ATK IV
NUMEERINEN OHJELMOINTI
76316S

Dos. Mikko Saarela
Teoreettinen fysiikka
Fysikaalisten tieteiden laitos
Oulun yliopisto
PL3000, 90401 Oulu

syksy 2004

Kurssikirja:
W.H.Press, B.P.Flannery, S.A.Teukolsky, W.T.Vetterling,
Numerical Recipes

Sisältö

Luku 1

Työkalut

1.1 Linux käyttöjärjestelmän alkeet

Opetellaan harjoituksissa

1.2 Fortran ohjelmointikieli

Harjoituksissa jaetaan Fortran moniste. Kurssilla käytetään GNU:n ilmaisversiota, joka noudattaa Fortran 77 standardia pääosin.

1.2.1 make-tiedoston käyttö

Harjoituksissa jaetaan make-tiedoston käyttöopas. Sen avulla voidaan hiukan yksinkertaistaa fortran ohjelmien suoritusta.

1.3 Numeerisia kirjastoja

- Numerical Recipes
  - käsittelee numeerisia että tilastollisia algoritmeja
  - sekä FORTRAN-, C- että PASCAL- kieliset versiot saatavilla
  - avustusjärjestelmä: ei ole saatavissa tällä hetkellä
  - kirjasto saatavissa lähdekielisenä
  - esimerkit saatavissa lähdekielisenä

1.4 Muita numeerisia kirjastoja

- NAG
  - sekä numeerisia että tilastollisia algoritmeja
  - avustusjärjestelmä: naghelp
  - kirjasto saatavissa binäärimuodossa, ei lähdekielisenä

  - esimerkit saatavissa lähdekielisenä

- IMSL
  - sekä numeerisia että tilastollisia algoritmeja ja erikoisfunktioita
  - avustusjärjestelmä: idf
  - kirjasto saatavissa binäärimuodossa, ei lähdekielisenä
  - esimerkit saatavissa lähdekielisenä

1.5 Kuvien tuottaminen

Harjoituksissa jaetaan gnuplot-ohjelmiston käyttöopas

1.6 Matemaattisen tekstin tuottaminen

Matemaattisen tekstin tuottamiseen käytetään Latex ladontakieltä. Opas kielen saloihin jaetaan harjoituksissa.

1.7 Kirjastojen käyttö

Kirjasto on tiedosto (tavallisesti tyyppiä `.a`, archive (UNIX) tai tyyppiä `.lib` (WIN)), johon on koottu **useita käännettyjä aliohjelmia** eli .o (UNIX) tai .obj (WIN) tyyppisiä objektitiedostoja. Aliohjelmakirjastoja käytettäessä on ensin kirjoitettava pääohjelma, jossa kutsutaan tehtävän ratkaisussa tarvittavia kirjastoaliohjelmia sekä mahdolliset omat funktio- tai subroutine- aliohjelmat.

UNIX-ympäristöissä kirjastot liitetään omiin ohjelmiin siten, että ensin nämä omat ohjelmat on käännetään konekielisiksi objektitiedostoiksi,
`% g77 -c main.f oma.f ali.f`
Sen jälkeen kirjastot linkitetään aliohjelmakirjastojen kanssa suoritettavaksi ohjelmatiedostoksi
`% g77 -o koe.out main.f oma.f ali.f -lnrf`
Optiolla `-l nimi` haetaan automaattisesti kirjastoa `libnimi.a` hakemistoista
`/lib, /usr/lib, /usr/local/lib`

Jos kirjasto ei ole missään näistä hakemistoista, niin optiolla `-L hakemisto` voidaan välittää kääntäjälle `g77` tieto siitä, mistä hakemistoista oletushakemistojen lisäksi optiolla `-l` tulee hakea kirjastoja; esimerkiksi
`% g77 -o koe.out main.f oma.f ali.f`
`-L/home/teof/atk4/lib -lnrf`
WINDOWS-ympäristössä voidaan menetellä vastaavasti, joskin komentojen nimet poikkeavat UNIX-nimistä. Toisaalta nykyään nykyään käytetään useimmiten ns. integroituja kehitystyökaluja (esim. Visual Studio), jotka lähes automaattisesti toteuttavat edellä mainitut työvaiheet.

Esimerkki:
Satunnaislukugeneraattori
Numerical Recipes kirjastossa on useita satunnaislukugeneraattoreita, esimerkiksi `ran0(iseed)`, missä `iseed` on jokin negatiivinen kokonaisluku, siemen. Tehdään pääohjelma, joka simuloi nopanheittoa.

```
c Satunnaislukugeneraattorin testausohjelma satu.f
c
      real r(100), ran0
      integer iseed
      print*, 'Anna siemen ja lukujen lukumaara:'
      read*, iseed, n
      do 10 i = 1,n
          r(i) = ran0(iseed)
10    continue
      print*, (int(6*r(i)+1), i=1,n)
      end
```

Ohjelma käännetään ja linkitetään Numerical Recipes kirjaston kanssa sekä suoritetaan seuraavasti:

```
% g77 satu.f -lnrf
% a.out
Anna siemen ja lukujen lukumaara:
 1 12
2     1     1     6     4     2
4     3     5     6     2     3
```

Luku 2

Numeerisia menetelmiä

2.1 Funktioista ja palautuskaavoista

Tässä luvussa esitetään muutamia menetelmiä, joilla lasketaan arvoja sellaisille funktioille, joita ei voida tai ei ole tarkoituksenmukaista esittää alkeisfunktioiden avulla.

2.1.1 Sarjakehitelmistä

Funktiota voidaan esittää erilaisten sarjakehitelmien avulla. Tyypillisiä esimerkkejä ovat potenssisajat

$$f(x) = \sum_{k=0}^{\infty} a_k (x - x_0)^k \ .$$

Jos kyseessä on Taylorin sarja, niin kertoimet a_k ovat funktion derivaattojen avulla esitettävissä

$$a_k = \frac{1}{k!} \frac{d^k f(x)}{dx^k}\Big|_{x=x_0}$$

Esimerkiksi

$$\cos(x) = \sum_{k=0}^{\infty} (-1)^k \frac{x^{2k}}{(2k)!}$$

ja Besselin funktion sarjakehitelmä on

$$J_n(x) = \left(\frac{x}{2}\right)^n \sum_{k=0}^{\infty} \frac{(-\frac{1}{4}x^2)^k}{k!(k+n)!}$$

Numeerisissa laskuissa ongelmaksi tulee se, että osoittaja ja nimittäjä erikseen laskettuina kasvavat helposti yli lukualueen rajojen. Sen vuoksi pitäisi pyrkiä laskemaan suoraan osoittajan ja nimittäjän suhdetta.
Esimerkiksi $\cos(x)$ arvon laskeminen sarjakehitelmästä tapahtuu kätevimmin seuraavasta muodosta

$$\cos(x) = 1 - \frac{x^2}{1 \cdot 2} - \frac{x^2}{3 \cdot 4}\left(\frac{-x^2}{2!}\right) - \frac{x^2}{5 \cdot 6}\left(\frac{x^4}{4!}\right) - \dots$$

Suluissa on aina edellinen termi kehitelmästä.

Yleisemmin kirjoitettuna määritellään astetta $2N$ oleva polynomiapproksimaatio.

$$\begin{aligned}\cos^{(N)}(x) &= \sum_{k=0}^{N}(-1)^k\frac{x^{2k}}{(2k)!}\\ &= \cos^{(N-1)}(x)+(-1)^N\frac{x^{2N}}{(2N)!}\\ &= \cos^{(N-1)}(x)+(-1)\frac{x^2}{(2N)(2N-1)}\\ &\times \left[(-1)^{N-1}\frac{x^{2N-2}}{(2N-2)!}\right]\end{aligned}$$

Ohjelmana algoritmi näyttäisi seuraavalta

```
   term = 1
   sum  = 1
   do 1 i = 2, n, 2
      term = -x*x/(i*(i-1))*term
      sum  = sum + term
1  continue
```

Tällä algoritmilla vältetään suurien kertomien ja potenssien aiheuttamia lukualueen ylivuotoja.

2.1.2 Polynomeista

Jos funktio esitetään polynomina, jonka kertoimet on taulukoitu, niin suositeltava algoritmi on *Hornerin*-kaava.

$$\begin{aligned}P(x) &= a_1x^{n-1}+a_2x^{n-2}+a_3x^{n-3}+\ldots+a_{n-1}x+a_n\\ &= a_n+x(a_{n-1}+x(a_{n-2}+\ldots+x(a_2+xa_1))\ldots)\end{aligned}$$

Jälkimmäinen muoto on esitettävissä yksinkertaisen palautuskaavan muodossa

$$\begin{aligned}P_i(x) &= a_i+xP_{i-1}(x), \quad \text{kun} \quad i=2,\ldots,n\\ P_1(x) &= a_1\end{aligned}$$

Ohjelmana Horner'in algoritmi, näyttäisi seuraavalta

```
     p=a(1)
     do 1 i = 2, n
        p = p*x + a(i)
1    continue
```

Polynomin derivaatta on myös helposti laskettavissa palautuskaavasta

$$\begin{aligned}P_i'(x) &= P_{i-1}(x)+xP_{i-1}'(x), \quad \text{kun} \quad i=2,\ldots,n\\ P_1'(x) &= 0\ ,\end{aligned}$$

joten ohjelma, joka laskee myös derivaatan saadaan pienellä lisäyksellä edellisestä ohjelmasta.

```
      dp = 0.
      p=a(1)
      do 1 i = 2, n
         dp = p + x*dp
         p = p*x + a(i)
1     continue
```

2.1.3 Palautuskaavoista

Erityisesti erikoisfunktioiden arvoja kannattaa laskea palautuskaavojen avulla varsinkin silloin, kun tarvitaan eri kertalukua olevia funktioita.

Tässä muutamia esimerkkejä palautuskaavoista

- Legendren polynomi

$$\begin{aligned}(n+1)P_{n+1}(x) &= (2n+1)xP_n(x) - nP_{n-1}(x)\\ P_0(x) &= 1\\ P_1(x) &= x\end{aligned}$$

- Besselin funktio

$$J_{n+1}(x) = \frac{2n}{x}J_n(x) - J_{n-1}(x)\ .$$

  Jos $J_0(x)$ ja $J_1(x)$ tunnetaan jossakin pisteessä, niin korkeamman kertaluvun termit voidaan laskea palautuskaavasta.

- Moninkertaisen kulman kosini ja sini, kun $n = 2, 3, \ldots$

$$\begin{aligned}\cos(n\theta) &= 2\cos(\theta)\cos((n-1)\theta) - \cos((n-2)\theta)\\ \sin(n\theta) &= 2\cos(\theta)\sin((n-1)\theta) - \sin((n-2)\theta)\end{aligned}$$

Numeerinen ongelma palautuskaavoissa on tarkkuuden säilyminen, kun kertaluku kasvaa suureksi. Jos tarkkuus ei säily riittävänä, niin silloin täytyy palautuskaavaa käyttää käänteisessä järjestyksessä ja lähteä liikkeelle termistä, jolla on korkein kertaluku. Sen, miten päin esimerkiksi palautuskaava

$$f_{n+1}(x) = \alpha(x)f_n(x) + \beta(x)f_{n-1}(x)$$

suppenee saa testattua seuraavasti:

1. Aloitetaan arvauksella $f_0(x) = 0$ ja $f_1(x) = 1$ ja lasketaan palautuskaavasta termit $n = 2, \ldots, 20$.
2. Aloitetaan toisella arvauksella $\bar{f}_0(x) = 1$ ja $\bar{f}_1(x) = 0$ ja lasketaan palautuskaavasta termit $n = 2, \ldots, 20$.

Lasketaan suure

$$\Delta_n(x) = f_n(x) - \bar{f}_n(x)$$

- Jos $|\Delta_n(x)| \sim 1$, kun $n = 2, \cdots, 20$ niin palautuskaava suppenee, kun n kasvaa.
- Jos $|\Delta_n(x)|$ kasvaa hitaasti, kun $n \to 20$ niin palautuskaava voi suppeta, kun n kasvaa.
- Jos $|\Delta_n(x)|$ kasvaa nopeasti, kun $n \to 20$ niin palautuskaava hajaantuu, kun n kasvaa. Tällöin palautuskaavaa täytyy käyttää käänteisessä järjestyksessä siten, että n pienenee.

2.1.4 Funktion kehittäminen Chebyshev'in polynomien avulla

Esimerkkinä palautuskaavojen ja summien yhteiskäytöstä käsitellään tapaus, jossa funktio $f(x)$ voidaan kehittää Chebysev'in polynomien $T_n(x)$ avulla välillä $x \epsilon [-1, 1]$ seuraavasti

$$f(x) \approx -\frac{1}{2}c_0 + \sum_{k=0}^{N-1} c_k T_k(x)$$

Chebysev'in polynomin ominaisuuksia:

- Esitys alkeisfunktioiden avulla

$$\begin{aligned} T_n(x) &= \cos(n \arccos(x)), \quad \text{kun} \quad x \epsilon [-1, 1] \\ T_0(x) &= 1 \\ T_1(x) &= x \\ T_2(x) &= 2x^2 - 1 \end{aligned}$$

- Palautuskaava

$$T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x)$$

  Palautuskaava on sama kuin moninkertaisen kulman cosinille.

- Ortogonaalisuus

$$\int_{-1}^{1} \frac{T_i(x)T_j(x)}{\sqrt{1-x^2}} dx = \begin{cases} 0, & \text{kun } i \neq j \\ \frac{\pi}{2}, & \text{kun } i = j \neq 0 \\ \pi, & \text{kun } i = j = 0. \end{cases}$$

- Nollakohdat $T_n(x_k) = 0$

$$x_k = \cos \frac{\pi(k - \frac{1}{2})}{n}, \quad \text{kun} \quad k = 1, 2, \ldots, n$$

- Ääriarvot $T_n'(x_k) = 0$

$$x_k = \cos \frac{\pi k}{n}, \quad \text{kun} \quad k = 0, 1, \ldots, n$$

  Kaikissa maksimeissa $T_n(x_{max}) = 1$ ja kaikissa minimeissä $T_n(x_{min}) = -1$

- Ortogonaalisuus polynomin T_m nollakohdissa x_k, kun $i < m$ ja $j < m$

$$\sum_{k=1}^{m} T_i(x_k)T_j(x_k) = \begin{cases} 0, & \text{kun } i \neq j \\ \frac{m}{2}, & \text{kun } i = j \neq 0 \\ m, & \text{kun } i = j = 0. \end{cases}$$

Näitä ominaisuuksia käyttäen voidaan funktio $f(x)$ kehittää astetta $N-1$ olevaksi potenssisarjaksi Chebysev'in polynomien avulla

$$f(x) \approx -\frac{1}{2}c_0 + \sum_{k=0}^{N-1} c_k T_k(x)$$

Kertoimet c_k määrätään siten, että kehitelmä on tarkka Chebysev'in polynomin $T_N(x)$ nollakohdissa $x_k, \quad k = 1, \ldots, N$.

$$\begin{aligned} c_j &= \frac{2}{N} \sum_{k=1}^{N} f(x_k) T_j(x_k) \qquad (2.1) \\ &= \frac{2}{N} \sum_{k=1}^{N} f\left(\cos \frac{\pi(k-\frac{1}{2})}{N} \right) \cos \frac{\pi j (k-\frac{1}{2})}{N} \end{aligned}$$

Näin saatu polynomi on hyvin lähellä sitä polynomia, jolla kaikista astetta $N-1$ olevista polynomeista on pienin maksimipoikkeama oikeasta arvosta (ns. *minimax*-polynomi).

Osoitetaan, että kehitelmä on konsistentti:

$$\begin{aligned} c_j &= \frac{2}{N} \sum_{k=1}^{N} \left(-\frac{1}{2} c_0 + \sum_{l=0}^{N-1} c_l T_l(x_k) \right) T_j(x_k) \\ &= \frac{2}{N} \left[-\frac{1}{2} c_0 \sum_{k=1}^{N} T_0(x_k) T_j(x_k) \right. \\ &+ \left. \sum_{l=0}^{N-1} c_l \sum_{k=1}^{N} T_l(x_k) T_j(x_k) \right] \\ &= \begin{cases} \frac{2}{N} \left[-\frac{1}{2} c_0 N + c_0 N \right] = c_0, & \text{kun } j = 0 \\ \frac{2}{N} c_j \frac{N}{2} = c_j, & \text{kun } j > 0. \end{cases} \end{aligned}$$

2.1.5 Clenshaw'n palautuskaava

Clenshaw'n palautuskaavan avulla voidaan näppärästi laskea edellä esitetyn tyyppisiä summia,

$$f(x) = \sum_{k=0}^{N} c_k F_k(x) \ ,$$

jossa kantafunktiot $F_k(x)$ noudattavat tyyppiä

$$F_{n+1}(x) = \alpha(n,x) F_n(x) + \beta(n,x) F_{n-1}(x)$$

olevaa palautuskaavaa. Tässä $\alpha(n,x)$ ja $\beta(n,x)$ ovat tunnettuja funktioita.
Määritellään funktiot $y_k(x), \quad k = N+2, N+1, \ldots, 1$

$$\begin{aligned} y_{N+2}(x) &= y_{N+1}(x) \equiv 0 \\ y_k(x) &= \alpha(k,x) y_{k+1}(x) + \beta(k+1,x) y_{k+2}(x) + c_k, \\ &\quad \text{kun} \quad k = N, N-1, \ldots, 1 \end{aligned}$$

Ratkaistaan c_k ja sijoitetaan funktion kehitelmään.

$$\begin{aligned} f(x) &= \ldots \\ &+ [y_8 - \alpha(8,x) y_9 - \beta(9,x) y_{10}] F_8(x) \\ &+ [y_7 - \alpha(7,x) y_8 - \beta(8,x) y_9] F_7(x) \\ &+ [y_6 - \alpha(6,x) y_7 - \beta(7,x) y_8] F_6(x) \end{aligned}$$

$$
\begin{aligned}
&+ \quad [y_5 - \alpha(5,x)y_6 - \beta(6,x)y_7]F_5(x) \\
&+ \quad [y_4 - \alpha(4,x)y_5 - \beta(5,x)y_6]F_4(x) \\
&+ \quad [y_3 - \alpha(3,x)y_4 - \beta(4,x)y_5]F_3(x) \\
&+ \quad [y_2 - \alpha(2,x)y_3 - \beta(3,x)y_4]F_2(x) \\
&+ \quad [y_1 - \alpha(1,x)y_2 - \beta(2,x)y_3]F_1(x) \\
&+ \quad [c_0 + \beta(1,x)y_2 - \beta(1,x)y_2]F_0(x)
\end{aligned}
$$

Termin $y_8(x)$ kerroin on

$$F_8(x) - \alpha(7,x)F_7(x) - \beta(7,x)F_6(x) = 0\,,$$

koska $F_n(x)$ toteuttaa edellä esitetyn palautuskaavan. Vastaavasti voidaan osoittaa, että suurin osa termeistä summassa häviää. Jäljelle jää

$$f(x) = \beta(1,x)F_0(x)y_2(x) + F_1(x)y_1(x) + F_0(x)c_0\ ,$$

josta funktion arvo voidaan laskea. Näin summa on kutistunut kolmeen termiin ja alkuperäisen palautuskaavan sijasta käytetään funktioille $y_k(x)$ määriteltyä palautuskaavaa.

Sovelletaan tätä tekniikkaa Chebysev'in polynomien avulla tehtyyn kehitelmään. Silloin

$$
\begin{aligned}
\alpha(n,x) &= 2x \\
\beta(n,x) &= -1
\end{aligned}
$$

ja palautuskaava funktioille $y_j(x)$

$$
\begin{aligned}
y_j(x) &= 2xy_{j+1}(x) - y_{j+2}(x) + c_j\ , \\
y_{N+2}(x) &= y_{N+1}(x) = 0\ .
\end{aligned}
$$

Kertoimet c_j saadaan kaavasta (2.1) ja itse funktion arvo saadaan sitten laskettua yksikertaisesti

$$f(x) = -y_2(x) + xy_1(x) + \frac{1}{2}c_0\,.$$

2.1.6 Padén approksimaatio

Rationaalifunktiota

$$R(x) = \frac{\sum_{k=0}^{M} a_k x^k}{1 + \sum_{k=1}^{N} b_k x^k}$$

sanotaan sarjan

$$f(x) = \sum_{k=0}^{\infty} c_k x^k$$

Padén approksimaatioksi, jos

$$R(0) = f(0)$$

ja lisäksi

$$\frac{d^k}{dx^k}R(x)\bigg|_{x=0} = \frac{d^k}{dx^k}f(x)\bigg|_{x=0}\ ,\ k = 1,2,\ldots,M+N.$$

Kertoimet $a_0,\ldots,a_M$ ja $b_1,\ldots,b_N$ määräytyvät näistä ehdoista ($N+M+1$ kpl).
Polynomien osamääränä Padén approksimaatio on singulaarinen nimittäjän nollakohdissa (eivät välttämättä reaalisia). Näin ollen Padén approksimaatio voi heijastaa myös approksimoitavan funktion analyyttisiä ominaisuuksia. Tästä syystä sitä käytetään esim. funktion $\tan x$ (navat pisteissä $x = \pm\pi/2$) laskentaan.

