Toteutusdokumentti

Lasse Lybeck

27. helmikuuta 2013

1 Ohjelman yleisrakenne

2 Saavutetut aika- ja tilavaativuudet

2.1 Yhteenlasku

```
function add(Matrix a, Matrix b)
    for i = 1 .. a.rows
        for j = 1 .. a.columns
            result(i,j) = a(i,j) + b(i,j)
    return result
```

Kahden $m \times n$ matriisin yhteenlaskun aikavaatimukseksi saadaan selvästi $O(mn)$. Jos kyseessä on kahden $n \times n$ neliömatriisin yhteenlasku saadaan siis aikavaatimukseksi $O(n^2)$.

Vähennyslasku on täysin ekvivalentti yhteenlaskun kanssa, joten vaativuudet ovat samat.

2.2 Kertolasku

2.2.1 Skalaarilla kertominen

```
scale(Matrix m, float k)
    for i = 1 .. m.rows
        for j = 1 .. m.columns
            m(i,j) *= k
```

$m \times n$ matriisin skalaarilla kertominen on selvästi aikavaativuudeltaan $O(mn)$. $n \times n$ neliömatriisille tämä siis on $O(n^2)$.

2.2.2 Matriisikertolasku

```
mul(Matrix a, Matrix b)
    for i = 1 .. a.rows
        for j = 1 .. b.columns
            result(i,j) = 0
            for k = 1 .. a.columns
                result(i,j) += a(i,k) * b(k,j)
    return result
```

Matriisikertolaskussa ensimmäisen matriisin sarakkeiden määrä tulee olla sama kuin toisen matriisin rivien määrä. Matriisien $A \in \mathbb{R}^{m \times n}$ ja $B \in \mathbb{R}^{n \times p}$ tuloksena saadaan siis matriisi $AB \in \mathbb{R}^{m \times p}$. Naiivia algoritmia (yllä) seuraamalla aikavaativuudeksi saadaan $O(mnp)$, joka neliömatriisien $A, B \in \mathbb{R}^{n \times n}$ tapauksessa on $O(n^3)$.

Kappaleessa 3 sivulla 4 verrataan naiivia algoritmia Strassenin asymptoottisesti nopeampaan algoritmiin.

2.3 Potenssiin korottaminen

```
pow(Matrix a, int e)
    if e == 1
        return a
    else if e mod 2 == 0
        return pow(a * a, e/2)
    else
        m * pow(a * a, (e-1)/2)
```

2.4 LU-hajotelma ja determinantti

```
LU(Matrix m)
    U = copy(m)
    for i = 1 .. n
        for j = i + 1 .. n
            c = -U(j,i) / U(i,i)
            addMulRow(U, i, j, c)
            L(j,i) = -c
    return L, U

addMulRow(Matrix m, i, j, c)
    for k = 1 .. n
```

```
        m(j,k) += c * m(i,k)
```

2.5 Käänteismatriisin laskeminen

```
inv(Matrix m)
    L,U = LU(m)
    e = eye(n,n)
    Vector[] eCols = getColumns(e)
    for i = 1 .. n
        xCols[i] = solveSystem(L, U, e)
    return matrixFromColumns(xCols)

solveSystem(Matrix L, Matrix U, Matrix e)
    z = forwardSubstitute(L, e)
    x = backwardSubstitute(U, z)
    return x

forwardSubstitute(Matrix L, Matrix e)
    x = new Vector(n,1)
    x(1,1) = e(1,1)
    for i = 1 .. n
        sum = 0
        for j = 1 .. i - 1
            sum += L(i,j) * x(j,1)
        x(i,1) = e(i,1) - sum
    return x

backwardSubstitute(Matrix U, Matrix e)
    x = new Vector(n,1)
    x(n,1) = e(n,1) / U(n,n)
    for i = n - 1 .. 1
        sum = 0
        for j = n .. i + 1
            sum += U(i,j) * x(j,1)
        x(i,1) = (e(i,1) - sum) / U(i,i)
    return x
```

3 Suorituskyky- ja O-analyysivertailu

4 Työn mahdolliset puutteet ja parannusehdotukset

5 Lähteet