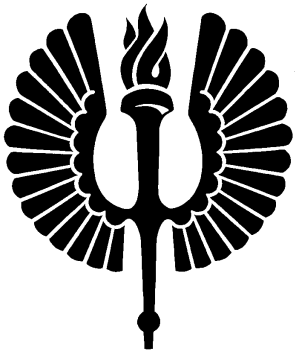


NUMEERISIA MENETELMIÄ JAVA-KIELELLÄ

Ari Heino

Pro gradu -tutkielma
Joulukuu 2005

UNIVERSITY OF TURKU
DEPARTMENT OF MATHEMATICS
FIN-20014 TURKU
FINLAND

Sisältö

1 Johdanto

Java-ohjelmointikieltä ja erityisesti sillä luotavia appletteja on perinteisesti pidetty lähinnä web-sivuja rikastuttavana elementtinä. Fortran, C ja C++ ovat kieliä, joita on totuttu käyttämään matemaattisissa ohjelmointitöissä. Mahdollisesti nuoresta iästään johtuen on Javan soveltuminen tämänkaltaisiin tehtäviin jäänyt vähemmälle huomiolle. Tässä työssä tutkitaan, miten Java-kielellä sujuvat esimerkiksi lineaarialgebran tehtävät, numeerinen integrointi sekä derivointi. Lukuisia tehtäviä esitellään ja niihin annetaan ratkaisuesimerkkejä Java-kielen keinoin. Osa tehtävistä perustuu Helsingin yliopistossa keväällä 1999 pidetyn kurssin 'Numeeriset menetelmät ja C-kieli' [23] harjoitustehtäviin.

Javan valmiit luokkakirjastot eivät ole riittäviä vaativiin matemaattisiin tehtäviin, joten sellaisia varten on luotava omia ohjelmia ja/tai käytettävä muualla luotuja luokkakirjastoja. Tässä esitellyissä tehtävissä, erityisesti lineaarialgebran osassa, on käytetty JAMA-kirjastoa [3] , joka on tehty matriisien monipuoliseen käsittelyyn.

Myös numeerisen datan esittäminen halutulla tavalla (muotoilu, tasaus) on ollut Javalla hankalaa. Kielen uusin versio sisältää `printf()`-metodin, jolla tulostusta voidaan muotoilla esimerkiksi C-kielestä tutulla tavalla. Nyt annettavat esimerkit on kuitenkin kirjoitettu käyttämällä Java-kielen vanhempaa versiota. Ongelman ratkaisuksi löytyivät Henrik Bengtssonin tekemät luokat [1] , joiden avulla päästään samaan lopputulokseen.

Kolmas tärkeä luokkakirjasto oli JSGL [5] , joka sisältää luokkia erilaisten kuvaajien piirtämiseen.[1] Sillä on myös helppo toteuttaa ohjelmia, jotka vaativat interaktiivisuutta käyttäjän kanssa.

Osa funktioita käsittelevistä ohjelmista on toteutettu niin, että käyttäjä voi itse syöttää haluamansa funktion. Tämän toteuttamiseksi on käytetty JeksParser-kirjastoa [4] , jonka luokat mahdollistavat funktion muodostamisen merkkijonosyötteestä ja sen evaluoinnin annetussa pisteessä.

[1] Esimerkkien yhteydessä olevat kuvat poikkeavat painoteknisistä syistä värien ja koon puolesta siitä, mitä ne todellisuudessa ovat.

Interpolointiin löytyi hyödyllisiä luokkia osoitteesta [2].

Tutkielmassa esitetyt teoriaosuudet (ja myös osa ohjelmista) pohjautuvat eri numeerista laskentaa käsitteleviin kirjoihin ja verkkodokumentteihin, jotka on lueteltu tutkielman lopussa kirjallisuusluettelossa. Osa lähteistä käsittelee aihetta yleisesti, osa tietyn ohjelmointikielen näkökulmasta.

Esimerkkiohjelmien lisäksi on luotu muutamia apuluokkia, lähinnä lineaarialgebran tarpeisiin. Kaikki ohjelmat perivät `Base`-nimisen luokan, johon on koottu usein tarvittuja metodeja. Osa toteutetuista ohjelmista sisältää valmiita yksittäisiä metodeja, joiden lähteet on mainittu. Omien apuluokkien lähdekoodit ovat tutkielman lopussa liitteenä.

Tutkielmaan liittyy kirjallisen osan lisäksi CD-ROM-levy sekä internetsivusto [6], joilta löytyvät itse tutkielma, suoritettavat esimerkkiohjelmat lähdekoodeineen sekä käytettyjen valmisohjelmien lähdekoodit, joita laajuudestaan johtuen ei katsottu järkeväksi liittää kirjalliseen osaan.

2 Perusasioita numeerisista menetelmistä

2.1 Menetelmistä ja virhelähteistä

Monet matemaattiset tehtävät ovat niin vaikeita tai muuten luonteeltaan sellaisia, ettei niitä voida ratkaista analyyttisesti. Tällaisia ongelmia varten on kehitetty erilaisia *numeerisia menetelmiä*. Niillä approksimoidaan ja yksinkertaistetaan ongelmia niin, että ne ovat laskettavissa tietokoneella, riittävällä tarkkuudella. Esimerkkinä numeerinen integrointi, jossa integraalia approksimoidaan äärellisellä summalla. Yleisimmistä tehtävätyypeistä, kuten juuri integroinnista, on kehitetty monia algoritmeja, jotka toimivat ihanteellisesti juuri tietyntyyppisille ongelmille — integrointi äärettömän välin yli, jaksollisen funktion integrointi jne.

Numeerisia menetelmiä käytettäessä tai ylipäätään koneilla laskettaessa ei voida välttyä erityyppisiltä virheiltä. Virheet johtuvat joko menetelmistä tai käytetystä laitteistosta. Menetelmävirheitä syntyy esimerkiksi, jos kuvattavasta ilmiöstä luodaan liian yksinkertainen matemaattinen malli tai mallissa käytetyt lukuarvot ja vakiot ovat väärin valittuja.

Laitteistovirheitä syntyy, kun laskenta annetaan koneiden tehtäväksi. Tietokoneet toimivat äärellisellä tarkkuudella, joten niillä ei välttämättä päästä tarkkoihin tuloksiin. Lisäksi ne käyttävät sisäisessä laskennassaan binäärilukuja, joten esimerkiksi desimaalilukua $0{,}1_{10}$ ei voida esittää tarkasti, koska kaksikantaisessa lukujärjestelmässä sitä esittää päättymätön jaksollinen binääriluku $0{,}000110011_2\ldots$. Tästä seuraa pyöristysvirheitä yksinkertaisissakin laskuissa. Java-kielessä ongelma on periaatteessa kierrettävissä luokkien `BigInteger` ja `BigDecimal` (jotka mallintavat rajattoman tarkkuuden kokonais- ja reaalilukuja) avulla, mutta tällöin koodista tulee kömpelöä ja ohjelman suoritus hidastuu.

Numeerisia menetelmiä kehitettäessä pyritään tunnetut virhetekijät minimoimaan ottamalla huomioon esimerkiksi paras laskujärjestys: Jos on laskettava $a+b-c$, voi järkevin tapa (lukujen suuruusluokasta riippuen) ollakin $a+(b-c)$. Vastaavasti lausekkeen $\sqrt{x^2+y^2}$ arvo kannattaa laskea muo-

dossa $|x| \cdot \sqrt{1+(y/x)^2}$, kun x ja y ovat itseisarvoltaan suuria. Liukulukujen esitystarkkuuteen liittyen: jos on laskettava pitkä summalauseke, esimerkiksi $\sum_{i=1}^{n} 1/n$, on summaus järkevintä aloittaa pienimmästä luvusta. Tällöin vältytään eri suuruusluokkaa olevien lukujen yhteenlaskuilta, jotka heikentävät tuloksen tarkkuutta.

Perusteellista tietoa virhetarkastelusta ja numeerisesta laskennasta yleisesti löytyy esim. kirjasta [21].

2.2 Liukulukuaritmetiikasta

Tietokoneiden äärellisestä muistista ja tehosta johtuen reaalilukuja ja niiden laskutoimituksia ei voida esittää tarkasti, vaan on käytettävä *liukulukuaritmetiikkaa*. Java-kielessä laskentaa voidaan suorittaa kahden tyyppisillä liukuluvuilla: *Perustarkkuuden* liukuluvut, joita vastaa tyyppi `float`, ovat 32 bitin kokoisia ja niiden (positiivinen) arvoalue on noin $1{,}4 \cdot 10^{-45} \ldots 3{,}4 \cdot 10^{38}$. *Kaksoistarkkuuden* liukuluvut (`double`) ovat 64 bitin kokoisia ja niiden arvoalue on noin $4{,}9 \cdot 10^{-324} \ldots 1{,}8 \cdot 10^{308}$. Tutkielman esimerkit on toteutettu 64-bittisiä liukulukuja käyttäen. Tämä siitäkin syystä, että Java-kielen sisäiset matemaattiset funktiot käyttävät niitä.

Tietokoneille voidaan laitteistokohtaisesti määritellä ns. *konevakio* tai *kone-epsilon* ϵ_{kone}, joka on pienin liukuluku, joka toteuttaa epäyhtälön

$$1{,}0 + \epsilon_{\text{kone}} > 1{,}0\,.$$

Luku kuvaa tietokoneen suhteellista aritmeettista tarkkuutta. Konevakio voidaan laskea seuraavan esimerkin tapaan:

Ohjelmaesimerkki 2.1

```
package examples;

// konevakion määritys

import com.braju.format.*;

public class MachineEpsilon extends Base {
  public void main() {
    showText();
    println("Lasketaan konevakio käyttäen\n"
      +"32- ja 64-bittisiä liukulukuja.\n");

    float fEps=1.0f;
```

```
    do {
      fEps/=2.0f;
    } while (1.0f+fEps>1.0f);

    println(Format.sprintf("Konevakio (32-bit) = %-10.4e",
      new Parameters(fEps)));

    double dEps=1.0d;

    do {
      dEps/=2.0d;
    } while (1.0d+dEps>1.0d);

    println(Format.sprintf("Konevakio (64-bit) = %-10.4e",
      new Parameters(dEps)));
  }
}
```

Ohjelman tulostus:

```
Lasketaan konevakio käyttäen
32- ja 64-bittisiä liukulukuja.

Konevakio (32-bit) = 5.9605e-08
Konevakio (64-bit) = 1.1102e-16
```

Konevakio kertoo, millaiseen tarkkuuteen eri algoritmeilla voidaan päästä. Toisin sanoen, jos esim. funktion nollakohdalle x_0 on laskettu likiarvo $x_0{}'$, on syytä olettaa, että $|x_0 - x_0{}'| \geq \epsilon_{\text{kone}}$, koska tietokone ei yleisessä tapauksessa enää erota näitä lukuja toisistaan (olettaen, että $|x_0| > 1$).

2.3 Kiintopistemenetelmä

Matemaattiset tehtävät voidaan usein ratkaista monella eri tavalla. Tarkastellaan esimerkkinä erään juurenhakumenetelmän variaatioita, jotka poikkeavat numeerisilta suorituskyvyiltään.

Kiintopistemenetelmän nimitys tulee siitä, että yhtälön $x = g(x)$ juuri ei muutu kuvauksessa g; se on kuvauksen *kiintopiste*. Mielivaltainen yhtälö $f(x) = 0$ voidaan palauttaa edelliseen tilanteeseen huomaamalla, että yhtälöt $f(x) = 0$ ja $g(x) = x + \alpha f(x) = x$, $\alpha \neq 0$, ovat ekvivalentteja. Tästä saadaan iteraatio

$$x_{k+1} = g(x_k) = x_k + \alpha f(x_k)\,.$$

Iteraatiota jatketaan, kunnes saavutetaan riittävä tarkkuus tai on suoritettu riittävän monta iteraatiota (estetään ikuinen silmukka).

Menetelmä voidaan johtaa muillakin tavoilla. Onkin suositeltavaa käyttää useampia tapoja, sillä iteraatio ei aina suppene tai voi supeta hyvin hitaasti. Se, mikä menetelmä milloinkin suppenee, seuraa *Banachin kiintopistelausees-ta* ([11] , s. 205):

Lause 1 (Banachin kiintopistelause). *Jos g on kutistava kuvaus välillä $[a, b]$, niin pisteestä $x_0 \in [a, b]$ alkava kiintopisteiteraatio suppenee kuvauksen g yksikäsitteiseen kiintopisteeseen $x^* \in [a, b]$.*

Tutkitaan yhtälöä $x^3 + 4x^2 - 10 = 0$, jolla on yksikäsitteinen juuri välillä $[1, 2]$. Tarkasteltava yhtälö saadaan esimerkiksi seuraaviin muotoihin:

$$
\begin{aligned}
x &= g_1(x) = x - x^3 - 4x^2 + 10 \\
x &= g_2(x) = \sqrt{\frac{10}{x} - 4x} \\
x &= g_3(x) = \frac{1}{2}\sqrt{10 - x^3} \\
x &= g_4(x) = \sqrt{\frac{10}{4 + x}} \\
x &= g_5(x) = x - \frac{x^3 + 4x^2 - 10}{3x^2 + 8x}
\end{aligned}
$$

Alin kaava on *Newtonin menetelmän* (kappale 2.5) mukainen. Aloitetaan iteraatio kohdasta $x_0 = 1{,}5$ ja listataan suppenevien menetelmien antamat juuren likiarvot 20 ensimmäiseltä iteraatiokierrokselta.

Ohjelmaesimerkki 2.2

```
package examples;

// kiintopistemenetelmä
// FPI = Fixed Point Iteration

import com.braju.format.*;

public class FPI extends Base {
  public void main() {
    showText(80, 30);
    println("Etsitään yhtälön x^3+4x^2-10=0 juuri "
```

```
    +"väliltä [1,2].\nKäytetään viittä erilaista yhtälöä,\n"
    +"joiden kiintopisteenä on ko. juuri.\n");

  boolean converging[]=new boolean[5];
  double a[][]=new double[21][5];

  java.util.Arrays.fill(converging, true);
  java.util.Arrays.fill(a[0], 1.5);

  for (int i=1, n=a.length; i<n; i++) {
    for (int j=0, m=a[i].length; j<m; j++) {
      if (converging[j]) {
        a[i][j]=val(a[i-1][j], j);

        if (Math.abs(a[i][j]-1.5)>1.0) {
          converging[j]=false;
          println("iteraatio "+(j+1)+" ei suppene.");
        }
      }
    }
  }

  for (int i=0, n=a[0].length; i<n; i++) {
    if (converging[i]) {
      print(Format.sprintf("%11s ",
        new Parameters("iteraatio "+(i+1))));
    }
  }

  println();

  for (int i=0, n=a.length; i<n; i++) {
    for (int j=0, m=a[i].length; j<m; j++) {
      if (converging[j]) {
        print(Format.sprintf("%11.6f ", new Parameters(a[i][j])));
      }
    }

    println();
  }
}

public double val(double x, int i) {
```

```
    switch (i) {
      case 0: return x*(1.0-x*(x-4.0))+10.0;
      case 1: return Math.sqrt(10.0/x-4.0*x);
      case 2: return 0.5*Math.sqrt(10.0-x*x*x);
      case 3: return Math.sqrt(10.0/(4.0+x));
    }

    return x-(x*x*(x+4.0)-10.0)/(x*(3.0*x+8.0));  // case 4
  }
}
```

Ohjelman tulostus:

```
Etsitään yhtälön x^3+4x^2-10=0 juuri väliltä [1,2].
Käytetään viittä erilaista yhtälöä,
joiden kiintopisteenä on ko. juuri.

iteraatio 1 ei suppene.
iteraatio 2 ei suppene.
iteraatio 3 iteraatio 4 iteraatio 5
   1.500000    1.500000    1.500000
   1.286954    1.348400    1.373333
   1.402541    1.367376    1.365262
   1.345458    1.364957    1.365230
   1.375170    1.365265    1.365230
   1.360094    1.365226    1.365230
   1.367847    1.365231    1.365230
   1.363887    1.365230    1.365230
   1.365917    1.365230    1.365230
   1.364878    1.365230    1.365230
   1.365410    1.365230    1.365230
   1.365138    1.365230    1.365230
   1.365277    1.365230    1.365230
   1.365206    1.365230    1.365230
   1.365242    1.365230    1.365230
   1.365224    1.365230    1.365230
   1.365233    1.365230    1.365230
   1.365228    1.365230    1.365230
   1.365231    1.365230    1.365230
   1.365230    1.365230    1.365230
   1.365230    1.365230    1.365230
```

Kolme viimeistä iteraatiota suppenevat; viimeinen eli Newtonin menetelmä hyvinkin nopeasti. Juuren likiarvoksi saadaan $x^* \approx 1{,}365230$. Kaksi ensimmäistä iteraatiota eivät sen sijaan suppene. Tutkimalla menetelmiä

huomataan, että kiintopisteen x^* läheisyydessä $|g_i{}'(x^*)| > 1$, kun $i = 1, 2$, mutta $|g_i{}'(x^*)| < 1$, kun $i = 3, 4, 5$.

Johtopäätös: matemaattisesti ekvivalentit algoritmit voivat numeerisilta ominaisuuksiltaan poiketa suuresti toisistaan. Etusija on luonnollisesti pyrittävä antamaan tarkimmalle menetelmälle.

2.4 Välinpuolitusmenetelmä

Välinpuolitus on juurenhakumenetelmä, joka perustuu Bolzanon lauseeseen ([11], s. 444):

Lause 2 (Bolzanon lause). *Olkoon funktio g jatkuva suljetulla välillä $[a, c]$. Jos $g(a)$ ja $g(c)$ ovat erimerkkiset, niin funktiolla g on ainakin yksi nollakohta välillä (a, c). Pisteillä a ja c aloitettu välinpuolitusmenetelmä suppenee funktion g nollakohtaan välillä (a, c).*

Ratkaistava yhtälö muutetaan ensin muotoon $g(x) = 0$. Iteraatiota varten tarvitaan funktion g määrittelyvälin kaksi pistettä a ja c, $a < c$, joissa funktio $g(x)$ saa erimerkkiset arvot (jolloin $g(a)g(c) < 0$). Välillä (a, c) on Bolzanon lauseen nojalla ainakin yksi nollakohta. Laskemalla funktion arvo välin puolivälissä, pisteessä $b = (a+c)/2$, voidaan tutkittavaa väliä kaventaa puoleen valitsemalla uudeksi tarkasteluväliksi aina se puolikas, $[a, b]$ tai $[b, c]$, jonka päätepisteissä funktio saa erimerkkiset arvot. Välin kavetessa saadaan yhtälön juurelle aina tarkempia likiarvoja; n iteraation jälkeen välin pituus on $(c - a)/2^n$. Iterointia jatketaan, kunnes haluttu tarkkuus saavutetaan. Menetelmä on turvallinen, koska sillä päästään aina ratkaisuun: haettava nollakohta pysyy aina tarkasteluvälillä.

On huomattava, että jos funktiolla on tarkasteluvälillä useita nollakohtia, on väli jaettava tarpeeksi pieniin osaväleihin ja sovellettava menetelmää kuhunkin niistä erikseen.

Menetelmällä ei löydetä nollakohtia funktiolle, joka ei vaihda merkkiään nollakohdassa, esim. $y = x^2$.

Etsitään uudelleen yhtälön $x^3 + 4x^2 - 10 = 0$ juuri, nyt välinpuolitusmenetelmällä. Tarkasteluväliksi valitaan $[1, 2]$ jonka päätepisteissä funktio $g(x) = x^3 + 4x^2 - 10$ saa erimerkkiset arvot. Ohjelma on toteutettu niin, että se kysyy funktion käyttäjältä, joten se soveltuu yleisempäänkin käyttöön. Lisäksi tarkistetaan, että käyttäjän antamissa pisteissä funktio saa erimerkkiset arvot.

Ohjelma piirtää lisäksi funktion kuvaajan tarkasteluvälillä (ks. kuva 1). Siitä nähdään, onko välillä on muitakin nollakohtia. Nämä voidaan etsiä kaventamalla tarkasteluväliä.

Ohjelmaesimerkki 2.3

```
package examples;

// välinpuolitusmenetelmä

import com.braju.format.*;
import com.eteks.parser.*;
import java.awt.Color;
import uk.co.jscieng.Plottable;

public class IntHalv extends Base implements Plottable {
  CompiledFunction f;

  public void main() {
    double a, b, c, ga, gb, gc, dx, dy;

    showText(80, 35);
    println("Etsitään yhtälön g(x)=0 juuri\n"
      +"välinpuolitusmenetelmällä.\n");

    try {
      print("Syötä funktio g(x): ");
      f=new FunctionParser().compileFunction("g(x)="+readln());

      boolean loop=false;

      do {
        print("Syötä tarkasteluvälin alkupiste: ");
        a=Double.valueOf(readln()).doubleValue();

        print("Syötä tarkasteluvälin loppupiste: ");
        c=Double.valueOf(readln()).doubleValue();

        if (c<a) {
          double temp=a;
          a=c;
          c=temp;
        }

        ga=f(a);
        gc=f(c);
        dx=(c-a)/128.0;
        dy=(Math.abs(gc-ga)+4)/128.0;
```

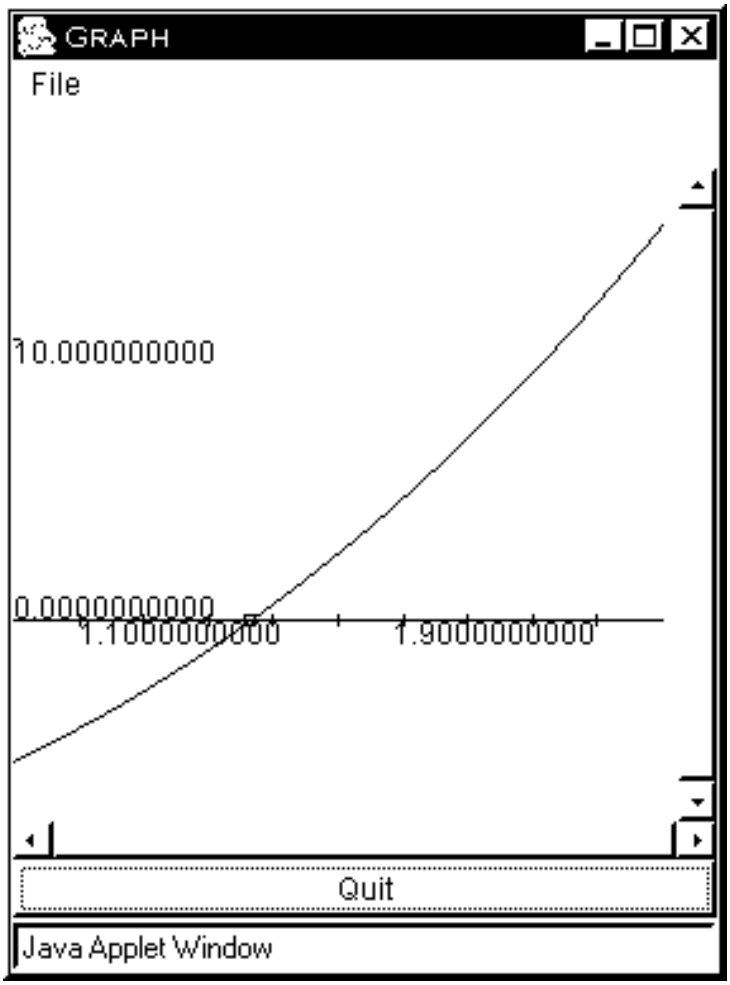
```
      if (ga*gc>=0.0) {
        println("Funktion arvot välin päätepisteissä "
          +"ovat samanmerkkiset.");
        loop=true;
      } else {
        loop=false;
      }
    } while (loop);

    if (ga<gc) {
      showGraph(a, c, ga-2, gc+2, PTS, PTS);
    } else {
      showGraph(a, c, gc-2, ga+2, PTS, PTS);
    }

    functionPlot(this, a, c, Color.red, 1);

    println(Format.sprintf("\n%10s %12s",
      new Parameters("x").add("g(x)")));

    do {
      b=0.5*(a+c);
      gb=f(b);
      ga=f(a);
      gc=f(c);

      println(Format.sprintf("%10.6f %12.4e",
        new Parameters(b).add(gb)));

      if (ga*gb>0) {   // ga ja gb samanmerkkiset
        a=b;
      } else {         // ga ja gb erimerkkiset
        c=b;
      }
    } while (Math.abs(gb)>1.0e-7);

    plotMark(b, gb, dx, dy);
  } catch (Exception exception) {
    print("Virhe:\n"+exception);
  }
}
```

```
  public double f(double x) {
    return f.computeFunction(new double[] {x});
  }
}
```

Ohjelman tulostus:

```
Etsitään yhtälön g(x)=0 juuri
välinpuolitusmenetelmällä.

Syötä funktio g(x): x^3+4*x^2-10
Syötä tarkasteluvälin alkupiste: 1
Syötä tarkasteluvälin loppupiste: 2

         x          g(x)
  1.500000   2.3750e+00
  1.250000  -1.7969e+00
  1.375000   1.6211e-01
  1.312500  -8.4839e-01
  1.343750  -3.5098e-01
  1.359375  -9.6409e-02
  1.367188   3.2356e-02
  1.363281  -3.2150e-02
  1.365234   7.2025e-05
  1.364258  -1.6047e-02
  1.364746  -7.9893e-03
  1.364990  -3.9591e-03
  1.365112  -1.9437e-03
  1.365173  -9.3585e-04
  1.365204  -4.3192e-04
  1.365219  -1.7995e-04
  1.365227  -5.3963e-05
  1.365231   9.0310e-06
  1.365229  -2.2466e-05
  1.365230  -6.7174e-06
  1.365230   1.1568e-06
  1.365230  -2.7803e-06
  1.365230  -8.1176e-07
  1.365230   1.7251e-07
  1.365230  -3.1962e-07
  1.365230  -7.3556e-08
```

20 iteraatiolla päästään tässä esimerkissä samaan tarkkuuteen kuin edellisellä kiintopistemenetelmällä.



Kuva 1: Funktion $g(x) = x^3 + 4x^2 - 10$ kuvaaja välillä $[1, 2]$.

2.5 Newtonin menetelmä

Newtonin menetelmä on suosittu yhtälönratkaisumenetelmä. Se käyttää hyväkseen funktion derivaattaa. Menetelmä voidaan johtaa seuraavasti: Olkoon x_0 yhtälön $f(x) = 0$ ratkaisun ensimmäinen likiarvo. Pisteeseen $(x_0, f(x_0))$ asetetun tangentin yhtälö on

$$y - f(x_0) = f'(x_0)(x - x_0)\,.$$

Asettamalla edelliseen yhtälöön $y = 0$ saadaan tangentin ja x-akselin leikkauspiste

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}\,.$$

Jos $f'(x_0) \neq 0$, voimme toivoa, että x_1 on lähempänä todellista juurta kuin x_0. Näin ollen saadaan iteraatio

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)},$$

jota jatketaan, kunnes $|x_{k+1} - x_k|$ on tarpeeksi pieni tai jokin ennalta rajoitettu määrä iteraatioita on jo suoritettu.

Menetelmä epäonnistuu, jos jossain vaiheessa $f'(x_k) = 0$. Tällöinhän kaavassa jaetaan nollalla ja uusi iteraatti karkaa äärettömyyteen. Juuren senhetkistä arvoa voidaan yrittää vielä parantaa muilla menetelmillä, kuten välinpuolituksella.

Seuraavan ohjelman avulla käyttäjä voi ratkaista haluamansa yhtälön Newtonin menetelmän avulla. Jos funktion derivaattaa ei syötetä, se lasketaan käyttäen *keskeisdifferenssiä*:

$$g'(x) \approx \frac{g(x+h) - g(x-h)}{2h},$$

missä askeleeksi h on valittu 10^{-7}. Optimaalisin askelpituus 64-bittisiä liukulukuja käytettäessä olisi $h \approx \sqrt{\epsilon_{\text{kone}}} \approx 10^{-8}$ ([13], s. 23).

Etsitään funktion

$$g(x) = x + \frac{e^{2\sin(2x)}}{\sqrt{1+x^2}} \tag{1}$$

nollakohta Newtonin menetelmällä (ks. kuva 2).