2.2 Interpolointi

Interpolointia tarvitaan, kun tunnetaan funktion arvot määrätyissä pisteissä

$$y_i = f(x_i), \text{ kun } \quad i = 1, \ldots, n \ ,$$

ja halutaan laskea funktion arvo jossakin pisteessä x.
Funktiota voidaan approksimoida näiden n:n pisteen kautta kulkevalla $(n-1)$:n asteen polynomilla

$$y(x) = \sum_{j=1}^{n} a_j \, x^{j-1} \ .$$

Yksi mahdollisuus kertoimien määräämiseen on ratkaista ne lineaarisesta yhtälöryhmästä, joka saadaan sijoittamalla yhtälöön funktion arvot tunnetuissa pisteissä.

2.2.1 Lagrangen interpolaatiopolynomi

Nopeampi tapa on kuitenkin käyttää **Lagrangen polynomeja** $L_j(x)$, jolloin interpoloiva polynomi voidaan kirjoittaa muotoon:

$$y(x) \approx P(x) = \sum_{j=1}^{n} y_j \, L_j(x) \ .$$

Vaaditaan, että $L_j(x)$ on polynomi, jonka asteluku on $\leq n-1$, ja se toteuttaa ehdot

$$\begin{aligned} L_j(x_i) &= 0, \quad \text{kun} \quad i \neq j \\ L_j(x_j) &= 1 \ . \end{aligned}$$

Tällöin

$$y(x_i) = P(x_i) = y_i \ .$$

Ensimmäinen ehto totetutuu, kun

$$L_j(x) = C(x - x_1) \cdots (x - x_{j-1}) \, (x - x_{j+1}) \cdots (x - x_n)$$

ja vakio C saadaan määrättyä toisesta ehdosta.
Molemmat ehdot toteutuvat siten, kun

$$L_j(x) = \frac{(x - x_1) \cdots (x - x_{j-1})(x - x_{j+1}) \cdots (x - x_n)}{(x_j - x_1) \cdots (x_j - x_{j-1})(x_j - x_{j+1}) \cdots (x_j - x_n)} \ .$$

2.2.2 Newtonin interpolaatiopolynomi

Ensimmäisen asteen (tai lineaarinen) interpolaatio on yksinkertaisin mahdollinen interpolaatiokaava ja saadaan approksimoimalla funktiota kahden pisteen kautta kulkevalla suoralla. Funktion arvo pisteessä x saadaan silloin laskettua kaavasta:

$$\begin{aligned} f(x) &= c_1 + c_2 \, (x - x_1) \\ &= f(x_1) + \frac{f(x_2) - f(x_1)}{x_2 - x_1} \, (x - x_1) \end{aligned}$$

Yleinen $(n-1)$:n asteen Newtonin interpolaatiokaava on puolestaan muotoa:

$$\begin{aligned} f(x) &= c_1 + c_2(x - x_1) + \cdots \\ &+ c_n(x - x_1)\cdots(x - x_{n-1}) \end{aligned}$$

Kertoimien $c_1, \ldots, c_n$ määräämiseksi täytyy tuntea funktion arvot n:ssä pisteessä.

Esimerkki: Kun $n = 3$ voidaan helposti laskea kertoimet:

$$\begin{aligned} c_1 &= f(x_1) \\ c_2 &= \frac{f(x_2) - c_1}{x_2 - x_1} = \frac{f(x_2) - f(x_1)}{x_2 - x_1} \\ c_3 &= \frac{f(x_3) - c_1}{(x_3 - x_1)(x_3 - x_2)} - \frac{c_2}{x_3 - x_2} \\ &= \frac{f(x_3) - f(x_1)}{(x_3 - x_1)(x_3 - x_2)} - \frac{f(x_2) - f(x_1)}{(x_2 - x_1)(x_3 - x_2)} \end{aligned}$$

Jos tunnetut pisteet sijaitsevat tasavälein

$$x_i = x_1 + (i-1)\Delta x, \qquad i = 1, 2, \ldots, n$$

saadaan tulokseksi

$$\begin{aligned} c_1 &= f(x_1) \\ c_2 &= \frac{f(x_2) - f(x_1)}{\Delta x} = \frac{\Delta f_1}{\Delta x} \\ c_3 &= \frac{f(x_3) - 2\,f(x_2) + f(x_1)}{2\,(\Delta x)^2} = \frac{\Delta^2 f_1}{2!(\Delta x)^2} \end{aligned}$$

---

missä

$$\begin{aligned} \Delta f_i &= f_{i+1} - f_i \\ \Delta^2 f_i &= \Delta f_{i+1} - \Delta f_i. \end{aligned}$$

Induktiolla voitaisiin osoittaa, että

$$c_n = \frac{\Delta^n f_1}{n!(\Delta x)^n} \ ,$$

missä $\Delta^n f_i$ on n:n kertaluvun erotus (differenssi) pisteessä x_i ja lasketaan palautuskaavasta

$$\Delta^n f_i = \Delta^{n-1} f_{i+1} - \Delta^{n-1} f_i \ .$$

2.2.3 Neville'n algoritmi

Muodostetaan interpoloiva polynomi Neville'n algoritmia käyttäen. Algoritmi antaa funktion arvon halutussa pisteessä, mutta myös arvion approksimaation tarkkuudesta, jota sitten voidaan parantaa ottamalla mukaan lisää pisteitä.
Käytetään merkintöjä:

- $P_1 = y_1, P_2 = y_2, \cdots, P_n = y_n$,
- $P_{1,2}(x)$ on ensimmäisen asteen polynomi, joka kulkee pisteiden (x_1, y_1) ja (x_2, y_2) kautta.
- $P_{i,i+1}(x)$ kulkee pisteiden (x_i, y_i) ja (x_{i+1}, y_{i+1}) kautta.
- $P_{i,i+1,i+2}(x)$ on toisen asteen polynomi, joka kulkee pisteiden $(x_i, y_i), (x_{i+1}, y_{i+1})$ ja (x_{i+2}, y_{i+2}) kautta.
- $P_{1,\cdots,n}(x)$ on $(n-1)$ asteen polynomi, joka kulkee kaikkien pisteiden $(x_i, y_i), i = 1, \cdots, n$ kautta.

Muodostetaan taulukko:

$$
\begin{array}{lllll}
x_1 : y_1 = P_1 & & & & \\
 & P_{1,2}(x) & & & \\
x_2 : y_2 = P_2 & & P_{1,2,3}(x) & & \\
 & P_{2,3}(x) & & P_{1,2,3,4}(x) & \\
x_3 : y_3 = P_3 & & P_{2,3,4}(x) & & \\
 & P_{3,4}(x) & & & \\
x_4 : y_4 = P_4 & & & &
\end{array}
$$

Neville'n algoritmissa polynomien arvot täytetään taulukkoon seuraavan palautuskaavan avulla.

$$
P_{i,\cdots,i+m}(x) =
$$

$$
\frac{(x - x_{i+m})P_{i,\cdots,i+m-1}(x) + (x_i - x)P_{i+1,\cdots,i+m}(x)}{(x_i - x_{i+m})}
$$

Esimerkki: Lasketaan $P_{1,2,3}$ pisteissä x_1, x_2 ja x_3.

$$
\begin{aligned}
P_{1,2,3}(x) &= \frac{(x - x_3)P_{1,2}(x) + (x_1 - x)P_{2,3}(x)}{x_1 - x_3} \\
P_{1,2}(x) &= \frac{(x - x_2)P_1 + (x_1 - x)P_2}{x_1 - x_2} \\
P_{2,3}(x) &= \frac{(x - x_3)P_2 + (x_2 - x)P_3}{x_2 - x_3}
\end{aligned}
$$

$$
P_{1,2,3}(x_1) = P_{1,2}(x_1) = P_1 = y_1
$$

$$
\begin{aligned}
P_{1,2,3}(x_2) &= \frac{(x_2 - x_3)P_{1,2}(x_2) + (x_1 - x_2)P_{2,3}(x_2)}{x_1 - x_3} \\
&= \frac{(x_2 - x_3)P_2 + (x_1 - x_2)P_2}{x_1 - x_3} = P_2 = y_2
\end{aligned}
$$

$$
P_{1,2,3}(x_3) = P_{2,3}(x_3) = P_3 = y_3
$$

Parannettu palautuskaava saadaan, kun lasketaan polynomien erotuksia.

$$\begin{aligned} C_{m,i}(x) &\equiv P_{i,\cdots,i+m}(x) - P_{i,\cdots,i+m-1}(x) \\ D_{m,i}(x) &\equiv P_{i,\cdots,i+m}(x) - P_{i+1,\cdots,i+m}(x) \end{aligned}$$

$C_{m,i}(x)$ ja $D_{m,i}(x)$ ovat m:nnen asteen polynomeja, joiden avulla saadaan interpoloivan polynomin astelukua nostettua yhdellä

$$P_{i,\cdots,i+m}(x) = P_{i,\cdots,i+m-1}(x) + C_{m,i}(x)$$

tai

$$P_{i,\cdots,i+m}(x) = P_{i+1,\cdots,i+m}(x) + D_{m,i}(x)$$

Käyttämällä polynomien $P_{i,\cdots,i+m}(x)$ palautuskaavoja saadaan esitykset

$$\begin{aligned} C_{m,i} &= \frac{(x - x_i)}{(x_i - x_{i+m})} \left[P_{i,\cdots,i+m-1} - P_{i+1,\cdots,i+m}\right] \\ D_{m,i} &= \frac{(x - x_{i+m})}{(x_i - x_{i+m})} \left[P_{i,\cdots,i+m-1} - P_{i+1,\cdots,i+m}\ \right], \end{aligned}$$

joista johdetaan palautuskaavat erotuspolynomeille $C_{m,i}(x)$ ja $D_{m,i}(x)$:

$$\begin{aligned} C_{m+1,i}(x) &= \frac{(x_i - x)}{(x_i - x_{i+m+1})} \Big[C_{m,i+1}(x) - D_{m,i}(x)\Big] \\ D_{m+1,i}(x) &= \frac{(x_{i+m+1} - x)}{(x_i - x_{i+m+1})} \Big[C_{m,i+1}(x) - D_{m,i}(x)\Big]\ . \end{aligned}$$

Alkuarvoina ovat $C_{0,i} = y_i$ ja $D_{0,i} = y_i$.

Esimerkki: Lasketaan ensimmäinen askel

$$\begin{aligned} C_{1,i}(x) &= \frac{(x_i - x)}{(x_i - x_{i+1})}(y_{i+1} - y_i) \\ D_{1,i}(x) &= \frac{(x_{i+1} - x)}{(x_i - x_{i+1})}(y_{i+1} - y_i)\ . \end{aligned}$$

Pisteiden (x_i, y_i) ja (x_{i+1}, y_{i+1}) kautta kulkeva suora voidaan sitten esittää muodoissa

$$\begin{aligned} y(x) &= y_i + C_{1,i}(x) \\ y(x) &= y_{i+1} + D_{1,i}(x)\ . \end{aligned}$$

Polynomit $C_{m,i}(x)$ ja $D_{m,i}(x)$ antavat korjauksen, jolla nostetaan interpoloivan polynomin astetta yhdellä, kun siirrytään sarakkeelta m sarakkeelle $m + 1$ alla olevassa taulukossa.

$$
\begin{array}{lll}
x_1 : y_1 = P_1 & \searrow C_{1,1} & \\
 & & P_{1,2}(x) \\
x_2 : y_2 = P_2 & \nearrow D_{1,1} & \\
 & & P_{2,3}(x) \\
x_3 : y_3 = P_3 & \searrow C_{1,3} & \\
 & & P_{3,4}(x) \\
x_4 : y_4 = P_4 & \nearrow D_{1,3} &
\end{array}
\qquad
\begin{array}{ll}
 & \\
 & P_{1,2,3}(x) \\
 & \\
 & P_{2,3,4}(x) \\
 &
\end{array}
\qquad
P_{1,2,3,4}(x)
$$

Polku interpoloivaan polynomiin.

$$
\begin{array}{lllll}
y_1 & & & & \\
 & P_{1,2} & & & \\
y_2 & & P_{1,2,3} & & \\
 & P_{2,3} & & & P_{1,2,3,4} \\
\nearrow D_{1,2} & & \searrow C_{2,2} & \nearrow D_{3,1} & \\
y_3 & & P_{2,3,4} & & \\
 & P_{3,4} & & & \\
y_4 & & & &
\end{array}
$$

Neljän annetun pisteen kautta kulkeva polynomi antaa interpoloinnin tulokseksi pisteessä x arvon

$$P_{1,2,3,4}(x) = y_3 + D_{1,2}(x) + C_{2,2}(x) + D_{3,1}(x) \ .$$

Korjaus, joka saadaan, kun piste x_1 otetaan mukaan interpolointiin pisteiden x_2, x_3 ja x_4 lisäksi on $D_{3,1}(x)$. Tätä voidaan käyttää virheen arvioinnissa hyväksi.
Interpolointi kannattaa suorittaa siten, että $|x - x_i|, \ \ i = 1, \ldots n$ on pienin mahdollinen ja polku, jota pitkin edetään, kulkee mahdollisimman keskellä kaaviota.

Tehtävä: Oletetaan, että edellä esitetty polynomi kulkee pisteiden (1,1), (2,3), (3,6) ja (4,8) kautta. Laske sen arvo pisteessä x=2.5. Tehdään tehtävä käsin, mutta Numerical Receipes kirjastossa on ohjelmaa `POLINT`, jolla voi suorittaa interpoloinnin.

2.2.4 Rationaalifunktio-interpolointi

Merkintä:

$$R_{i,\cdot,i+m}(x) \equiv \frac{P_\mu(x)}{Q_\nu(x)} = \frac{p_0 + p_1 x + \cdot + p_\mu x^\mu}{q_0 + q_1 x + \cdot + q_\nu x^\nu}$$

on rationaalifunktio, joka kulkee pisteiden $(x_i, y_i), \cdots, (x_{i+m}, y_{i+m})$ kautta. Tällöin $m+1 = \mu+\nu+1$, sillä q_0 on mielivaltainen.
Approksimaatio on erittäin hyvä, kun interpoloitavalla funktiolla on napa reaaliakselilla, mutta on myös tarpeellinen, jos kompleksitason napa on lähellä interpolointialuetta. Pade approksimaatio mm. käyttää hyväkseen rationaalifunktiota sellaisten funktioiden kehittämiseen, joilla on napoja.

Bulirsch-Stoer algoritmi rationaalifunktion laskemiseksi on hyvin samanlainen kuin Neville'n algoritmi. Käytetyt polynomit rajataan siten, että $\mu = \nu$, kun m on parillinen ja $\mu = \nu - 1$, kun m on pariton.
Rationaalifunktiolle $R_{i,\cdots,i+m}$ voidaan johtaa palautuskaava

$$R_{i,\cdots,i+m} = R_{i+1,\cdots,i+m}$$

$$+ \frac{R_{i+1,\cdots,i+m} - R_{i,\cdots,i+m-1}}{\frac{(x-x_i)}{(x-x_{i+m})}\left(1 - \frac{R_{i+1,\cdots,i+m} - R_{i,\cdots,i+m-1}}{R_{i+1,\cdots,i+m} - R_{i+1,\cdots,i+m-1}}\right) - 1} \ ,$$

jossa $m+1$:n pisteen kautta kulkeva funktio saadaan m:n ja $m-1$:n pisteen kautta kulkevien funktioiden avulla.
Alkuarvot ovat

$$\begin{aligned} R_i &= y_i \\ R_{i,\cdots,i+m} &= 0\ , \quad \text{kun} \quad m=-1\ . \end{aligned}$$

Määritellään erotusfunktiot

$$\begin{aligned} C_{m,i}(x) &\equiv R_{i,\cdots,i+m}(x) - R_{i,\cdots,i+m-1}(x) \\ D_{m,i}(x) &\equiv R_{i,\cdots,i+m}(x) - R_{i+1,\cdots,i+m}(x). \end{aligned}$$

Ne toteuttavat ehdon

$$C_{m+1,i}(x) - D_{m+1,i}(x) = C_{m,i+1}(x) - D_{m,i}(x)\ ,$$

ja saadaan laskettua palautuskaavasta

$$\begin{aligned} C_{m+1,i} &= \frac{\frac{(x-x_i)}{(x-x_{i+m+1})} D_{m,i}\Big(C_{m,i+1} - D_{m,i}\Big)}{\frac{(x-x_i)}{(x-x_{i+m+1})} D_{m,i} - C_{m,i+1}} \\ D_{m+1,i} &= \frac{C_{m,i+1}\Big(C_{m,i+1} - D_{m,i}\Big)}{\frac{(x-x_i)}{(x-x_{i+m+1})} D_{m,i} - C_{m,i+1}}\ . \end{aligned}$$

Esimerkki: Käytetään Numerical Receipes paketin ohjelmaa `RATINT` interpoloivan polynomin laskemisessa.

2.2.5 Spline-interpolaatio

Useissa käytännön sovelluksissa interpolaatiota joudutaan soveltamaan suureen joukkoon pisteitä, (x_i, y_i), $i = 1, \ldots, n$, jolloin olisi käytettävä korkean asteen interpolaatiopolynomia, kun halutaan saada tarkkoja tuloksia. Laskennallisten tai polynomiapproksimaation laskentatarkkuuteen liittyvien ongelmien vuoksi (esimerkiksi korkean asteen interpolaatiopolynomeissa esiintyy usein oskillaatioita), on kuitenkin suositeltavampaa käyttää jaksoittaista alemman kertaluvun interpolaatiopolynomia.

Funktiota $f(x)$ approksimoidaan kolmannen asteen polynomilla (**cubic spline**) kullakin välin $[x_1, x_n]$ osavälillä $[x_i, x_{i+1}]$ erikseen eli

$$f(x) \approx s_i(x), \quad \text{kun} \quad x_i \le x \le x_{i+1}$$

ja

$$s_i(x) = y_i + b_i(x - x_i) + c_i(x - x_i)^2 + d_i(x - x_i)^3\ .$$

Vaaditaan, että polynomi $s_i(x)$ toteuttaa seuraavat ehdot:

- $s_i(x_i) = f(x_i) = y_i$
- $s_i(x_{i+1}) = f(x_{i+1}) = y_{i+1}$

- derivaattojen $s_i'(x)$ ja $s_i''(x)$ on oltava jatkuvia pisteissä x_i

Polynomin muodostamiseksi approksimoidaan funktion $f(x)$ toista derivaattaa $f''(x)$ välillä $x_i \leq x \leq x_{i+1}$ suoralla:

$$f''(x) \approx s_i''(x) = s_i''(x_i)\frac{x_{i+1}-x}{x_{i+1}-x_i} + s_i''(x_{i+1})\frac{x-x_i}{x_{i+1}-x_i} \ .$$

Kertoimet $s_i''(x_i)$ ja $s_i''(x_{i+1})$ ovat toistaiseksi tuntemattomia ja niiden laskeminen on spline-interpoloinnin pääongelma.
Integroimalla kaksi kertaa saadaan kolmannen asteen approksimaatiopolynomiksi $s_i(x)$ tällä välillä i

$$\begin{aligned} s_i(x) &= s_i''(x_i)\frac{(x_{i+1}-x)^3}{6\Delta x_i} + s_i''(x_{i+1})\frac{(x-x_i)^3}{6\Delta x_i} \\ &+ C^{(1)}x + C^{(2)} \ , \end{aligned}$$

missä $\Delta x_i = x_{i+1} - x_i$.

