Tietorakenteiden ja algoritmien harjoitustyön toteutusdokumentti

Risto Tuomainen

7. syyskuuta 2014

Ohjelman käyttöliittymä kerää käyttäjältä komentoja ja välittää ne komennonkäsittelijä-luokalle, joka suorittaa laskutoimitukset käyttäen BasicMatrix ja YaleMatrix-luokkia. Näistä ensimmäinen on tavallinen taulukkomatriisi ja jälkimmäinen Yale-matriisi. Molemmat toteuttavat Matrix-rajapinnan. Taulukkomuotoisten matriisien laskutoimitukset ovat BasicMatrix-luokan metodeja, mutta varsinainen laskeminen tapahtuu Peruslasku-luokan tarjoamien metodien avulla, joita BasicMatrix-luokasta kutsutaan. Lisäksi Taulukko-luokassa on taulukkojen käsittelyyn tarkoitettuja metodeja, joita tarvitaan myös algoritmien suorittamisessa. Yale-matriisien osalta sen sijaan kaikki toiminnallisuus on rehdisti yhdessä luokassa, mutta eipä sitä toiminnallisuutta toisaalta kovin paljoa olekaan.

Strassen algoritmissa matriisien osittaminen pienempiin neljään osamatriisiin on tehty kopioimalla alkioita. Tämä ei vaikuta asymptoottiseen aikavaativuuteen, mutta on käytännön kannalta katastrofi tehokkuuden kannalta. Tässä olisikin selkeä parannuskohde, mikäli vielä kehittäisin ohjelmaa. Lisäksi harvojen matriisien kanssa toimiminen jäi tällä kertaa lapsipuolen asemaan, ja näihin liittyvien algoritmien lisääminen olisi sekin luonteva lisä työhön. Myös tietyt keskeiset laskennalliset ongelmat jäivät tämän työn osalta ratkomatta, kuten esim. yhtälöryhmien PNS-ratkaisut, ominaisarvot ja niihin liittyvät sovellutukset. Eräs keskeinen toteuttamatta jäänyt ominaisuus olisi johdonmukainen toiminta pyöristämisen suhteen. Matriisilaskennassa vaikeudet pitkien desimaalilukujen kanssa ovat tavallisia, sillä pienillä virheillä on taipumus kasaantua pitkissä laskukaavoissa, ja lisäksi on hyvin tavallista, esim. Gauss-Jordan eliminoinnissa, että tuloksen pitäisi algoritmin oikean toiminnan kannalta olla täsmällinen. Tälle ei ohjelmassa tehty mitään, paitsi Gauss-Jordan eliminoinnnissa, jossa alle 0.0001 olevat luvut tulkitaan nolliksi.

Yale-matriisin kertolasku

Yale-matriisin kertolaskualgoritmissa jokaisen tulosmatriisin $\mathbf{T}$ alkion $\mathbf{T}_{ij}$ kohdalla lasketaan kerrottavan matriisin rivin i and kertovan matriisin sarakkeen j pistetulo. Tämä tapahtuu käymällä läpi rivillä olevia alkioita ja kertomalla kukin niistä oikealla sarakevektorin alkiolla. Algoritmissa hyödynnetään lisäksi

havaintoa, että jos kerrottavassa matriisissa on vain nollia jollain rivillä, myös tulosmatriisissa on nollarivi vastaavassa kohdassa.

```
1:  procedure KERRO(ia, ja, a, B)
2:      m ← ia.length−1
3:      n ← B.sarakkaidenmäärä
4:      for i = 0 to m − 1 do
5:          l_1 ← ia_i
6:          if l_1 = −1 then
7:              for j = 0 to n − 1 do
8:                  C_ij = 0
9:              end for
10:         else
11:             c = ia.length
12:             for h = 0 to m do
13:                 if ia_h ≠ −1 then
14:                     c ← ia_h
15:                     break
16:                 end if
17:             end for
18:             for j = 0 to 1 do
19:                 l_2 ← c − 1
20:                 s ← 0
21:                 for u = l_1 to l_2 do
22:                     t = ja_u
23:                     s ← s + a_u × B_tj
24:                 end for
25:                 C_ij ← s
26:             end for
27:         end if
28:     end for
29:     Return C
30: end procedure
```

Algoritmi jäljittelee tietyllä tavalla naiivia algoritmia: siinä käydään jokainen tulosmatriisin alkio läpi ja lasketaan tälle arvo rivin ja sarakkeen pistetulona. Toisin kuin naivissa algoritmissa, jonka aikavaativuus on $O(nmp)$, tässä ei tarvitse laskea kaikki pistetulon tekijöitä läpi, vaan nollat voidaan hypätä yli. Niinpä aikavaatimus on sama kuin naiivilla, paitsi että p (joka on kerrottavan sarakkeiden ja kertojan rivien lukumäärä) korvautuu kerrottavan matriisin ei-nolla-alkioiden keskimääräisellä lukumäärällä z, ja saadaan aikavaativuudeksi $O(nmz)$. Lisäksi algoritmi ei käytä mitään aputietorakenteita, joten aikavaatimus on vakio.

