Toteutusdokumentti

Juuso Nyyssönen

Matriisilaskimen yleisrakenne

Ohjelman logiikka koostuu kolmesta eri luokasta, jotka ovat Fraction, Matrix ja LUPDecomposition. Fraction on murtolukuja esittävä luokka, jota LUP-Decomposition käyttää laskuissaan. Matrix on taas matriisien peruslaskutoimitukset sisältävä luokka. Se hoitaa siis kokonaislukumatriisien yhteenlaskun, vähennyslaskun, skalaarikertolaskun ja kertolaskun. Lisäksi se hoitaa myös Strassenin kertolaskualgoritmin toteutuksen. LUPDecomposition on luokka, joka hoitaa annetun Matrix-olion LUP-hajotelman laskemisen. Tämän lisäksi luokka sisältää metodit, jotka laskevat determinantin ja käänteismatriisin LUP-hajotelman avulla.

CalculatorUI on käyttöliittymäluokka, joka hoitaa tekstikäyttöliittymän toiminnan.

Toteutetut algoritmit ja niiden aika- ja tilavaativuudet

Ohjelma toteuttaa kokonaislukumatriisien peruslaskutoimitukset, eli yhteenlaskun, vähennyslaskun, skalaarikertolaskun ja kertolaskun. Näistä yhteenlasku, vähennyslasku ja skalaarikertolasku tekevät yhden laskutoimituksen per matriisin alkio, joten näiden algoritmien aikavaativuus on $\mathcal{O}(mn)$ $m \times n$-matriiseille. Algoritmien tilavaativuudet ovat myös luokkaa $\mathcal{O}(mn)$.

Algorithm 1 Matriisikertolasku

```
1: function MUL(A, B)
2:     let C be an m × p matrix
3:     for i ← 1 to m do
4:         for k ← 1 to p do
5:             C_ik ← 0
6:             for j ← 1 to n do
7:                 C_ik ← C_ik + A_ij · B_jk
8:             end for
9:         end for
10:    end for
11: end function
```

Yllä matriisikertolaskun pseudokoodi $m \times n$-matriisille A ja $n \times p$-matriisille B. Pseudokoodi sisältää kolme sisäkkäistä for-looppia ja kunkin loopin sisällä tehdään vakiomäärä työtä, joten algoritmin aikavaativuus on $\mathcal{O}(mnp)$. Tilavaativuus on $\mathcal{O}(mp)$.

Lisäksi ohjelma toteuttaa neliömatriisin LUP-hajotelman laskemisen. Alla sen pseudokoodi $n \times n$-matriisille A. LU-matriisi kootaan yhteen taulukkoon LU siten, että taulukon diagonaalin alapuolella olevat alkiot ovat alakolmiomatriisin L alkioita ja diagonaalilla ja sen yläpuolella olevat alkiot ovat yläkolmiomatriisin U alkioita. Matriisi L on normitettu siten, että sen diagonaalilla olevat alkiot ovat ykkösiä. Permutaatiomatriisi on esitetty yksiulotteisena taulukkona P. Kolmen sisäkkäisen for-loopin rakenteen perusteella LUP-hajotelman aikavaativuus on $\mathcal{O}(n^3)$. Tilavaativuus on $\mathcal{O}(n^2)$.

Algorithm 2 LUP-hajotelma

```
1: function LUP-Decomposition(A)
2:     LU ← A
3:     let P be an array of length n
4:     for i ← 1 to n do
5:         P[i] ← i
6:     end for
7:     for k ← 1 to n do
8:         k′ ← 0
9:         for i ← k to n do
10:            if LU_ik ≠ 0 then
11:                k′ ← i
12:                break
13:            end if
14:        end for
15:        if k′ = 0 then
16:            error "singular matrix"
17:        end if
18:        exchange P[k] ↔ P[k′]
19:        for i ← 1 to n do
20:            exchange LU_ki ↔ LU_k′i
21:        end for
22:        for i ← k + 1 to n do
23:            LU_ik ← LU_ik/LU_kk
24:            for j ← k + 1 to n do
25:                LU_ij ← LU_ij − LU_ikLU_kj
26:            end for
27:        end for
28:    end for
```

Matriisin determinantti ja sen käänteismatriisi lasketaan ylläolevaa LUP-hajotelmaa käyttäen. $n \times n$ neliömatriisin A determinantti voidaan laskea laskemalla sen LUP-hajotelma ja kertomalla LU-hajotelman diagonaalialkiot keskenään ja kertomalla tämä vielä permutaatiomatriisin P merkillä. Diagonaalialkioiden kertominen keskenään ja permutaatiomatriisin merkin löytäminen onnistuvat lineaarisessa ajassa, joten aikavaativuuden määrä LUP-hajotelman laskemisen aikavaativuus. Siis determinantin laskemisen aikavaativuus on luokkaa $\mathcal{O}(n^3)$. Tilavaativuus on $\mathcal{O}(n^2)$.

