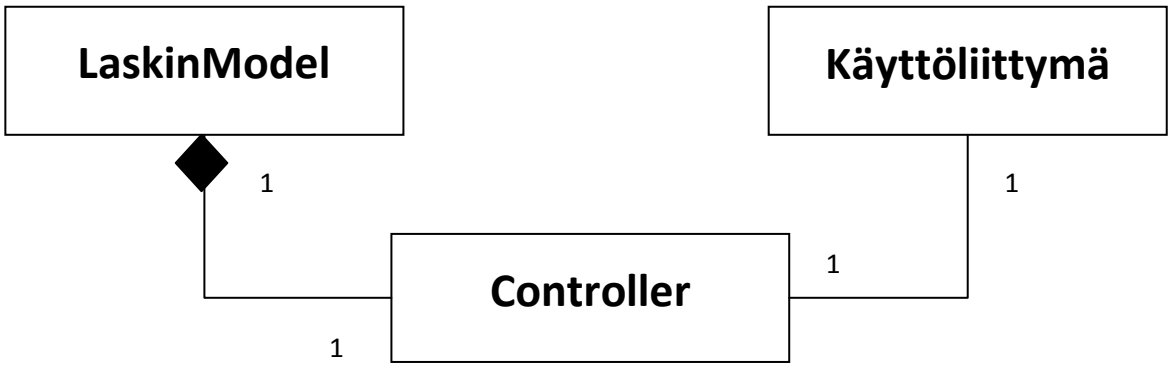
Tiralabra loppukesä 2013
Ilkka Vähämaa
013581244
ilkka.vahamaa@helsinki.fi
https://github.com/Ikriva/TiraLabra

Toteutusdokumentti Matriisi-laskin

1 Ohjelman yleisrakenne



Ohjelma käynnistyy Controllerin luomisella, joka luo laskimen ja Käyttöliittymän. Controller käsittelee käyttöliittymältä tulleet syötteet, muuttaa ne oikeaan muotoon laskinta varten ja laskin toteuttaa laskemisen. Tämän jälkeen se palauttaa controllerille tuloksen, mikä muuttaa sen taas oikeaan muotoon käyttöliittymää varten ja antaa sille esitettävän tuloksen.

Tiralabra loppukesä 2013
Ilkka Vähämaa
013581244
ilkka.vahamaa@helsinki.fi
https://github.com/Ikriva/TiraLabra

2 Saavutetut aika- ja tilavaativuudet

2.1 Matriisin kertominen skalaarilla

```
public double[][] kerroSkalaarilla(double[][] matriisi, double skalaari) {
        for (int i = 0; i < matriisi.length; i++) {          // √n
            for (int j = 0; j < matriisi[0].length; j++) {   // √n
                matriisi[i][j] *= skalaari;                  // vakio
            }
        }
        return matriisi;
    }
```

Aikavaativuus $O(\sqrt{n}^2) \approx O(n)$. Tilavaativuus O(1), koska apumuuttujia tai uusia taulukoita ei tarvita.

2.2 Matriisien yhteenlasku

Aikavaativuus on tehty olettaen syötteenä olevat matriisit n-alkioisiksi neliömatriiseiksi.

```
public double[][] laskeYhteen(double[][] matriisiYksi,double[][] matriisiKaksi){
         for (int i = 0; i < matriisiYksi.length; i++) {          // √n
             for (int j = 0; j < matriisiKaksi[0].length; j++) {  // √n
                 matriisiYksi[i][j] += matriisiKaksi[i][j];       // vakio
             }
         }
         return matriisiYksi;
```

Aikavaativuus $O(\sqrt{n}^2) \approx O(n)$. Tilavaativuus O(1), koska apumuuttujia tai uusia taulukoita ei tarvita.

2.3 Matriisin kertominen toisella matriisilla

Aikavaativuus on tehty olettaen syötteenä olevat matriisit n-alkioisiksi neliömatriiseiksi.

Apumetodi:

```
private int laskeAlkio(int rivi, int sarake, double[][] matriisi1, double[][]
matriisi2) {
        int alkio = 0;
        for (int k = 0; k < matriisi1.length; k++) {          // √n
            alkio += (matriisi1[rivi][k] * matriisi2[k][sarake]);  // vakio
        }
        return alkio;
    }
```

Varsinainen metodi:

```
public double[][] kerroMatriisit(double[][] matriisi1, double[][] matriisi2)
if (matriisi1.length == matriisi2[0].length) {
         double[][] uusi = new double[matriisi1.length][matriisi2[0].length];
```

Tiralabra loppukesä 2013
Ilkka Vähämaa
013581244
ilkka.vahamaa@helsinki.fi
https://github.com/Ikriva/TiraLabra

```
        for (int rivi = 0; rivi < uusi.length; rivi++) {            // √n
         for (int sarake = 0; sarake < uusi[0].length; sarake++) {   // √n
          uusi[rivi][sarake] = laskeAlkio(rivi,sarake,matriisi1,matriisi2);//√n
             }
         }

           return uusi;
    }
```

Yhteensä aikavaativuus enintään $O((\sqrt{n})^3) \approx O(n^2)$

Tilavaativuus apumuuttujien määrän ollessa vakio on vaadittava uusi n-alkioinen matriisi eli O(n).