Ohjelmaesimerkki 2.4

```
package examples;

// Newtonin menetelmä

import com.braju.format.*;
import com.eteks.parser.*;
import uk.co.jscieng.Plottable;

public class Newton extends Base implements Plottable {
  private CompiledFunction f, df;

  public void main() {
    showText(80, 35);
    println("Etsitään yhtälön g(x)=0 juuri "
      +"Newtonin menetelmällä.\n");
```

```
try {
  print("Syötä funktio g(x): ");
  f=new FunctionParser().compileFunction("g(x)="+readln());

  print("Syötä derivaatta g'(x) tai anna tyhjä syöte: ");
  String input=readln();

  if (input.equals("")) {
    df=null;
  } else {
    df=new FunctionParser().compileFunction("dg(x)="+input);
  }
} catch (Exception exception) {
  print("Virhe:\n"+exception);
}

print("Syötä juuren alkuarvaus: ");
double x=Double.valueOf(readln()).doubleValue();

print("Syötä juuren hakualueen pituus: ");
double d=Double.valueOf(readln()).doubleValue();

double f0=f(x-0.5*d),
  f1=f(x+0.5*d),
  fx=f(x);

showGraph(x-0.5*d, x+0.5*d,
    Math.min(f0, f1)-1.0, Math.max(f0, f1)+1.0, PTS, PTS);
functionPlot(this, x-0.5*d, x+0.5*d, colors[8], 1);

println(Format.sprintf("\n%10s %10s\n%10.6f %10.4e",
  new Parameters("x").add("g(x)").add(x).add(fx)));

while (Math.abs(fx)>1.0e-9) {
  x-=fx/df(x);
  fx=f(x);

  println(Format.sprintf("%10.6f %10.4e",
    new Parameters(x).add(fx)));
}

plotMark(x, fx, d/128.0, (Math.abs(f1-f0)+2.0)/128.0);
```

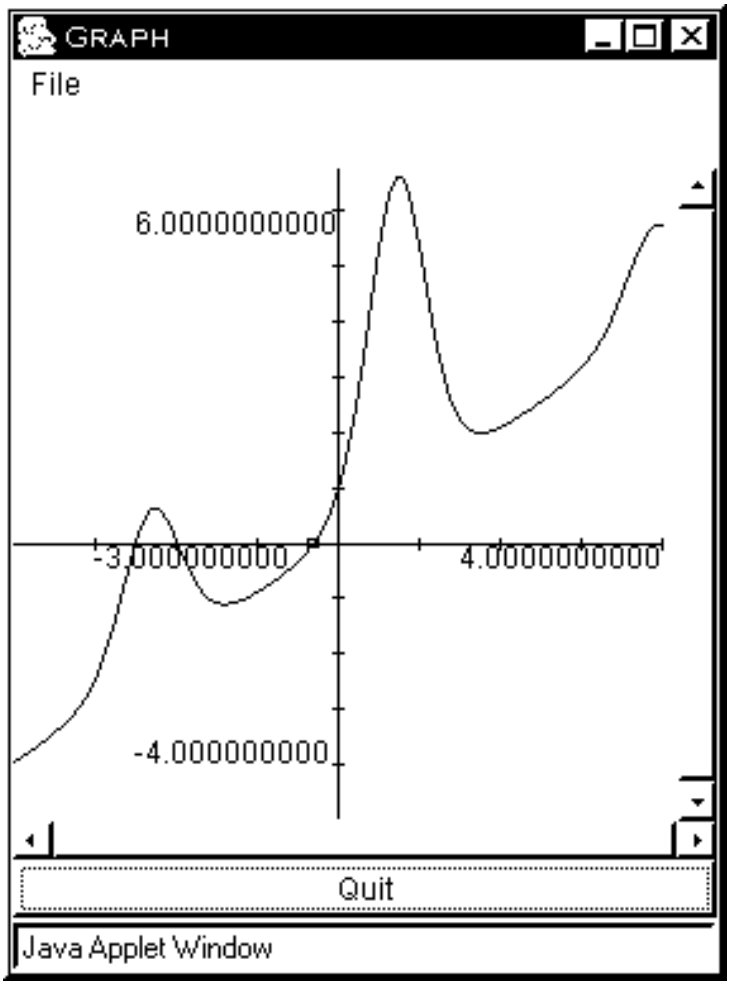
```
  }

  public double f(double x) {
    return f.computeFunction(new double[] {x});
  }

  private double df(double x) {
    if (df==null) {
      double eps=1.0e-7;

      return (f.computeFunction(new double[] {x+eps})
        -f.computeFunction(new double[] {x-eps}))/(2.0*eps);
    }

    return df.computeFunction(new double[] {x});
  }
}
```

Ohjelman tulostus:

```
Etsitään yhtälön g(x)=0 juuri
Newtonin menetelmällä.

Syötä funktio g(x): x+exp(2*sin(2*x))/sqrt(1+x^2)
Syötä derivaatta g'(x) tai anna tyhjä syöte:
Syötä juuren alkuarvaus: 0
Syötä juuren hakualueen pituus: 8

         x       g(x)
  0.000000 1.0000e+00
 -0.200000 2.5003e-01
 -0.291099 2.8594e-02
 -0.304375 4.6020e-04
 -0.304596 1.2237e-07
 -0.304596 8.6597e-15
```

Iteraatio suppeni melko nopeasti nollakohtaan $x^* \approx -0{,}304596$. Kuvaajan avulla voidaan etsiä muut välillä näkyvät nollakohdat, kun alkuarvaukset valitaan tarpeeksi läheltä niitä.

Newtonin menetelmäkään ei aina toimi tai se suppenee hitaasti, jos aloituspiste valitaan huonosti. Yhtälöllä $e^x = 3x^2$ on kaksi juurta välillä $(0, 4)$. Yritetään ratkaista yhtälö Newtonin menetelmällä lähtemällä pisteistä $x_j = 0{,}2j$, $j = 0, 1, 2, \ldots, 20$. Suoritetaan kuusi iteraatioaskelta ja katsotaan suppeneeko iteraatio jompaakumpaa positiivista juurta kohti.



Kuva 2: Funktion (1) kuvaaja välillä $[-4, 4]$.

Ohjelmaesimerkki 2.5

```
package examples;

// Newtonin iteraation suppenemistarkastelu

import com.braju.format.*;

public class NewtConv extends Base {
  public void main() {
    showText(80, 30);
    println("Tutkitaan, suppenevatko yhtälön\n"
```

```
      +"exp(x)=3x^2 pisteistä x_j = j*0.2, j = 0, 1, 2, ..., 20,\n"
      +"aloitetut Newton-iteraatiot kuudella askeleella\n"
      +"kohti jompaakumpaa positiivista juurta.\n");

    println(Format.sprintf("%10s %15s %13s",
      new Parameters("alkuarvaus").add(
        "6. iteraatti").add("virhe")));

    for (int i=0; i<21; i++) {
      double x=0.2*i,
        fx=f(x),
        dfx=df(x);

      print(Format.sprintf("%10.1f", new Parameters(x)));

      for (int j=1; j<7; j++) {
        x=x-fx/dfx;
        fx=f(x);
        dfx=df(x);
      }

      println(Format.sprintf("%16.5f %13.5e",
        new Parameters(x).add(Math.abs(f(x)))));
    }
  }

  private double f(double x) {
    return Math.exp(x)-3.0*x*x;
  }

  private double df(double x) {
    return Math.exp(x)-6.0*x;
  }
}
```

Ohjelman tulostus:

```
Tutkitaan, suppenevatko yhtälön
exp(x)=3x^2 pisteistä x_j = j*0.2, j = 0, 1, 2, ..., 20,
aloitetut Newton-iteraatiot kuudella askeleella
kohti jompaakumpaa positiivista juurta.

alkuarvaus    6. iteraatti         virhe
       0.0        -0.45896   1.11022e-16
```

```
0.2        -1.60747   7.55152e+00
0.4         0.91001   4.44089e-16
0.6         0.91001   4.44089e-16
0.8         0.91001   4.44089e-16
1.0         0.91001   4.44089e-16
1.2         0.91001   4.44089e-16
1.4         0.91001   4.44089e-16
1.6         0.91001   4.44089e-16
1.8         0.91001   4.44089e-16
2.0         0.91001   4.44089e-16
2.2         0.91001   4.44089e-16
2.4         0.91001   4.44089e-16
2.6        -0.45896   1.11022e-16
2.8        -0.62018   6.16021e-01
3.0         3.74792   2.91955e-01
3.2         3.73308   2.73772e-11
3.4         3.73308   7.10543e-15
3.6         3.73308   7.10543e-15
3.8         3.73308   7.10543e-15
4.0         3.73308   7.10543e-15
```

Huomataan, että aivan kaikista pisteistä ei iteraatio suppene (ainakaan nopeasti) kohti juurta. Kahdesta pisteestä päädyttiin tarkasteluvälin ulkopuolella olevaan negatiiviseen juureen. Tämä on osoituksena ja varoituksena siitä, että hyvätkään menetelmät eivät aina toimi, ainakaan jos alkuarvaus valitaan huonosti. (Tässä esimerkissä iteraatioita lisäämällä kaikista pisteistä lopulta päädyttiin johonkin juurista.)

2.6 Puolisuunnikassääntö

Puolisuunnikassääntö on keskipistesäännön jälkeen yksinkertaisin numeeriseen integrointiin kehitetty menetelmä. Se on yksi *suljetuista Newtonin ja Cotesin kaavoista* (suljettu tarkoittaa tässä yhteydessä sitä, että funktion arvoja integrointivälin päätepisteissä käytetään hyväksi, toisin kuin *avoimissa* kaavoissa). Newtonin ja Cotesin kaavat perustuvat funktion *interpolointiin* (kappale 4.2).

Puolisuunnikassäännössä funktiota interpoloidaan lineaarisesti ja integraalia approksimoidaan summalla, jossa käytetään funktion arvoja vain integrointivälin päätepisteissä ([22] , s. 100):

$$\int_a^b f(x)\,\mathrm{d}x \approx \frac{b-a}{2}(f(a)+f(b))\,.$$

Jos integrointiväli jaetaan n osaan ja sovelletaan menetelmää joka osavälillä, saadaan *yhdistetty puolisuunnikassääntö*:

$$\int_a^b f(x)\,\mathrm{d}x \approx h\left(\frac{1}{2}f_0 + f_1 + f_2 + \ldots + f_{n-1} + \frac{1}{2}f_n\right),$$

missä $h = \frac{b-a}{n}$ ja $f_i = f(a+ih),\ i = 0, 1, 2, \ldots, n$. Puolisuunnikassäännön *kertaluku* on 1, joka tarkoittaa, että vain ensimmäisen asteen polynomit integroituvat tarkasti. Kappaleessa 5.1 esitellään korkeamman kertaluvun menetelmä.

Integroidaan kaksi erilaista funktiota puolisuunnikassäännön avulla. Ohjelma pyytää integroitavan funktion ja integrointivälin käyttäjältä.

Ohjelmaesimerkki 2.6

```
package examples;

// puolisuunnikassääntö

import com.braju.format.*;
import com.eteks.parser.*;
import matutl.Maths;

public class PSS extends Base {
  private CompiledFunction f;

  public void main() {
    showText();
    println("Lasketaan funktion määrätty integraali\n"
      +"puolisuunnikassäännön avulla 1, ..., 10000 jakovälillä.\n");

    try {
      print("Syötä integroitava funktio: ");
      f=new FunctionParser().compileFunction("f(x)="+readln());
    } catch (Exception e) {
      print("Virhe:\n"+e);
    }

    print("Syötä integrointivälin alkupiste: ");
    double a=Double.valueOf(readln()).doubleValue();

    print("Syötä integrointivälin loppupiste: ");
    double b=Double.valueOf(readln()).doubleValue();
```

```
    println(Format.sprintf("\n%10s %14s",
      new Parameters("jakovälejä").add("integraali")));

    for (int N=1; N<=10000; N*=10) {
      double[] y=new double[N+1];
      double h=(b-a)/N;

      for (int j=0; j<=N; j++) {
        y[j]=f.computeFunction(new double[] {a+j*h});
      }

      double I=Maths.trapezoid(y, h);

      println(Format.sprintf("%10d %14.7f",
        new Parameters(N).add(I)));
    }
  }
}
```

Ohjelman tulostus syötteellä $y = 2x + 4$:

```
Lasketaan funktion määrätty integraali
puolisuunnikassäännön avulla 1, ..., 10000 jakovälillä.

Syötä integroitava funktio: 2*x+4
Syötä integrointivälin alkupiste: -1
Syötä integrointivälin loppupiste: 4

jakovälejä     integraali
         1     35.0000000
        10     35.0000000
       100     35.0000000
      1000     35.0000000
     10000     35.0000000
```

Lineaarisen funktion integraali on tarkka, kuten menetelmän kertaluku lupaa.

Ohjelman tulostus syötteellä $y = \sin(x)$:

```
Lasketaan funktion määrätty integraali
puolisuunnikassäännön avulla 1, ..., 10000 jakovälillä.

Syötä integroitava funktio: sin(x)
Syötä integrointivälin alkupiste: 0
```

```
Syötä integrointivälin loppupiste: 3.1415926535

jakovälejä     integraali
         1      0.0000000
        10      1.9835235
       100      1.9998355
      1000      1.9999984
     10000      2.0000000
```

Yksinkertaisuudestaan huolimatta yhdistetty puolisuunnikassääntö on tehokas menetelmä, kun integroidaan jaksollista funktiota jaksonpituuden mittaisen välin yli.

2.7 Monte Carlo -menetelmä

Monte Carlo -menetelmä on satunnaislukujen systemaattiseen käyttöön perustuva integrointimenetelmä. Siinä integroitavan funktion $f(u)$ arvoja lasketaan integrointivälin $[a, b]$ satunnaisissa pisteissä u_i yhteen, kerrotaan välin pituudella $b - a$ ja jaetaan pisteiden lukumäärällä N:

$$I_{MC} = \frac{b-a}{N} \sum_{i=1}^{N} f(u_i)\,.$$

Satunnaismuuttujan I_{MC} odotusarvoksi saadaan

$$\begin{aligned} E(I_{MC}) &= (b-a) \sum_{i=1}^{N} \frac{1}{N} E(f(u_i)) \\ &= (b-a) \sum_{i=1}^{N} \frac{1}{N} \int_a^b f(u) \frac{\mathrm{d}u}{b-a} \\ &= \int_a^b f(x)\,\mathrm{d}x = I, \end{aligned}$$

joka on juuri laskettavan integraalin tarkka arvo ([15] , s. 37).

Integroidaan taas kaksi funktiota, nyt Monte Carlo -menetelmällä. Ohjelma kysyy integroitavan funktion ja integrointivälin.

Ohjelmaesimerkki 2.7

```
package examples;

// Monte Carlo -integrointi
```

```
import com.braju.format.*;
import com.eteks.parser.*;
import Jama.Matrix;

public class MonteCarlo extends Base {
  private CompiledFunction f;
  public void main() {
    showText();
    println("Lasketaan funktion määrätty integraali\n"
      +"Monte Carlo -menetelmän avulla käyttämällä "
      +"10, ..., 100000 pistettä.\n");

    try {
      print("Syötä integroitava funktio: ");
      f=new FunctionParser().compileFunction("f(x)="+readln());
    } catch (Exception e) {
      println("Virhe:\n"+e);
    }

    print("Syötä integrointivälin alkupiste: ");
    double a=Double.valueOf(readln()).doubleValue();

    print("Syötä integrointivälin loppupiste: ");
    double b=Double.valueOf(readln()).doubleValue(),
      d=b-a;

    println(Format.sprintf("\n%10s %14s",
      new Parameters("pisteitä").add("integraali")));

    for (int N=10; N<=100000; N*=10) {
      double I=0.0;

      for (int j=0; j<=N; j++) {
        I+=f.computeFunction(new double[] {a+d*Math.random()});
      }

      I*=(d/N);

      println(Format.sprintf("%10d %14.7f",
        new Parameters(N).add(I)));
    }
  }
```

```
}
```

Ohjelman tulostus syötteellä $y = x$:

```
Lasketaan funktion määrätty integraali
Monte Carlo -menetelmän avulla käyttämällä 10, ..., 100000 pistettä.

Syötä integroitava funktio: x
Syötä integrointivälin alkupiste: 0
Syötä integrointivälin loppupiste: 4

  pisteitä     integraali
        10      6.5203216
       100      7.9465665
      1000      7.7170939
     10000      7.9623797
    100000      8.0038272
```

Ohjelman tulostus syötteellä $y = e^x$:

```
Lasketaan funktion määrätty integraali
Monte Carlo -menetelmän avulla käyttämällä 10, ..., 100000 pistettä.

Syötä integroitava funktio: exp(x)
Syötä integrointivälin alkupiste: 0
Syötä integrointivälin loppupiste: 1

  pisteitä     integraali
        10      1.8713101
       100      1.7136253
      1000      1.7212639
     10000      1.7102512
    100000      1.7166354
```

Ensimmäisen integraalin tarkka arvo on 8 ja toisen $e - 1 \approx 1{,}7182818$. Vaikka pisteitä olisi paljonkin, ei menetelmä silti anna kovin tarkkoja arvoja. Satunnaislukuihin perustuvaa menetelmää tulisikin soveltaa niin, että lasketaan esim. 10 arvoa ja niistä keskiarvo, jolloin satunnaisuudesta johtuvat virheet kumoavat toisiaan ja saadaan tarkempi likiarvo.

3 Numeerinen lineaarialgebra

Lineaarialgebra on tärkeä matematiikan osa-alue, joka käsittelee mm. vektoreita, matriiseja ja lineaarisia yhtälöryhmiä. Eri luonnontieteistä löytyy paljon lineaarialgebran sovelluskohteita. Tämä kappale sisältää useita erityyppisiä tehtäviä, joissa käsitellään vektoreita ja matriiseja.

3.1 Vektorilaskentaa

Kerrataan lyhyesti vektorilaskennan alkeita.

Vektori on yksiulotteinen lukutaulukko. Vektoria $\boldsymbol{x}$, jonka alkiot (tai komponentit) ovat $x_i,\ i = 1, \ldots, n$, merkitään joko

$$\boldsymbol{x} = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix},$$

jota sanotaan *pystyvektoriksi*, tai

$$\boldsymbol{x} = (x_1 \ \cdots \ x_n),$$

joka on *vaakavektori*. Asiayhteydestä yleensä selviää, kumpaa tarkoitetaan.

Vektorin pituus on sen alkioiden neliöiden summan neliöjuuri:

$$|\boldsymbol{x}| = \sqrt{x_1^2 + \cdots + x_n^2}.$$

Vektoreiden yhteen- ja vähennyslasku tapahtuu alkioittain:

$$\boldsymbol{x} \pm \boldsymbol{y} = \boldsymbol{z},$$

missä $z_i = x_i \pm y_i,\ i = 1, \ldots, n$.

Skalaarilla kerrottaessa jokainen vektorin alkioista kerrotaan erikseen:

$$c \cdot \boldsymbol{x} = (cx_1 \ \cdots \ cx_n).$$

Vektoreiden $\boldsymbol{x}$ ja $\boldsymbol{y}$ *skalaari-* tai *pistetulo* on

$$\boldsymbol{x} \cdot \boldsymbol{y} = x_1 y_1 + \cdots + x_n y_n \,.$$

Jos skalaaritulo on nolla, ovat vektorit kohtisuorassa toisiaan vastaan.

Kolmiulotteisen avaruuden vektoreille voidaan määritellä *risti-* tai *vektoritulo* ja *skalaarikolmitulo*.

Ristitulo määritellään

$$\boldsymbol{x} \times \boldsymbol{y} = \boldsymbol{z} = \begin{vmatrix} \boldsymbol{i} & \boldsymbol{j} & \boldsymbol{k} \\ x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \end{vmatrix},$$

missä $\boldsymbol{i}$, $\boldsymbol{j}$ ja $\boldsymbol{k}$ ovat avaruuden *kantavektorit*. Vektori $\boldsymbol{z}$ on kohtisuorassa vektoreita $\boldsymbol{x}$ ja $\boldsymbol{y}$ vastaan ja sen pituus $|\boldsymbol{z}|$ on näiden määrittämän suunnikkaan pinta-ala.

Skalaarikolmitulo määritellään

$$\boldsymbol{x} \cdot \boldsymbol{y} \times \boldsymbol{z} = \boldsymbol{x} \times \boldsymbol{y} \cdot \boldsymbol{z} = \begin{vmatrix} x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \\ z_1 & z_2 & z_3 \end{vmatrix}.$$

Tämän itseisarvo on vektoreiden virittämän särmiön tilavuus.

Seuraava esimerkki sisältää kootusti peruslaskutoimituksia vektoreilla.

Ohjelmaesimerkki 3.1

```
package examples;

// vektorilaskennan peruslaskutoimituksia

import com.braju.format.*;
import matutl.Vec;

public class VecCalc extends Base {
  public void main() {
    Vec A=new Vec(new double[] {1.0, 2.5, 4.0}),
      B=new Vec(new double[] {-3.0, 0.0, 1.5}),
      C=new Vec(new double[] {6.0, 5.0, -3.5});

    showText();
    println("Ohjelma esittelee vektorilaskennan\n"
      +"peruslaskutoimituksia.\n");
```

```
    tulosta("A", A, 1);
    tulosta("B", B, 1);
    tulosta("C", C, 1);

    println("Pituus: |A| = "
      +Format.sprintf("%7.5f", new Parameters(A.norm(2))));
    tulosta("\nSumma: A + B", A.plus(B), 2);
    tulosta("Erotus: A - B", A.minus(B), 2);

    tulosta("Skalaarilla kertominen: 3 * C", C.times(3.0), 2);
    println("Skalaaritulo: A . B = "+A.dotProd(B));
    tulosta("\nRistitulo: A x B", A.crossProd(B), 2);
    print("Skalaarikolmitulo: A . B x C = "+A.scalar3Prod(B, C));
  }
}
```

Ohjelman tulostus:

```
Ohjelma esittelee vektorilaskennan
peruslaskutoimituksia.

A =     1.0    2.5    4.0

B =    -3.0    0.0    1.5

C =     6.0    5.0   -3.5

Pituus: |A| = 4.82183

Summa: A + B =    -2.00    2.50    5.50

Erotus: A - B =     4.00    2.50    2.50

Skalaarilla kertominen: 3 * C =    18.00   15.00  -10.50

Skalaaritulo: A . B = 3.0

Ristitulo: A x B =     3.75  -13.50    7.50

Skalaarikolmitulo: A . B x C = -71.25
```

3.2 Matriisilaskentaa

Lukutaulukkoa A, jossa on m riviä ja n saraketta, kutsutaan *matriisiksi* ja merkitään

$$A = A_{m\times n} = (a_{ij}) = \begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{pmatrix}.$$

Merkintä $A \in \mathbb{R}^{m\times n}$ kertoo, että $m \times n$ matriisin A alkiot ovat reaalilukuja. Kompleksiarvoinen $m \times n$ matriisin B tapauksessa käytetään vastaavasti merkintää $B \in \mathbb{C}^{m\times n}$.

Matriisi, jossa on vain yksi rivi tai sarake, voidaan samaistaa vastaavat alkiot sisältävän vektorin kanssa.

Neliömatriisissa on yhtä paljon rivejä ja sarakkeita. Tällöin ei tarvitse käyttää kahta alaindeksiä, vaan voidaan käyttää merkintää A_n.

Neliömatriisi D_n on *diagonaalinen*, jos siinä on nollasta poikkeavia arvoja vain lävistäjällä:

$$D_n = \begin{pmatrix} d_{11} & & \mathbf{0} \\ & \ddots & \\ \mathbf{0} & & d_{nn} \end{pmatrix} = \operatorname{diag}(d_1, \ldots, d_n).$$

Identiteettimatriisi I on diagonaalimatriisi, jonka lävistäjäalkiot ovat ykkösiä.

Matriisi K_n on *kvasidiagonaalinen*, jos se voidaan kirjoittaa *lohkomuodossa*

$$K_n = \begin{pmatrix} K_{11} & & \mathbf{0} \\ & \ddots & \\ \mathbf{0} & & K_{kk} \end{pmatrix} = \operatorname{diag}(K_1, \ldots, K_k),$$

missä lohkot $K_1, \ldots, K_k$ ovat neliömatriiseja (voivat olla vaihtelevan kokoisia).

Matriisin A *transponoitu matriisi* A^T saadaan vaihtamalla matriisin rivit ja sarakkeet keskenään (sama pätee vektoreille). Matriisi A on *symmetrinen*, jos $A^T = A$.

Kompleksiarvoisen matriisin *adjungoitu matriisi* saadaan *kompleksikonjugoinnin* ja transponoinnin yhdistelmänä: $A^* = \bar{A}^T$. Tätä toimenpidettä kutsutaan myös *hermitoinniksi*, merkitään A^H. Jos matriisi A on reaaliarvoinen, niin $A^* = A^H = A^T$.

Jos reaaliarvoinen neliömatriisi A_n toteuttaa yhtälön

$$A^T A = A A^T = I_n,$$

niin sanotaan, että matriisi A on *ortogonaalinen*. Jos kompleksiarvoinen neliömatriisi B_n toteuttaa yhtälön

$$B^*B = BB^* = I_n\,,$$

sitä sanotaan *unitaariseksi* ([22], s. 193–194).

Myös matriisi $A \in \mathbb{R}^{m\times n}$, $m \neq n$, voi olla ortogonaalinen siinä mielessä, että $A^TA = I_n$.

Matriisi U on *yläkolmiomatriisi*, jos $u_{ij} = 0$, $i > j$, eli sen alkiot lävistäjän alapuolella ovat nollia. Vastaavasti määritellään *alakolmiomatriisi* $L : l_{ij} = 0$, $j > i$.

Matriisien yhteen- ja vähennyslasku sekä skalaarilla kertominen tapahtuvat alkioittain, kuten vektoreillakin.

3.2.1 Matriisien kertolaskua

Matriisien $A_{l\times m}$ ja $B_{m\times n}$ tulo on matriisi $C_{l\times n}$, joka määritellään seuraavasti:

$$C = \begin{pmatrix} \sum_{i=1}^{m} a_{1i}b_{i1} & \cdots & \sum_{i=1}^{m} a_{1i}b_{in} \\ \vdots & \ddots & \vdots \\ \sum_{i=1}^{m} a_{li}b_{i1} & \cdots & \sum_{i=1}^{m} a_{li}b_{in} \end{pmatrix}.$$

Matriisien ja vektoreiden kertolasku on myös mahdollinen, kunhan niiden dimensiot täsmäävät. Vektori ajatellaan tällöin matriisiksi, jonka toinen dimensio on yksi (edellä l tai n).

Seuraavassa ohjelmassa lasketaan kahden matriisin tulo. Matriisien dimensiot pyydetään aluksi, jotta kertolasku varmasti onnistuu. Toimintavarmuuden lisäämiseksi varmistetaan, että dimensiot ovat positiivisia kokonaislukuja. Todetaan myös, että reaaliarvoisia matriiseja koskeva yhtälö

$$(AB)^T = B^TA^T$$

toteutuu laskemalla $||(AB)^T - B^TA^T||_1$, jonka pitäisi olla nolla. $||\cdot||_1$ on matriisin *1-normi* eli sarakesummien maksimi:

$$||A||_1 = \max_j \sum_i |a_{ij}|\,.$$

Tämä on vain yksi matriisilaskennassa käytössä olevista matriisinormeista. Funktio $||\cdot|| : \mathbb{R}^{m\times n} \to \mathbb{R}$ on matriisinormi, jos se toteuttaa ehdot

1. $||A|| \geq 0$
2. $||A|| = 0 \iff A = \mathbf{0}$

3. $||\alpha A|| = |\alpha|\,||A||\,,\ \alpha \in \mathbb{R}$

4. $||A + B|| \leq ||A|| + ||B||$

kaikille matriiseille $A\,,\ B \in \mathbb{R}^{m \times n}$ ([22] , s. 189).

Ohjelmaesimerkki 3.2

```
package examples;

// matriisien kertolasku

import com.braju.format.*;
import Jama.Matrix;
import matutl.MatrixIO;

public class MatMul extends Base {
  public void main() {
    showText(80, 35);
    println("Ohjelma laskee matriisien A ja B tulon ja "
      +"tarkistaa, että\n    (AB)^T = B^T * A^T.\n"
      +"Matriisien dimensiot: A: i x j, B: j x k, i, j, k > 0.\n");

    int v=valinta(),
      i=dimensio("i", 1),
      j=dimensio("j", 1),
      k=dimensio("k", 1);
    Matrix A, B;

    if (v==1) {
      A=luoMatriisi("A", i, j);
      B=luoMatriisi("A", j, k);
    } else {
      A=Matrix.random(i, j).times(5);
      B=Matrix.random(j, k).times(5);
    }

    Matrix AB=A.times(B),
      ABT=AB.transpose(),
      BTAT=B.transpose().times(A.transpose());
    int decim=desimaalit();

    tulosta("A", A, 5, decim);
    tulosta("B", B, 5, decim);
```

```
    tulosta("AB", AB, 5, decim);
    tulosta("(AB)^T", ABT, 5, decim);
    tulosta("B^T * A^T", BTAT, 5, decim);
    print("||(AB)^T - B^T * A^T|| = "+Format.sprintf("%-8.3e",
      new Parameters(ABT.minus(BTAT).norm1())));
  }

  public int dimensio(String nimi, int min) {
    int n=0;

    do {
      print("Anna dimensio "+nimi+": ");

      try {
        n=Integer.parseInt(readln());
      } catch (Exception e) {}
    } while (n<min);

    return n;
  }
}
```

Ohjelman tulostus:

```
Ohjelma laskee matriisien A ja B tulon ja tarkistaa, että
    (AB)^T = B^T * A^T.
Matriisien dimensiot: A: i x j, B: j x k, i, j, k > 0.

Valitse 1, jos haluat syöttää luvut itse.
Valitse 2, jos haluat ohjelman generoivan satunnaiset luvut.
Valinta: 2
Anna dimensio i: 3
Anna dimensio j: 4
Anna dimensio k: 2
Matriisialkiot esitetään oletusarvoisesti kolmella desimaalilla.
Hyväksy oletus antamalla tyhjä syöte tai
anna uusi desimaalien määrä: 4
A =
    2.5663    2.0494    0.0997    3.0407
    4.2022    0.8966    3.6905    2.7982
    2.9726    3.0699    2.9790    2.8314

B =
    3.6166    4.0402
```

```
    1.6108    4.6471
    3.0113    0.1084
    0.7856    2.2609

AB =
   15.2717   26.7778
   29.9535   27.8707
   26.8908   33.0001

(AB)^T =
   15.2717   29.9535   26.8908
   26.7778   27.8707   33.0001

B^T * A^T =
   15.2717   29.9535   26.8908
   26.7778   27.8707   33.0001

||(AB)^T - B^T * A^T|| = 0.000e+00
```

3.2.2 Neliömatriisin käänteismatriisi ja determinantti

Neliömatriisin A_n *käänteismatriisi* A^{-1} on matriisi, jolle pätee

$$AA^{-1} = A^{-1}A = I_n\,.$$

Neliömatriisi on *säännöllinen*, jos sillä on käänteismatriisi, muuten se on *singulaarinen*. Ortogonaaliselle neliömatriisille $A^{-1} = A^T$.

