

# NUMEERISET MENETELMÄT JA C-KIELI

Kevät 2016

Matematiikan ja tilastotieteen laitos, Helsingin yliopisto

Matti Vuorinen, vuorinen@utu.fi

Luennot: ma 13-16, B321

Harjoitukset: ma 16-18, B321

**Materiaali saatavilla kurssin kotisivulla:**

<https://wiki.helsinki.fi/pages/viewpage.action?pageId=184356956>

HUOM. Tämä on versio 2015-12-30, päivityksiä tulee tarpeen mukaan.

## JOHDANTO

Tämän kurssin päätavoitteena on harjaannuttaa osallistujat käyttämään C/C++ -ohjelmointikieliä numeerisessa laskennassa. Luennot pohjautuvat pääosin teokseen

**(NR) Press et. al: Numerical Recipes in C++ 2nd ed.**

Cambridge Univ. Press 2002, jonka oheismateriaalina on esimerkiksi kirja ja CD-ROM. Tämä kokeellisen numeriiikan tuhti tietopaketti on saavuttanut suurta suosiota, koska kattavuudeltaan vastaavia kilpailijoita ei juuri ole. Käytämme lähinnä em. kirjaston uudempia C++ -versioita.

Tekijänoikeus rajoittaa tämän ja muiden kaupallisten ohjelmien käyttöä oppimateriaalina. Marraskuussa 2001 julkistettiin C-kielinen GSL(GNU Scientific Library)-kirjasto, jonka käyttö on vapaata opetustarkoituksiin ns. GNU-lisenssin mukaisesti. Laskuharjoitusten ratkaisut voi tehdä myös GSL:n avulla jos pitää sitä parempana kuin NRC:tä.

C++/C-kielen opiskelu kannattaa, koska

- C-kieltä osaaville matemaatikoille on työmarkkinoilla kysyntää
- NR:n ja GSL:n avulla opitaan matemaattista mallintamista ja ohjelman kehitystä tavalla, joka on Windows ja UNIX-yhteensopiva, mikä jo sinänsä on hyödyllistä,
- C-kielille on käyttöä kurssin jälkeenkin.

Kurssin [www-sivulle](http://www.sivulle) tulevat viikottain laskuharjoitustehtävät, kurssitiedotteet, malliratkaisut jne. Laskuharjoitustehtävän voi monesti ratkaista muokkaamalla luentojen esimerkkiohjelmaa.

Käytämme lähinnä GNU-kääntäjiä g++/gcc (Linux). Käyrien ja pintojen piirtoon käytämme gnuplot-julkisohjelmaa.

Helsinki 30.12. 2015

Matti Vuorinen

vuorinen@utu.fi

## Sisältö

Johdanto	1
1 KOKEELLINEN NUMERIIKKA JA C-KIELI	3
2 LINEAARISET YHTÄLÖRYHMÄT	84
3 INTERPOLOINTI JA EKSTRAPOLOINTI	125
4 NUMEERINEN INTEGROINTI	157
5 FUNKTIOIDEN APPROKSIMOINTI	167
6 ERIKOISFUNKTIOT	177
7 SATUNNAISLUVUISTA	187
8 LAJITTELUMENETELMISTÄ	194
9 EPÄLINEAARISET YHTÄLÖRYHMÄT	196
10 MINIMOINTI JA MAKSIMOINTI	215
11 OMINAISARVOT	269
12 FOURIER-MUUNNOS	279
13 DATAN TILASTOLLINEN ANALYYSI	284
14 DATAN MALLINTAMINEN	286
15 TAVALLISET DIFFERENTIAALIYHTÄLÖT	298
16 REUNA-ARVOTEHTÄVÄT	312
17 DIFFERENSSIMENETELMISTÄ	318

Tämä jaottelu on NR:n mukainen. Osa lukujen 11-16 materiaalista voi jäädä pois, luettelo on alustava.

**Huom.** Luentojen osana olevat C/C++-ohjelmat (www-sivulla paketti myexamples.zip) on testattu gcc-kääntäjällä ja gnuplotilla Linux-ympäristössä seuraavilla versioilla:

```
gcc version 4.8.4 (Ubuntu 4.8.4-2ubuntu1~14.04)
```

```
G N U P L O T
Version 4.6 patchlevel 4    last modified 2013-10-02
Build System: Linux x86_64
```

HUOM. Kurssin www-sivulla oleva paketti sisältää ohjelmien uusimmat versiot: tämän monisteen ohjelmia ei ole päivitetty—tiedostojen nimet ovat kuitenkin samoja.

NR:n vanhempi versio on ollut aiemmin saatavissa osoitteessa:

<http://www.fizyka.umk.pl/nrbook/bookcpdf.html>

Ilmoita os. vuorinen@utu.fi mahdolliset uusimmat tiedot. GSL:n osalta kannattaa käyttää webistä löytyvää materiaalia, ks. esim.

<http://www.gnu.org/software/gsl/>

# 1 KOKEELLINEN NUMERIikka JA C-KIELI

Tämän luvun tarkoituksena on luoda yleiskatsaus eräisiin numerii-  
kan peruskysymyksiin ja antaa esimerkkejä C-kielen käytöstä nu-  
meerisissa sovelluksissa. Samalla kerrataan C/C++-kielen perus-  
rakenteita jatkoa silmällä pitäen. Lisäksi esittelemme muutamien  
apuohjelmien käyttöä, kuten kuvaajien piirtoa gnuplot-ohjelman  
avulla.

Aluksi pari huomautusta kurssia tukevasta kirjallisuudesta ja oh-  
jelmistoista. Glassey'n kirja [G] on selkeä johdatus C-kielen numee-  
risiin sovelluksiin. Erityisesti [G]:n kaksi ensimmäistä lukua ovat  
erinomainen johdatus [NR]:n vaativampiin ohjelmiin. C++-kielen  
osalta vastaava teos on Yang [Y]. Kurssin kotisivulle on koottu mm.  
ohjelmointiin ja numeriiikkaan liittyviä linkkejä.

Käsikirja [AS] on erittäin hyödyllinen lähde teos kokeelliselle nu-  
meriiikalle. Taustakirjallisuudesta on tehty www-sivulle laitettu luet-  
telo, johon myös eo. viittaukset kohdistuvat.

**1.1. Tiedostojen nimeäminen.** Koska kurssilla luotavien tiedostojen  
määrä saattaa nousta satoihin, on tärkeää omaksua systemaatti-  
set ohjelmointitottumukset. Tiedostojen nimeämisessä kannattaa  
noudattaa seuraavia vinkkejä. Tiedostolle annetaan lyhyt kuvaava  
nimi, esimerkiksi:

```
h011.c   harj. 1 tehtävän 1 ratkaisu
h011.dat harj. 1 tehtävä 1, tuotettu data
```

Jatkossa käytetään vain tekstitiedostoja. Kunkin tekstitiedoston  
alkuun ja loppuun olisi syytä laittaa tiedoston nimi, jolloin tiedos-  
tojen järjestyksen ylläpito on helpompaa, esimerkiksi:

```
/* FILE: h011.c begins. */
..
..   tiedoston sis\alt\"o
..
/* FILE: h011.c ends. */
```

Vinkki: Tee tiedosto `beg.cpp`, jossa on valmiina alku/loppukommentit. Jatkossa se on helppo liittää mukaan uuteen tiedostoon.

```
/* FILE: beg.cpp begins */  
/* FILE: beg.cpp ends */
```

Laajempi muoto tiedostosta `beg.cpp`, jossa mukana mm. usein tarvittavat `include/otsikkotiedostot`, on tämän luvun lopussa.

**1.2. Liukuluvut.** Reaalilukujen esitys tietokoneen muistissa tapahtuu koneesta ja ohjelmasta riippuvalla tarkkuudella binäärilukujen avulla. Eri kantajärjestelmien välinen konversio on esitetty algebran kurssilla (ks. myös [AS, s. 1012]). Reaalilukujen esitystä tietokoneessa kutsutaan *liukulukuesitykseksi* (floating point representation). Periaatteessa liukulukuesitys on samanlainen kuin ns. *tieteellinen merkintätapa*, jossa esim lukuja 297.2 ja  $-0.00029$  merkitään  $2.972 \cdot 10^2$  ja  $-2.9 \cdot 10^{-4}$  ja  $637000 = 6.370 \cdot 10^5$  (jos 4 merkitsevää numeroa). Tieteellinen merkintätapa koostuu siis seuraavista osista: a) etumerkki b) mantissa (välillä  $[1, 10)$ ) c) eksponenttiossa, ja tässä mantissan pituus määräytyy merkitsevien numeroiden mukaan.

Liukulukuesityksen tarkkuus määräytyy mantissan pituudesta, joten siinä yleensä aina on virhettä. Liukulukuja on vain äärellinen määrä. Positiivisten liukulukujen joukossa on siis suurin liukuluku ja myös pienin nollaa suurempi liukuluku. Jos laskutoimituksen tulos johtaisi liukulukualueen ulkopuolelle, joudutaan virhetilanteeseen, jota kutsutaan ylivuodoksi tai alivuodoksi tilanteen mukaan. Liukuluvut eivät ole jakautuneet tasaisesti ääri rajojen väliin, vaan itseisarvoltaan pienet ovat tiheämmässä kuin itseisarvoltaan suuret. Liukulukuaritmetiikkaan kuuluvat yhteen-, vähennys-, jako-, kertolasku, loogiset vertailuoperaatiot (esim.  $a > b$ ) ja eräät muut reaalilukujen operaatiot. Liukulukuaritmetiikassa pätevät samankaltaiset laskusäännöt kuin reaaliluvuille, joten reaalilukujen laskusääntöjä voidaan käyttää kunhan muistetaan esitystarkkuudesta johtuvat rajoitukset. Periaatteelliselta kannalta on kuitenkin tärkeää

huomata, että eräin osin syntyy eroja. On syytä varautua virheisiin mm. analysoimalla laskennan tuloksia, ja sikäli kun mahdollista, ryhtyä tarpeellisiin toimiin pyöristysvirheiden välttämiseksi. Esim. käskyt  $a = 1/3$ ;  $b = 1.0/3$ ; voivat tuottaa eri arvot  $a$ :lle ja  $b$ :lle. Joitakin hyödyllisiä vinkkejä esitetään myöhemmin. Kattava tietopaketti liukuluvuista löytyy osoitteesta

<http://www.validlab.com/goldberg/paper.pdf> .

**1.3. Pyöristyksen sivuvaikutuksia.** Reaaliluvuille  $b, c > 0$  pätee:  $(b + c)/2 \leq \max\{b, c\}$ . Pyöristys kolmeen desimaaliin antaa liukuluvuille  $C = 0.982$ ,  $B = 0.984$  summaksi  $B + C = 1.97$  ja keskiarvoksi  $(B + C)/2 = 0.985$ . Siis liukulukuaritmetiikassa voi olla  $(B + C)/2 > \max\{B, C\}$ ! Tämän ei-toivotun sivuvaikutuksen välttämiseksi on syytä käyttää parempaa laskutapaa:  $B + \frac{1}{2}(C - B)$ .

**1.4. Kone-epsilon.** Kuten jo yllä kohdassa 1.2 todettiin, liukulukuesityksellä reaaliluku voidaan esittää kone-esityksestä riippuvalla tarkkuudella. Ns. *kone-epsilon* mittaa tätä tarkkuutta. Kone-epsilon meps on  $2^{-p}$  jos  $c < c + 2^{-p}$  pätee liukulukuaritmetiikassa mutta  $c < c + 2^{-p-1}$  ei päde ja  $p$  on positiivinen kokonaisluku sekä (tavallisesti)  $c = 1$ .

```
/* FILE: mymeps1.cpp begins */
/* g++ mymeps1.cpp -o a -lm
Program finds out machine epsilon. CPU dependent machine epsilon
can be considered as an estimate of the minimum error made by
computer in calculus operations.
*/

#include <stdlib.h>
#include <cstdio>
#include <ctime>
#include <climits>
#include <cmath>
#include <values.h>

int main()
{
```

```

int c = 1, p = 1;
float x = 1.0;
double xd = 1.0;
printf("\n MACHINE EPSILON 1 = ");
while (c + pow(2.0, -p) > c)
    p++;
printf("  %16.6E = 2^(-%3d) \n", pow(2, 1 - p), p - 1);
p = 1;
printf("\n MACHINE EPSILON 2 = ");
while (x + (float) pow(2.0, -p) > x)
    p++;
printf("  %16.6E = 2^(-%3d) \n", pow(2.0, 1 - p), p - 1);
printf("\n MACHINE EPSILON 3 = ");
p = 1;
while (xd + pow(2.0, -p) > xd)
    p++;
printf("  %16.6E = 2^(-%3d) \n", pow(2.0, 1 - p), p - 1);
printf
    ("\n FLT_EPSILON = %12.4E, DBL_EPSILON = %12.4E \n",
     FLT_EPSILON, DBL_EPSILON);
printf("\n DBL_MIN = %12.4E  DBL_MAX = %12.4E \n",
       DBL_MIN, DBL_MAX);
printf("\n FLT_MIN = %12.4E  FLT_MAX = %12.4E \n",
       FLT_MIN, FLT_MAX);
return 0;
}
/* FILE: mymeps1.cpp  ends */

/* Output:

MACHINE EPSILON 1 =      1.084202E-19 = 2^(- 63)
MACHINE EPSILON 2 =      1.084202E-19 = 2^(- 63)
MACHINE EPSILON 3 =      1.084202E-19 = 2^(- 63)
FLT_EPSILON =    1.1921E-07, DBL_EPSILON =    2.2204E-16
DBL_MIN =    2.2251E-308  DBL_MAX =    1.7977E+308
FLT_MIN =    1.1755E-38  FLT_MAX =    3.4028E+38

*/

```

Ohjelman käynnös tapahtuu esimerkiksi antamalla kommentteissa oleva komento.

Kone-epsilonia voidaan pitää jonkinlaisena arviona sille, kuinka suuri laskennan virhe vähintään on. Kone-epsilonin vaikutusta voi-



daan havainnollistaa myös taulukoimalla funktioita  $\sin^2 x + \cos^2 x - 1$  ja  $\text{asin}(\sin(x)) - x$  arvot välillä  $(0, 1)$ . Molemmat ovat matemaattisesti identtisesti nolliä, mutta numeerisesti syntyy luokkaa  $10^{-16}$  olevia poikkeamia. Luonnollisesti vastaavia testejä voidaan tehdä monille muillekin C-kielen tuntemille funktioille. Reaalilukuja esitetään C-kielessä tavallisesti tyyppiä `float` tai `double` olevilla muuttujilla. Laskentatarkkuus paranee `double`-tyyppisiä muuttujia käytettäessä, mutta ohjelman aika- ja tilavaatimukset yleensä kasvavat. Tällä kurssilla käytämme etupäässä `double`-tyyppisiä muuttujia.

**1.5. NRC:n ohjelmat.** Kirjan NR alussa, sivuilla 15-27, on hyödyllinen katsaus ohjelmissa noudatettavaan periaatteisiin. Ks. kurssimappi. Myös kurssin laskuharjoituksissa pyrimme seuraamaan kirjan suuntalinjoja tältä osin. Kirjan ohjelmiin voi parhaiten perehtyä tutkimalla ajokelpoisia pääohjelmia (vrt. erilliset ohjeet). Melkein kaikki ohjelmat käyttävät otsikkotiedostoja `nr.h` ja `nrutil.h`.

Laskuharjoituksissa tehtävät uudet määrittelyt on, jos tarpeen, laitettava omaan otsikkotiedostoonsa esim. `lh1.h`. Siinä tapauksessa, että teet omia viritelmiä kurssilla esiintyviin tiedostoihin niin muuta myös tiedostonimi. Jotta laskuharjoituksissa muiden olisi helpompi seurata ratkaisujasi, olisi parasta olla muokkaamatta NR:n/GSL:n otsikko- tms. tiedostoja tai ainakin nimetä ne uudelleen jos muutat niitä.

Yksinkertainen esimerkki NRC-ohjelmien käytöstä on seuraavassa. Alla ohjelman listaus ja tulostus. Huomaa alussa/lopussa olevat direktiivit.

```
/* FILE: mynrctst.cpp begins */
/* g++ -Wall mynrctst.cpp -I../utils -o a -lm */
/* A simple example of how to use NRC programs. */

#include <stdlib.h>
#include <cstdlib>
#include <cstdio>
```

```

#include "nr.h"
#include "nrutil.h"

int main()
{
  for (int j = 1; j <= 5; j++)
  {
    double x = 0.2 * j;
    printf("log(Gamma( %6.4f )) = %12.5e \n", x, NR::gammln(x));
  }
  return 0;
}
/* FILE: mynrctst.cpp ends */

log(Gamma( 0.2000 )) = 1.52406e+00
log(Gamma( 0.4000 )) = 7.96678e-01
log(Gamma( 0.6000 )) = 3.98234e-01
log(Gamma( 0.8000 )) = 1.52060e-01
log(Gamma( 1.0000 )) = 0.00000e+00

```

Unix:issa ohjelma voidaan kääntää usealla eri tavalla. Helpoin tapa on

```
g++ -Wall mynrctst.cpp -I../utils -o a -lm . ja ajo tapahtuu tavalliseen tapaan ./a .
```

**1.6. Ellipsin pinta-ala.** Teemme pienen ohjelman, joka pyytää käyttäjältä ellipsin puoliakselit ja antaa vastaukseksi ellipsin pinta-alan.

```

/* FILE: myarea1.cpp begins */
/* g++ myarea1.cpp -o a -lm */
/* Area of an ellipse */

#include <cstdio>
#include <cmath>
#define PI M_PI /* defines PI 3.14159... */

double area(double a, double b)
{ return PI * a * b; }

int main()
{
  double semi1, semi2, ar;

```

```

double area(double, double);
int i, n;
printf("How many ellipses? \n");
scanf("%d", &n);
for (i = 1; i <= n; i++)
{
    printf("Ellipse No %2d : semiaxis1 = ? \n", i);
    scanf("%lf", &semi1);
    printf("Ellipse No %2d : semiaxis2 = ? \n", i);
    scanf("%lf", &semi2);
    ar = area(semi1, semi2);
    printf("Ellipse No %2d : Area = %16.5e\n", i, ar);
}
return 0;
}

/* FILE: myarea1.cpp ends */

```

Ohjelman toiminta lienee luettavissa suoraan koodista.

**1.7. C-kielen matemaattisia piirteitä.** Edellä, kohdassa 1.6 käytettiin C-kielen tuntemaa vakiota `M_PI`, joka on header-tiedostossa `math.h` (C++: `cmath`). Alla on joukko sieltä löytyviä muita matemaattisia vakiota.

```

/* Constants rounded for 21 decimals. */
#define M_E          2.71828182845904523536
#define M_LOG2E     1.44269504088896340736
#define M_LOG10E    0.434294481903251827651
#define M_LN2       0.693147180559945309417
#define M_LN10      2.30258509299404568402
#define M_PI        3.14159265358979323846
#define M_PI_2      1.57079632679489661923
#define M_PI_4      0.785398163397448309616
#define M_1_PI      0.318309886183790671538
#define M_2_PI      0.636619772367581343076
#define M_1_SQRTPI  0.564189583547756286948
#define M_2_SQRTPI  1.12837916709551257390
#define M_SQRT2     1.41421356237309504880
#define M_SQRT_2    0.707106781186547524401

```

C-kielen perusteoksista löytyy myös luetteloita käytettävissä olevista matemaattisista funktioista. Tavallisimmat matemaattiset funk-

tiot kuten `sin`, `cos`, `exp`, `sinh` sekä näiden käänteisfunktiot `asin`, `acos`, `log`, `asinh` on määritelty tiedostossa `math.h` (C++: `cmath`).

Satunnaislukuja voidaan generoida käyttäen C-kielen satunnaislukugeneraattoria `rand()`. Satunnaislukuja käytetään paljon mm. testidatan tuottamiseen tälläkin kurssilla. Alla muutamia esimerkkejä satunnaislukujen käytöstä. Näistä ensimmäinen simuloi nopanheittoa. Koska satunnaislukugeneraattori toimii deterministisesti, se tuottaa samoilla "alustuksilla" samat satunnaisluvut. Tavallisesti halutaan kuitenkin satunnaislukujen muuttuvan ohjelman eri ajokerroilla. Allaolevissa ohjelmissa tähän päästään alustamalla satunnaisgeneraattori systeemikellosta.

```
/* FILE: mygenrdm.cpp begins.          */
/* g++ mygenrdm.cpp -o a -lm -lnr     */

#include <cstdlib> // Used in putmat2
#include <cstdio>  // Used in putmat2
#include <ctime>
#include <string>
#include <fstream>
#include <iostream>
#include <iomanip>
#include <climits>
using namespace std;

void Die_Casting()
{
    unsigned seed = time(NULL);
    /* Generates a seed for the random number
       generator using the system clock. */
    int i, j, freq[10];
    srand(seed);                /* Initiates the generator */
    for (i = 0; i < 10; i++)
        freq[i] = 0;
    for (i = 0; i < 40; i++)
    {
        j = 1 + rand() % 6;
        printf("%3d", j);
        if (i == 20)
            printf("\n");
        freq[j]++;
    }
}
```

```

    }
    printf("\n");
    for (i = 1; i <= 6; i++)
    {
        printf("%2d :", i);
        for (j = 1; j <= freq[i]; j++)
            printf("x");
        printf("\t : %3d\n", freq[i]);
    }
}

void Normalized_Rdm(int mi, int ma)
{
    unsigned seed = time(NULL);
    /* This generates a seed for the random number
       generator using system clock. */
    int i, j;
    srand(seed); /* This initiates the random number
                  generator with the given seed. */
    printf("Prints normalized random numbers. r=random \n");
    printf("    Rint    rint in [%3d,%3d]    rfloat in [0,1] \n",
           mi,ma);
    for (i = 0; i < 6; i++)
    {
        j = rand();
        printf("%12d %3d %20.5f\n", j,
              (int) (mi + (j * 1.0 / RAND_MAX) * (ma - mi)),
              (float) ((j * 1.0 / RAND_MAX)));
    }
    printf("RAND_MAX= %d \n", RAND_MAX);
}

void GenRdm_xyPairs()
{
    /* This program prints (x[i],y[i]) pairs, where
       x[i] < x[i+1]. Both x and y values are random numbers. */
    unsigned seed = time(NULL);
    /* This generates a seed for the random number
       generator using system clock. */
    int i;
    float s = 0.0; /* x-values are accumulated in s */
    float x[11], y[11];
    for (i = 0; i < 11; i++)
    {
        x[i] = 0.0;

```

```

    y[i] = 0.0;
}

srand(seed);                /* This initiates the random number
                             generator with the given seed. */

printf(" i      x      y \n");
for (i = 0; i < 11; i++)
{
    s += rand() * 1.0 / RAND_MAX;
    x[i] = s;
    y[i] = rand() * 1.0 / RAND_MAX;
    printf("%2d  %8.5f  %8.5f\n", i, x[i], y[i]);
}
}

int main()
{
    Die_Casting();
    Normalized_Rdm(1, 99);
    GenRdm_xyPairs();
}
/* FILE: mygenrdm.cpp ends.                                     */

/* Output:

    2 3 2 1 5 4 6 2 6 5 1 4 6 4 1 5 4 2 5 3 1
    1 1 5 3 1 1 2 4 1 5 3 1 6 4 4 3 3 5 6
1 :xxxxxxxxx : 10
2 :xxxxxx   : 5
3 :xxxxxx   : 6
4 :xxxxxx   : 7
5 :xxxxxx   : 7
6 :xxxxxx   : 5
Prints normalized random numbers. r=random
    Rint    rint in [ 1, 99]    rfloat in [0,1]
1700129767  78                0.79168
1806595916  83                0.84126
 563531767  26                0.26241
 884334330  41                0.41180
1078927036  50                0.50241
 497360169  23                0.23160
RAND_MAX= 2147483647
i      x      y
0     0.79168  0.84126
1     1.05410  0.41180

```

```
2      1.55651      0.23160
(.....)
9      5.19532      0.27466
10     6.12516      0.96904
```

```
*/
```

Edellä algoritmisesti tuotetut satunnaisluvut eivät ehkä ole satunnaislukuja todennäköisyysteorian mielessä mutta kuitenkin riittävän hyvä vastine niille tällä kurssilla.

**1.8. Syöttö ja tulostus tiedostojen avulla.** Edellä, kohdassa 1.6 ohjelma pyytää käyttäjältä syötteitä. Joskus voi olla edullista käyttää syötteen ja/tai tulostuksen uudelleen suuntaamista tiedostojen avulla. Yksinkertaisimmin se tapahtuu seuraavasti

```
myarea1 < myarea1.inp > myarea1.aux,
```

jolloin syöte luetaan tiedostosta area1.inp ja tulokset kirjoitetaan tiedostoon myarea1.aux. Tiedosto myarea1.inp voisi olla seuraava:

```
2
1.0
1.0
1.0
2.2
```

jolloin ylläoleva komento tuottaa seuraavan tiedoston myarea1.aux:

```
How many ellipses?
Ellipse No 1 : semiaxis1 = ?
Ellipse No 1 : semiaxis2 = ?
Ellipse No 1 : Area =      3.14159e+00
Ellipse No 2 : semiaxis1 = ?
Ellipse No 2 : semiaxis2 = ?
Ellipse No 2 : Area =      6.91150e+00
```

**1.9. Kirjoittaminen tiedostoon.** Tarkastelemme tällä kurssilla pelkästään tekstitiedostoja. Seuraava esimerkki osoittaa miten tiedostoon kirjoittaminen tapahtuu. Huomaa aputiedoston `wrthed.cpp` mukaanotto. Tiedoston otsikon kirjoittaminen `writeheader` on määritelty

siellä. writeheader kirjoittaa mm. päivämäärän ja kellonajan tiedoston alkuun.

```
// FILE: myex.cpp begins
#include <cstdlib>
#include <iostream>
#include <iomanip>
#include <fstream>
#include <cmath>
#include <ctime>

#include "wrthed.cpp"

using namespace std;
const char *tabfile = "myex.dat";

int main()
{
    ofstream fp(tabfile); // create myex.dat
    if (!fp)
    {
        std::cerr<< "Error in myex.cpp: can't open file."<<endl;
        exit(1);
    }
    writeheader(fp, tabfile);
    for (int i=0;i<=10;i++)
        fp << 0.1*(double)i <<" " << sin(i*0.1)<<endl;
    fp<<"\nFILE: "<<tabfile<<" ends.";
    fp.close();
    cout<<"FILE: myex.dat written."<<endl;
    return 0;
}
// FILE: myex.cpp ends
```

Ohjelman myex.cpp tuottama tiedosto on seuraava:

```
FILE: myex.dat begins. 04- Jan. 2002 11.10:03
0 0
0.1 0.0998334
0.2 0.198669
(...)
0.9 0.783327
1 0.841471
```



**1.10. Matemaattinen mallintaminen.** Matemaattisella mallintamisella tarkoitetaan reaali maailman ilmiön kuvaamista sellaisessa matemaattisessa muodossa, josta tarkasteltavat suureet voidaan ratkaista joko eksaktisti tai likimääräisesti. Usein kysymyksessä on monivaiheinen prosessi, jossa osina ovat mm. numeerisen mallin muodostaminen ja ratkaiseminen. Tavallisesti malli kuvaa ko. ilmiötä puutteellisesti ja se sisältää yksinkertaistuksia. Numeerista mallia muodostettaessa tehdään lähes aina diskretointi tai approksimointi. Tällöin jatkuva muuttuja korvataan diskreetillä ja ääretöntä approksimoidaan äärellisellä. Seuraavassa on muutama asiaa valaiseva esimerkki:

$$(1) \sum_{n=1}^{\infty} a_n = \lim_{p \rightarrow \infty} \sum_{n=1}^p a_n; \sum_{n=1}^p \text{ voidaan laskea}$$

$$(2) \int_a^b f(x) dx = \lim_{n \rightarrow \infty} \sum_{j=1}^n f\left(a + j \frac{b-a}{n}\right) \cdot \frac{b-a}{n}; \sum_1^n \text{ voidaan laskea}$$

$$(3) \prod_{j=1}^{\infty} b_j = \lim_{n \rightarrow \infty} \prod_{j=1}^n b_j; \prod_1^n \text{ voidaan laskea}$$

Perusesimerkki jatkuvasta mallista on differentiaaliyhtälö. Ne ovat erilaisissa luonnontieteen ja tekniikan sovelluksissa varsin tavallisia. Niiden avulla voidaan mallintaa mm. monia fysiikan ja kemian ilmiöitä heittoliikkeestä radioaktiiviseen hajoamiseen. Differentiaaliyhtälön numeerinen ratkaisu puolestaan tapahtuu erilaisilla tarkoitukseen kehitetyillä diskretointimenettelyillä. Eräs ensimmäisiä menetelmiä oli differentiaaliyhtälöiden kurssista tuttu Eulerin menetelmä, joka nykyään on korvattu paljon tehokkaammilla menetelmillä.

**1.11. Virhelähteistä.** Numeerisen laskennan virheet johtuvat usean erilaisen virhetekijän yhteisvaikutuksesta, joita nyt tarkastelemme.

- (1) *Mallivirheet*, Syy: yksinkertaistavat oletukset.
- (2) *Menetelmävirheet*: Syy: numeerinen malli ratkaistaan äärellisellä määrällä laskutoimituksia:

-*katkaisuvirhe*: Korvataan ääretön summa äärellisellä  $e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!} = \sum_{n=0}^k + R_{n+1}(x) \approx \sum_{n=0}^k \frac{x^n}{n!}$ , kun  $|R_{n+1}(x)|$  on pieni.

-*diskretointivirhe* Esim.  $\int_0^1 f(x) dx \approx \sum f(x_i) \Delta x$ .

(3) *Lähtöarvovirheet*: Esimerkiksi analysoitava data (kokeelliset mittaukset) on virheellinen.

(4) *Pyöristysvirheet*: Numeerisen mallin ratkaisu voi sisältää suuren määrän laskutoimituksia, joista kukin voi tuottaa (koneesta riippuvan) virheen  $\Rightarrow$  virheiden kumuloituminen laskennan aikana.

(5) *Epästabiilit algoritmit*: Menetelmävirhe voi aiheutua siitä, että käytetään *epästabiilia* algoritmia.

(6) *Inhimilliset virheet*: Ohjelmointivirheet (esim.).