2.4 Matriisin determinantin määrittäminen

Determinantti määritetään tekemällä matriisille ensin LU-dekompositio Doolittlen algoritmilla. Tämän jälkeen determinantti saadaan laskemalla ylemmän kolmiomatriisin halkaisija. LU-dekompositioon on valittu ylemmän kolmiomatriisin palauttava metodi, sillä se on koodiltaan selkeämmin hahmotettava kuin samaan matriisin tallentava metodi.

2.4.1 LU-dekompositio - Doolittlen algoritmi

```
public double[][] luDekompositioDoolittleYlempiKolmiomatriisi(double[][]
matriisi)
           double[][] u = new double[matriisi.length][matriisi.length];
           double[][] l = new double[matriisi.length][matriisi.length];

            for (int i = 0; i < matriisi.length; i++) {           // √n
                for (int j = i; j < matriisi.length; j++) {         // √n
                    u[i][j] = matriisi[i][j];                       // vakio
                    for (int k = 0; k < i; k++) {                     // √n
                        u[i][j] = u[i][j] - l[i][k] * u[k][j];         // vakio
                    }
                }
                for (int j = i + 1; j < matriisi.length; j++) {     // √n
                    l[j][i] = matriisi[j][i];                       // vakio
                    for (int k = 0; k < i; k++) {                     // √n
                        l[j][i] = l[j][i] - l[j][k] * u[k][i];         // vakio
                    }
                    l[j][i] = l[j][i] / u[i][i];                    // vakio
                }
            }
            return u;
```

Kaikki algoritmin loopit toistetaan korkeintaan matriisin sivun pituuden verran, eli suhteessa alkioiden määrään $\sqrt{n}$ verran. Tällöin koko algoritmin aikavaativuus on korkeintaan:

$$O(\sqrt{n} \times \left(\sqrt{n} \times \left(\sqrt{n} \times vakio\right) + \sqrt{n} \times \left(\sqrt{n} \times vakio\right)\right)) = O(\sqrt{n} \times 2n) \approx O(n^2)$$

Tiralabra loppukesä 2013
Ilkka Vähämaa
013581244
ilkka.vahamaa@helsinki.fi
https://github.com/Ikriva/TiraLabra

Käytännössä kaksi sisempää looppia toistuu ainoastaan ensimmäisellä iteraatiolla $\sqrt{n}$ kertaa ja tämän jälkeen pienemmissä loopeissa. Valitettavasti kuitenkaan käytännön suorituskykytestaus ei luotettavasti onnistunut, vaan metodin suoritusaika nopeutui jatkuvasti johtuen todennäköisesti välimuistista.

2.4.2 Determinantin määrittäminen kolmiomatriiseista

```
public double laskeDeterminantti(double[][] matriisi)
  double[][] laskettavaKolmiomatriisi =
luDekompositioDoolittleYlempiKolmiomatriisi(matriisi); // O(n^2)
        if (laskettavaKolmiomatriisi == null) {
            return 0;
        }
        double determinantti = 1;
        for (int i = 0; i < laskettavaKolmiomatriisi.length; i++) {     //√n
         determinantti = determinantti * laskettavaKolmiomatriisi[i][i]; //vakio
        }
        return determinantti;
```

Determinantin määrittäminen kolmiomatriisista tapahtuu laskemalla diagonaalin tulo. Determinantin määritelmän mukaisesti $det\ A\ =\ (det\ L)\ \times (det\ U)$. Koska alemman kolmiomatriisin diagonaali on ykkösiä, on sen tulo aina yksi. Tällöin determinantin saa suoraan laskemalla ylemmän kolmiomatriisin determinantti. Diagonaalin tulon laskeminen aikavaativuudeltaan $O(\sqrt{n})$.

Yhteensä determinantin laskemisen aikavaativuus on $O(n^2 + \sqrt{n}) \approx O(n^2)$.

Tilavaativuus on $O(2n) \approx O(n)$, koska apumuuttujia ei tarvita, vaan ainoastaan kaksi uutta n-alkioista matriisia.

3 Puutteet ja parannusehdotukset

Parannuksena tehokkuuteen olisi käyttöliittymä, joka ottaa jo valmiiksi liukulukutaulukkona matriisin eikä tällöin tarvittaisi controllerissa paljon String ja double olioiden välisiä muutoksia.

4 Lähteet

Introduction to Algorithms Third edition 2009 (Thomas H. Cormen etc.)