Käänteismatriisin avulla voidaan esimerkiksi ratkaista yhtälöryhmä

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2 \\ \qquad\qquad \cdots & \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n &= b_n \end{cases}$$

muuntamalla se ensin matriisimuotoon

$$A\boldsymbol{x} = \boldsymbol{b}\,,$$

missä matriisi A sisältää kertoimet a_{ij}, vektori $\boldsymbol{x}$ tuntemattomat x_i ja vektori $\boldsymbol{b}$ kertoimet b_i. Kertomalla yhtälö vasemmalta matriisilla A^{-1} saadaan $\boldsymbol{x}$:

$$A^{-1}A\boldsymbol{x} = \boldsymbol{x} = A^{-1}\boldsymbol{b}\,.$$

Käytännön sovelluksissa käänteismatriisin laskeminen eksplisiittisesti on turhan työläs operaatio. Niinpä yhtälöryhmät ratkaistaan yleensä eri tavalla,

joko *Gaussin eliminoinnilla* (kappale 3.2.3) tai erilaisista *matriisihajotelmista*, joihin palataan myöhemmin. Erityisesti tapauksessa, jossa suuri osa kerroinmatriisin A alkioista on nollia (ns. harva matriisi), Gaussin eliminointi voi olla merkittävästi tehokkaampi kuin käänteismatriisin käyttöön perustuva menetelmä. Harvan matriisin käänteismatriisi on nimittäin yleensä tiheä ([13] , s. 37).

Neliömatriisin A_n *determinantti* $|A| = \det(A)$ on luku, joka voidaan määritellä esimerkiksi seuraavan rekursion avulla:

$$|A_1| = a_{11} \,,$$
$$|A_n| = \sum_{j=1}^{n} a_{ij} C_{ij} \,,$$

missä $C_{ij} = (-1)^{(i+j)} \det(A_{ij})$ on alkion a_{ij} *komplementti*. *Alimatriisi* A_{ij} saadaan, kun pyyhitään alkuperäisestä matriisista pois i:s vaakarivi ja j:s pystyrivi.

Determinantti voidaan *kehittää* minkä tahansa vaakarivin suhteen, ts. edellisessä kaavassa i voidaan valita vapaasti. Determinantti voidaan kehittää myös minkä tahansa pystyrivin suhteen vaihtamalla summausindeksiksi i .

Determinantin laskeminen rekursiivisesti vaatii $n!$ yhteenlaskua, jossa jokainen yhteenlaskettava on n:n luvun tulo ja niissäkin jonkin verran redundanssia. Isoilla matriiseilla tämä olisi hidasta ja epäkäytännöllistä. Determinantti voidaan laskea nopeammin matriisin *LU-hajotelmasta* (kappale 3.2.4).

Seuraavassa esimerkissä käyttäjältä pyydetään neliömatriisi, josta lasketaan determinantti ja käänteismatriisi. Käänteismatriisin oikeellisuus tarkistetaan laskemalla normi $||AA^{-1} - I||_1$. Jos matriisi on singulaarinen, annetaan virheilmoitus.

Ohjelmaesimerkki 3.3

```
package examples;

// käänteismatriisi ja determinantti

import com.braju.format.*;
import Jama.Matrix;

public class InvDet extends Base {
  public void main() {
    showText(80, 35);
    println("Ohjelma laskee neliömatriisin A "
      +"käänteismatriisin X ja\ndeterminantin |A|.\n");
```

```
    int v=valinta(),
      n=lukumaara("rivien", 1);
    Matrix A;

    if (v==1) {
      A=luoMatriisi("A", n, n);
    } else {
      A=Matrix.random(n, n).times(10);
    }

    int decim=desimaalit();

    tulosta("A", A, decim);

    try {
      Matrix X=A.inverse();
      double diff=A.times(X).minus(Matrix.identity(n, n)).norm1();

      tulosta("X", X, decim);
      println(Format.sprintf("||AX - I|| = %-8.3e",
        new Parameters(diff)));
    } catch (Exception e) {
      println("Virhe: "+e);
    } finally {
      print(Format.sprintf("|A| = %-8.3e",
        new Parameters(A.det())));
    }
  }
}
```

Ohjelman tulostus:

```
Ohjelma laskee neliömatriisin A käänteismatriisin X ja
determinantin |A|.

Valitse 1, jos haluat syöttää luvut itse.
Valitse 2, jos haluat ohjelman generoivan satunnaiset luvut.
Valinta: 2
Anna rivien lukumäärä: 3
Matriisialkiot esitetään oletusarvoisesti kolmella desimaalilla.
Hyväksy oletus antamalla tyhjä syöte tai
anna uusi desimaalien määrä: 4
A =
```

```
    9.8799    4.9830    1.9219
    2.2111    1.8119    6.1845
    8.6704    4.4951    1.6305

X =
    3.3880   -0.0701   -3.7274
   -6.8203    0.0756    7.7524
    0.7869    0.1646   -0.9387

||AX - I|| = 8.438e-15
|A| = -7.333e+00
```

3.2.3 Gaussin eliminointi

Gaussin eliminointi on tärkeä algoritmi, jota tarvitaan mm. lineaaristen yhtälöryhmien ratkaisemisessa. Sen avulla yhtälöryhmä $A\boldsymbol{x} = \boldsymbol{b}$ muutetaan ekvivalenttiin muotoon $U\boldsymbol{x} = \boldsymbol{g}$ missä matriisi U on yläkolmiomatriisi. Tästä muodosta tuntematon $\boldsymbol{x}$ on helppo ratkaista. Muunnos suoritetaan käyttämällä *matriisioperaatioita*

- rivin kertominen nollasta eroavalla luvulla
- kahden rivin vaihto
- rivin lisääminen toiseen riviin nollasta eroavalla luvulla kerrottuna.

Käydään menetelmä tarkemmin läpi seuraavan ohjelman avulla. Se on toteutettu ilman matriisi- ja vektoriluokkia, jotta se olisi mahdollisimman riippumaton. Lisäksi tavallisten taulukoiden käyttö yksinkertaistaa koodia ja nopeuttanee ohjelman suoritusta.

Yhtälöryhmästä $A\boldsymbol{x} = \boldsymbol{b}$, joka sisältää n yhtälöä ja tuntematonta, muodostetaan *augmentoitu matriisi* $[A|\boldsymbol{b}]$, joka saadaan lisäämällä vektori $\boldsymbol{b}$ matriisin A viimeiseksi sarakkeeksi. Tämä muutetaan yllä esitellyillä matriisioperaatioilla *porrasmuotoon* $[U|\boldsymbol{g}]$, eli muotoon

$$\left(\begin{array}{ccccc|c} 1 & u_{12} & u_{13} & \cdots & u_{1n} & g_1 \\ 0 & 1 & u_{23} & \cdots & u_{2n} & g_2 \\ 0 & 0 & 1 & \cdots & u_{3n} & g_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & g_n \end{array}\right).$$

Jos matriisi A on singulaarinen, menetelmä epäonnistuu eikä yhtälöryhmällä ole ratkaisua.

Vektori $\boldsymbol{x}$ eli yhtälöryhmän tuntemattomat saadaan nyt ratkaistua *takaisinsijoituksella*: Augmentoidun matriisin alimmalta riviltä saadaan suoraan $x_n = g_n$. Loput muuttujat ratkaistaan kaavalla

$$x_k = g_k - \sum_{j=k+1}^{n} u_{kj} x_j \,,\ k = n-1, n-2, \ldots, 1 \,.$$

Tulokset tarkistetaan laskemalla *residuaali* $|A\boldsymbol{x} - \boldsymbol{b}|$.

Ohjelmaesimerkki 3.4

```
package examples;

// yhtälöryhmän ratkaiseminen Gaussin eliminoinnilla

import com.braju.format.*;

public class GaussElim extends Base {
  private int n;
  private double[][] aug;
  private double[] x;

  public void main() {
    showText(80, 35);
    println("Ratkaistaan yhtälöryhmä Ax = b "
      +"(n yhtälöä, n tuntematonta)\nGaussin eliminoinnilla.\n");

    int v=valinta();
    n=lukumaara("yhtälöiden", 1);
    aug=new double[n][n+1];

    if (v==1) {
      for (int i=0; i<n; i++) {
        for (int j=0; j<n; j++) {
          print("Anna A("+(i+1)+","+(j+1)+"): ");
          aug[i][j]=Double.valueOf(readln()).doubleValue();
        }

        print("Anna b("+(i+1)+"): ");
        aug[i][n]=Double.valueOf(readln()).doubleValue();
      }
    } else {
      for (int i=0; i<n; i++) {
```

```
      for (int j=0; j<=n; j++) {
        aug[i][j]=1.0*(int)(10.0*Math.random()-5.0);
      }
    }
  }

  int decim=desimaalit();
  String format="%"+(5+decim)+"."+decim+"f";

  println("\nMuodostetaan augmentoitu matriisi [A|b]...");

  if (n<21) {
    printAug(format);
  } else {
    println("Matriisi on liian suuri tulostettavaksi (n>20).");
  }

  if (compute()) {
    if (n<21) {
      println("\n\nGaussin eliminoinnilla "
        +"saadaan yhtälöryhmä porrasmuotoon");
      printAug(format);
    }

    println("\n\nYhtälöryhmällä on ratkaisu [x]\n");

    for (int i=0; i<n; i++) {
      println(Format.sprintf(format, new Parameters(x[i])));
    }

    print(Format.sprintf("\nResiduaali |Ax-b| = %-8.3e",
      new Parameters(residual())));
  } else {
     print("\nYhtälöryhmällä ei ole ratkaisua.");
  }
}

private void printAug(String format) {
  for (int i=0; i<n; i++) {
    println();

    for (int j=0; j<=n; j++) {
      print(Format.sprintf(format, new Parameters(aug[i][j])));
```

```
    }
  }
}


private double residual() {
  double r=0.0;

  for (int i=0; i<n; i++) {
    double diff=aug[i][n];

    for (int j=0; j<n; j++) {
      diff-=aug[i][j]*x[j];
    }

    r+=diff*diff;
  }

  return Math.sqrt(r);
}

// original code by Stephen R. Schmitt written in Java Script
// http://home.att.net/~srschmitt/script_gauss_elimination3.html
// modified by Ari Heino

// compute solution to system of linear equations
public boolean compute() {
  x=new double[n];
  java.util.Arrays.fill(x, 0.0);

  // solve using Gaussian elimination with back substitution
  if (eliminate()) {
    substitute();
  } else {
    return false;
  }

  return true;
}

// convert matrix [A] to upper diagonal form
private boolean eliminate() {
  for (int i=0; i<n; i++) {
```

```
    int max_row=i;

    // find row with maximum in column i
    for (int j=i+1; j<n; j++) {
      if (Math.abs(aug[j][i])>Math.abs(aug[max_row][i])) {
        max_row=j;
      }
    }

    // swap max row with row i of [A|b]
    for (int j=i; j<=n; j++) {
      double tmp=aug[i][j];
      aug[i][j]=aug[max_row][j];
      aug[max_row][j]=tmp;
    }

    // pivot approx. zero => algorithm fails
    if (Math.abs(aug[i][i])<1.0e-8) {
      return false;
    }

    // 1's to diagonal
    for (int j=n; j>=i; j--) {
      aug[i][j]/=aug[i][i];
    }

    // eliminate lower diagonal elements of [A]
    for (int j=i+1; j<n; j++) {
      for (int k=n; k>=i; k--) {
        aug[j][k]-=(aug[i][k]*aug[j][i]);
      }
    }
  }

  return true;
}

// compute the values of vector x starting from the bottom
private void substitute() {
  for (int j=n-1; j>=0; j--) {
    double diff=aug[j][n];

    for (int k=j+1; k<n; k++) {
```

```
        diff-=aug[j][k]*x[k];
      }

      x[j]=diff;
    }
  }
}
```

Ohjelman tulostus:

```
Ratkaistaan yhtälöryhmä Ax = b (n yhtälöä, n tuntematonta)
Gaussin eliminoinnilla.

Valitse 1, jos haluat syöttää luvut itse.
Valitse 2, jos haluat ohjelman generoivan satunnaiset luvut.
Valinta: 2
Anna yhtälöiden lukumäärä: 3
Matriisialkiot esitetään oletusarvoisesti kolmella desimaalilla.
Hyväksy oletus antamalla tyhjä syöte tai
anna uusi desimaalien määrä: 4

Muodostetaan augmentoitu matriisi [A|b]

   0.0000  -2.0000   4.0000  -4.0000
  -2.0000   2.0000  -1.0000   2.0000
   0.0000  -3.0000   2.0000   0.0000

Gaussin eliminoinnilla saadaan yhtälöryhmä porrasmuotoon

   1.0000  -1.0000   0.5000  -1.0000
   0.0000   1.0000  -0.6667   0.0000
   0.0000   0.0000   1.0000  -1.5000

Yhtälöryhmällä on ratkaisu [x]

  -1.2500
  -1.0000
  -1.5000

Residuaali |Ax-b| = 1.110e-16
```

3.2.4 LU-hajotelma

Yksi tärkeimmistä matriisihajotelmista on *LU-hajotelma*, jonka olemassaoloa koskee seuraava lause ([22] , s. 234):

Lause 3 (LU-hajotelma). *Jokaista säännöllistä matriisia A kohti on olemassa* permutaatiomatriisi P *siten, että LU-hajotelma*

$$PA = LU$$

voidaan muodostaa. Permutaatiomatriisi P voidaan valita siten, että alakolmiomatriisin L alkiot toteuttavat ehdon $|l_{ij}| \leq 1$.

JAMA-kirjastossa LU-hajotelma voidaan laskea myös muille kuin säännöllisille neliömatriiseille. Matriisin $A_{m\times n}$, $m \geq n$, LU-hajotelma koostuu tällöin alakolmiomatriisista $L_{m\times n}$, yläkolmiomatriisista U_n sekä sellaisesta permutaatiomatriisista P_m, että $PA = LU$.

Permutaatiomatriisi P tarvitaan numeerisen stabiilisuuden parantamiseksi: Hajotelma lasketaan Gaussin eliminointia soveltaen ja osittaistuennan johdosta matriisin rivejä joudutaan yleensä vaihtamaan keskenään. Permutaatiomatriisilla kertomalla rivit palaavat alkuperäisille paikoilleen. Esimerkiksi permutaatiomatriisi

$$P = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

kertoo, että hajotelmassa on vaihdettu alkuperäisen matriisin kaksi ylintä riviä keskenään.

LU-hajotelma voidaan esittää myös muodossa $A = PLU$, mutta tästä päästään helposti edellä esitettyyn muotoon:

$$A = PLU \iff P^T A = LU\,,$$

koska kaikki permutaatiomatriisit ovat ortogonaalisia ([22] , s. 203).

Hajotelmat ovat siis permutaatiomatriisia lukuun ottamatta samat. Monilla matriiseilla A on olemassa myös puhtaasti muotoa $A = LU$ oleva hajotelma. Esimerkiksi matriisilla

$$A = \begin{pmatrix} 1 & 2 & 6 \\ 0 & 4 & 1 \\ 8 & 2 & 2 \end{pmatrix}$$

on hajotelmat

$$LU = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 8 & -3\frac{1}{2} & 1 \end{pmatrix} \begin{pmatrix} 1 & 2 & 6 \\ 0 & 4 & 1 \\ 0 & 0 & -42\frac{1}{2} \end{pmatrix}$$

ja

$$P^T LU = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ \frac{1}{8} & \frac{7}{16} & 1 \end{pmatrix} \begin{pmatrix} 8 & 2 & 2 \\ 0 & 4 & 1 \\ 0 & 0 & 5\frac{5}{16} \end{pmatrix},$$

mutta esimerkiksi matriisille

$$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

ei voida laskea LU-hajotelmaa ilman tuentaa ([12] , s. 58). LU-hajotelma ei ole yksikäsitteinen; permutoimalla matriisin rivejä eri tavoilla saadaan erilaisia hajotelmia.

Permutaatiomatriisia P tarvitaan, kuten edellä mainittiin, erityisesti parantamaan numeerista stabiilisuutta hajotelmaa koneella laskettaessa. Tuentaan on turvauduttava myös silloin, kun matriisin lävistäjällä on nollia tai niitä syntyy hajotelmaa laskettaessa. Stabiilisuutta kaivataan erityisesti silloin, kun matriisit ovat isoja tai lähes singulaarisia. LU-hajotelma voidaan laskea myös singulaarisille matriiseille.

Seuraavan ohjelma laskee LU-hajotelman matriisille $A_{m\times n}$, $m \geq n$. Lasketaan hajotelmat sekä neliömatriisille, että matriisille, jossa on rivejä enemmän kuin sarakkeita.

Ohjelmaesimerkki 3.5

```
package examples;

// LU-hajotelma

import com.braju.format.*;
import Jama.*;
import matutl.Maths;

public class LUtest extends Base {
  public void main() {
    showText(80, 35);
    println("Lasketaan m x n matriisin A LU-hajotelma (m >= n).\n");

    int v=valinta(),
      m=lukumaara("rivien", 1),
      n=lukumaara("sarakkeiden", 1, m);
    Matrix A;

    if (v==1) {
      A=luoMatriisi("A", m, n);
    } else {
      A=Matrix.random(m, n).times(10);
    }

    LUDecomposition LUD=new LUDecomposition(A);
```

```
    Matrix L=LUD.getL(),
      U=LUD.getU(),
      P=Matrix.identity(m, m).getMatrix(LUD.getPivot(), 0, m-1),
      PA=P.times(A),
      LU=L.times(U);

    int decim=desimaalit();

    tulosta("A", A, decim);
    tulosta("L", L, decim);
    tulosta("U", U, decim);
    tulosta("P", P, decim);
    print("||PA - LU|| = "+Format.sprintf("%-8.3e",
      new Parameters(PA.minus(LU).norm1())));
  }
}
```

Ohjelman tulostus (valitaan $m = n$):

```
Lasketaan m x n matriisin A LU-hajotelma (m >= n).

Valitse 1, jos haluat syöttää luvut itse.
Valitse 2, jos haluat ohjelman generoivan satunnaiset luvut.
Valinta: 2
Anna rivien lukumäärä: 4
Anna sarakkeiden lukumäärä: 4
Matriisialkiot esitetään oletusarvoisesti kolmella desimaalilla.
Hyväksy oletus antamalla tyhjä syöte tai
anna uusi desimaalien määrä:
A =
    4.786    1.425    5.269    7.616
    6.557    7.729    6.944    3.018
    6.548    8.164    1.233    3.860
    9.967    6.665    5.412    7.472

L =
    1.000    0.000    0.000    0.000
    0.657    1.000    0.000    0.000
    0.658    0.884    1.000    0.000
    0.480   -0.469    0.291    1.000

U =
    9.967    6.665    5.412    7.472
    0.000    3.785   -2.322   -1.049
```

```
    0.000    0.000    5.435   -0.970
    0.000    0.000    0.000    3.819

P =
    0.000    0.000    0.000    1.000
    0.000    0.000    1.000    0.000
    0.000    1.000    0.000    0.000
    1.000    0.000    0.000    0.000

||PA - LU|| = 2.220e-16
```

Ohjelman tulostus (valitaan $m > n$):

```
Lasketaan m x n matriisin A LU-hajotelma (m >= n).

Valitse 1, jos haluat syöttää luvut itse.
Valitse 2, jos haluat ohjelman generoivan satunnaiset luvut.
Valinta: 2
Anna rivien lukumäärä: 3
Anna sarakkeiden lukumäärä: 2
Matriisialkiot esitetään oletusarvoisesti kolmella desimaalilla.
Hyväksy oletus antamalla tyhjä syöte tai
anna uusi desimaalien määrä:
A =
    4.845    0.029
    4.311    6.931
    4.986    8.897

L =
    1.000    0.000
    0.972    1.000
    0.865    0.088

U =
    4.986    8.897
    0.000   -8.616

P =
    0.000    0.000    1.000
    1.000    0.000    0.000
    0.000    1.000    0.000

||PA - LU|| = 4.441e-16
```

LU-hajotelma on hyödyllinen, jos on ratkaistava useita yhtälöryhmiä, joissa vain oikean puolen vektori vaihtuu: LU-hajotelmahan koskee vain vasemman puolen kerroinmatriisia. Yhtälöryhmän $A\boldsymbol{x} = \boldsymbol{b}$ ratkaisu tapahtuu tällöin seuraavasti ([12] , s. 57):

- muodostetaan matriisin A LU-hajotelma $PA = LU$
- ratkaistaan $\boldsymbol{z}$ yhtälöstä $L\boldsymbol{z} = P\boldsymbol{b}$
- ratkaistaan $\boldsymbol{x}$ yhtälöstä $U\boldsymbol{x} = \boldsymbol{z}$.

Tämä tuottaa oikean ratkaisun, sillä

$$PA\boldsymbol{x} = LU\boldsymbol{x} = L(U\boldsymbol{x}) = L\boldsymbol{z} = Pb\,.$$

Koska L ja U ovat kolmiomatriiseja, on edellä mainitut yhtälöt helppo ratkaista (vrt. Gaussin eliminoinnin takaisinsijoitusvaihe).

Vielä turvallisemmin yhtälöryhmä voidaan ratkaista käyttämällä *täydellistä tuentaa*, jossa vaihdetaan kerroinmatriisissa A sekä rivejä että sarakkeita. Tästä syystä myös muuttujien järjestys vaihtuu. Täydellistä tuentaa käytettäessä LU-hajotelma on muotoa

$$PA\Pi = LU\,,$$

missä Π on uusi permutaatiomatriisi. Yhtälöryhmän ratkaisemisessa on tällöin seuraavat vaiheet ([12] , s. 66–67):

- muodostetaan matriisin A LU-hajotelma $PA\Pi = LU$
- ratkaistaan $\boldsymbol{y}$ yhtälöstä $L\boldsymbol{y} = P\boldsymbol{b}$
- ratkaistaan $\boldsymbol{z}$ yhtälöstä $U\boldsymbol{z} = \boldsymbol{y}$
- permutoimalla saadaan $\boldsymbol{x} = \Pi\boldsymbol{z}$.

Täydellinen tuenta vaatii kuitenkin niin paljon lukujen vertailuja (jolloin algoritmin suoritusaika pitenee), että yleensä tyydytään osittaistuentaan.

LU-hajotelmaa käyttäen neliömatriisin A determinantti voidaan laskea helposti: Tunnetusti neliömatriisien X_i tulolle pätee ([12] , s. 2)

$$\det\left(\prod_i X_i\right) = \prod_i \det(X_i)\,.$$

Soveltamalla tätä matriisin A LU-hajotelmaan saadaan

$$\begin{aligned} PA = LU &\implies \det(PA) = \det(LU) \\ &\iff \det(P)\det(A) = \det(L)\det(U) \\ &\iff \pm 1 \cdot \det(A) = 1 \cdot \det(U) \\ &\iff \det(A) = \pm\det(U) = \pm\mathrm{tr}(U)\,. \end{aligned}$$

Mielivaltaisen ala/yläkolmiomatriisin K determinantti on sen lävistäjäalkioiden tulo eli *jälki*, merkitään $\mathrm{tr}(K)$. Matriisin L lävistäjäalkiot ovat ykkösiä, joten $\det(L) = 1$. Permutaatiomatriisin determinantti on aina ± 1 riippuen siitä, onko hajotelmassa vaihdettu rivejä parillinen vai pariton määrä. Matriisin transponointi ei vaikuta determinantin arvoon.

3.2.5 Matriisin ominaisarvot ja -vektorit

Neliömatriisin A_n *ominaisarvo* λ on (kompleksi)luku, jolle pätee

$$A\boldsymbol{x} = \lambda\boldsymbol{x}$$

jollekin vektorille $\boldsymbol{x} \neq \boldsymbol{0}$. Tätä vektoria sanotaan *ominaisvektoriksi* ([13], s. 351).

Koska $\lambda\boldsymbol{x} = \lambda I\boldsymbol{x}$, missä I on identiteettimatriisi, niin

$$A\boldsymbol{x} = \lambda\boldsymbol{x} \iff (A - \lambda I)\boldsymbol{x} = \boldsymbol{0} \iff |A - \lambda I| = 0\,.$$

Polynomia $|A - \lambda I|$ sanotaan matriisin A *karakteristiseksi polynomiksi*. Matriisin A ominaisarvot ovat tämän astetta n olevan polynomin juuret ([22], s. 188). Ominaisarvoja on n kappaletta, joista osa voi olla yhtäsuuria tai kompleksisia. Ominaisvektorit $\boldsymbol{x}_j$ löydetään ratkaisemalla yhtälöryhmät

$$(A - \lambda_j I)\boldsymbol{x}_j = \boldsymbol{0}\,,\ j = 1, \ldots, n\,.$$

Oletetaan, että matriisilla A_n on n erisuurta ominaisarvoa λ_i ja vastaavat lineaarisesti riippumattomat ominaisvektorit $\boldsymbol{x}_i$. Kootaan ominaisvektorit matriisiksi $V = (\boldsymbol{x}_1\ \boldsymbol{x}_2 \cdots\ \boldsymbol{x}_n)$. Tällöin

$$\begin{aligned} AV &= A(\boldsymbol{x}_1\ \boldsymbol{x}_2 \cdots\ \boldsymbol{x}_n) \\ &= (A\boldsymbol{x}_1\ A\boldsymbol{x}_2 \cdots\ A\boldsymbol{x}_n) \\ &= (\lambda_1\boldsymbol{x}_1\ \lambda_2\boldsymbol{x}_2 \cdots\ \lambda_n\boldsymbol{x}_n) \\ &= (\boldsymbol{x}_1\ \boldsymbol{x}_2 \cdots\ \boldsymbol{x}_n) \cdot \mathrm{diag}(\lambda_1\ \lambda_2 \cdots\ \lambda_n) \\ &= VD\,. \end{aligned}$$

Tästä saadaan seuraava lause:

Lause 4 (Ominaisarvohajotelma). *Olkoon diagonaalimatriisin D lävistäjällä matriisin A ominaisarvot ja matriisin V sarakkeina vastaavat ominaisvektorit. Edellyttäen, että V on neliömatriisi, voidaan matriisi A kirjoittaa muodossa*

$$A = VDV^{-1}\,,$$

jota sanotaan matriisin A ominaisarvohajotelmaksi. *Jos matriisi A on symmetrinen, matriisi V on ortogonaalinen* [7].

JAMA-kirjaston `EigenvalueDecomposition`-luokka on edellisen lauseen mukainen. Lisäksi, jos matriisi A ei ole symmetrinen, D on kvasidiagonaalinen: reaaliset ominaisarvot ovat 1×1 lohkoina ja kompleksiset ominaisarvot $(\lambda_j \pm i\mu_j)$ 2×2 lohkoina muodossa

$$\begin{bmatrix} \lambda_j & \mu_j \\ -\mu_j & \lambda_j \end{bmatrix}.$$

Matriisin A ollessa symmetrinen (jolloin matriisi V on ortogonaalinen) hajotelma voidaan kirjoittaa muodossa

$$A = VDV^T.$$

Seuraavan ohjelman avulla voidaan etsiä neliömatriisin ominaisarvot ja -vektorit sekä näistä saatava ominaisarvohajotelma.