Integroimisvakiot $C^{(1)}$ ja $C^{(2)}$ määrätään siten, että polynomi kulkee pisteiden (x_i, y_i) ja (x_{i+1}, y_{i+1}) kautta, eli $s_i(x_i) = y_i$ ja $s_i(x_{i+1}) = y_{i+1}$. Tuloksena saadaan polynomi

$$\begin{aligned} s_i(x) &= s_i''(x_i)\frac{(x_{i+1}-x)^3}{6\Delta x_i} + s_i''(x_{i+1})\frac{(x-x_i)^3}{6\Delta x_i} \\ &+ \left[\frac{y_i}{\Delta x_i} - \frac{\Delta x_i}{6}s_i''(x_i)\right](x_{i+1}-x) \\ &+ \left[\frac{y_{i+1}}{\Delta x_i} - \frac{\Delta x_i}{6}s_i''(x_{i+1})\right](x-x_i) \ , \end{aligned}$$

Sijoittamalla $x_{i+1} - x = \Delta x_i - (x - x_i)$ saadaan myös alussa esitetyn kolmannanen asteen polynomin kertoimet laskettua.

$$\begin{aligned} d_i &= \frac{1}{6\Delta x_i}[s_i''(x_{i+1}) - s_i''(x_i)] \\ c_i &= \frac{1}{2}s_i''(x_i) \\ b_i &= \frac{1}{\Delta x_i}[y_{i+1} - y_i] \\ &- \frac{\Delta x_i}{6}[s_i''(x_{i+1}) + 2s_i''(x_i)] \ . \end{aligned}$$

Vaaditaan lisäksi, että ensimmäinen derivaatta on jatkuva solmupisteissä

$$s_i'(x_i) = s_{i-1}'(x_i) \ ,$$

eli välin i vasemmanpuoleisessa päätepisteessä x_i laskettu derivaatta on sama kuin välin $(i-1)$ oikeanpuoleisessa päätepisteessä x_i laskettu derivaatta. Silloin saadaan seuraava lineaarinen yhtälöryhmä toisen derivaatan arvoille:

$$\Delta x_{i-1}s_{i-1}''(x_{i-1}) + 2(\Delta x_i + \Delta x_{i-1})s_i''(x_i)$$

$$+\Delta x_i s_{i+1}''(x_{i+1}) = 6[\frac{y_{i+1}-y_i}{\Delta x_i} - \frac{y_i - y_{i-1}}{\Delta x_{i-1}}] \ ,$$

missä $i = 2, \ldots, n-1$, ja n on pisteiden lukumäärä, $\Delta x_{i-1} = x_i - x_{i-1}$. Lisäksi pitää huomata, että

$$\begin{aligned} s_i''(x_{i+1}) &= s_{i+1}''(x_{i+1}) \\ s_{i-1}''(x_i) &= s_i''(x_i) \end{aligned}$$

Yhtälöitä on $(n-2)$ kappaletta ja niissä n tuntematonta $s_i''(x_i), \quad i = 1, 2, \ldots, n$, joten tarvitaan kaksi lisäehtoa yhtälöryhmän ratkaisemiseksi. Tavallisin tapa (*natural cubic spline*) on asettaa toinen derivaatta taulukoitujen arvojen ääripisteissä nollaksi

$$s_1''(x_1) = s_n''(x_n) = 0 \ .$$

Tuloksena saadaan funktio, joka on mahdollisimman sileä, eli spline-ehdot toteuttavista kuutiollisista polynomeista natural cubic spline antaa integraalille

$$\int_{x_1}^{x_n} \sum_i (s_i''(x))^2 dx$$

pienimmän arvon.

Näin on muodostettu lineaarinen yhtälöryhmä funktion toisen derivaatan laskemiseksi solmupisteissä. Yhtälön kerroinmatriisi on muotoa,

$$\begin{pmatrix} 2(l_2+l_1) & l_2 & 0 & \ldots & 0 & 0 \\ l_2 & 2(l_3+l_2) & l_3 & \ldots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \ldots & l_{n-2} & 2(l_{n-1}+l_{n-2}) \end{pmatrix}$$

,

missä on käytetty merkintää $l_i = \Delta x_i$.

Numerical Recipes kirjastossa on kaksi ohjelmaa, joita käyttäen spline approksimaatio voidaan laskea. Ensin on laskettava spline kertoimet eli $s_i''(x_i), \ i = 1, 2, \ldots, n$ aliohjelmassa `SPLINE`, jonka jälkeen itse funktion arvo pisteessä x voidaan laskea aliohjelmalla `SPLINT`.
Huomaa, että ohjelmassa `SPLINE` ei käytetä natural splinea vaan lisäehdoiksi on valittu ensimmäisen derivaatan arvot välin päätepisteissä `YP1` ja `YP2`. Asettamalla nämä luvut suuremmiksi tai yhtä suuriksi kuin $1 \times^{30}$ ohjelma laskee natur spline-kertoimet.
Aliohjelma `SPLINE` on itse asiassa tridiagonaalisen yhtälöryhmän ratkaisu Gaussin elimointimenetelmällä. Asia käsitellään tässä kurssissa tarkasti myöhemmin. Aliohjelmassa `SPLINT` etsitään ensin se väli, missä piste x sijaitsee ja käytetään sen jälkeen cubic spline interpolaatiopolynomia tällä välillä funktion arvon laskemiseen.

2.3 Numeerinen derivointi

Oletetaan, että funktio tunnetaan pisteissä (x_i, y_i), kun $i = 1, \cdots, n$. Derivointiin voidaan käyttää interpoloivaa polynomia tai splinen antamaa polynomia ja derivoida tämä.
Funktion derivaatan määritelmä on

$$f'(x) = \frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

2.3.1 Lineaarinen interpolaatio; kahden pisteen kaava

Kun tunnetaan funktion arvo kahdessa pisteessä, sen derivaatan arvo voidaan laskea suoraan derivaatan määritelmästä,

$$f'(x) = \frac{f(x_{i+1}) - f(x_i)}{h},$$

missä $h = x_{i+1} - x_i$.
Tämä tarkoittaa itseasiassa sitä, että funktiota approksimoidaan pisteiden kautta kulkevalla suoralla

$$y = y_1 + \frac{y_2 - y_1}{x_2 - x_1}\,(x - x_1),$$

jolloin derivaatta on

$$y'(x) = \frac{y_2 - y_1}{x_2 - x_1} = \frac{f(x_1 + h) - f(x_1)}{h}.$$

2.3.2 Parabeli-interpolaatio; kolmen pisteen kaava

Oletetaan, että funktio kulkee pisteiden (x_1, y_1), (x_2, y_2) ja (x_3, y_3) kautta ja olkoon kahden peräkkäisen pisteen väli h, kun pisteet sijaitsevat tasavälein. Silloin kolmen pisteen Lagrangen interpolaatiokaava

$$\begin{aligned}
y(x) &\approx P(x) = \sum_{j=1}^{3} y_j\, L_j(x)\ , \\
L_1(x) &= \frac{(x - x_2)(x - x_3)}{(x_1 - x_2)(x_1 - x_3)} \\
L_2(x) &= \frac{(x - x_1)(x - x_3)}{(x_2 - x_1)(x_2 - x_3)} \\
L_3(x) &= \frac{(x - x_1)(x - x_2)}{(x_3 - x_1)(x_3 - x_2)}
\end{aligned}$$

voidaan kirjoittaa muotoon

$$y(x_2 + t) = \frac{1}{2h^2}\left[y_1 t(t-h) - 2y_2(t^2 - h^2) + y_3 t(t+h)\right],$$

missä $t = x - x_2$.
Määritellään parametri $p = t/h$, jolloin saadaan

$$y(x_2 + ph) = \frac{1}{2}\left[p(p-1)\, y_1 - 2(p^2 - 1)\, y_2 + p(p+1)\, y_3\right]\ ,$$

ja derivaatta puolestaan on

$$y'(x_2 + ph) = \frac{1}{2h}\left[(2p-1)\,y_1 - 4p\,y_2 + (2p+1)\,y_3\right],$$

koska

$$p = \frac{t}{h} = \frac{x - x_2}{h}.$$

Valitsemalla parametrille p sopiva arvo voidaan kaavaa käyttää sekä välin keskellä että sen päätepisteissä. Välin keskellä olevassa pisteessä $p = 0$, jolloin derivaatta pisteessä $x = x_2$ on

$$y'(x_2) = \frac{1}{2h}\left(-y_1 + y_3\right) = \frac{y_3 - y_1}{2h}.$$

Alkupisteessä $x = x_1$ $p = -1$, jolloin derivaattaksi saadaan

$$y'(x_1) = \frac{1}{2h}\left(-3y_1 + 4y_2 - y_3\right)$$

ja välin loppupisteessä $x = x_3$ $p = 1$

$$y'(x_3) = \frac{1}{2h}\left(y_1 - 4y_2 + 3y_3\right).$$

Esimerkki Ohjelma taulukoidun funktion derivaatan laskemiseksi kolmen pisteen kaavalla kaikissa taulukointipisteissä:

```
C deriv1.f
      subroutine deriv1(h, y, dy, n)
      integer n
      real    h, hp, y(n), dy(n)
      hp = 0.5 / h
C paatepisteet
      dy(1) = hp * (-3*y(1) + 4*y(2) - y(3))
      dy(n) = hp * (y(n-2) - 4*y(n-1) + 3 y(n))
C valipisteet
      do i = 2,n-1
        dy(i) = hp * (y(i+1) - y(i-1))
      end do
      return
      end
```

2.3.3 Viiden pisteen kaava

Oletetaan, että funktion kulkee pisteiden (x_i, y_i), $i = 1, 2, 3, 4, 5$ kautta. Viiden pisteen Lagrangen polynomista voidaan johtaa samalla tavalla kuin kolmen pisteen tapauksessa derivaatalle kaava

$$\begin{aligned} y'(x_3 + ph) \;=\; & \frac{1}{h}\left[\frac{2p^3 - 3p^2 - p + 1}{12}\,y_1\right. \\ & - \frac{4p^3 - 3p^2 - 8p + 4}{6}\,y_2 \\ & + \frac{2p^3 - 5p}{2}\,y_3 \\ & - \frac{4p^3 + 3p^2 - 8p - 4}{6}\,y_4 \\ & + \left.\frac{2p^3 + 3p^2 - p - 1}{12}\,y_5\right] \end{aligned}$$

Keskimmäisessä pisteissä derivaatan arvo saadaan asettamalla jälleen $p = 0$.

$$y'(x_3) = \frac{1}{h}\,(\frac{1}{12}y_1 - \frac{2}{3}y_2 + \frac{2}{3}y_4 - \frac{1}{12}y_5).$$

2.3.4 Toinen derivaatta polynomiapproksimaatiosta

Derivoimalla kolmen pisteen Lagrangen interpolaatiopolynomi saadaan kolmen pisteen kaavaksi

$$y''(x) = \frac{1}{h^2}\,(y_1 - 2y_2 + y_3).$$

Tämä ei ole kuitenkaan kovin hyvä approksimaatio, koska kolmen pisteen kautta piirretyn parabelin toinen derivaatta on vakio. Toisen derivaatan laskemiseen on siksi parempi käyttää viiden pisteen kaavaa, joka antaa esimerkiksi pisteessä x_3 arvon.

$$y''(x_3) = \frac{1}{12h^2}\,(-y_1 + 16y_2 - 30y_3 + 16y_4 - y_5).$$

2.3.5 Derivointi splinen avulla

Oletetaan, että funktiota $y(x)$ interpoloiva cubic spline polynomi

$$\begin{aligned} y(x) &\approx s_i(x) = \\ y_i &+ b_i\,(x - x_i) + c_i\,(x - x_i)^2 + d_i\,(x - x_i)^3 \end{aligned}$$

tunnetaan, kun $x_i \le x \le x_{i+1}$ ja $i = 1, \ldots, n$.
Silloin saadaan approksimaatiot ensimmäiselle derivaatalle

$$y'(x) \approx s_i'(x) = b_i + 2c_i\,(x - x_i) + 3d_i\,(x - x_i)^2$$

ja toiselle derivaatalle

$$y''(x) \approx s_i''(x) = 2c_i + 6d_i\,(x - x_i)\ .$$

Kertoimet laskettiin jo edellä

$$\begin{aligned} d_i &= \frac{1}{6\Delta x_i}[s_i''(x_{i+1}) - s_i''(x_i)] \\ c_i &= \frac{1}{2}s_i''(x_i) \\ b_i &= \frac{1}{\Delta x_i}[y_{i+1} - y_i] \\ &- \frac{\Delta x_i}{6}[s_i''(x_{i+1}) + 2s_i''(x_i)]\ . \end{aligned}$$

Solmupisteissä lasketut derivaatat ovat silloin

$$\begin{aligned} y'(x_i) &\approx s_i'(x_i) = b_i \\ &= \frac{1}{\Delta x_i}(y_{i+1} - y_i) - \frac{\Delta x_i}{6}(s_i''(x_{i+1}) + 2s_i''(x_i)) \\ y''(x_i) &\approx s_i''(x_i) = 2c_i. \end{aligned}$$

2.4 Funktion nollakohdat (juuret)

Etsitään yleisen yhden muuttujan funktion nollakohdat (juuret)

$$f(x) = 0.$$

2.4.1 Puolitusmenetelmä

Puolitusmenetelmässä valitaan lähtöarvoiksi kaksi pistettä, joiden välissä nollakohdan tiedetään sijaitsevan. Olkoon

$$\epsilon_0 = x_2 - x_1 \ .$$

Väliä puolitetaan

$$\epsilon_{n+1} = \epsilon_n/2 \ ,$$

kunnes juuren arvo tunnetaan halutulla tarkkuudella

$$\epsilon = \frac{\epsilon_0}{2^n} \ .$$

Myös uuden välin päätepisteiden on oltava nollakohdan eri puolilla eli toisin sanoen funktion arvojen on oltava erimerkkiset välin päätepisteissä ($f(x_1)f(x_2) \leq 0$). Iterointiin tarvitaan siis $n = \log_2 \epsilon_0/\epsilon$ iterointikertaa, mutta ratkaisu löytyy varmasti.

Yleisesti ottaen iterointi suppenee lineaarisesti, jos

$$\epsilon_{n+1} = \text{vakio} \times \epsilon_n \ ,$$

kuten puolitusmenetelmässä ja superlinaarisesti, jos

$$\epsilon_{n+1} = \text{vakio} \times (\epsilon_n)^m \ ,$$

kun $m > 1$.

Puolitusmenetelmän algoritmi ohjelmaksi kirjoitettuna voisi olla seuraava. Lähtöpisteet saadaan esimerkiksi piirtämällä funktion kuvaaja.

```
      yl = fun(xl)
      yr = fun(xr)
1     if (abs(xr-xl) .gt. acc) then
         x = 0.5*(xr+xl)
         y = fun(x)
         if( y*yl .ge. 0.0) then
            yl = y
            xl = x
         else
            yr = y
            xr =x
         end if
         goto 1
      end if
```

Numerical Recipes kirjastossa on aliohjelma `RTBIS`, jossa käytetään puolitusmenetelmää. Lisäksi Numerical Recipes kirjastossa oleva ohjelma `ZBRAC` etsii funktion $FUNC(x)$ yhtä juurta lähtien väliltä

$[X1, X2]$ ja palauttaa välin päätepisteet 50:n puolituksen jälkeen (muuttuja SUCCES kertoo onnistuiko haku). Ohjelma `ZBRAK` puolestaan etsii function $FUNC(x)$ juuria (korkeintaan NB kappaletta) väliltä $[X1, X2]$ jakamalla välin N:ään yhtäsuureen osaan. Se palauttaa juurien lukumäärän NB sekä niiden välien päätepisteet taulukoissa XB1 ja XB2, joissa juuret sijaitsevat. Näitä ohjelmia voidaan käyttää välin alkuarvojen etsimiseen, mikäli ei haluta tehdä sitä graafisesti.

2.4.2 Sekanttimenetelmä ja regula falsa

Regula falsa menetelmässä (käänteinen lineaarinen interpolaatio) approksimoidaan funktiota suoralla sellaisella välillä, jossa juuren tiedetään sijaitsevan.

1. Määrätään kaksi aloituspistettä (x_1, y_1) ja (x_2, y_2), joiden välissä juuren tiedetään sijaitsevan eli funktion arvot ovat erimerkkiset välin päätepisteissä, $y_1 * y_2 \leq 0$.

2. Approksimoidaan funktiota pisteiden (x_1, y_1) ja (x_2, y_2) kautta kulkevalla suoralla ja etsitään suoran nollakohta,

$$y = y_1 + \frac{y_2 - y_1}{x_2 - x_1}(x - x_1) = 0$$

   ja nollakohta on

$$x = x_1 - \frac{x_1 - x_2}{y_1 - y_2} y_1.$$

3. Lasketaan arvioitua nollakohtaa vastaava funtion arvo $y = f(x)$

4. Valitaan uudet pisteet (x_1, y_1) ja (x_2, y_2) siten, että, jos $y\, y_1 \geq 0$, niin $x_1 = x,\ y_1 = y$ ja muussa tapauksessa $x_2 = x,\ y_2 = y$.

5. Testataan onko haluttu tarkkuus saavutettu: $|y| < \epsilon$ tai $|x - x_1| < \delta$. Jos tarkkuus on saavutettu, niin palataan pääohjelmaan, muussa tapauksessa palataan kohtaan 2.

Subroutine aliohjelma `regula`:

```
      subroutine regula(fun, xl, xr, x, acc)
      real fun
      real xl, xr, yl, yr, dx, x, y, acc
      yl = fun(xl)
      yr = fun(xr)
      dx = xr - xl
1     if (dx .gt. acc) then
        x = xl - (xl - xr) / (yl - yr) * yl
        y = fun(x)
        if( y*yl .ge. 0.0) the
          yl = y
          dx = abs(x - xl)
          xl = x
        else
          yr = y
          dx = abs(x - xr)
          xr = x
        end if
```

```
        goto 1
      end if
      return
      end
```

Sekanttimenetelmä on muutoin samanlainen, mutta siinä käytetään aina kahta viimeisintä arvoa riippumatta siitä, ovatko ne eri vai samalla puolella juurta (eli ei tarkistetta että $y_1 * y_2 = f(x_1) * f(x_2) \leq 0$):

$$x_{n+1} = x_n - \frac{x_n - x_{n-1}}{y_n - y_{n-1}} y_n = \frac{x_{n-1}y_n - x_n y_{n-1}}{y_n - y_{n-1}}$$

Numerical Recipes kirjastossa on aliohjelmat `RTFLSP`, joka käyttää regula falsa menetelmää (false position method englanniksi) ja `RTSEC`, joka käyttää puhdasta sekanttimenetelmää. Menetelmien erona on se että regula falsa menetelmässä tarkistetaan aina että uusissa pisteissä funktiolla on eri merkkiset arvot. `RTFLSP` suppenee sen vuoksi joskus hitaammin kuin `RTSEC`, mutta se on varmempi (iterointi ei "karkaa").

2.4.3 Yleinen iterointimenetelmä

Kirjoitetaan yhtälö $f(x) = 0$ muotoon

$$x = \phi(x) \ .$$

Tämä yhtälö voidaan silloin ratkaista yleisellä iterointimenetelmällä seuraavasti:

1. Annetaan muuttujalle x lähtöarvo x_0.
2. Lasketaan uusi arvo yhtälöstä.

$$x_{i+1} = \phi(x_i), \qquad i = 0, 1, \ldots$$

3. Testataan ratkaisun tarkkuutta. Jos se on riittävä palataan pääohjelmaan, mutta muussa tapauksessa jatketaan iterointia uudella x:n arvolla.

Algoritmi ohjelmana

```
1     if (abs(dx) .gt. acc) then
          x = fun(x0)
          dx = x - x0
          x0 = x
          goto 1
      end if
```

Ratkaisu suppenee, jos funktio $\phi(x)$ on määritelty ja differentioituva jollakin välillä $a \leq x \leq b$ siten, että kaikki funktion arvot ovat rajoitettuja kyseisellä välillä ($a \leq \phi(x) \leq b$) ja lisäksi juuren lähiympäristössä

$$|\phi'(x)| = k < 1$$

Jos $\phi'(x) > 0$ ratkaisun ympäristössä, niin suppeneminen on monotonista.
Jos $\phi'(x) < 0$ ratkaisun ympäristössä, niin iterointi oskilloi ratkaisupisteen ympärillä.
Väliarvolauseen mukaan juuren (olkoon se α) ympäristössä voidaan kirjoittaa

$$x_{i+1} - \alpha \approx \phi'(\alpha)(x_i - \alpha)$$

Silloin

$$|x_n - \alpha| \approx k^n \, |x_0 - \alpha|$$

ja

$$\lim_{n\to\infty} |x_n - \alpha| = |x_0 - \alpha| \lim_{n\to\infty} k^n = 0$$

Ratkaisu siis suppenee, $x_n \to \alpha$, kun $k < 1$.

2.4.4 Newton-Raphsonin menetelmä

Tässä menetelmässä käytetään funktiota approksimoivan suoran yhtälössä suoran kulmakertoimen laskemiseen funktion derivaattaa.. Newton-Raphsonin menetelmän suppeneminen on superlineaarista ja suppenee siis nopeammin kuin puolitus- tai sekanttimenetelmät.
Menetelmä on muutoin samanlainen kuin sekanttimenetelmä paitsi, että suoran yhtälö on

$$y = y_1 + y_1' \, (x - x_1) \ ,$$

joten sen nollakohta ratkaistaan yhtälöstä

$$x = x_1 - \frac{y_1}{y_1'} \ .$$

Sekanttimenetelmää vastaava iterointi on

$$x_{n+1} = x_n - \frac{y_n}{y_n'} \ .$$

Numerical Recipes kirjastossa on aliohjelmat `RTNEWT` ja `RTSAFE`, joissa käytetään Newton-Raphsonin menetelmää. Jälkimmäisessä iterointi pysyttelee annetuissa rajoissa.

2.4.5 Brentin menetelmä; suositeltavin menetelmä

Tämä menetelmä yhdistää puolitusmenetelmän ja parabeliapproksimaation kolmen pisteen (x_a, y_a), (x_b, y_b) ja (x_c, y_c) kautta kulkevalle monotoniselle funktiolle.

$$\begin{aligned} x &= \frac{(y-y_a)(y-y_b)}{(y_c-y_a)(y_c-y_b)} x_c \\ &+ \frac{(y-y_b)(y-y_c)}{(y_a-y_b)(y_a-y_c)} x_a \\ &+ \frac{(y-y_c)(y-y_a)}{(y_b-y_c)(y_b-y_a)} x_b \end{aligned}$$

Asetetaan $y = 0$ ja kirjoitetaan yo. yhtälö muodossa

$$x = x_b + \frac{P}{Q}$$

missä

$$\begin{aligned} P &= S\,[T(R-T)(x_c - x_b) - (1-R)(x_b - x_a)] \\ Q &= (T-1)(R-1)(S-1) \\ R &= y_b/y_c \\ S &= y_b/y_a \\ T &= y_a/y_c \end{aligned}$$

Toivomus on, että x_b on kohtuullisen hyvä approksimaatio nollakohdalle ja P/Q pieni korjaus siihen. Jos Q on liian pieni, niin iterointi suppenee hitaasti. Iterointia nopeutetaan asetamalla rajat P/Q suppenemiselle. Jos suppeneminen ei ole riittävän nopeaa, niin nopeutetaan sitä puolitusmenetelmällä. Numerical Recipes ohjelma `ZBRENT` tekee tämän.