(7) *Jako nolalla* voi johtaa arvaamattomiin seurauksiin, kuten myös laskennan aikana tapahtuva ”liukulukualueen rajojen ylitys”, ts. yli- tai alivuoto.

**1.12. Suhteellinen ja absoluuttinen virhe.** Vektorin  $x = (x_1, \dots, x_n) \in R^n$  approksimaatiota merk.  $\tilde{x} = (\tilde{x}_1, \dots, \tilde{x}_n) \in R^n$ . Approksimaation virhe on  $v = x - \tilde{x}$  (tai  $\tilde{x} - x$ ). Virheen normi  $\|v\|$  voidaan laskea eri tavoin:

$$\|v\|_{\infty} = \max\{|v_i| : 1 \leq i \leq n\} \quad (1)$$

$$\|v\|_1 = \sum_{i=1}^n |v_i|$$

$$\|v\|_2 = (\sum_{i=1}^n |v_i|^2)^{1/2}.$$

*Virhearvio*:  $\|x - \tilde{x}\| \leq u$ ,  $u$  virhearvio. Jos  $x \neq 0$  niin *suhteellinen virhe* on  $\frac{\|x - \tilde{x}\|}{\|x\|}$ . Tulos on ilmaistu  $p\%$  tarkkuudella, jos suhteellinen virhe  $< \frac{p}{100}$ .

Tulos on ilmaistu tarkkuudella  $\varepsilon$ , jos  $\|x - \tilde{x}\| \leq \varepsilon$ .

**1.13. Merkitsevien numeroiden kumoutuminen.** Samanmerkkisten lukujen vähennyslaskussa voi tapahtua merkitsevien numeroiden kumoutumista, joka olennaisesti huonontaa tarkkuutta. Seuraavas-

sa muutamia esimerkkejä, joissa matemaattisella identiteetillä voidaan torjua tarkkuuden huononemista.

(1)  $\sqrt{x^2 + 1} - 1 = \frac{x^2}{1 + \sqrt{1 + x^2}}$ . Kun  $x \sim 0$  on jälkimmäinen muoto parempi.

(2)  $\tan(x) - \sin(x) = \frac{1}{2}x^3 + \frac{1}{8}x^5 + \frac{13}{240}x^7 + \dots$ . Jälkimmäinen muoto parempi kun  $x \sim 0$ .

(3)  $ax^2 + bx + c = 0$  ( $a \neq 0$ ,  $b^2 - 4ac \geq 0$ ,  $b \neq 0$ )

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}.$$

Haitallista kumoutumista voi esiintyä, jos  $4ac \ll b^2$ . Oikea laskutapa

$$q = -\frac{1}{2}(b + \operatorname{sgn}(b)\sqrt{b^2 - 4ac}), \quad x_1 = \frac{q}{a}, \quad x_2 = \frac{c}{q}.$$

(4) Positiivisten lukujen  $0 < x_1 < \dots < x_n$  yhteenlasku on edullista tehdä suuruusjärjestyksessä

```
s=0; for (j=1;j<=n;j++) {s+=x[j];}
```

(5) Reaalilukujen  $x_i, i = 1, \dots, n$ , varianssille  $s_n$  on kaava

$$s_n^2 = \frac{1}{n-1} \sum_{i=1}^n (x_i - \bar{x})^2; \quad \bar{x} = \frac{1}{n} \sum_{i=1}^n x_i.$$

Kirjallisuudessa käytetään joskus myös numeerisiin tarkoituksiin huonompaa, mutta matemaattisesti identtistä kaavaa

$$s_n^2 = \frac{1}{n-1} \left( \sum_{i=1}^n x_i^2 - \frac{1}{n} \left( \sum_{i=1}^n x_i \right)^2 \right).$$

Edellä oleviin esimerkkeihin viitaten voidaan todeta, että  
**MATEMAATTINEN IDENTITEETTI  $\neq$**   
**NUMEERINEN IDENTITEETTI**

**1.14. Numeerisen algoritmin stabiilius.** Laskutoimitus tai algoritmi on *epästabiili* l. *häiriöaltis* l. *pahanlaatuinen* (unstable, sensitive,

ill-conditioned), jos pienet virheet lähtötiedoissa vaikuttavat merkittävästi lopputulokseen. Muulloin laskutoimitus tai algoritmi on *stabiili* l. *hyvänlaatuinen* (stable).

Perusesimerkki epästabiilista ongelmasta on yhtälöryhmän  $Ax = b$ , missä  $A$  on  $n \times n$  matriisi, ratkaiseminen kun  $A$  on ”huono”. Yksikäsitteinen ratkaisu ei yleensä ole mahdollista jos  $\det(A) = 0$ , siis tällaiset (singulaariset) matriisit ovat tässä mielessä huonoja. Mutta myös tapauksessa  $\det(A) \neq 0$  voi matriisi olla ”huono”. Neliömatriisin häiriöalttiutta voidaan kvantitatiivisesti mitata ns. *kuntoisuusluvulla*. Kuntoisuusluvun suuruus ( $\geq 1$ ) on parempi kvantitatiivinen mitta häiriöalttiudelle kuin  $|\det(A)|$ .

### 1.15. Esimerkki. Suorien

$$\begin{cases} y = x, & \delta = 10^{-p}, p = 2, 3, \dots \\ y = (1 + \delta)x - 1 \end{cases}$$

leikkauspiste on  $(\frac{1}{\delta}, \frac{1}{\delta})$ . Siis tehtävä on epästabiili. Suorien

$$\begin{cases} y = x, & \delta = 10^{-p}, p = 2, 3, \dots \\ y = -(1 + \delta)x + 1 \end{cases}$$

leikkauspiste on  $(\frac{1}{2+\delta}, \frac{1}{2+\delta})$ . Siis tämä tehtävä on stabiili.

**1.16. Kuvaajan piirto ASCII-merkein.** Oppikirjamme NR tarjoaa ohjelman `scrsho.cpp` kuvaajan hahmottelemiseksi ASCII-merkein. Ohjelman avulla piirrämme funktion  $y = \tan(\sin(x) + \cos(x))$  kuvaajan. Annamme käskyn:

```
g++ -Wall myascplt.cpp -L../lib -I../utils -o a -lm -lnr
```

missä `myascplt.cpp` on seuraava ohjelma:

```
/* FILE: myascplt.cpp begins */
/* g++ -Wall myascplt.cpp -L../lib -I../utils -o a -lm -lnr */
/* Plots a function with ASCII characters */
/* Modified from xscrsho.cpp/NR for NRC02 */

#include <ctime>
#include <string>
#include <fstream>
#include <iostream>
```

```

#include <iomanip>
using namespace std;
#include "nr.h"

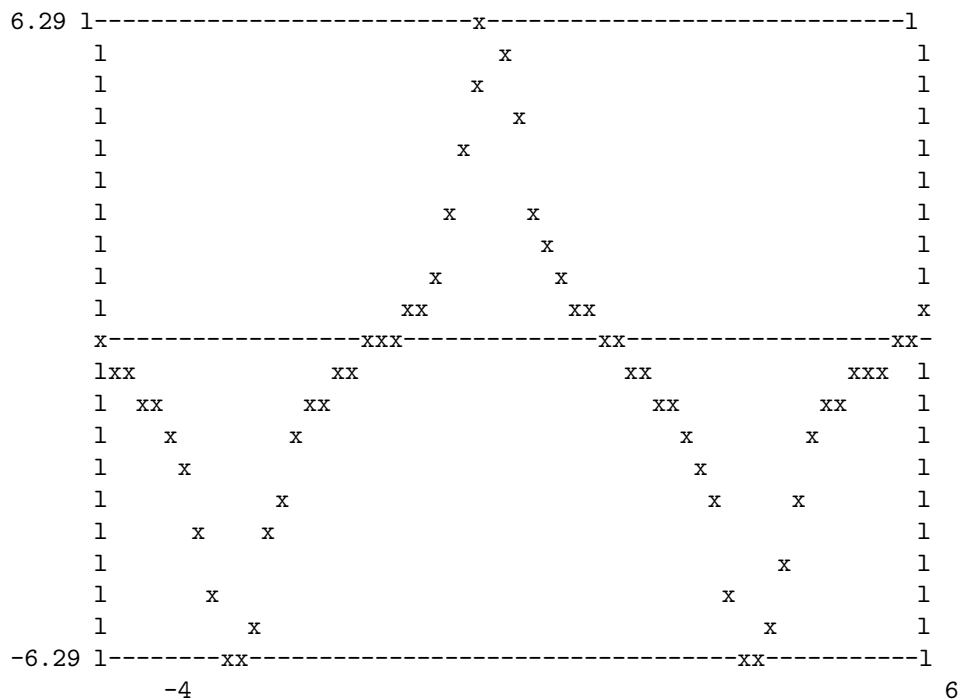
double fx(const double x)
{
    return tan(sin(x) + cos(x));
}

int main()
{
    NR::scrsho(fx);
    return 0;
}
/* FILE: myascplt.cpp ends */

```

Ajamalla ajokelpoisen ohjelman ./a saamme seuraavan kuvan:

Enter x1 x2 (x1=x2 to stop):



Enter x1 x2 (x1=x2 to stop):

**1.17. Käyrien ja pintojen piirto gnuplotilla.** Edellä esitelty kuvaajan piirtotapa soveltuu tilanteisiin, joissa riittää kuvaajan karkea hah-

mottelu. Menettely on käytettävissä kaikissa käyttöympäristöissä ja päätetyypeissä. Merkittävästi parempia kuvia saadaan julkisohjelmalla gnuplot, jolla voidaan piirtää käyriä ja pintoja. Gnuplot on asennettu mm. Matematiikan laitoksen useisiin mikroihiin ja myös Yliopiston ATK-osaston laitteisiin ja se löytyy myös RedHat Linux-asennusrumpulta.

Piirto tapahtuu kahdessa vaiheessa. Ensin funktio taulukoidaan kirjoittamalla lukuparit x y riveittäin tiedostoon tmp.dat. Sen jälkeen käynnistetään gnuplot käyttöjärjestelmän komennolla gnuplot ja piirretään kuvaaja gnuplotin käskyllä plot 'tmp.dat'.

Gnuplotin käytön helpottamiseksi C-kielen sisältä on kurssia varten tehty ohjelma gnuplt1.c, joka on saatavissa hakemistossa

/pub/nrc02/gnuplot02.

Siellä on myös muita aiheeseen liittyviä ohjelmia. Ohjelman gnuplt1 avulla voidaan samaan kuvaan piirtää enintään viiden funktion kuvaajat. Pääohjelma gnuplt1:n käyttämiseksi voisi olla vaikkapa seuraava:

```
/* FILE: myplt.cpp begins                                     */
/* g++ myplt.cpp -I../gnuplot02 -o a -lm */
/* This program uses "gnuplt1.c" to draw y=cos(x)
and y=sin(x^2) in the same picture. You can input
the x ranges to the aforementioned functions.
It is also shown how the program "plot.c"
can be used for the same purpose.
Observe that gnuplt1 generates the picture gnuplt1.ps
and that plot generates the picture plot.ps
if PRINT is set equal to 1.
N.B. this is independent of any libraries */

#include <cstdlib>
#include <cstdio>
#include <ctime>
#include <string>
#include <fstream>
#include <iostream>
#include <iomanip>
#include <math.h>
```

```

#define PRINT 1    /* If = 0 then no .ps files! */
using namespace std;

#include "gnuplt1.h"
#include "plot.h"

double f(double x)
{
    return sin(x * x);
}

void MyPlot0()
{
    const char *fname[5];
    fname[0]="z0.dat";
    fname[1]="z1.dat";
    fname[2]="z2.dat";
    FILE *fp;
    fp=fopen(fname[0],"w");
    if (fp!=NULL)
    for (int i=0;i<20;i++)
        {
            fprintf(fp,"%10.4f %10.4f \n",i*0.1, cos(i*0.1));
        }
    fclose(fp);
    fp=fopen(fname[1],"w");
    if (fp!=NULL)
    for (int i=0;i<30;i++)
        {
            fprintf(fp,"%10.4f %10.4f \n",i*0.1, f(i*0.1));
        }
    fclose(fp);
    plot(fname[0],"b-3", fname[1],"rs2",NULL);
}

int main()
{
    gnuplt1( cos, "cos(x)", 0, f, "f(x)", 2, NULL);
    MyPlot0();
    return 0;
}
/* FILE: myplt.cpp ends */

```

Edellä käytetty ohjelma gnuplt1.c taulukoi myös piirrettävät

funktiot ja kirjoittaa lopuksi taulukon tiedostoon. Alla ohjelman tuottama tiedosto tyypistettynä.

FILE: z.tmp begins. 31-Dec-2001, 0:58

Output of gnuplt1() tabulation of functions:

f0 = cos(x)

f1 = f(x)

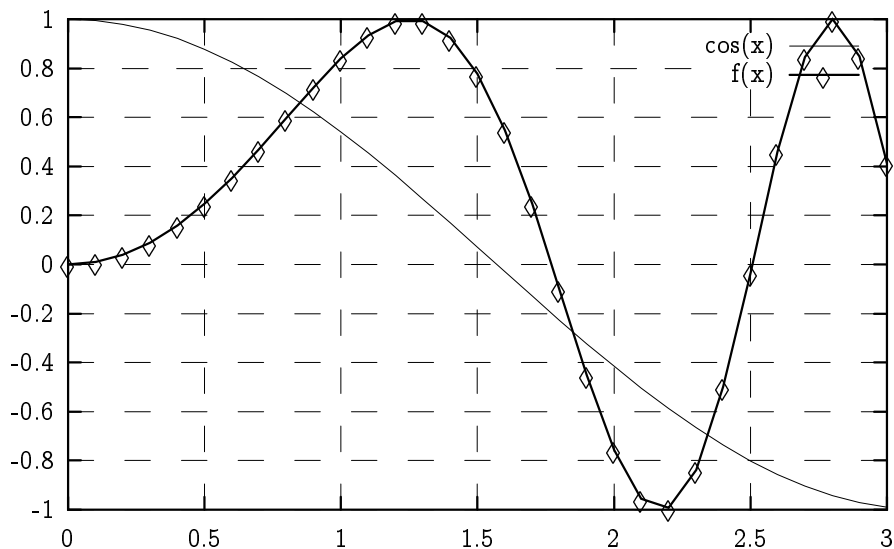
x1	x2	f1	f2
0.000000e+00	0.000000e+00	1.000000e+00	0.000000e+00
1.000000e-01	1.000000e-01	9.950042e-01	9.999833e-03
2.000000e-01	2.000000e-01	9.800666e-01	3.998933e-02
(....)			
2.900000e+00	2.900000e+00	-9.709582e-01	8.493634e-01
3.000000e+00	3.000000e+00	-9.899925e-01	4.121185e-01

Minimum of each function:

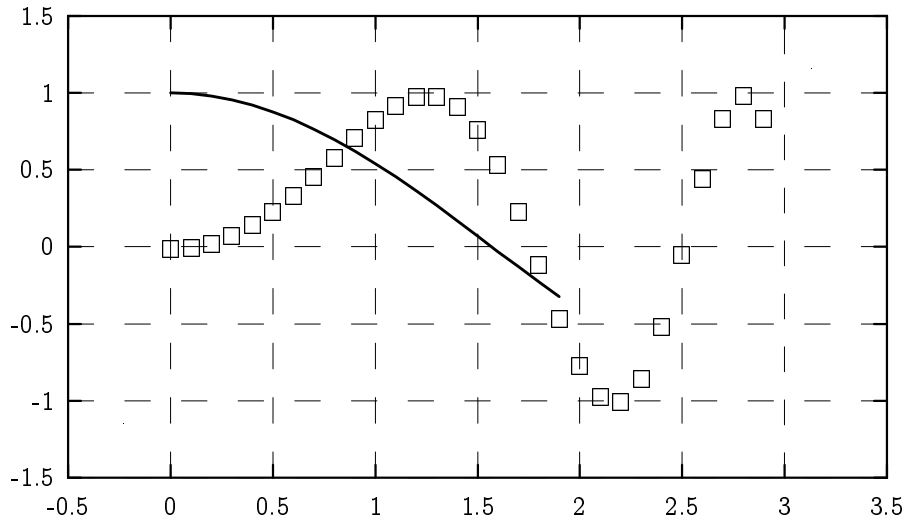
3.000000e+00 2.200000e+00 -9.899925e-01 -9.918688e-01

Maximum of each function:

0.000000e+00 2.800000e+00 1.000000e+00 9.999023e-01







Tue Jan 08 15:38:06 2002

Joissakin tapauksissa `gnuplt1` voi antaa hieman kulmikkaan kuvaajan säännölliselle funktiolle. Näin voi käydä esimerkiksi silloin, kun datapisteitä on käytettävissä vain vähän. Kulmikkauden korjaamiseksi voidaan muokata `gnuplt1:n` generoimaa tiedostoa `gnuplt1.cmd` sopivasti. Tiedoston sisältö voi olla esim. seuraava

```
set grid

plot "z1.tmp" title 'sin(x)' with lines lw 1, "z2.tmp" title 'f(x)' with linespoints
lw 1
pause -1
set terminal postscript
set output "gnuplt1.ps"
replot
```

Korjataan tiedosto seuraavasti

```
set grid

plot "z1.tmp" title 'sin(x)' with lines lw 1, "z2.tmp" smooth csplines with
lines lw 2, "z2.tmp" with points pt 3 ps 3
pause -1
```

Käynnistetään `gnuplot` Linux promptista käskyllä  
`gnuplot gnuplt1.cmd`.  
Silloin saadaan sileämpi kuvaaja.

Seuraava esimerkki käyttää jo varsin monipuolisesti NR-kirjastoa ja piirtää kuvan *Gaussin hypergeometrisesta funktiosta*. Kompleksiluvuille  $|z| < 1$  funktio määritellään sarjalla

$${}_2F_1(a, b; c; z) = \sum_{n=0}^{\infty} \frac{(a, n)(b, n)}{(c, n)n!} z^n,$$

missä  $(a, n) = a \cdots (a + n - 1)$ ,  $(a, 0) = 1$ . Monet funktiot ovat sen erikoistapauksia, mm.  ${}_2F_1(a, b; b; z) = (1 - z)^{-a}$  (tapauksessa  $a = 1$  saadaan geom. sarja) ja  ${}_2F_1(1, 1; 2; z) = -(1/z) \log(1 - z)$ . Allaolevan ohjelman avulla lasketaan funktion

$$g(z) = {}_2F_1(a, b; c; z) - (1 - z)^{-a} {}_2F_1(a, c - b; c; \frac{z}{z - 1})$$

arvoja kun  $(a, b, c) = (\frac{1}{2}, \frac{1}{2}, 1)$  ja voidaan todeta arvot nolliksi (näin pitääkin olla), kun  $z$  on reaalinen välillä  $(0, 1)$  (seuraavan ohjelman mysurf avulla voitaisiin piirtää funktion  $g(x, y)$  pinta.)

```

/* FILE: myhypgeo.cpp begins.                                     */
/* g++ myhypgeo.cpp -L../lib -I../utils -o a -lm -lnr          */

#include <cstdlib> // Used in putmat2
#include <cstdio> // Used in putmat2

#include <ctime>
#include <string>
#include <fstream>
#include <iostream>
#include <iomanip>
#define PLOT 1

using namespace std;
#include "nr.h"
#include "matutl02.h"

#include "plot.h"

int MyPlot(Vec_DP x0, Vec_DP y0)
{
    int n=x0.size();
    const char *fname[10];

```

```

fname[0]="z0.dat";
for (int j=0;j<1;j++)
{
    ofstream fp(fname[j]);
    if (fp.fail())
    {
        cout << "File " << fname[j]
            << " could not be opened.\n" << endl;
        abort();
    }
    if (j==0)
        for (int i=0;i<n;i++)
            fp<<x0[i]<<" " <<y0[i]<<endl;
    fp.close();
}
plot(fname[0],"b-2",NULL);
return 0;
}

DP Tst_2F1_idty(complex <double> a,complex <double> b,
               complex <double> c,complex <double> z)
{
    complex<DP> z1=-z/(1.0-z),q1,q2;
    q1=NR::hypgeo(a,c-b,c,z1);
    q2=pow(1.0-z,a)*NR::hypgeo(a,b,c,z);
    return( abs(q1-q2));
}
int main(void)
{
    DP x,y;
    Vec_DP xx(31), yy(31);
    complex<DP> a(0.5,0.0),b(0.5,0.0),c(1.0,0.0);
    complex<DP> z,q1;
    cout << fixed << setprecision(6);
    for (;;) {
        cout <<
            "Input X Y of Complex Argument (or 0 0 to end):" << endl;
        cin >> x >> y;
        cout << endl;
        if ((x == 0.0) && (y == 0.0)) break;
        z=complex<DP> (x,y);
        DP t=Tst_2F1_idty(a,b,c,z);
        cout << "Difference =\n" << endl;
        cout << setw(11) << t << endl;
    }
}

```

```

}
for (int i=0;i<=30;i++)
{
    xx[i]=i*0.03;
    z=complex<DP> (xx[i],0);
    q1=NR::hypgeo(a,b,c,z);
    yy[i]=real(q1); //Tst_2F1_idty(a,b,c,z);
}
MyPlot(xx,yy);
return 0;
}
/* FILE: myhypgeo.cpp ends.                                     */

Input X Y of Complex Argument (or 0 0 to end):
0.4
0

Difference =

8.88178e-16
Input X Y of Complex Argument (or 0 0 to end):
0
0

```

Edellä esitelty gnuplotin käyttötapa, joka toimii kaikilla sovel-lusaluustoilla, on tälle kurssille täysin riittävä. www-sivun esimerkki-ohjelmien mukana on myös em. ohjelman myplt.cpp. kehitty-neempi versio mypltavd.cpp joka käyttää C++:n erityispiirteitä. Linux-ympäristössä on mahdollista putkien avulla tapahtuva käyttö suoraan C-ohjelman sisältä, mutta tätä emme jatkossa käytä. Seu-raava esimerkki valaisee ohjelman gnusurf.c käyttöä pinnan piir-tämiseksi.

```

/* FILE: mysurf.cpp begin                                     */
/*g++ mysurf.cpp -L../lib -I../utils -I../gnuplot02 -o a -lm -lnr */
/*
This program uses "gnusurf" to draw the function z=x+sin(y)
and to calculate its minimum and maximum values. The user
enters the x and y ranges to the aforementioned function.

```

```

*/

#include <cstdlib>
#include <cstdio>
#include <ctime>
#include <cmath>

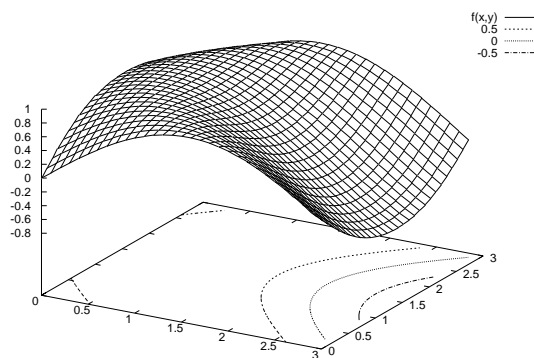
#define PRINT 1
#define SCALE 1 // Positive constant "scales" the z-axis

#include "gnusurf.h"

double f(double x, double y)
{
    return sin(x + sin(y));
}

int main()
{
    double xx[] = { -5.0, 5.0 }, yy[] = { -5.0, 5.0};
    gnusurf(f, xx, yy, SCALE, 0, "f(x,y)", "f(x,y)");
    return 0;
}
/* FILE: mysurf.cpp end */

```



Ohjelma tekee tiedoston `gnusurf.tmp`, johon kirjoitetaan myös funktion suurin ja pienin arvo lasketussa pisteistössä.

FILE: `gnusurf.tmp` begins. 31-Dec-2001, 1:19

`f = f(x,y)`

In the region  $-2.3 < x < 2.7$ ,  $-3.4 < y < 3.4$   
Minimal function value  $-0.999999$   
Attained at  $x = -1.1$ ,  $y = -2.65$

Maximal function value  $0.999983$   
Attained at  $x = 1.9$ ,  $y = -2.8$

$f = f(x,y)$

In the region  $-2.3 < x < 3.4$ ,  $-3.4 < y < 3.4$   
Minimal function value  $-9.99998e-01$   
Attained at  $x = -2.00000$ ,  $y = 2.70000$

Maximal function value  $9.99998e-01$   
Attained at  $x = 2.00000$ ,  $y = -2.70000$

Seuravaaksi piirrämme kuvan pinnasta, jolla on  $m$  kpl minimejä.

```
/* FILE: mysurf2.cpp begin                                     */
/*g++ mysurf2.cpp -L../lib -I../utils -I../gnuplot02 -o a -lm -lnr */
/*
For a random integer m in [5,MAX_NR-1] we choose m random
points (x[i],y[i]) in (0,1)x(0,1) and define f2(a,b) to
be equal to the logarithm of the distance from (a,b) to the
set of these m random points. The surface f2 is graphed.
It should have m minima!
*/

#include <cstdlib>
#include <cstdio>
#include <ctime>
#include <cmath>
#include <climits>

//#define PRINT 0
#define SCALE 1
#define MAX_NR 10
```

```

#include "gnusurf.h"

float x[MAX_NR], y[MAX_NR];    /* Global variables */

double f(double x, double y)
{
    return sin(x + sin(y));
}

double f2(double a, double b)
{
    double mi =
        pow(pow(a - x[0], 2) + pow(b - y[0], 2), 0.5), s = 0.0;
    for (int i = 1; i < MAX_NR; i++)
    {
        s = pow(pow(a - x[i], 2) + pow(b - y[i], 2), 0.5);
        if (s < mi)
            mi = s;
    }
    return log(1e-20+mi); /* log(0) is not defined ! */
}

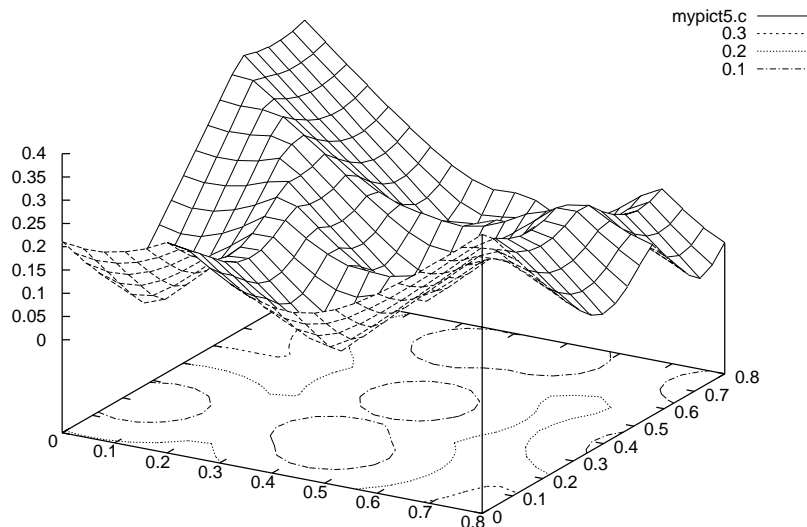
int main()
{
    double xx[] = { -5.0, 5.0 }, yy[] =
        { -5.0, 5.0};
    unsigned seed = time(NULL);
    /* This generates a seed for the random number
       generator using system clock. */
    int m, mi = 6, ma = MAX_NR - 1;
    srand(seed);
    /* This initiates the random number
       generator with the given seed. */
    m = (int) (mi + (rand() * 1.0 / INT_MAX) * (ma - mi));
    /* m is rdm integer in (mi,ma) */
    for (int i = 0; i <= MAX_NR - 1; i++)
    {
        x[i] = (float) ((rand() * 1.0 / INT_MAX));
        y[i] = (float) ((rand() * 1.0 / INT_MAX));
        if (i > m - 1)
        {
            x[i] = x[0];
            y[i] = x[0];
        }
    }
}

```

```

/* (x[i],y[i]) are random points in (0,1) */
gnusurf(f2, xx, yy, SCALE, 0, "f2(x,y)", "f2(x,y)");
return 0;
}
/* FILE: mysurf2.cpp end */

```



Ohjelma tekee tiedoston `gnusurf.tmp`, johon kirjoitetaan funktion suurin ja pienin arvo.

```
FILE: gnusurf.tmp begins. 4-Jan-2002, 21:47
```

```
f = f(x,y)
```

```
In the region 0.00000 < x < 1.00000, 0.00000 < y < 1.00000
```

```
Minimal function value 0.00000e+00
```

```
Attained at x = 0.00000 , y = 0.00000
```

```
Maximal function value 9.99994e-01
```

```
Attained at x = 0.85000 , y = 0.80000
```

**1.18. Välinpuolitusmenetelmä.** Olkoon  $f : [a, b] \rightarrow \mathbb{R}$  jatkuva funktio, jolla  $f(a)$  ja  $f(b)$  ovat erimerkkiset. Silloin  $f$ :llä on ko. välillä



ainakin yksi nollakohta. Oletamme nyt lisäksi, että funktiolla on täsmälleen yksi nollakohta  $x_0$ . Silloin likiarvo  $x_0$ :lle voidaan löytää seuraavaavan koulukurssista tunnetun *välinpuolitusmenetelmän* avulla. Olkoon  $c = (a+b)/2$ . Jos  $f(a)f(c) < 0$ , asetetaan  $b$ :lle arvoksi  $c$ . Jos  $f(b)f(c) < 0$ , asetetaan  $a$ :lle arvoksi  $c$ . Jos  $|a - b|$  tai  $|f(c)|$  on kyllin pieni, on löydetty riittävän hyvä likiarvo nollakohdalle. Muussa tapauksessa toistetaan iteraatiota uusilla  $a$ :n ja  $b$ :n arvoilla. On selvää, että samalla menetelmällä voidaan etsiä myös yhtälön  $f(x) = y$  juuri annetulla  $y$ , ts. löytää käänteisfunktion arvo. Yksityiskohdat ilmenevät allaolevasta koodista, jossa juurenhaun lisäksi samalla algoritmilla etsitään todennäköisyyslaskennasta tutun normaalijakauman kertymäfunktion  $\Phi(x)$  käänteisfunktio.

```

/* FILE: mybisect.cpp begins                                     */
/* g++ mybisect.cpp -I../utils -L../lib -o a -lm -lnr */
/* This program computes the inverse function of a given
   function f, i.e. for a fixed y, solves f(x)=y for x.
   This algorithm is based on bisection (see D. Yang: C++ and
   object oriented numeric computing, Springer2001, p.164).
*/
#include <cmath>
#include <iostream>
#include <cstdio>
#include <cstdlib>
#include <iomanip>
using namespace std;
#include "nr.h"
#include "plot.h"
double invf(double a, double b, double y, double (*f)(double),
            double delta, double epsn, int maxit)
/*
invf returns a value x on the interval (a,b) such that f(x)=y.
Based on bisection of the interval (a,b) such that x remains
inside. Stopping criterion: iteration stopped when one of
the following is satisfied:
    (1)interval length <delta
    (2)number of iterations >maxit
    (3)residual tolerance < epsn
NB. c=a+e, e=(b-a)/2 and (a+b)/2 are mathematically
equivalent but the former is preferable because of its
numerical properties.