Ohjelmaesimerkki 3.6

```
package examples;

// ominaisarvohajotelma

import com.braju.format.*;
import Jama.*;
import matutl.Maths;

public class EVtest extends Base {
  public void main() {
    showText(80, 35);
    println("Lasketaan n x n matriisin A ominaisarvohajotelma.\n");

    int v=valinta(),
      n=lukumaara("rivien", 1);
    Matrix A;

    if (v==1) {
      A=luoMatriisi("A", n, n);
    } else {
      A=Matrix.random(n, n).times(10);
    }

    EigenvalueDecomposition EVD=new EigenvalueDecomposition(A);
    Matrix V=EVD.getV(),
      D=EVD.getD();
```

```
    int decim=desimaalit();

    tulosta("A", A, decim);
    tulosta("V", V, decim);
    tulosta("D", D, decim);

    if (Maths.isSymmetric(A)) {
      Matrix M=V.times(D).times(V.transpose());

      println("A on symmetrinen.");
      tulosta("VDV^T", M, decim);
      print("||A - VDV^T|| = "+Format.sprintf("%-8.3e",
        new Parameters(A.minus(M).norm1())));
    } else {
      Matrix M=V.times(D).times(V.inverse());

      tulosta("VDV^(-1)", M, decim);
      print("||A - VDV^(-1)|| = "+Format.sprintf("%-8.3e",
        new Parameters(A.minus(M).norm1())));
    }
  }
}
```

Ohjelman tulostus (symmetrinen matriisi):

```
Lasketaan n x n matriisin A ominaisarvohajotelma.

Valitse 1, jos haluat syöttää luvut itse.
Valitse 2, jos haluat ohjelman generoivan satunnaiset luvut.
Valinta: 1
Anna rivien lukumäärä: 2
Anna A(1,1): 1
Anna A(1,2): 2
Anna A(2,1): 2
Anna A(2,2): 1
Matriisialkiot esitetään oletusarvoisesti kolmella desimaalilla.
Hyväksy oletus antamalla tyhjä syöte tai
anna uusi desimaalien määrä: 5
A =
    1.00000    2.00000
    2.00000    1.00000

V =
    0.70711    0.70711
```

```
   -0.70711    0.70711

D =
   -1.00000    0.00000
    0.00000    3.00000

A on symmetrinen.
VDV^T =
    1.00000    2.00000
    2.00000    1.00000

||A - VDV^T|| = 4.441e-16
```

Nähdään, että matriisin $\begin{pmatrix} 1 & 2 \\ 2 & 1 \end{pmatrix}$ ominaisarvot ovat -1 ja 3. Vastaavat ominaisvektorit ovat $\frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ -1 \end{pmatrix}$ ja $\frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ 1 \end{pmatrix}$. Tarkkoja arvoja ei tietenkään saada, mutta tässä esimerkissä ne voidaan arvata ja tarkistaa käsin laskemalla. Kerroin $\frac{1}{\sqrt{2}}$ tarvitaan, jotta matriisi V olisi ortogonaalinen.

Ohjelman tulostus (epäsymmetrinen matriisi ja reaaliset ominaisarvot):

```
Lasketaan n x n matriisin A ominaisarvohajotelma.

Valitse 1, jos haluat syöttää luvut itse.
Valitse 2, jos haluat ohjelman generoivan satunnaiset luvut.
Valinta: 1
Anna rivien lukumäärä: 2
Anna A(1,1): 1
Anna A(1,2): 0
Anna A(2,1): -1
Anna A(2,2): 2
Matriisialkiot esitetään oletusarvoisesti kolmella desimaalilla.
Hyväksy oletus antamalla tyhjä syöte tai
anna uusi desimaalien määrä: 5
A =
    1.00000    0.00000
   -1.00000    2.00000

V =
   -0.70711    0.00000
   -0.70711   -1.41421

D =
    1.00000    0.00000
```

```
    0.00000    2.00000

VDV^(-1) =
    1.00000   -0.00000
   -1.00000    2.00000

||A - VDV^(-1)|| = 6.796e-16
```

Ohjelman tulostus (epäsymmetrinen matriisi ja kompleksiset ominaisarvot):

```
Lasketaan n x n matriisin A ominaisarvohajotelma.

Valitse 1, jos haluat syöttää luvut itse.
Valitse 2, jos haluat ohjelman generoivan satunnaiset luvut.
Valinta: 1
Anna rivien lukumäärä: 2
Anna A(1,1): 1
Anna A(1,2): -2
Anna A(2,1): 2
Anna A(2,2): 1
Matriisialkiot esitetään oletusarvoisesti kolmella desimaalilla.
Hyväksy oletus antamalla tyhjä syöte tai
anna uusi desimaalien määrä: 2
A =
    1.00   -2.00
    2.00    1.00

V =
   -1.00    0.00
    0.00    1.00

D =
    1.00    2.00
   -2.00    1.00

VDV^(-1) =
    1.00   -2.00
    2.00    1.00

||A - VDV^(-1)|| = 0.000e+00
```

Matriisin $\begin{pmatrix} 1 & 2 \\ -2 & 1 \end{pmatrix}$ ominaisarvot ovat siis $1 \pm 2i$. Vastaavat ominaisvek-

torit ovat $\begin{pmatrix} -1 \\ 0 \end{pmatrix}$ ja $\begin{pmatrix} 0 \\ 1 \end{pmatrix}$.

Monista numeerisista menetelmistä on kehitetty versioita, jotka toimivat joissakin erityistapauksissa. Myös matriisin ominaisarvot ja -vektorit voidaan tietyissä tapauksissa laskea helpommin kuin edellä esitettiin. Seuraavassa yksi esimerkki.

Jos $n \times n$ matriisi T on *tridiagonaalinen* ($t_{ij} = 0$, jos $|i-j| > 1$) ja lisäksi muotoa

$$T = \begin{pmatrix} d & u & 0 & \cdots \\ v & d & u & \cdots \\ 0 & v & d & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{pmatrix},$$

niin sen ominaisarvot ovat

$$\lambda_s = d + 2u * r \cos\left(\frac{s * \pi}{n+1}\right), \quad r = \sqrt{v/u}\,,\ s = 1, \ldots, n$$

ja vastaavat ominaisvektorit

$$v(s)_j = r^j \sin\left(\frac{j * s * \pi}{n+1}\right), \quad j = 1, \ldots, n\,.$$

Seuraavassa ohjelmassa käyttäjältä pyydetään tarvittavat parametrit ja lasketaan ominaisarvot ja vektorit yo. kaavoista. Todetaan, että kaikilla ominaisarvoilla ehto

$$||T * v_s - \lambda_s * v_s||_1 < 10^{-5}$$

toteutuu hyvin (päästään paljon tarkempiinkin tuloksiin).

Ohjelmaesimerkki 3.7

```
package examples;

// tridiagonaalimatriisin ominaisarvot ja -vektorit

import com.braju.format.*;
import Jama.Matrix;
import matutl.*;

public class TriDiag extends Base {
  public void main() {
    showText(80, 35);
    println("Luodaan n x n tridiagonaalimatriisi A ja tutkitaan "
      +"ehdon\n    e_s = ||A*v_s - l_s*v_s|| < 1.0E-5 , 1 <= s <= n"
```

```
      +"\ntoteutumista. l_s on s. ominaisarvo ja "
      +"v_s vastaava ominaisvektori.\n");

    int n=lukumaara("rivien", 3);

    print("Anna lävistäjäalkio d: ");
    double d=Double.valueOf(readln()).doubleValue();

    print("Anna lävistäjän yläpuolinen alkio u: ");
    double u=Double.valueOf(readln()).doubleValue();

    print("Anna lävistäjän alapuolinen alkio v: ");
    double v=Double.valueOf(readln()).doubleValue();

    double r=Math.sqrt(v/u);
    Matrix A=Maths.triDiagonal(n, v, d, u);
    int decim=desimaalit();

    tulosta("A", A, decim);

    for (int s=1; s<=n; s++) {
      double eigenval=d+2.0*u*r*Math.cos(s*Math.PI/(n+1.0));
      Vec eigenvec=new Vec(n);

      for (int j=1; j<=n; j++) {
        eigenvec.set(j-1,
          Maths.power(r, j)*Math.sin(j*s*Math.PI/(n+1.0)));
      }

      print(Format.sprintf("\nl_%1d = %5.3f\nv_%1d =",
        new Parameters(s).add(eigenval).add(s)));
      tulosta(eigenvec, decim);

      Vec left=eigenvec.times(A.transpose()),
        right=eigenvec.times(eigenval);

      println(Format.sprintf("e_%1d = %7.3e",
        new Parameters(s).add(left.minus(right).norm(2))));
    }
  }
}
```

Ohjelman tulostus:

```
Luodaan n x n tridiagonaalimatriisi A ja tutkitaan ehdon
    e_s = ||A*v_s - l_s*v_s|| < 1.0E-5 , 1 <= s <= n
toteutumista. l_s on s. ominaisarvo ja v_s vastaava ominaisvektori.

Anna rivien lukumäärä: 4
Anna lävistäjäalkio d: 2
Anna lävistäjän yläpuolinen alkio u: .5
Anna lävistäjän alapuolinen alkio v: 1.5
Matriisialkiot esitetään oletusarvoisesti kolmella desimaalilla.
Hyväksy oletus antamalla tyhjä syöte tai
anna uusi desimaalien määrä: 5
A =
    2.00000    0.50000    0.00000    0.00000
    1.50000    2.00000    0.50000    0.00000
    0.00000    1.50000    2.00000    0.50000
    0.00000    0.00000    1.50000    2.00000


l_1 = 3.401
v_1 =    1.01807    2.85317    4.94183    5.29007

e_1 = 3.580e-15

l_2 = 2.535
v_2 =    1.64728    1.76336   -3.05422   -8.55951

e_2 = 0.000e+00

l_3 = 1.465
v_3 =    1.64728   -1.76336   -3.05422    8.55951

e_3 = 3.689e-15

l_4 = 0.599
v_4 =    1.01807   -2.85317    4.94183   -5.29007

e_4 = 0.000e+00
```

3.2.6 Singulaariarvohajotelma ja yleistetty käänteismatriisi

Tarkastellaan matriisin *singulaariarvohajotelmaa*, joka liittyy myös matriisin ominaisarvoihin. Sen määrittelee seuraava lause ([22] , s. 204):

Lause 5 (Singulaariarvohajotelma). *Jos matriisi* $A \in \mathbb{R}^{m\times n}$, *niin on olemassa ortogonaalimatriisit* $U \in \mathbb{R}^{m\times m}$ *ja* $V \in \mathbb{R}^{n\times n}$ *siten, että*

$$U^T A V = \Sigma = \mathrm{diag}(\sigma_1, \sigma_2, \ldots, \sigma_k) \in \mathbb{R}^{m\times n}, \quad k = \min\{m, n\}.$$

Diagonaalimatriisin Σ *lävistäjäalkiot, joita kutsutaan matriisin* A singulaariarvoiksi, *toteuttavat ehdon*

$$\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_r \geq \sigma_{r+1} = \ldots = \sigma_k = 0,$$

missä $r = \mathrm{r}(A)$ *on matriisin* A aste *eli sen lineaarisesti riippumattomien vaaka/pystyrivien lukumäärä.*

Matriisin A singulaariarvohajotelma on lauseen nojalla

$$A = U\Sigma V^T,$$

koska matriisit U ja V ovat ortogonaalisia. Singulaariarvot σ_i ovat matriisin $A^T A$ ominaisarvojen λ_i, $i = 1, \ldots, n$ (jotka kaikki ovat ≥ 0), neliöjuuret suurimmasta pienimpään ([22], s. 205).

JAMA-kirjaston `SingularValueDecomposition`-luokka poikkeaa lauseen määritelmästä matriisien dimensioiden osalta: matriisi $U \in \mathbb{R}^{m\times n}$, $V \in \mathbb{R}^{n\times n}$ ja $\Sigma \in \mathbb{R}^{n\times n}$, $m \geq n$. Tapauksessa $m > n$ matriisi U on ortogonaalinen siinä mielessä, että $U^T U = I_n$.

Kompleksiarvoisella matriisilla B on singulaariarvohajotelma

$$B = U\Sigma V^*.$$

Matriisit U ja V ovat tällöin unitaarisia ja singulaariarvot saadaan matriisin $A^* A$ ominaisarvoista, jotka myös ovat reaalisia ja ≥ 0 ([17], s. 93).

Singulaariarvohajotelman avulla voidaan laskea matriisin A *pseudoinverssi* eli *yleistetty käänteismatriisi*

$$A^- = V\Sigma^- U^*,$$

missä $\Sigma^- = \mathrm{diag}(\sigma_1^{-1}, \sigma_2^{-1}, \ldots, \sigma_r^{-1}, 0, \ldots, 0)$ ([12], s. 139).

Seuraava ohjelma laskee annetun matriisin A singulaariarvohajotelman ja pseudoinverssin A^-. Todetaan myös, että $U^T U = V^T V = I_n$. Pseudoinverssin laskemisessa käytetään singulaariarvojen "editointia": singulaariarvot, jotka ovat pienempiä kuin $\sigma_1 \cdot \mathrm{eps}$ (eps on pieni luku, tyypillisesti 10^{-6}), korvataan nollilla. Tämä lisää numeerista stabiilisuutta, kun matriisi on lähes singulaarinen ([18], s. 63–65).

Ohjelmaesimerkki 3.8

```
package examples;

// singulaariarvohajotelma

import com.braju.format.*;
import Jama.*;
import matutl.Maths;

public class SVtest extends Base {
  public void main() {
    showText(80, 35);
    println("Lasketaan m x n matriisin A "
      +"singulaariarvohajotelma (m >= n).\n");

    int v=valinta(),
      m=lukumaara("rivien", 1),
      n=lukumaara("sarakkeiden", 1, m);
    Matrix A;

    if (v==1) {
      A=luoMatriisi("A", m, n);
    } else {
      A=Matrix.random(m, n).times(10);
    }

    SingularValueDecomposition SVD
      =new SingularValueDecomposition(A);
    Matrix U=SVD.getU(),
      S=SVD.getS(),
      V=SVD.getV(),
      USV=U.times(S).times(V.transpose());
    int decim=desimaalit();

    tulosta("A", A, decim);
    tulosta("U", U, decim);
    tulosta("U^TU", U.transpose().times(U), decim);
    tulosta("S", S, decim);
    tulosta("V", V, decim);
    tulosta("V^TV", V.transpose().times(V), decim);
    tulosta("USV*", USV, decim);
    println("||A - USV*|| = "+Format.sprintf("%-8.3e",
      new Parameters(A.minus(USV).norm1())));
```

```
    tulosta("A:n pseudoinverssi A-",
      Maths.pseudoinverse(A, true), decim);
  }
}
```

Ohjelman tulostus:

```
Lasketaan m x n matriisin A singulaariarvohajotelma (m >= n).

Valitse 1, jos haluat syöttää luvut itse.
Valitse 2, jos haluat ohjelman generoivan satunnaiset luvut.
Valinta: 1
Anna rivien lukumäärä: 3
Anna sarakkeiden lukumäärä: 2
Anna A(1,1): 1
Anna A(1,2): 1
Anna A(2,1): 2
Anna A(2,2): 2
Anna A(3,1): 2
Anna A(3,2): 2
Matriisialkiot esitetään oletusarvoisesti kolmella desimaalilla.
Hyväksy oletus antamalla tyhjä syöte tai
anna uusi desimaalien määrä: 5
A =
    1.00000    1.00000
    2.00000    2.00000
    2.00000    2.00000

U =
   -0.33333   -0.66667
   -0.66667    0.66667
   -0.66667   -0.33333

U^TU =
    1.00000   -0.00000
   -0.00000    1.00000

S =
    4.24264    0.00000
    0.00000    0.00000

V =
   -0.70711   -0.70711
   -0.70711    0.70711
```

```
V^TV =
    1.00000    0.00000
    0.00000    1.00000

USV* =
    1.00000    1.00000
    2.00000    2.00000
    2.00000    2.00000

||A - USV*|| = 2.220e-16
A:n pseudoinverssi A- =
    0.05556    0.11111    0.11111
    0.05556    0.11111    0.11111
```

Pseudoinverssiä käyttämällä voidaan esimerkiksi etsiä likimääräisratkaisu *ylideterminoidulle* (yhtälöitä enemmän kuin muuttujia) yhtälöryhmälle $A\boldsymbol{x} = \boldsymbol{b}$, missä kerroinmatriisi $A \in \mathbb{R}^{m\times n}$, $m \geq n$. Voidaan osoittaa, että $\boldsymbol{x} = A^{-}\boldsymbol{b}$ on tällöin yhtälöryhmän ratkaisu *pienimmän neliösumman* mielessä ([19], s. 114), eli se minimoi *Euklidisen normin*

$$\sqrt{\sum_{i=1}^{m}\left(\sum_{j=1}^{n} a_{ij}x_j - b_i\right)^2} = |A\boldsymbol{x} - \boldsymbol{b}|\,.$$

Seuraava ohjelma ratkaisee yhtälöryhmän pienimmän neliösumman mielessä. Kannattaa huomata, että jos $A \in \mathbb{R}^{n\times n}$ ja yhtälöryhmällä on tarkka ratkaisu, niin se on sama kuin yo. menettelyn antama ratkaisu.

Ohjelmaesimerkki 3.9

```
package examples;

// yhtälöryhmän ratkaiseminen singulaariarvohajotelman avulla

import com.braju.format.*;
import Jama.Matrix;
import matutl.Maths;

public class SVsolve extends Base {
  public void main() {
    showText(80, 35);
    println("Ratkaistaan yhtälöryhmä Ax = b,\n"
      +"jossa yhtälöitä on vähintään yhtä paljon kuin muuttujia.\n"
```

```
      +"Käytetään hyväksi A:n singulaariarvohajotelmaa.\n");

    int v=valinta(),
      m=lukumaara("yhtälöiden", 1),
      n=lukumaara("muuttujien", 1, m);
    Matrix A, b;

    if (v==1) {
      A=luoMatriisi("A", m, n);
      b=luoMatriisi("b", m, 1);
    } else {
      A=Matrix.random(m, n).times(10);
      b=Matrix.random(m, 1).times(10);
    }

    Matrix x=Maths.SVDsolve(A, b, true);
    int decim=desimaalit();

    tulosta("A", A, decim);
    tulosta("b", b, decim);
    tulosta("x", x, decim);
    print("|Ax - b| = "+Format.sprintf("%-8.3g",
      new Parameters(A.times(x).minus(b).normF())));
  }
}
```

Ohjelman tulostus (singulaarinen kerroinmatriisi):

```
Ratkaistaan yhtälöryhmä Ax = b,
jossa yhtälöitä on vähintään yhtä paljon kuin muuttujia.
Käytetään hyväksi A:n singulaariarvohajotelmaa.

Valitse 1, jos haluat syöttää luvut itse.
Valitse 2, jos haluat ohjelman generoivan satunnaiset luvut.
Valinta: 1
Anna yhtälöiden lukumäärä: 2
Anna muuttujien lukumäärä: 2
Anna A(1,1): 1
Anna A(1,2): 1
Anna A(2,1): 1
Anna A(2,2): 1
Anna b(1,1): 2
Anna b(2,1): 4
Matriisialkiot esitetään oletusarvoisesti kolmella desimaalilla.
```

```
Hyväksy oletus antamalla tyhjä syöte tai
anna uusi desimaalien määrä:
A =
    1.000    1.000
    1.000    1.000

b =
    2.000
    4.000

x =
    1.500
    1.500

|Ax - b| = 1.41
```

Yhtälöryhmän

$$\begin{cases} x_1 + x_2 = 2 \\ x_1 + x_2 = 4 \end{cases}$$

likimääräisratkaisuksi saatiin intuitiivisestikin selvä ratkaisu $x_1 = x_2 = 3/2$.

Ohjelman tulostus (ylideterminoitu yhtälöryhmä):

```
Ratkaistaan yhtälöryhmä Ax = b,
jossa yhtälöitä on vähintään yhtä paljon kuin muuttujia.
Käytetään hyväksi A:n singulaariarvohajotelmaa.

Valitse 1, jos haluat syöttää luvut itse.
Valitse 2, jos haluat ohjelman generoivan satunnaiset luvut.
Valinta: 2
Anna yhtälöiden lukumäärä: 3
Anna muuttujien lukumäärä: 2
Matriisialkiot esitetään oletusarvoisesti kolmella desimaalilla.
Hyväksy oletus antamalla tyhjä syöte tai
anna uusi desimaalien määrä: 5
A =
    2.33630    4.57693
    3.08518    3.81666
    1.96314    2.31848

b =
    7.50131
    2.07295
    7.23264
```

```
x =
   -0.89659
    2.03565

|Ax - b| = 5.19
```

3.2.7 QR-hajotelma

Käsitellään vielä yhtä matriisihajotelmaa ([22] , s. 241):

Lause 6 (QR-hajotelma). *Jokainen reaaliarvoinen matriisi* $A_{m\times n}$, $m \geq n$ *voidaan esittää matriisitulona*

$$A = Q \begin{pmatrix} R \\ 0 \end{pmatrix},$$

missä matriisi Q_m *on ortogonaalinen,* R_n *yläkolmiomatriisi ja* $0 \in \mathbb{R}^{(m-n)\times n}$.

Tämä matriisin A *QR-hajotelma* voidaan esittää myös muodossa

$$A = \widetilde{Q}\widetilde{R},$$

missä $\widetilde{Q}_{m\times n}$ sisältää n ensimmäistä saraketta matriisista Q. Se on ortogonaalinen siinä mielessä, että $\widetilde{Q}^T\widetilde{Q} = I_n$ ([19] , s. 82). JAMA-kirjaston `QRDecomposition`-luokka on toteutettu tämän jälkimmäisen määritelmän mukaisesti.

QR-hajotelman tärkein käyttötarkoitus on ylideterminoitujen lineaaristen yhtälöryhmien ratkaiseminen. Alkuperäisen yhtälöryhmän ratkaiseminen muuttuu tällöin helpomman yläkolmioyhtälöryhmän ratkaisemiseksi:

$$A\boldsymbol{x} = QR\boldsymbol{x} = \boldsymbol{b} \Longleftrightarrow R\boldsymbol{x} = Q^T\boldsymbol{b}.$$

Seuraavassa esimerkissä lasketaan matriisin QR-hajotelma.

Ohjelmaesimerkki 3.10

```
package examples;

// QR-hajotelma

import com.braju.format.*;
import Jama.*;
import matutl.MatrixIO;
```

```
public class QRtest extends Base {
  public void main() {
    showText(80, 35);
    println("Lasketaan m x n matriisin A "
      +"QR-hajotelma (m >= n).\n");

    int v=valinta(),
      m=lukumaara("rivien", 1),
      n=lukumaara("sarakkeiden", 1, m);
    Matrix A;

    if (v==1) {
      A=luoMatriisi("A", m, n);
    } else {
      A=Matrix.random(m, n).times(10);
    }

    QRDecomposition QRD=new QRDecomposition(A);
    Matrix Q=QRD.getQ(),
      R=QRD.getR(),
      QR=Q.times(R);
    int decim=desimaalit();

    tulosta("A", A, decim);
    tulosta("Q", Q, decim);
    tulosta("R", R, decim);
    tulosta("QR", QR, decim);
    tulosta("Q^T*Q", Q.transpose().times(Q), decim);
    print("||A - QR|| = "+Format.sprintf("%-8.3e",
      new Parameters(A.minus(QR).norm1())));
  }
}
```

Ohjelman tulostus:

```
Lasketaan m x n matriisin A QR-hajotelma (m >= n).

Valitse 1, jos haluat syöttää luvut itse.
Valitse 2, jos haluat ohjelman generoivan satunnaiset luvut.
Valinta: 2
Anna rivien lukumäärä: 4
Anna sarakkeiden lukumäärä: 2
Matriisialkiot esitetään oletusarvoisesti kolmella desimaalilla.
```

```
Hyväksy oletus antamalla tyhjä syöte tai
anna uusi desimaalien määrä:
A =
    1.299    8.326
    4.812    6.067
    4.715    2.765
    6.083    2.614

Q =
   -0.142    0.907
   -0.525    0.265
   -0.514   -0.138
   -0.663   -0.297

R =
   -9.169   -7.519
    0.000    8.004

QR =
    1.299    8.326
    4.812    6.067
    4.715    2.765
    6.083    2.614

Q^T*Q =
    1.000   -0.000
   -0.000    1.000

||A - QR|| = 5.773e-15
```

JAMA-kirjastossa on vielä luokka `CholeskyDecomposition`, jolla voidaan laskea symmetrisen *positiividefiniitin* ([22] , s. 194) matriisin A *Choleskyn hajotelma*

$$A = LL^T ,$$

missä L on alakolmiomatriisi ([22] , s. 235).

3.2.8 Newtonin menetelmä epälineaarisille yhtälöryhmille

Kappaleessa (2.5) esiteltiin Newtonin menetelmä yhden muuttujan funktioille. Menetelmä voidaan yleistää myös useammalle muuttujalle ([13] , s. 173) matriisien avulla.

Olkoon ratkaistavana yhtälö $\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{0}$, missä $\boldsymbol{f} : \mathbb{R}^n \to \mathbb{R}^n$ on jatkuvasti derivoituva funktio. Pisteessä $\boldsymbol{x} + \boldsymbol{s}$ funktiota voidaan arvioida *Taylorin*

sarjalla

$$\boldsymbol{f}(\boldsymbol{x}+\boldsymbol{s}) \approx \boldsymbol{f}(\boldsymbol{x}) + J(\boldsymbol{x})\boldsymbol{s}\,,$$

missä J on *Jacobin matriisi* $J_{ij} = \partial f_i/\partial x_j$. Asetetaan $\boldsymbol{f}(\boldsymbol{x}+\boldsymbol{s}) = \boldsymbol{0}$, jolloin ratkaistavaksi jää yhtälöryhmä

$$J(\boldsymbol{x})\boldsymbol{s} = -\boldsymbol{f}(\boldsymbol{x})\,.$$

Tämä on lineaarinen yhtälöryhmä, josta askel $\boldsymbol{s}$ voidaan ratkaista. Saadaan iteraatio (vrt. yhden muuttujan tapaus)

$$\begin{aligned} \boldsymbol{s}_k &= -J^{-1}(\boldsymbol{x}_k)\boldsymbol{f}(\boldsymbol{x}_k) \\ \boldsymbol{x}_{k+1} &= \boldsymbol{x}_k + \boldsymbol{s}_k \end{aligned}$$

Iteraatiota jatketaan, kunnes $|\boldsymbol{s}_k|$ on tarpeeksi pieni. Ylemmässä yhtälössä ei tietenkään eksplisiittisesti käytetä Jacobin matriisin käänteismatriisia, vaan sen LU-hajotelmaa.

Ohjelmaesimerkki 3.11

```
package examples;

// Newtonin menetelmä epälineaarisille yhtälöryhmille

import com.braju.format.*;
import com.eteks.parser.*;
import Jama.Matrix;
import matutl.*;

public class NewtNonlin extends Base {
  private CompiledFunction[] func;
  private int n;

  public void main() {
    showText();
    println("Ratkaistaan epälineaarinen yhtälöryhmä f(x) = 0,\n"
      +"missä x = (x0, x1, x2, ..., xn), Newtonin menetelmällä.\n");

    try {
      n=lukumaara("yhtälöiden (ja muuttujien)", 1);
      func=new CompiledFunction[n];

      for (int i=0; i<n; i++) {
        String name=funcName(i);
```

```
      print("Syötä funktio "+name+": ");
      func[i]=new FunctionParser().compileFunction(
         name+"="+readln());
    }

    Matrix x=new Matrix(n, 1, 1.0);
    double eps=1.0e-8,
      norm;

    do {
      Matrix J=jacobian(x),
        val=f(x),
        step=J.solve(val);
      x.minusEquals(step);
      norm=step.normF();
    } while (norm>eps);

    for (int i=0; i<n; i++) {
      println(Format.sprintf("x"+i+" = %-12.6g",
        new Parameters(x.get(i, 0))));
    }

    print(Format.sprintf(
      "|f(x)| = %-9.3e", new Parameters(norm)));
  } catch (Exception exception) {
    print("Virhe:\n"+exception);
  }
}

private Matrix f(Matrix m) {
  Matrix val=new Matrix(n, 1);
  double[] x=m.getColumnPackedCopy();

  for (int i=0; i<n; i++) {
    val.set(i, 0, func[i].computeFunction(x));
  }

  return val;
}

// Laskee funktion Jacobin matriisin keskeisdifferenssin avulla
private Matrix jacobian(Matrix x) {
```

```
    Matrix jacobi=new Matrix(n, n);
    double h=1.0e-6,
      h2=1.0/(2.0*h);

    for (int i=0; i<n; i++) {
      Matrix e=new Matrix(n, 1, 0.0),
        df;

      e.set(i, 0, h);    // j:s yksikkövektori * h
      df=(f(x.plus(e)).minus(f(x.minus(e)))).timesEquals(h2);
      jacobi.setMatrix(0, n-1, i, i, df);
    }

    return jacobi;
  }

  private String funcName(int num) {
    String name="f"+num+"(x0";

    for (int i=1; i<n; i++) {
      name+=(",x"+i);
    }

    return name+")";
  }
}
```

Ohjelman tulostus:

```
Ratkaistaan epälineaarinen yhtälöryhmä f(x) = 0,
missä x = (x0, x1, x2, ..., xn), Newtonin menetelmällä.

Anna yhtälöiden (ja muuttujien) lukumäärä: 2
Syötä funktio f0(x0,x1): exp(x0)-x0-x1-1
Syötä funktio f1(x0,x1): x0-x1*x1+1
x0 = 1.36001
x1 = 1.53623
|f(x)| = 1.334e-11
```

4 Numeerinen approksimointi ja interpolointi

Approksimoinnilla tarkoitetaan jonkin tunnetun ilmiön likimääräistä kuvaamista. Käytettävä data voi sellaisenaan olla liian hankalaa suoraan käytettäväksi, mutta pieniä likimääräistyksiä sallimalla voidaan luoda malli, jota on helppo hallita ja joka silti kuvaa riittävän tarkasti käsiteltävän suureen käyttäytymistä. Mallissa esiintyvien parametrien lukumäärä on tyypillisesti paljon pienempi kuin datapisteiden määrä.