2.4.6 Numeerisia esimerkkejä

Esimerkkinä aliohjelman `RTSAFE` käytöstä etsitään Besselin funktion $J_0(x)$ juuret. Besselin funktio $J_0(x)$ voidaan esittää sarjakehitelmän avulla.

$$J_0(x) = \sum_{s=0}^{\infty} \frac{(-1)^s}{(s!)^2} \left(\frac{x}{2}\right)^{2s}$$

Numerical Recipes kirjaston funktiot `BESSJ0` ja `BESSJ1` laskevat Besselin funktioiden $J_0(x)$ ja $J_1(x)$ arvot pisteessä x. Tarvittava derivaatta on

$$\frac{dJ_0(x)}{dx} = -J_1(x).$$

Pääohjelma

```
      PROGRAM D9R9
C Driver for routine RTSAFE
      EXTERNAL FUNCD,BESSJ0
      PARAMETER(N=100,NBMAX=20,X1=1.0,X2=50.0)
      DIMENSION XB1(NBMAX),XB2(NBMAX)
      NB=NBMAX
      CALL ZBRAK(BESSJ0,X1,X2,N,XB1,XB2,NB)
      WRITE(*,'(/1X,A)') 'Roots of BESSJ0:'
      WRITE(*,'(/1X,T19,A,T31,A/)') 'x','F(x)'
      DO 11 I=1,NB
        XACC=(1.0E-6)*(XB1(I)+XB2(I))/2.0
        ROOT=RTSAFE(FUNCD,XB1(I),XB2(I),XACC)
        WRITE(*,'(1X,A,I2,2X,F12.6,E16.4)')
     1 'Root ',I,ROOT,BESSJ0(ROOT)
11    CONTINUE
      END
```

ja funktion arvot lasketaan aliohjelmalla:

```
      SUBROUTINE FUNCD(X,FN,DF)
      FN=BESSJ0(X)
      DF=-BESSJ1(X)
      RETURN
      END
```

2.5 Numeerinen integrointi

Numeerisessa integroinnissa lasketaan määrätyn integraalin

$$I = \int_a^b f(x)\,d\,x$$

arvo. Usein tulos halutaan jollakin ennalta määrätyllä tarkkuudella, jolloin algoritmissa on kiinnitettävä erityistä huomiota x:n arvojen valintaan.

2.5.1 Puolisuunnikaskaava

Yksinkertaisin tapa laskea määrätty integraali on käyttää puolisuunnikaskaavaa (trapezoidal rule):

- Funktion arvot tunnetaan pisteissä (x_i, y_i), $\quad i = 0, \ldots, n$, ja x:n arvot on taulukoitu tasavälein, $x_i = a + i\,h$. Askeleen pituus on h.
- Approksimoidaan funktiota kahden peräkkäisen pisteen välillä suoralla.
- Yhden tällaisen puolisuunnikkaan pinta-ala on

$$A = \frac{h}{2}(y_i + y_{i+1}) = \frac{h}{2}\Big[f(x_i) + f(x_{i+1})\Big] + O(h^3 f''(\xi))$$

Kun kaikkien puolisuunnikkaiden alat lasketaan yhteen saadaan tulokseksi

$$I = \int_a^b f(x)\,dx = h \sum_{i=0}^{n} w_i\, y_i,$$

missä

$$x_i = a + i\,h, \quad \text{kun} \quad h = \frac{b-a}{n}$$

Painokertoimet ovat

$$w_0 = w_n = \frac{1}{2}$$

ja

$$w_i = 1, \quad \text{kun} \quad i = 1, \ldots, n-1\ .$$

Tuloksen tarkkuutta voi parantaa kasvattamalla laskentapisteiden lukumäärää.

Aliohjelma määrätyn integraalin laskemiseksi puolisuunnikaskaavalla:

```
C trapez.f
C Lasketaan tasavalein taulukoidun funktion
C integraalin arvo res kun valin pituus h, pisteita n
C ja y on taulukko joka sisaltaa funktion arvot
      subroutine trapez(h, y, n, res)
      integer i, n
      real    h, res, y(n), sum
      sum = 0.5 * y(1)
      do i = 2, n-1
        sum = sum + y(i)
      end do
      res = h * ( sum + 0.5 * y(n))
      return
      end
```

Vastaavasti funktion `FUNC` määrätty integraali voidaan laskea Numerical Recipes kirjaston ohjelmalla `TRAPZD`, joka laskee n:nnen tarkennuksen integraalin arvoon. Ensimmäisessä arviossa, kun $n = 1$, oletetaan, että funktio on koko integrointivälillä suora, joka kulkee alku- ja loppupisteiden $(a, f(a))$ ja $(b, f(b))$ kautta. Parannettu arvio saadaan, kun $n = 2, 3, \ldots$ lisäämällä 2^{n-2} välipistettä tasavälein.

```
subroutine trapzd(func,a,b,s,n)
integer n
real a,b,s,func
external func
integer it, j
real del, sum, tnm, x
if (n.eq.1) then
  s=0.5*(b-a)*(func(a)+func(b))
else
  it = 2**(n-2)
  tnm=it
  del=(b-a)/tnm
  x=a+0.5*del
  sum=0.
  do j=1,it
    sum=sum+func(x)
    x=x+del
  end do
  s=0.5*(s+(b-a)*sum/tnm)
endif
return
end
```

Usein halutaan tulos jollakin ennalta määrätyllä tarkkuudella `EPS`. Seuraava Numerical Recipes kirjaston ohjelma laskee integraalin tarkkuudella 10^{-6} tai kun 20 parannusta on tehty.

```
subroutine qtrap(func,a,b,s)
integer jmax,j
real a,b,func,s,eps,j
parameter (EPS=1.e-6, jmax=20)
olds=-1.e30
do j=1,jmax
  call trapzd(func,a,b,s,j)
  if (abs(s-olds).lt.EPS*abs(olds)) return
  olds=s
end do
pause 'too many steps.'
end
```

2.5.2 Simpsonin kaava

Tarkkuutta voidaan yrittää parantaa myös approksimoimalla funktiota lineaarista korkeamman asteen polynomilla taulukointipisteiden välillä. Kun approksimointiin käytetään Lagrangen interpolaatiopolynomia saadaan tulokseksi ns. Newton-Coatesin integrointikaavat. Toisen asteen polynomia (parabelia) vastaavaa kaavaa kutsutaan **Simpsonin kaavaksi**.

Johdetaan Simpsonin kaava tasavälein h taulukoidulle funktiolle.

Lasketaan ensin kolmen pisteen $(x_1, y_1), (x_2, y_2)$ ja (x_3, y_3) kautta kulkevan parabelin integraali

$$f(x_2 + ph) = \frac{1}{2}\left[p(p-1)\,y_1 - 2(p^2-1)\,y_2 + p(p+1)\,y_3\right] ,$$

missä $p = (x - x_2)/h$,

$$\begin{aligned}\int_{x_1}^{x_3} f(x)dx &= h\int_{-1}^{1} f(x_2+ph)dp \\ &= \frac{h}{3}(y_1 + 4y_2 + y_3) + O(h^5 f^{(4)}(\xi)) \ .\end{aligned}$$

Kun kaikki välit lasketaan yhteen, saadaan tulokseksi

$$I = \int_a^b f(x)\,dx = h\sum_{i=0}^{n} w_i\,y_i \ ,$$

$$x_i = a + i\,h, \quad \text{kun} \quad h = \frac{b-a}{n}$$

ja painokertoimet ovat

$$w_0 = w_n = \frac{1}{3}$$

$$w_i = \frac{4}{3}, \quad \text{kun} \quad i = 1, 3, \ldots$$

$$w_i = \frac{2}{3}, \quad \text{kun} \quad i = 2, 4, \ldots$$

On huomattava, että pisteiden lukumäärän on oltava **pariton**, jotta päätepisteet tulevat oikein käsitellyksi. Tuloksen tarkkuuta voi jälleen parantaa kasvattamalla laskentapisteiden lukumäärää.

Ohjelma taulukoidun funktion integraalin laskemiseksi Simpsonin kaavalla:

```
C simps.f
C Lasketaan tasavalein taulukoidun funktion
C integraalin arvo res (valin pituus h, pisteita n)
C Pisteiden lukumäärä pariton
      subroutine simps(h, y, n, res)
      integer i, n
      real    h, res, y(n), sum, w
      sum =  y(1)
      w = 2.0
      do i = 2, n-1
          w = 6.0 - w
          sum = sum + w* y(i)
      end do
      res = h * ( sum + y(n))/ 3.0
      return
      end
```

Numerical Recipes kirjastossa on aliohjelma `QSIMP`, jossa käytetään hyväksi kaavaa

$$S = \frac{4}{3}S_{2N} \ - \ \frac{1}{3}S_N$$

missä osasummien S_{2N} (askeleen pituus h) sekä S_N (askeleen pituus $2h$) laskemiseen käytetään $2N$ ja N pisteen puolisuunnikaskaavoja.

$$\begin{aligned} S &= \frac{4}{3}h(\frac{1}{2}y_1 + y_2 + y_3 + y_4 + y_5 + \ldots) \\ &- \frac{1}{3}2h(\frac{1}{2}y_1 + y_3 + y_5 + \ldots) \\ &= \frac{1}{3}h(y_1 + 4y_2 + 2y_3 + 4y_4 + \ldots) \end{aligned}$$

2.5.3 Rombergin integrointi

Eulerin summakaava (vuodelta 1738) esittää puolisuunnikaskaavan virhetermin askeleen pituuden **parillisten potenssien** potenssisarjana

$$\begin{aligned} S &= \int_{x_1}^{x_N} f(x)dx \\ &= h(\frac{1}{2}y_1 + y_2 + y_3 + y_4 + y_5 + \ldots) \\ &- C_2h^2 - C_4h^4 - \ldots - C_{2k}h^{2k} - \ldots \ , \end{aligned}$$

missä

$$C_{2k} = \frac{B_{2k}}{(2k)!}(f_N^{(2k-1)} - f_1^{(2k-1)})$$

ja B_{2k} ovat Bernoullin lukuja.

Tavoitteena Rombergin integroinnissa on saada integraalin arvolle $S(h)$ arvio, kun $h = 0$. Tähän käytetään **Richardsonin menetelmää**, jossa lasketaan S askeleen pituuksilla $h, h/2, h/4, \ldots, h/2^n$, muodostetaan näiden pisteiden kautta kulkeva polynomi ja lasketaan sen arvo, kun askeleen pituus on nolla.
Simpsonin kaava on saatu siten, että lasketaan integraalin arvot $S(2h)$ ja $S(h)$ askeleen pituuksilla $2h$ ja h ja lasketaan näiden pisteiden kautta kulkevan suoran ja y-akselin leikkaus,

$$S = S(2h) + \frac{S(h) - S(2h)}{h^2 - 4h^2}(0 - 4h^2) = \frac{4}{3}S(h) - \frac{1}{3}S(2h) \ .$$

Kuten edellä korostettiin, suora lasketaan h^2:n funktiona.
Korkeamman asteen interpoloivan polynomin muodostamiseen Numerical Receipes käyttää Neville'n algoritmia ja POLINT-ohjelmaa. Tämä parantaa integraalin arvon suppenemista ja antaa arvion integraalin tarkkuudesta.

```
      SUBROUTINE QROMB(FUNC,A,B,SS)
      PARAMETER(EPS=1.E-6,JMAX=20
     1    ,JMAXP=JMAX+1,K=5,KM=4)
      DIMENSION S(JMAXP),H(JMAXP)
      H(1)=1.
      DO J=1,JMAX
        CALL TRAPZD(FUNC,A,B,S(J),J)
        IF (J.GE.K) THEN
          L=J-KM
          CALL POLINT(H(L),S(L),K,0.,SS,DSS)
```

```
      IF (ABS(DSS).LT.EPS*ABS(SS)) RETURN
    ENDIF
    H(J+1)=0.25*H(J)
  END DO
  PAUSE 'Too many steps.'
  END
```

2.5.4 Spline-integrointi

Oletetaan, että funktion $f(x)$ cubic spline interpolaatio

$$y(x) \approx s(x) = y_i + b_i\,(x - x_i) + c_i\,(x - x_i)^2 + d_i\,(x - x_i)^3$$

tunnetaan, kun $x_i \le x \le x_{i+1}$ ja $i = 1, \ldots, n$.
Silloin integraali välin $[x_i, x_{i+1}]$ yli voidaan laskea,

$$\begin{aligned} & \int_{x_i}^{x_{i+1}} y(x)\,dx \approx \int_{x_i}^{x_{i+1}} s(x)\,dx \\ = \;& \Delta x_i\,y_i + \frac{1}{2}\,(\Delta x_i)^2\,b_i + \frac{1}{3}\,(\Delta x_i)^3\,c_i + \frac{1}{4}\,(\Delta x_i)^4\,d_i \end{aligned}$$

missä $\Delta x_i = x_{i+1} - x_i$.

Kertoimet laskettiin jo edellä

$$\begin{aligned} d_i &= \frac{1}{6\Delta x_i}[s_i''(x_{i+1}) - s_i''(x_i)] \\ c_i &= \frac{1}{2}s_i''(x_i) \\ b_i &= \frac{1}{\Delta x_i}[y_{i+1} - y_i] \\ &- \frac{\Delta x_i}{6}[s_i''(x_{i+1}) + 2s_i''(x_i)]\ . \end{aligned}$$

Kun nämä sijoitetaan integraalin lausekkeeseen, niin lopputulos voidaan silloin kirjoittaa muotoon,

$$\begin{aligned} & \int_{x_1}^{x_n} y(x)\,dx \approx \sum_{i=1}^{n-1} \int_{x_i}^{x_{i+1}} y(x)\,dx \\ = \;& \frac{1}{2}\sum_{i=1}^{n-1} \Delta x_i\big[\,y_i + y_{i+1} - \frac{1}{12}\,\Delta x_i^2\,(\,y_i'' + y_{i+1}'')\big]\ . \end{aligned}$$

2.5.5 Gaussin integrointikaavat

Funktioille, joita voidaan approksimoida hyvin jollakin tunnetulla ortogonaalipolynomilla halutulla integrointivälillä, voidaan käyttää Gaussin integrointikaavoja.
Gaussin integrointikaavoissa esiintyy tunnettu painofunktio $W(x)$, jonka ominaisuuksia hyväksikäyttäen voidaan integraali kirjoittaa summaksi,

$$\int_a^b W(x)\,f(x)\,dx \approx \sum_{i=1}^{n} w_i\,f(x_i),$$

Painokertoimet w_i ja pisteet x_i määrätään siten, että summa antaa tarkan tuloksen välillä $[a, b]$ kun $f(x)$ on polynomi, jonka asteluku on $2n - 1$ tai pienempi.

Kaavojen johtaminen on monimutkaisempaa kuin Simpson'in säännön johtaminen. Voidaan osoittaa että, pisteet x_i ovat painofunktioon liittyvän ortogonaalisen polynomin nollakohtia. Gaussin integroinnin idean selventämiseksi tarkastellaan kuitenkin yksinkertaistettua tapausta. Esimerkiksi välillä $[-1, 1]$ ja painofunktiolla $W(x) = 1$ yhden pisteen Gaussin kaavan

$$\int_{-1}^{1} f(x)\,dx \approx w_1 f(x_1)$$

tulee antaa tarkka tulos, kun $f(x)$ on mielivaltainen astetta $2 \cdot 1 - 1 = 1$ (tai 0) oleva polynomi

$$f(x) = bx + c.$$

On siis oltava

$$\int_{-1}^{1} (bx + c)\,dx = 2c = w_1 f(x_1).$$

Ehto toteutuu, kun valitaan $x_1 = 0$ ja $w_1 = 2$. Gaussin integraalikaava on tällöin

$$\int_{-1}^{1} f(x)dx \approx \int_{-1}^{1} (bx + c)dx = 2c = 2f(0).$$

Gaussin kaavat ovat tämän idean yleistyksiä korkeamman asteen polynomeille.
Kun painofunktiona on $W(x) = 1$ ja rajat ovat $a = -1, b = 1$, niin ortogonaalipolynomi on Legendren polynomi, ja jos taas painofunktiona on

$$W(x) = \frac{1}{\sqrt{1 - x^2}}$$

polynomina on Chebyshevin polynomi.

Esimerkki: Laske integraalin

$$\int_{-1}^{1} \frac{e^{-\cos^2(x)}}{\sqrt{1 - x^2}} dx \approx \sum_{i=1}^{n} w_i e^{-\cos^2(x_i)}$$

arvo.
Painofunktio on

$$W(x) = \frac{1}{\sqrt{1 - x^2}}\ ,$$

Chebyshevin polynomien nollakohdat

$$x_i = \cos(\frac{2i - 1}{2}\frac{\pi}{n})$$

ja niitä vastaavat painot

$$w_i = \frac{\pi}{n}$$

```
C gausch.f
C Gauss- Chebyshev integrointi
      subroutine gausch()
      integer n, i
      real    pi, w, x, sum, arg
      pi = 4.0*atan(1.0)
      w = pi / real(n)
      sum = 0.0
      do 10 i = 1,n
        x = cos(0.5* (2* i -1) *w)
        arg = cos(x)
        sum = sum + w * exp (-arg *arg)
  10  continue
      return
      end
```

Usein Gaussin kaavoissa integrointirajat ovat erilaiset kuin laskettavana olevassa integraalissa. Silloin on tehtävä muuttujan vaihto; esimerkiksi muuttujan vaihdolla

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

voidaan integraali yli välin $[a, b]$ muuttaa integraaliksi yli välin $[-1, 1]$

$$\int_a^b \frac{f(x)}{\sqrt{(x-a)(b-x)}}\, dx = \int_{-1}^{1} \frac{f(x(t))}{\sqrt{1-t^2}}\, dt \approx \sum_{i=1}^{n} w_i\, f(x_i)\ ,$$

missä

$$x_i = \frac{b+a}{2} + \frac{b-a}{2}\ \cos(\frac{2i-1}{2}\,\frac{\pi}{n})\ .$$

Tavallisimmat Gaussin integraalityypit:

$$\int_a^b W(x)\, f(x)\, dx \approx \sum_{i=1}^{n} w_i\, f(x_i)\ ,$$

missä

rajat a, b	painofunktio $W(x)$	Ortogonaalipolynomi
-1,1	1	Legendre
-1,1	$\frac{1}{\sqrt{1-x^2}}$	Chebyshev
0,∞	$x^c e^{-x}$	Laquerre, $c = 0, 1, \ldots$
-∞, ∞	e^{-x^2}	Hermite

Numerical Recipes kirjastossa on ohjelma `GAULEG`, joka laskee Gauss-Legendren integrointikaavassa tarvittavat pisteet ja painokertoimet välillä $[X1, X2]$ (eli itseasiassa ratkaisee Legendren polynomin juuret Newtonin menetelmällä):

```
      SUBROUTINE GAULEG(X1,X2,X,W,N)
      IMPLICIT REAL*8 (A-H,O-Z)
      REAL*4 X1,X2,X(N),W(N)
      PARAMETER (EPS=3.D-14)
      M=(N+1)/2
      XM=0.5D0*(X2+X1)
      XL=0.5D0*(X2-X1)
      DO 12 I=1,M
        Z=COS(3.141592654D0*(I-.25D0)/(N+.5D0))
1       CONTINUE
          P1=1.D0
          P2=0.D0
          DO 11 J=1,N
            P3=P2
            P2=P1
            P1=((2.D0*J-1.D0)*Z*P2-(J-1.D0)*P3)/J
11        CONTINUE
          PP=N*(Z*P1-P2)/(Z*Z-1.D0)
          Z1=Z
          Z=Z1-P1/PP
        IF(ABS(Z-Z1).GT.EPS)GO TO 1
        X(I)=XM-XL*Z
        X(N+1-I)=XM+XL*Z
        W(I)=2.D0*XL/((1.D0-Z*Z)*PP*PP)
        W(N+1-I)=W(I)
12    CONTINUE
      RETURN
      END
```

2.6 Lineaarinen yhtälöryhmä

Lineaarinen yhtälöryhmä

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

voidaan kirjoittaa matriisimuotoon

$$Ax = b \ ,$$

missä

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix}$$

ja x sekä b ovat pystyvektoreita

$$x = \begin{pmatrix} x_1 \\ x_2 \\ \cdots \\ x_n \end{pmatrix} \ ,$$

$$b = \begin{pmatrix} b_1 \\ b_2 \\ \cdots \\ b_n \end{pmatrix} \ .$$

2.6.1 Gaussin menetelmä

Tarkastellaan lineaarista yhtälöryhmää

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= a_{1n+1} \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= a_{2n+1} \\ & \cdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n &= a_{nn+1} \ . \end{aligned}$$

Merkintöjen yksinkertaistamiseksi on yhtälön oikealle puolelle otettu käyttöön merkintä, $b_i = a_{in+1}$. Ratkaistaan ensin tuntematon x_1 ensimmäisestä yhtälöstä jakamalla kertoimella a_{11}.

$$x_1 = \frac{a_{1n+1}}{a_{11}} - \frac{a_{12}}{a_{11}}\, x_2 - \cdots - \frac{a_{1n}}{a_{11}}\, x_n \ .$$

ja sijoitetaan tämä ratkaisu alempiin yhtälöihin.