```

```

*/
{
    double u = f(a)-y;           // fcn value at left pt
    double e = b - a;           // interval length
    double c = b;
    for (int k = 1; k <= maxit; k++) // main iteration loop
    {
        e *= 0.5;                // shrink interval by half
        c = a + e;                // update middle pt
        double w = f(c)-y;        // fcn value at middle pt
        if
            ((fabs(e) < delta) || ( fabs(w) < epsn) || (k==maxit)) return c;
            ((u > 0 && w < 0) || (u < 0 && w > 0)) ? (b = c) :
                (a = c, u = w);
    }
    return c;
}
}

```

```

double fb(double x)
{return (x - exp(-x)); }

```

```

double fc(double x)
{return (exp(x)); }

```

```

double Phi(double x) /* Phi(t) = P({x <t}) with
                    x in N(0,1) */
{ return 0.5*(1+erff(x/pow(2.0, 0.5)));}

```

```

void Tabulate()
{
    double del=1e-15, eps=1e-15;
    for (int i=1;i<=20;i++)
    {
        double x=i*0.05;
        double y=fc(x);
        double t=invf(-1.0,2.0,y,fc,del,eps, 60);
        printf("%10.6f %10.6f %10.6f %12.4e \n",x,y,t,t-x);
    }
    cout<<"  p      InvPhi(p)      Error:"<<endl;
    for (int i=1;i<20;i++)
    {
        double y; //Phi(x);
        (i>10)?(y=0.9+(i-10)*0.01):y=0.5+(i-1)*0.05;
        double t=invf(-1.0,8.0,y,Phi,del,eps, 200);
    }
}

```

```

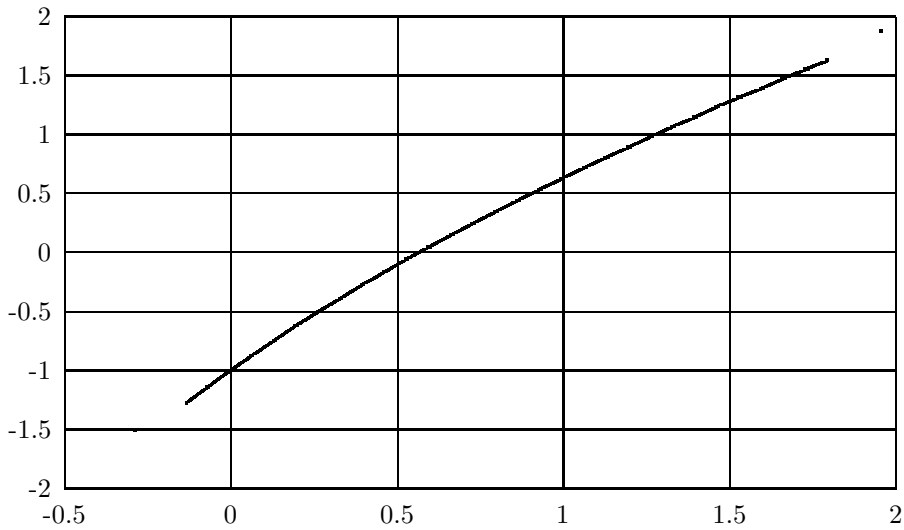
        printf("%10.6f %10.6f %12.4e  \n",y,t, Phi(t)-y);
    }
    cout<<" InvPhi(0.99995)= "
        <<invf(0, 8, 0.99995,Phi,del,eps,200)<<endl;
}

int MyPlot(Vec_DP x,Vec_DP y)
{
    int n=x.size();
    const char *fname="a.dat";
    ofstream fp(fname);
    if (fp.fail())
        {cout << "File " << fname << " could not be opened.\n"<<endl;
        abort();}
    for (int i=0;i<n;i++) fp<<x[i]<<" " <<y[i]<<endl;
    fp.close();
    plot(fname,"b-4",NULL);
    return 0;
}

int main() {
    double epsn=1e-5, delta=epsn, root, y=0.0;
    root = invf(1e-2, 2.0,y, fb, delta, epsn, 20);
    cout << "Approximate root of fb()-y=0 by invf() is: "
        << root << '\n';
    cout << "Fcn value at approximate root (residual) is:"
        << fb(root)-y << '\n';

    Vec_DP x(30),yval(30);
    for (int i=0;i<30;i++)
    {
        x[i]=root-1+(2.0/30)*i;
        yval[i]= fb(x[i]);
    }
    MyPlot(x,yval);
    Tabulate();
}
/* FILE: mybisect.cpp ends */

```



Sat Jan 05 11:40:49 2002

Approximate root of fb()-y=0 by invf() is: 0.567144  
 Fcn value at approximate root (residual) is:1.78914e-06

0.050000	1.051271	0.050000	2.6368e-16
0.100000	1.105171	0.100000	8.3267e-17
(. . . .)			
0.950000	2.585710	0.950000	-3.3307e-16
1.000000	2.718282	1.000000	-4.4409e-16
p	InvPhi(p)	Error:	
0.500000	0.000000	1.7716e-16	
0.550000	0.125661	3.4499e-09	
(. . . .)			
0.970000	1.880794	3.4438e-09	
0.980000	2.053749	-2.8952e-09	
0.990000	2.326348	7.6522e-10	
InvPhi(0.99995)= 3.89059			

**1.19. Kiintopisteiteraatio.** Toinen erittäin yksinkertainen menetelmä juuren etsimiseksi on kiintopisteiteraatio. Olkoon  $f : R \rightarrow R$  jatkuva funktio ja  $x_0 \in R$ . Määritellään  $x_{n+1} = f(x_n), n = 0, 1, \dots$ . Millä ehdolla jono  $(x_n)$  suppenee? Ns. Banachin kiintopistelauseen nojalla eräs riittävä ehto on, että on olemassa vakio  $L \in (0, 1)$  siten,

että kaikki pisteet sijaitsevat välillä  $[a, b]$  ja  $|f(x) - f(y)| \leq L|x - y|$  kaikilla  $x, y \in [a, b]$ . Jonon rajapiste  $\bar{x}$  toteuttaa yhtälön  $f(\bar{x}) = \bar{x}$ .

Seuraava algoritmi käyttää tätä ideaa. Joskus on syytä konvergenssin takaamiseksi soveltaa algoritmia muokattuun funktioon, jolla on sama juuri.

```

/* FILE: myfxit.cpp begins                                     */
/* g++ myfxit.cpp -o a -lm                                   */
/* Fixed point iteration  $x_{n+1}=f(x_n)$  to solve  $f(x)=x$ . */

#include <stdlib.h>
#include <cstdlib>
#include <stdio.h>
#include <ctime>
#include <string>
#include <fstream>
#include <iostream>
#include <iomanip>
#include <cmath>

#include "gnuplt1.h"

double f(double x)
{
    return pow(1+0.5*pow(x,2.0),-1.0);
}

double idty(double x)
{
    return x;
}

int main()
{
    double x, y, e, tol = 5.0e-10;
    int count = 0;
    printf("Enter the starting approximation:");
    scanf("%lf", &x);
    printf("\n");

    do
    {

```

```

    y = f(x);
    e = fabs(x - y);
    printf("%10.5lf %10.5lf  %10.4e\n",x,y,e);
    x = y;
    count++;
    if (count > 60)
    {
        printf("Maximum number of iterates exceeded.\n");
        exit(1);
    }
}
while (e > fabs(y) * tol);

printf("root= %lf\n", y);
printf("iterates required = %d\n", count);
gnuplt1(f,"f(x)",4,idty,"id(x)",23, NULL);
return 0;
}
/* FILE: myfxit.cpp ends */

```

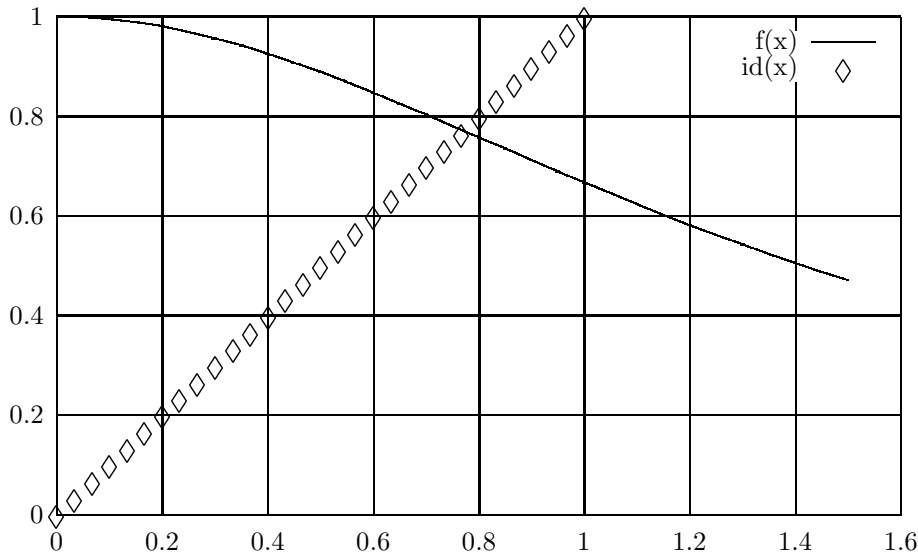
Enter the starting approximation: 2

Enter the starting approximation:2

```

    2.00000    0.33333    1.6667e+00
    0.33333    0.94737    6.1404e-01
    (.....)
    0.77092    0.77092    8.5546e-10
    0.77092    0.77092    3.9194e-10
    0.77092    0.77092    1.7958e-10
root= 0.770917
iterates required = 30

```



**1.20. Numeerinen integrointi.** Jatkuvan funktion  $f : [a, b] \rightarrow R$  integraalia  $\int_a^b f(x)dx$  voidaan approksimoida useilla eri tavoilla. Yksinkertaisinta lienee käyttää Riemannin summia  $R_n(f) = \sum_{j=0}^{n-1} f(x_j)h$  jakopisteinä  $x_j = a + j * h, j = 0, \dots, n - 1$  missä  $h = (b - a)/n$ . Lähes yhtä yksinkertainen menetelmä on puolisuunnikaskaava (eli trapetsikaava)  $T_n(f) = \sum_{j=0}^{n-1} (f(x_j) + f(x_{j+1}))h/2$ . Riemannin summia muodostettaessa funktiota approksimoidaan välillä  $[x_j, x_{j+1}]$  vakioarvolla  $f(x_j)$ , trapetsikaavassa keskiarvolla  $(f(x_j) + f(x_{j+1}))/2$ . Parempia approksimoiteja etsittäessä voidaan käyttää kaavaa  $w_1 f(x_j) + w_2 f(x_{j+1}) + w_3 f(x_{j+2})$  välillä  $[x_j, x_{j+2}]$ , missä kertoimet  $w_1, w_2, w_3$  valitaan määräämättömien kertoimien menetelmällä:

$$\int_0^{2h} 1 dx = 2h = (w_1 + w_2 + w_3) * 2h$$

$$\int_0^{2h} x dx = 2h^2 = (w_1 * 0 + w_2 * h + w_3 * 2 * h) * 2h$$

$$\int_0^{2h} x^2 dx = 8h^3/3 = (w_1 * 0 + w_2 * h^2 + w_3 * (2 * h)^2) * 2h$$

josta  $w_1 = \frac{1}{6}, w_2 = \frac{2}{3}, w_3 = \frac{1}{6}$ . Tähän perustuu Simpsonin kaava, jota alla verrataan kokeellisesti trapetsikaavaan.

```

/* FILE: mynumint.cpp begins. */
/* g++ mynumint.cpp -L../lib -I../utils -o a -lm -lnr */
/* Compares numerical integration by Simpson's method and */
/* the trapezoid formula. */

#include <cstdlib> // Used in putmat2
#include <cstdio> // Used in putmat2
#include <ctime>
#include <string>
#include <fstream>
#include <iostream>
#include <iomanip>
using namespace std;
#include "nr.h"
#include "matut102.h"

double c[5]; /* Global variable */

double f(double x)
{ return c[0]*sin(c[1]*x)+ c[2]*2.0*x; }

double prim_of_f(double x)
/* \int_a^b f(x) dx = prim_of_f(b)- prim_of_f(a) */
{ return -(c[0]/c[1])*cos(c[1]*x)+c[2]*x*x; }

double trapez(int m, double a, double b,
              double (*func_ptr) (double ))
/* Trapezoidal formula for numerical integration */
{
    double h=(b-a)/m;
    double s=(*func_ptr)(a)+(*func_ptr)(b);
    s=-0.5*s;
    for (int i=0;i<=m;i++){
        s +=func_ptr(a+i*h);
    }
    s *= h;
    return s;
}

double Simpson(int m, double a, double b,
              double (*func_ptr) ( double))
/* Simpson's formula for numerical integration */
/* m = number of subdivisions
   (a,b) interval of integration */
{

```



```

double h=(b-a)/m;
double s=(*func_ptr)(a)+(*func_ptr)(b);

for (int i=1;i<=m-1;i++){
    s += (4.0*(*func_ptr)(a+(i-.5)*h)+
          2.0*(*func_ptr)(a+i*h));
}
s += (4.0*(*func_ptr)(a+(m-.5)*h));
s *= (h/6.0);
return s;
}

void DoTst(int num_subdiv, double a,double b)
{
    int m=num_subdiv;
    double s1,s2,err1,err2;
    /* function prototype */
    double Simpson(int m, double a, double b,
                   double (*func_ptr) (double ));
    /* definition and initialization */
    double (*func_ptr)(double)=f;
    s1=Simpson(m, a, b, func_ptr);
    s2=trapez(m, a, b, func_ptr);
    err1= s1-(prim_of_f(b) -prim_of_f(a) );
    err2= s2-(prim_of_f(b) -prim_of_f(a) );
    printf("% 15.10lf % 8.4e % 15.10lf % 8.4e\n",
           s1,err1,s2,err2);
}

int main()
{
    init_srand();
    int num_subdiv=500;
    double a=0, b=1;
    printf(" Simpson           Error           Trapez           Error\n");
    for (int i=1;i<=5;i++)
    {
        for (int j=0;j<5;j++) c[j]=rdm(0.5, 2.5);
        a= rdm(-2, 2.5); b=a+rdm(0.5, 2.5);
        DoTst(num_subdiv,a,b);
    }
}
/* FILE: mynumint.cpp ends.                                     */

```

Simpson	Error	Trapez	Error
-5.7622377489	-3.9968e-15	-5.7622376068	1.4209e-07
24.8493022581	-7.0344e-13	24.8493086674	6.4094e-06
-2.3825109332	-8.8352e-13	-2.3825069902	3.9430e-06
15.9783696594	-5.3557e-13	15.9783725920	2.9326e-06
25.9879374125	3.5527e-15	25.9879363855	-1.0270e-06

**1.21. Suoran sovitus dataan.** Eräs perustehtävä tilastollisessa data-analyysissä on sovittaa suora datapisteisiin  $(x_j, y_j), j = 1, \dots, m,$   $x_j < x_{j+1}$ . Seuraava ohjelma tekee sovituksen ja piirtää datan ja sovitetun suoran kuvaajat.

```

/* FILE: mylsq.cpp begins.                                     */
/* g++ -Wall mylsq.cpp -L../lib -I../gnuplot02 -I../utils -o a -lm -lnr */

#include <stdlib.h>
#include <cstdlib> // Used in putmat2
#include <cstdio> // Used in putmat2
/*
#include <ctime>
#include <string>
#include <fstream>
#include <iostream>
#include <iomanip>
*/
#define PRINT 1

using namespace std;
#include "nr.h"
#include "matut102.h"
#include "plot.h"

int MyPlot(Vec_DP x0,Vec_DP y0, Vec_DP x1,Vec_DP y1,
           Vec_DP x2,Vec_DP y2)
/*
Plots three curves x0,y0, x1,y1, x2, y2.
Here x0, y0 have equal length likewise for x1,y1 and x2,y2.
But the lengths of x0 and x1 need not be the same.
*/
{
    int n0=x0.size(), n1=x1.size(),n2=x2.size();
    const char *fname[10];
    fname[0]="z0.dat";

```

```

fname[1]="z1.dat";
fname[2]="z2.dat";
for (int j=0;j<3;j++)
{
    ofstream fp(fname[j]);
    if (fp.fail())
        {cout << "File " << fname[j]
            <<" could not be opened.\n"<<endl;
            abort();}
    if (j==0) for (int i=0;i<n0;i++)
        fp<<x0[i]<<" "<<y0[i]<<endl;
    if (j==1) for (int i=0;i<n1;i++)
        fp<<x1[i]<<" "<<y1[i]<<endl;
    if (j==2) for (int i=0;i<n2;i++)
        fp<<x2[i]<<" "<<y2[i]<<endl;
    fp.close();
}
plot(fname[0],"b-3",fname[1],"r-4",fname[2],"ks3",NULL);
return 0;
}

```

```

void LSQcoef(Vec_DP &x, Vec_DP & y, DP &a, DP &b )
/* For given data x, y fits a line a*t + b */
{
    double s1=0.0,s2=0.0,s3=0.0,s4=0.0, s5=0.0, xave;
    int m=x.size();
    for (int i =0;i<m;i++)
    {
        s1+=x[i];
        s2+=y[i];
        s3+=x[i]*y[i];
        s4+=x[i]*x[i];
    }
    s5=0.0; xave=(1.0*s1)/m;
    for (int i =0;i<m;i++)
        { s5+=pow(abs(x[i]- xave), 2.0);}
    /* The above method for computing s5 is numerically
        better than the formula s5=s4-(1.0/m)*s1*s1;.
        Mathematically both are the same.
    */
    if (s5<1e-20)
        {cout<<"Data with zero variance in LSQcoef"<<endl;
            s5=1e-20;
        }
}

```

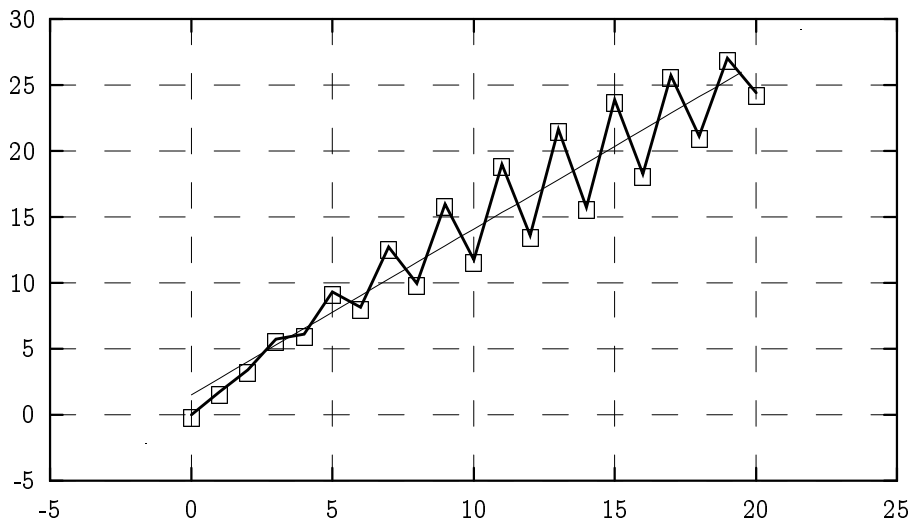
```

a=1.0*(s3-(1.0/m)*s1*s2)/(1.0*s5);
/*
   If you replace above (1.0/m) by (1/m)
   you will not get the correct result!
*/
b=(s2-a*s1)/m;
}
int main()
{
  int m=21;
  Vec_DP x(m), y(m), xi(40), yi(40);
  double a,b;
  for (int i=0;i<m;i++)
    {x[i]= i*1.0;
     y[i]= x[i]+ log(1.0+x[i]*x[i]);
     y[i]= y[i]*(1.0+ 0.2*sin(3*x[i]));
    }
  LSQcoef(x, y, a, b );
  for (int i=0;i<40;i++)
    {
      xi[i]=i*0.5;
      yi[i]=a*xi[i] + b;
    }
  cout<<"a= "<<a<<" , b= "<<b<<endl;
  MyPlot(x,y,xi,yi,x,y);
}
/* FILE: mylsq.cpp ends.                                     */

```

OUTPUT:

a= 1.25512, b= 1.51009



Tue Jan 08 19:54:40 2002

**1.22. Monte Carlo-integrointi.** Monte Carlo-integrointi on nimetty Monacossa sijaitsevan kasinon mukaan. Lähtökohtana on suorakulmiossa

$$B = [a_1, b_1] \times [a_2, b_2] \times \dots \times [a_m, b_m]$$

sijaitseva kappale  $K$ , jonka tilavuus  $v(K)$  halutaan määrittää. Kappaleen määrittelee sen karakteristinen funktio  $\chi_K(x)$ , joka saa arvon 1 kun  $x \in K$  ja 0 muulloin. Muodostamme tasaisen jakauman mukaan  $n$  pistettä suorakulmiosta  $B$  ja olkoon  $p$  niiden pisteiden lukumäärä, joissa  $\chi_K(x)$ , saa arvon 1. Silloin

$$v(K) \approx \frac{p}{n}v(B).$$

Seuraavalla ohjelmalla laskemme yksikkökuulan tilavuudelle likiarvoja ja vertaamme tulosta tarkkaan arvoon.

```

/* FILE: mymoncar.cpp begins.                                     */
/* g++ mymoncar.cpp -L../lib -I../utils -o a -lm -lnr          */

#include <cstdlib> // Used in putmat2
#include <cstdio> // Used in putmat2
#include <ctime>
#include <string>
#include <fstream>

```

```

#include <iostream>
#include <iomanip>
using namespace std;
#include "nr.h"
#include "matutl02.cpp"

long tstRes(int Ndim, long howMany)
{
    int i=0,count=0;
    double tmp,tmp2;
    Vec_DP x(Ndim);
    while (i <=howMany)
    {
        for (int k=0;k<Ndim;k++)
        {
            x[k]=(float)rdm(0.0, 1.0);
        }
        if (vnormp(x,2.0)<1.0) count++;
        // vnormp is part of matutl02.cpp
        i++;
    }
    return count;
}

int BallVolume(void)
{
    long k, i=0,count =0,np ,nrt;
    double tmp,tmp2;
    init_srand();
    printf(
"\n n    2000    4000    6000    8000    10000  Expected\n");
    for (np=2;np<=8;np++)
    {
        printf(" %2ld",np);
        for (nrt=2; nrt<=10;nrt+=2)
        {
            count=tstRes(np,nrt*1000);
            printf("%8.4f",count*0.001/nrt);
        }
        tmp=pow(M_PI,0.5*np)/exp(NR::gammln(1.0+0.5*np));
        printf("%8.4f \n", tmp*pow(2.0,-(double)np));
    }
    return 0;
}

```

```

int main()
{
    BallVolume();
}
/* FILE: mymoncar.cpp ends. */

/*

n    2000    4000    6000    8000    10000    Expected
2    0.7930    0.7790    0.7893    0.7925    0.7811    0.7854
3    0.5250    0.5185    0.5285    0.5214    0.5246    0.5236
4    0.3045    0.3038    0.3088    0.3040    0.3049    0.3084
5    0.1750    0.1598    0.1630    0.1540    0.1624    0.1645
6    0.0840    0.0850    0.0903    0.0780    0.0821    0.0807
7    0.0300    0.0338    0.0378    0.0385    0.0395    0.0369
8    0.0135    0.0158    0.0175    0.0124    0.0146    0.0159

*/

```

**1.23. Eulerin menetelmän epästabiilisuus.** Alkuarvotehtävän  $y' = -100y + 100$ ,  $y(0) = y_0$  ratkaisu on  $y(t) = (y_0 - 1)\exp(-100 * t) + 1$ . Eulerin menetelmää kiinteällä askelpituudella  $h$  voidaan käyttää sen ratkaisun numeeriseen approksimointiin:

$$x_0 = 0; \quad y_0 = y_0; \quad x_{k+1} = x_k + h;$$

$$y_{k+1} = y_k + h * (100 - 100 * y_k) = (1 - 100 * h) * y_k + 100 * h.$$

Jos  $y_0 = 2$  niin ratkaisu on  $y(t) = 1 + \exp(-100 * t)$ .

Induktiolla nähdään, että em. Eulerin rekursio johtaa kaavaan:  $y_k = (y_0 - 1)(1 - 100 * h)^k + 1$ . Siinä tapauksessa, että  $|1 - 100h|$  on kyllin pieni Eulerin menetelmä näyttää toimivan hyvin, kun taas arvoilla  $h > 0.02$  divergoivan. Tämä ilmenee myös ohjelman tulostuksesta.

```

/* FILE: myeuler.cpp begins */
/* g++ -Wall myeuler.cpp -o a */
/* This program demonstrates the instability
of Euler's method */

```

```

#include <stdio>

double myf(double x, double y)
{
    return (1.0-y)*100.0;
}

int main(void)
{
    double h=0.005, x=0.0,y;
    printf("Instability of Euler's method\n");
    for (int j=1;j<=12;j++)
        {
            x=0.0;y=2.0;
            h=h+0.005;
            printf("-----\nh= %6.3lf\n-----\n"
                " %6s  %12s\n",h,"x","y");
            while (x<=1.0)
                {
                    x=x+h;
                    y=y+h*myf(x,y);
                    printf("%8.4lf %15.4lf \n",x,y);
                }
        }
    return 0;
}
/* FILE: myeuler.c ends */

```

TULOSTUSTA:

Instability of Euler's method

-----  
h= 0.010  
-----

x	y
0.0100	1.0000
0.0200	1.0000
0.0300	1.0000
0.0400	1.0000
(....)	
0.9800	1.0000
0.9900	1.0000
1.0000	1.0000

-----  
h= 0.015



```

-----
      x           y
0.0150      0.5000
0.0300      1.2500
0.0450      0.8750
0.0600      1.0625
0.0750      0.9688

```

```

(....)
0.9450      1.0000
0.9600      1.0000
0.9750      1.0000
0.9900      1.0000
1.0050      1.0000

```

```

-----
h= 0.020
-----

```

```

      x           y
0.0200     -0.0000
0.0400      2.0000
0.0600     -0.0000
0.0800      2.0000
0.1000     -0.0000

```

```

(....)

0.9200      2.0000
0.9400     -0.0000
0.9600      2.0000
0.9800     -0.0000
1.0000      2.0000

```

```

-----
h= 0.025
-----

```

```

      x           y
0.0250     -0.5000
0.0500      3.2500
0.0750     -2.3750
0.1000      6.0625
0.1250     -6.5938

```

```

(....)
0.8750 -1456108.6060
0.9000  2184165.4091
0.9250 -3276245.6136
0.9500  4914370.9204
0.9750 -7371553.8806

```

```

1.0000 11057333.3209
-----
h= 0.030
-----
      x          y
0.0300      -1.0000
0.0600       5.0000
0.0900      -7.0000
0.1200      17.0000
0.1500     -31.0000
(. . . .)
0.9300 -2147483647.0000
0.9600 4294967297.0000
0.9900 -8589934591.0000
1.0200 17179869185.0001

```

## VEKTORIT JA MATRIISIT

**1.24. NRC-ohjelmien käyttöohjeet.** Kutakin NR:n aliohjelmaa kohti on pääohjelma, joka demoaa aliohjelman käyttöä. Nämä pääohjelmat tekijät ovat koonneet kirjaksi NR Example Book, jonka kopio on 5. krs:n kurssimapissa. Samassa mapissa on myös kurssia varten tehtyjä erilaisia ohjeita mm. kirjaston laatimisesta, jotka löytyvät myös kurssin kotisivulta.

**1.25. Vektori ja matriisi tietorakenteina NR:n mukaan.** Vektori ja matriisi on määritelty C++:n luokiksi, joiden metodeina ovat monet numeerisen lineaarialgebran perusoperaatiot. Luokkien ilmentymät määritellään seuraavaan tyyliin.

```

Mat_DP a(5,3);
Vec_DP x(3);
for (int i=0;i<x.size();i++) x[i]= i*3.2;
for (int i=0;i<a.nrows();i++)
for (int j=0;j<a.ncols();j++) a[i][j]= 1.0/(i+j+1);
cout <<"a= " <<a <<endl;
Vec_DP z(5);
matvecmul(a,x,z);
cout<<"z=" <<z <<endl;

```

Seuraavassa on luonnosteltu matriisien manipulointiin soveltuvia toimintoja sekä annettu esimerkki näiden toimintojen käytöstä pääohjelmassa `myutl4.cpp`.