Jatkuvassa approksimoinnissa etsitään tunnetulle, hankalalle funktiolle $f(x)$ jokin helpommin käsiteltävä funktio $\tilde{f}(x)$, joka kuitenkin approksimoi alkuperäistä funktiota riittävän tarkasti.

Diskreetissä approksimoinnissa tunnetaan funktiosta $f(x)$ vain äärellinen määrä arvoja f_i pisteissä x_i. Tähän pistejoukkoon $\{(x_i, f_i)\}$ pyritään sovittamaan pisteiden jakaumaa approksimoiva funktio $\tilde{f}(x)$. Jos vaaditaan, että funktion kuvaaja kulkee pisteiden kautta, on kyseessä *interpolointitehtävä*.

4.1 Pienimmän neliösumman approksimointi

Data-aineiston approksimointi voidaan tietysti tehdä monella tapaa. Yksinkertaisin tapa on sovittaa pistejoukkoon suora. Parhaiten aineistoa kuvaava suora löydetään *pienimmän neliösumman approksimoinnilla*: jos on koottu mittaussarja, joka koostuu pisteistä (x_i, y_i), $i = 1, \ldots, n$, niin suoralle voidaan johtaa [9] yhtälö

$$y = a + bx\,.$$

Kertoimien a ja b määrittämiseksi lasketaan keskiarvot

$$\bar{x} = \frac{1}{n}\sum x_i$$
$$\bar{y} = \frac{1}{n}\sum y_i$$

ja neliösummat

$$ss_{xx} = \sum (x_i - \bar{x})^2$$
$$ss_{yy} = \sum (y_i - \bar{y})^2$$
$$ss_{xy} = \sum (x_i - \bar{x})(y_i - \bar{y})\,.$$

Näiden perusteella voidaan laskea *regressiokerroin*

$$b = \frac{ss_{xy}}{ss_{xx}}\,,$$

vakiotermi

$$a = \bar{y} - b\bar{x}$$

ja *korrelaatiokerroin*

$$r = \frac{ss_{xy}}{\sqrt{ss_{xx}ss_{yy}}}\,.$$

Mitä lähempänä korrelaatiokertoimen r itseisarvo on ykköstä, sitä paremmin havainnot osuvat suoralle.

Seuraavan esimerkin avulla voidaan annettuun dataan sovittaa pienimmän neliösumman suora. Suoran yhtälön lisäksi lasketaan korrelaatiokerroin ja piirretään havainnot ja suoran kuvaaja.

Ohjelmaesimerkki 4.1

```
package examples;

// pienimmän neliösumman suora

import com.braju.format.*;
import uk.co.jscieng.Plottable;

public class PNS extends Base implements Plottable {
  private double[] x, y, params;
  private int xMin=0, xMax=0, yMin=0, yMax=0;

  public void main() {
    showText(80, 35);
    println("Generoidaan pisteet (x_i, y_i) ja sovitetaan siihen\n"
      +"pienimmän neliösumman suora.\n");

    int v=valinta(),
      n=lukumaara("pisteiden", 2);
```

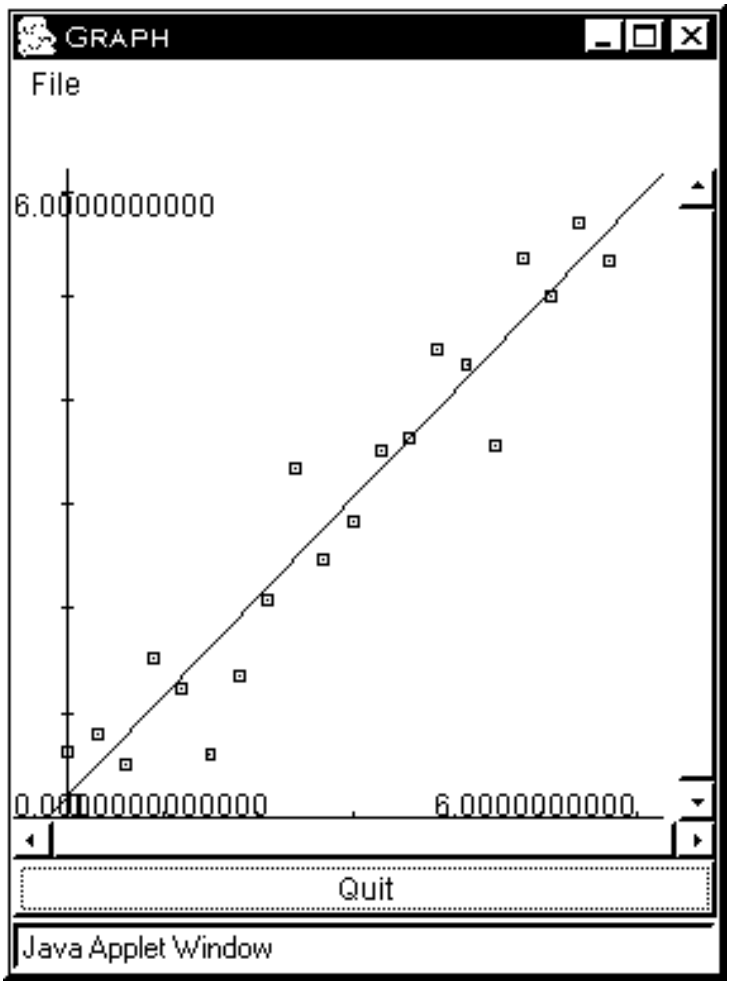
```
  if (v==1) {
    x=luoVektori("x", n).getColumnPackedCopy();
    y=luoVektori("y", n).getColumnPackedCopy();

    findXminMax();
  } else {
    x=new double[n];
    y=new double[n];

    for (int i=0; i<n; i++) {
      x[i]=0.3*i;
      y[i]=x[i]-1.0+2.0*Math.random();
    }

    xMin=0;
    xMax=n-1;
  }

  params=matutl.Maths.leastSquaresFitting(x, y);

  findYminMax();

  double dx=Math.max(0.01, x[xMax]-x[xMin]),
    dy=Math.max(0.01, y[yMax]-y[yMin]);

  showGraph(x[xMin]-0.1*dx, x[xMax]+0.1*dx,
            y[yMin]-0.1*dy, y[yMax]+0.1*dy, PTS, PTS);
  plotData();
  functionPlot(this, java.awt.Color.red, 4);

  println(Format.sprintf(
    "\ny = a + bx,\na = %-10.6f\nb = %-10.6f",
    new Parameters(params[0]).add(params[1])));
  print(Format.sprintf(
    "\nKorrelaatiokerroin r = %-10.6f",
    new Parameters(params[2])));
}

public double f(double x) {
  return params[0]+params[1]*x;
}
```

```
  private void findXminMax() {
    for (int i=1, n=x.length; i<n; i++) {
      if (x[i]<x[xMin]) {
        xMin=i;
      } else if (x[i]>x[xMax]) {
        xMax=i;
      }
    }
  }

  private void findYminMax() {
    for (int i=1, n=y.length; i<n; i++) {
      if (y[i]<y[yMin]) {
        yMin=i;
      } else if (y[i]>y[yMax]) {
        yMax=i;
      }
    }
  }

  private void plotData() {
    double dx=(x[xMax]-x[xMin]+1)/128.0,
      dy=(y[yMax]-y[yMin]+1)/128.0;

    for (int i=0, n=x.length; i<n; i++) {
      double xx=x[i],
        yy=y[i];

      plotMark(xx, yy, dx, dy);
    }
  }
}
```

Ohjelman tulostus:

```
Generoidaan pisteet (x_i, y_i) ja sovitetaan siihen
pienimmän neliösumman suora.

Valitse 1, jos haluat syöttää luvut itse.
Valitse 2, jos haluat ohjelman generoivan satunnaiset luvut.
Valinta: 2
Anna pisteiden lukumäärä: 20

y = a + bx,
```

```
a = 0.203825
b = 0.951811

Korrelaatiokerroin r = 0.960200
```



Kuva 3: Pienimmän neliösumman suoran sovitus annettuun dataan.

4.2 Polynomi-interpolaatio

Kun halutaan, että data-aineistoon sovitettava käyrä kulkee täsmälleen kaikkien pisteiden kautta, puhutaan interpoloinnista. Käsitellään tässä interpolaatiota polynomien avulla.

Kahden pisteen läpi kulkee yksikäsitteinen suora, kolmen pisteen kautta yksikäsitteinen paraabeli jne. Yleisesti, $n+1$ pisteen (x_i, y_i), $i = 0, \ldots, n$ kautta kulkee yksikäsitteinen n asteen polynomi. Yksi sen esitysmuodoista on *Lagrangen interpolaatiopolynomi*

$$P_n(x) = \sum_{i=0}^{n} \frac{(x-x_0)\cdots(x-x_{i-1})(x-x_{i+1})\cdots(x-x_n)}{(x_i-x_0)\cdots(x_i-x_{i-1})(x_i-x_{i+1})\cdots(x_i-x_n)}\, y_i\,. \qquad (2)$$

Tässä muodossaan se on hankala käyttää eikä sitä yleensä kannata ratkaista eksplisiittisesti, jos halutaan laskea vain muutamia funktion arvoja.

Interpolaatiopolynomi voidaan laskea käyttämällä *Nevillen algoritmia*, jonka C-kielinen implementointi löytyy kirjasta [18] , s. 108. Algoritmi vaatii syötteekseen pisteet, joiden kautta funktio kulkee, sekä pisteen, jossa funktion arvo halutaan laskea. Tämä on mielekästä tietenkin vain äärimmäisten datapisteiden välissä. Funktion arvon lisäksi algoritmi antaa myös virhearvion.

Luokan `Maths` metodi `polint` on em. algoritmin suora käännös Javakielelle. Seuraavassa ohjelmassa käyttöesimerkki.

Ohjelmaesimerkki 4.2

```
package examples;

// polynomi-interpolaatio

import com.braju.format.*;
import matutl.Maths;

public class Polint extends Base {
  public void main() {
    showText(80, 35);
    println("Etsitään annettujen pisteiden (x_i, f_i) kautta "
      +"kulkeva\ninterpolaatiopolynomi ja approksimoidaan "
      +"funktion arvoja\nmielivaltaisissa pisteissä.\n");

    int v=valinta(),
      n=lukumaara("pisteiden", 2);
    double[] x, y;

    if (v==1) {
      x=luoVektori("x", n).getColumnPackedCopy();
      y=luoVektori("y", n).getColumnPackedCopy();
    } else {
```

```
      x=new double[n];
      y=new double[n];

      for (int i=0; i<n; i++) {
        x[i]=i-n/2;
        y[i]=x[i]*x[i]-3.0*Math.random();
      }
    }

    println("Pisteistö:");

    for (int i=0; i<n; i++)
      println(Format.sprintf("%8.3f %10.3f",
        new Parameters(x[i]).add(y[i])));

    boolean loop=true;

    do {
      print("Anna piste, jossa funktiota approksimoidaan: ");

      try {
        double arg=Double.valueOf(readln()).doubleValue();
        double[] ans=Maths.polint(x, y, n, arg);

        println(Format.sprintf(
          "Funktion arvo = %-9.4g\nvirhearvio = %-7.4g",
          new Parameters(ans[0]).add(ans[1])));
      } catch (Exception e) {
        print("Ohjelman suoritus on loppunut.");
        loop=false;
      }
    } while (loop);
  }
}
```

Ohjelman tulostus:

```
Etsitään annettujen pisteiden (x_i, f_i) kautta kulkeva
interpolaatiopolynomi ja approksimoidaan funktion arvoja
mielivaltaisissa pisteissä.

Valitse 1, jos haluat syöttää luvut itse.
Valitse 2, jos haluat ohjelman generoivan satunnaiset luvut.
Valinta: 2
```

```
Anna pisteiden lukumäärä: 4
Pisteistö:
  -2.000      2.493
  -1.000     -0.806
   0.000     -1.832
   1.000      0.124
Anna piste, jossa funktiota approksimoidaan: -1.5
Funktion arvo = 0.6042
virhearvio = 0.04436
Anna piste, jossa funktiota approksimoidaan: 0.1
Funktion arvo = -1.783
virhearvio = 0.02732
Anna piste, jossa funktiota approksimoidaan: 1
Funktion arvo = 0.1236
virhearvio = 0.000
Anna piste, jossa funktiota approksimoidaan:
Ohjelman suoritus on loppunut.
```

Jos interpolaatiopolynomi halutaan ratkaista eksplisiittisesti, se voidaan tehdä seuraavasti:

Oletetaan, että data-aineisto sisältää pisteet (x_i, y_i), $i = 0, \ldots, n-1$. Haettava interpolaatiopolynomi on siis muotoa $y = a_{n-1}x^{n-1} + \cdots + a_1 x + a_0$ ja se saa jokaisessa pisteessä x_i arvon y_i. Saadaan yhtälöryhmä

$$\begin{cases} a_{n-1}x_0^{n-1} + \cdots + a_1 x_0 + a_0 = y_0 \\ \qquad\qquad\qquad\qquad\qquad\quad \vdots \\ a_{n-1}x_{n-1}^{n-1} + \cdots + a_1 x_{n-1} + a_0 = y_{n-1} \end{cases},$$

joka voidaan kirjoittaa matriisimuodossa

$$\begin{pmatrix} x_0^{n-1} & \cdots & x_0 & 1 \\ \vdots & \ddots & \vdots & \vdots \\ x_{n-1}^{n-1} & \cdots & x_{n-1} & 1 \end{pmatrix} \begin{pmatrix} a_{n-1} \\ \vdots \\ a_1 \\ a_0 \end{pmatrix} = \begin{pmatrix} y_0 \\ \vdots \\ y_{n-1} \end{pmatrix}.$$

Tuntemattomat polynomin kertoimet voidaan ratkaista esim. LU-hajotelmaa käyttäen.

Seuraavassa esimerkissä Lagrangen polynomin kertoimet lasketaan luokan `LagrangeInterpolation` avulla [2]. Siinä muodostetaan matriisiyhtälö yllä kuvatulla tavalla. Yhtälö ratkaistaan JAMA-kirjaston `Matrix`-luokan `solve`-metodilla, eli LU-hajotelmaa käyttäen.

Ohjelmaesimerkki 4.3

```
package examples;

// Lagrangen interpolaatiopolynomi

import com.braju.format.*;
import no.geosoft.cc.math.calculus.LagrangeInterpolation;
import uk.co.jscieng.Plottable;

public class Lagrange extends Base implements Plottable {
  double[] x, y, factors;
  double xInset=0.5, yInset=5.0;
  int xMin=0, xMax=0, yMin=0, yMax=0;

  public void main() {
    showText(80, 35);
    println("Generoidaan pisteiden (x_i, y_i) kautta kulkeva\n"
      +"Lagrangen interpolaatiopolynomi\n"
      +"y = a_(n-1) * x^(n-1) + ... + a_1 * x + a_0.\n");

    int v=valinta(),
      n=lukumaara("pisteiden", 2);

    if (v==1) {
      x=luoVektori("x", n).getColumnPackedCopy();
      y=luoVektori("y", n).getColumnPackedCopy();

      findXminMax();
    } else {
      x=new double[n];
      y=new double[n];

      for (int i=0; i<n; i++) {
        x[i]=1.0+i;
        y[i]=(double)(int)(10.0*Math.random());
      }

      xMin=0;
      xMax=n-1;
    }

    factors=LagrangeInterpolation.findPolynomialFactors (x, y);
```

```
  findYminMax();
  showGraph(x[xMin]-xInset, x[xMax]+xInset,
            y[yMin]-yInset, y[yMax]+yInset, PTS, PTS);
  plotData();
  functionPlot(this, java.awt.Color.red, 1);

  for (int i=0; i<n; i++) {
    println(Format.sprintf("a_%1d = %-10.5f",
      new Parameters(n-i-1).add(factors[i])));
  }
}

public double f(double x) {
  double val=factors[0];

  for (int i=1, n=factors.length; i<n; i++) {
    val=x*val+factors[i];
  }

  return val;
}

void findXminMax() {
  for (int i=1, n=x.length; i<n; i++) {
    if (x[i]<x[xMin]) {
      xMin=i;
    } else if (x[i]>x[xMax]) {
      xMax=i;
    }
  }
}

void findYminMax() {
  for (int i=1, n=y.length; i<n; i++) {
    if (y[i]<y[yMin]) {
      yMin=i;
    } else if (y[i]>y[yMax]) {
      yMax=i;
    }
  }
}

void plotData() {
```

```
    double dx=(x[xMax]-x[xMin]+2*xInset)/128.0,
      dy=(y[yMax]-y[yMin]+2*yInset)/128.0;

    for (int i=0, n=x.length; i<n; i++) {
      double xx=x[i],
        yy=y[i];

      plotMark(xx, yy, dx, dy);
    }
  }
}
```

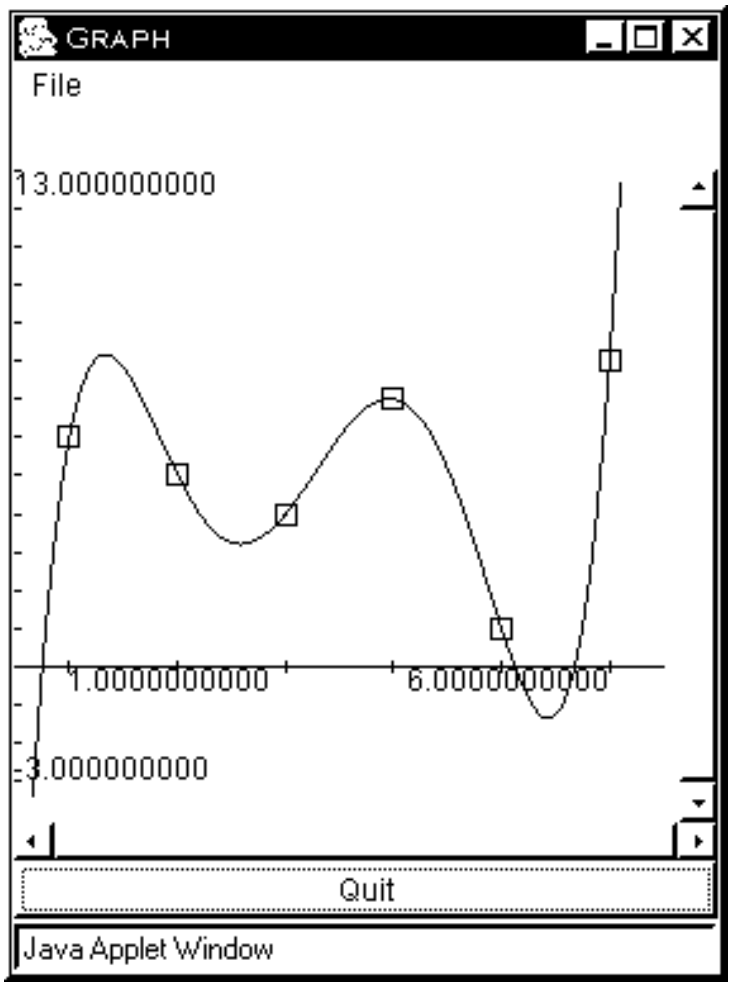
Ohjelman tulostus:

```
Generoidaan pisteiden (x_i, y_i) kautta kulkeva
Lagrangen interpolaatiopolynomi
y = a_(n-1) * x^(n-1) + ... + a_1 * x + a_0.

Valitse 1, jos haluat syöttää luvut itse.
Valitse 2, jos haluat ohjelman generoivan satunnaiset luvut.
Valinta: 2
Anna pisteiden lukumäärä: 6
a_5 = 0.43333
a_4 = -7.20833
a_3 = 44.58333
a_2 = -126.29167
a_1 = 160.48333
a_0 = -66.00000
```

4.3 Splini-interpolaatio

Data-aineiston interpolointi ei yleensä parane nostamalla interpoloinnin kertalukua, sillä korkea-asteiset polynomit heittelehtivät entistä rajummin pisteiden välillä. Järkevämpää on jakaa interpolointiväli lyhyempiin osaväleihin ja käyttää *palapolynomeja*, eli funktioita, jotka voidaan paloittain ilmaista polynomeina. Palapolynomin aste on p, jos kunkin osan asteluku on p. Palapolynomin ei tarvitse olla jatkuva *solmupisteissä* eli palojen päätepisteissä. Käytännössä jatkuvuutta (ja differentioituvuutta) kuitenkin vaaditaan, joten paloittaiseen interpolointiin käytetäänkin yleensä *splinejä*. Ne ovat astetta p olevia palapolynomeja, joiden derivaatat ovat jatkuvia kertalukuun $p-1$ asti ([13] , s. 97–98).

Yksinkertaisin, 1. asteen splini on datapisteiden kautta kulkeva murtoviiva. Tällaisen konstruointi on triviaalia eikä se ole kovin käytännöllinen.



Kuva 4: 5. asteen Lagrangen interpolaatiopolynomi.

Parempi ja suosituin splini on *kuutiollinen splini*, joka nimensä mukaisesti koostuu paloittain määritellyistä kolmannen asteen polynomeista. Solmukohdissa vaaditaan splinin itsensä lisäksi sen ensimmäisen ja toisen derivaatan jatkuvuus. Graafisesti tarkasteltuna tämä tarkoittaa sitä, että kuutiosplinin kuvaaja kaareutuu kaikkialla kauniisti.

Tietokoneiden kehityksen myötä splinien käyttö on mahdollistanut mm. korkeatasoisen 3D-grafiikan tuottamisen ([16] , s. 453–495).

Seuraava ohjelma sovittaa datapisteistöön vertailun vuoksi sekä *B-splinin* että *Catmull-Rom-splinin*. B-splini ei yleensä interpoloi pisteitä, vaan ainoastaan approksimoi. Catmull-Rom-splini sen sijaan on ”aito” interpolaatiopolynomi.

Ohjelmaesimerkki 4.4

```
package examples;

// splini-interpolaatio

import java.awt.Color;
import matutl.Vec;
import no.geosoft.cc.geometry.spline.SplineFactory;

public class Splines extends Lagrange {
  public void main() {
    showText(80, 35);
    println("Generoidaan pisteitä (x_i, y_i) interpoloivat\n"
      +"B-splini ja Catmull-Rom-splini.\n");

    int v=valinta(),
      n=lukumaara("pisteiden", 2);

    if (v==1) {
      x=luoVektori("x", n).getColumnPackedCopy();
      y=luoVektori("y", n).getColumnPackedCopy();

      findXminMax();
    } else {
      x=new double[n];
      y=new double[n];

      for (int i=0; i<n; i++) {
        x[i]=10.0*i/n;
        y[i]=10.0*Math.random();
      }

      xMin=0;
      xMax=n-1;
    }

    findYminMax();

    xInset=yInset=1.0;
    showGraph(x[xMin]-xInset, x[xMax]+xInset,
              y[yMin]-yInset, y[yMax]+yInset, PTS, PTS);
    plotData();
```

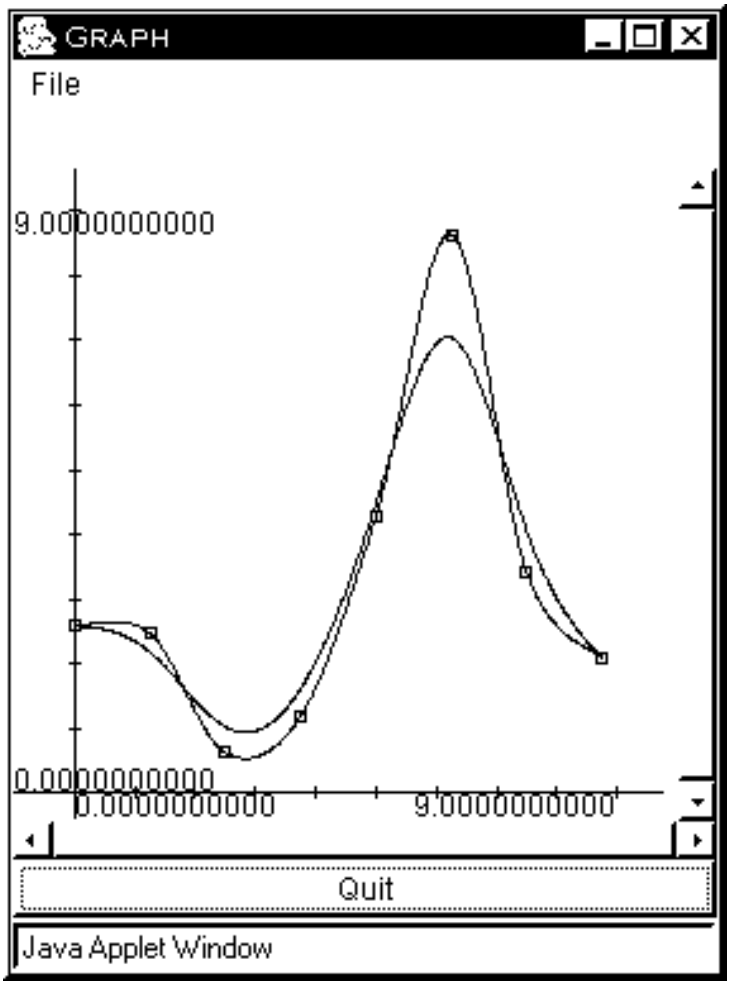
```
    double[] data=new double[3*n];

    for (int i=0; i<n; i++) {
      data[3*i]=x[i];
      data[3*i+1]=y[i];
      data[3*i+2]=0.0;
    }

    int nParts=10*n;
    double[] bs=SplineFactory.createCubic(data, nParts),
      cr=SplineFactory.createCatmullRom(data, nParts);

    print("B-splini näkyy punaisena,\n"
      +"Catmull-Rom-splini sinisenä.");
    drawSpline(bs, Color.red);
    drawSpline(cr, Color.blue);
  }

  private void drawSpline(double[] spline, Color col) {
    line(spline[0], spline[1], spline[3], spline[4], col);

    for (int i=6, sl=spline.length; i<sl; i+=3) {
      line(spline[i], spline[i+1], col);
    }
  }
}
```

Ohjelman tulostus:

```
Generoidaan pisteitä (x_i, y_i) interpoloivat
B-splini ja Catmull-Rom-splini.

Valitse 1, jos haluat syöttää luvut itse.
Valitse 2, jos haluat ohjelman generoivan satunnaiset luvut.
Valinta: 2
Anna pisteiden lukumäärä: 8
B-splini näkyy punaisena,
Catmull-Rom-splini sinisenä.
```

4.4 Numeerinen derivointi

Tässä esiteltävä derivointimenetelmä perustuu aiemmin esiteltyyn Lagrangen interpolaatiopolynomiin (2, s. 72) ja erityisesti tapaukseen, jossa käytetään



Kuva 5: B-splini ja Catmull-Rom-splini.

funktion viittä tasavälisesti sijaitsevaa pistettä $x_0 + ih$ ja vastaavia funktion arvoja f_i, $i = -2, \ldots, 2$ ([10], s. 879, 25.2.15):

$$\begin{aligned} f_p &= f(x_0 + ph) \\ &\approx \frac{(p^2-1)p(p-2)}{24} f_{-2} \\ &\quad -\frac{(p-1)p(p^2-4)}{6} f_{-1} + \frac{(p^2-1)(p^2-4)}{4} f_0 \\ &\quad -\frac{(p+1)p(p^2-4)}{6} f_1 + \frac{(p^2-1)p(p+2)}{24} f_2 \end{aligned}$$

Derivoimalla tämä polynomi saadaan funktion derivaatalle approksimaatio ([10] , s. 883, 25.3.6)

$$\begin{aligned} f_p{}' &= f'(x_0 + ph) \\ &\approx \frac{1}{h}\left(\frac{2p^3 - 3p^2 - p + 1}{12} f_{-2}\right. \\ &\quad -\frac{4p^3 - 3p^2 - 8p + 4}{6} f_{-1} + \frac{2p^3 - 5p}{2} f_0 \\ &\quad \left. - \frac{4p^3 + 3p^2 - 8p - 4}{6} f_1 + \frac{2p^3 + 3p^2 - p - 1}{12} f_2\right) \end{aligned}$$

Seuraavan ohjelman avulla voidaan piirtää funktion ja sen derivaattojen kuvaajat. Käyttäjä voi valita, montako derivaatan kuvaajaa piirretään. Korkeamman kertaluvun derivaatat lasketaan soveltamalla ylläolevaa kaavaa toistuvasti. Piirtoalueelta voidaan lisäksi valita yksi piste, jossa funktion ja sen derivaattojen arvot lasketaan. Kunkin derivaatan (liki)arvo ko. pisteessä lasketaan käyttämällä lineaarista interpolointia kahden lähimmän pisteen välillä.

Piirretään esimerkkinä funktion

$$y(x) = \frac{x^3 - x^2 + x - 12}{4} \tag{3}$$

ja sen kolmen derivaatan kuvaajat. Lasketaan funktion ja derivaattojen arvot pisteessä $x = 3{,}5$.