Silloin saadaan uusi yhtälöryhmä

$$\begin{aligned} x_1 + c_{12}x_2 + \cdots + c_{1n}x_n &= c_{1n+1} \\ a^{(1)}_{22}x_2 + \cdots + a^{(1)}_{2n}x_n &= a^{(1)}_{2n+1} \\ & \cdots \\ a^{(1)}_{n2}x_2 + \cdots + a^{(1)}_{nn}x_n &= a^{(1)}_{nn+1} \ , \end{aligned}$$

missä kertoimet ovat

$$\begin{aligned} c_{1j} = \frac{a_{1j}}{a_{11}} \quad &, \quad j = 2, \ldots, n+1 \\ a_{kj}^{(1)} = a_{kj} - a_{k1}c_{1j} \quad &, \quad j = 2, \ldots, n+1 \\ &, \quad k = 2, \ldots, n+1 \ . \end{aligned}$$

Seuraavaksi ratkaistaan toisesta yhtälöstä x_2 ja sijoitetaan saatu tulos alempiin yhtälöihin. Jatkamalla samalla tavalla saadaan p:n eliminoinnin jälkeen yhtälöryhmäksi

$$\begin{aligned} x_1 + c_{12}x_2 + \cdots + c_{1p}x_p + \cdots + c_{1n}x_n &= c_{1n+1} \\ x_2 + \cdots + c_{2p}x_p + \cdots + c_{2n}x_n &= c_{2n+1} \\ &\cdots \\ x_p + \cdots + c_{pn}x_n &= c_{pn+1} \\ a_{p+1p+1}^{(p)}x_{p+1} + \cdots + a_{p+1n}^{(p)}x_n &= a_{p+1n+1}^{(p)} \\ &\cdots \\ a_{np+1}^{(p)}x_{p+1} + \cdots + a_{nn}^{(p)}x_n &= a_{nn+1}^{(p)} \end{aligned}$$

Kertoimet lasketaan palautuskaavoista

$$\begin{aligned} c_{ij} &= \frac{a_{ij}^{(i-1)}}{a_{ii}^{(i-1)}} \ , \quad i = 1, \ldots, p; \quad j = i+1, \ldots, n \\ a_{kj}^{(p)} &= a_{kj}^{(p-1)} - a_{kp}^{(p-1)}c_{pj} \ , \\ &\quad k = p+1, \ldots, n; \, j = p+1, \ldots, n \end{aligned}$$

Jatkamalla eliminointia saadaan lopuksi n:n eliminointikierroksen jälkeen tulos

$$\begin{aligned} x_1 + c_{12}x_2 + c_{13}x_2 + \cdots + c_{1n}x_n &= c_{1n+1} \\ x_2 + c_{23}x_2 + \cdots + c_{2n}x_n &= c_{2n+1} \\ &\cdots \\ x_{n-1} + c_{n-1,n}x_n &= c_{n-1n+1} \\ x_n &= c_{nn+1} \end{aligned}$$

Yhtälöryhmän lopullinen ratkaisu saadaan sen jälkeen sijoittamalla aina alemmista yhtälöstä saadut tulokset ylempiin yhtälöihin.

2.6.2 Esimerkki Gaussin menetelmän ohjelmoinnista

Kirjoitetaan ensin aliohjelma `gauss`, joka ratkaisee lineaarisen yhtälöryhmän Gaussin eliminointimenetelmällä

```
C gauss.f
      subrroutine gauss(a,nn,n,x,b)
      integer nn, n, i, j, k
      real    a(nn,nn), x(nn), b(nn)
      do 1 i = 1,n
        a(i, n+1) = b(i)
   1  continue
C
```

```
      do 2 i = 1,n
        do 3 j = i+1, n+1
          a(i,j) = a(i,j)/a(i,i)
   3    continue
        do 4 k = i+1,n
          do 5 j = i+1,n+1
            a(k,j) = a(k,j) - a(k,i)*a(i,j)
   5      continue
   4    continue
   2  continue
C
      x(n) = a(n, n+1)
      do 6 i = n-1,1,-1
        x(i) = a(i,n+1)
        do 7 j = i+1, n
          x(i) = x(i) - a(i,j)*x(j)
   7    continue
   6  continue
      return
      end
```

Pääohjelma aliohjelman `gauss` käyttöä varten voisi olla seuraavanlainen:

```
C paaohjelma gauss.f varten
      program main
      integer nn, n, i, j
      parameter (nn=10)
      real    a(nn,nn), x(nn), b(nn)
      print*, 'Anna dimensio: '
      read*, n
      print*, 'Anna matriisi riveittain: '
      read*, ((a(i,j), j=1,n), i=1,n)
      print*, 'Anna oikea puoli riveittain: '
      read*, (b(i), i=1,n)
C
      call gaus(a, nn, n, x, b)
C
      print*, 'Ratkaisu on: '
      print*, (x(i), i=1,n)
      end
```

Gaussin menetelmässä tarvitaan

$$\sum_{i}^{n}(n-i)^2 \approx \frac{1}{3}n^3$$

kerto- ja yhteenlaskuoperaatiota eliminoinnin suorittamiseen sekä

$$\sum_{i}^{n}(n-i) = \frac{1}{2}n^2$$

kerto- ja yhteenlaskuoperaatiota x:n arvojen ratkaisemiseen.

Gaussin eliminointimenetelmän ongelmana on se, että eliminoitavan tuntemattoman x_k kerroin $a_{kk}^{(k-1)}$ voi olla nolla, jolloin eliminointia ei voida suorittaa, tai se on itseisarvoltaan hyvin pieni, jolloin laskentatarkkuus huononee.

Tästä ongelmasta selviää vaihtamalla kerroinmatriisin rivejä tai sarakkeita (pivoting). Tämähän ei muuta yhtälöryhmän ratkaisua. Stabiilein ratkaisu saadaan, kun jakajaksi valitaan kertoimista suurin.

Seuraava lisäys edelliseen ohjelmaan suorittaa rivien vaihdon.

```
ind = i
do k = i+1, n
   if(abs(a(k,i)).gt.abs(a(ind,i))) ind = k
end do
if (ind.ne.i) then
   do k = i, n
      dum = a(i,k)
      a(i,k) = a(ind,k)
      a(ind,k) = dum
   end do
end if
```

2.6.3 Choleskin menetelmä eli LU-hajoitelma

Choleskin menetelmässä matriisi A hajotetaan ylä- ja alakolmiomatriisien L ja U tuloksi

$$L\ U = A$$

Esimerkiksi 4×4- matriisit kirjoitettaisiin silloin muotoon

$$\begin{pmatrix} L_{11} & 0 & 0 & 0 \\ L_{21} & L_{22} & 0 & 0 \\ L_{31} & L_{32} & L_{33} & 0 \\ L_{41} & L_{42} & L_{43} & L_{44} \end{pmatrix} \qquad \begin{pmatrix} U_{11} & U_{12} & U_{13} & U_{14} \\ 0 & U_{22} & U_{23} & U_{24} \\ 0 & 0 & U_{33} & U_{34} \\ 0 & 0 & 0 & U_{44} \end{pmatrix}$$

$$= \quad \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix}$$

Kun LU-hajoitelma on suoritettu, niin lineaarinen yhtälöryhmä

$$A\ x = (L\ U)\ x = L\ (U\ x) = b$$

voidaan ratkaista laskemalla ensin vektori y yhtälöryhmästä

$$L\ y = b$$

ja ratkaisemalla sen jälkeen yhtälöryhmä

$$U\ x = y\ .$$

Näiden kolmiollisten yhtälöryhmien ratkaiseminen on helppoa sijoitusmenetelmiä käyttäen.

Vektorin y komponentit ratkaistaan palautuskaavoilla

$$\begin{aligned} y_1 &= \frac{b_1}{L_{11}} \\ y_i &= \frac{1}{L_{ii}}\Big[b_i - \sum_{j=1}^{i-1} L_{ij}\, y_j \Big] \ , \\ & \qquad \text{kun} \quad i = 2, 3, \ldots, n \ , \end{aligned}$$

ja vektorin x komponentit suorittamalla sijoitus takaperin,

$$\begin{aligned} x_n &= \frac{y_n}{U_{nn}} \\ x_i &= \frac{1}{U_{ii}}\Big[y_i - \sum_{j=i+1}^{n} U_{ij}\, x_j \Big] \ , \\ & \qquad \text{kun} \quad i = n-1, n-2, \ldots, 1 \ . \end{aligned}$$

2.6.4 Crout'in algoritmi LU-hajoitelmalle

Tarkastellaan yhtälön $LU = A$ komponenttia ij,

$$L_{i1}U_{1j} + L_{i2}U_{2j} + \ldots = a_{ij}$$

Ylä- ja alakolmiomatriisien määritelmistä seuraa, että

$$\begin{aligned} L_{ik} &= 0, \quad \text{kun} \ \ i < k \\ U_{kj} &= 0, \quad \text{kun} \ \ k > j \ . \end{aligned}$$

Voidaan erottaa kolme tapausta,

(1) $i < j$

$$L_{i1}U_{1j} + L_{i2}U_{2j} + \ldots + L_{ii}U_{ij} = a_{ij}$$

(2) $i = j$

$$L_{i1}U_{1j} + L_{i2}U_{2j} + \ldots + L_{ii}U_{jj} = a_{ij}$$

(3) $i > j$

$$L_{i1}U_{1j} + L_{i2}U_{2j} + \ldots + L_{ij}U_{jj} = a_{ij}$$

Yhtälöitä on n^2 kpl ja tuntemattomia $n^2 + n$ kpl.

Valitaan diagonaalielementit matriisissa L ykkösiksi

$$L_{ii} = 1, \quad i = 1, \ldots, n \ .$$

Muut matriisielementit saadaan ratkaistua jokaiselle j:n arvolle erikseen palautuskaavoista seuraavasti:

- Lasketaan U_{ij} yhtälöistä (1) ja (2), kun $i = 1, \ldots, j$,

$$U_{ij} = a_{ij} - \sum_{k=1}^{i-1} L_{ik}U_{kj} \ .$$

  Jos $i = 1$, niin summa on nolla.

- Kun $i = j+1, j+2, \ldots, n$, niin käytetään yhtälöä (3) matriisielementtien L_{ij} laskemiseen,

$$L_{ij} = \frac{1}{U_{jj}} \Big(a_{ij} - \sum_{k=1}^{j-1} L_{ik} U_{kj} \Big) \ .$$

Tämä algoritmi muodostaa LU-hajoitelman sarakkeittain siten, että tulos voidaan sijoittaa alkuperäisen matriisin päälle tietokoneen muistissa,

$$\begin{pmatrix} U_{11} & U_{12} & U_{13} & \ldots & U_{1n} \\ L_{21} & U_{22} & U_{23} & \ldots & U_{2n} \\ L_{31} & L_{32} & U_{33} & \ldots & U_{3n} \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ L_{n1} & L_{n2} & L_{n3} & \ldots & U_{nn} \end{pmatrix} \ .$$

LU-hajoitelman muodostaminen vaatii $N^3/3$ kerto- ja yhteenlaskua, mikä on sama kuin Gaussin menetelmässä.
Numerical Recipes kirjassa on aliohjelma `LUDCMP` LU- hajoitelman tekemiseen ja `LUBKSB` ratkaisuvektoreiden y ja x laskemiseen.

```
      SUBROUTINE LUBKSB(A,N,NP,INDX,B)
      DIMENSION A(NP,NP),INDX(N),B(N)
      II=0
      DO 12 I=1,N
        LL=INDX(I)
        SUM=B(LL)
        B(LL)=B(I)
        IF (II.NE.0)THEN
          DO 11 J=II,I-1
            SUM=SUM-A(I,J)*B(J)
11        CONTINUE
        ELSE IF (SUM.NE.0.) THEN
          II=I
        ENDIF
        B(I)=SUM
12    CONTINUE
      DO 14 I=N,1,-1
        SUM=B(I)
        IF(I.LT.N)THEN
          DO 13 J=I+1,N
            SUM=SUM-A(I,J)*B(J)
13        CONTINUE
        ENDIF
        B(I)=SUM/A(I,I)
14    CONTINUE
      RETURN
      END
      SUBROUTINE LUDCMP(A,N,NP,INDX,D)
      PARAMETER (NMAX=100,TINY=1.0E-20)
      DIMENSION A(NP,NP),INDX(N),VV(NMAX)
      D=1.
      DO 12 I=1,N
        AAMAX=0.
```

```
        DO 11 J=1,N
          IF (ABS(A(I,J)).GT.AAMAX) AAMAX=ABS(A(I,J))
11      CONTINUE
        IF (AAMAX.EQ.0.) PAUSE 'Singular matrix.'
        VV(I)=1./AAMAX
12    CONTINUE
      DO 19 J=1,N
        IF (J.GT.1) THEN
          DO 14 I=1,J-1
            SUM=A(I,J)
            IF (I.GT.1)THEN
              DO 13 K=1,I-1
                SUM=SUM-A(I,K)*A(K,J)
13            CONTINUE
              A(I,J)=SUM
            ENDIF
14        CONTINUE
        ENDIF
        AAMAX=0.
        DO 16 I=J,N
          SUM=A(I,J)
          IF (J.GT.1)THEN
            DO 15 K=1,J-1
              SUM=SUM-A(I,K)*A(K,J)
15          CONTINUE
            A(I,J)=SUM
          ENDIF
          DUM=VV(I)*ABS(SUM)
          IF (DUM.GE.AAMAX) THEN
            IMAX=I
            AAMAX=DUM
          ENDIF
16      CONTINUE
        IF (J.NE.IMAX)THEN
          DO 17 K=1,N
            DUM=A(IMAX,K)
            A(IMAX,K)=A(J,K)
            A(J,K)=DUM
17        CONTINUE
          D=-D
          VV(IMAX)=VV(J)
        ENDIF
        INDX(J)=IMAX
        IF(J.NE.N)THEN
          IF(A(J,J).EQ.0.)A(J,J)=TINY
          DUM=1./A(J,J)
          DO 18 I=J+1,N
            A(I,J)=A(I,J)*DUM
18        CONTINUE
        ENDIF
19    CONTINUE
      IF(A(N,N).EQ.0.)A(N,N)=TINY
```

```
      RETURN
      END
```

2.6.5 Käänteismatriisin A^{-1} laskeminen

Kun LU-hajoitelma on tehty, niin käänteismatriisin matriisielementit voidaan laskea sarake kerrallaan, kuten lineaarisesta yhtäloryhmästä.

$$A = LU \quad ; \quad AA^{-1} = I,$$

missä I on yksikkömatriisi, joten

$$LUA^{-1} = I.$$

Käänteismatriisi lasketaan sarake kerrallaan yhtälöstä

$$LUx = b$$

sijoittamalla oikeaksi puoleksi vuoron perään yksikkövektorit. Ratkaisuvektorit x muodostavat käänteismatriisin.

Fortran ohjelma näyttäisi seuraavalta:

```
      ⋮
      do i = 1, n
          do j = 1, n
             y(i,j) = 0.0
          end do
          y(i,i) = 1.0
      end do
      call ludcmp(a,n,np,indx,d)
      do j = 1, n
          call lubksb(a,n,np,indx,y(1,j))
      end do
```

2.6.6 Matriisin determinantin laskeminen

Matriisien tulon determinantti on tekijöiden determinanttien tulo,

$$\det A \ = \det L \ \det U \ .$$

Koska $\det L \ = \prod_{j=1}^{n} L_{jj} = 1$, niin $\det A \ = \prod_{j=1}^{n} U_{jj}$, joten determinantti voidaan laskea suoraan LU-hajoitelmasta ja fortran ohjelma näyttäisi seuraavalta:

```
      ⋮
      call ludcmp(a,n,np,indx,d)
      do j = 1, n
          d = d * a(j,j)
      end do
      ⋮
```

2.7 Matriisin ominaisarvot

Ratkaistavana on lineaarinen homogeeninen yhtälöryhmä

$$Ax = \lambda x \ ,$$

missä A on $n \times n$ matriisi ja λ on matriisin ominaisarvo. Ominaisarvoja λ on n kpl, joista osa voi olla samojakin. Kutakin ominaisarvoa vastaa ominaisvektori x.

Yhtälöryhmällä

$$(A - \lambda I)x = 0 \ ,$$

missä I on yksikkömatriisi, on ei-triviaali ratkaisu, jos kerroindeterminantti on nolla,

$$\det |A - \lambda I| = 0 \ .$$

Tämä yhtälö on n:nnen asteen polynomi λ:n suhteen, ja sen juuret ovat yhtälön ominaisarvoja. Niitä on n kpl, joista kuitenkin osa voi olla samoja ($\Rightarrow$ ominaisarvot voivat olla degeneroituneita).

2.7.1 Määritelmiä

Matriisi on

- symmetrinen, jos $A = A^T$
- hermiittinen, jos $A = A^\dagger$
- ortogonaalinen, jos $AA^T = A^T A = I$, joten $A^{-1} = A^T$
- unitaarinen, jos $AA^\dagger = A^\dagger A = I$, joten $A^{-1} = A^\dagger$
- normaali, jos se kommutoi Hermiten konjugaatti matriisinsa kanssa, $AA^\dagger = A^\dagger A$.

2.7.2 Matriisin diagonalisointi

Määritellään oikean- ja vasemmanpuoleiset ominaisvektorit.

$$\begin{aligned} Ax &= \lambda x \\ yA &= \bar{\lambda} y \end{aligned}$$

Transponoimalla $[(A\ B)^T = B^T A^T]$ saadaan

$$\begin{aligned} x^T A^T &= \lambda^T x^T \\ A^T y^T &= \bar{\lambda}^T y^T \end{aligned}$$

Koska $\det A = \det A^T$, niin

$$\det |A^T - \bar{\lambda}^T I| = \det |A - \bar{\lambda} I| = 0 \ .$$

Koska matriisin A ominaisarvot λ saadaan yhtälön myös $\det |A - \lambda I| = 0$ ratkaisuna, niin $\bar{\lambda} = \lambda$ ja oikean- ja vasemmanpuoleiset ominaisarvot ovat samat.
Jos matriisi on symmetrinen, niin $A = A^T$, ja ominaisvektoreilla y^T ja x on samat komponentit.

Muodostetaan matriisi X_R siten, että matriisin sarakkeina ovat oikeanpuoleiset ominaisvektorit ja matriisi X_L siten, että sen riveinä ovat vasemmanpuoleiset ominaisvektorit. Tällöin

$$\begin{aligned} A\ X_R &= X_R\ \mathrm{diag}(\lambda_1, \cdots, \lambda_n) \\ X_L\ A &= \mathrm{diag}(\lambda_1, \cdots, \lambda_n)\ X_L\ . \end{aligned}$$

Merkinnällä $\mathrm{diag}(\lambda_1, \cdots, \lambda_n)$ tarkoitetaan diagonaalimatriisia, jonka diagonaalilla ovat luvut $\lambda_1, \cdots, \lambda_n$.

Yhtälöistä ensimmäinen kerrotaan vasemmalta X_L:llä ja toinen oikealta X_R:llä, jolloin nähdään, että

$$(X_L\ X_R)\ \mathrm{diag}(\lambda_1, \cdots, \lambda_n) = \mathrm{diag}(\lambda_1, \cdots, \lambda_n)\ (X_L\ X_R)$$

Matriisi $(X_L\ X_R)$ kommutoi siis diagonaalisen matriisin kanssa, jolla on erilliset alkiot diagonaalilla (olettaen, että ominaisarvot ovat keskenään eri suuria).
$\Rightarrow$ Matriisi $(X_L\ X_R)$ on diagonaalinen matriisi.
Ominaisvektoreiden normitus voidaan valita siten, että niiden pituus on yksi, joten $X_L = X_R^{-1}$.
$\Rightarrow$ Matriisi A diagonalisoidaan muunnoksella

$$X_R^{-1} A\ X_R = \mathrm{diag}(\lambda_1, \cdots, \lambda_n)$$

Similariteettimuunnoksessa

$$A \to Z^{-1}\ A\ Z$$

matriisin ominaisarvot säilyvät muuttumattomina.

$$\begin{aligned} \det|Z^{-1}\ A\ Z - \lambda I| &= \det|Z^{-1}(A - \lambda I)Z| \\ &= \det|Z|\ \det|A - \lambda I|\ \det|Z^{-1}| \\ &= \det|A - \lambda I|\ , \end{aligned}$$

koska $\det|Z^{-1}| = 1/\det|Z|$.
Täten, mikä tahansa täydellisen ominaisvektorijoukon omaava matriisi voidaan diagonalisoida similariteettimuunnoksella siten, että oikeanpuoleiset ominaisvektorit muodostavat matriisin sarakkeet ja vasemmanpuoleiset ominaisvektorit sen käänteismatriisin rivit.