Huomaa, että mukana on “satunnaismatriiseja” generoivia toimintoja kuten `ranmat`. Joskus voi käydä niin, että eri ajokerroilla generoituukin sama satunnaismatriisi. Jos tätä ei haluta, pitää satunnaislukugeneraattori käynnistää uudelleen. Tämä tapahtuu komennolla `init_srand()` joka on määritelty kurssin `www`-sivulta löytyvässä tiedostossa `matutl02.cpp`.

```
/* FILE matutl02.cpp begins      */
/* Last updated 2002-01-10     */
/*                               */
#include <ctime>
#include <string>
#include <fstream>
#include <iostream>
#include <iomanip>

/*
get_int prompts the user to enter an integer
in the given range [low, up].
*/
void get_int(int &m,int low,int up)
{
    m=low-1;
    int count=0;
    while (((m<low)||(m>up))&&(count<3))
        {cout<< "Please enter integer in ["
            <<low<<","<< up<<"]: "<<endl;
            cin>>m; count++;}
    if ((m<low)||(m>up))
        {cout <<"Input failed."<<endl;
        abort();}
}

/*
getfname uses the seed fname to generate a name for
a file which does not exist already.
*/
char *getfname(char *fname)
```

```

{
    FILE *fp;
    int count=1;
    char *name,*beg,*end,*point;
    name=(char *) malloc(20);
    beg=(char *) malloc(20);
    sprintf(name,"%s",fname);
    sprintf(beg,"%s",fname);
    point=strchr(beg,'.');
    if(point!=NULL)
        *point='\0';
    end=".dat";
    while(!((fp=fopen(name,"r"))==NULL)){
        sprintf(name,"%s%d%s",beg,count++,end);
        fclose(fp);
    }
    return name;
}

/*
    init_srand initializes of the random number
    generator with system clock.
*/
void init_srand()
{ unsigned seed=time(NULL);
  srand(seed); }

/*
    rdm produces a random number in the range [low,up].
*/
DP rdm(DP low, DP up)
{ DP x=rand(), der=DP( (up - low)/RAND_MAX);
  return (der*x + low); }

/*
    ranmat2 produces a random matrix with entries
    in the range [low,up]. The space for the matrix
    must be reserved in advance.
*/
void ranmat2(Mat_DP &a, DP low, DP up)
{ int m= a.nrows(),n=a.ncols();
  for (int i = 0; i <= m-1; i++)
    { for (int j = 0; j <= n-1; j++)
      {
          a[i][j] = rdm(low, up);
      }
    }
}

```

```

    }
}

/*
entermat
prompts the user to enter a matrix entry by entry and
displays the input on the screen. Matrix size must be
known in advance and the space reserved for the matrix.
*/
void entermat( Mat_IO_DP &a)
{
    for (int j=0;j<a.nrows();j++)
        for (int k=0;k<a.ncols();k++)
            {cout << "Enter A[" <<j<<","<<k<<"]: ";
             cin>>a[j][k];
             cout << "The input you gave: "<<a[j][k]<<endl;
            } ;
}

/*
entermat2
prompts the user to enter a matrix entry by entry and
displays the input on the screen. Mat_DP a(1,1);
must be defined in the calling program.
On return, after the call a=entermat2();
a has the dimensions given.
*/

Mat_DP entermat2(const int low,const int up)
{
    int m,n;
    cout<<"Enter matrix size (m,n):";
    get_int(m,low,up);
    get_int(n,low,up);
    Mat_DP a(m,n);
    for (int j=0;j<a.nrows();j++)
        for (int k=0;k<a.ncols();k++)
            {cout << "Enter A[" <<j<<","<<k<<"]: ";
             cin>>a[j][k];
             cout << "The input you gave: "<<a[j][k]<<endl;
            } ;
    return a;
}

```

```

/*
    showvec displays a vector on the screen.
*/
void showvec( Vec_DP &a)
{
    cout.precision(8);
    cout<<endl<<"A vector with "<< a.size()<< " components:" <<endl;
    for (int j=0;j<a.size();j++)
        cout <<a[j]<<" ";
    cout<<endl;
}

/*
    showmat displays a matrix on the screen.
*/
void showmat( Mat_DP &a)
{
    int m=a.nrows(), n=a.ncols();
    cout.precision(8);
    cout<<endl<< m<< "x" <<n <<" matrix: "<<endl;
    for (int j=0;j<m;j++)
        for (int k=0;k<n;k++)
            {cout <<a[j][k]<<" ";
             if (k==n-1) cout<<endl;}
}

/*
    showmat2 displays a matrix on the screen with
    a specified format .
*/
void showmat2( Mat_DP &a, char *fmt)
{
    int m=a.nrows(), n=a.ncols();
    cout<<endl<< m<< "x" <<n <<" matrix: "<<endl;
    for (int j=0;j<m;j++)
        for (int k=0;k<n;k++)
            {printf(fmt,a[j][k]);
             if (k==n-1) cout<<endl;}
}

/*
    putmat stores a matrix in a file given by fname.
    Format as in getMat.
*/
void putmat( Mat_DP &a, char *fname)
{

```

```

DP luku=0.0;
int m=a.nrows(), n=a.ncols();
ofstream fp(fname);
if (fp.fail())
{
    cout << "File " << fname
        << " could not be opened.\n" << endl;
    abort();
}
fp << m << " " << n << endl;
fp.precision(9);
fp.width(16);
for (int i=0; i<m; i++)
for (int j=0; j<n; j++)
{
    luku=a[i][j];
    fp << luku << " ";
    // cout << luku << " ";
    if (j==n-1){fp << endl; cout << endl;}
}
cout << "Matrix written in file "
    << fname << endl << "=====" << endl;
fp.close();
}

/* putmat2 writes a matrix into a file with the
   name obtained from getfname using the seed tied
*/

char *putmat2(Mat_DP &a, char *tied)
{
    FILE *fp;
    int i, j, m=a.nrows(), n=a.ncols();
    char *fname;
    fname=getfname(tied);
    if ((fp=fopen(fname, "w"))==NULL){
        fprintf(stderr, "Cannot open file %s\n", fname);
        exit(1);
    }
    fprintf(fp, "%d %d\n", m, n);
    for (i=0; i<m; i++){
        for (j=0; j<n; j++) fprintf(fp, "%25.16e ", a[i][j]);
        fprintf(fp, "\n");
    }
    fclose(fp);
    return fname;
}

```

```

/* getmat reads a matrix file stored in this format:
 2 3
 1.1 2.2 -3.3
-4.4 5.5 -6.6
Prior to the call of getmat, Mat_DP a(1,1);
must be defined in the calling program
on return a has the dimensions read from the file.
*/

Mat_DP getmat(char *fname)
{
    int m,n;
    DP d;
    ifstream fp(fname);
    if (fp.fail())
    {
        cout << "File " << fname << " could not be opened.\n" << endl;
        abort();
    }
    fp >> m; fp >> n;
    Mat_DP a(m,n);
    for (int i=0;i<m;i++)
        for (int j=0;j<n;j++)
        {
            fp >> d;
            a[i][j]=d;
        }
    fp.close();
    return a;
}

/*
matmul multiplies two matrices a and b and stores the result in c.
*/
void matmul(Mat_DP &a, Mat_DP &b, Mat_IO_DP &c)
{
    int ma=a.nrows(),na=a.ncols(),mb=b.nrows(),nb=b.ncols();
    if (na !=mb)
    {
        cout<<"Incompatible matrix dimensions: "<<
            " na, mb= "<<na<<" "<<mb<<endl;
        abort();
    }
}

```



```

    DP s;
    for (int i=0;i<ma;i++)
        for (int j=0;j<nb;j++)
            {
                s=0.0;
                for (int k=0;k<na;k++)
                    s+=a[i][k]*b[k][j];
                c[i][j]=s;
            }
    }

/*
    matsum adds two matrices a and b and stores the result in c.
*/
void matsum(Mat_DP &a, Mat_DP &b, Mat_IO_DP &c)
{
    int ma=a.nrows(), na=a.ncols(), mb=b.nrows(), nb=b.ncols();
    if ((ma !=mb) ||( na !=nb))
        {
            cout<<"Incompatible matrix dimensions: "
                <<" ma, na= "<<ma<<" "<<na
                <<" mb, nb= "<<mb<<" "<<nb <<endl;
            abort();
        }
    for (int i=0;i<ma;i++)
        for (int j=0;j<na;j++)
            {
                c[i][j]=a[i][j]+b[i][j];
            }
    }

/*
    scalmatmul multiplies a matrix a by s and stores result in b.
*/
void scalmatmul(Mat_DP &a, DP s,Mat_DP &b)
{
    int ma=a.nrows(), na=a.ncols();
    for (int i=0;i<ma;i++)
        for (int j=0;j<na;j++)
            {
                b[i][j]=s*a[i][j];
            }
    }
}

```

```

/*
    scalmatsum adds a scalar to a matrix a by s
    and stores result in b.
*/
void scalmatsum(Mat_DP &a, DP s, Mat_DP &b)
{
    int ma=a.nrows(), na=a.ncols();
    for (int i=0;i<ma;i++)
        for (int j=0;j<na;j++)
            {
                b[i][j]=s+a[i][j];
            }
}

/*
    matvecmul multiplies a matrix a by a vector v
    and stores result in b.
*/
void matvecmul(Mat_DP &a, Vec_DP &v, Vec_DP &b)
{
    int ma=a.nrows(), na=a.ncols(), nv=v.size(), nb=b.size();
    DP s=0.0;
    if ((na !=nv) ||(ma != nb))
        {
            cout<<"Incompatible factors, ma,na,nv, nb ="
                <<ma<<" ", <<na<<" ", <<nv<<nb<<" ", <<nb<<endl;
            abort();
        }
    for (int i=0;i<ma;i++)
        {
            s=0.0;
            for (int j=0;j<na;j++) s+=a[i][j]*v[j];
            b[i]=s;
        }
}

/*
    vecmatmul multiplies a vector by a matrix
    and stores result in b.
*/
void vecmatmul(Vec_DP &v, Mat_DP &a, Vec_DP &b)
{
    int ma=a.nrows(), na=a.ncols(), nv=v.size(), nb=b.size();
    DP s=0.0;

```

```

if ((na !=nb) ||(ma != nv))
{
    cout<<"Incompatible factors in vecmatmul, ma,na,nv, nb ="
        <<ma<<"<<na<<"<<nv<<nb<<"<<nb<<endl;
    abort();
}
for (int i=0;i<b.size();i++)
{
    s=0.0;
    for (int j=0;j<na;j++) s+=a[j][i]*v[j];
    b[i]=s;
}
}

/*
mnormp returns matrix norm:
(Sum_{i,j} |a(i,j)|^{p})^{1/p}
*/
DP mnormp(Mat_DP &a, DP p)
{
    DP s=0.0;
    int m= a.nrows(), n= a.ncols();
    for (int i=0;i<m;i++){
        for (int j=0;j<n;j++){
            s=s+pow(fabs(a[i][j]),p);}}
    return pow( s, 1.0/p) ;
}

/* vnormp returns vector norm:
(Sum_{i} |a(i)|^{p})^{1/p} */
DP vnormp(Vec_DP &a, DP p)
{ DP s=0.0;
    int m= a.size();
    for (int i=0;i<m;i++)
        s=s+pow(fabs(a[i]),p);
    return pow( s, 1.0/p) ;
}

/*
ones returns matrix with all elements = 1,
of given size
*/
void ones(Mat_DP &a)
{

```

```

int m=a.nrows(), n=a.ncols();
for (int i = 0; i < m; i++)
    {
        for (int j = 0; j < n; j++)
            {
                a[i][j] = 1.0;
            }
    }
}

/*
unitmat returns a matrix with elements =1 on diagonal,
0 elsewhere, of given size
*/
void unitmat(Mat_DP &a)
{
    int m=a.nrows(), n=a.ncols();
    for (int i = 0; i < m; i++)
        {
            for (int j = 0; j < n; j++)
                {
                    a[i][j] = 0.0; if (i==j) a[i][j] = 1.0;
                }
        }
}

/* Transpose of a matrix */
void transp(Mat_DP &a, Mat_DP &aT)
{
    int m=a.nrows(), n=a.ncols();
    for (int i = 0; i < m; i++)
        for (int j = 0; j < n; j++)
            aT[j][i] = a[i][j];
}

/* Determinant of a matrix */
DP detmat(Mat_DP &a)
{
    int m=a.nrows(), n=a.nrows();
    if ((m!=n)|| (m<=1))
        {cout<<"Matrix/vector dimension error in detmat. "
          <<endl;
         abort();
        }
    n=m;
}

```

```

    Mat_DP aa=a;
    Vec_INT indx(n);
    DP t;
    NR::ludcmp(aa,indx, t);
    for (int j=0;j<n;j++){ t = t*aa[j][j];}
    return t;
}

DP residual(Mat_DP &a, Vec_DP &x, Vec_DP &b)
{
    int ma=a.nrows(),na=a.ncols();
    if ((na!=x.size())||(ma!=b.size()))
        {cout<<"Argument error in residual"<<endl;
        abort();}
    Vec_DP c(ma);
    matvecmul(a,x,c);
    for (int i=0;i<c.size();i++) c[i]=c[i]-b[i];
    return vnormp(c,2.0);
}

/* Overloading sum operator for matrices */
inline Mat_DP operator+(Mat_DP &a, Mat_DP &b)
{
    Mat_DP c(a.nrows(),a.ncols());
    matsum(a,b,c);
    return c;
}

/* Overloading product operator for matrices */
inline Mat_DP operator*(Mat_DP &a, Mat_DP &b)
{
    Mat_DP c(a.nrows(),b.ncols());
    matmul(a,b,c);
    return c;
}

/* Overloading product operator for matrix times scalar */
inline Mat_DP operator*(Mat_DP &a,DP x)
{
    Mat_DP c(a.nrows(),a.ncols());
    scalmatmul(a,x,c);
    return c;
}

```

```

/* Overloading product operator for scalar times matrix */
inline Mat_DP operator*(DP x,Mat_DP &a)
{
    Mat_DP c(a.nrows(),a.ncols());
    scalmatmul(a,x,c);
    return c;
}

/* Overloading minus operator for matrices */
inline Mat_DP operator-(Mat_DP &a, Mat_DP &b)
{
    Mat_DP c(a.nrows(),a.ncols());
    Mat_DP bb=(-1.0)*b;
    matsum(a,bb,c);
    return c;
}

/* Overloading plus operator for matrix plus scalar */
inline Mat_DP operator+(Mat_DP &a,DP x)
{
    Mat_DP c(a.nrows(),a.ncols());
    scalmatsum(a,x,c);
    return c;
}

/* Overloading minus operator for matrix minus scalar */
inline Mat_DP operator-(Mat_DP &a,DP x)
{
    Mat_DP c(a.nrows(),a.ncols());
    scalmatsum(a,-x,c);
    return c;
}

/* Overloading plus operator for scalar plus matrix */
inline Mat_DP operator+(DP x,Mat_DP &a)
{
    Mat_DP c(a.nrows(),a.ncols());
    scalmatsum(a,x,c);
    return c;
}

/* */
/* Overloading ! operator to denote matrix transpose */
inline Mat_DP operator!(Mat_DP &a)
{
    Mat_DP b(a.ncols(),a.nrows());
    transp(a,b);
    return b;
}

```

```

}
/* Overloading ^ operator for matrix power */
inline Mat_DP operator^(Mat_DP &a, unsigned int b)
{
    int m=a.nrows(), n=a.ncols();
    if(m!=n)
    {
        cerr<<"Incompatible matrix dimensions.\n";
        abort();
    }
    Mat_DP d(m,n);
    if(b>0) d=a;
    for(int i=2;(unsigned int)i<=b;i++)
        d=d*d;
    return d;
}
/* Overloading << operator for matrix output */
ostream &operator<<(ostream &stream, const Mat_DP &a)
{
    int m=a.nrows(), n=a.ncols();
    for (int j=0;j<m;j++)
        for (int k=0;k<n;k++)
            {
                stream << fixed<<scientific<<setw(14);
                stream << setprecision(8)<<(a[j][k]) << " ";
                if (k==n-1)
                    stream<<endl;
            }
    return stream;
}

/* Overloading << operator for vector output */
ostream &operator<<(ostream &stream, const Vec_DP &a)
{
    stream << fixed<<scientific<<setw(14);
    for (int j=0;j<a.size();j++)
        {
            stream << setprecision(8)<<(a[j]) << " ";
        }
    stream<<endl;
    return stream;
}

/* cond gives the condition number of a square matrix */

```

```

DP cond(Mat_DP &a)
{
    int ma=a.nrows(), na=a.ncols();
    if ((ma!=na))
        {cout<<"Matrix/vector dimension error in cond. "
          <<endl;
         abort();
        }
    Vec_DP w(na);
    Mat_DP u(ma,na),v(na,na);
    u=a;
    NR::svdcmp(u,w,v);
    // find maximum singular value
    DP wmax=w[0], wmin=w[0];
    int n=ma;
    for (int k=0;k<n;k++)
        { if (w[k] > wmax) wmax=w[k];
          if (w[k] < wmin) wmin=w[k];}
    return wmax/(wmin+1e-100);
}
/* invmat returns matrix inverse of a square matrix */
void invmat(Mat_DP &a, Mat_DP &inva)
{
    int ma=a.nrows(), na=a.ncols(), mi=inva.nrows(),
        ni=inva.ncols() ;
    if ((ma!=na)|| (mi !=ni) || (ma!=mi))
        {cout<<"Matrix/vector dimension error in invmat. "
          <<endl;
         abort();
        }
    int n=ma;
    Mat_DP ai(n,n), sol(n,1);
    sol=1.0;
    ai=a;
    NR::gaussj(ai,sol);
    inva=ai;
}

/* FILE matutl02.cpp begins */

```



```

/* FILE: myutl4.cpp begins
// g++ myutl4.cpp -I../utils -L../lib -o a -lm -lnr
/* USAGE: ./a <myutl4.inp > a.jnk where myutl4.inp is
2 3
1.2 3.4 -2.1 6.4 0.2 3.12
3 2
-2.1 6.4 0.2 3.12 0.03 -0.12
*/
*/
The purpose of this program is to demonstrate the
matrix and vector classes and their properties, as
defined in the file matutl02.cpp.
Some operations, such as matrix product can be
computed in two ways, with matmul or with
the overloaded * operator.
*/

#include <stdlib.h>
#include <cstdlib>
#include <cstdio>
/*#include <ctime>
#include <string>
#include <fstream>
#include <iostream>
#include <iomanip> */
using namespace std;
#include "nr.h"
#include "matutl02.h"

int main(void)
{
    int j,k,l,m,n;
    const char *fname="myutl1.dat";
    init_srand(); // Initialize the random number generator.
    Mat_DP a(1,1);
    a=entermat2(2,4); // The size of a will change!!
    cout<<a;
    Mat_DP d(4,2),e(2,4),f(4,4);

```

```

ranmat2(d, -10.0,10.0);
ranmat2(e, -10.0,10.0);
cout<<"Random matrix: "<<endl;
showmat2(d," %8.4lf%");
showmat2(e," %8.4lf%");
f=d*e;
showmat2(f," %10.6e");
fname="myutl1.dat";
fname=putmat2(f,fname);
cout<<"Matrix f saved in file "<< fname <<endl;
Mat_DP g(1,1), h(1,1);
g=getmat(fname);
h=g; // Tehdaan h ja g samankokoisiksi
cout<<endl<<"Matrix g read from the file: "<<fname;
showmat(g);
h=f-g;
cout<< "Difference h=f-g of original matrix \n"<<
      "and the one read from the file:" <<
      " \n"<<endl;
showmat2(h," %8.4e");
printf("norm_2(h)= %10.4e\n ", mnormp(h,2));
Vec_DP dd(4),vv(4) ;
dd[0]=0.2313; dd[1]=-0.342342; dd[3]=2.23132;
matvecmul(f,dd,vv);
showvec(vv);
cout
<<"\nTest of accuracy in 10x10 matrix inversion: \n"<<endl;
Mat_DP a1(10,10), b(10,10), ai(10,10), u(10,10);
unitmat(u);
ranmat2(a1,-1,1);
invmat(a1,ai);
b=a1*ai;
a1=b-u;
printf("norm(a*inv(a)-id)= %10.4e\n ",mnormp(a1,2));
return 0;
}
/* FILE: myutl4.cpp ends */

```

Ohjelman myutl4.cpp kääntäminen tapahtuu tavalliseen tapaan kommenteista ilmenevällä tavalla. Ohjelma voidaan ajaa komentamalla ./a, jolloin syötteet annetaan näppäimistöltä ja tulostus tapahtuu kuvaruudulle. Toinen mahdollisuus on ohjata syöte ja tulostus tiedostojen kautta tapahtuvaksi seuraavasti:

```
./a <myutl4.inp >myutl4.dat
```

```
Tiedosto myutl4.inp:
```

```
2 3
1.435 -0.97879 2.00564 -4.55343 3.0004545 0.0678
3 2
1.756 -2.8888 0.3243 -0.11 2.33 6.33
```

```
Tiedosto myutl4.dat:
```

```
Enter matrix size (m,n):Please enter integer in [2,6]:
```

```
Please enter integer in [2,6]:
```

```
Enter A[0,0]: The input you gave: 1.2
```

```
Enter A[0,1]: The input you gave: 3.4
```

```
Enter A[0,2]: The input you gave: -2.1
```

```
Enter A[1,0]: The input you gave: 6.4
```

```
Enter A[1,1]: The input you gave: 0.2
```

```
Enter A[1,2]: The input you gave: 3.12
```

```
1.2 3.4 -2.1
```

```
6.4 0.2 3.12
```

```
Random matrix:
```

```
4x2 matrix:
```

```
-5.5544 9.6048
```

```
9.1445 8.3267
```

```
8.3998 5.0998
```

```
-9.4150 7.6505
```

```
2x4 matrix:
```

```
3.6151 1.9018 7.8569 9.0707
```

```
-8.0862 -3.9453 3.9931 -5.6617
```

```
4x4 matrix:
```

```
-9.774543e+01 -4.845769e+01 -5.287328e+00 -1.047618e+02
```

```
-3.427241e+01 -1.545997e+01 1.050967e+02 3.580391e+01
```

```
-1.087161e+01 -4.145204e+00 8.636056e+01 4.731865e+01
```

```
-9.589913e+01 -4.808961e+01 -4.342255e+01 -1.287150e+02
```

```
Matrix f saved in file myutl1.dat
```

```
Matrix g read from the file: myutl1.dat
```

```
4x4 matrix:
```

```
-97.745426 -48.457686 -5.2873278 -104.76183
```

```
-34.272411 -15.459971 105.09673 35.803906
```

```
-10.871607 -4.1452043 86.360556 47.318648
```

```
-95.899126 -48.089609 -43.422552 -128.71496
```

Difference f-g of original matrix  
and the one read from the file:

4x4 matrix:

```
0.0000e+00 0.0000e+00 0.0000e+00 0.0000e+00
0.0000e+00 0.0000e+00 0.0000e+00 0.0000e+00
0.0000e+00 0.0000e+00 0.0000e+00 0.0000e+00
0.0000e+00 0.0000e+00 0.0000e+00 0.0000e+00
norm_2(h)= 0.0000e+00
```

A vector with 4 components:

```
-239.77659 77.25536 104.48752 -292.92264
```

Test of accuracy in 10x10 matrix inversion:

```
norm(a*inv(a)-id)= 1.7708e-15
```

**1.26. Numeerisen lineaarialgebran perusoperaatiot.** Kuten yo. ohjelmasta myut14.cpp ja sen tulosteesta ilmenee, monet matriisialgebran operaatiot voidaan C++ -koodissa esittää lähes samoin merkinnöin kuin matematiikassa. Esimerkkejä ovat matriisien summa, tulo, vakiolla kertominen, ja matriisinormi sekä neliömatriiseille käänteismatriisi ja determinantti. Näiden ohella tarvitsemme jatkossa mm. sarakkeiden tai rivien vaihtoa sekä matriisien muuntamista erityisiin muotoihin tarkoituksena sovellukset lineaarialgebran ongelmiin erityisesti lin. yht.ryhmän ratkaisuun ja ominaisarvotehtäviin. Lineaariset yhtälöryhmät, joissa kerroinmatriisilla on erityinen muoto, voidaan usein ratkaista helpommin tai tehokkaammin kuin yleiset lin. yhtälöryhmät. On tarpeen tarkastella niitä esivalmisteluna yleiseen tapaukseen.

**1.27. Erityistyyppiset matriisit.** Tärkeitä matriisityyppejä numeerisessa laskennassa ovat:

- yläkolmio-, alakolmio-, lohkomatriisit. Yläkolmiomatriisissa  $a_{i,j} = 0$  jos  $i > j$ .
- nauhamatriisi,  $p$ -diagonaalimatriisi ( $a_{i,j} = 0$ , kun  $|i - j| \geq p$ )

- symmetrinen ( $A^T = A$ ), ortogonaalinen matriisi ( $A^T = A^{-1}$ )
- riveittäin lävistäjävoittoinen (eli diagonaalidominoiva) (ts.  $|a_{i,i}| > \sum_{k=1, k \neq i}^n |a_{i,k}|$  kaikilla  $i = 1, 2, \dots, n$ )
- permutointimatriisi saadaan vaihtamalla identtisessä matriisissa sarakkeet  $k$  ja  $h$ . Se on ortogonaalinen.
- neliömatriisi  $A$  on positiivisesti definiitti (semidefiniitti) jos  $x^T Ax > 0$  (vast.  $x^T Ax \geq 0$ ) kaikilla  $x \in R^n \setminus \{0\}$ .

**1.28. Lause (Choleskyn hajoitelma)**  $n \times n$  matriisi  $A$  on positiivisesti definiitti jos ja vain jos on olemassa säännöllinen  $n \times n$  alakolmio-matriisi  $L$  positiivisin diagonaali-alkioin se.  $A = LL^T$ .

**1.29. Kokeelliset virhearviot.** Keskeinen osa ohjelmointityössä on ohjelman toiminnan oikeellisuuden varmistus. Matriisilaskennassa testauksen osana voidaan tuottaa virhearvioita kokeellisesti satunnaismatriisien avulla. Tarkastamme kahta esimerkkiä.

Olkoon  $A$   $m \times n$  matriisi. Oletamme, että meillä on  $m \times m$  matriisi  $Q$  ja  $m \times n$  matriisi  $R$ , joilla pitäisi olla voimassa esityksen  $A = QR$ . Joukolle satunnaismatriiseja  $A$  voimme tuottaa ko. esityksen ja muodostaa erotuksen  $\|A - QR\|$  ja käyttää suurinta erotuksen arvoa virhearviona.

Oletamme, että meillä on ohjelma `LINsolve(a,b,x)`, jonka pitäisi ratkaista lin. yhtälöryhmä  $Ax = b$  missä  $A$  on  $n \times n$  matriisi. Muodostamme satunnaismatriisille  $A$  vektorin  $z$  jonka kaikki komponentit ovat tunnettuja (esim. = 1) ja  $b = Az$ . Olkoon  $x$  `LINsolve`:n tuottama numeerinen ratkaisu. Merkitään  $r_1 = \|z - x\|$  ja  $r_2 = \|Ax - b\|$ . Silloin  $r_1$  on numeerisen ratkaisun virhe ja  $r_2$  ns. residuaali. Molempia voidaan käyttää virheen arviointiin, jos ne tunnetaan. Huomaa, että  $r_2$  voidaan laskea aina, kun taas  $r_1$  pelkästään siinä tapauksessa, että tarkka ratkaisu tiedetään etukäteen.

Molempien esimerkkien tapauksessa virhe on pieni. Kuten luvussa 2 tullaan näkemään, lin. yhtälöiden ratkaisuisa virheen suuruusluokka on useasti  $10^{-14}$  tai parempi.

**1.30. Matriisin kuntoisuusluku.** Numeerisen matriisialgebran tär-

keimpiä käsitteitä on  $n \times n$  matriisin ns. *kuntoisuusluku* (engl. condition number), joka kuvastaa matriisin numeerisia ominaisuuksia. Jos  $A$  on neliömatriisi, niin  $\text{cond}(A) \geq 1$  ja numeeriset ominaisuudet huononevat  $\text{cond}(A)$ :n kasvaessa. Kuntoisuuslukua voitaisiin kutsua myös herkkyysparametriksi tai häiriöalttiudeksi, sillä kuten tulemme näkemään jatkossa, se kuvastaa virheen voimistumista lineaarikuvauksessa. Jatkossa tulemme esittelemään lyhyesti  $\text{cond}(A)$ :n käyttöä. Huomaa, että  $\text{cond}(A)$  oli jo osa `matut102.cpp` pakettia.

```

/* FILE: mycond.cpp begins.                                     */
/* g++ mycond.cpp -L../lib -I../utils -o a -lm -lnr          */

#include <stdlib.h>
#include <cstdlib>
#include <cstdio>

using namespace std;
#include "nr.h"
#include "matut102.h"

void ShowUWV(Mat_DP &a)
{
    int ma=a.nrows(),na=a.ncols();
    Mat_DP u(ma,na), v(na,na),w1(na,na);
    Vec_DP w(na);
    u=a;
    w1=0.0;
    cout<<"A= "<<endl;      showmat2(a,"% 10.5f ");
    NR::svdcmp(u,w,v);
    for (int i=0;i<na;i++) w1[i][i]=w[i];
    cout<<"u= "<<endl;      showmat2(u, "%10.5lf ");
    cout<<"w^T= "<<w <<endl;
    cout<<"v= "<<endl;
    showmat2(v, "%10.5lf ");
    /* Estimate SVD decomposition error: */
    Mat_DP tmp1(ma,na), tmp2(na,na), tmp3(ma,na);
    matmul(u,w1,tmp1);          transp(v,tmp2);
    matmul(tmp1,tmp2,tmp3 );    tmp1=a-tmp3;
    printf("norm(A- u *w * v^T) = %10.4e\n",mnormp(tmp1,2.0));
}
int main()
{

```

```

int ma=3,na=3,mb=5,nb=5;
Mat_DP a(ma,na), b(mb,nb);
double con1, con2;
ranmat2(a,0.1, 1.5);
ShowUWV(a);
con1=cond(a);
printf("cond(A) = %16.4e \n",con1);
unitmat(b); b[0][0]=17.0;
cout <<"B= " <<endl; showmat2(b,"% 8.5g ");
con2=cond(b);
printf("cond(B) = %16.4e \n",con2);
return 0;
}
/* FILE: mycond.cpp ends. */

```

### Testiajon tuottama tulostus:

A=

3x3 matrix:

1.27626	0.65214	1.19634
1.21782	1.37631	0.37657
0.56931	1.17552	0.48888

u=

3x3 matrix:

-0.61134	-0.04273	-0.79021
-0.64003	0.61397	0.46196
-0.46543	-0.78817	0.40269

w<sup>T</sup>= 2.8470869 0.33740483 0.85421621

v=

3x3 matrix:

-0.64088	0.72452	-0.25366
-0.64159	-0.32414	0.69519
-0.42146	-0.60828	-0.67258

norm(A- u \*w \* v<sup>T</sup>) = 8.1016e-16

cond(A) = 8.4382e+00

B=

5x5 matrix:

17	0	0	0	0
0	1	0	0	0
0	0	1	0	0
0	0	0	1	0

```

0      0      0      0      1
cond(B) = 1.7000e+01

```

**1.31. Kolmiomatriisit.** Lin.  $n \times n$  yhtälöryhmän  $Ax = b$  ratkaisu helpottuu oleellisesti, jos  $A$  on joko yläkolmiomatriisi  $U$  tai alakolmiomatriisi  $L$ . Ensimmäisessä tapauksessa voimme viimeisestä yhtälöstä ratkaista  $x_n:n$ , sijoittaa sen toiseksi viimeiseen ja ratkaista  $x_{n-1} :n$ , “peruuttaen” kunnes kaikki komponentit on ratkaistu. Toisessa tapauksessa voimme taas edetä päinvastaisessa järjestyksessä.