Ohjelmaesimerkki 4.5

```
package examples;

// numeerinen derivointi

import com.braju.format.*;
import com.eteks.parser.*;
import java.awt.Color;
import matutl.Maths;
import uk.co.jscieng.Plottable;

public class Derivative extends Base implements Plottable {
  CompiledFunction f;

  public void main() {
    showText();
```

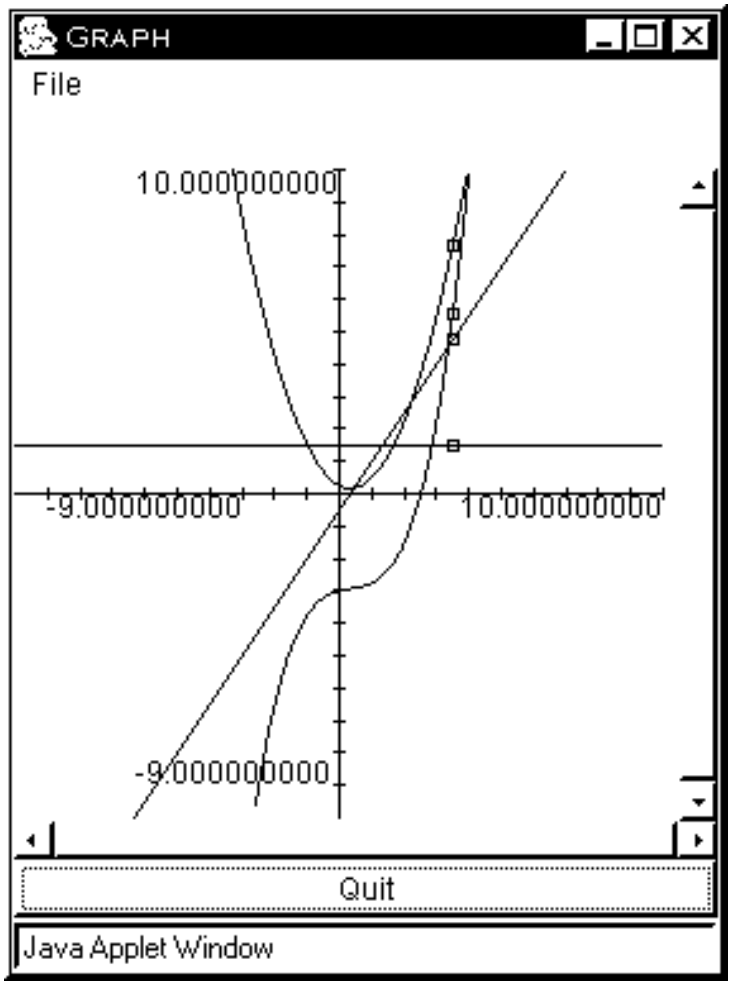
```
println("Piirretään funktio ja sen derivaattojen "
  +"kuvaajia.\nFunktio piirretään sinisellä, "
  +"derivaatat eri väreillä.\nLasketaan funktion ja"
  +"derivaattojen arvo annetussa pisteessä.\n");

try {
  print("Syötä derivoitava funktio: ");
  f=new FunctionParser().compileFunction("f(x)="+readln());
  double x0=-10.0,
    x1=10.0,
    y0=-10.0,
    y1=10.0;

  println("Piirtoalueen oletusrajat: -10<x<10, -10<y<10.\n"
    +"Valitse 1, jos haluat hyväksyä oletusarvot.\n"
    +"Valitse 2, jos haluat syöttää uudet rajat.");

  if (valinta2()==2) {
    print("Syötä piirtoalueen vasen raja: ");
    x0=Double.valueOf(readln()).doubleValue();

    print("Syötä piirtoalueen oikea raja: ");
    x1=Double.valueOf(readln()).doubleValue();

    print("Syötä piirtoalueen alaraja: ");
    y0=Double.valueOf(readln()).doubleValue();

    print("Syötä piirtoalueen yläraja: ");
    y1=Double.valueOf(readln()).doubleValue();
  }

  double dxx=(x1-x0)/128.0,
    dyy=(y1-y0)/128.0;

  print("Syötä piirtoväliltä piste, jossa\n"
    +"funktion ja derivaattojen arvot lasketaan: ");

  double xx;

  do {
    xx=Double.valueOf(readln()).doubleValue();
  } while (xx<x0 || xx>x1);
```

```
  showGraph(x0, x1, y0, y1, PTS, PTS);
  functionPlot(this, x0, x1, colors[0], 1);
  println(Format.sprintf("x = %-10.3f\nf(x) = %-10.3f",
    new Parameters(xx).add(f(xx))));
  plotMark(xx, f(xx), dxx, dyy);

  double y[]=new double[PTS],
    x[]=new double[PTS];
  double h=(x1-x0)/PTS;
  int ii=0;

  for (int i=0; i<PTS; i++) {
    x[i]=x0+i*h;
    y[i]=f(x[i]);

    if (xx>=x[i]) {
      ii=i;
    }
  }

  print("Syötä piirrettävien derivaattojen lukumäärä: ");
  int n=Integer.parseInt(readln());
  String deg="";

  for (int i=1, j=colors.length; i<=n; i++) {
    double[] dy=Maths.numDer(y, h, PTS);
    deg+="\'";

    point(x[0], dy[0], colors[i%j]);

    for (int k=1; k<PTS; k++) {
      line(x[k], dy[k], colors[i%j]);
    }

    y=dy;
    double yy=(y[ii+1]-y[ii])*(xx-x[ii])/h+y[ii];

    println(Format.sprintf("f"+deg+"(x) = %-10.2f",
      new Parameters(yy)));
    plotMark(xx, yy, dxx, dyy);
  }
} catch (Exception e) {
  println("Virhe:\n"+e);
```

```
    }
  }

  public double f(double x) {
    return f.computeFunction(new double[] {x});
  }
}
```

Ohjelman tulostus:

```
Piirretään funktio ja sen derivaattojen kuvaajia.
Funktio piirretään sinisellä, derivaatat eri väreillä.
Lasketaan funktion jaderivaattojen arvo annetussa pisteessä.

Syötä derivoitava funktio: (x^3-x^2+x-12)/4
Piirtoalueen oletusrajat: -10<x<10, -10<y<10.
Valitse 1, jos haluat hyväksyä oletusarvot.
Valitse 2, jos haluat syöttää uudet rajat.
Valinta: 1
Syötä piirtoväliltä piste, jossa
funktion ja derivaattojen arvot lasketaan: 3.5
x = 3.500
f(x) = 5.531
Syötä piirrettävien derivaattojen lukumäärä: 3
f'(x) = 7.69
f''(x) = 4.75
f'''(x) = 1.50
```



Kuva 6: Funktion (3) ja sen kolmen derivaatan kuvaajat.

5 Numeerinen integrointi

Integrointi on keskeinen matemaattinen tehtävä, joten sitä varten on kehitetty useita erilaisia menetelmiä. Tarkastellaan tässä yhteydessä kahta yleistä menetelmää.

5.1 Simpsonin sääntö

Simpsonin sääntö on aiemmin esiteltyä puolisuunnikassääntöä parempi integrointimenetelmä. Sen kertaluku on kolme, eli kaikki korkeintaan kolmannen asteen polynomit integroituvat tarkasti. Menetelmässä käytetään integrointivälin päätepisteiden lisäksi keskipistettä ([22] , s. 100):

$$\int_a^b f(x)\,\mathrm{d}x \approx \frac{h}{3}\left(f(a) + 4f\left(\frac{a+b}{2}\right) + f(b)\right) ,$$

missä $h = \frac{b-a}{2}$.

Jos integrointiväli jaetaan n osaan (n parillinen) ja sovelletaan menetelmää joka osavälillä, saadaan *yhdistetty Simpsonin sääntö*:

$$\int_a^b f(x)\,\mathrm{d}x \approx \frac{h}{3}\left(f_0 + 4f_1 + 2f_2 + \cdots + 2f_{n-2} + 4f_{n-1} + f_n\right) ,$$

missä $f_i = f(a+ih),\ i = 0, 1, 2, \ldots, n$.

Integroidaan kaksi erilaista funktiota Simpsonin säännön avulla. Ohjelma pyytää integroitavan funktion ja integrointivälin käyttäjältä.

Ohjelmaesimerkki 5.1

```
package examples;

// Simpsnin sääntö

import com.braju.format.*;
```

```
import com.eteks.parser.*;
import matutl.Maths;

public class Simpson extends Base {
  private CompiledFunction f;

  public void main() {
    showText();
    println("Lasketaan funktion määrätty integraali\n"
      +"Simpsonin säännön avulla 2, ..., 2000 jakovälillä.\n");

    try {
      print("Syötä integroitava funktio: ");
      f=new FunctionParser().compileFunction("f(x)="+readln());
    } catch (Exception e) {
      print("Virhe:\n"+e);
    }

    print("Syötä integrointivälin alkupiste: ");
    double a=Double.valueOf(readln()).doubleValue();

    print("Syötä integrointivälin loppupiste: ");
    double b=Double.valueOf(readln()).doubleValue();

    println(Format.sprintf("\n%10s %14s",
      new Parameters("jakovälejä").add("integraali")));

    for (int N=2; N<=2000; N*=10) {
      double[] y=new double[N+1];
      double h=(b-a)/N;

      for (int j=0; j<=N; j++) {
        y[j]=f.computeFunction(new double[] {a+j*h});
      }

      double I=Maths.simpson(y, h);

      println(Format.sprintf("%10d %14.7f",
        new Parameters(N).add(I)));
    }
  }
}
```

Ohjelman tulostus syötteellä $y = (x + 2)(x + 1)(x - 4)$:

```
Lasketaan funktion määrätty integraali
Simpsonin säännön avulla 2, ..., 2000 jakovälillä.

Syötä integroitava funktio: (x+2)*(x+1)*(x-4)
Syötä integrointivälin alkupiste: -2
Syötä integrointivälin loppupiste: 2

jakovälejä     integraali
         2    -37.3333333
        20    -37.3333333
       200    -37.3333333
      2000    -37.3333333
```

Ohjelman tulostus syötteellä $y = x\sin(x)$:

```
Lasketaan funktion määrätty integraali
Simpsonin säännön avulla 2, ..., 2000 jakovälillä.

Syötä integroitava funktio: x*sin(x)
Syötä integrointivälin alkupiste: 0
Syötä integrointivälin loppupiste: 10

jakovälejä     integraali
         2    -41.0311610
        20      7.8502925
       200      7.8466945
      2000      7.8466942
```

Kolmannen asteen polynomi integroituu odotetusti tarkasti. Toisenkin funktion kanssa päästään seitsemän desimaalin tarkkuuteen 2000 jakovälillä.

Korkean kertaluvun Newtonin ja Cotesin kaavat ovat numeerisesti epästabiileja, sillä ne sisältävät negatiivisia painokertoimia. Suurempaan tarkkuuteen pyrittäessä on parempi käyttää erityyppisiä integrointimenetelmiä.

5.2 Gaussin integrointi

Newtonin ja Cotesin kaavojen ohella tunnetuimpia integroimissääntöjä ovat Gaussin säännöt. Ne on määritelty välille $[-1, 1]$, mutta mielivaltaiselta (äärelliseltä) väliltä $[a, b]$ voidaan muuttujanvaihdoksella

$$z = \frac{b-a}{2}\,x + \frac{b+a}{2}$$

siirtyä välillä $[-1, 1]$ määriteltyyn integroimismuuttujaan x.

Gaussin ja Legendren sääntö perustuu Legendren polynomeihin, jotka ovat Legendren differentiaaliyhtälön

$$\left(1 - x^2\right) y'' - 2xy' + n(n+1)y = 0$$

ratkaisuja. Ne voidaan esittää muodossa

$$P_n(x) = \frac{1}{2^n\, n!} \frac{\mathrm{d}^n}{\mathrm{d}x^n} \left(x^2 - 1\right)^n \,.$$

Kaksi ensimmäistä Legendren polynomia ovat $P_0(x) = 1$ ja $P_1(x) = x$. Käyttämällä rekursiota

$$(n+1)P_{n+1}(x) - (2n+1)xP_n(x) + nP_{n-1}(x) = 0$$

voidaan ratkaista korkeamman asteen polynomit ([20] , s. 162).

Gaussin ja Legendren säännön solmupisteet $x_k^{(n)}$ ovat polynomin P_n nollakohdat. Vastaavat painokertoimet ovat

$$\omega_k^{(n)} = \frac{2P_n'\left(x_k^{(n)}\right)^2}{1 - x_k^{(n)2}} \,.$$

Näiden perusteella voidaan kirjoittaa Gaussin ja Legendren integrointisääntö

$$\int_{-1}^{1} f(x)\,\mathrm{d}x \approx \sum_{k=1}^{n} \omega_k^{(n)} f\left(x_k^{(n)}\right)$$

välille $[-1, 1]$ ja

$$\int_a^b f(x)\,\mathrm{d}x \approx \frac{b-a}{2} \sum_{k=1}^{n} \omega_k^{(n)} f\left(\frac{b-a}{2}\, x_k^{(n)} + \frac{b+a}{2}\right)$$

mielivaltaiselle välille $[a, b]$ ([20] , s. 222).

Ohjelmaesimerkissä käytetty integrointimetodi on suora käännös Java-kielelle C-kielisestä lähteestä ([18] , s. 152). Ohjelma tulostaa myös käytetyt solmupisteet ja painokertoimet.

Ohjelmaesimerkki 5.2

```
package examples;

// integrointi Gaussin säännöllä

import com.braju.format.*;
import com.eteks.parser.*;
import matutl.Maths;
import uk.co.jscieng.Plottable;

public class GauLeg extends Base implements Plottable {
  private CompiledFunction f;

  public void main() {
    showText();
    println("Lasketaan funktion määrätty integraali\n"
      +"käyttäen Gaussin ja Legendren kvadratuuria.\n");

    try {
      print("Syötä integroitava funktio: ");
      f=new FunctionParser().compileFunction("f(x)="+readln());
    } catch (Exception e) {
      print("Virhe:\n"+e);
    }

    print("Syötä integrointivälin alkupiste: ");
    double a=Double.valueOf(readln()).doubleValue();

    print("Syötä integrointivälin loppupiste: ");
    double b=Double.valueOf(readln()).doubleValue();

    int n=lukumaara("kvadratuurin pisteiden", 1);
    double[] x=new double[n+1],
      w=new double[n+1];
    double I=Maths.gauLeg(a, b, x, w, n, this);

    println("Kvadratuurin pisteet ja painokertoimet:");

    for (int i=1; i<=n; i++) {
      println(Format.sprintf("x_"+i+" = %-10.5f"
        +"w_"+i+" = %-8.5f", new Parameters(x[i]).add(w[i])));
    }
```

```
    print(Format.sprintf("Integraalin arvo = %-12.6f",
      new Parameters(I)));
  }

  public double f(double x) {
    return f.computeFunction(new double[] {x});
  }
}
```

Ohjelman tulostus:

```
Lasketaan funktion määrätty integraali
käyttäen Gaussin ja Legendren kvadratuuria.

Syötä integroitava funktio: exp(-x)
Syötä integrointivälin alkupiste: 0
Syötä integrointivälin loppupiste: 4
Anna kvadratuurin pisteiden lukumäärä: 4
Kvadratuurin pisteet ja painokertoimet:
x_1 = 0.27773   w_1 = 0.69571
x_2 = 1.32004   w_2 = 1.30429
x_3 = 2.67996   w_3 = 1.30429
x_4 = 3.72227   w_4 = 0.69571
Integraalin arvo = 0.981662
```

Integraalin likiarvo kuudella desimaalilla on 0,981684 , joten jo neljällä pisteellä saatiin hyvä likiarvo.

6 Differentiaaliyhtälöt

Differentiaaliyhtälöillä voidaan kuvata monia ilmiöitä niin fysiikassa kuin muissakin luonnontieteissä. Differentiaaliyhtälöt voidaan jakaa *alkuarvo-* ja *reuna-arvotehtäviin*. Ensimmäisen tyypin tehtävissä ratkaistavan systeemin tila lähtöpisteessä (ajan hetkellä $t = 0$ tms.) tiedetään. Jälkimmäisissä taas tunnetaan ehtoja, jotka systeemin on toteutettava tarkasteluvälin molemmissa päissä. Reuna-arvotehtävien ratkaiseminen on vaikeampaa kuin alkuarvotehtävien, eikä ratkaisun olemassaolokaan ole aina itsestään selvää.

6.1 Differenssimenetelmä

Tarkastellaan *differenssimenetelmää*, joka soveltuu reuna-arvotehtävien ratkaisemiseen. Differenssimenetelmässä funktion derivaatat korvataan sopivilla erotusosamäärillä ja alkuperäistä yhtälöä approksimoidaan yhtälöryhmän avulla diskreetissä pisteistössä ([13] , s. 209).

Johdetaan menetelmä *toisen kertaluvun* (kertaluku ilmoittaa korkeimman derivaatan asteluvun) reuna-arvotehtävälle

$$y'' + p(x)\, y' + q(x)\, y = r(x), \quad y(a) = \alpha\,,\, y(b) = \beta\,. \tag{4}$$

Derivaattojen diskretointi: ensimmäistä derivaattaa approksimoi lauseke

$$y' \approx \frac{y(x+h) - y(x-h)}{2h}$$

ja toista

$$y'' \approx \frac{y(x+h) - 2y(x) + y(x-h)}{h^2}\,.$$

Muuttujien diskretointi: Jaetaan tarkasteluväli $[a, b]$ n:ään osaväliin, jolloin askel $h = \frac{b-a}{n}$ ja $x_i = a + ih\,,\ i = 0, \ldots, n\,.$ Vastaavasti käytetään merkintää $f_i = f(x_i) = f(a + ih)\,.$ Kohdassa x_j differentiaaliyhtälö saa siis muodon

$$\frac{y_{j+1} - 2y_j + y_{j-1}}{h^2} + p_j\, \frac{y_{j+1} - y_{j-1}}{2h} + q_j\, y_j = r_j\,, \quad 0 < j < n\,.$$

Järjestelemällä termit uudestaan ja kertomalla termillä h^2 saadaan

$$\left(1 - \frac{h}{2}\,p_j\right) y_{j-1} - \left(2 - h^2 q_j\right) y_j + \left(1 + \frac{h}{2}\,p_j\right) y_{j+1} = h^2 r_j\,, \quad 0 < j < n\,.$$

Tämä voidaan esittää matriisimuodossa

$$A\boldsymbol{y} = \boldsymbol{b}\,,$$

missä (huomaa kerroinmatriisin tridiagonaalisuus)

$$A = \begin{pmatrix} -2 + h^2 q_1 & 1 + \frac{h}{2}\,p_1 & 0 & \cdots & 0 \\ 1 - \frac{h}{2}\,p_2 & -2 + h^2 q_2 & 1 + \frac{h}{2}\,p_2 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 1 - \frac{h}{2}\,p_{n-2} & -2 + h^2 q_{n-2} & 1 + \frac{h}{2}\,p_{n-2} \\ 0 & \cdots & 0 & 1 - \frac{h}{2}\,p_{n-1} & -2 + h^2 q_{n-1} \end{pmatrix},$$

$$\boldsymbol{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_{n-2} \\ y_{n-1} \end{pmatrix} \quad \text{ja} \quad \boldsymbol{b} = \begin{pmatrix} h^2 r_1 - \left(1 - \frac{h}{2}\,p_1\right)\alpha \\ h^2 r_2 \\ \vdots \\ h^2 r_{n-2} \\ h^2 r_{n-1} - \left(1 + \frac{h}{2}\,p_{n-1}\right)\beta \end{pmatrix}.$$

Tämä matriisiyhtälö voidaan ratkaista normaalisti LU-hajotelmaa käyttäen (tridiagonaalimatriisiyhtälöille on kehitetty myös omia, tehokkaampia ratkaisumenetelmiä).

Seuraavan ohjelman avulla voidaan etsiä likimääräisratkaisuja yhtälön (4) kaltaisille differentiaaliyhtälöille. Ratkaistaan esimerkkinä *Besselin differentiaaliyhtälö* ([20] , s. 150, 27.1)

$$x^2 y'' + xy' + (x^2 - n^2)y = 0\,, \tag{5}$$

kun $n = 0$. Tämän ratkaisu on *Besselin funktio* ([20] , s. 150, 27.5)

$$J_0(x) = 1 - \frac{x^2}{2^2} + \frac{x^4}{2^2 \cdot 4^2} - \frac{x^6}{2^2 \cdot 4^2 \cdot 6^2} + \cdots$$

Ohjelmaa varten yhtälö (5) on muutettava muotoon

$$y'' + \frac{1}{x}\,y' + y = 0\,. \tag{6}$$

Valitaan tarkasteluväliksi $[0; 11{,}7915]$, missä oikea päätepiste on funktion $J_0(x)$ neljännen nollakohdan likiarvo ([20] , s. 254). Selvästi $J_0(0) = 1$.

Ohjelmaesimerkki 6.1

```
package examples;

// FDM = Finite Difference Method
// lineaarisen reuna-arvotehtävän
// y''(x) + p(x)y'(x) + q(x)y(x) = r(x), y(a) = alfa, y(b) = beeta
// ratkaiseminen differenssimenetelmällä

import com.braju.format.*;
import com.eteks.parser.*;
import Jama.Matrix;
import matutl.Maths;

public class FDM extends Base {
  private CompiledFunction p, q, r;
  private Matrix X, Y;
  private double h, h2;

  public void main() {
    showText(80, 35);
    println("Ratkaistaan reuna-arvotehtävä\n"
      +"y''(x) + p(x)y'(x) + q(x)y(x) = r(x), "
      +"y(a)=alfa, y(b)=beeta,\n"
      +"differenssimenetelmällä.\n");

    try {
      print("Syötä funktio p(x): ");
      p=new FunctionParser().compileFunction("p(x)="+readln());

      print("Syötä funktio q(x): ");
      q=new FunctionParser().compileFunction("q(x)="+readln());

      print("Syötä funktio r(x): ");
      r=new FunctionParser().compileFunction("r(x)="+readln());
    } catch (Exception exception) {
      print("Virhe:\n"+exception);
    }

    print("Anna alkupiste a: ");
    double a=Double.valueOf(readln()).doubleValue();

    print("Anna reuna-arvo alfa: ");
    double alpha=Double.valueOf(readln()).doubleValue();
```

```
print("Anna loppupiste b: ");
double b=Double.valueOf(readln()).doubleValue();

print("Anna reuna-arvo beeta: ");
double beta=Double.valueOf(readln()).doubleValue();

print("Anna menetelmän osavälien lukumäärä : ");
int n=Integer.parseInt(readln());

X=new Matrix(n+1, 1);
Y=new Matrix(n+1, 1);
Matrix triDiag=new Matrix(n-1, n-1, 0.0),
  rhs=new Matrix(n-1, 1);

h=(b-a)/n;
h2=h*h;
double x=a+h;

X.set(0, 0, a);
Y.set(0, 0, alpha);
rhs.set(0, 0, h2*r(x)-(1-0.5*h*p(x))*alpha);

for (int i=0; i<n-1; i++) {
  X.set(i+1, 0, x);

  if (i>0) {
    triDiag.set(i, i-1, 1.0-0.5*h*p(x));
  }

  triDiag.set(i, i, -2.0+h2*q(x));

  if (i<n-2) {
    triDiag.set(i, i+1, 1.0+0.5*h*p(x));
  }

  if (i>0 && i<n-2) {
    rhs.set(i, 0, h2*r(x));
  }

  x+=h;
}
```

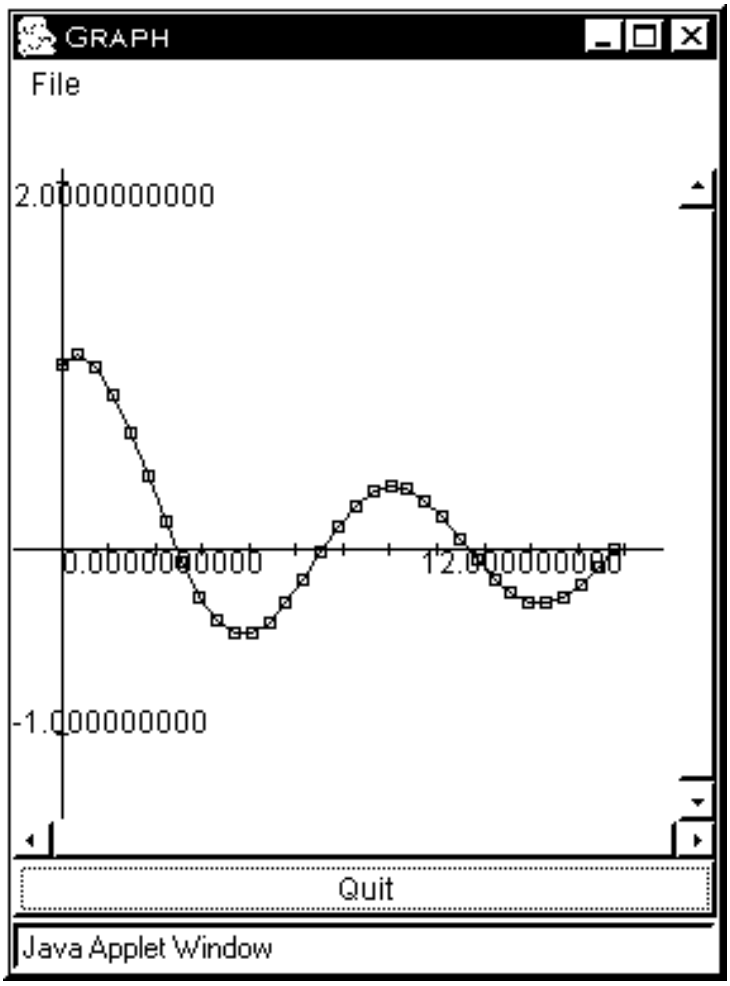
```
  rhs.set(n-2, 0, h2*r(x)-(1+0.5*h*p(x))*beta);
  X.set(n, 0, b);
  Y.set(n, 0, beta);
  Y.setMatrix(1, n-1, 0, 0, triDiag.solve(rhs));

  println("Differentiaaliyhtälön ratkaisu:");
  println(Format.sprintf("%6s %10s",
    new Parameters("x").add("f(x)")));

  for (int i=0; i<=n; i++) {
    println(Format.sprintf("%6.3f %10.5f",
      new Parameters(X.get(i, 0)).add(Y.get(i, 0))));
  }

  double xMin=Maths.getMin(X, 0)-1.0,
    xMax=Maths.getMax(X, 0)+1.0,
    yMin=Maths.getMin(Y, 0)-1.0,
    yMax=Maths.getMax(Y, 0)+1.0,
    dx=(xMax-xMin)/128.0,
    dy=(yMax-yMin)/128.0;

  showGraph(xMin, xMax, yMin, yMax, PTS, PTS);

  for (int i=0; i<=n; i++) {
    plotMark(X.get(i, 0), Y.get(i, 0), dx, dy);
  }

  point(X.get(0, 0), Y.get(0, 0), colors[8]);

  for (int i=1; i<=n; i++) {
    line(X.get(i, 0), Y.get(i, 0), colors[8]);
  }
}

private double p(double x) {
  return p.computeFunction(new double[] {x});
}

private double q(double x) {
  return q.computeFunction(new double[] {x});
}

private double r(double x) {
```

```
    return r.computeFunction(new double[] {x});
  }
}
```

Ohjelman tulostus:

```
Ratkaistaan reuna-arvotehtävä
y''(x) + p(x)y'(x) + q(x)y(x) = r(x), y(a)=alfa, y(b)=beeta,
differenssimenetelmällä.

Syötä funktio p(x): 1/x
Syötä funktio q(x): 1
Syötä funktio r(x): 0
Anna alkupiste a: 0
Anna reuna-arvo alfa: 1
Anna loppupiste b: 11.7915
Anna reuna-arvo beeta: 0
Anna menetelmän osavälien lukumäärä : 32
Differentiaaliyhtälön ratkaisu:
     x       f(x)
 0.000    1.00000
 0.368    1.06467
 0.737    0.98985
 1.105    0.83743
 1.474    0.63110
 1.842    0.39445
 2.211    0.15214
 2.579   -0.07196
 2.948   -0.25706
 3.316   -0.38754
 3.685   -0.45443
 4.053   -0.45618
 4.422   -0.39854
 4.790   -0.29355
 5.159   -0.15797
 5.527   -0.01102
 5.896    0.12790
 6.264    0.24155
 6.633    0.31685
 7.001    0.34622
 7.370    0.32828
 7.738    0.26773
 8.107    0.17449
 8.475    0.06222
```

```
 8.844   -0.05354
 9.212   -0.15745
 9.581   -0.23632
 9.949   -0.28074
10.318   -0.28612
10.686   -0.25314
11.055   -0.18749
11.423   -0.09895
11.791    0.00000
```



Kuva 7: Differentiaaliyhtälön (6) ratkaisun kuvaaja.

7 Muita matemaattisia sovelluksia

Tarkastellaan lopuksi vielä käyrän ja pinnan piirtämistä sekä erästä historiallisestikin tärkeätä esimerkkiä matematiikan maailmasta.