Gauss-Jordan-eliminointi

Gauss-Jordan eliminointi soveltuu lineaaristen yhtälöryhmien ratkaisemiseen ja esim. kääntömatriisin laskemiseen. Toteutus oli ulkomuistista lineaarialgebran kurssin perusteella, joten erilaiset hienoudet jäivät toteuttamatta. Esimerkik-

si kunnollisissa toteutuksissa voidaan käyttää sarakkeiden vaihto-operaatiota tai suhtautua täsmällisemmin pyöristysongelmiin. Varsinaisessa toteutuksessa Gauss-Jordan on pilkottu kahtia Gaussin ja Jordanin eliminointeihin lähinnä testausta helpottamaan. Tässä kuitenkin algoritmi on esitetty yhtenä kokonaisuutena.

```
1:  procedure GAUSS-JORDAN(A)
2:      m ← A.rivienmäärä
3:      n ← A.sarakkeidenmäärä
4:      k_1 ← 0
5:      h ← 0
6:      for i = 0 to m − 1 do
7:          p = 0
8:          while p = 0 and l < n do
9:              for k = l to m − 1 do
10:                 if |A_kl| > p then
11:                     p ← A_kl
12:                     k_1 ← k
13:                 end if
14:             end for
15:         end while
16:         vaihdaRivit(A, i, k_1)
17:         l ← l + 1
18:         for h = i + 1 to m − 1 do
19:             c ← A_{i,l−1}/p
20:             for j = 0 to n − 1 do
21:                 s ← A_hj − cA_ij
22:                 if |s| < 0.00001 then
23:                     A_ij ← 0
24:                 else
25:                     A_ij ← s
26:                 end if
27:             end for
28:         end for
29:     end for
30:     for i = m − 1 to 0 do
31:         for j = 0 to n do
32:             if A_ij ≠ 0 then
33:                 l ← A_ij
34:                 break
35:             else
36:                 l ← −1
37:             end if
38:         end for
39:         if l ≠ −1 then
40:             for h = i − 1 to 0 do
41:                 c ← A_hl/A_il
42:                 for j = l to n − 1 do
43:                     s = A_ij − cA_ij
44:                     if |s| < 0.00001 then
```

```
45:                     A_ij ← 0
46:                 else
47:                     A_ij ← s
48:                 end if
49:             end for
50:         end for
51:     end if
52: end for
53: jaakukinRiviPivotillaan(A)
54: end procedure
```

Algoritmin kaksi osaa ovat aikavaatimuksen kannalta olennaisesti samoja, ja ne suoritetaan peräkkäin, joten riittää tarkastella vaikka vain Gaussin eliminoinnin aikavaatimusta. Algoritmissa jokainen rivi vähennetään sen alapuolella olevista riveistä sopivalla vakiolla kerrottuna. Rivien vähentäminen toisistaan vie aikaa suhteessa rivin pituuteen. Näiden vähennysten määräksi saadaan suunnilleen $\sum_{i=1}^{m} i$, joka tunnetun kaavan nojalla on $m(m+1)/2$. Niinpä algoritmin aikavaatimus on luokkaa $O(n(m(m+1))/2) = O(nm^2)$. Algoritmissa ei käytetä aputietorakenteita, joten sen muistivaatimus on vakio.

Strassen algoritmi

Strassen algoritmissa on seurattu tarkasti oppikirjaesitystä[1, s. 99], jota ei toisteta tässä. Myöskään aikavaativuusanalyysia, jonka johtopäätöksenä algoritmin aikavaativuus on luokkaa $O(n^{lg7})$ ei toisteta. Sen sijaan algoritmin muistivaatimus on kiinnostavampi. Harjoitustyössä algoritmi toteutettiin siten, että matriisi ositettiin neljään osaan kopioimalla sen arvot neljään uuteen pienempään matriisiin. Lopussa nämä koottiin yhteen kopioimalla kaikkien arvot jälleen uuteen, suureen matriisiin. Niinpä muistia tarvitaan enemmän tai vähemmän: en nimittäin nyt saanut laskettua muistivaatimusta.

Viitteet

[1] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms.* The MIT Press, 2009.