Jos  $n \times n$  matriisilla  $A$  on hajoitelma  $A = LU$  missä  $L$  ja  $U$  vastaavasti ovat säännöllisiä ala- ja yläkolmiomatriiseja, niin yhtälöryhmän  $Ax = y$  ratkaiseminen palautuu seuraaviksi kahdeksi yhtälöryhmäksi

$$Lz = y, \quad Ux = z.$$

Molemmat voidaan ratkaista edelläkuvatulla menettelyllä.

**1.32. Choleskyn hajoitelma.** Edellä kuvattua hajoitelmaa  $n \times n$  matriisille ei aina ole olemassa. Eräissä erikoistapauksissa hajoitelma voidaan muodostaa helposti. Choleskyn lauseen nojalla näin on erityisesti silloin kun  $A$  on positiivisesti definiitti ja symmetrinen.

Allaoleva algoritmi kirjoittaa  $A:n$  päälle matriisin  $L$ .

---

**Algorithm (Cholesky factorization)**

---

```

for  $k = 1$  to  $n$                                 { loop over columns }
   $a_{kk} = \sqrt{a_{kk}}$ 
  for  $i = k + 1$  to  $n$ 
     $a_{ik} = a_{ik} / a_{kk}$                         { scale current column }
  end
  for  $j = k + 1$  to  $n$                                 { from each remaining column,
    for  $i = k + 1$  to  $n$                                 subtract multiple
       $a_{ij} = a_{ij} - a_{ik} \cdot a_{jk}$                 of current column }
    end
  end
end
end

```

---



Kohdan 1.31 mukaan Choleskyn hajoittelun avulla voimme ratkaista yhtälöryhmän  $Ax = y$  positiivisesti definitille matriisille  $A$ .

Yo. muodossa algoritmi on implementoitu ohjelmaan `mychol3.cpp` joka löytyy kurssin `www`-sivulta. Ks. myös `NR::cholsl`, `NR::choldc`.

**1.33. Householderin QR hajoitelma.** Numeerisen lineaarialgebran tärkeimpiä algoritmeja on  $m \times n$ ,  $m \geq n$ , matriisin  $A$  ns. QR hajoitelma joka antaa esityksen  $A = QR$  missä  $Q$  on  $m \times m$  ortogonaalinen matriisi ja  $R$   $m \times n$  matriisi jolla diagonaalien alapuoliset alkiot ovat nollia.

Hajoitelmaan päästään tekemällä peräkkäisiä ortogonaalimuunnoksia  $n$  kappaletta. Muunnos  $H_j$ ,  $j = 1, \dots, n$  nollaa  $A$ :n diagonaalien alapuoleiset komponentit sarakkeessa  $j$  kuitenkin siten, että k. sarakkevektorin pituus säilyy muunnoksessa. Kukin tällainen ortogonaalimuunnos voidaan tulkita peilaukseksi, kuten alla esitetään.

Householderin muunnos on  $m \times m$  matriisi, jolla on muoto

$$H = I - 2 \frac{vv^T}{v^T v}$$

missä  $v \neq 0$ . Silloin  $H = H^T = H^{-1}$ , joten  $H$  on symmetrinen ja ortogonaalinen. Haluamme valita annetulle vektorille  $a$  vektorin  $v$ , jolle kaikki muut paitsi ensimmäinen komponentti  $Ha$ :sta on nolla, siis  $Ha = \alpha e_1$  jollakin  $\alpha$ . Silloin

$$\begin{aligned} \alpha e_1 = Ha &= \left( I - 2 \frac{vv^T}{v^T v} \right) a = a - 2v \frac{v^T a}{v^T v}, \\ \Rightarrow v &= (a - \alpha e_1) \frac{v^T v}{2v^T a}. \end{aligned}$$

Skalaaritekijäksi valitaan 1 jolloin  $v = a - \alpha e_1$ . Koska  $H$  säilyttää normin pitää valita  $\alpha = \pm |a|$ . Merkitsevien numeroiden kumoutumisen ehkäisemiseksi valitaan  $\alpha = \text{sign}(a_1)|a|$ .

Tavallinen sovellustilanne on pns-yhtälöiden ratkaiseminen, joka johtaa yhtälöryhmään  $Ax = b$  missä  $A$  on  $m \times n$ ,  $m \geq n$ . Yhtälöryhmä säilyy samana kun se kerrotaan  $H_1$ :lla:  $H_1 Ax = H_1 b$ . Jatkaen näin saadaan se lopulta muotoon

$$H_n \cdots H_1 Ax = Q^T b; \quad Q^T = H_n \cdots H_1.$$

josta  $x$  voidaan helposti ratkaista koska  $Q^T A$ :n diagonaalien alapuoleiset alkioit ovat nollia. Ratkaisuun ei tarvita  $Q$ :ta.

Allaoleva algoritmi tuottaa  $A$  matriisista  $m \times n$ ,  $m \geq n$ , peräkkäisillä ortogonaalimuunnoksilla matriisin  $R$ . (Matriisia  $Q$  ei konstruoida.)

Algoritmi on implementoitu ohjelmaan `myhouse2.cpp`, ja siihen palataan myöhemmin.

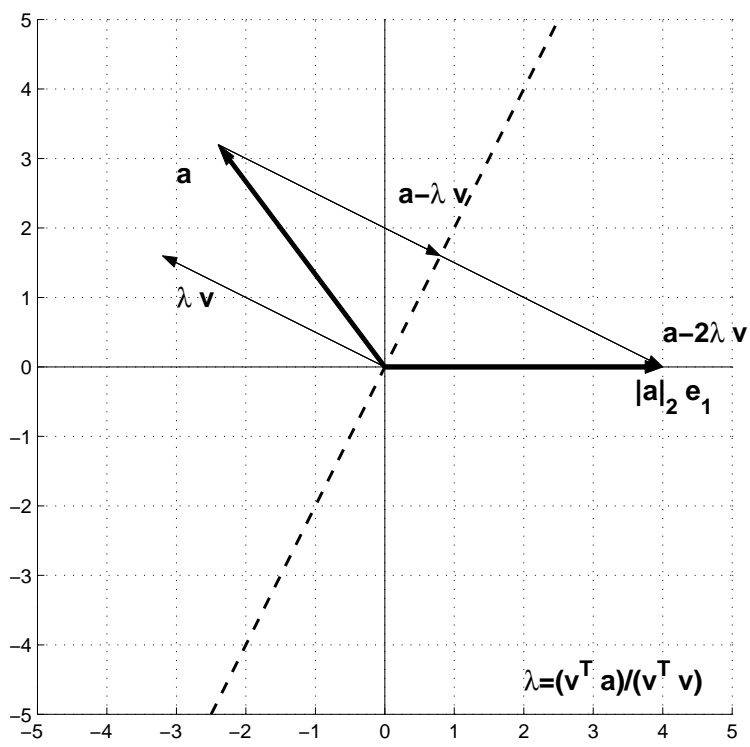
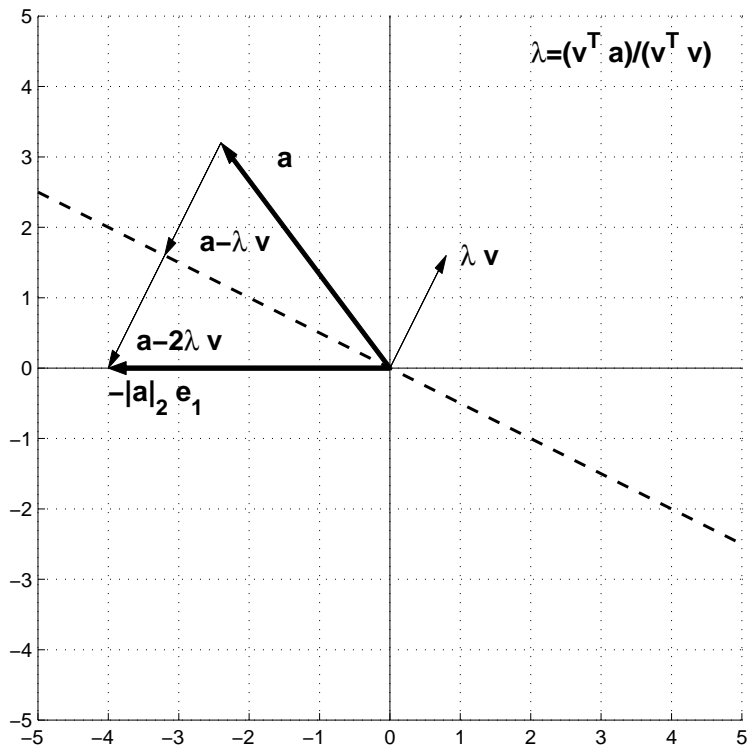
---

### Algorithm (QR Factorization)

---

<b>for</b> $k = 1$ <b>to</b> $n$	{loop over columns }
$\alpha_k = -\text{sign}(a_{kk})\sqrt{a_{kk}^2 + \dots + a_{mk}^2}$	{compute Householder
$\mathbf{v}_k = [0 \dots 0 \ a_{kk} \dots a_{mk}]^T - \alpha_k \mathbf{e}_k$	vector for current
	col }
$\beta_k = \mathbf{v}_k^T \mathbf{v}_k$	
<b>if</b> $\beta_k = 0$ <b>then</b>	{skip current column
continue with next $k$	if it's already zero }
<b>for</b> $j = k$ <b>to</b> $n$	{apply transformation
$\gamma_j = \mathbf{v}_k^T \mathbf{a}_j$	to remaining
$\mathbf{a}_j = \mathbf{a}_j - (2\gamma_j/\beta_k)\mathbf{v}_k$	submatrix }
<b>end</b>	
<b>end</b>	

---



**1.34. Iteratiiviset algoritmit.** Tavanomaisilla lineaarisen yhtälöryhmän ratkaisumenetelmillä on yhteinen piirre: ne perustuvat lineaarialgebran tuloksiin ja tarvittavien laskutoimitusten määrä, eli laskennan vaativuus, on luokkaa  $n^3$ , jos yht.ryhmän koko on  $n \times n$ . Laajojen yhtälöryhmien tapauksessa voi kuitenkin olla parempaa käyttää ns. *iteratiivisia menetelmiä*. Perusajatuksena on kirjoittaa  $n \times n$  yhtälöryhmä toisin:

$$Ax = b \Leftrightarrow x = Tx + c,$$

ja soveltaa iteraatiota  $x^{(k+1)} = Tx^{(k)} + c$ , joka yleistää yhden muuttujan funktion tapauksessa esiteltyä kiintopisteiteraatiota. Hajotelma löytyy seuraavasti. Kirjoitetaan  $A = L + D + U$  (lower, diagonal, upper; tässä diagonaali-alkiot mukana pelkästään  $D$ :ssä), jolloin

$$Ax = b \Leftrightarrow Dx = -(L + U)x + b.$$

Siis  $T = D^{-1}(-(L + U))$  ja  $c = D^{-1}b$ . Komponenttimuodossa saadaan esitys

$$x_j^{(k+1)} = (b_j - (L + U)x_j^{(k)})/a_{jj}, j = 1, 2, \dots, n,$$

tai yhtäpitävästi

$$(J) \quad x_j^{(k+1)} = (b_j - \sum_{i=1, i \neq j}^n a_{ij}x_i^{(k)})/a_{jj},$$

missä yläindeksi vastaa iteraation numeroa. Iteraatio on nimeltään *Jacobin iteraatio*. Tämä iteraatio suppenee kaikilla alkuarvoilla vain tietyin matriisia  $A$  koskevin oletuksin (vrt. seur. lause). Laskennan vaativuus riippuu nyt halutusta tarkkuudesta ja kaavan (J) vaativuudesta. Huomaa: jos matriisi  $A$  on harva (nollasta eroavien termien määrä  $O(n)$ ), voidaan kaavan (J) summaa tehostaa rajoittamalla nollasta eroaviin termeihin.

Jacobin menetelmä on lähtökohtana erälle muille iteratiivisille menetelmille. Ns. *Gaussin-Seidelin iteraatio* perustuu seuraavaan huomioon. Jos kaavassa (J) laskenta tapahtuu indeksin  $j$  mukaan kasvavassa järjestyksessä ja on jo laskettu arvoilla  $j = 1, \dots, j_0$ ,

niin vaiheessa  $j_0 + 1$  voitaisiin termien  $x_j^{(k)}$ ,  $j = 1, \dots, j_0$ , asemasta käyttää termejä  $x_j^{(k+1)}$ ,  $j = 1, \dots, j_0$ . Toisin sanoen, Gaussin-Seidelin menetelmässä käytetään kaavaa:

$$(GS) \quad x_j^{(k+1)} = (b_j - \sum_{i=1, i < j}^n a_{ij} x_i^{(k+1)} - \sum_{i=1, i > j}^n a_{ij} x_i^{(k)}) / a_{jj}.$$

Mainitsemme vielä kolmannenkin iteratiivisen menetelmän ns. *SOR-iteraation* (successive overrelaxation). Siinä kaava (GS) korvataan seuraavalla:

$$(SOR) \quad x_j^{(k+1)} = (1 - \omega)x_j^{(k)} + \omega(b_j - \sum_{i=1, i < j}^n a_{ij} x_i^{(k+1)} - \sum_{i=1, i > j}^n a_{ij} x_i^{(k)}) / a_{jj}.$$

SOR-menetelmä suppenee sopivilla parametrin  $\omega \in (0, 2)$  arvoilla nopeammin kuin Gaussin-Seidelin menetelmä.

Riittäviä ehtoja iteraatioiden (J) ja (GS) suppenemiselle on esitetty mm. diagonaalipainotteisille matriiseille. Näille matriiseille on aina olemassa myös kääntematriisi.

**1.35. Lause.** [Stoer-Bulirsch, Thm 8.2.6, 8.2.12] Jacobin ja Gaussin-Seidelin iteroinnit suppenevat kaikille matriiseille  $A$ , joille pätee riittävä diagonaalinen dominointiehto

$$|a_{ii}| > \sum_{k=1, k \neq i}^n |a_{ik}|, i = 1, 2, \dots, n.$$

Seuraava ohjelma implementoi sekä Jacobin että Gaussin-Seidelin iteraatiot lineaarisen  $n \times n$  yhtälöryhmän ratkaisemiseksi. Generoimme ensin jonkin satunnaismatriisin  $A$  ja kasvatamme sen diagonaalialkioita, niin että yo. suppenemislauseen ehto tulee voimaan. Iteroinnin alkuarvona on origo. Lopuksi vertaamme löydettyä ratkaisua alunperin jo tiedossamme olevaan juureen saadaksemme arvion virheestä. Esilläolevassa tapauksessa molemmat iteraatiot johtavat samaan juureen, ja huomaamme, että Gaussin-Seidelin iteraatiossa suppeneminen on hieman nopeampaa.

```

/* FILE: myjgsit.cpp begins.                                     */
/* g++ myjgsit.cpp -L../lib -I../utils -o a -lm -lnr          */
/* Jacobi/Gauss-Seidel iterations to solve                    */
/* a linear system */

#include <cstdlib> // Used in putmat2
#include <cstdio>  // Used in putmat2
#include <ctime>
#include <string>
#include <fstream>
#include <iostream>
#include <iomanip>
using namespace std;
#include "nr.h"
#include "matut102.h"

void Jacobi_iter(Mat_DP &a, Vec_DP &b,
Vec_DP &x, Vec_DP &xold)
{
    DP s;
    int i,j,ma=a.nrows();
    if ((ma!=a.ncols())||(ma!=b.size())||
        (ma!=x.size())||(ma!=xold.size()))
        {cout<<"Argument error in Jacobi_iter\n"<<endl;
        abort();
        }
    for (i=0;i<ma;i++) { /* This is Jacobi iteration */
        s=0.0;
        for (j=0;j<ma;j++)
            if (i != j) s+=a[i][j]*xold[j];
        xold[i]=x[i];
        x[i]=(b[i]-s)/a[i][i];
    }
}

void GaussSeidel_iter(Mat_DP &a, Vec_DP &b,
Vec_DP &x, Vec_DP &xold)
{
    DP s;
    int i,j,ma=a.nrows();
    if ((ma!=a.ncols())||(ma!=b.size())||
        (ma!=x.size())||(ma!=xold.size()))
        {cout<<"Argument error in GaussSeidel_iter\n"<<endl;
        abort();
        }
}

```

```

    }

for (i=0;i<ma;i++) { /* This is Gauss-Seidel iteration */
    s=0.0;
    for (j=0;j<ma;j++){
        if (j <=i-1) s+=a[i][j]*x[j];
        if (j >=i+1) s+=a[i][j]*xold[j];}
    xold[i]=x[i];
    x[i]=(b[i]-s)/a[i][i];
    }
}

int main()
{
    int ma;
    int i,j,k,ikind,ikindold,new1 ;
    ma=6;
    Mat_DP a(ma,ma);
    Vec_DP b(ma),x(ma),xold(ma),sol(ma);
    DP er,s;
    init_srand(); /* Give some values for a and b */
    ranmat2(a,-0.99, 0.99);
    for (i=0;i<ma;i++)
        for (j=0;j<ma;j++)
            a[i][j]=ceil(10*a[i][j]);
    /* We give integer values for the entries of a */
    for (i=0;i<ma;i++)
        {s=0.0;
        for (j=0;j<ma;j++) if (j !=i) s= s+fabs(a[i][j]);
        if (fabs(a[i][i]) < s ) a[i][i]=s+ma/2;}
    /* Diag. entries are increased a bit, to ensure
    convergence of the iteration */
    cout<<"Matrix a ="<<endl;
    showmat2(a,"% 10.5f ");
    /* We make a solution in integer components */
    for (i=0;i<ma;i++) sol[i]=(int)(ma/2)-i;
    /* and, the corresponding RHS: */
    matvecmul(a,sol,b);
    for (i=0;i<ma;i++) {x[i] =0.0; xold[i]=0.0;}
    new1=0; ikind=0;
    do {
        ikindold=ikind;
        printf("Enter iteration: (1) Jacobi, (2) Gauss-Seidel \n");
        scanf("%ld",&ikind);
        printf("\n ***\n ikind = %ld \n",ikind);

```

```

for (i=0;i<ma;i++) printf("      x[%ld]",i);
printf("\n");
      /* Restart with x=0 if ikind !=ikindold */
if (ikind != ikindold)
    for (i=0;i<ma;i++) {x[i] =0.0; xold[i]=0.0;}
printf("\n 1 :");
for (i=0;i<ma;i++) printf(" %9.6f",x[i]);
printf("\n");
for (k=1;k<=40;k++)
    {
        if (ikind ==1) Jacobi_iter(a,b,x,xold);
        if (ikind ==2) GaussSeidel_iter(a,b,x,xold);
        if (k==5*((int)(k/5)) ) {
            printf("%2d :",k);
            for (i=0;i<ma;i++) printf(" %9.6f",x[i]);
        }
    }
for (i=0;i<ma;i++) printf(" %12.9f",x[i]);
s=residual(a,x,b);
printf("\nResidual = %12.4e \n",s);
printf("Enter 0 if iterate again: \n");
scanf("%ld",&new1);
} while (new1==0) ;
return 0;
}
/* FILE: myjgsit.cpp                                     */

```

Ohjelman tulostusta:

Matrix a =

6x6 matrix:

31.00000	7.00000	7.00000	-5.00000	-7.00000	2.00000
3.00000	23.00000	6.00000	2.00000	-7.00000	2.00000
-6.00000	7.00000	26.00000	5.00000	-0.00000	5.00000
7.00000	-0.00000	2.00000	18.00000	-0.00000	-6.00000
7.00000	-5.00000	6.00000	3.00000	32.00000	-8.00000
-7.00000	4.00000	8.00000	10.00000	8.00000	40.00000

Enter iteration: (1) Jacobi, (2) Gauss-Seidel

\*\*\*

ikind = 1

	x[0]	x[1]	x[2]	x[3]	x[4]	x[5]
--	------	------	------	------	------	------

1 :	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
-----	----------	----------	----------	----------	----------	----------



```

5 : 2.902777 2.006929 1.073602 -0.089565 -1.001327 -1.879540 6.9473e+00
10 : 3.007191 2.004650 1.009034 -0.026923 -1.011529 -1.981269 1.0736e+00
15 : 3.002872 2.000989 1.000257 -0.003249 -1.001706 -1.998851 1.5405e-01
20 : 3.000374 2.000122 0.999908 -0.000143 -1.000112 -2.000079 1.6855e-02
25 : 3.000020 2.000006 0.999981 0.000023 -0.999994 -2.000029 1.5958e-03
30 : 2.999998 1.999999 0.999998 0.000006 -0.999997 -2.000004 2.2342e-04
35 : 2.999999 2.000000 1.000000 0.000001 -1.000000 -2.000000 3.1502e-05
40 : 3.000000 2.000000 1.000000 0.000000 -1.000000 -2.000000 3.3040e-06
    2.999999927 1.999999977 1.000000021 0.000000020 -0.999999981 -1.999999978
Residual = 3.3040e-06
Enter 0 if iterate again:
Enter iteration: (1) Jacobi, (2) Gauss-Seidel

```

```

***
ikind = 1
    x[0]      x[1]      x[2]      x[3]      x[4]      x[5]

1 : 3.000000 2.000000 1.000000 0.000000 -1.000000 -2.000000
5 : 3.000000 2.000000 1.000000 -0.000000 -1.000000 -2.000000 3.2154e-07
10 : 3.000000 2.000000 1.000000 -0.000000 -1.000000 -2.000000 4.6960e-08
15 : 3.000000 2.000000 1.000000 -0.000000 -1.000000 -2.000000 6.3876e-09
20 : 3.000000 2.000000 1.000000 -0.000000 -1.000000 -2.000000 6.4617e-10
25 : 3.000000 2.000000 1.000000 0.000000 -1.000000 -2.000000 6.5531e-11
30 : 3.000000 2.000000 1.000000 0.000000 -1.000000 -2.000000 9.8543e-12
35 : 3.000000 2.000000 1.000000 0.000000 -1.000000 -2.000000 1.2763e-12
40 : 3.000000 2.000000 1.000000 0.000000 -1.000000 -2.000000 1.2297e-13
    3.000000000 2.000000000 1.000000000 0.000000000 -1.000000000 -2.000000000
Residual = 1.2297e-13
Enter 0 if iterate again:
Enter iteration: (1) Jacobi, (2) Gauss-Seidel

```

```

***
ikind = 2
    x[0]      x[1]      x[2]      x[3]      x[4]      x[5]

1 : 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
5 : 2.974984 2.025822 0.990099 0.024431 -0.980725 -2.014942 1.1610e+00
10 : 2.997034 2.000487 0.999925 0.002184 -0.998699 -2.001359 1.2265e-01
15 : 3.000071 2.000014 0.999995 -0.000068 -1.000036 -1.999964 3.4590e-03
20 : 3.000007 2.000001 1.000000 -0.000007 -1.000004 -1.999996 3.4302e-04
25 : 3.000000 2.000000 1.000000 0.000000 -1.000000 -2.000000 1.0556e-05
30 : 3.000000 2.000000 1.000000 0.000000 -1.000000 -2.000000 1.0347e-06
35 : 3.000000 2.000000 1.000000 -0.000000 -1.000000 -2.000000 3.1755e-08
40 : 3.000000 2.000000 1.000000 -0.000000 -1.000000 -2.000000 3.1126e-09
    3.000000000 2.000000000 1.000000000 -0.000000000 -1.000000000 -2.000000000

```

```

Residual = 3.1126e-09
Enter 0 if iterate again:
Enter iteration: (1) Jacobi, (2) Gauss-Seidel

***
ikind = 2
      x[0]      x[1]      x[2]      x[3]      x[4]      x[5]

 1 : 3.000000  2.000000  1.000000 -0.000000 -1.000000 -2.000000
 5 : 3.000000  2.000000  1.000000  0.000000 -1.000000 -2.000000 9.5513e-11
10 : 3.000000  2.000000  1.000000  0.000000 -1.000000 -2.000000 9.3610e-12
15 : 3.000000  2.000000  1.000000 -0.000000 -1.000000 -2.000000 2.9735e-13
20 : 3.000000  2.000000  1.000000 -0.000000 -1.000000 -2.000000 3.3751e-14
(... )
40 : 3.000000  2.000000  1.000000  0.000000 -1.000000 -2.000000 0.0000e+00
      3.000000000  2.000000000  1.000000000  0.000000000 -1.000000000 -2.000000000
Residual = 0.0000e+00
Enter 0 if iterate again:

```

**1.36. Permutaatiomatriisit.** Kertomalla  $n \times n$  matriisi vasemmalta voidaan vaihtaa matriisin rivit ja kertomalla oikealta sarakkeet. [www-sivulla](#) on tähän liittyvä esimerkkiohjelma `mymatope.cpp`.

**1.37. Kurssilla käytettävä hakemistorakenne.**

```

nrc02
|- answers
|- data (dates1.dat)
|- drivers (xairy.cpp)
|- lib (libnr.a)
|- responses (sfroid.resp,...)
|- sources (addint.cpp,.. NR ohjelmat)
|- utils (mybugs.h, nr.h, nrutil.h, ...)
|- gnuplot02 (gnuplot-ohjelmia)
|- democpp02 (luentojen demo-ohjelmia, esim. myutl4.cpp )
|- wrk (tyohakemisto)
|
|- harj
|   |- h00
|   |- h01
|   | ..
|   |- h10

```

**1.38. Aputiedosto beg.cpp.** Kohdassa 1.1 mainittua aputiedostoa beg.cpp voimme mukauttaa joustavaan käyttöön kurssillamme tekemällä siihen erilaisia täydennyksiä, ks. alla.

```
/* FILE: beg.cpp begins. */
/* g++ -Wall beg.cpp -L../lib -I../utils -o a -lm -lnr */

#include <cstdlib> // Used in putmat2
#include <cstdio> // Used in putmat2
#include <ctime>
#include <string>
#include <fstream>
#include <iostream>
#include <iomanip>
using namespace std;
#include "nr.h"
#include "matutl02.h"

int main()
{
    Mat_DP a(4,4), ai(4,4), b(4,4), u(4,4);
    ranmat2(a, -1, 1);
    invmat(a, ai);
    unitmat(u);
    matmul(a, ai, b); /* Also b=a*ai; */
    a=b-u; // a= a*inv(a)-Id
    cout <<"a*inv(a)-Id = \n"<<a<<endl;
    cout<<"norm2(a*inv(a)-Id) = "<<mnormp(a,2)<<endl;
}
/* FILE: beg.cpp ends. */
```

**1.39. NR:n esimerkkiohjelmat.** Seuraavasta istuntelokista ilmenee toisaalta mitä vaikeuksia voi tulla vastaan kun kokeilee ohjelmien toimintaa ja toisaalta miten vaikeudet voitetaan. Tässä on katsottu pelkästään ohjelmaa ludcmp, mutta muiden ohjelmien käyttö on täysin analogista.

Pikaohje:

- 1) Get userid to dx0-mat-01.mathstat.helsinki.fi from Seppo.Pulkkinen@fmi.fi
  - 2) Login ssh -X tunnus@dx0-mat-01.mathstat.helsinki.fi
- and give the following commands (assuming that you are using Linux)  
If you use Windows terminal, you need to use putty.

```

506 cd ..
507 mkdir jnk2016
508 cd jnk2016
509 ls /home/NRC
510 cp /home/NRC/*zip .
511 unzip myexamples.zip
512 cd myexamples/
513 ls
514 cmake .
515 make
516 cd democpp
517 ls
518 ./beg
519 cp beg.cpp beg2.cpp
520 gedit beg2.cpp &
521 ./myc beg2.cpp
522 ./mytemp4
523 cd ../demogs1
524 ./beg
525 ls *c
526 ./mysteep
527 cd ../../..
528 cd jnk2016
529 ls
530 mkdir mynrctst
531 cd mynrctst/
532 ls /usr/local/numerical-recipes/
533 find /usr/local/numerical-recipes/recipes_cpp/ -name xludcmp.cpp
534 cp /usr/local/numerical-recipes/recipes_cpp/demo/src/xludcmp.cpp .
535 find /usr/local/numerical-recipes/ -name *a
536 ls
537 g++ -L/usr/local/numerical-recipes/recipes_cpp/ xludcmp.cpp -o a -lnr
538 find /usr/local/numerical-recipes/ -name nr.h
539 g++ -I/usr/local/numerical-recipes/recipes_cpp/utils/ -L/usr/local/numerical-recipes/ ./a
540 ./a
541 find /usr/local/numerical-recipes/ -name 'matrx*'
542 cp /usr/local/numerical-recipes/recipes_cpp/matrx*.dat .
543 ./a
544 ls
545 mkdir doc
546 cd doc
547 history > myhistory.txt

```



## 2 LINEAARISET YHTÄLÖRYHMÄT

**2.1. Lineaariset yhtälöryhmät ja laskennallinen tiede.** Lineaarisia yhtälöryhmiä esiintyy useissa tekniikan ja luonnontieteiden sovelluksissa sekä monenlaisissa tilastollisissa tarkasteluissa yhteiskuntatieteiden ja kaupallisten tieteiden alueella. Laskennallisen tieteen piirissä numeeriset tehtävät sisältävät tavallisesti osanaan lineaarisen yhtälöryhmän ratkaisun. Yhtälöryhmiin johdutaan matemaattisen mallintamisen kautta. Mallia sopivasti yksinkertaistamalla pyritään tavallisesti siihen, että alkuperäinen mahdollisesti epälineaarinen tehtävä voitaisiin palauttaa lineaariseksi eli linearisoida. Esimerkiksi osittaisdifferentiaaliyhtälöiden numeerisen ratkaisun keskeisenä osana on laajojen lineaaristen yhtälöryhmien ratkaisu. Mikäli yhtälöryhmä on niin suuri, että kerroinmatriisin muistitilan tarve tai laskennan vaatima aika muodostuvat ongelmaksi, pyritään hyödyntämään kerroinmatriisin rakennetta tallettamalla vain nolosta eroavat alkioit. Matriisin mahdollista symmetriaa voidaan myös käyttää samaan tapaan muistiresurssien säästämiseksi.