7.1 Käyrän piirtäminen

Niin käyrän kuin pinnankin piirtämiseen luodut luokat toimivat niin, että niille voidaan antaa komentoriviparametrina piirrettävän funktion lauseke. Valinnaisilla parametreilla voidaan säädellä piirtoalueen kokoa. Ohjelma sisältää myös mahdollisuuden verkossa suorittamiseen. Tällöin parametrit syötetään vasta ohjelmassa. Jos ohjelma suoritetaan komentoriviltä, on oikea kutsu muotoa

```
java examples.Graph ''f(x)'' [xMin xMax [yMin yMax]]
```

Hakasulkeissa olevat, piirtoalueen määrittelevät parametrit ovat valinnaisia. Verkossa ajettaessa ainoa ero on se, että piirrettävää funktiota ei laiteta lainausmerkkeihin.

Jos piirtoaluetta ei määritellä erikseen, käytetään ohjelman oletuspiirtoväliä $-8 \leq x \leq 8$. Pystysuunnassa piirtoaluetta skaalataan niin, että funktion kuvaaja näkyy kokonaisuudessaan.

Ohjelmaesimerkki 7.1

```
package examples;

// komentoriviparametrina annetun funktion piirtäminen

import com.braju.format.*;
import com.eteks.parser.*;
import java.util.StringTokenizer;
import uk.co.jscieng.*;

public class Graph extends Base implements Plottable {
  private CompiledFunction f;
```

```
private double xMin, xMax, yMin, yMax;

public Graph() {}

public Graph(CompiledFunction cf,
    double x0, double x1, double y0, double y1) {
  f=cf;
  xMin=x0;
  xMax=x1;
  yMin=y0;
  yMax=y1;
}

public Graph(CompiledFunction cf,
    double x0, double x1) {
  f=cf;
  xMin=x0;
  xMax=x1;
  yMin=1.0E10;
  yMax=-1.0E10;

  for (int i=0; i<PTS; i++) {
    try {
      double y=f(xMin+i*(xMax-xMin)/PTS);

      if (y>Double.NEGATIVE_INFINITY &&
          y<Double.POSITIVE_INFINITY) {
        yMin=Math.min(yMin, y);
        yMax=Math.max(yMax, y);
      }
    } catch (Exception exception) {
      showText();
      println("Virhe:\n"+exception);
    }
  }
}

public void main() {
  showText();
  println("Syötä piirrettävän funktion f(x) lauseke.\n"
    +"Lisäksi voit syöttää piirtoalueen rajat.\n"
    +"Erota argumentit välilyönneillä.");
```

```
  StringTokenizer st=new StringTokenizer(readln(), " ");
  int n=st.countTokens();
  String[] args=new String[n];

  for (int i=0; i<n; i++) {
    args[i]=st.nextToken();
  }

  main(args);
}

public static void main(String[] args) {
  Graph g;

  try {
    switch (args.length) {
      case 1:
        g=new Graph(
          new FunctionParser().compileFunction("f(x)="+args[0]),
          -8.0, 8.0);
        break;
      case 3:
        g=new Graph(
          new FunctionParser().compileFunction("f(x)="+args[0]),
          Double.valueOf(args[1]).doubleValue(),
          Double.valueOf(args[2]).doubleValue());
        break;
      case 5:
        g=new Graph(
          new FunctionParser().compileFunction("f(x)="+args[0]),
          Double.valueOf(args[1]).doubleValue(),
          Double.valueOf(args[2]).doubleValue(),
          Double.valueOf(args[3]).doubleValue(),
          Double.valueOf(args[4]).doubleValue());
        break;
      default:
        throw new Exception(
          "Funktio virheellinen tai muu virhe.");
    }

    g.draw();
  } catch (Exception exception) {
    showText();
```

```
      print("Virhe:\n"+exception+"\nKayttoesim.: "
        +"java examples.Graph \"sin(x)*cos(x)\" -6 6 -3 3"
        +"\nLainausmerkkejä tulee käyttää vain"
        +"komentoriviltä ajettaessa.");
    }
  }

  public void draw() {
    showText();
    print("Piirretään funktio "+f.getDefinition()+"...");
    showGraph(xMin, xMax, yMin, yMax, PTS, PTS);
    functionPlot(this, xMin, xMax, colors[8], 1);

  }

  public double f(double x) {
    return f.computeFunction(new double[] {x});
  }
}
```

Piirretään esimerkkinä funktion $f(x) = x \sin x$ kuvaaja ja määritellään piirtoalueeksi $-6 \leq x, y \leq 6$ (ks. kuva 8). Komentorivillä annetaan tällöin käsky

```
java examples.Graph ''x*sin(x)'' -6 6 -6 6
```

Ohjelman tulostus (komentorivi):

```
Piirretään funktio f(x)=x*sin(x)...
```

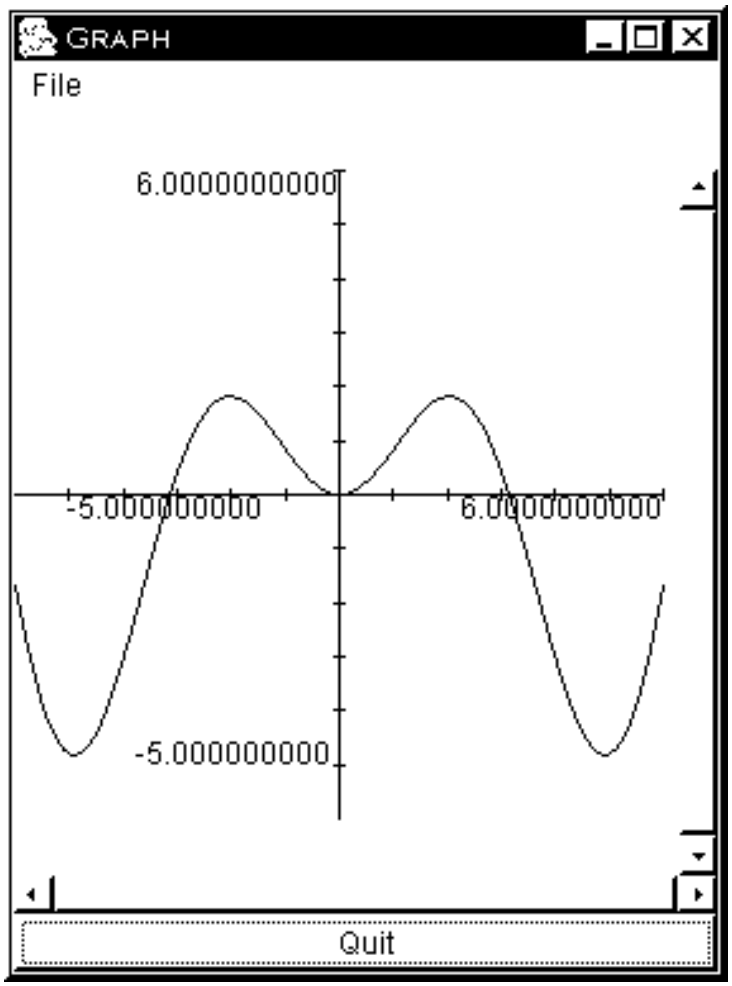
Ohjelman tulostus (appletti):

```
Syötä piirrettävän funktion f(x) lauseke.
Lisäksi voit syöttää piirtoalueen rajat.
Erota argumentit välilyönneillä.
x*sin(x) -6 6 -6 6
Piirretään funktio f(x)=x*sin(x)...
```

7.2 Pinnan piirtäminen

Pinnan piirtäminen tapahtuu samaan tyyliin kuin käyränkin piirtäminen. Käsky komentorivillä on nyt muotoa

```
java examples.Graph ''f(x,y)'' [xMin xMax [yMin yMax][zMin zMax]
[viivojen lkm]]
```



Kuva 8: Funktion $f(x) = x\sin(x)$ kuvaaja.

Hakasulkeissa olevat parametrit ovat jälleen valinnaisia. Nyt voidaan määritellä myös z-koordinaatin rajat sekä viivojen (tai oikeastaan viivojen väliin jäävien ruutujen) lukumäärä, joka määrää, montako viivaa (ruutua) x- ja y-suunnassa piirretään. Oletusarvo on 32. Piirtoalueen oletusrajat ovat $-8 \leq x, y \leq 8$. Pystysuunnassa piirtoaluetta skaalataan jälleen kuvaajan mukaan.

Kannattaa huomata, että piirtoikkunassa näkyvät koordinaatit tarkoittavat nyt y- ja z-koordinaatteja, koska alustana toimii edelleen `SciGraph`, joka on tarkoitettu vain kaksiulotteisten käyrien piirtämiseen. Kuvaan saadaan kolmiulotteisuuden tuntua projisoimalla x-koordinaatti sopivasti tasoon.

Ohjelmaesimerkki 7.2

```
package examples;

// komentoriviparametrina annetun funktion piirtäminen

import com.braju.format.*;
import com.eteks.parser.*;
import Jama.Matrix;
import java.util.StringTokenizer;
import uk.co.jscieng.*;

public class Graph3D extends Base {
  public static final int GRID=32;
  public static final double C=1.0/(2.0*Math.sqrt(2.0));
  private CompiledFunction f;
  private double xMin, xMax, yMin, yMax, zMin, zMax;
  private int grid;
  private Matrix zMat;

  public Graph3D() {}

  public Graph3D(CompiledFunction cf,
      double x0, double x1, double y0, double y1, int g) {
    f=cf;
    xMin=x0;
    xMax=x1;
    yMin=y0;
    yMax=y1;
    zMin=1.0E10;
    zMax=-1.0E10;
    grid=g;
    zMat=new Matrix(grid+1, grid+1);

    for (int i=0; i<=grid; i++) {
      double x=xMin+i*(xMax-xMin)/grid;

      for (int j=0; j<=grid; j++) {
        try {
          double z=f(x, yMin+j*(yMax-yMin)/grid);

          if (z>Double.NEGATIVE_INFINITY &&
              z<Double.POSITIVE_INFINITY) {
            zMin=Math.min(zMin, z);
```

```
          zMax=Math.max(zMax, z);
          zMat.set(i, j, z);
        }
      } catch (Exception exception) {
        showText();
        println("Virhe:\n"+exception);
      }
    }
  }
}

public Graph3D(CompiledFunction cf,
    double x0, double x1, double y0, double y1,
    double z0, double z1, int g) {
  this(cf, x0, x1, y0, y1, g);

  zMin=z0;
  zMax=z1;
}

public void main() {
  showText();
  println("Syötä piirrettävän funktion f(x,y) lauseke.\n"
    +"Lisäksi voit syöttää piirtoalueen rajat sekä viivojen "
    +"lukumäärän.\nErota argumentit välilyönneillä.");

  StringTokenizer st=new StringTokenizer(readln(), " ");
  int n=st.countTokens();
  String[] args=new String[n];

  for (int i=0; i<n; i++) {
    args[i]=st.nextToken();
  }

  main(args);
}

public static void main(String[] args) {
  Graph3D g;

  try {
    switch (args.length) {
      case 1:
```

```
  g=new Graph3D(
    new FunctionParser().compileFunction("f(x,y)="+args[0]),
    -8.0, 8.0, -8.0, 8.0, GRID);
  break;
case 2:
  g=new Graph3D(
    new FunctionParser().compileFunction("f(x,y)="+args[0]),
    -8.0, 8.0, -8.0, 8.0, Integer.parseInt(args[1]));
  break;
case 3:
  g=new Graph3D(
    new FunctionParser().compileFunction("f(x,y)="+args[0]),
    Double.valueOf(args[1]).doubleValue(),
    Double.valueOf(args[2]).doubleValue(),
    -8.0, 8.0, GRID);
  break;
case 4:
  g=new Graph3D(
    new FunctionParser().compileFunction("f(x,y)="+args[0]),
    Double.valueOf(args[1]).doubleValue(),
    Double.valueOf(args[2]).doubleValue(),
    -8.0, 8.0, Integer.parseInt(args[3]));
  break;
case 5:
  g=new Graph3D(
    new FunctionParser().compileFunction("f(x,y)="+args[0]),
    Double.valueOf(args[1]).doubleValue(),
    Double.valueOf(args[2]).doubleValue(),
    Double.valueOf(args[3]).doubleValue(),
    Double.valueOf(args[4]).doubleValue(), GRID);
  break;
case 6:
  g=new Graph3D(
    new FunctionParser().compileFunction("f(x,y)="+args[0]),
    Double.valueOf(args[1]).doubleValue(),
    Double.valueOf(args[2]).doubleValue(),
    Double.valueOf(args[3]).doubleValue(),
    Double.valueOf(args[4]).doubleValue(),
    Integer.parseInt(args[5]));
  break;
case 7:
  g=new Graph3D(
    new FunctionParser().compileFunction("f(x,y)="+args[0]),
```

```
          Double.valueOf(args[1]).doubleValue(),
          Double.valueOf(args[2]).doubleValue(),
          Double.valueOf(args[3]).doubleValue(),
          Double.valueOf(args[4]).doubleValue(),
          Double.valueOf(args[5]).doubleValue(),
          Double.valueOf(args[6]).doubleValue(), GRID);
        break;
      case 8:
        g=new Graph3D(
          new FunctionParser().compileFunction("f(x,y)="+args[0]),
          Double.valueOf(args[1]).doubleValue(),
          Double.valueOf(args[2]).doubleValue(),
          Double.valueOf(args[3]).doubleValue(),
          Double.valueOf(args[4]).doubleValue(),
          Double.valueOf(args[5]).doubleValue(),
          Double.valueOf(args[6]).doubleValue(),
          Integer.parseInt(args[7]));
        break;
      default:
        throw new Exception(
          "Funktio virheellinen tai muu virhe.");
    }

    g.draw();
  } catch (Exception exception) {
    showText();
    print("Virhe:\n"+exception
      +"\nKayttoesim.: java examples.Graph3D \"sin(x)*cos(y)\" "
      +"-3 3 -3 3 -3 3 64\nLainausmerkkejä tulee käyttää vain"
      +"komentoriviltä ajettaessa.");
  }
}

public void draw() {
  showText();
  print("Piirretään funktio "+f.getDefinition()+"...");

  showGraph(Math.min(-C*xMax, yMin),
    Math.max(-C*xMin, yMax), Math.min(-C*xMax, zMin),
    Math.max(-C*xMin, zMax), PTS, PTS);
  line(-C*xMax, -C*xMax, -C*xMin, -C*xMin);

  double dx=(xMax-xMin)/grid,
```

```
      dy=(yMax-yMin)/grid;

    for (int i=0; i<grid; i++) {
      for (int j=0; j<grid; j++) {
        double x=xMin+i*dx,
          y=yMin+j*dy;

        line(y-C*x, zMat.get(i, j)-C*x,
          (y+dy)-C*x, zMat.get(i, j+1)-C*x, colors[8]);
        line(y-C*x, zMat.get(i, j)-C*x,
          y-C*(x+dx), zMat.get(i+1, j)-C*(x+dx), colors[8]);
      }
    }
  }

  public double f(double x, double y) {
    return f.computeFunction(new double[] {x, y});
  }
}
```

Piirretään funktion $f(x,y) = 2\sin(x^2+y^2)/(x^2+y^2)$ kuvaaja ja määritellään piirtoalueeksi $-6 \leq x,\, y \leq 6\,, -5 \leq z \leq 5$ (ks. kuva 9). Komentorivillä annetaan nyt käsky

```
java examples.Graph3D ''2*sin(x^2+y^2)/(x^2+y^2)''
-6 6 -6 6 -5 5 64
```

Ohjelman tulostus (komentorivi):

```
Piirretään funktio f(x,y)=2*sin(x^2+y^2)/(x^2+y^2)...
```

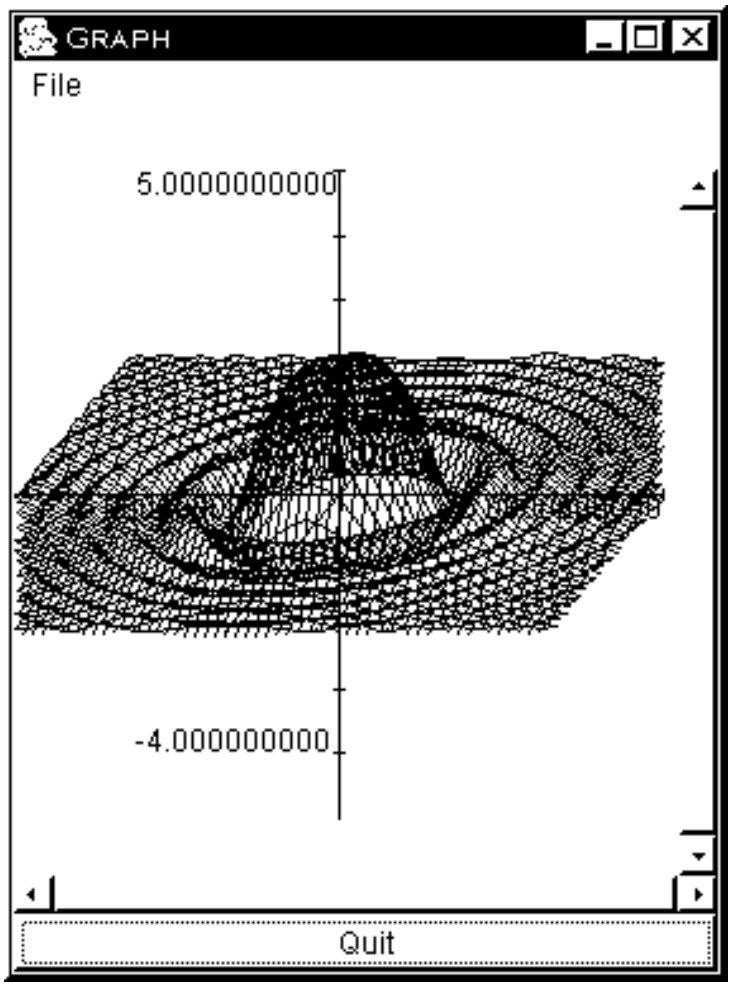
Ohjelman tulostus (appletti):

```
Syötä piirrettävän funktion f(x,y) lauseke.
Lisäksi voit syöttää piirtoalueen rajat sekä viivojen lukumäärän.
Erota argumentit välilyönneillä.
2*sin(x^2+y^2)/(x^2+y^2) -6 6 -6 6 -5 5 64
Piirretään funktio f(x,y)=2*sin(x^2+y^2)/(x^2+y^2)...
```

7.3 Hypergeometrinen funktio

Hypergeometrinen differentiaaliyhtälö on toisen kertaluvun differentiaaliyhtälö, jolla on kolme *säännöllistä singulaaripistettä*. Sopivalla lineaarimuunnoksella yhtälö saadaan muotoon

$$x(1-x)y'' + (c-(a+b+1)x)y' - aby = 0\,.$$



Kuva 9: Funktion $f(x,y) = 2\sin(x^2+y^2)/(x^2+y^2)$ kuvaaja.

Tämän differentiaaliyhtälön ratkaisu voidaan esittää sarjakehitelmänä

$$F(a,b;c;x) = {}_2F_1(a,b;c;x) = 1 + \frac{ab}{1\cdot c}\,x + \frac{a(a+1)b(b+1)}{1\cdot 2\cdot c(c+1)}\,x^2 + \cdots$$
$$= \sum_{n=0}^{\infty} \frac{(a)_n (b)_n}{(c)_n}\frac{x^n}{n!}\,, \qquad (7)$$

jota kutsutaan *(Gaussin) hypergeometriseksi funktioksi*. Kaavassa käytetty merkintä

$$(\alpha)_n = \alpha(\alpha+1)\cdots(\alpha+n-1)\,,\ (\alpha)_0 = 1$$

on ns. *Pochhammerin symboli* [8].

Sarjakehitelmä (7) on määritelty myös kompleksiluvuille z ja se suppenee yksikköympyrän $|z| = 1$ sisällä. Ympyrän kehällä sarja

- hajaantuu, jos $\mathrm{Re}(c - a - b) \leq -1$
- suppenee, jos $\mathrm{Re}(c - a - b) > 0$
- suppenee ehdollisesti, jos $-1 < \mathrm{Re}(c - a - b) \leq 0$; poislukien piste $z = 1$.

Parametrien a tai b kokonaislukuarvoilla sarja katkeaa polynomiksi ([10], s. 556).

Erityispiirteidensä vuoksi Gaussin funktiota on tutkittu jo vuosisatojen ajan ja se on merkittävässä roolissa koko erikoisfunktioiden tutkimuksen saralla.

Seuraavan ohjelman avulla voidaan piirtää hypergeometrisen funktion ${}_2F_1$ kuvaajia eri parametreilla (ks. kuva 10). Funktion arvot lasketaan kaavasta (7). Summaan otetaan sarjan alusta termit, joiden itseisarvo on suurempi kuin 10^{-17}.

Ohjelmaesimerkki 7.3

```
package examples;

// hypergeometrinen funktio 2F1

import com.braju.format.*;
import java.awt.Color;
import uk.co.jscieng.Plottable;

public class HypGeom extends Base implements Plottable {
  protected final int N=colors.length;
  protected double a, b, c;

  public void main() {
    showGraph(-1.1, 1.1, -1.0, 7, PTS, PTS);
    showText();
    println("Piirretään hypergeometrinen funktio\n2F1(a, b; c; x) "
      +"= 1 + ab/(1!c)x + a(a+1)b(b+1)/(2!c(c+1))x^2 + ...\n\n"
      +"Piirtämisen jälkeen\n"
      +"valitse 1, jos haluat syöttää uudet arvot, tai\n"
      +"valitse 2, jos haluat tyhjentää ruudun.");

    int i=0;
```

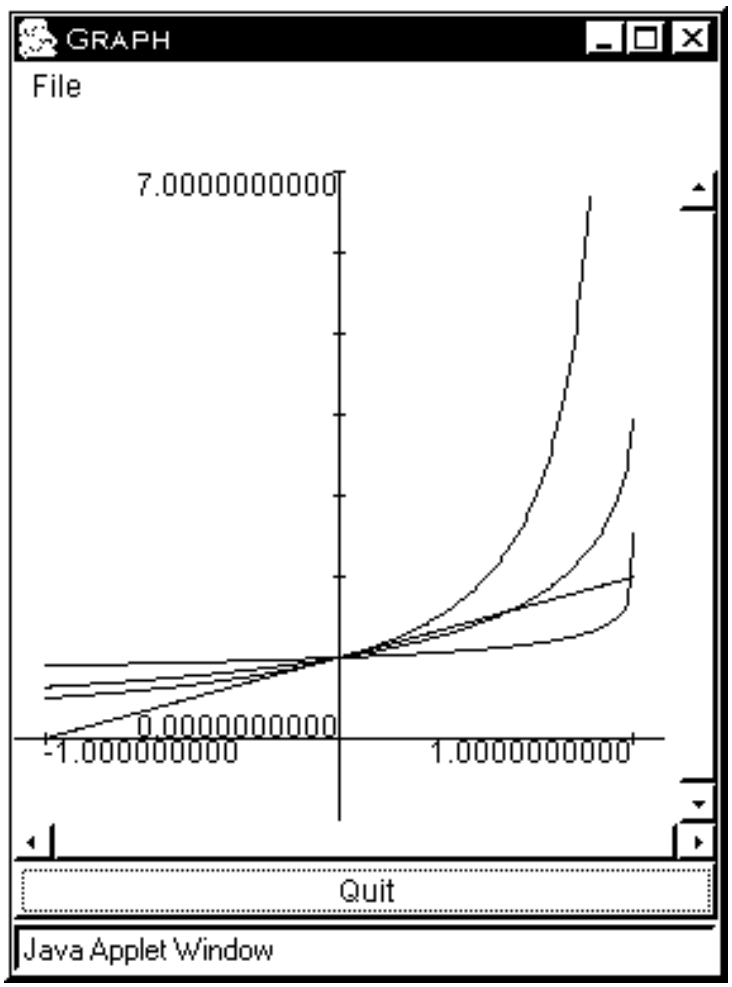
```
  while (true) {
    a=param("a");
    b=param("b");
    c=param("c");

    functionPlot(this, -0.999, 0.999, colors[i++%N], 1);
    println(Format.sprintf(
      "Piirretään 2F1(%3.1f, %3.1f; %3.1f; x).",
      new Parameters(a).add(b).add(c)));

    if (choose()==2) {
      clear();
    }
  }
}

public double f(double x) {
  return hypGeom(a, b, c, x);
}

public double param(String name) {
  print("Anna parametri "+name+": ");

  try {
    return Double.valueOf(readln()).doubleValue();
  } catch (Exception e) {
    print("Yritä uudelleen. ");

    return param(name);
  }
}

// 2F1(a,b,c;x) = 1 + ab/(1!c)x + a(a+1)b(b+1)/(2!c(c+1))x^2 + ...
public static double hypGeom(
    double a, double b, double c, double x) {
  double sum=1.0, term=1.0;
  int j=1;

  do {
    term*=(a*b/(j*c)*x);
    sum+=term;
    a+=1.0;
```

```
      b+=1.0;
      c+=1.0;
      j++;
    } while (Math.abs(term)>1.0e-17 && j<25000);

    return sum;
  }

  public int choose() {
    int c=0;

    do {
      print("Valinta: ");

       try {
        c=Integer.parseInt(readln());
      } catch (Exception e) {}
    } while (c!=1 && c!=2);

    return c;
  }
}
```

Ohjelman tulostus:

```
Piirretään hypergeometrinen funktio
2F1(a, b; c; x) = 1 + ab/(1!c)x + a(a+1)b(b+1)/(2!c(c+1))x^2 + ...

Piirtämisen jälkeen
valitse 1, jos haluat syöttää uudet arvot, tai
valitse 2, jos haluat tyhjentää ruudun.
Anna parametri a: 1
Anna parametri b: 1
Anna parametri c: 1
Piirretään 2F1(1.0, 1.0; 1.0; x).
Valinta: 1
Anna parametri a: .2
Anna parametri b: .3
Anna parametri c: .4
Piirretään 2F1(0.2, 0.3; 0.4; x).
Valinta: 1
Anna parametri a: 1
Anna parametri b: 3
Anna parametri c: 5
```

```
Piirretään 2F1(1.0, 3.0; 5.0; x).
Valinta: 1
Anna parametri a: -1
Anna parametri b: -1
Anna parametri c: 1
Piirretään 2F1(-1.0, -1.0; 1.0; x).
Valinta:
```



Kuva 10: Hypergeometrisen funktion ${}_2F_1$ kuvaajia eri parametreilla.

Toteutuksen tarkkuutta voidaan arvioida tarkastelemalla identiteetin

$$ {}_2F_1(a, b+1; c+1; x) - {}_2F_1(a, b; c; x) = \frac{a(c-b)x}{c(c+1)}\, {}_2F_1(a+1, b+1; c+2; x) \quad (8) $$

virhettä eri parametreilla. Seuraava ohjelma piirtää kuvaajan (ks. kuva 11) virheen logaritmista $\lg(|\text{virhe}| + \epsilon)$, missä $\epsilon = 10^{-16}$ estää logaritmin ottamisen nollasta. Perimällä edellä olevan `HyperGeom`-luokan ohjelman koodia on saatu lyhyemmäksi.

Ohjelmaesimerkki 7.4

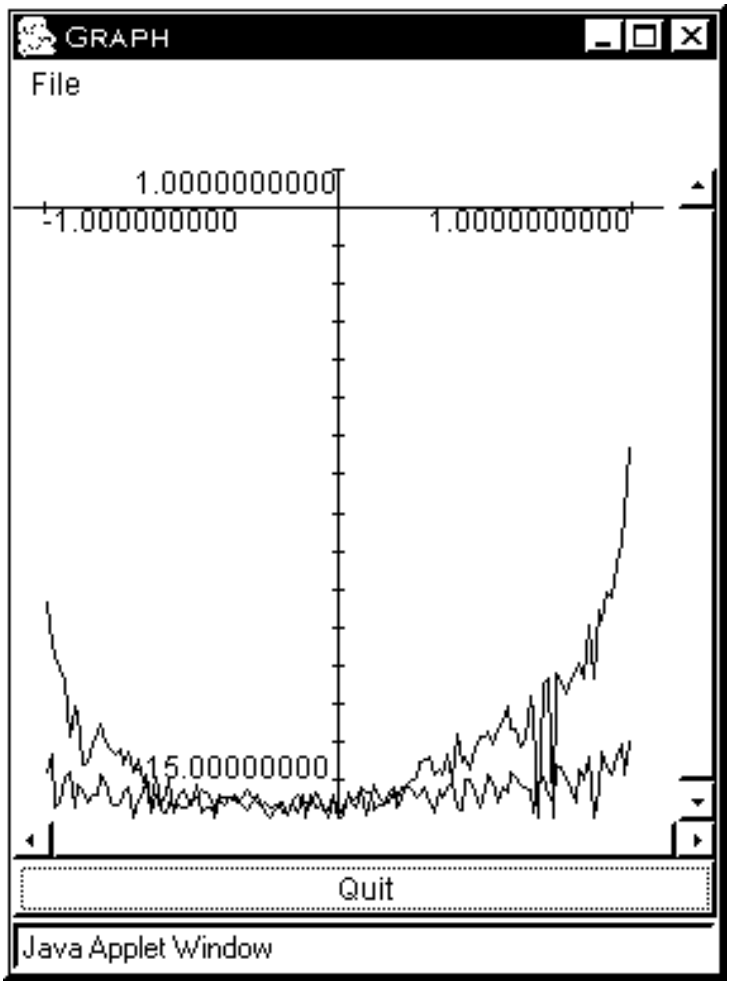
```
package examples;

// hypergeometrista funktiota 2F1 koskeva identiteetti

import java.awt.Color;
import uk.co.jscieng.Plottable;

public class HypGeomId extends HypGeom implements Plottable {
  final static double invLn10=1.0/Math.log(10);

  public void main() {
    showGraph(-1.1, 1.1, -16.0, 1.0, PTS, PTS);
    showText(55, 30);
    println("Testataan identiteetti\n"
      +"2F1(a, b+1; c+1; x) - 2F1(a, b; c; x)\n"
      +"= a(c-b)x/(c(c+1)) * 2F1(a+1, b+1; c+2; x)\n"
      +"piirtämällä virheen itseisarvon 10-kantainen logaritmi.\n\n"
      +"Piirtämisen jälkeen\n"
      +"valitse 1, jos haluat syöttää uudet arvot, tai\n"
      +"valitse 2, jos haluat tyhjentää ruudun.");

    int i=0;

    while (true) {
      a=param("a");
      b=param("b");
      c=param("c");

      functionPlot(this, -0.99, 0.99, colors[i++%N], 2);

      if (choose()==2) {
        clear();
      }
    }
  }

  public double f(double x) {
```

```
    return Math.log(1.0e-16+Math.abs(hypGeom(a, b+1.0, c+1.0, x)
      -hypGeom(a, b, c, x)
      -a*(c-b)*x/(c*(c+1.0))*hypGeom(a+1.0, b+1.0, c+2.0, x)))
      *invLn10;
  }
}
```

Ohjelman tulostus:

```
Testataan identiteetti
2F1(a, b+1; c+1; x) - 2F1(a, b; c; x)
= a(c-b)x/(c(c+1)) * 2F1(a+1, b+1; c+2; x)
piirtämällä virheen itseisarvon 10-kantainen logaritmi.

Piirtämisen jälkeen
valitse 1, jos haluat syöttää uudet arvot, tai
valitse 2, jos haluat tyhjentää ruudun.
Anna parametri a: 3
Anna parametri b: 2
Anna parametri c: 1
Valinta: 1
Anna parametri a: .2
Anna parametri b: .4
Anna parametri c: .2
Valinta:
```

Kuvassa y-akselilla luku 1.0 (ylhäällä) tarkoittaa virhettä $10^1 = 10$ ja -15.0 (alhaalla) virhettä 10^{-15}. Koska virhe on keskimäärin vain luokkaa $10^{-15} \ldots 10^{-12}$, voidaan toteutusta pitää onnistuneena.