Numeerisesti diagonalisointi tapahtuu siten, että suoritetaan sarja peräkkäisiä similariteettimuunnoksia, jotka diagonalisoivat matriisin halutulla tarkkuudella.

$$\begin{aligned} A &\to P_1^{-1}\ A\ P_1 \to \\ &\to P_2^{-1} P_1^{-1}\ A\ P_1 P_2 \\ &\to P_3^{-1} P_2^{-1} P_1^{-1}\ A\ P_1 P_2 P_3 \to \ldots \end{aligned}$$

Ominaisvektorit saadaan matriisin

$$X_R = P_1 P_2 P_3 \ldots$$

sarakkeista.

2.7.3 Jacobin muunnos symmetriselle matriisille

Jacobin muunnoksessa tehdään sarja ortogonaalisia similariteettimuunnoksia. Jokainen näistä muunnoksista vastaa tason kiertoa ja niillä nollataan aina yksi ei-diagonaalinen matriisin alkio kerrallaan. Tarkastellaan esimerkkinä symmetristä 3×3 matriisia,

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{12} & a_{22} & a_{23} \\ a_{13} & a_{23} & a_{33} \end{pmatrix} ,$$

ja suoritetaan sille muunnos P_{23}, jolla eliminoidaan alkio 2,3.

$$P_{23} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & c & s \\ 0 & -s & c \end{pmatrix} .$$

Jos ajatellaan tason kiertoa, niin alkio $c = \cos(\phi)$ ja $s = \sin(\phi)$, missä ϕ on kiertokulma.

Muunnettu matriisi A' on muotoa

$$A' = P_{23}^T A P_{23} = \begin{pmatrix} a'_{11} & a'_{12} & a'_{13} \\ a'_{12} & a'_{22} & a'_{23} \\ a'_{13} & a'_{23} & a'_{33} \end{pmatrix}$$

missä

$$\begin{aligned} a'_{11} &= a_{11} \\ a'_{12} &= c\ a_{12} - s\ a_{13} \\ a'_{13} &= c\ a_{13} + s\ a_{12} \\ a'_{22} &= c^2 a_{22} + s^2\ a_{33} - 2s\ c\ a_{23} \\ a'_{33} &= s^2 a_{22} + c^2\ a_{33} + 2s\ c\ a_{23} \\ a'_{23} &= (c^2 - s^2)\ a_{23} + s\ c\ (a_{22} - a_{33}) \end{aligned}$$

Huom. Ortogonaalimatriiseille $P^{-1} = P^T$.

Asetetaan $a'_{23} = 0$. Tällöin

$$\theta \equiv \cot(2\phi) = \frac{c^2 - s^2}{2s\ c} = \frac{a_{33} - a_{22}}{2a_{23}}$$

Merkitään $t \equiv s/c = \tan(\phi)$, jolloin θ'n määritelmä voidaan kirjoittaa muotoon.

$$t^2 + 2t\theta - 1 = 0$$

Tämän yhtälön juurista pienempi vastaa kiertoa $\phi < |\pi/4|$.

$$t = \frac{\mathrm{sgn}(\theta)}{|\theta| + \sqrt{\theta^2 + 1}}$$

Tämä valinta antaa stabiilimman iteraatiopolun diagonalisointiin. Jos θ on niin suuri, että θ^2 aiheuttaa ylivuodon, niin asetetaan $t = 1/2\theta$. Kierrossa tarvittavat c ja s saadaan siten ratkaistua,

$$\begin{aligned} c &= \frac{1}{\sqrt{t^2 + 1}} \\ s &= tc\ . \end{aligned}$$

Muunnetun matriisin alkiosta $a'_{23} = 0$ ja muut alkiot saadaan entisistä alkioista plus pienistä muutoksista niihin,

$$\begin{aligned} a'_{11} &= a_{11} \\ a'_{12} &= a_{12} - s\ (a_{13} + \tau a_{12}) \\ a'_{13} &= a_{13} + s\ (a_{12} - \tau a_{13}) \\ a'_{22} &= a_{22} + t\ a_{23} \\ a'_{33} &= a_{33} + t\ a_{23}\ , \end{aligned}$$

missä

$$\tau \equiv \tan(\phi/2) = \frac{s}{1+c}$$

Tarkastellaan muunnoksen P_{23} vaikutusta ei-diagonaalisten elementtien neliöiden summaan, $S = \sum_{r \neq s} |a_{rs}|^2$.

$$\begin{aligned} |a'_{12}|^2 + |a'_{13}|^2 &= |a_{12}|^2 + |a_{13}|^2 \\ |a'_{23}| &= 0 \end{aligned}$$

Summan arvo pienenee muunnoksessa poistuneesta alkiosta johtuen,

$$S' = S - 2|a_{23}|^2\ .$$

Kerroin kaksi johtuu siitä, että matsiisi A on symmetrinen.

Muunnos P_{23} on ortogonaalinen, joten kaikkien alkioiden neliöiden summan täytyy säilyä muuttumattomana:

$$\begin{aligned} \sum_{i,j} |a'_{ij}|^2 &= \sum_{i,j} \sum_{k,l} p_{ki} a_{kl} p_{lj} \sum_{k',l'} p_{k'i} a_{k'l'} p_{l'j} \\ &= \sum_{k,l} |a_{kl}|^2\ , \end{aligned}$$

koska ortogonaalimatriiseille $P_{23}\ P_{23}^T = I$,

$$\sum_i p_{ki} p_{k'i} = \delta_{k,k'}\ .$$

Tämän vuoksi diagonaalialkioiden neliöiden summa kasvaa $2|a_{23}|^2$ verran.

Lopulta matriisi on diagonaalinen halutun tarkkuuden puitteissa. Diagonaalinen muoto

$$D = V^T\ A\ V$$

on saavutettu peräkkäisillä Jacobin rotaatioilla

$$V = P_1 P_2 P_3 \ldots$$

Ominaisvektorit ovat matriisin V sarakkeita. Approksimaatio niille voidaan laskea jokaisella iteraatiokierroksella

$$V' = V\ P_i$$

asettamalla alussa $V = I$.

$$\begin{aligned} v'_{11} &= v_{11} \\ v'_{12} &= c\, v_{12} - s\, v_{13} \\ v'_{13} &= s\, v_{12} + c\, v_{13} \\ \vdots &= \vdots \end{aligned}$$

Nämä yhtälöt kannattaa kirjoittaa edellä esitetyllä tavalla parametrin τ avulle päyristysvirheiden minimoimiseksi.

Syklisessä Jacobin menetelmässä eliminoidaan elementti kerrallaan yksinkertaisessa järjestyksessä esimerkiksi riveittäin: $P_{12}, P_{13}, \ldots, P_{1n}, P_{23}, P_{24}, \ldots$. Suppeneminen on yleensä neliöllistä.

2.7.4 Householderin menetelmä

Householderin menetelmässä redusoidaan $n \times n$ symmetrinen matriisi A tridiagonaaliseen muotoon $(n-2)$:lla ortogonaalimuunnoksella. Jokainen muunnos nollaa halutun osan sekä sarakkeesta että vastaavasta rivistä. Householderin menetelmä perustuu muunnokseen P,

$$P = I - 2w \cdot w^T,$$

missä w on reaalinen vektori, jonka pituuden neliö $|w|^2 = w^T \cdot w = 1$.
Ortogonaalisuus:

$$\begin{aligned} P^2 &= (I - 2w \cdot w^T) \cdot (I - 2w \cdot w^T) \\ &= I - 4w \cdot w^T + 4w \cdot (w^T \cdot w) \cdot w^T \\ &= I\ . \end{aligned}$$

Täten $P = P^{-1}$, mutta koska $P^T = P$ niin $P^T = P^{-1}$, joka on ortogonaalisuuden vaatimus.

Kirjoitetaan P muotoon

$$P = I - \frac{u \cdot u^T}{H}\ ,$$

missä H on

$$H \equiv \frac{1}{2}|u|^2$$

ja u on mikä tahansa vektori.
Olkoon x vektori, joka on matriisin A ensimmäinen sarake. Valitaan

$$u = x \pm |x|\hat{e}_1$$

missä $\hat{e}_1$ on yksikkövektori $(1, 0, \ldots, 0)^T$ ja merkin valinta tehdään myöhemmin. Silloin

$$\begin{aligned} P \cdot x &= x - \frac{u}{H} \cdot (x \pm |x|\hat{e}_1)^T \cdot x \\ &= x - \frac{u \cdot (|x|^2 \pm |x|x_1)}{|x|^2 \pm |x|x_1} \\ &= x - u \\ &= \mp |x|\hat{e}_1\ . \end{aligned}$$

Tämä osoittaa, että Householderin matriisi P operoidessaan vektoriin x nollaa kaikki sen komponentit ensimmäistä lukuunottamatta.

Symmetrisen matriisin A,

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{12} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{1n} & a_{2n} & \dots & a_{nn} \end{pmatrix} ,$$

redusoimiseksi tridiagonaaliseen muotoon valitaan ensimmäisessä Householderin matriisissa P_1 vektoriksi x matriisin A ensimmäisen sarakkeen $n-1$ alinta alkiota

$$P_1 = \left(\begin{array}{c|ccc} 1 & 0 & \dots & 0 \\ \hline 0 & & & \\ 0 & & P_1^{(n-1)} & \\ \vdots & & & \\ 0 & & & \end{array}\right) ,$$

missä $P_1^{(n-1)}$ tarkoittaa $(n-1) \times (n-1)$ Householderin matriisia.

Tällöin

$$P_1 \cdot A = \left(\begin{array}{c|ccc} a_{11} & a_{12} & \dots & a_{1n} \\ \hline k & & & \\ 0 & & \text{epäolennaista} & \\ \vdots & & & \\ 0 & & & \end{array}\right)$$

Suure $k = |x| = |a_{12}, a_{13}, \dots, a_{1n}|$.
Ortogonaalimuunnos kokonaisuudessaan

$$A' = P_1 \cdot A \cdot P_1 = \left(\begin{array}{c|ccc} a_{11} & k & \dots & 0 \\ \hline k & & & \\ 0 & & \text{epäolennaista} & \\ \vdots & & & \\ 0 & & & \end{array}\right) ,$$

kun muistetaan, että $P_1^T = P_1$.

Seuraavaksi valitaan vektori x siten, että se sisältää matriisin A toisen sarakkeen $n-2$ alinta alkiota ja tämän vektorin avulla muodostetaan toinen Housholderin matriisi P_2,

$$P_2 = \left(\begin{array}{cc|ccc} 1 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ \hline 0 & 0 & & & \\ 0 & 0 & & {}^{(n-2)}P_2 & \\ \vdots & \vdots & & & \\ 0 & 0 & & & \end{array}\right) ,$$

Vasemmalle ylös muodostuva diagonaalimatriisi takaa sen, että muunnos ei tuhoa edellisellä kierroksella aikaansaatua tridiagonaalimuodon alkua. Täten $(n-2)$ vastaavalla tavalla muodostettua muunnosta saattaa matriisin kokonaisuudessaan tridiagonaalimuotoon.

Sen sijaan, että suorittaisi matriisien $P \cdot A \cdot P$ kertolaskun kannattaa muodostaa vektori

$$p \equiv \frac{A \cdot u}{H}$$

Tällöin

$$A \cdot P = A \cdot (I - \frac{u \cdot u^T}{H}) = A - p \cdot u^T$$

ja

$$A' = P \cdot A \cdot P = A - p \cdot u^T - u \cdot p^T + 2Ku \cdot u^T$$

missä skalaari K on

$$K = \frac{u^T \cdot p}{2H} \ .$$

Jos vielä määritellään suure q

$$q \equiv p - Ku \ ,$$

niin

$$A' = A - q \cdot u^T - u \cdot q^T \ ,$$

mikä on numeerisesti käyttökelpoinen muoto.

Käytännössä Householderin menetelmä alkaa matriisin n:nnestä sarakkeesta eikä ensimmäisestä, kuten edellä selostettiin.
Vaiheessa m $(m = 1, \ldots, n-2)$ vektori u voidaan kirjoittaa muodossa

$$u^T = [a_{i1}, \ldots, a_{i,i-2}, a_{i,i-1} \pm \sqrt{\sigma}, 0, \ldots, 0]$$

missä $i \equiv n - m + 1 = n, n-1, \ldots, 3$ ja

$$\sigma = (a_{i,1})^2 + \ldots + (a_{i,i-1})^2$$

Merkki σ:n edessä on sama kuin termin $a_{i,i-1}$ merkki.
Muuttujat lasketaan seuraavassa järjestyksessä: $\sigma, u, H, p, K, q, A'$. Jokaisessa vaiheessa m matriisi A on tridiagonaalinen ainakin viimeisten $m-1$ rivin ja sarakkeen osalta.

2.7.5 QR ja QL-algoritmit

QR ja QL-algoritmeilla lasketaan tridiagonaalimatriisin ominaisarvot ja ominaisvektorit. Perusajatuksena on, että mikä tahansa reaalinen matriisi voidaan kirjoittaa muodossa

$$A = Q \cdot R \ ,$$

missä Q on ortogonaalimatriisi ja R yläkolmiomatriisi. Tämä muoto saadaan soveltamalla Householderin muunnoksia peräkkäin niin, että matriisin A alakolmion alkiot saadaan nolliksi.
Muodostetaan matriisi

$$A' = R \cdot Q$$

ja todetaan, että $R = Q^T \cdot A$, koska Q on ortogonaalinen. Tällöin

$$A' = Q^T \cdot A \cdot Q$$

eli matriisi A' saadaan ortogonaalimuunnoksella matriisista A.

Vastaavasti voidaan kirjoittaa

$$A = Q \cdot L \ ,$$

missä L on alakolmiomatriisi. Alakolmiomatriisin käyttö osoittautuu numeerisesti stabiilimmaksi, joten sitä kannattaa käyttää yläkolmiomatriisin sijasta.
QL-algoritmi muodostuu sarjasta muunnoksia

$$\begin{aligned} A_s &= Q_s \cdot L_s \\ A_{s+1} &= L_s \cdot Q_s \quad (= Q_s^T A_s Q_s) \end{aligned}$$

Muunnokselle voidaan todistaa kaksi teoreemaa

1. Jos matriisilla A on ominaisarvot λ_i, joiden itseisarvot ovat erisuuret, niin $A_s \to$ alakolmiomatriisin muotoa, kun $s \to \infty$. Tällöin ominaisarvot ilmaantuvat diagonaalille itseisarvoltaan kasvavassa järjestyksessä.

2. Jos matriisilla A on p kertainen ominaisarvo $|\lambda_i|$, niin $A_s \to$ alakolmiomatriisin muotoa, kun $s \to \infty$ paitsi diagonaalille muodostuvissa $p \times p$ lohkoissa, joiden ominaisarvot $\to \lambda_i$.

Näiden teoreemojen todistusta ei esitetä tässä kurssissa.

Käytännössä kannattaa menetellä siten, että Householderin muunnoksella muodostetaan tridiagonaalimatriisi ja siitä jatketaan tasonkierto muunnoksilla, kuten Jacobin menetelmässä. Täten jokainen Q_s on tason kiertojen tulo

$$Q_s^T = P_1^{(s)} \cdot P_2^{(s)} \ldots \cdot P_{n-1}^{(s)}$$

missä P_i hävittää alkion $a_{i,i+1}$ ja symmetriasta johtuen myös alkion $a_{i+1,i}$.
Numerical Recipes kirjastosta löytyy ohjelma *TQLI*, jossa QL-algoritmilla lasketaan tridiagonaalimatriisin ominaisarvot.

2.8 Differentiaaliyhtälöt

Tehtävänä on ratkaista ensimmäisen kertaluvun differentiaaliyhtälö

$$y'(x) = f(x, y)$$

tai differentiaaliyhtälöryhmä

$$\begin{aligned} y_1'(x) &= f_1(x, y_1, \ldots, y_n) \\ \cdots &\quad \cdots \\ y_n'(x) &= f_n(x, y_1, \ldots, y_n). \end{aligned}$$

Toisen kertaluvun differentiaaliyhtälöt

$$y''(x) = f(x, y, y')$$

voidaan palauttaa ensimmäisen kertaluvun kytketyiksi yhtälöiksi ottamalla käyttöön toinen tuntematon funktio

$$\begin{aligned} y_1(x) &= y(x) \\ y_2(x) &= y'(x). \end{aligned}$$

Silloin toisen kertaluvun yhtälön ratkaisu saadaan laskettua ensimmäisen kertaluvun kytketyn yhtälöryhmän ratkaisuista

$$\begin{aligned} y_1'(x) &= y_2(x) \\ y_2'(x) &= f(x, y_1, y_2). \end{aligned}$$

2.8.1 Eulerin menetelmä

Tarkastellaan differentiaaliyhtälön

$$y' = f(x, y)$$

ratkaisemista alkuarvolla $y(x_0) = y_0$. Haluamme laskea yhtälön numeerisen ratkaisun $y_i = y(x_i)$ pisteissä

$$x_i = x_0 + ih \ , \quad \text{kun} \quad i = 1, 2, \ldots, n$$

Yksinkertaisin algoritmi on **Eulerin menetelmä**. Se etenee askel askeleelta siten, että ratkaisu pisteessä x_{i+1} saadaan tunnetusta ratkaisusta pisteessä x_i.

$$y_{i+1} = y_i + h\, f(x_i, y_i).$$

Menetelmässä funktiota approksimoidaan pisteeseen x_i piirretyn tangentin avulla

$$y(x) = y(x_i) + y'(x_i)(x - x_i) \ ,$$

jolloin ratkaisu pisteessä $x_{i+1} = x_i + h$ on

$$y(x_{i+1}) = y(x_i) + hy'(x_i) = y(x_i) + hf(x_i, y_i) \ .$$

Approksimoinnin virhe on kertalukua $O[h^2]$.

Esimerkki: Ratkaistaan yhtälö $y' = y$ lähtien pisteestä $x_0 = 0$, $y_0 = 1$. Eulerin menetelmällä saadaan ratkaisut

$$y_{i+1} = y_i + hy_i = (1 + h)y_i$$

eli pisteessä x_n ratkaisu on

$$y_n = (1 + h)^n = 1 + nh + \frac{n(n-1)}{2}h^2 + O[h^3] \; .$$

Oikea ratkaisu sen sijaan on

$$y_n = e^{nh} = 1 + nh + \frac{1}{2}n^2h^2 + O[h^3] \; .$$

Virhe tässä tapauksessa on $-nh^2/2 + O[h^3]$

2.8.2 Runge-Kutta menetelmä (2. kertaluvun kaava)

Eulerin menetelmän ongelmana on sen epätarkkuus. Suositumpi menetelmä differentiaaliyhtälön ratkaisemiseksi on **Runge-Kutta menetelmä**. Siinä funktion $f(x)$ arvoja approksimoidaan välillä $[x_i, x_{i+1}]$ tietyissä laskentatarkkuuden kannalta sopivasti valituissa pisteissä ja näitä arvoja käyttäen saadaan approksimaatio funktiolle pisteessä x_{i+1}. Menetelmällä saadaan myös arvio virheestä, ja siten ratkaisun tarkkuutta voidaan kontrolloida lyhentämällä ja pitentämällä askeleen pituutta.

Toisen kertaluvun Runge-Kutta menetelmässä (kutsutaan myös Heunin menetelmäksi) pisteiden x_i ja x_{i+1} puolessa välissä laskettu derivaatta määrää approksimaation funktion arvolle pisteessä x_{i+1}. Merkitään

$$\begin{aligned} k_1 &= hy'_i = hf(x_i, y_i) \\ k_2 &= hy'_{i+\frac{1}{2}} = hf(x_i + \frac{1}{2}h, y_{i+\frac{1}{2}}) \end{aligned}$$

Seuraavaksi käytetään Eulerin kaavaa ratkaisun arvon laskemiseen välipisteessä

$$y_{i+\frac{1}{2}} = y_i + \frac{h}{2}y'_i = y_i + \frac{1}{2}k_1,$$

jolloin funktiolle k_2 saadaan yhtälö

$$k_2 = h\, f(x_i + \frac{h}{2}, y_i + \frac{1}{2}k_1).$$

Koko askel on silloin

$$y_{i+1} = y_i + hy'_{i+\frac{1}{2}} = y_i + k_2 + O[h^3] \; .$$

Seuraava aliohjelma ratkaisee ensimmäisen kertaluvun differentiaaliyhtälön Runge-Kutta menetelmällä.

```
C runge.f
      subroutine runge(x0, y0, h, n, y, fun)
      integer n,i
      real    x0, y0, y(n), fun, x, k1, k2
      extern fun
      y(1) = y0
      do i = 1,n
        x = x0 + (i-1)*h
        k1 = h*fun(x, y(i))
        k2 = h*fun(x+0.5*h, y(i)+0.5*k1)
        y(i+1) = y(i) + k2
      end do
      return
      end
```

Esimerkkinä ohjelman soveltamisesta ratkaistaan differentiaaliyhtälö

$$y'(x) = -\alpha\, y(x).$$

Tällöin funktioaliohjelma on

```
C fun.f
      real function fun(x,y)
      real x,y, alfa
      common/par/alfa
      fun = -alfa*y
      return
      end
```

Funktion tarvitsemat parametrit (tässä tapauksessa `alfa`) on välitettävä funktioaliohjelmalle `COMMON` alueella.