Lineaarisen yhtälöryhmän yleinen muoto on

$$(1) \quad \begin{cases} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n = b_2 \\ \vdots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n = b_m \end{cases}$$

tai matriisien avulla lausuttuna

$$(2) \quad AX = B, \quad A = \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ a_{21} & \cdots & a_{2n} \\ \vdots & & \vdots \\ a_{m1} & \cdots & a_{mn} \end{bmatrix}, \quad X = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \quad B = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix}$$

Lineaarialgebra antaa täydellisen vastauksen kysymykseen matriisiyhtälön (2) ratkaisun olemassaolosta ja yksikäsitteisyydestä. Jos  $M < N$  ei ratkaisu yleensä ole yksikäsitteinen. Jos  $M > N$

ei ratkaisua yleensä ole olemassa lineaarialgebran mielessä, mutta voidaan etsiä ratkaisua pienimmän neliösumman (pns) mielessä. Tapauksessa  $M = N$  yhtälöryhmällä (2) on yksikäsitteinen ratkaisu jos ja vain jos on olemassa käänteismatriisi  $A^{-1}$ .

Matriisin esitystä tietokoneessa tarkasteltiin jo edellisen luvun lopussa. Lähtökohtana on ”looginen koko”  $M \times N$ . Tarvitaan ”fyysinen kaavio” (osa tietokoneen muistia), jonka koko on  $MP \times NP$  ”muistiyksikköä” (yksikön koko riippuu mm. tietotyypistä: int, long int, float, double jne.) ja (tavallisesti)  $MP \geq M$ ,  $NP \geq N$ . Poikkeuksia voi esiintyä, jos matriiseissa tiedetään jotkin rivit nolliksi tai myös, jos nollasta eroavia alkioita on vähän, jolloin matriisia ei talleteta kokonaan, vaan ainoastaan nollasta eroavat alkioita. Jatkossa oletamme, että matriisin alkioita ovat reaalityyppisiä. Matriisia, jossa nollasta eroavia alkioita on vähän sanotaan *harvaksi* (engl. sparse). Sen vastakohta on *täysi* (engl. full) matriisi, jonka kaikki alkioita talletetaan. Harvojen matriisien käsittelyyn on kehitetty algoritmeja, jotka voivat olla merkittävästi tehokkaampia kuin vastaavien täysien matriisien algoritmit.

Matriisiyhtälön (2) ratkaisemisen yhteydessä tarkastelemme seuraavia ongelmia:

- (a)  $Ax = b_j$ ,  $j = 1, \dots, p$  (monta  $b_j$ :tä,  $A$  sama).
- (b)  $M = N$ : Etsi  $A^{-1}$ ,  $\det A$ .
- (c) Singulaariarvohajotelma (singular value decomposition, SVD)
- (d)  $M > N$ : (2):n pns-ratk. *normaaliyhtälöt*.

**2.2. Ratkaisun vaativuus.** Jos  $m = n$ , niin täyden matriisin tapauksessa yhtälöryhmän (1) ratkaisulle tällä kurssilla esitettävät algoritmit ovat vaativuudeltaan (vaativuus ja kompleksisuus ovat synonyymejä) suuruusluokkaa  $n^3$ , kun taas tridiagonaalimatriisin tapauksessa vaativuusluokka on  $n$ . Edellä potenssi 3 ei ole paras mahdollinen: tehokkaammassa ns. Strassenin algoritmissa potenssi

on  $\log_2 7 \approx 2.807$ . Kysymys tarkasta alarajasta yhtälöryhmän (1) vaativuudelle on avoin.

Numeerinen lineaarialgebra on numeerisen analyysin tärkeimpiä osa-aloja sovellusten kannalta. Valtaosa tieteellisen laskennan menetelmistä käyttää osanaan numeerista lineaarialgebraa. Numeerisen lineaarialgebran algoritmien tehokkuus on näinollen tieteellisen laskennan pullonkauloja. Massiivisessa numeerisessa laskennassa yhtälöryhmät voivat sisältää esim. 500 000 muuttujaa. Näin laajojen lineaaristen yhtälöryhmien ratkaisu nykyisin tietokonein vaatii ”täsmälaskentaa”, joka huomioi yhtälöryhmän rakenteen (symmetria, lohkorakenne) ja erityisesti sen, onko matriisi harva.

Lineaarialgebran perusoperaatiot on koottu ns. LINPACK pakettiin. Luettelo numeerisen matematiikan public domain C-ohjelmista on saatavana tiedostona osoitteesta:

`ftp://usc.edu/pub/C-numanal/numcomp-free-c.gz` .

$n \times n$  matriisien kertolaskun vaativuutta verrattiin kokeellisesti potenssiin  $n^3$ . Saatiin seuraavia tuloksia (ohjelma `mycxy.c`).

Time complexity T(n) of n\*n matrix multiplication

n	T(n) (ms)	$n^3$
10	0.00	1000
20	1.00	8000
30	5.00	27000
40	12.00	64000
50	22.00	125000

Voidaan osoittaa, että vaativuudeltaan  $n \times n$  matriisin kääntäminen on samaa luokkaa kuin  $n \times n$  matriisien tulo.

**2.3. Esimerkkejä säännöllisistä matriiseista.** Palautamme mieleen lineaarialgebrasta, että yhtälöryhmällä (1) on tapauksessa  $m = n$  yksikäsitteinen ratkaisu täsmälleen silloin kun matriisi on säännöllinen, ts.  $\det(A) \neq 0$ . Mitään erityisen kätevää ehtoa säännöllisyydelle ei tunneta. Muutamissa erikoistapauksissa ratkeavuus, eli  $A^{-1}$ :n olemassaolo, on selvää tai ainakin tunnettua:

- jos  $A$  on diagonaalimatriisi, jonka diagonaalialkiot ovat nollasta eroavia lukuja  $d_{ii}$ , niin  $A^{-1}$  on myös diagonaalimatriisi ja diagonaalialkiot ovat  $1/d_{ii}$ ,



- jos  $A$  on riveittäin diagonaalipainotteinen
- jos  $A$  on ylä- tai alakolmiomatriisi ja diagonaalialkiot  $\neq 0$
- jos  $A$  on positiividefiniitti

**2.4. Gaussin-Jordanin eliminointi.** Menetelmä soveltuu yhtälöryhmän  $Ax = b_j$ ,  $j = 1, \dots, p$ , ratkaisemiseen.

Hyviä puolia:

- stabiili, jos käytetään täydellistä pivotointia (ks. alla),
- tuottaa myös  $A^{-1}$ :n,
- menetelmän tehokkuus paranee  $p$ :n mukana.

Huono puoli:

- ei sovellu jos käänteismatriisia  $A^{-1}$  ei ole.

Yleensä jatkossa esitettävä LU-menetelmä on parempi.

Olkoon  $A = (a_{ij})$ ,  $Y = (y_{ij})$   $N \times N$  matriisi  $x_j = (x_{j1}, \dots, x_{jN})^T$  sekä  $\mathbf{1}$   $N \times N$  yksikkömatriisi. Samanaikaiset yhtälöryhmät

$$(I) \quad A \cdot x_j = b_j, \quad A \cdot Y = \mathbf{1}, \quad j = 1, 2, 3$$

merkitään lyhyemmin

$$(II) \quad A \cdot [x_1 \ x_2 \ x_3 \ Y] = [b_1 \ b_2 \ b_3 \ \mathbf{1}]$$

Perusominaisuuksia (I):lle:

- Jos vaihdetaan  $A$ :n ja  $b$ :n (II:n) kaksi vaakariviä, niin  $x_j(Y)$  ei muutu.
- Ratkaisu  $x_j(Y)$  ei muutu jos  $A$ :n ja  $b_j$ :n (II:n) jokin vaakarivi korvataan vaakarivien lineaarikombinaatiolla.
- Sarakkeitten vaihto  $A$ :ssa antaa saman ratkaisun vain jos vastaavat  $x_j$ :t ja  $Y$ :n vastaavat rivit vaihdetaan.

Gaussin-Jordanin algoritmi (I):n ratkaisemiseksi, olettaen että esiintyvät jakajat ovat nollassa poikkeavia, on seuraava:

Muodostetaan (I):stä uusi yhtälöryhmä s.e.

- 1. rivi jaetaan  $a_{11}$ :llä
- rivistä  $j, j \neq 1$ , vähennetään alkuperäinen 1. rivi vakiolla  $a_{j1}/a_{11}$  kerrottuna jolloin saadaan uusi yhtälöryhmä (1)', jonka 1. sarakkeen kertoimet ovat  $\delta_{j1}$

Muodostetaan (I)':sta uusi yhtälöryhmä s.e.

- 2. rivi jaetaan  $a'_{22}$ :lla
- $j$ . rivistä,  $j \neq 2$ , vähennetään (1)':n 2. rivi vakiolla  $a'_{j2}/a'_{22}$  kerrottuna jolloin saadaan uusi yhtälöryhmä (I)'', jonka 2. sarakkeen kertoimet ovat  $\delta_{j2}$ .

Toistetaan  $N$  kertaa  $\Rightarrow$  saadaan  $x = Ix = b'$ .

**Pivointi.** Numeerisen tarkkuuden parantamiseksi on todettu tarpeelliseksi käyttää pivointia (tuenta) ts. rivien ja mahdollisesti lisäksi sarakkeiden vaihtoa s.e. ”jakajaksi” kussakin vaiheessa tulee jokin erityisen toivottava luku (joka tapauksessa  $\neq 0$ ). Esimerkiksi jos vaiheessa  $p$  matriisin kertoimet ovat  $(a_{ij}^{(p)})$ , niin voidaan valita tukialkioksi suurin luvuista  $|a_{ij}^{(p)}|$ ,  $i \geq p$  tai  $j \geq p$ . Toinen mahdollinen valinta on *implisiittinen tuenta*, jossa ennen eliminoinnin aloittamista kukin yhtälö erikseen kerrotaan puolittain sellaisella luvulla, että ko. rivin itseisarvoltaan suurimmaksi kertoimeksi tulee 1.

Ks. algm NR: :gaussj, s. 39, Example book, s. 6.

**2.5. Gaussin eliminointi takaisinsijoituksella.** Muuten sama kuin Gaussin-Jordanin algoritmi, mutta vaiheessa  $j$  vähentäminen kohdistuu pelkästään yhtälöihin  $k > j$ . Lopputilanteessa yhtälöryhmä on saatettu muotoon, jossa matriisin kaikki diagonaalin alapuoleiset alkiot ovat nollia:

$$\begin{bmatrix} a'_{11} & a'_{12} & \cdots & a'_{1n} \\ 0 & a'_{22} & \cdots & a'_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & a'_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b'_1 \\ b'_2 \\ \vdots \\ b'_n \end{bmatrix}$$

Tätä tyyppiä oleva matriisi voidaan helposti kääntää aloittamalla alariviltä kuten jo aiemmin nähtiin. Viimeisestä yhtälöstä saadaan  $x_n = b'_n/a'_{nn}$ . Viimeistä edellisestä ratkaistaan  $x_{n-1}$  jne. Menettely, joka antaa luvut  $x_n, x_{n-1}, \dots, x_1$  on nimeltään *takaisinsijoitus* (engl. backsubstitution).

Gaussin algoritmia koskeva kirjallisuus on erittäin laaja ja algoritmin eri versioita kehitetään edelleen. Siitä on kehitetty mm. lohkomuotoisille matriiseille sopivia versioita ja tutkittu rinnakkaislaskennan näkökulmasta.

**2.6. LU-hajoitelma.** Oletetaan, että neliömatriisi  $A$  voidaan kirjoittaa muotoon

$$A = L \cdot U,$$

missä  $L$  on *alacolmiomatriisi* (diagonaalin yläpuoliset alkiot ovat nollia) ja  $U$  on *yläkolmiomatriisi* (diagonaalin alapuoliset alkiot ovat nollia) ja kaikki diagonaalialkiot nolasta eroavia. Yo. tyyppiä olevien matriisien  $L$  ja  $U$  kääntäminen on helppoa kuten jo Gaussin eliminoinnin yhteydessä totesimme.

Lineaarinen yhtälöryhmä  $Ax = b$  voidaan kirjoittaa muotoon

$$(1) \quad Ax = (L \cdot U)x = L(Ux) = b.$$

Ratkaisu tehdään kahdessa vaiheessa niin, että ensin etsitään  $y$  s.e.

$$(2) \quad Ly = b, \quad L = (\alpha_{ij}\Theta_{ij}), \quad \Theta_{ij} = \begin{cases} 0 & j > i \\ 1 & i \geq j \end{cases}$$

ja tämän jälkeen etsitään  $x$  s.e.

$$(3) \quad U \cdot x = y, \quad U = (\beta_{ij}\Theta_{ij}).$$

Nyt siis (1) on hajoitettu kahdeksi yhtälöryhmäksi (2), (3).

Saavutettu etu: Ylä- ja aladiagonaalisten yhtälöryhmien (2), (3) ratkaisu on helppoa (Gaussin eliminointi takaisinsijoituksella).

**2.7. LU-hajoitelman muodostaminen.** Matriisitulon kaavasta saadaan

$$(LU)_{ij} = \sum_{k=1}^n (L)_{ik}(U)_{kj} = \sum_{k=1}^n \alpha_{ik}\Theta_{ik}\beta_{kj}\Theta_{jk}$$

$$(4) \quad (LU)_{ij} = \begin{cases} (i < j) : \alpha_{i1}\beta_{1j} + \alpha_{i2}\beta_{2j} + \dots + \alpha_{ii}\beta_{ij} = a_{ij} \\ (i = j) : \alpha_{i1}\beta_{1j} + \alpha_{i2}\beta_{2j} + \dots + \alpha_{ii}\beta_{jj} = a_{ij} \\ (i > j) : \alpha_{i1}\beta_{1j} + \alpha_{i2}\beta_{2j} + \dots + \alpha_{ij}\beta_{jj} = a_{ij} \end{cases}$$

Yhtälöryhmässä (4) on  $N^2$  yhtälöä,  $N^2 + N$  tuntematonta  $\alpha_{ij}, \beta_{ij}$ . Näistä voidaan siis  $N$  kpl kiinnittää mielivaltaisesti ja ratkaista muut. Seuraavassa algoritmissa kiinnitetään diagonaalialkiot

$$(5) \quad \alpha_{ii} = 1 \quad \forall i = 1, \dots, N.$$

**Croutin algoritmi.** Algm antaa luvut  $\alpha, \beta$  seuraavasti:

- a) Asetetaan  $\alpha_{11} = 1$  ja  $\beta_{11} = a_{11}$
- b) Jokaisella  $j = 2, 3, \dots, N$  tehdään seuraavat 2 operaatiota. Kun  $i = 1, \dots, j$  ratkaistaan (4) & (5):n nojalla

$$\beta_{ij} = a_{ij} - \sum_{k=1}^{i-1} \alpha_{ik}\beta_{kj} \quad (i \leq j)$$

ja kun  $i = j + 1, \dots, N$  ratkaistaan (4):n nojalla

$$\alpha_{ij} = \frac{1}{\beta_{jj}}(a_{ij} - \sum_{k=1}^{j-1} \alpha_{ik}\beta_{kj}) \quad (i > j).$$

Croutin algoritmi generoi siis matriisin

$$\begin{bmatrix} \beta_{11} & \dots & \beta_{1n} \\ \alpha_{21} & \dots & \beta_{2n} \\ \vdots & & \vdots \\ \alpha_{N1} & \dots & \beta_{NN} \end{bmatrix}$$

sarakkeittain vasemmalta oikealle ja kunkin sarakkeen ylhäältä alas.

Croutin algoritmin yhteydessä tuenta on olennaista, ts. ylläolevaa algoritmia on vielä tältä osin täydennettävä. Osittainen tuenta (rivien vaihto) on kirjan algoritmissa `NR::ludcmp` (s. 46) todettu riittäväksi.

**Olemassaolo.** LU-hajoitelmaa ei ole olemassa kaikille säännöllisille neliömatriiseille. Kuitenkin säännölliselle matriisille on olemassa sellainen permutaatiomatriisi  $P$ , että  $PA$ :lla on LU-hajoitelma.

Kun LU-hajoitelma on tehty ohjelman `NR::ludcmp` avulla, voidaan yhtälöryhmän ratkaisu toteuttaa algoritmilla `NR::lubksb` (s. 47).

**Matriisin kääntäminen LU-hajoitelmalla.** LU-hajoitelman avulla matriisin kääntäminen palautuu ylä- ja aladiagonaalisten matriisien kääntämiseksi ja on siis helppoa (Gauss takaisinsijoituksella). Olkoon  $A$  annettu  $n \times n$  matriisi ja  $Y$  yksikkömatriisi,  $j$ :s sarake  $y_j$ . Kullakin  $j$  ratkaistaan  $LUx = y_j$  `NR::lubksb:n` avulla, jolloin saadaan  $x_j$ . Nyt  $A^{-1} = [x_1, \dots, x_n]$ . Kurssin matriisipaketissa `matut102.cpp` matriisinkääntö tehtiin `NR::gaussj:n` avulla.

**2.8. Matriisin determinantti.** LU-hajoitelmana annetun matriisin determinantti voidaan ilmaista tulona  $\beta_{11} \dots \beta_{NN}$ .

Jos käytetään (kuten kirjan ohjelmissa) tuentaa niin rivien järjestys on ehkä muuttunut ja siis myös determinantin merkki on voinut muuttua – näin käy, jos vaihtojen lukumäärä on pariton. Determinantti saadaan seuraavasti:

```
DP t;
NR::ludcmp(aa,indx, t); /* Returns t = +1 or -1 */
for (int j=0;j<n;j++){ t = t*aa[j][j];}
return t;
```

**2.9. Tridiagonaalinen yhtälöryhmä.** Sanotaan, että matriisi on tridiagonaalinen, jos sen nollostasta eroavat alkiot ovat  $a_{ij}$ ,  $|i - j| \leq 1$ . Tällainen matriisi on tärkeä esimerkki harvasta matriisista. Koska

harvojen matriisien alkioista suurin osa on nollia, on tarkoituksenmukaista käyttää hyväksi tätä tietoa matriisikertolaskussa ja muissa matriisialgebran operaatioissa sekä myös siten että talletetaan vain nollostasta eroavat alkiot.

Algoritmi `NR::tridag` (s. 51) ratkaisee tridiagonaalisen yhtälöryhmän em. näkökohdat huomioiden.

**2.10. Jälki-iterointi.** Oletetaan, että  $Ax = b$ , ja tunnetaan  $x + \delta x$  ( $\delta x = \text{virhe}$ ), jolloin

$$A(x + \delta x) = b + \delta b \Rightarrow A \cdot \delta x = \delta b = A(x + \delta x) - b.$$

Likiarvon parantamiseksi tästä voidaan ratkaista  $\delta x$ . Saadaan  $x = x + \delta x - \delta x$ . Algoritmi `NR::mprove` (s. 56) käyttää tätä ideaa.

**2.11. Singulaariarvohajoitelma (SVD).** Esittelemme nyt, ilman todistuksia kuten tavallista, erään matriisihajoitelman, jonka avulla saadaan monipuolista tietoa matriisin numeerisista ominaisuuksista. Tämä matriisihajoitelma, SVD, on tietokoneiden yleistyessä noussut keskeiseen asemaan numeerisessa lineaarialgebrassa, mutta sitä ei välttämättä esitetä lineaarialgebran peruskurssilla.

Aluksi palautamme mieleen että  $m \times n$  matriisin  $A = (a_{ij})$  *transpoosi*  $A^T$  on  $n \times m$  matriisi  $(A^T)_{ij} = a_{ji}$ .  $m \times n$  matriisi on *sarakkeittain ortogonaalinen*, jos sen sarakkeet ovat ortogonaalisia. Neliömatriisi  $B$  on *ortogonaalinen*, jos se on säännöllinen (ts.  $\det(B) \neq 0$ ) ja  $B^T = B^{-1}$ . Seuraavan tärkeän lauseen todistus löytyy pidemmälle menevistä numeriiikan oppikirjoista, esim. Stoer-Bulirsch, s. 332-334, Lause 6.4.10 tai Golub-Van Loan, s. 71, Lause 2.5.1.

**2.12. Lause (SVD).** Reaalille  $m \times n$ ,  $m \geq n$ , matriisille  $A$  on olemassa sarakkeittain ortogonaalinen  $m \times n$  matriisi  $U$  ja ortogonaalinen  $n \times n$  matriisi  $V$  s.e.

$$A = UWV^T,$$

missä  $W$  on  $n \times n$  diagonaalimatriisi. Jos  $W = \text{diag}(w_1, \dots, w_n)$ , niin  $w_1 \geq w_2 \geq \dots \geq w_n \geq 0$ .

Lause 2.12 vastannee suunnilleen `algm:n NR::svdcmp` sisältöä-valitettavasti tekijät eivät ole täsmällisesti ilmaisseet mitä ko. `algm` tekee. Matemaattisessa kirjallisuudessa tulos formuloidaan joskus hieman toisin:  $U$  ja  $V$  ovat  $m \times m$  ja  $n \times n$  ortogonaalimatriiseja ja  $W$   $m \times n$  diagonaalimatriisi, lisäksi ehto  $m \geq n$  on epäoleellinen. Ohjelma `NR::svdcmp` (versio 2.08) tuottaa yo. tyyppiä olevan hajoitelman, mutta siinä singulaariarvot eivät ole kasvavassa järjestyksessä.

Diagonaalimatriisi  $W$ :n alkiot ovat  $A$ :n *singulaariarvoja*. Siinä tapauksessa, että  $A$  on  $n \times n$  neliömatriisi ja kaikki singulaariarvot ovat nollaa suurempia määritellään  $A$ :n *kuntoisuuskulu* lausekkeella

$$\text{cond}(A) = w_{\max}/w_{\min},$$

missä  $w_{\max}$  ja  $w_{\min}$  ovat vastaavasti suurin ja pienin singulaariarvo.

Tapauksessa  $\det(A) \neq 0$  on  $\text{cond}(A)$  kvantitatiivinen mitta sille, ”kuinka paha”  $A$  on numeerisilta ominaisuuksiltaan tai kuinka lähellä  $A$  on singulaarista matriisiä. (Välihuomautus:  $|\det(A)|$  ei ole sopiva mitta tähän tarkoitukseen, sillä  $A$  ja  $\lambda A$  ovat numeerisilta ominaisuuksiltaan samanveroisia kun taas  $\det(\lambda A) = \lambda^n \det(A)$ , jos  $A$  on  $n \times n$  matriisi ja  $\lambda \in R$ .) Edellisessä luvussa esittelimme jo menettelyn sille, miten  $\text{cond}(A)$  voidaan laskea kirjan algoritmin `svdcmp` avulla.  $n \times n$  yhtälöryhmän ratkaisu voidaan tehdä käyttäen algoritmeja `NR::svdcmp` ja `NR::svbksb`. Jos  $m > n$  saadaan ratkaisu pienimmän neliösumman mielessä.

**2.13. Pseudoinverssi.** Reaaliluvun  $\sigma \neq 0$  pseudoinverssi on  $1/\sigma$  ja 0:n pseudoinverssi on 0. Diagonaalimatriisin  $W$  pseudoinverssi  $W^+$  (l. valesymmetrisematriisi) saadaan transponoimalla matriisi ja korvaamalla diagonaali-alkiot pseudoinversseillään. Yleisen  $m \times n$ ,  $m \geq n$ , matriisin  $A$ , jolla on singulaariarvohajotelma  $UWV^T$ , pseudoinverssi on  $n \times m$  matriisi  $A^+ = VW^+U^T$ . Siinä tapauksessa, että  $A$  on ei-singulaarinen neliömatriisi sen pseudoinverssi on sama kuin  $A^{-1}$ . Kaikissa tapauksissa yhtälöryhmän  $Ax = b$  PNS-ratkaisu saadaan kaavasta  $x = A^+b$ .

Pseudoinverssi em. ideoiden mukaisesti tapauksessa  $m \geq n$  on implementoitu pakettiin `solve.cpp`, joka esitellään alempana.

Pseudoinverssi on olemassa myös tapauksessa  $m < n$ , mutta em. ideaan pohjautuva implementointi ei näytä onnistuvan. Sekä NR että GSL eivät kumpikaan sano juuri mitään tästä tapauksesta. Onnistumiskriteerinä voi pitää allamainittuja Mooren-Penrosen ehtoja (a)-(d), jotka karakterisoivat pseudoinverssin.

Tapauksen  $m < n$  selvittämiseksi palaamme ensin tapaukseen  $m > n$ . Oletamme, että  $m > n$  ja  $A$ :n sarakeasteluku on  $n$  ts. että  $A$ :n sarakevektorit muodostavat lin.riippumattoman joukon. Silloin voidaan helposti osoittaa, että  $A^T A$  on positiivisesti definiitti ja siis kääntyvä. Yht.ryhmä  $Ax = b$  voidaan siis kirjoittaa  $A^T Ax = A^T b$  ja ratkaista kaavalla

$$(2.13a) \quad x = (A^T A)^{-1} A^T b,$$

josta päätellään, että  $A^+ \equiv (A^T A)^{-1} A^T$ . Siinä tapauksessa, että  $m < n$  menetellään vastaavasti, olettaen, että  $A$ :n riviasteluku on  $m$  (jolloin  $A^T$ :n sarakeasteluku on  $m$ ). Soveltaen edelläolevaa  $A^T$ :hen, todetaan, että on olemassa  $((A^T)^T A^T)^{-1} = (AA^T)^{-1}$ , joten  $Ax = b$  voidaan kirjoittaa muotoon

$$Ax = (AA^T)(AA^T)^{-1}b$$

josta

$$(2.13b) \quad x = A^+ b; \quad A^+ \equiv A^T (AA^T)^{-1} b.$$

Yhdistäen ylläolevat kaavat todetaan, että jokaiselle  $m \times n$  matriisille  $A$ , jonka asteluku (pienempi riviasteluvusta ja sarakeasteluvusta) on  $\min\{m, n\}$ , voidaan pseudoinverssi konstruoida kaavoilla (2.13a) ja (2.13b).

Tiedoston `solve.cpp` osana oleva `pseudoinv` tuottaa  $m \times n$  matriisin pseudoinverssin sekä tapauksissa  $m \geq n$  että  $m \leq n$ . `Algm:n` `pseudoinv` toimintaa ja Moore-Penrosen yhtälöiden pätemistä matriisiparille  $A$ ,  $A^+$  testataan ohjelmalla `mypseudo.cpp`, joka on `www-` sivulla.



**2.14. Moore-Penrosen ehdot.** Voidaan osoittaa, että matriisi  $A^+$  on  $A$ :n pseudoinverssi jos ja vain jos se toteuttaa seuraavat ehdot:

$$\left\{ \begin{array}{l} (a) \quad AA^+A = A, \\ (b) \quad A^+AA^+ = A^+, \\ (c) \quad (AA^+)^T = AA^+, \\ (d) \quad (A^+A)^T = A^+A. \end{array} \right.$$

**2.15. Singulaariarvojen editointi.** NR esittää seuraavan idean, *singulaariarvojen editoinnin*, käyttämistä  $n \times n$  yhtälöryhmän ratkaisussa tapauksissa, joissa  $\text{cond}(A)$  on suuri. Ensin muodostetaan singulaariarvohajoitelma ja kiinnitetään jokin pieni luku, *editointikyynys*, TINY (esim.  $10^{-6}$ ). Huomataan, että

$$A^+ = V[\text{diag}(1/w_j)]U^T.$$

Seuraavaksi korvataan nolllalla kaikki sellaiset singulaariarvot  $w[i]$  jotka ovat pienempiä kuin  $TINY * w_{max}$  missä  $w_{max}$  on suurin singulaariarvoista. Tämä NR:n idea kuulostaa paradoksaaliselta, mutta idealle ovat perusteena pseudoinverssin eräät ominaisuudet. Tämän jälkeen ratkaisu etenee kuten edellä algoritmien `svdcmp` ja `svbksb` avulla. Lukija voi halutessaan kokeellisesti varmistua idean hyödyllisyydestä esim. tapauksessa, jossa yhtälöryhmän matriisi on  $n \times n$  Hilbertin matriisi alkioina  $a_{i,j} = 1/(i + j - 1)$ .

**2.16. Esimerkki. Yhtälöryhmällä**

$$\left( \begin{array}{l} 2.000x + 0.6667y = 2.000; \\ 1.000x + 0.3333y = 1.000 \end{array} \right. \quad \text{cond}(a) = 5.5556 \cdot 10^4$$

on 1-käsitteinen ratkaisu  $x = 1.000$ ,  $y = 0.000$ . Sen sijaan yhtälöryhmällä

$$\left( \begin{array}{l} 2.000x + 0.6666y = 2.000 \\ \frac{1}{2}(2.000x + 0.6666y) = 1.000 \end{array} \right.$$

on äärettömän monta ratkaisua

$$\left( \begin{array}{l} x = 1.000 - 0.3333k \\ y = k, \quad k \in R^1, \end{array} \right. .$$

Huomaa, että  $\text{cond}(a)$  on suuri, ja siten tehtävä epästabiili.