Kuva 11: Identiteetin (8) virheen kuvaajia eri parametreilla.

8 Loppusanat

Esimerkkien laatiminen, testaus ja taustatiedon hankkiminen on ollut haastavaa ja mielenkiintoista. Paljon jäi vielä tekemättäkin. Työ jatkuu.

Tutkielmassa esitellyt lukuisat esimerkit toivottavasti osoittavat, että matemaattisten ongelmien ratkaisuun on mahdollista toteuttaa ohjelmia Java-kielelläkin. Heikkouksiakin toki on, mutta kielen alustariippumattomuus ja mahdollisuus suorittaa ohjelmia verkossa ovat ehdottomasti sen vahvuuksia muihin ohjelmointikieliin verrattuna. Internetissä kahlatessa sen myös huomaa: pienellä hakemisella erilaisia Java-kielellä toteutettuja matemaattisia luokkakokonaisuuksia alkaa löytyä.

Java-kieli ja sen standardikirjastot ovat suhteellisen nuoresta iästään johtuen vielä kehitysvaiheessa ja erityisesti matemaattisiin ominaisuuksiin on yleisesti kaivattu parannuksia. Tässäkin työssä käytettyä JAMA-kirjastoa on esimerkiksi esitetty Java-kielen lineaarialgebran peruskirjastoksi.

Tällä hetkellä Javan ja numeriikan ala tuntuu elävän hiljaiseloa. Ehkä kaikkialla odotetaan, mitä kielessä tapahtuu. Nyt ja tulevaisuudessa onkin mielenkiintoista seurata, miten kieli kehittyy ja löytääkö se lopulta paikkansa ”vanhojen konkareiden” joukosta, kun kyse on matemaattisesta ohjelmoinnista. Ennen sitä on kehitystä tapahduttava ja monia ennakkoluuloja murrettava.

Kirjallisuutta

[1] *Java™ printf package v1.6.* http://www.braju.com/ [1]

[2] *Geosoft - Software* http://geosoft.no/software/ [1]

[3] *JAMA : A Java Matrix Package.*
http://math.nist.gov/javanumerics/jama/ [1]

[4] *What is JeksParser?*
http://www.eteks.com/jeks/en/#WhatJeksParser [1]

[5] *The JSGL, a Scientific Graphics Library for Java.*
http://jscieng.co.uk/downloads.html#jsgl [1]

[6] *Numeerisia menetelmiä Java-kielellä.*
http://users.utu.fi/athein/gradu/ [1]

[7] Weisstein Eric W. *Eigen Decomposition Theorem.* From *MathWorld–*A Wolfram Web Resource. http://mathworld.wolfram.com/EigenDecompositionTheorem.html [1]

[8] Weisstein Eric W. *Hypergeometric Function.* From *MathWorld–*A Wolfram Web Resource. http://mathworld.wolfram.com/HypergeometricFunction.html [1]

[9] Weisstein Eric W. *Least Squares Fitting.* From *MathWorld–*A Wolfram Web Resource. http://mathworld.wolfram.com/LeastSquaresFitting.html [1]

[10] Abramowitz Milton, Stegun Irene A. *Handbook of Mathematical Functions, With Formulas, Graphs, and Mathematical Tables.* Dover Publications, New York, 1972.

[1] Vierailtu 21.11.2005.

[11] Estep Donald J. *Practical Analysis in One Variable*. Springer-Verlag, New York, 2002.

[12] Golub Gene H., Van Loan Charles F. *Matrix Computations*. The Johns Hopkins University Press, 1983.

[13] Haataja Juha, Heikonen Jussi, Leino Yrjö, Rahola Jussi, Ruokolainen Juha, Savolainen Ville. *Numeeriset menetelmät käytännössä*. CSC - Tieteellinen laskenta Oy, 1999.

[14] Hardle W., Chambers J., Eddy W. *Numerical Analysis for Statisticians*. Springer-Verlag, New York, 1999.

[15] Leipälä Timo. *Numeerinen analyysi*. Luentomoniste, Turun yliopisto, matematiikan laitos, 2002.

[16] Lengyel Eric. *Mathematics for 3D Game Programming and Computer Graphics*. Charles River Media, 2003.

[17] Metsänkylä Tauno. *Matriisilaskenta*. Luentomoniste, Turun yliopisto, matematiikan laitos, 2000.

[18] Press William H., Teukolsky Saul A., Vetterling William T., Flannery Brian P. *Numerial Recipes in C: The Art of Scientific Computing*. Cambridge University Press, 2002.

[19] Quarteroni Alfio, Sacco Riccardo, Saleri Fausto. *Numerical Mathematics*. Springer-Verlag, New York, 2000.

[20] Spiegel Murray R., Liu John. *Mathematical Handbook of Formulas and Tables*. McCraw-Hill, Singapore, 1999.

[21] Ueberhuber Christoph W. *Numerical Computation 1: Methods, Software and Analysis*. Springer-Verlag, Berlin Heidelberg, 1997.

[22] Ueberhuber Christoph W. *Numerical Computation 2: Methods, Software and Analysis*. Springer-Verlag, Berlin Heidelberg, 1997.

[23] Vuorinen Matti. *Numeeriset menetelmät ja C-kieli*. Luentokurssi, Helsingin yliopisto.

A examples-kirjasto

Kaikki esimerkkiohjelmat kuuluvat examples-kirjastoon.

A.1 Base

Esimerkkiohjelmat periytyvät Base-luokasta, johon on koottu usein tarvittuja metodeja. Base

```
package examples;

// kaikkien esimerkkiluokkien yläluokka,
// sisältää usein käytettyjä rutiineja ja attribuutteja

import Jama.*;
import java.awt.Color;
import matutl.*;
import uk.co.jscieng.SciGraph;

public abstract class Base extends SciGraph {
  public static final int DECIM=3,
    PTS=512;

  //piirtämisessä käytetyt värit
  public static final Color[] colors=new Color[] {
    Color.blue, Color.cyan, Color.darkGray, Color.gray, Color.green,
    Color.magenta, Color.orange, Color.pink, Color.red, Color.yellow
  };

  public abstract void main();

  public int valinta() {
    println("Valitse 1, jos haluat syöttää luvut itse.");
    println("Valitse 2, jos haluat ohjelman generoivan "
```

```
    +"satunnaiset luvut.");

  return valinta2();
}

public int valinta2() {
  int v=0;

  do {
    print("Valinta: ");

    try {
      v=Integer.parseInt(readln());
    } catch (Exception e) {}
  } while (v!=1 && v!=2);

  return v;
}

public int lukumaara(String minka, int min) {
  return lukumaara(minka, min, Integer.MAX_VALUE);
}

// pyytää kokonaisluvun (lukumäärän esim. matriisin riveille)
// väliltä min...max
public int lukumaara(String minka, int min, int max) {
  int r=0;

  do {
    print("Anna "+minka+" lukumäärä: ");

    try {
      r=Integer.parseInt(readln());
    } catch (Exception e) {}
  } while (r<min || r>max);

  return r;
}

// luo m x n mariisin
public Matrix luoMatriisi(String nimi, int m, int n) {
  Matrix A=new Matrix(m, n);
  boolean uudestaan=false;
```

```
  for (int i=0; i<m; i++) {
    for (int j=0; j<n; j++) {
      do {
        try {
          print("Anna "+nimi+"("+(i+1)+","+(j+1)+"): ");
          A.set(i, j, Double.valueOf(readln()).doubleValue());
          uudestaan=false;
        } catch (Exception e) {
          uudestaan=true;
        }
      } while (uudestaan);
    }
  }

  return A;
}

// luo n alkion vektorin
public Vec luoVektori(String nimi, int n) {
  Vec A=new Vec(n);
  boolean uudestaan=false;

  for (int i=0; i<n; i++) {
    do {
      try {
        print("Anna "+nimi+"("+(i+1)+"): ");
        A.set(i, Double.valueOf(readln()).doubleValue());
        uudestaan=false;
      } catch (Exception e) {
        uudestaan=true;
      }
    } while (uudestaan);
  }

  return A;
}

// tulostettavien desimaalien lukumäärä
public int desimaalit() {
  print("Matriisialkiot esitetään oletusarvoisesti "
    +"kolmella desimaalilla.\nHyväksy oletus antamalla "
    +"tyhjä syöte tai\nanna uusi desimaalien määrä: ");
```

```
    String input=readln();

    if (input.equals("")) {
      if ("0123456789".indexOf(input)<0) {
        println("Desimaalien määrä: "+DECIM);
      }

      return DECIM;
    }

    return Integer.parseInt(input);
  }

  public void tulosta(String nimi, Matrix M, int des) {
    println(nimi+" =\n"+MatrixIO.matrixToString(M, des));
  }

  public void tulosta(String nimi, Matrix M, int kok, int des) {
    println(nimi+" =\n"+MatrixIO.matrixToString(M, kok, des));
  }

  public void tulosta(Vec V, int des) {
    println(MatrixIO.vecToString(V, des));
  }

  public void tulosta(String nimi, Vec V, int des) {
    println(nimi+" = "+MatrixIO.vecToString(V, des));
  }

  // piirtää pisteen
  public void plotMark(double x, double y, double dx, double dy) {
    line(x, y, x, y);
    line(x-dx, y+dy, x+dx, y+dy);
    line(x+dx, y-dy);
    line(x-dx, y-dy);
    line(x-dx, y+dy);
  }
}
```

B matutl-kirjasto

Itse tehtyjä matemaattisia apuluokkia.

B.1 Vec

Vektoriluokka, jonka toteutus perustuu JAMA-kirjaston `Matrix`-luokkaan.

```
package matutl;

// Luokka, joka mallintaa vektorin.
// Toteutus ja metodit pohjautuvat Jama-kirjaston Matrix-luokkaan.
// Kaikkia Matrix-luokan metodeja ei ole toteutettu.

import Jama.Matrix;

public class Vec implements Cloneable, java.io.Serializable {
  private Matrix mat;
  private int m, n;
  private boolean isRow;

  public Vec(int n) {
    mat=new Matrix(1, n);
    m=1;
    this.n=n;
    isRow=true;
  }

  public Vec(int n, double s) {
    mat=new Matrix(1, n, s);
    m=1;
    this.n=n;
    isRow=true;
  }
```

```
public Vec(double[][] A) {
  mat=new Matrix(A);
  m=mat.getRowDimension();
  n=mat.getColumnDimension();

  if (m!=1 && n!=1) {
    throw new IllegalArgumentException(
      "Row or column dimension must be equal to one.");
  }

  isRow=(m==1);
}

public Vec(double[] A) {
  mat=new Matrix(A, 1);
  m=1;
  n=mat.getColumnDimension();
  isRow=true;
}

public Vec(Matrix mat) {
  this.mat=mat;
  m=mat.getRowDimension();
  n=mat.getColumnDimension();
  isRow=(m==1);
}

public double[] getColumnPackedCopy() {
  return mat.getColumnPackedCopy();
}

public double[] getRowPackedCopy() {
  return mat.getRowPackedCopy();
}

public int getRowDimension() {
  return m;
}

public int getColumnDimension() {
  return n;
}
```

```
public boolean isRow() {
  return isRow;
}

public double get(int i) {
  return isRow ? mat.get(0, i) : mat.get(i, 0);
}

public void set(int i, double s) {
  if (isRow) {
    mat.set(0, i, s);
  } else {
    mat.set(i, 0, s);
  }
}

public void setVec(int i0, int i1, Vec X) {
  if (isRow) {
    mat.setMatrix(0, 0, i0, i1, X.getMatrix());
  } else {
    mat.setMatrix(i0, i1, 0, 0, X.getMatrix());
  }
}

public void setVec(int[] c, Vec X) {
  if (isRow) {
    mat.setMatrix(0, 0, c, X.getMatrix());
  } else {
    mat.setMatrix(c, 0, 0, X.getMatrix());
  }
}

public Matrix getMatrix() {
  return mat;
}

public double minValue() {
  double min=mat.get(0, 0);

  for (int i=0; i<m; i++) {
    for (int j=0; j<n; j++) {
      min=Math.min(min, mat.get(i, j));
```

```
    }
  }

  return min;
}

public double maxValue() {
  double max=mat.get(0, 0);

  for (int i=0; i<m; i++) {
    for (int j=0; j<n; j++) {
      max=Math.max(max, mat.get(i, j));
    }
  }

  return max;
}

public double norm(int p) {
  if (p<1) {
    throw new IllegalArgumentException(
      "Argument must be positive");
  } else if (p==1) {
    return mat.norm1();
  }

  double norm=0.0;

  for (int i=0; i<m; i++) {
    for (int j=0; j<n; j++) {
      norm=Maths.nRoot(norm, Math.abs(mat.get(i, j)), p);
    }
  }

  return norm;
}

public double normInf() {
  return mat.normInf();
}

public Vec plus(Vec B) {
  return plus(B.getMatrix());
```

```
}

public Vec plus(Matrix B) {
  return new Vec(mat.plus(B));
}

public Vec minus(Vec B) {
  return minus(B.getMatrix());
}

public Vec minus(Matrix B) {
  return new Vec(mat.minus(B));
}

public Vec times(double s) {
  return new Vec(mat.times(s));
}

public Vec times(Matrix B) {
  int k=B.getRowDimension();

  if (k==n) {
    return new Vec(mat.times(B));
  } else if (k==m) {
    return new Vec(mat.times(B.transpose()));
  }

  throw new IllegalArgumentException(
    "Vector dimensions must agree.");
}

public double dotProd(Vec B) {
  return dotProd(this, B);
}

// vektorien A ja B pistetulo
public static double dotProd(Vec A, Vec B) {
  int na=A.getColumnDimension(),
    nb=B.getColumnDimension();

  if (na==nb) {
    double dp=0.0;
```

```
      for (int i=0; i<na; i++) {
        dp+=A.get(i)*B.get(i);
      }

      return dp;
    } else {
      throw new IllegalArgumentException(
        "Argument Vecs differ in length");
    }
  }

  // vektorien A ja B ristitulo
  public static Vec crossProd(Vec A, Vec B) {
    if (A.getColumnDimension()==3 && B.getColumnDimension()==3) {
      double ax=A.get(0),
        ay=A.get(1),
        az=A.get(2),
        bx=B.get(0),
        by=B.get(1),
        bz=B.get(2);
      double[] cp={ay*bz-az*by, az*bx-ax*bz, ax*by-ay*bx};

      return new Vec(cp);
    } else {
      throw new IllegalArgumentException(
        "Arguments must be Vecs of three elements");
    }
  }

  public Vec crossProd(Vec B) {
    return crossProd(this, B);
  }

  // skalaarien A, B ja C skalaarikolmitulo
  public static double scalar3Prod(Vec A, Vec B, Vec C) {
    return dotProd(A, crossProd(B, C));
  }

  public double scalar3Prod(Vec B, Vec C) {
    return scalar3Prod(this, B, C);
  }
}
```

B.2 Maths

Rutiineja mm. numeeriseen integrointiin ja derivointiin, interpolointiin sekä yhtälönratkaisemiseen.

```
package matutl;

import Jama.*;
import uk.co.jscieng.Plottable;

public class Maths {

  /**
   * Raise a double to a positive integer power.
   * Fast version of Math.pow.
   * @param x number to be taken to a power.
   * @param n power to take x to. 0 <= n <= Integer.MAX_VALUE
   * Negative numbers will be treated as unsigned positives.
   * @return x to the power n
   * @author Patricia Shanahan pats@acm.org
   */
  public static double power(double x, int n) {
    int bitMask=n;
    double evenPower=x,
      result;

    if ((bitMask & 1)!=0)
      result=x;
    else
      result=1;

    bitMask >>>= 1;

    while (bitMask!=0) {
      evenPower *= evenPower;

      if ((bitMask & 1) != 0)
        result *= evenPower;

      bitMask >>>= 1;
    } // end while

    return result;
  } // end power
```

```
// (a^n + b^n)^(1/n) ilman yli/alivuotoa.
public static double nRoot(double a, double b, int n) {
  if (Math.abs(a)>Math.abs(b))
    return Math.abs(a)*Math.pow(1.0+power(b/a, n), 1.0/n);

  return Math.abs(b)*Math.pow(1.0+power(a/b, n), 1.0/n);
}

public static boolean isSymmetric(Matrix mat) {
  int n=mat.getRowDimension();

  if (n!=mat.getColumnDimension())
    throw new IllegalArgumentException("Matrix non-diagonal");

  for (int i=0; i<n; i++)
    for (int j=i+1; j<n; j++)
      if (Math.abs(mat.get(i, j)-mat.get(j, i))>1.0e-10)
        return false;

  return true;
}

public static Matrix diagonal(int n, double diag) {
  Matrix dm=new Matrix(n, n, 0.0);

  for (int i=0; i<n; i++)
    dm.set(i, i, diag);

  return dm;
}

public static Matrix triDiagonal(
    int n, double lower, double diag, double upper) {
  Matrix tdm=diagonal(n, diag);

  tdm.set(0, 1, upper);

  for (int i=1; i<n-1; i++) {
    tdm.set(i, i-1, lower);
    tdm.set(i, i+1, upper);
  }
```

```
  tdm.set(n-1, n-2, lower);

  return tdm;

}

// Matriisin A pseudoinverssi,
// määritetään singulaariarvohajotelman avulla.
public static Matrix pseudoinverse(Matrix A, boolean edit) {
  SingularValueDecomposition SVD=A.svd();
  Matrix U=SVD.getU(),
    S=SVD.getS(),
    V=SVD.getV();
  double[] sv=SVD.getSingularValues();
  double eps=sv[0]*1.0e-6;

  for (int i=0; i<sv.length; i++) {
    if (sv[i]>0.0) {
      if (edit && sv[i]<eps)
        S.set(i, i, 0.0);
      else
        S.set(i, i, 1.0/sv[i]);
    }
  }

  return V.times(S).times(U.transpose());
}

public static Matrix SVDsolve(Matrix A, Vec B, boolean edit) {
  return SVDsolve(A, B.getMatrix().transpose(), edit);
}

// Yhtälöryhmän ratkaiseminen singulaariarvohajotelman avulla.
// Mahdollisuus singulaariarvojen editointiin.
public static Matrix SVDsolve(Matrix A, Matrix B, boolean edit) {
  if (B.getRowDimension()!=A.getRowDimension())
    throw new IllegalArgumentException(
      "Matrix row dimensions must agree.");

  return pseudoinverse(A, edit).times(B);
}

public static double getMin(Matrix mat, int col) {
```

```
  double min=mat.get(0, col);

  for (int i=1; i<mat.getRowDimension(); i++)
    min=Math.min(min, mat.get(i, col));

  return min;
}

public static double getMax(Matrix mat, int col) {
  double max=mat.get(0, col);

  for (int i=1; i<mat.getRowDimension(); i++)
    max=Math.max(max, mat.get(i, col));

  return max;
}

// Palauttaa taulukon [a b r], missä
// a ja b ovat pienimmän neliösumman suoran y=a+bx kertoimet ja
// r korrelaatiokerroin.
public static double[] leastSquaresFitting(
    double[] x, double[] y) {
  int n=x.length;
  double xAv=0.0, yAv=0.0, a, b, r, ssxx=0.0, ssyy=0.0, ssxy=0.0;

  for (int i=0; i<n; i++) {
    xAv+=x[i];
    yAv+=y[i];
  }

  xAv/=n;
  yAv/=n;

  for (int i=0; i<n; i++) {
    ssxx+=power(x[i]-xAv, 2);
    ssyy+=power(y[i]-yAv, 2);
    ssxy+=(x[i]-xAv)*(y[i]-yAv);
  }

  b=ssxy/ssxx;
  a=yAv-b*xAv;
  r=ssxy/Math.sqrt(ssxx*ssyy);
```

```
  return new double[] {a, b, r};
}

// puolisuunnikassääntö
public static double trapezoid(double[] fx, double h) {
  int n=fx.length-1;
  double sum=(fx[0]+fx[n])/2;

  for (int i=1; i<n; i++)
    sum+=fx[i];

  return h*sum;
}

// Simpsonin sääntö
public static double simpson(double[] fx, double h) {
  int n=fx.length-1;

  if (n%2!=0)
    throw new IllegalArgumentException("Array length even");

  double sum=fx[0]+fx[n];

  for (int i=1; i<n; i+=2)
    sum+=4*fx[i];

  for (int i=2; i<n; i+=2)
    sum+=2*fx[i];

  return h/3*sum;
}

// Original algorithm: Numerical Recipes.
// Given the lower and upper limits of integration x1 and x2
// and given n, this routine returns arrays x[0..n-1] and
// w[0..n-1] of length n containing the abscissas and weights of
// the Gauss-Legendre n point quadrature formula.
public static void gauLeg(double x1, double x2,
    double[] x, double[] w, int n) {
  double xm=0.5*(x2+x1),
    xl=0.5*(x2-x1),
    EPS=3.0e-11;              // EPS is the relative precision.
```

```
  // Loop over the desired roots. The roots are symmetric
  // in the interval, so we only have to find half of them.
  for (int i=1, m=(n+1)/2; i<=m; i++) {

    // Starting with the approximation below to the ith root,
    // we enter the main loop of refinement by Newton's method.
    double z=Math.cos(Math.PI*(i-0.25)/(n+0.5)),
      z1, pp;

    do {
      double p1=1.0,
        p2=0.0;

      // Loop up the recurrence relation to get the
      // Legendre polynomial evaluated at z.
      for (int j=1; j<=n; j++) {
        double p3=p2;
        p2=p1;
        p1=((2.0*j-1.0)*z*p2-(j-1.0)*p3)/j;
      }

      // p1 is now the desired Legendre polynomial. We next
      // compute pp, its derivative, by a standard relation
      // involving also p2, the polynomial of one lower order.
      pp=n*(z*p1-p2)/(z*z-1.0);
      z1=z;
      z=z1-p1/pp;   // Newton's method.
    } while (Math.abs(z-z1)>EPS);

    x[i]=xm-xl*z;       // Scale the root to the desired interval,
    x[n+1-i]=xm+xl*z;   // and put in its symmetric counterpart.
    w[i]=2.0*xl/((1.0-z*z)*pp*pp);  // Compute the weight and
    w[n+1-i]=w[i];                  // its symmetric counterpart.
  }
}

public static double gauLeg(double x1, double x2,
    double[] x, double[] w, int n, Plottable p) {
  gauLeg(x1, x2, x, w, n);
  double I=0.0;

  for (int i=1; i<=n; i++) {
    I+=(w[i]*p.f(x[i]));   // I=sum(w_i*f(x_i))
```

```
  }

  return I;
}

// routine adapted from Numerical Recipes in C
// converting to java and modification 19.09.2004 by Ari Heino
public static double[] polint(
    double[] xb, double[] yb, int n, double x) {
  int ns=1;
  double den, dif, dift, ho, hp, w, y, dy=0;
  double[] xa=new double[n+1],
    ya=new double[n+1],
    c=new double[n+1],
    d=new double[n+1];

  // input arrays are assumed to be unit-offset.
  System.arraycopy(xb, 0, xa, 1, n);
  System.arraycopy(yb, 0, ya, 1, n);

  dif=Math.abs(x-xa[1]);

  // find the index ns of the closest table entry
  for (int i=1; i<=n; i++) {
    if ((dift=Math.abs(x-xa[i]))<dif) {
      ns=i;
      dif=dift;
    }

    //initialize the tableau of c's and d's
    c[i]=ya[i];
    d[i]=ya[i];
  }

  y=ya[ns--]; // initial approximation to y

  // for each column of the tableau,
  // loop over the current c's and d's and update them
  for (int m=1; m<n; m++) {
    for (int i=1; i<=n-m; i++) {
      ho=xa[i]-x;
      hp=xa[i+m]-x;
      w=c[i+1]-d[i];
```

```
      if ((den=ho-hp)==0.0)
        throw new IllegalArgumentException(
          "Two input xa's are (to within roundoff) identical.");

      den=w/den;
      c[i]=ho*den;  // c's and d's
      d[i]=hp*den;  // are updated
    }

    // After each column in the tableau is completed,
    // we decide which correction, c or d,
    // we want to add to our accumulating value of y,
    // i.e., which path to take through the tableau
    // -forking up or down. We do this in such a way as to take
    // the most "straightline" route through the tableau
    // to its apex, updating ns accordingly to keep track of
    // where we are. This route keeps the partial approximations
    // centered (insofar as possible)on the target x.
    // The last dy added is thus the error indication.
    y+=(dy=(2*ns<(n-m) ? c[ns+1] : d[ns--]));
  }

  double[] ans={y, dy};

  return ans;
}

// Derivaatan numeerinen arvio. Algoritmi perustuu viiden pisteen
// kaavaan Abramowitz-Stegun 25.3.6. Syötteenä annetaan funktion
// arvoja f(a+ih), i=0, ..., n-1, askel h sekä pisteiden määrä n.
public static double[] numDer(double[] arr1, double h, int n)
    throws Exception {
  if (n<5) {
    throw new
      Exception("Too few points for numerical differentiation.");
  }

  double[] arr2=new double[n];

  for (int i=0, p; i<n; i++) {
    if (i<=2) {
      p=i-2;
```

```
      } else if (i>2 && i<=n-3) {
        p=0;
      } else {
        p=i-n+3;
      }

      double p2=p*p,
        p3=p*p2,
        a=(2*p3-3*p2-p+1)/12.0,
        b=(4*p3-3*p2-8*p+4)/6.0,
        c=(2*p3-5*p)/2.0,
        d=(4*p3+3*p2-8*p-4)/6.0,
        e=(2*p3+3*p2-p-1)/12.0;

      arr2[i]=((a*arr1[i-p-2])-(b*arr1[i-p-1])
        +(c*arr1[i-p])-(d*arr1[i-p+1])+(e*arr1[i-p+2]))/h;
    }

    return arr2;
  }
}
```

B.3 MatrixIO

Matriisien ja vektoreiden tulostusrutiineja.

```
package matutl;

import com.braju.format.*;
import Jama.Matrix;

public class MatrixIO {
  public static String vecToString(Vec vec, int fpart) {
    return matrixToString(vec.getMatrix(), 5, fpart);
  }

  public static String matrixToString(Matrix mat) {
    return matrixToString(mat, 5, 3);
  }

  public static String matrixToString(Matrix mat, int fpart) {
    return matrixToString(mat, 5, fpart);
  }
```

```
  // matriisin muuttaminen merkkijonoksi
  // ipart=kokonaisosan pituus
  // fpart=desimaalien lukumäärä
  public static String matrixToString(
      Matrix mat, int ipart, int fpart) {
    int m=mat.getRowDimension(),
      n=mat.getColumnDimension();
    String format="%"+(ipart+fpart+1)+"."+fpart+"f";
    StringBuffer s=new StringBuffer ("");

    for (int i=0; i<m; i++) {
      for (int j=0; j<n; j++)
        s.append(Format.sprintf(
          format, new Parameters(mat.get(i, j))));

      s.append("\n");
    }

    return s.toString();
  }
}
```