Pääohjelma differentiaaliyhtälön ratkaisemikseksi on puolestaan

```
C     program diffeq
      parameter (ndim = 100)
      integer n,i
      real    x0, y0, h, y(ndim), fun, alfa
      common/par/alfa
      external fun
C
      print*, 'Anna alkuarvot:'
      read*, x0, y0
      print*, 'Anna parametri alfa:'
      read*, alfa
      print*, 'Anna askel ja pisteiden lukumaara:'
      read*, h, n
C
      call runge(x0, y0, h, n, y, fun)
      print*, 'Ratkaisu:'
      write(6, '('' x   '', '' y(x)    '')')
      do i i=1,n
        write(6, '(f10.3, e15.5)')  x0+(i-1)*h, y(i)
   1  continue
      end
```

2.8.3 Runge-Kutta menetelmä (4. kertaluvun kaava)

Tavallisimmin käytetty menetelmä differentiaaliyhtälön ratkaisemiseksi on ns. **klassinen Runge-Kutta kaava** eli neljännen kertaluvun Runge-Kutta kaava. Siinä pisteessä $x_{i+\frac{1}{2}}$ lasketun derivaatan arvoa parannetaan yhden kerran ennenkuin otetaan koko askel ja kertoimet derivaatan arvoille eri approksimaatioissa valitaan siten, että virhe saadaan mahdollisimman pieneksi.

$$\begin{aligned} k_1 &= h\, f(x_i, y_i) \\ k_2 &= h\, f(x_i + \frac{h}{2}, y_i + \frac{1}{2}k_1) \\ k_3 &= h\, f(x_i + \frac{h}{2}, y_i + \frac{1}{2}k_2) \\ k_4 &= h\, f(x_i + h, y_i + k_3)\ , \end{aligned}$$

jolloin seuraava piste lasketaan kaavasta

$$y_{i+1} = y_i + \frac{k_1}{6} + \frac{k_2}{3} + \frac{k_3}{3} + \frac{k_4}{6} + O[h^5]\ .$$

Numerical Recipes kirjassa on aliohjelma `RK4`, jossa käyteään neljännen kertaluvun Runge-Kutta kaavaa. Aliohjelmassa `DERIVS` lasketaan funktioiden derivaatat differentiaaliyhtälöstä.

```
      SUBROUTINE RK4(Y,DYDX,N,X,H,YOUT,DERIVS)
      PARAMETER (NMAX=10)
      DIMENSION Y(N),DYDX(N),YOUT(N)
     1 ,YT(NMAX),DYT(NMAX),DYM(NMAX)
      HH=H*0.5
      H6=H/6.
      XH=X+HH
      DO 11 I=1,N
        YT(I)=Y(I)+HH*DYDX(I)
11    CONTINUE
      CALL DERIVS(XH,YT,DYT)
      DO 12 I=1,N
        YT(I)=Y(I)+HH*DYT(I)
12    CONTINUE
      CALL DERIVS(XH,YT,DYM)
      DO 13 I=1,N
        YT(I)=Y(I)+H*DYM(I)
        DYM(I)=DYT(I)+DYM(I)
13    CONTINUE
      CALL DERIVS(X+H,YT,DYT)
      DO 14 I=1,N
        YOUT(I)=Y(I)+H6*(DYDX(I)+DYT(I)+2.*DYM(I))
14    CONTINUE
      RETURN
      END
```

2.8.4 Toisen kertaluvun differentiaaliyhtälöt

Kuten edellä mainittiin toisen kertaluvun differentiaaliyhtälö

$$y''(x) = f(x, y, y')$$

voidaan palauttaa ensimmäisen kertaluvun kytketyiksi yhtälöiksi ottamalla käyttöön toinen tuntematon funktio

$$u(x) \quad = \quad y'(x).$$

Silloin toisen kertaluvun yhtälön ratkaisu saadaan laskettua ensimmäisen kertaluvun kytketyn yhtälöryhmän ratkaisuista

$$\begin{aligned} y'(x) &= u(x) \\ u'(x) &= f(x, y, u). \end{aligned}$$

Nämä yhtälöt voidaan ratkaista joko Eulerin menetelmällä tai Runge- Kutta menetelmällä. Ratkaisu toisen kertaluvun Runge- Kutta menetelmällä on

$$\begin{aligned} k_1 &= h\, f(x_i, y_i, u_i) \\ \kappa_1 &= h\, u_i \end{aligned}$$

$$\begin{aligned} k_2 &= h\, f(x_i + \frac{1}{2}h, y_i + \frac{1}{2}\kappa_1, u_i + \frac{1}{2}k_1) \\ \kappa_2 &= h\,(u_i + \frac{1}{2}k_1), \end{aligned}$$

jolloin ratkaisun seuraavat pisteet lasketaan yhtälöistä

$$\begin{aligned} u_{i+1} &= u_i + k_2 \\ y_{i+1} &= y_i + \kappa_2 \end{aligned}$$

2.8.5 Numeerisia esimerkkejä

Ratkaistaan neljän ensimmäisen Besselin funktion differentiaaliyhtälöiden muodostama ryhmä

$$\begin{aligned} y_0'(x) &= -y_1(x) \\ y_1'(x) &= y_0(x) - \frac{1}{x}y_1(x) \\ y_2'(x) &= y_1(x) - \frac{2}{x}y_2(x) \\ y_3'(x) &= y_2(x) - \frac{3}{x}y_3(x) \end{aligned}$$

käyttämällä Numerical Recipes kirjaston ohjelmaa `RK4`.

Pääohjelma on:

```
      PROGRAM D15R1
C      Driver for routine RK4
      EXTERNAL DERIVS
      PARAMETER(N=4)
      DIMENSION Y(N),DYDX(N),YOUT(N)
```

```
      X=1.0
      Y(1)=BESSJ0(X)
      Y(2)=BESSJ1(X)
      Y(3)=BESSJ(2,X)
      Y(4)=BESSJ(3,X)
      DYDX(1)=-Y(2)
      DYDX(2)=Y(1)-Y(2)
      DYDX(3)=Y(2)-2.0*Y(3)
      DYDX(4)=Y(3)-3.0*Y(4)
      WRITE(*,'(/1X,A,T19,A,T31,A,T43,A,T55,A)')
     *            'Bessel Function:','J0','J1','J3','J4'
      DO 11 I=1,5
            H=0.2*I
            CALL RK4(Y,DYDX,N,X,H,YOUT,DERIVS)
            WRITE(*,'(/1X,A,F6.2)') 'For a step size of:',H
            WRITE(*,'(1X,A10,4F12.6)') 'RK4:',(YOUT(J),J=1,4)
            WRITE(*,'(1X,A10,4F12.6)') 'Actual:',BESSJ0(X+H),
     *                        BESSJ1(X+H),BESSJ(2,X+H),BESSJ(3,X+H)
11      CONTINUE
      END
```

ja derivaatat lasketaan aliohjelmassa

```
      SUBROUTINE DERIVS(X,Y,DYDX)
      DIMENSION Y(4),DYDX(4)
      DYDX(1)=-Y(2)
      DYDX(2)=Y(1)-(1.0/X)*Y(2)
      DYDX(3)=Y(2)-(2.0/X)*Y(3)
      DYDX(4)=Y(3)-(3.0/X)*Y(4)
      RETURN
      END
```

2.8.6 Tarkkuuden kontrollointi askeleen pituuden avulla

Yksinkertaisin keino testata tarkkuutta on pienentää askeleen pituus puoleen ja verrata siten saatua tulosta alkuperäiseen, jos tarkkuus ei ole riittävä, niin puolitetaan askel. Neljännen asteen Runge-Kutta menetelmässä tämä aiheuttaa ylimääräistä työtä kertoimen

$$\frac{12-1}{8} = 1.375$$

verran. Jos ensin lasketaan yksi askel $x \to x+2h \to y_1$ ja sitten sama väli kahdella askeleella $x \to x+h \to x+2h \to y_2$ niin siihen tarvitaan 12-1 funktion kutsua, mutta samalla saavutetaan kuitenkin parempi tarkkuus. Sitä pitää verrata puolta lyhemmin askelin etenevään laskuun, jossa ei tehdä tarkkuuskontrollia. Silloin funktiota kutsutaan 8 kertaa.

Eksakti ratkaisu voidaan esittää muodossa

$$\begin{aligned} y(x+2h) &= y_1 + (2h)^5\phi + O[h^6] \\ y(x+2h) &= y_2 + 2h^5\phi + O[h^6]\ , \end{aligned}$$

missä y_1 on yhdellä pitkällä askeleella $(2h)$ saatu ratkaisu ja y_2 kahdella lyhyellä askeleella (h) saatu ratkaisu.

$$\phi = \frac{y^{(5)}(x)}{5!}$$

Virhearvioksi saadaan $\Delta \equiv y_2 - y_1$.
Tästä päästään helposti viidennen kertaluvun Runge-Kutta menetelmään. Valitsemalla sopivasti näiden kahden ratkaisun, y_2 ja y_1, lineaarikombinaatio voidaan h^5 termi kokonaan hävittää.

$$\begin{aligned} y(x+2h) &= \frac{16y_2 - y_1}{15} + O[h^6] \\ &= y_2 + \frac{\Delta}{15} + O[h^6] \ . \end{aligned}$$

Tätä menetelmää ei kuitenkaan kannata käyttää, sillä tarkkuutta kasvatettaessa on menetetty virhearvio, minkä säilyttäminen ratkaisussa on tärkeää.

2.8.7 Runge-Kutta-Fehlberg menetelmä

Tässä menetelmässä käytetään neljännen ja viidennen kertaluvun Runge-Kuttaa, mutta molemmissa kertaluvuissa esiintyvät samat välipisteet eri kertoimilla.
Viidennen kertaluvun Runge-Kutta voidaan kirjoittaa yleisesti muodossa

$$\begin{aligned} k_1 &= h\, f(x_i, y_i) \\ k_2 &= h\, f(x_i + a_2 h, y_i + b_{21} k_1) \\ \ldots &\quad \ldots \\ k_6 &= h\, f(x_i + a_6 h, y_i + b_{61} k_1 + \ldots + b_{65} k_5) \\ y_{i+1} &= y_i + c_1 k_1 + c_2 k_2 + \ldots + c_6 k_6 + O[h^6] \ . \end{aligned}$$

Samoja pisteitä käyttävä neljännen kertaluvun Runge-Kutta on

$$y^*_{i+1} = y_i + c^*_1 k_1 + c^*_2 k_2 + \ldots + c^*_6 k_6 + O[h^5] \ .$$

Virhearvioksi saadaan silloin

$$\Delta \equiv y_{i+1} - y^*_{i+1} = \sum_{i=1}^{6} (c_i - c^*_i) k_i + O[h^5]$$

Cash-Karp parametrit Runge-Kutta menetelmään.

i	a_i	b_{ij}					c_i	c^*_i
1							$\frac{37}{378}$	$\frac{2825}{27648}$
2	$\frac{1}{5}$	$\frac{1}{5}$					0	0
3	$\frac{3}{10}$	$\frac{3}{40}$	$\frac{9}{40}$				$\frac{250}{621}$	$\frac{18575}{48384}$
4	$\frac{3}{5}$	$\frac{3}{10}$	$\frac{-9}{10}$	$\frac{6}{5}$			$\frac{125}{594}$	$\frac{13525}{55296}$
5	1	$\frac{-11}{54}$	$\frac{5}{2}$	$\frac{-70}{27}$	$\frac{35}{27}$		0	$\frac{277}{14336}$
6	$\frac{7}{8}$	$\frac{1631}{55296}$	$\frac{175}{512}$	$\frac{575}{13824}$	$\frac{44275}{110592}$	$\frac{253}{4096}$	$\frac{512}{1771}$	$\frac{1}{4}$
j=		1	2	3	4	5		

Virhe Δ on kertalukua $O[h^5]$, joten jos Δ_0 on haluttu tarkkuus ja virhe Δ on laskettu askeleella h, niin uusi askeleen pituus on

$$h_0 = h \left| \frac{\Delta_0}{\Delta} \right|^{0.2}$$

2.9 Differentiaaliyhtälöt, joilla on ominaisarvo

Differentiaaliyhtälöillä, joilla on ominaisarvo, on tärkeä merkitys fysiikassa. Mikromaailman ominaisuuksien mallintaminen perustuu lineaarisen toisen kertaluvun differentiaaliyhtälön ratkaisuihin. Tätä kvanttimekaniikan perusyhtälöä kutsutaan *Schrödingerin yhtälöksi*.

2.9.1 Schrödingerin yhtälö

Kvanttimekaniikan kursseilla on tutkittu Schrödingerin yhtälön,

$$-\frac{\hbar^2}{2m}\nabla^2\Psi(\mathbf{r}) + V(\mathbf{r})\Psi(\mathbf{r}) = E\Psi(\mathbf{r})$$

analyyttisiä ominaisuuksia, joista lyhyt yhteenveto.
Yhtälö on muodoltaan homogeeninen toisen kertaluvun differentiaaliyhtälö, jolla on ominaisarvo E. Yhtälön ratkaisuna saadaan aaltofunktio $\Psi(\mathbf{r})$, kun tunnetaan vuorovaikutus $V(\mathbf{r})$ ja massa m. Yhtälön ratkaiseminen yksinkertaistuu, kun oletetaan, että hiukkasten välinen vuorovaikutus riippuu vain niiden välisestä etäisyydestä $|\mathbf{r}|$,

$$V(\mathbf{r}) = V(|\mathbf{r}|) \ .$$

Tällöin aaltofunktio separoituu, eli voidaan kirjoittaa muodossa

$$\Psi(\mathbf{r}) = R(r)Y_{lm}(\theta, \phi) \ .$$

Kulmariippuvuus esitetään pallofunktioiden $Y_{lm}(\theta, \phi)$ avulla.

Muuttujasta r riippuva osa eli radiaalinen yhtälö voidaan kirjoittaa muotoon (vrt Kvanttimekaniikka I kurssi)

$$-\frac{\hbar^2}{2m}\left[\frac{1}{r}\frac{d^2}{dr^2}r - \frac{l(l+1)}{r^2}\right]R(r) + V(r)R(r) = ER(r) \ .$$

Otetaan käyttöön merkinnät

$$\begin{aligned} u(r) &= rR(r) \\ v(r) &= \frac{2m}{\hbar^2}V(r) \\ \epsilon &= \frac{2m}{\hbar^2}E \ , \end{aligned}$$

jolloin radiaalinen Schrödingerin yhtälö saadaan kirjoitettua yksinkertaiseen muotoon

$$-u''(r) + \left(v(r) - \frac{l(l+1)}{r^2}\right)u(r) = \epsilon u(r) \ .$$

Tällä yhtälöllä voi olla diskreettejä ratkaisuja ja jatkumoratkaisuja riippuen vuorovaikutuksesta ja asetetuista rajaehdoista.

Ratkaistaan esimerkkinä numeerisesti atomi- ja ydinfysiikkaan liittyvä ongelma, jossa oletetaan, että vuorovaikutus $V(|\mathbf{r}|)$ kuvaa keskeisvoimaa ja sitoo elektronin tai nukleonin ytimeen. Aaltofunktion rajaehdot määräytyvät siten, että

$$u(r=0)=0, \quad \text{ja} \quad u(r=\infty)=0 \ .$$

Kvanttimekaniikan kurssista tiedetään, että alimman eli perustilan aaltofunktiolla ei ole muita kuin rajaehdon määräämät nollakohdat. Ensimmäisellä viritystilalla on yksi nollakohta rajojen välissä, toisella kaksi jne. Tästä saadaan pääkvanttiluku $n = 1, 2, \ldots$.

Yleistä numeerista algoritmia, joka ottaisi kaikki tapaukset huomioon automaattisesti ei ole vielä kehitetty, joten ratkaisun löytyminen vaatii huolellista perehtymistä annetun probleeman ominaisuuksiin.

Tällöin numeerinen algoritmi etenee seuraavasti:

1. Arvataan ensin ominaisarvo ϵ. Perustilalle hyvä arvaus on $\epsilon \approx v(r_{min})$.
2. Jaetaan differentiaaliyhtälö kahdeksi ensimmäisen kertaluvun differentiaaliyhtälöksi ja ratkaistaan origosta lähtien (käyttäen esimerkiksi Runge-Kutta-Fehlberg menetelmää) pisteeseen r_m. Merkitään ratkaisua $u_o(r)$.
3. Ratkaistaan differentiaaliyhtälö alkaen jostakin suuresta r:n arvosta, $r = M$, kohti origoa pisteeseen r_m saakka. Piste M valitaan niin, että se on potentiaalin kantaman ulkopuolella. Tätä ratkaisua merkintään $u_\infty(r)$. Alkuehdoiksi valitaan
$$\begin{aligned} u_o(r=0) &= 0, \quad \text{ja} \quad u_o'(r=0) = C_0 = \text{vakio} \\ u_\infty(r=M) &= 0, \quad \text{ja} \quad u_\infty'(r=M) = C_\infty = \text{vakio} \end{aligned}$$
4. Vaaditaan, että $u(r)$ ja $u'(r)$ ovat jatkuvia pisteessä r_m. Tämä vaatimus toteutetaan vaiheittain. Ensin saatetaan suhteet,
$$\frac{u_o'(r_m)}{u_o(r_m)} = \frac{u_\infty'(r_m)}{u_\infty(r_m)} \ ,$$
yhtäsuuriksi.
Otetaan käyttöön merkintä
$$\Delta(\epsilon) = \frac{u_o'(r_m)}{u_o(r_m)} - \frac{u_\infty'(r_m)}{u_\infty(r_m)} \ .$$
5. Etsitään ominaisarvoa ϵ iteroimalla suureen $\Delta(\epsilon)$ nollakohta. Iteroinnissa käytetään esimerkiksi binaarihakua, jolloin tarvitaan kaksi arvausta ominaisarvolle ϵ siten, että niistä lasketuilla arvoilla $\Delta(\epsilon_1)$ ja $\Delta(\epsilon_2)$ on eri merkki ja sen jälkeen pienennetään väliä $|\epsilon_1 - \epsilon_2|$, kunnes haluttu tarkkuus on saavutettu.
6. Valitaan C_∞ niin, että ratkaisut kohtaavat pisteessä r_m
$$C_\infty = \frac{u_o(r_m)}{u_\infty(r_m)} \ .$$
Tällöin myös derivaatta on jatkuva pisteessä r_m.
7. Normitetaan ratkaisu siten, että
$$\begin{aligned} \int d^3r |\Psi(\mathbf{r})|^2 &= 1 \\ \int d^3r R^2(r) |Y_{lm}(\Omega)|^2 &= 1 \\ \int_0^\infty dr u^2(r) &= 1 \ . \end{aligned}$$

Tämä määrää kertoimen C_o,

$$C_o = \frac{1}{\sqrt{\int_0^\infty dr u^2(r)}} .$$

Tuloksena on saatu ominaisarvo ja aaltofunktio, joka toteuttaa halutut rajaehdot ja normituksen.

2.9.2 Kytketyt Schrödingerin yhtälöt

Edellä esitetty ratkaisumenetelmä voidaan yleistää kytkettyjen Schrödingerin yhtälöiden ryhmälle,

$$u_i''(r) = (f_{ii}(r) - \epsilon)u_i(r) + \sum_{j\neq i} f_{ij}(r)u_j(r) ,$$

missä $i = 1, \ldots, n$ ja $j = 1, \ldots, n$.
Ensimmäisen kertaluvun differentiaaliyhtälöiden ryhmänä tämä voidaan kirjoittaa muotoon,

$$\begin{aligned} w_i'(r) &= (f_{ii}(r) - \epsilon)u_i(r) + \sum_{j\neq i} f_{ij}(r)u_j(r) , \\ u_i'(r) &= w_i(r) , \end{aligned}$$

missä $i = 1, \ldots, n$ ja $j = 1, \ldots, n$.

Ratkaisujen alkuarvot valitaan seuraavasti

$$\begin{aligned} u_{io}(r=0) &= 0, \\ u_{io}'(r=0) &= w_{io}(r=0) = C_{io}, \\ u_{i\infty}(r=M) &= 0, \\ u_{i\infty}'(r=M) &= w_{i\infty}(r=M) = C_{i\infty} \end{aligned}$$

Asetataan yksi alkuarvoista C_{io} ja toisaalta $C_{i\infty}$ kerrallaan nollasta poikkeavaksi ja kaikki muut nolliksi. Tällöin saadaan n kappaletta lineaarisesti riippumattomia ratkaisuja.

