Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Osnove numerične matematike

Bojan Orel

Ljubljana, 21. februar 2019

Kazalo

Poglavje 1

Uvod

1.1 Zakaj numerične metode

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

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

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

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

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

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

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

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

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

1.2 Napake in numerično računanje

Pri numeričnem računanju se ne moremo izogniti napakam. Če hočemo, da bo napaka pri končnem rezultatu manjša od maksimalne dopustne napake, moramo poznati izvor napak pri numeričnem reševanju problemov in njihov

vpliv na končni rezultat. Zato si najprej oglejmo možne izvore napak.

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

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

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

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

Nenatančni podatki Često so vhodni podatki problemov rezultat meritev ali predhodnih izračunov in zato ne morejo biti natančni. Z metodami numerične analize teh napak ne moremo odpraviti, lahko pa ugotovimo, kako vplivajo na končni rezultat. Z izbiro ustreznih numeričnih algoritmov lahko tudi do neke mere vplivamo na velikost napake, ki jo povzroča nenatančnost vhodnih podatkov.

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

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

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

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

relativna napaka pa kot

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

Relativno napako lahko podamo direktno ali v odstotkih.

Primer 1.1. Kot ilustracijo si poglejmo znani približek

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

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

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

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

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

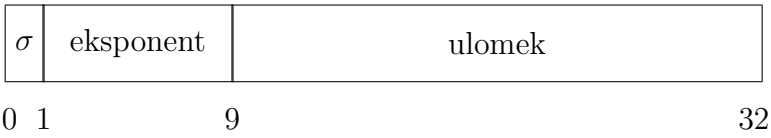
je v desetiškem zapisu vrednosti ustrezne količine pravilnih vsaj m mest. Npr. število 22/7 kot približek za število π ima točna 3 mesta.

1.3 Računalniška aritmetika

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

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

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



Slika 1.1: Zapis realnega števila v računalnikovo besedo

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

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

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

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

tako da velja

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

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

Zaokrožitvene napake se ne pojavljajo le pri vnosu realnih števil v računalnik, ampak tudi kot rezultat aritmetičnih operacij v računalniku samem. Naj nam simbol $*$ označuje poljubno aritmetično operacijo (seštevanje, odštevanje množenje ali deljenje). Namesto pravega rezultata $x * y$ bomo dobili njegov približek $fl(x * y)$. Običajno lahko privzamemo, da tudi v tem primeru velja

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

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

Primer 1.2. Izračunajmo vrednost izraza

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

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

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

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

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

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

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

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

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

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

Za relativno napako kvocienta torej velja

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

torej lahko tudi deljenje smatramo za neproblematično operacijo.

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

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

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

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

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

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

Podobno kot pri množenju in deljenju je relativna napaka linearna kombinacija relativnih napak obeh sumandov, vendar koeficienta nista več ± 1, ampak sta lahko poljubno velika. Če imata x in y isti predznak, potem sta oba koeficienta pozitivna in omejena. V tem primeru velja

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

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

1.4 Računanje vrednosti polinoma

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

Polinom je običajno podan v *naravni obliki*

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

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

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

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

Oglejmo si ta algoritem podrobneje in komentirajmo posamezne ukaze:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Tudi ta algoritem si oglejmo podrobneje in ga komentirajmo:

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

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

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

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

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

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

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

1.5 Problemi

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

2. Približno vrednost integrala

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

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

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

   potem je

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

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

3. Izračunaj vrednost funkcije

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

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

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

Kateri od obeh rezultatov ima manjšo napako?

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

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

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

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

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

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

5. Preveri veljavnost identitete

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

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

6. Vrednosti integralov

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

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

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

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

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

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

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

(c) Če rekurzivno formulo (1.6) obrnemo

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

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

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

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

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

Literatura

[1] T. J. Akai: *Numerical Methods*, John Wiley & Sons, Inc., New York 1994.

[2] K. Atkinson: *Elementary Numerical Analysis*, John Wiley & Sons, Inc., New York, 1985.

[3] Z. Bohte: *Numerične metode*, DMFA Slovenije, Ljubljana 1991.

[4] Z. Bohte: *Numerično reševanje nelinearnih enačb*, DMFA, Ljubljana 1993.

[5] Z. Bohte: *Numerično reševanje sistemov linearnih enačb*, DMFA Slovenije, Ljubljana 1994.

[6] Z. Bohte: *Uvod v numerično računanje*, DMFA Slovenije, Ljubljana 1995.

[7] S. D. Conte, C. de Boor: *Elementary Numerical Analysis - An Algoritmic Approach*, McGraw-Hill, New York 1980.

[8] B. N. Datta:*Numerical Linear Algebra and Applications*, Brooks/Cole Publishing Co., Pacific Grove 1995

[9] J. W. Demmel: *Uporabna numericna linearna algebra*, DMFA, Ljubljana 2000.

[10] J. F. Epperson:*Numerical Methods and Analysis*, John Wiley & Sons, Inc. New York 2002.

[11] G. H. Golub, C.F. van Loan: *Matrix Computations*, The John Hopkins University Press, Baltimore 1989.

[12] E. Hairer, S. P. Nørsett, G. Wanner: *Solving Ordinary Differential Equations I: Nonstiff Problems* (2nd ed.), Springer-Verlag, Berlin 1991.

[13] P. Henrici: *Discrete Variable Methods in Ordinary Differential Equations*, John Wiley & Sons, Inc, New York 1962.

[14] A. Iserles: *A First Course in the Numerical Analysis of Differential Equations*, Cambridge University Press, Cambridge 1996.

[15] J. Kozak: *Numerična analiza*, DMFA založništvo, Ljubljana 1994.

[16] G. W. Stewart: *Afternotes on Numerical Analysis*, SIAM, Philadelphia 1996.

[17] Vetterling, W.T. and Press, W.H. and Teukolsky, S.A. and Flannery, B.P.: *Numerical Recipes 3rd Edition: The Art of Scientific Computing*, Cambridge University Press, Cambridge 1996.