**2.17. Yhteenveto NR:n yhtälöryhmän ratkaisualgoritmeista.** Yhtälöryhmän ratkaisussa ei yleensä tarvita käänteismatriisia. NR suosittaa LU-menetelmän käyttöä lineaarisen yhtälöryhmän ratkaisussa. Allaoleva ohjelma `mysolve` ja siihen liittyvä paketti `solve.cpp` sisältävät yhteenvedon NR:n erilaisista ratkaisumenetelmistä (`GJsolve`, `LUsolve`, `SVDsolve`). Näistä `SVDsolve` sisältää singulaariarvojen editoinnin kiinteällä editointikykyksellä. `SVDsolve2`:ssa editointikykyksen voi antaa parametrina. Tulostuksesta ilmenee, että virhe on kaikissa tapauksissa noin luokkaa  $10^{-14}$ .

```
/* FILE: mysolve3.cpp begins. */
/* g++ mysolve3.cpp -L../lib -I../utils -o a -lm -lnr */

#include <cstdlib>
#include <cstdio>
#include <ctime>

using namespace std;
#include "nr.h"
#include "matut102.h"
#include "solve.h"

void DoTest(int n)
{
    time_t time1, time2;
    int i,m;
    m=500; /* 1000 nxn lin. equations are solved.
            difftime returns time in seconds, hence
            for T(n) the unit is millisecond, ms */
    Mat_DP a(n,n), atmp(n,n);
    Vec_DP rhs(n),sol(n), err(4), tmp(n), d(4);
    ranmat2(a,-1.0,1.0);
    atmp=a;
    sol=1.0;
    matvecmul(a,sol,rhs);
    /* We build a system a*x = rhs with the
       exact solution x= 1. */
    time1=time(NULL);
    for (i=0;i<m; i++){ GJsolve(atmp,rhs,sol);}
    time2=time(NULL);
```

```

for (int ii=0;ii<n;ii++) tmp[ii]=sol[ii]-1;
err[0]=vnormp(tmp,2.0);
d[0]= difftime(time2,time1);
    atmp=a;
time1=time(NULL);
for (i=0;i<m; i++){ LUsolve(atmp,rhs,sol);}
atmp=a;
time2=time(NULL);
for (int ii=0;ii<n;ii++) tmp[ii]=sol[ii]-1;
err[1]=vnormp(tmp,2.0);
d[1]= difftime(time2,time1);
atmp=a;
time1=time(NULL);
for (i=0;i<m; i++){ SVDsolve(atmp,rhs,sol);}
time2=time(NULL);
for (int ii=0;ii<n;ii++) tmp[ii]=sol[ii]-1;
err[2]=vnormp(tmp,2.0);
d[2]= difftime(time2,time1);
atmp=a;
time1=time(NULL);
for (i=0;i<m; i++){ QRsolve(atmp,rhs,sol);}
time2=time(NULL);
for (int ii=0;ii<n;ii++) tmp[ii]=sol[ii]-1;
err[3]=vnormp(tmp,2.0);
d[3]= difftime(time2,time1);
printf("%4d   %8.21f   %8.21f   %8.21f   %8.21f   %10d\n",
        n, d[0], d[1], d[2],d[3],n*n*n);
printf("err=  %8.2e   %8.2e   %8.2e   %8.2e \n",
        err[0], err[1], err[2],err[3]);
}
int main(void)
{
    init_srand();
    printf("\nTime complexity of n*n linear system solution\n");
    printf("  n      GJsolve    LUsolve    SVDsolve    QRsolve (ms)    n^3\n");
    for (int j=1;j<7;j++){
        DoTest(j*10);
    }
    return 0;
}
/* FILE: mysolve3.cpp ends.                                     */

/*

```

Time complexity of n\*n linear system solution

n	GJsolve	LUsolve	SVDsolve	QRsolve (ms)	n <sup>3</sup>
10	0.00	0.00	2.00	0.00	1000
err=	3.42e-15	4.69e-15	4.62e-15	2.35e-15	
20	1.00	1.00	9.00	1.00	8000
err=	2.60e-14	2.53e-14	2.28e-14	3.88e-14	
30	5.00	1.00	29.00	2.00	27000
err=	5.12e-15	7.42e-14	1.07e-13	1.78e-14	
40	11.00	3.00	68.00	5.00	64000
err=	2.32e-14	1.73e-14	2.82e-14	3.37e-14	
50	22.00	6.00	130.00	10.00	125000
err=	1.14e-14	2.12e-14	3.54e-14	1.83e-14	
60	37.00	10.00	221.00	16.00	216000
err=	2.29e-14	3.43e-14	4.86e-14	6.74e-14	

```

*/
/* FILE: solve.cpp begins.                               */
/* Last updated: 2002-01-18                             */
#include "matut102.h"
#include "solve.h"

#include <cstdlib> // Used in putmat2
#include <cstdio>  // Used in putmat2
#include <ctime>
#include <string>
#include <fstream>
#include <iostream>
#include <iomanip>
using namespace std;

void GJsolve(Mat_DP &a, Vec_DP &b, Vec_DP &x)
/* If a is nxn and b is nx1 this
   returns the solution x of a*x=b */
{
    int ma=a.nrows(), na=a.ncols(), mb=b.size(),mx=x.size() ;
    if ((ma!=na)|| (ma!=mb)|| (ma!=mx))
        {cout<<"Matrix/vector dimension error in QRsolve. "
          <<endl;
          abort();
        }
    int n=ma;
    Vec_INT indx(n);
    Mat_DP ac(n,n), sol(n,1);
    ac=a;
    for (int j=0;j<n;j++) sol[j][0]=b[j];
}

```

```

        NR::gaussj(ac,sol);
        for (int j=0;j<n;j++) x[j]=sol[j][0];
    }

void LUSolve(Mat_DP &a, Vec_DP &b, Vec_DP &x)
/* If a is nxn and b is nx1 this
   returns the solution x of a*x=b */

{
    int ma=a.nrows(), na=a.ncols(), mb=b.size(),mx=x.size() ;
    if ((ma!=na)|| (ma!=mb)|| (ma!=mx))
        {cout<<"Matrix/vector dimension error in QRsolve. "
          <<endl;
         abort();
        }
    int n=ma; DP p;
    Vec_INT indx(n);
    Mat_DP ac(n,n);
    Vec_DP sol(n);
    ac=a;
    sol=b;
    NR::ludcmp(ac,indx,p);
    NR::lubksb(ac,indx,sol);
    x=sol;
}

void QRsolve(Mat_DP &a, Vec_DP &b, Vec_DP &x)
/* If a is nxn and b is nx1 this
   returns the solution x of a*x=b */

{
    int ma=a.nrows(), na=a.ncols(), mb=b.size(),mx=x.size() ;
    if ((ma!=na)|| (ma!=mb)|| (ma!=mx))
        {cout<<"Matrix/vector dimension error in QRsolve. "
          <<endl;
         abort();
        }
    Vec_DP c(ma), d(ma);
    Mat_DP ac(ma,ma);
    ac=a;
    bool sing;
    NR::qrdcmp(ac, c, d, sing);
    if (sing ==1) {cout<<"Singularity in qrdcmp"<<endl;
                  abort();}
    x=b;
}

```

```

    NR::qrsolv(ac,c,d,x);

}

void SVDsolve(Mat_DP &a, Vec_DP &b, Vec_DP &x)
/*
    If a is mxn and b is mx1 this
    returns the LSQ-solution x of a*x=b
    Singular value w[i] is set = 0 if
    w[i]/max{w[k]}< 1e-6;
*/

{
    int ma=a.nrows(), na=a.ncols(), mb=b.size(),mx=x.size() ;
    if ((na!=mx)|| (ma!=mb))
        {cout<<"Matrix/vector dimension error in SVDsolve. "
          <<endl;
         abort();
        }
    Vec_DP w(na), c(ma), xnum(na);
    Mat_DP u(ma,na),v(na,na);
    u=a;
    c=b;
    NR::svdcmp(u,w,v);
    // find maximum singular value
    DP wmax=0.0, wmin;
    for (int k=0;k<na;k++)
        if (w[k] > wmax) wmax=w[k];
    // define "small"
    wmin=wmax*(1.0e-6);
    // zero the "small" singular values
    for (int k=0;k<na;k++)
        if (w[k] < wmin) w[k]=0.0;
    NR::svbksb(u,w,v,c,xnum);
    x=xnum;
}

void SVDsolve2(Mat_DP &a, Vec_DP &b, Vec_DP &x,DP eps_edit)
/*
    If a is mxn and b is mx1 this
    returns the LSQ-solution x of a*x=b
    Singular value w[i] is set = 0 if
    w[i]/max{w[k]}< eps_edit;
*/

```

```

{

int ma=a.nrows(), na=a.ncols(), mb=b.size(),mx=x.size() ;
if ((na!=mx)||(ma!=mb))
    {cout<<"Matrix/vector dimension error in SVDSolve. "
      <<endl;
      abort();
    }
Vec_DP w(na), c(ma), xnum(na);
Mat_DP u(ma,na),v(na,na);
u=a;
c=b;
NR::svdcmp(u,w,v);
// find maximum singular value
DP wmax=0.0, wmin;
for (int k=0;k<na;k++)
    if (w[k] > wmax) wmax=w[k];
// define "small"
wmin=wmax*(eps_edit);
// zero the "small" singular values
for (int k=0;k<na;k++)
    if (w[k] < wmin) w[k]=0.0;
NR::svbksb(u,w,v,c,xnum);
x=xnum;
}

```

```

void psinv(Mat_DP &a, Mat_DP &pia)
/* psinv is used by pseudoinv to
   form the pseudoinverse of a matrix
*/
{
int ma=a.nrows(), na=a.ncols(),
    mb=pia.nrows(),nb=pia.ncols() ;
if ((ma>na)||(mb!=na)||(nb!=ma))
    {cout<<"Matrix/vector dimension error in psinv2. "
      <<endl;
      abort();
    }
Mat_DP a1(ma,ma),a2(ma,ma), aT(na,ma),a3(na,na),a4(na,na);
transp(a,aT);
a3=aT*a;
void pseudoinv(Mat_DP& ,Mat_DP&);
pseudoinv(a3,a4); /* psinv2 calls pseudoinv with m=n */
pia=a4*aT;
}

```

```

}

void pseudoinv(Mat_DP &a, Mat_DP &pia)
/*
    If a is mxn matrix, then pia
    will be nxm, the pseudoinverse of a.
*/
{
    int ma=a.nrows(), na=a.ncols(),
        mb=pia.nrows(),nb=pia.ncols() ;
    if ((na!=mb)|| (ma!=nb))
        {cout<<"Matrix/vector dimension error in pseudoinv. "
            <<endl;
        abort();
        }
    if (ma<na) psinv(a,pia);
    else
        {
            Vec_DP w(na); //, c(ma), xnum(ma);
            Mat_DP u(ma,na),v(na,na),wi(na,na);
            u=a;
            wi=0.0;
            NR::svdcmp(u,w,v);
            for (int i=0;i<w.size();i++)
                wi[i][i]= w[i]*w[i]>1e-16? 1.0/w[i]: 0;
            Mat_DP uT(na,ma);
            transp(u,uT);
            Mat_DP tmp(na,ma);
            tmp=wi*uT;
            pia= v*tmp;
        }
}
/* FILE: solve.cpp ends.                                     */

```

Em. QRsolve on toteutettu NR algm:ien pohjalta– vastaava kotikutoinen algm on ohjelmassa myhouse2.cpp.

**2.18. Residuaali ja kuntoisuusluku.** Jos  $\tilde{x}$  on yhtälöryhmän  $Ax = b$  ratkaisun approksimaatio, niin tuntuu luonnolliselta ajatella, että jos *residuaali*  $r = A\tilde{x} - b$  on pieni, niin myös  $\|x - \tilde{x}\|$  olisi pieni. Usein näin onkin, mutta ei aina.

**2.19. Lause.** Seuraavat epäyhtälöt ovat voimassa, jos  $A^{-1}$  on ole-



massa:

$$\begin{aligned} \|x - \tilde{x}\| &\leq \|r\| \|A^{-1}\| \\ \frac{\|x - \tilde{x}\|}{\|x\|} &\leq \|A\| \|A^{-1}\| \frac{\|r\|}{\|b\|}, \end{aligned}$$

kun  $x \neq 0, b \neq 0$ .

**Tod.** Koska  $A$  on ei-singulaarinen, yhtälöstä  $r = b - A\tilde{x} = Ax - A\tilde{x}$  seuraa  $x - \tilde{x} = A^{-1}r$  joten

$$\|x - \tilde{x}\| = \|A^{-1}r\| \leq \|A^{-1}\| \|r\|.$$

(Lineaarikuvaukselle  $B$  aina  $\|Bx\| \leq \|B\| \|x\|$ .) Toisaalta ehdosta  $b = Ax$  saadaan  $\|b\| \leq \|A\| \|x\|$  ja

$$\frac{\|x - \tilde{x}\|}{\|x\|} \leq \frac{\|A\| \|A^{-1}\| \|r\|}{\|b\|}.$$

Voidaan osoittaa, että  $\|A\| \|A^{-1}\|$ , jota yo. lauseen nojalla voitaisiin sanoa virheen vahvistumiskertoimeksi, on sama kuin aikaisemmin esitelty kuntoisuusluku  $\text{cond}(A)$ .

**2.20. Kuntoisuusluvun säätö.** Testaustarkoituksia varten voi olla kätevää tuottaa matriiseja, joiden kuntoisuusluku on ennalta annettu. Tähän päästään seuraavasti:

- (1) generoidaan satunnaismatriisin  $a$  SVD,  $a = usv$ ,
- (2) muutetaan singulaariarvoja  $s$  suotuisasti, tuloksena  $s_1$ ,
- (3) muokatulla matriisilla  $b = us_1v$  on haluttu  $\text{cond}(b)$ .

Nämä ideat on implementoitu ohjelmiin `rancon` ja `myrancon`, jotka löytyvät [www-sivulta](http://www.sivulta).

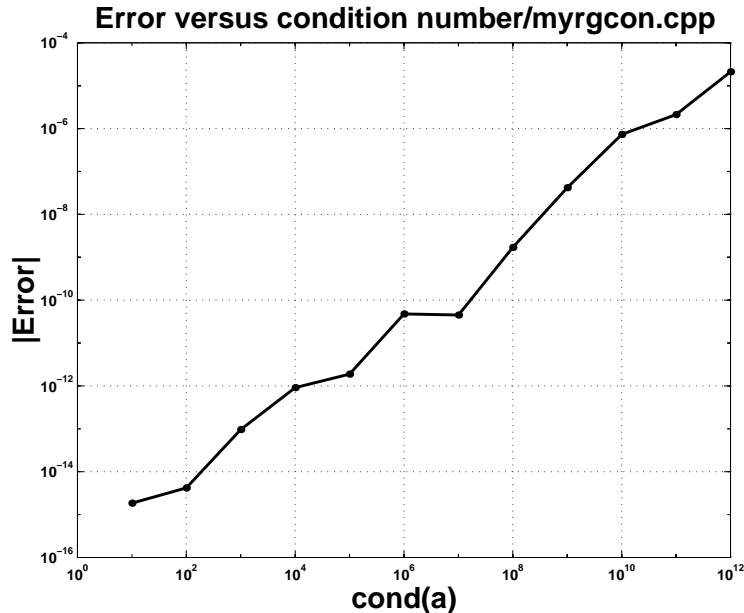
**2.21. Kuntoisuusluvun vaikutus virheeseen.** Lineaarisen yhtälöryhmän  $Ax = b$ , missä  $A$  on  $n \times n$  matriisi, numeerisessa ratkaisemisessa esiintyvää virhettä voidaan tutkia seuraavalla laskennallisella kokeella, jossa käytämme hyväksi edellä esitettyä ideaa kuntoisuusluvun säätöön. Ratkaisijana on `LUsolve`. Virheen mittarina

käytämme todellista virhettä, ts. konstruoimme lin. systeemin, jonka ratkaisukin on tunnettu. Annamme  $\text{cond}(A)$ :n kasvaa ja piirrämme kuvan virheestä.

Seuraavassa kuvassa on esitetty virhe  $\text{cond}(A)$ :n funktiona loglog-skaalassa.

Tämä on toteutettu ohjelmana `myrgcon.cpp` [www-sivulla](#).

cond(a)	residual	error
1.0000e+01	3.3601e-15	1.8444e-15
1.0000e+02	1.5906e-15	4.1836e-15
1.0000e+03	7.8554e-16	9.6393e-14
1.0000e+04	1.4100e-15	9.1327e-13
1.0000e+05	2.8682e-16	1.8822e-12
1.0000e+06	2.2814e-15	4.7712e-11
1.0000e+07	4.1732e-16	4.5108e-11
1.0000e+08	5.7220e-16	1.6954e-09
1.0000e+09	1.4402e-15	4.2031e-08
1.0000e+10	9.5060e-16	7.2847e-07
1.0000e+11	5.8237e-16	2.1276e-06
1.0000e+12	9.9959e-16	2.1141e-05



Johtopäätöksenä kuviosta todetaan, että käytetyssä loglog-akselistossa virheen ja kuntoisuusluvun välillä on likimain lineaarinen riippuvuus. Lisäksi todetaan, että residuaali on koko ajan samaa suuruusluokkaa vaikka virhe tulee  $10^{10}$  kertaiseksi.

**2.22. Sovellus PNS-ongelmiin.** Astetta  $p$  olevan polynomien sovittaminen dataan  $(x_j, y_j), j = 1, \dots, m$ , missä  $x_j < x_{j+1}$  ja  $m > p$ , johtaa ylimäärätyyn lineaariseen yht.ryhmään. Olkoon sovittava polynomi  $f(w, x) = \sum_{j=0}^p w[j]x^{p-j}$ . Yhtälöryhmä on

$$\begin{cases} f(w, x_1) = y_1 \\ \vdots \\ f(w, x_m) = y_m \end{cases}$$

Käyttämällä C-kielen 0-kantaista indeksointia kerroinmatriisille  $A$  pätee  $a[i][j] = x_{i+1}^{p-j}, i = 0, \dots, m-1, j = 0, \dots, p$  ja yhtälöryhmä voidaan kirjoittaa muotoon  $Aw = y$  ja sen ratkaisu  $w = A^+y$  löydetään esim. `SVDsolve`:lla.

PNS-tehtävään liittyvät *normaaliyhtälöt* ovat

$$\frac{\partial S(w)}{\partial w_j} = 0; \quad S(w) = \sum_{j=1}^m (f(w, x_j) - y_j)^2; \quad j = 1, \dots, m.$$

Siinä tapauksessa, että funktiolla  $f(w, x)$  on muoto  $\sum_{j=0}^p w_j g_j(x)$  (kuten polynomisovituksessa, jossa  $g_j(x) = x^j$ ) osittaisderivoinnit on helppo laskea ja normaaliyhtälöt muodostavat lin. yhtälöryhmän kertoimille  $w_j$ . Kaava  $w = A^+y$  antaa näiden ratkaisun.

```
/* FILE: myffit2.cpp begins. */
/* g++ myffit2.cpp -L../lib -I../utils -o myffit2 -lm -lnr */
```

(.....)

```
Vec_DP polyfit(Vec_DP &x, Vec_DP &y, int deg )
{ int ndata=x.size();
  Vec_DP coef(deg+1);
  Mat_DP a(ndata,deg+1);
  if ((ndata<deg+1) ||(ndata!=y.size()))
    {cout<<"Argument error in polyfit\n" <<endl;
     abort();
    }
  for (int i =0;i<ndata;i++)
  for (int j=0;j<=deg;j++)
    a[i][j]=pow(x[i],(double)(deg-j));
  SVDsolve(a,y,coef);
```

```

    cout<<"Coefficient c= \n"<<coef<<endl;
    cout<<
"Format: c[0]x^p +c[1]x^{p-1} +...+ c[p] ; p= c.size()-1\n";
    getchar();
    return coef;
}
int main()
{
    for (int poldeg=2;poldeg<7;poldeg++)
        {
            int mdata=poldeg+9;
            Vec_DP x(mdata), y(mdata), coef(poldeg+1);
            GenData(x,y);
            coef=polyfit(x,y,poldeg);
            PlotPolyData(x,y,coef);
        }
}
/* FILE: myffit2.cpp ends. */

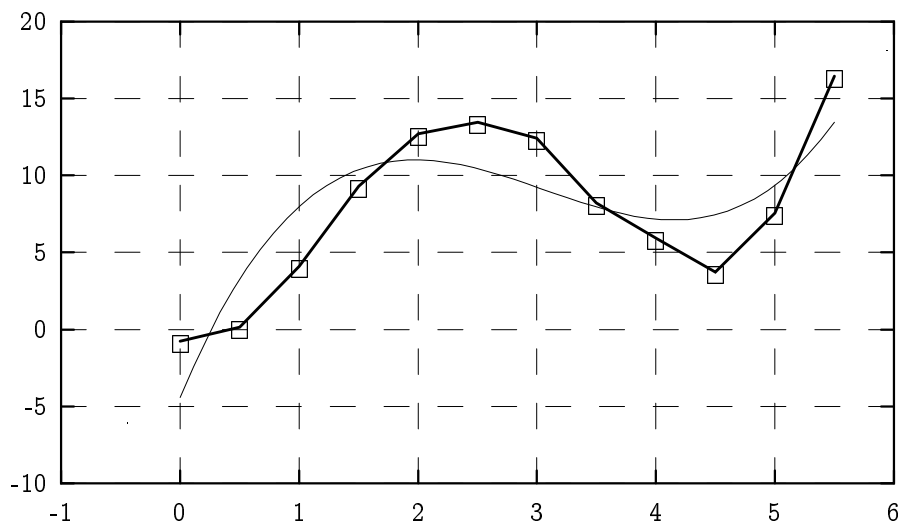
```

Tulostuksesta:

Coefficient c=

0.75599576 -6.9448571 18.579934 -4.4081638

Format:  $c[0]x^p + c[1]x^{p-1} + \dots + c[p]$  ;  $p = c.size() - 1$



Fri Jan 18 10:01:27 2002

**2.23. PNS-sovitus annetuihin funktioihin.** Edelläoleva mypfit2.cpp voidaan hyvin helposti muuntaa niin, että polynomifunktiot korvataan esimerkiksi trigonometrisilla tai muilla sopivilla funktiolla. Seuraavaksi katsomme miten tämä tapahtuu. Lähtökohtana meillä on annettu funktiojoukko, esimerkiksi  $\{g_j(x)\}, j = 0, \dots, p$ , ja data  $(x_i, y_i), i = 0, \dots, m, m \geq p$ . Etsimme PNS-mielessä parhaiten dataan sopivaa funktiota  $\sum_{j=0}^p w[j]g_j(x)$ . Muodostamme summan

$$S(w) = \sum_{i=0}^m (y_i - w[j] * g_j(x_i))^2$$

joka halutaan minimoida kertoimia  $w[j]$  vaihtelemalla. Optimaalinen  $w$  saadaan SVDsolve:n avulla, kuten edellä nähtiin.

Tässä tehtävässä, kuten tällä kurssilla usein muulloinkin, on ongelmana mistä saadaan sopivaa dataa. Sovellamme *synteettisen datan menetelmää*, jolla tarkoitetaan seuraavaa. Kiinnitämme luvut  $m, p$  ja vektorit  $x, \tilde{w}$ , ja muodostamme  $y$ :n seuraavasti  $y[i] = \sum_{j=0}^p \tilde{w}[j]g_j(x[i]), i = 0, \dots, m$ . Lopuksi lisäämme vielä “satunnaisvirhettä”  $y$ :hyn. Näin muokattu data annetaan algoritmille syötteenä. Siinä tapauksessa, että lisätty virhe on pientä, on odotettavissa, että algoritmin tuottama arvo sovituspärametrille  $w$  on lähellä arvoa  $\tilde{w}$ .

```

/* FILE: mylsqfit.cpp begins.                                     */
/* g++ mylsqfit.cpp -L../lib -I../utils -o mylsqfit -lm -lnr */
/* This program generates some data and fits
   in the LSQ sense, a set of functions g_j, j=0,...,p,
   to this data.
   Data: x[i], y[i], i=0,...,n-1 (n>p)
   Model: f(x)=sum_{j=0}^{p}{w[j] g_j(x[i])}
   System of n-equations f(x[i]) = y[i], i=0,...,n-1
   is of the form A w= y, A[i][j]=g_j(x[i]), j=0,...,p
   and its LSQ solution is w= A^+ y where A^+ is the
   pseudoinverse (A^T A)^{-1} A^T.
   SVDsolve gives w.
*/

(.....Ks. mypfit2.....)

double g(int j, double x)

```

```

{
  if (j==0) return 1.0;
  if (j==1) return log(1.0+x*x);
  if (j==2) return 1./(1+x*x);
  if (j==3) return x;
}

void GenData(Vec_DP &x, Vec_DP &y )
{
  int ndata=x.size();
  for (int i =0;i<ndata;i++)
    {x[i]= 1.0*i;
     y[i]= (1.0*g(0,x[i])+ 2.0*g(1,x[i])
           -0.5*g(2,x[i]))*rdm(0.9,1.1) ;}
}

DP MyModel(Vec_DP c, DP x)
/* MyModel returns c[0]*g(0,x) + c[1]*g(1,x) + c[2]*g(2,x)
   p= c.size()-1 */
{ int m=c.size();
  if (m!=3)
    {cout<<"Argument error in MyModel\n"; exit(1);}
  DP s=0.0;
  for (int j=0;j<m;j++) s+= c[j]*g(j,x);
  return s;
}

void PlotPolyData(Vec_DP &x, Vec_DP &y, Vec_DP &coef)
{
  (.....Ks. mypfit2 )
  for (int i=0;i<50;i++)
    yymodel[i]= MyModel(coef,xx[i]);
  MyPlot(x,y,xx,yymodel,x,y);
}

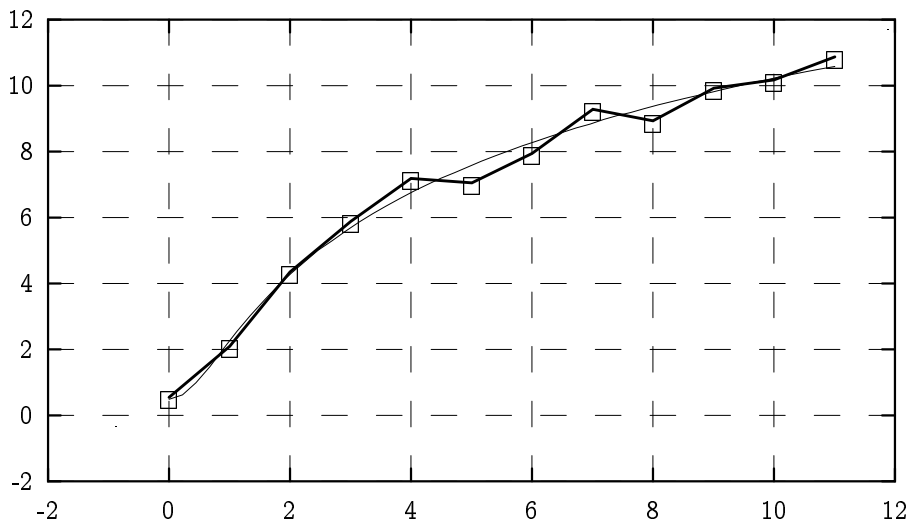
Vec_DP lsqfit(Vec_DP &x, Vec_DP &y, int nr_func )
{ int ndata=x.size();
  Vec_DP coef(nr_func);
  Mat_DP a(ndata,nr_func);
  if ((ndata<nr_func) ||(ndata!=y.size()))
    {cout<<"Argument error in polyfit\n" <<endl;
     abort();
    }
  for (int i =0;i<ndata;i++)
    for (int j=0;j<nr_func;j++)

```

```

    a[i][j]=g(j,x[i]);
SVDsolve(a,y,coef);
cout<<"Coefficient c= \n"<<coef<<endl;
cout<<
    "Format: c[0]x^p +c[1]x^{p-1} +...+ c[p] ; p= c.size()-1\n";
getchar();
return coef;
}
int main()
{
    int mdata=12;
    Vec_DP x(mdata), y(mdata), coef(3);
    GenData(x,y);
    coef=lsqfit(x,y,3);
    PlotPolyData(x,y,coef);
}
/* FILE: mylsqfit.cpp ends.                                     */

```



Fri Jan 25 10:27:48 2002

**2.24. Isotermit.** Sääkartoissa lämpötilavyöhykkeet muodostavat alueita, joiden rajat merkitään käyrillä. Käyrä, jolla lämpötila on vakio, on nimeltään *isotermi*. Lämpötilavyöhykkeen reuna muodostuu yhdestä tai useammasta isotermistä. Isotermit muodostetaan sääasemien mittaustulosten perusteella.

Tarkastamme nyt tehtävää konstruoida isotermit säähavaintojen pohjalta. Tähän liittyen haluamme "ennustaa" isotermien avulla

lämpötilan muutamassa mittauspisteiden ulkopuolisessa paikassa.

Teemme ensin matemaattisen mallin lämpötilalle. Oletamme, että lämpötilaa kuvaa toisen asteen polynomifunktio

$$f(x, y) = w_0x^2 + w_1y^2 + w_2xy + w_3x + w_4y + w_5,$$

jossa kerroinvektori  $w$  halutaan estimoida mittauksista. Ongelma on täsmälleen sama kuin edellä esitelty funktiosovitus, erityisesti se riippuu lineaarisesti  $w$ :n komponenteista ja johtaa PNS-yhtälöihin, jotka voimme ratkaista SVDsolve:lla.

Mallin toteutus voidaan jakaa seuraaviin osatehtäviin:

- (1) kartan piirto
- (2) havaintopisteiden merkitseminen karttaan
- (3) tutkimuspisteiden merkitseminen karttaan
- (4) isotermien piirto
- (5) tulosten kirjoitus tutkimuspaikkakunnilla.

Lähtökohtana olevat säähavainnot

City name	xcrd	ycrd	Temp.
Helsinki	4.6	0.6	2.6
Kotka	6.0	1.0	1.6
Joensuu	7.6	3.9	0.2
Kajaani	6.6	6.1	-0.3
Oulu	5.1	7.1	0.7
Kokkola	3.7	5.7	1.3
Uusikaupunki	2.4	1.5	2.8
Turku	2.8	0.8	0.8

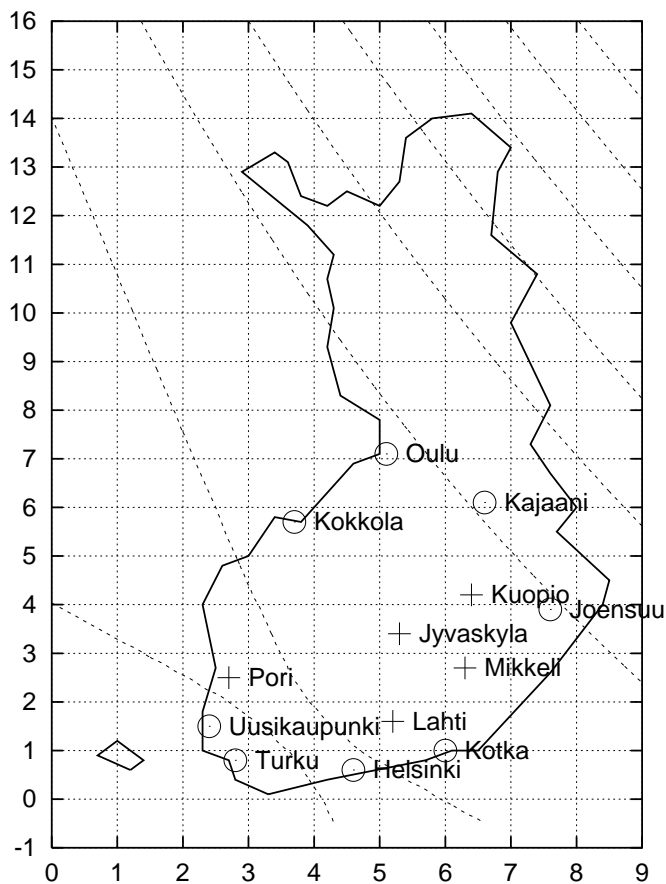
Tuloksena olevat lämpötilaennustukset:

City name	xcrd	ycrd	Real. temp.	Estim. temp.
Jyvaskyla	5.30	3.40	0.30	1.52
Mikkeli	6.30	2.70	0.40	1.32
Lahti	5.20	1.60	0.90	1.88
Pori	2.70	2.50	1.00	2.04
Kuopio	6.40	4.20	0.40	0.77

Vektorin  $w$  kertoimet:

```
coef= -0.055796284 -0.024153191 -0.095990948 0.56111105
       0.43611549 0.63546522
```





Ko. ohjelma mytemp4.cpp aputiedostoineen löytyy kurssin www-sivulta. Huomaa, että tämä malli tuottaa melko hyviä tuloksia kuten nähdään tulostuksesta. Kuitenkin mallia vastaan voidaan esittää kritiikkiä sikäli että lämpötilojen arvot vaihtelevat rajoittelulla välillä mutta mallina käytetyllä polynomifunktiolla ei ole tätä ominaisuutta.