Annetuista lähtöarvoista lähtien lasketaan ratkaisut kohtaamispisteessä r_m ja muodostetaan seuraava lineaarinen yhtälöryhmä.

$$\begin{aligned} & a_1 u_{1o}^{(1)}(r_m) + \ldots + a_n u_{1o}^{(n)}(r_m) \\ = \; & b_1 u_{1\infty}^{(1)}(r_m) + \ldots + b_n u_{1\infty}^{(n)}(r_m) \\ & \\ & a_1 w_{1o}^{(1)}(r_m) + \ldots + a_n w_{1o}^{(n)}(r_m) \\ = \; & b_1 w_{1\infty}^{(1)}(r_m) + \ldots + b_n w_{1\infty}^{(n)}(r_m) \\ & \ldots \\ & a_1 u_{no}^{(1)}(r_m) + \ldots + a_n u_{no}^{(n)}(r_m) \\ = \; & b_1 u_{n\infty}^{(1)}(r_m) + \ldots + b_n u_{n\infty}^{(n)}(r_m) \\ & \\ & a_1 w_{no}^{(1)}(r_m) + \ldots + a_n w_{no}^{(n)}(r_m) \\ = \; & b_1 w_{n\infty}^{(1)}(r_m) + \ldots + b_n w_{n\infty}^{(n)}(r_m) \end{aligned}$$

Ratkaisut $u_{1o}^{(i)}(r_m), w_{1o}^{(i)}(r_m), \ldots, u_{no}^{(i)}(r_m), w_{no}^{(i)}(r_m)$ saadaan, kun asetetaan $C_{io} \neq 0$, ja ratkaisut $u_{1\infty}^{(i)}(r_m), w_{1\infty}^{(i)}(r_m), \ldots, u_{n\infty}^{(i)}(r_m), w_{n\infty}^{(i)}(r_m)$, kun asetetaan $C_{i\infty} \neq 0$. Yläindeksi $^{(i)}$ kertoo siis, mikä kertoimista on nollasta poikkeava.

Yhtälöt muodostavat lineaarisen homogeenisen yhtälöryhmän, jonka ratkaisuna saadaan kertoimet a_i ja b_i. Yhtälöllä on ei-triviaali ratkaisu, jos sen kerroindeterminantti,

$$\begin{vmatrix} u_{1o}^{(1)} & \dots & u_{1o}^{(n)} & -u_{1\infty}^{(1)} & \dots & -u_{1\infty}^{(n)} \\ w_{1o}^{(1)} & \dots & w_{1o}^{(n)} & -w_{1\infty}^{(1)} & \dots & -w_{1\infty}^{(n)} \\ \dots & & \dots & \dots & & \dots \\ u_{no}^{(1)} & \dots & u_{no}^{(n)} & -u_{n\infty}^{(1)} & \dots & -u_{n\infty}^{(n)} \\ w_{no}^{(1)} & \dots & w_{no}^{(n)} & -w_{n\infty}^{(1)} & \dots & -w_{n\infty}^{(n)} \end{vmatrix} = D(\epsilon) = 0 \ .$$

Yhdelle Schrödingerin yhtälölle yo ehto palautuu muotoon

$$\begin{vmatrix} u_{1o}^{(1)} & -u_{1\infty}^{(1)} \\ w_{1o}^{(1)} & -w_{1\infty}^{(1)} \end{vmatrix} = 0 \ ,$$

josta saadaan tuttu jatkuvuusehto,

$$u_{1o}^{(1)} w_{1\infty}^{(1)} = u_{1\infty}^{(1)} w_{1o}^{(1)}$$

$$\frac{w_{1o}^{(1)}}{u_{1o}^{(1)}} = \frac{w_{1\infty}^{(1)}}{u_{1\infty}^{(1)}} \ .$$

Vanhoja merkintöjä käyttäen tämä on

$$\frac{u_o'}{u_o} = \frac{u_\infty'}{u_\infty} \ .$$

Yhtälöryhmä ratkaistaan siten, että

1. ominaisarvoa ϵ iteroidaan niin kauan kunnes kerroindeterminantti on nolla halutulla tarkkuudella ja näin saadaan ϵ.

2. Sen jälkeen kertoimet ratkaistaan siten, että yksi kertoimista, esimerkiksi a_1, asetetaan ykköseksi ja yksi yhtälöistä, esimerkiksi ensimmäinen yhtälö, poistetaan. Tällöin oletetaan, että poistettava yhtälö riippuu lineaarisesti muista yhtälöistä. Näin muodostuneella ei-homogeenisella $2n-1$ yhtälön ryhmällä on aina ratkaisu ja kertoimet a_i ja b_i saadaan laskettua kerrointa a_1 lukuunottamatta. Sen määrää normitus, joka valitaan tehtävään sopivalla tavalla.

3. Lopulliset aaltofunktiot saadaan valitsemalla uusiksi lähtöarvoiksi origosta lähdettäessä $a_i C_{io}$ ja äärettömyydestä päin lähdettäessä $b_i C_{i\infty}$ ja ratkaisemalla koko yhtälöryhmä näillä lähtöarvoilla.

2.10 Fourier-muunnoksista

Fourier-muunnosten avulla voidaan tutkia (ajan suhteen) mitatun signaalin taajuussisältöä. Kuvatkoon $h(t)$ ajan suhteen vaihtelevaa signaalia. Signaalissa on useita taajuuskomponentteja, siten että tietty taajuus on edustettuna painolla $H(f)$. Voidaan ajatella, että funktiot $h(t)$ ja $H(f)$ esittävät samaa asiaa eri avaruuksista katsottuna (time domain/frequency domain). Siirtyminen esityksestä toiseen tapahtuu Fourier-muunnosten avulla.

$$H(f) = \int_{-\infty}^{\infty} h(t) e^{2\pi i f t} dt$$

ja käänteismuunnos

$$h(t) = \int_{-\infty}^{\infty} H(f) e^{-2\pi i f t} df \ .$$

Käänteismuunnosta laskettaessa tarvitaan δ-funktion integraaliesitystä.

$$\delta(t' - t) = \int_{-\infty}^{\infty} e^{2\pi i f (t' - t)} df \ .$$

Tällöin

$$\begin{aligned} h(t) &= \int_{-\infty}^{\infty} e^{-2\pi i f t} df \int_{-\infty}^{\infty} h(t') e^{2\pi i f t'} dt' \\ &= \int_{-\infty}^{\infty} h(t') dt' \int_{-\infty}^{\infty} e^{2\pi i f (t' - t)} df \\ &= \int_{-\infty}^{\infty} h(t') \delta(t' - t) dt' = h(t) \end{aligned}$$

Taajuuden sijasta usein käytetään kulmanopeutta, $\omega = 2\pi f$ muuttujana, jolloin Fourier-muunnoksiin täytyy muistaa lisätä kerroin $1/2\pi$.

$$\begin{aligned} G(\omega) &= \int_{-\infty}^{\infty} g(t) e^{i\omega t} dt \\ g(t) &= \frac{1}{2\pi} \int_{-\infty}^{\infty} G(\omega) e^{-i\omega t} d\omega \ . \end{aligned}$$

Fourier-muunnoksen ominaisuuksia:

Jos $h(t)$ on	niin silloin
reaalinen	$H(-f) = H^*(f)$
imaginaarinen	$H(-f) = -H^*(f)$
parillinen	$H(-f) = H(f)$, parillinen
pariton	$H(-f) = -H(f)$ pariton

Ajan skaalaus ja ajan siirto

$$\begin{aligned} h(at) &\leftrightarrow \frac{1}{|a|} H(\frac{f}{a}) \\ h(t - t_0) &\leftrightarrow H(f) e^{2\pi i f t_0} \ . \end{aligned}$$

Kahden funktion konvoluutio

$$g * h \equiv \int_{-\infty}^{\infty} g(\tau)h(t-\tau)d\tau = h * g \ .$$

Konvoluution Fourier-muunnos on funktioiden Fourier-muunnosten tulo

$$\int_{-\infty}^{\infty} g * h \ \ e^{2\pi i f t}dt = G(f)H(f) \ .$$

Kahden funktion korrelaatio

$$\mathrm{Corr}(g,h) \equiv \int_{-\infty}^{\infty} g(\tau+t)h(\tau)d\tau \ .$$

Korrelaatio on ajan funktio ja sitä kutsutaan viiveeksi. Korrelaation Fourier-muunnos on

$$\int_{-\infty}^{\infty} \mathrm{Corr}(g,h) \ \ e^{2\pi i f t}dt = G(f)H(-f) = G(f)H^*(f)$$

Viimeinen yhtäsuuruus on voimassa mikäli $h(t)$ ja $g(t)$ on valittu reaalisiksi.
Funktion korrelaatio itsensä kanssa on autokorrelaatio, $\mathrm{Corr}(g,g)$. Sen Fourier-muunnokseksi saadaan,

$$\mathrm{Corr}(g,g) \Longleftrightarrow |G(f)|^2 \ .$$

Signaalin kokonaisteho on sama, laski sen sitten ajan tai taajuuden funktiona.

$$\text{Kokonaisteho} \equiv \int_{-\infty}^{\infty} |h(t)|^2 dt = \int_{-\infty}^{\infty} |H(f)|^2 df$$

2.10.1 Tasaisin aika-askelin poimitun datan Fourier-muunnos

Olkoon aika-askel Δ, ja käytetään merkintää

$$h_n = h(n\Delta), \quad \text{kun} \quad n = \ldots, -2, -1, 0, 1, 2, \ldots$$

Jokaista aika-askelta vastaa Nyquistin kriittinen taajuus f_c,

$$f_c \equiv \frac{1}{2\Delta}$$

Jos esimerkiksi sini-aallosta, $\sin(2\pi f_c t)$, jonka taajuus on Nyquistin kriittinen taajuus poimitaan arvo aallon maksimin kohdalta, niin seuraava arvo tulee sitten minimin kohdalta seuraava taas maksimista jne.

$$\sin\left(\frac{\pi}{2} + 2\pi f_c n\Delta\right) = (-1)^n \ ,$$

koska $2f_c\Delta = 1$.

Poiminta teoreema: Jos jatkuvasta funktiosta $h(t)$ poimitaan arvoja välein Δ ja funktion taajuuskaistan leveys on rajoitettu siten, että $H(f) = 0$ kun $|f| \geq f_c$, niin silloin poimitut arvot määräävät funktion $h(t)$ täysin. Itse asiassa voidaan kirjoittaa eksplisiittisesti

$$h(t) = \Delta \sum_{n=-\infty}^{\infty} h_n \frac{\sin[2\pi f_c(t-n\Delta)]}{\pi(t-n\Delta)}$$

Seurauslause: Jos funktio, jonka kaistaa ei ole rajoitettu, Fourier-muunnetaan niin taajuudet, jotka jäävät rajojen $-f_c < f < f_c$ ulkopuolelle kääntyvät rajojen sisäpuolelle aiheuttaen virhettä.

2.10.2 Diskreetti Fourier-muunnos

Lasketaan arvio funktion Fourier-muunnokselle käyttäen äärellistä määrää poimittuja arvoja. Oletetaan, että käytettävissä on N kpl peräkkäin poimittuja arvoja

$$h_k \equiv h(t_k), \quad t_k \equiv k\Delta, \quad k = 0, 1, 2, \ldots, N-1$$

Olkoon $h_k = 0$, kun $k \geq N$. Lisäksi oletetaan yksinkertaisuuden vuoksi, että N on parillinen.
Tehtävänä on etsiä Fourier-muunnos pisteissä

$$H(f_n) = H(\frac{n}{N\Delta}); \quad f_n = \frac{n}{N\Delta}; \quad n = -\frac{N}{2}, \ldots, \frac{N}{2}$$

joten Nyquistin kriittinen taajuus rajoittaa taajuusalueen, $-f_c \leq f \leq f_c$. Näitä arvoja on $N+1$ kpl eli yksi enemmän kuin funktion h arvoja. Itse asiassa rajoilla lasketut arvot ovat samoja,

$$H(f_{-N/2}) = H(f_{N/2}),$$

ja jäljelle jää N toisistaan riippumatonta arvoa.

Seuraavaksi arvioidaan Fourier-integraalia summalla

$$\begin{aligned} H(f_n) &= \int_{-\infty}^{\infty} h(t) e^{2\pi i f_n t} dt \approx \sum_{k=0}^{N-1} h_k e^{2\pi i f_n k\Delta} \Delta \\ &= \Delta \sum_{k=0}^{N-1} h_k e^{2\pi i n k/N} \end{aligned}$$

Merkitään

$$H_n \equiv \sum_{k=0}^{N-1} h_k e^{2\pi i n k/N} \ ,$$

joten

$$H(f_n) = \Delta H_n \ .$$

Edellä n on saanut arvot $-\frac{N}{2}, \ldots, \frac{N}{2}$, mutta helposti havaitaan, että funktio H_n on periodinen ja n:nnän perioidi on N, eli esimerkiksi

$$H_{-n} = H_{N-n} \ , \quad n = 1, 2, \ldots \ .$$

Käytännöllisin valinta n:nnän arvoille on kuitenkin $n = 0, \ldots, N-1$ yhden periodin aikana, koska silloin k ja n saavat täsmälleen samoja arvoja. Tällöin täytyy huomioida, että nolla taajuus $f = 0$ vastaa arvoa $n = 0$, positiiviset taajuudet $0 < f < f_c$ arvoja $1 \leq n \leq \frac{N}{2} - 1$, negatiiviset taajuudet $-f_c < f < 0$ arvoja $\frac{N}{2} + 1 \leq n \leq N-1$ ja arvo $n = \frac{N}{2}$ taajuuksia $f = f_c$ ja $f = -f_c$, koska $H(f_c) = H(-f_c)$.

Diskreetin muunnoksen käänteismuunnokseksi saadaan

$$h_k = \frac{1}{N} \sum_{k=0}^{N-1} H_n e^{-2\pi i n k/N} \ .$$

Lauseke on lähes samanlainen kuin itse Fourier-muunnos lukuunottamatta - merkkiä eksponentissa ja kerrointa $1/N$, joten Fourier-muunnoksen tekevä ohjelma tekee myös käänteismuunnoksen pienillä muutoksilla.
Kokonaisteho, ns. Parseval'in teoreema, voidaan siten diskreetille muunokselle kirjoittaa muodossa.

$$\sum_{k=0}^{N-1} |h_k|^2 = \frac{1}{N} \sum_{n=0}^{N-1} |H_n|^2 \ .$$

2.10.3 Nopea Fourier-muunnos, FFT

Johdetaan algoritmi nopean Fourier-muunnoksen suorittamiseksi.
Olkoon $W \equiv e^{2\pi i/N}$, joten

$$H_n = \sum_{k=0}^{N-1} W^{nk} h_k$$

Tätä voidaan ajatella kertalukua $\mathrm{O}(N^2)$ olevana lineaarisena matriisimuunnoksena. Nopea Fourien-muunnos sen sijaan käyttää $\mathrm{O}(N \log_2(N))$ operaatiota, joten parannus tavalliseen matriisimuunnokseen verrattuna on erittäin suuri, kun N on suuri luku.

Nopea Fourier-muunnos perustuu **Danielson-Lanczos**in algoritmiin, jossa muunnos jaetaan parillisten ja parittomien termien summaksi ja nämä summat edelleen parillisten ja parittomien termien summaksi kunnes päädytään yksittäisiin termeihin.

$$\begin{aligned} F_k &= \sum_{j=0}^{N-1} e^{2\pi ijk/N} f_j \\ &= \sum_{j=0}^{N/2-1} e^{2\pi i(2j)k/N} f_{2j} + \sum_{j=0}^{N/2-1} e^{2\pi i(2j+1)k/N} f_{2j+1} \\ &= \sum_{j=0}^{N/2-1} e^{2\pi ijk/(N/2)} f_{2j} + W^k \sum_{j=0}^{N/2-1} e^{2\pi ijk/(N/2)} f_{2j+1} \\ &\equiv F_k^e + W^k F_k^o \end{aligned}$$

merkintä F_k^e tarkoittaa parillisia termejä ja F_k^o parittomia termejä.

Algoritmi etenee seuraavasti:

- Jaa F_k^e ja F_k^o edelleen $N/4$ parilliseen ja $N/4$ parittomaan termiin, jolloin saadaan suureet F_k^{ee}, F_k^{eo}, F_k^{oe} ja F_k^{oo}
- Käytä lukumäärälle N vain arvoja, jotka ovat luvun 2 potensseja, $N = 2^\alpha$.
- Puolitukset päätyvät lopulta tilanteeseen, jossa

  $$\frac{N}{2^\alpha} = 1$$

  missä α on puolitusten lukumäärä. Täten jokaiselle joukolle indeksejä $\{eoeeoe\ldots\}$, jossa on $\log_2 N$ kappaletta alkioita, päädytään tilanteeseen

  $$F_k^{eoeeoe\ldots} = f_j \ ,$$

  missä f_j on jokin alkuperäisistä funktion arvoista.

- Indeksien $\{eeeoo\ldots\}$ järjestys määrää funktion f_j indeksin j. Se saadaan siirtymällä kaksijärjestelmän esitykseen $e = 0$ ja $o = 1$. Esimerkiksi jonoa $\{eeeoo\}$ vastaa binaariesitys $\{00011\}$. Indeksin j arvo saadaan kääntämällä bittien järjestys ja kirjoittamalla saadun luvun arvo kymmenjärjestelmässä, joten $\{00011\} \rightarrow \{11000\} = 24 = j$. Tämän termin parina on termi $\{00010\} \rightarrow \{01000\} = 8$, muunnoksessa

  $$F_k^{0001} = F_k^{00010} + W^{5k} F_k^{00011} = f_8 + W^{5k} f_{24} \ .$$

Kerron 5 eksponentissa W^{5k} on puolitusten lukumäärä. Vastaavasti voidaan laskea termi

$$F_k^{0000} = F_k^{00000} + W^{5k} F_k^{00001} = f_0 + W^{5k} f_{16} \ .$$

ja näistä kahdesta sitten

$$F_k^{000} = F_k^{0000} + W^{4k} F_k^{0001}$$

ja niin edelleen, kunnes päädytään haluttuun komponenttiin F_k

$$F_k = F_k^0 + W^k F_k^1 \ .$$

Tämän laskemiseksi on tarvittu $\log_2 N$ kertolaskua. Koska kertoimia F_k on N kpl niin yhteensä tarvitaan $N \log_2 N$ kertolaskua Fourier-muunnoksen tekemiseksi.

Lasketaaan esimerkkinä, kuinka Danielson-Lanczos algoritmin etenee tapauksessa, jossa $N = 2^4 = 16$.

0000	=	0		0		0	=	0000
0001	=	1		2		4	=	1000
0010	=	2		4		8	=	0100
0011	=	3		6		12	=	1100
0100	=	4		8		2	=	0010
0101	=	5		10		6	=	1010
0110	=	6		12		10	=	0110
0111	=	7	=>	14	=>	14	=	1110
1000	=	8		1		1	=	0001
1001	=	9		3		5	=	1001
1010	=	10		5		9	=	0101
1011	=	11		7		13	=	1101
1100	=	12		9		3	=	0011
1101	=	13		11		7	=	1011
1110	=	14		13		11	=	1110
1111	=	15		15		15	=	1111

Numerical Recipes ohjelma `FOUR1(data,nn,isign)` suorittaa kompleksimuuttujan Fourier-muunnoksen käyttäen Danielson-Lanczos -algoritmia. Siinä muunnettava funktio syötetään data-taulukkoon `data(1:2*nn)` ja muunnoksen tulos on luettavissa samasta taulukosta hiukan poikkeuksellisessa järjestyksessä (harj).
Parametri `nn` on muunnettavan funktion pisteiden lukumäärä ja sen täytyy olla joku kakkosen potenssi.
Parametri `isign` on kertoo kummin päin muunnos suoritetaan. Kun `isign=1` suoritetaan Fourier-muunnos ja kun `isign=-1` tehdään käänteismuunnos.

```
subroutine four1(data,nn,isign)
real*8 wr,wi,wpr,wpi,wtemp,theta
dimension data(*)
n=2*nn
j=1
do 11 i=1,n,2
  if(j.gt.i)then
    tempr=data(j)
    tempi=data(j+1)
    data(j)=data(i)
    data(j+1)=data(i+1)
```

```
          data(i)=tempr
          data(i+1)=tempi
        endif
        m=n/2
1       if ((m.ge.2).and.(j.gt.m)) then
          j=j-m
          m=m/2
        go to 1
        endif
        j=j+m
11    continue

      mmax=2
2     if (n.gt.mmax) then
        istep=2*mmax
        theta=6.28318530717959d0/(isign*mmax)
        wpr=-2.d0*dsin(0.5d0*theta)**2
        wpi=dsin(theta)
        wr=1.d0
        wi=0.d0
        do 13 m=1,mmax,2
          do 12 i=m,n,istep
            j=i+mmax
            tempr=sngl(wr)*data(j)-sngl(wi)*data(j+1)
            tempi=sngl(wr)*data(j+1)+sngl(wi)*data(j)
            data(j)=data(i)-tempr
            data(j+1)=data(i+1)-tempi
            data(i)=data(i)+tempr
            data(i+1)=data(i+1)+tempi
12        continue
          wtemp=wr
          wr=wr*wpr-wi*wpi+wr
          wi=wi*wpr+wtemp*wpi+wi
13      continue
        mmax=istep
      go to 2
      endif
      return
      end
```