485.

[11] J. F. Grcar, Mathematicians of Gaussian Elimination, AMS Notices **58**, 2011, str. 782–792.

[12] P. C. Hansen, *Regularization Tools. A Matlab Package for Analysis and Solution of Discrete Ill-Posed Problems*, 2008, `http://www2.imm.dtu.dk/~pch/Regutools/RTv4manual.pdf`

[13] M. T. Heath, *Scientific Computing: An Introductory Survey, 2nd edition*, McGraw-Hill, 2002.

[14] D. J. Higham, N. J. Higham, *Matlab Guide, Second Edition*, SIAM, Philadelphia, 2005.

[15] N. J. Higham, *Accuracy and stability of numerical algorithms*, SIAM, Philadelphia, 1996.

[16] C. T. Kelley, *Iterative Methods for Linear and Nonlinear Equations*, SIAM, Philadelphia, 1995.

[17] D. Kincaid, W. Cheney, *Numerical Analysis*, Brooks/Cole, Pacific Grove, 1996.

[18] J. Kozak, *Numerična analiza*, DMFA Slovenije, Ljubljana, 2008.

[19] C.-Y. Lam, *Applied Numerical Methods For Partial Differential Equations*, Prentice Hall, Singapore, 1994.

[20] B.-S. Liao, Z. Bai, L.-Q. Lee, K. Ko, Solving large scale nonlinear eigenvalue problems in next-generation accelerator design. Technical Report CSE-TR, University of California, Davis, 2006.

[21] C. D. Meyer, *Matrix Analysis and Applied Linear Algebra*, SIAM, Philadelphia, 2000.

[22] J. Miller, *Earliest Know Uses of Some of the Words of Mathematics*, `http://jeff560.tripod.com/mathworld.com`.

[23] K. W. Morton, D. F. Mayers, *Numerical Solution of Partial Differential Equations*, Cambridge University Press, Cambridge, 2002

[24] J. J. O'Connor, E. F. Robertson, *MacTutor History of Mathematics archive*, University of St Andrews, Scotland, `http://www-history.mcs.st-andrews.ac.uk/`.

[25] T. Papler, *Stabilno računanje lastnih vektorjev simetrične tridiagonalne matrike v* $\mathcal{O}(n^2)$, diplomsko delo, Ljubljana, 2005.

[26] Y. Saad, *Iterative Methods for Sparse Linear Systems, Second Edition*, SIAM, Philadelphia, 2003.

[27] G. D. Smith, *Numerical Solution of Partial Differential Equations: Finite Difference Methods, 3rd edition*, Clarendon Press, Oxford, 1985.

[28] G. W. Stewart, J.-G. Sun, *Matrix Perturbation Theory*, Academic Press, San Diego, 1990.

[29] F. Tisseur, K. Meerbergen: *The quadratic eigenvalue problem*, SIAM Review **43** (2001), 235–286.

[30] L. N. Trefethen, L. M. Trefethen, *How many shuffles to randomize a deck of cards?*, Proc. Royal Soc. A **456** (2002), str. 2561-–2568.

[31] S. Van Huffel, J. Vandewalle, *The Total Least Square Problem. Computational Aspecs and Analysis*, SIAM, Philadelphia, 1991.

[32] J. H. Wilkinson, *The Algebraic Eigenvalue Problem*, Oxford University Press, Oxford, 1965.

[33] T. J. Ypma, *Historical development of the Newton-Raphson method*, SIAM Review **37**, 1995, str. 531–555.

[34] E. Zakrajšek, *Uvod v numerične metode, druga izdaja*, DMFA Slovenije, Ljubljana, 2000.