$n \times n$ neliömatriisin A käänteismatriisi löydetään laskemalla matriisin A

LUP-hajotelma ja ratkaisemalla yhtälö $LUX = P$. Tämä yhtälö ratkaistaan ratkaisemalla yhtälö $LUX_k = P_k$ jokaiselle X:n sarakkeelle X_k ja permutaatiomatriisin P sarakkeelle P_k erikseen. Yhtälön $LUX_k = P_k$ ratkaisu onnistuu jokaisella k ajassa $\mathcal{O}(n^2)$ ns. forward ja backward substitutionia käyttäen. Tämä joudutaan tekemään jokaiselle sarakkeelle erikseen, joten kaikenkaikkiaan aikaa menee $\mathcal{O}(n^3)$. Siis käänteismatriisin laskemisen aikavaativuus on $\mathcal{O}(n^3)$. Tilavaativuus on jälleen $\mathcal{O}(n^2)$.

Ohjelmassa on toteutettu lisäksi Strassenin kertolaskualgoritmi neliömatriiseille. Sen ideana on jakaa annetut $n \times n$ neliömatriisit A ja B neljään yhtäsuureen $n/2$:n kokoiseen osaan $A_{11}, B_{11}, A_{12}, B_{12}, A_{21}, B_{21}, A_{22}$ ja B_{22}. Näiden matriisien avulla lasketaan ajassa $\mathcal{O}(n^2)$ näitä matriiseja yhteenlaskemalla ja vähentämällä 14 $n/2$:n kokoista alimatriisia $A_1, B_1, A_2, B_2, \ldots, A_7, B_7$. Tämän jälkeen lasketaan Strassenin algoritmilla rekursiivisesti 7 kertolaskua $P_i = A_i B_i$, jossa $i = 1, 2, \ldots, 7$. Tulomatriisi saadaan laskettua näistä seitsemästä matriisista P_i ajassa $\mathcal{O}(n^2)$ laskemalla ja vähentämällä näitä matriiseja yhteen.

Täten Strassenin algoritmin aikavaativuus $T(n)$ toteuttaa yhtälön

$$T(n) = 7T(n/2) + \mathcal{O}(n^2). \tag{1}$$

Nähdään (esim. Master Theoremin avulla), että tällöin $T(n) = \mathcal{O}(n^{log_2 7})$. Algoritmi toimii kuitenkin vain neliömatriiseille, joiden koko on kakkosen potenssi. Ongelma saadaan korjattua täydentämällä matriisit A ja B seuraavaan kakkosen potenssiin täyttämällä nollilla. Pahimmassakin tapauksessa täytetty matriisi on kuitenkin vain luokkaa $2n \times 2n$, joten algoritmin tilavaativuus on $\mathcal{O}(n^2)$.

Puutteita ja parannusehdotuksia

Strassenin algoritmi olisi mahdollista toteuttaa myös ei-neliömatriiseille. Lisäksi algoritmin voisi toteuttaa myös periaatteessa täydentämättä annettuja syötematriiseja seuraavaan kakkosen potenssiin. Tämä parantaisi ainakin algoritmin tilavaativuutta.

Matrix-luokan yhteenlaskumetodit voisi toteuttaa myös murtolukumatriiseille. Koodi ei olennaisesti muuttuisi tässä tapauksessa.

Ohjelman käyttöliittymä on aika kämäinen. Vähänkään isompien matriisien syöttäminen on aika työlästä ja syötteiden kirjoittamisen kanssa pitää olla tarkkana. Graafinen käyttöliittymä parantaisi asioita huomattavasti.

Laskin voisi sisältää myös lisää toimintoja, kuten esimerkiksi ominaisarvojen (tai oikeastaan niiden likiarvojen) laskemisen. Toinen idea olisi esimerkiksi diagonalisoituvuuden tarkastamisen mahdollistaminen.

Ohjelman luokkajako on myös vähän satunnainen. Refaktorointi voisi selkeyttää asioita.

Lähteet

Cormen, Thomas H.; Leiserson, Charles E; Rivest, Ronald L.:

Introduction to algorithms

1st edition, MIT Press 1999