**2.25. Komentoriviargumentit C-ohjelmassa.** Olemme jo Luvussa 1 nähneet, miten C-ohjelman syöttö ja tulostus voidaan suunnata tiedostoista tapahtuvaksi. Esim.

```
myprog <myprog.inp >myprog.out
```

ottaa syötteensä tiedostosta myprog.inp ja kirjoittaa tulostensa tiedostoon myprog.out. Nyt esittelemme joustavamman ta-

van ohjelman toiminnan ohjaamiseksi ns. *komentoriviargumenttien* avulla. Näin toimittaessa komento

```
myprog 3 4 a.dat
```

voisi ohjata ohjelman lukemaan  $3 \times 4$  matriisin tiedostosta `a.dat` tai kirjoittamaan sinne matriisin. Valmisteluna komentoriviargumenttien käytölle on pääohjelman alussa ilmaistava argumentit seuraavasti:

```
main(int argc, char *argv[ ])
```

Argumentit saavat arvoja vasta ajon aikana. Ohjelmakoodi voidaan siis tehdä riippuvaiseksi siitä millaisilla ja monellako argumentilla ohjelmaa kutsutaan. Kokonaisluvun `argc` arvo on yhtä suurempi kuin komentorivillä annettavien argumenttien lukumäärä ja merkijono `argv[0]` sisältää ohjelman nimen. Yo. esimerkissä

```
argc = 4, argv[1] = "3", argv[2] = "4", argv[3] = "a.dat" .
```

Jatkossa tulemme useasti käyttämään komentoriviargumentteja. Katsomme nyt muutamia asiaa valaisevia esimerkkejä.

**2.26. Tiedoston tulostus ruudulle.** Ensimmäinen sovellus komentoriviargumenteista on seuraava ohjelma, joka pyytää käyttäjältä ASCII tiedoston nimen ja tulostaa sitten tiedoston sisällön kuva-ruudulle.

```
/* FILE: myshow4.c begins */
/* USES:..... */
/* Displays an ascii file on the screen */
/* Unix version */
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

int main(int argc, char *argv[])
```

```

{
    char dest[20]="more ",name[20];

    if(argc>2) {
        printf("\nUsage: myshow4 a.dat or myshow4 \n");
        exit(1);
    }
    if (argc==2) {
        strcat(dest,argv[1]);
        system(dest);
        exit(1);
    }
    system("ls -x | more");
    printf("Enter the name of an ASCII file:\n");
    scanf("%s",&name);
    printf("The file name you entered: %s\n",name);
    strcpy(dest,"");
    strcat(dest,"more ");
    strcat(dest,name);
    system(dest);
    return 0;
}
/* FILE: myshow4.c end */

```

**2.27. Pieni laskin.** Toinen esimerkkinä on pieni laskinohjelma `myfeval.cpp`, jonka avulla voidaan helposti esim. muuntaa eurot markoiksi. Tämä ohjelma käyttää apunaan ohjelmaa `dofeval.cpp`, jonka se myös kääntää systeemikutsulla.

```

/* FILE: myfeval.cpp begins          */
/* g++ -Wall myfeval.cpp -o a -lm    */
/* Function evaluator                 */
/* Usage: ./myfeval "sin(0.4)"        */
/* OR:  ./myfeval "real(pow(complex<double>(0,1.0), 0.5))" */
/* Decimal point is important. Wrong result:
[vuorinen@mat-148]~/nrcpp02/democpp02 ./myfeval "6*987654321"
6*987654321 = 1630958630.00000000
[vuorinen@mat-148]~/nrcpp02/democpp02 g++ -v
gcc version egcs-2.91.66 19990314/Linux (egcs-1.1.2 release)
Effect of a decimal point:
[vuorinen@mat-148]~/nrcpp02/democpp02 ./myfeval "6*987654321.0"
6*987654321.0 = 5925925926.00000000
*/

```

```

#include <cstdlib> // Used in putmat2
#include <cstdio> // Used in putmat2
#include <ctime>
#include <string>
#include <fstream>
#include <iostream>
#include <iomanip>
#include <cmath>
#include <complex>

using namespace std;

/*
double myohj10f(void)
{double t;
t= ....;
return t;
}
*/

/* Write a function of the above format */
void write_func(char *cmdlarg )
{
    char *fname ="myfevalf.cpp";
    char *line1 ="double myfevalf( void)", *line2 ="{double t;";
        *line4 ="return t;}";
    FILE *fp;
    system("rm myfevalf.cpp");
    if ((fp =fopen(fname,"w")) ==NULL){
        printf("\nCannot open file %s", fname);
        exit(1);};
    fprintf(fp, " %s\n %s\n t= %s ;\n %s\n" ,
        line1,line2,cmdlarg,line4 );
    fclose(fp);
}

int main(int argc, char *argv[])
{
    char buff[256]="";
    if ((argc !=2))
        {printf("\nUsage: ./myfeval \"sin(0.4)\" \n");
        printf("Or: ./myfeval \""
        "real(pow(complex<double>(0,1.0), 0.5))\\"\n");
        exit(1);};
    if (argc ==2) {

```

```

    write_func(argv[1]);
    system("g++ dofeval.cpp -o dofeval -lm");
    sprintf(buff,"%s %s %s %s",
            "./dofeval ", "\"", argv[1], "\"");
    system(buff);
}
return 0;
}
/* FILE: myfeval.cpp end */

```

```

/* FILE: dofeval.cpp end */
/* g++ dofeval.cpp -o dofeval -lm */

```

```

#include <cstdlib> // Used in putmat2
#include <cstdio> // Used in putmat2
#include <ctime>
#include <string>
#include <fstream>
#include <iostream>
#include <iomanip>
#include <cmath>
#include <complex>
#include "myfevalf.cpp"

```

```

int main( int argc, char *argv[])
{
    double t=myfevalf();
    printf("%s = %16.8lf \n",argv[1],t);
    return 0;
}
/* FILE: dofeval.cpp ends */

```

```

./myfeval "6*12.34 "

```

```

6*12.34 = 74.04000000

```

Ohjelman kirjoittama funktio:

```

double myfevalf( void)
{double t;
t= 6*12.34 ;
return t;}

```

**2.28. Graafinen laskin mycal.** Seuraava kolmas esimerkki laajentaa edelläolevaa ideaa. Se osoittaa, miten C-kielen ja Gnuplotin yhteistoimintaa voidaan koordinoita siten, että Gnuplot toimii graafisen laskimen tavoin. Tämä hieman monimutkaisempi esimerkki osoittaa, miten komentoriviltä annettu, C-kielen tuntemista funktioista tehty C-kielen syntaksin mukainen funktio voidaan piirtää Gnuplotin avulla. Ohjelma rakentuu ajuriohjelman `mycal.cpp` ja apuohjelman `myta.cpp` yhteispeliin ja toimii seuraavasti:

1. Annetaan komento `./mycal 'sin(x+cos(x)) +x*x'`.
2. `mycal` kirjoittaa annetun lausekkeen tiedostoon `mytaf.cpp` C++-funktioiksi, joka on `myta.cpp`:n include-tiedosto.

Esimerkkimme tapauksessa kirjoitettava tiedosto on seuraava:

```
double mytaf(double x)
{ double t=0.0;
  t= sin(x+cos(x)) +x*x ;
  return t;}
```

3. Tämän jälkeen `mycal` systeemikutsulla kääntää ohjelman `myta.cpp`, joka puolestaan kirjoittaa tiedostoon funktion arvot ja Gnuplotin komentotiedoston ja edelleen systeemikutsulla käynnistää Gnuplotin piirtämään ko. tiedostossa olevan datan.

4. Huomaa lainausmerkit annetun funktiolausekkeen ympärillä.

```
/* FILE: mycal.cpp begin                                     */
// g++ mycal.cpp -o mycal -lm
/* mycal writes a function and compiles myta.cpp
   for plotting the function */
/* USAGE: ./mycal "sin(x) +NR::bessj0(x)" */
/* OR: ./mycal "real(pow(complex<DP>(2.0*x,x),2.0))" */

/* USES: myta.cpp, gnuplot */
/* DELETES: mytaf.cpp and writes it again */
/* 1 Jan. 1999, 20 Jan. 2002 */
```

```

#include <cstring>
#include <cstdlib> // Used in putmat2
#include <cstdio> // Used in putmat2
#include <ctime>
#include <string>
#include <fstream>
#include <iostream>
#include <iomanip>
#include <cmath>

using namespace std;

void wrtfunc(int dim, const char *fexpress, const char *funame, const char *filename)
/* Example: wrtfunc(1,argv[1], "fnew123","fnew123.c");
   wrtfunc(3,argv[2], "fnew321","fnew321.cpp");
*/
{
    FILE *fp;
    const char *fname =filename;
    const char *line2 ="{ double t=0.0;"; *line4 ="return (double) t;}";
    const char *line6 ="{double t;";
    char buff[80], line1[80], line5[80];
    strcpy(buff,"");
    strcat(buff,"rm -f ");
    strcat(buff,filename);
    system(buff);
    strcpy(buff,"");
    strcat(buff,"double ");
    strcat(buff," ");
    strcat(buff,funame);
    strcat(buff,"(double x[])");
    strcpy(line1,"");
    strcpy(line1,buff);
    strcpy(line5,"");
    strcpy(line5, "double ");
    strcat(line5,funame);
    strcat(line5, "( double x)");
    fp= fopen(fname, "w");
    if (fp==NULL){printf("File error mycal.cpp"); exit(1);}
    if (dim >1)
    {
        fprintf(fp,"%s\n%s\n",line1, line2);
    }
}

```

```

        fprintf(fp,"t= %s ;\n%s\n", fexpress,line4);
    }
else
    {
        fprintf(fp,"%s\n%s\n",line5,line6);
        fprintf(fp,"t= %s ;\n%s\n", fexpress,line4);
    }
fclose(fp);
}

int main(int argc, char *argv[])
{
    if (argc == 1)
        {printf("\nUsage: ./mycal \"sin(x)+NR::bessj0(x)\" \n");
        exit(1);}
    if (argc >=2) {
        wrtfunc(1,argv[1],"mytaf","mytaf.cpp");
        system("g++ myta.cpp -L../lib/ -I../utils -o myta -lm -lnr");
        system("./myta"); }
    return 0;
}
/* FILE: mycal.cpp end */

/* FILE: myta.cpp begin */
/* This is a slave program used by mycal.cpp */
/* USES: gnuplot */
/* DELETES: mytaf.c and writes it again */
/* Writes z.dat */
/* Tested for g++ 2002-01-20 */

#include <cstdlib> // Used in putmat2
#include <cstdio> // Used in putmat2
#include <ctime>
#include <string>
#include <fstream>
#include <iostream>
#include <iomanip>
using namespace std;

#include "nr.h"
#include "matut102.h"
#include "mytaf.cpp"

void tabula()

```



```

{
extern double mytaf(double);
double x, dx, x1, x2,y, tmp;
int i;
FILE *fp;
const char *fname, *name="z.dat";
printf("Enter x1: \n");
scanf("%lf", &x1);
printf("Enter x2: \n");
scanf("%lf", &x2);
tmp=x1;  if (x2 < x1) {x1=x2; x2=tmp;}
printf("Enter dx: \n");
scanf("%lf", &dx);
if (dx < 0) dx =-dx;
if ( dx > fabs(0.5*(x2-x1))) dx=(x2-x1)*0.05;
fname=getfname(name);
fp =fopen(fname,"w");
if (fp==NULL) {printf("File error in myta"); exit(1); }
x=x1;
do {
    y=mytaf(x);
    fprintf(fp,"%12.6e  %12.6e \n",x,y);
    x+=dx;
} while (x <x2);
fprintf(fp,"%12.6e  %12.6e \n",x2,mytaf(x2));
printf("Values written in file %s \n",fname);
fclose(fp);
system("rm gnuplot.cmd");
name="gnuplot.cmd";
if ((fp =fopen(name,"w")) ==NULL){
    printf("\nCannot open file %s", name);
    exit(1);};
fprintf(fp,"set grid\nset timestamp\n");
fprintf(fp,"plot \"%s\" with lines \npause -1\n",fname);
fprintf(fp,"set terminal postscript\nset output \"myta.ps\"\nreplot\n");
fclose(fp);
system("gnuplot gnuplot.cmd");
}
int main()
{
    tabula();
    return 0;
}
/* FILE: myta.cpp end */

```

Ohjelma `mycal.cpp` käännetään käskyllä, joka ilmenee alussa olevista kommentteista. Huomaa, että `mycal.cpp` kääntää systeemi-kutsulla ohjelman `myta.cpp`.

**2.29. 2D graafinen laskin.** Seuraavassa esimerkissä teemme pinnan-piirtoon soveltuvan muunnelman `mycal2d.cpp` ohjelmasta `mycal.cpp`. Käännös:

```
g++ mycal2d.c -o mycalc2d -lm .
```

Komento: `./mycal2d 'exp(-x*x-y*y)'` ajaa ohjelman.

```
/* FILE: mycal2d.cpp begin */
/* g++ mycal2d.cpp -o mycal2d -lm */
/* Example: ./mycalc2d "sin(x*y) +NR::gammln(x*y)"
   if no blank spaces, " " may be omitted */
/* ./mycal2d "real(NR::hypgeo(complex<double>(0.5,0),
   complex<double>(0.5,0),
   complex<double>(1.2,0),
   complex<double>(x,y)))" */
/* USES: mytas.cpp, gnuplot */
/* DELETES: mytasf.cpp and writes it again */
/* Tested only for g */
/* gcc: 1-Jan- 1999 g++: 1-Jan-2002 */

#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

void write_func(char *cmdlarg)
{
    const char *fname = "mytasf.cpp";
    const char *line1 = "double mytasf(double x,double y)";
    const char *line2 = "{ double t=0.0;";
    const char *line4 = "return t;}";
    FILE *fp;

    system("rm -f mytasf.cpp");
    if ((fp = fopen(fname, "w")) == NULL)
    {
        printf("\nCannot open file %s", fname);
    }
}
```

```

        exit(1);
    };
    fprintf(fp, "%s\n%s\n", line1, line2);
    fprintf(fp, "t= %s ;\n", cmdlarg);
    fprintf(fp, "%s\n", line4);
    fclose(fp);
}

int main(int argc, char *argv[])
{
    if (argc == 1)
    {
        printf("\nUsage: ./mycal2d \"sin(x*y)+NR::gammln(x*y)\" \" \n");
        exit(1);
    };
    if (argc >= 2)
    {
        write_func(argv[1]);
        system("g++ mytas.cpp -L../lib -I../utils -o mytas -lm -lnr");
        system("./mytas");
    }
    return 0;
}
/* FILE: mycal2d.cpp end */

/* FILE: mytas.cpp begin */
/* USES: gnuplot */
/* DELETES: mytasf.cpp and writes it again */

#include <cstdlib> // Used in putmat2
#include <cstdio> // Used in putmat2
#include <ctime>
#include <string>
#include <fstream>
#include <iostream>
#include <iomanip>
using namespace std;

#include "nr.h"

extern double mytasf(double, double);

#include "mytasf.cpp"

```

```

void tabula()
{
    extern double mytasf(double, double);
    double x, y, fxy, dx, dy, x1, x2, y1, y2, tmp;
    int i;
    FILE *fp;
    const char *fname, *name = "mytas.dat";
    printf("Enter x-interval (x1 x2):\n");
    scanf("%lf%lf", &x1, &x2);
    printf("Enter dx:\n");
    scanf("%lf", &dx);
    printf("Enter y-interval (y1 y2):\n");
    scanf("%lf%lf", &y1, &y2);
    printf("Enter dy: \n");
    scanf("%lf", &dy);

    /* Checks begin */
    tmp = x1;
    if (x2 < x1)
    {
        x1 = x2;
        x2 = tmp;
    }
    tmp = y1;
    if (y2 < y1)
    {
        y1 = y2;
        y2 = tmp;
    }
    if (dx < 0)
        dx = -dx;
    if (dy < 0)
        dy = -dy;
    if (dx > fabs(0.5 * (x2 - x1)))
        dx = (x2 - x1) * 0.05;
    if (dy > fabs(0.5 * (y2 - y1)))
        dy = (y2 - y1) * 0.05;
    /* Checks end */

    fname = name;
    fp = fopen(fname, "w");
    if (fp==NULL) { printf("File error mytas.cpp\n"); exit(1);}
    x = x1;
    y = y1;
    do

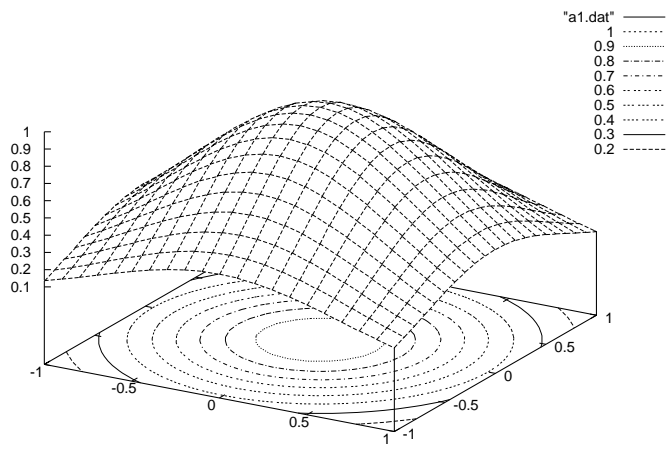
```

```

{
  do
  {
    fxy = mytasf(x, y);
    fprintf(fp, "%12.6e %12.6e %12.6e\n", x, y, fxy);
    x += dx;
  }
  while (x < x2 + 1e-10);
  x = x1;
  fprintf(fp, "\n");
  y += dy;
}
while (y < y2 + 1e-10);
printf("Values written in file %s \n", fname);
fclose(fp);
system("rm -f gnuplot.cmd");
name = "gnuplot.cmd";
fp = fopen(name, "w");
if (fp==NULL) { printf("File error mytas.cpp\n"); exit(1);}
fprintf(fp, "set contour\nset hidden3d\nset parametric\n");
fprintf(fp, "splot \"%s\" with lines\npause -1\n", fname);
fclose(fp);
}

int main(void)
{
  tabula();
  system("gnuplot gnuplot.cmd");
  return 0;
}
/* FILE: mytas.cpp end */

```



### 3 INTERPOLOINTI JA EKSTRAPOLOINTI

Mittausaineistoon perustuvat arviointitehtävät johtavat tavallisesti interpolointi- ja ekstrapolointikysymyksiin. Näihin kysymyksiin johdutaan erilaisissa tekniikan, luonnontieteen ja tilastotieteen ongelmissa. Laaja sovellusalue on myös teollinen muotoilu ja tietokonegrafiikka, jonka haasteet ovat antaneet pontta alan voimakkaalle kehitykselle viime vuosina. Taylorin kaavan nojalla tiedämme, että polynomeilla voidaan approksimoida funktiota yhdessä pisteessä. Näin on luonnollista lähteä liikkeelle polynomeista ja käyttää niitä funktion approksimointiin tarkasteltavalla välillä. Muita, usein parempia, mahdollisuuksia ovat trigonometriset approksimaatiot, rationaalifunktiot tai splinit, palapolynomit.

**Perustehtävä.** Olkoon  $f$  tuntematon funktio, jonka arvot  $y_1, \dots, y_N$  tunnetaan vain pisteissä  $x_1 < x_2 < \dots < x_N$ . Etsittävä arvio  $f(x)$ :lle kun  $x$  annettu.

Tehtävä on nimeltään *interpolointi* jos  $x_1 \leq x \leq x_N$ , muuten *ekstrapolointi*. Hyvä esitys aiheesta on teoksessa Stoer-Bulirsch [SB].

**Ratkaisuidea.** Oletetaan, että  $f$  on hyvin approksimoitavissa tunnettua tyyppiä olevilla funktioilla, joissa on  $N$  tuntematonta parametria  $a_1, \dots, a_N$ . (Usein  $f$  riippuu  $a_i$ :sta lineaarisesti). Saadaan  $N$ :n tuntemattoman yhtälöryhmä, jossa  $N$  yhtälöä ja josta tuntemattomat halutaan ratkaista.

Interpoloinnissa esiintyviä funktiotyyppejä:

-polynomi  $a_2x^2 + a_1x + a_0$

-rationaalifunktio  $\frac{a_1x+a_0}{c_1x+c_0}$

-splini eli ”palapolynomi”ts. kullakin välillä  $[x_i, x_{i+1}]$  interpolointifunktio on polynomifunktion rajoittuma tälle välille ja polynomien aste pysyy samana siirryttäessä välistä toiseen, mutta kertoimet voivat muuttua. Kahden polynomien liitospisteessä  $x_i$  mah-

dollisesti ehtoja derivaatan jatkuvuudelle

- eksponenttifunktioiden summa  $a_2e^{-x} + a_1e^{-2x} + a_0$

Perustehtävän ratkaisussa on periaatteessa kaksi ratkaisuvaihetta:

- 1) Sovitetaan interpolointifunktio annettuihin datapisteisiin.
- 2) Määrätään funktion arvo annetussa pisteessä  $x$ .

Vaativuusluokka on  $O(N^3)$  ( $N \times N$  matriisin kääntö). Käytännössä toisin:

- a) Valitaan naapuripiste  $x_i, y_i$
- b) Huomioidaan muiden pisteiden vaikutus laskukaaviolla.

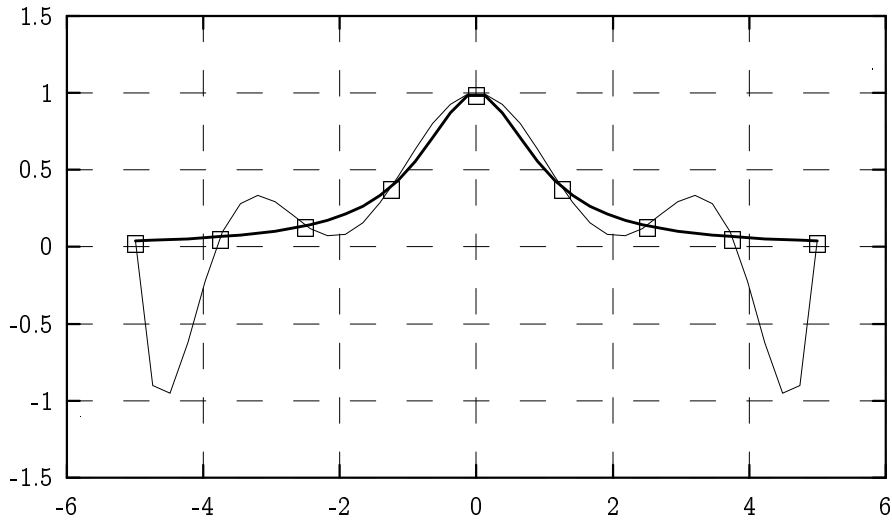
Vaativuusluokka  $O(N^2)$ , siis tehokkaampi kuin yllä.

**Rungen ilmiö.** Polynomi-interpoloinnissa ei datapisteiden lisääminen ja samalla polynomin asteen kasvattaminen välttämättä lisää tarkkuutta. NR suosittelee 3-6 pisteen käyttöä polynomi-interpoloinnissa. Olkoot  $x_1 < x_2 < \dots < x_N$  välin  $[-5, 5] = I$  tasavälinen jako,  $f(x) = 1/(1 + x^2)$  ja  $p_N$  arvoihin  $(x_i, f(x_i))$  liittyvä interpolointipolynomi. Silloin voidaan todistaa, että pätee

$$\lim_{N \rightarrow \infty} \max_{x \in I} \{ |p_N(x) - f(x)| \} = \infty.$$

Tällaisista ikävistä yllätyksistä päästään eroon käyttämällä splinejä. (Em. yllättävä ilmiö on peräisin Runge'n töistä /1901) ks. esim. Epperson Amer. Math. Monthly 94(1987,329-341) Gregory-Young (ks. kirjallisuusluettelo). Ohjelma myrunge.cpp, ks. www-sivu demoaa ilmiötä.





Mon Feb 04 00:38:42 2002

### 3.1. Polynomi-interpolointi ja ekstrapolointi.

Lagrangen kaava antaa pisteiden  $(x_i, y_i)$ ,  $i = 1, \dots, N$ ,  $x_i < x_{i+1}$ , kautta kulkevan polynomin muodossa

$$(1) \quad P(x) = \sum_{i=1}^N \prod_{j=1, j \neq i}^N \left[ \frac{(x - x_j)}{(x_i - x_j)} y_i \right].$$

Neville'n taulu  $P(x)$ :n laskemiseksi ilman kaavaa (1):

$$\begin{array}{ccccccc}
 x_1 & y_1 = P_1 & & & & & \\
 & & P_{12} & & & & \\
 x_2 & y_2 = P_2 & & P_{123} & & & \\
 & & P_{23} & & P_{1234} & & \\
 x_3 & y_3 = P_3 & & P_{234} & & & \\
 & & & P_{34} & & & \\
 x_4 & y_4 = P_4 & & & & & 
 \end{array}$$

$P_{34}$  on pisteiden  $(x_3, y_3)$ ,  $(x_4, y_4)$  kautta käyvän lineaarifunktion arvo  $x$ :ssä.  $P_{123}$  antaa pist.  $(x_i, y_i)$ ,  $i = 1, 2, 3$ , kautta kulkevan toisen asteen polynomin arvon  $x$ :ssä jne.

Yö. kaaviossa lausekkeiden  $P_*$  välillä vallitsee ns. *Neville'n kaavan* ilmaisema rekursiivinen yhteys, joka antaa interpolaatiopoly-

nomin arvon yhdessä pisteessä  $x$  :

$$P_{i(i+1)\dots(i+m)} = \frac{(x - x_{i+m})P_{i(i+1)\dots(i+m-1)} + (x_i - x)P_{(i+1)(i+2)\dots(i+m)}}{x_i - x_{i+m}}.$$

Kaava todella antaa tytärpolynomin arvon  $x$ :ssä seuraavista syistä.

**Perustelu:** 1) Se on astetta korkeampi kuin  $P_{i(i+1)\dots(i+m-1)}$

2)  $P_{i(i+1)\dots(i+m-1)}(x_j) = y_j$ ,  $j = i, i + 1, \dots, i + m - 1$

$$P_{(i+1)\dots(i+m)}(x_j) = y_j; \quad j = i + 1, \dots, i + m$$

$\Rightarrow$  Neville'n kaavassa  $P(x_j) = y_j$  kun  $j = i, i + 1, \dots, i + m$ .

Algoritmi NR: :polint s.109.

Interpoloinnin eräs sovellus on numeerinen derivointi.

Interpoloinnin yhteydessä puhutaan joskus käänteisinterpoloinnista. Tällä tarkoitetaan "akselien vaihtoa"  $(x_i, y_i) \rightarrow (y_i, x_i)$ . Sen sovelluksena on funktion 0-kohdan etsimiseen, vrt. NR:n Luku 9.

**Esim.** Allaoleva ohjelma tuottaa astetta  $N$  olevan polynomin arvoja  $N + 1$  pisteessä (polynomin kertoimet ovat tunnettuja). Algoritmin NR: :polint ja NR: :polcoe avulla pyrimme löytämään ko. kertoimet.

```
// FILE: mypolin5.cpp begins
/* We have the values of a polynomial of degree n
with known coefficients AGL at n+1 points
x_i, y_i, i=0,...,n which we use as input data
for polynomial interpolation with NR::polint.
We also use NR::polcoe to see whether we can
recover the original coefficients from this data.
*/

/* g++ -Wall mypolin5.cpp -L../lib -I../utils -I../democpp02 -o a -lm -lnr */

#include <cstdlib> // Used in putmat2
#include <cstdio> // Used in putmat2
#include <ctime>
#include <string>
#include <fstream>
```

```

#include <iostream>
#include <iomanip>

#include <cmath>
#include "nr.h"
#include "matut102.h"
using namespace std;

double xmax=2.0;

DP myf(Vec_DP AGL,DP x)
{
    int n=AGL.size();
    double s=AGL[n-1];
    for (int j=1;j<n;j++) s=s*x+AGL[n-j-1]; // Horner's rule
    return s;
}

void Gen_XYval(Vec_DP &x, Vec_DP &y,Vec_DP AGL)
{ int p=AGL.size();
  for (int i=0;i<p;i++)
    {x[i]=((DP)i)*xmax/(1.0*(DP)p);
     y[i]=myf(AGL,x[i]);    }
}

int main()
{
    DP dy,f,x,y;
    for (int n=2;n<=7;n++)
    {
        Vec_DP xa(n+1), ya(n+1), coef(n+1), AGL(n+1);
        for (int i=0;i<n+1;i++) AGL[i]=rdm(0.5,3.0);
        cout<<"\nPolyn. interpolation using "<<n+1<<" base points ";
        cout<<"on (0, xmax):\n";
        Gen_XYval(xa,ya,AGL);
        cout<<endl<<setw(9)<<"x"<<setw(14)<<"f(x)"<<setw(17);
        cout<<"interpolated"<<setw(20)<<"error"<<endl;
        for (int i=0;i<=2*n;i++)
        {
            x=((DP)i)*xmax/(2.0*(DP)n);
            f=myf(AGL,x);
            NR::polint(xa,ya,x,y,dy);
            cout<<endl<<setw(9)<<x<<setw(14)<<f<<setw(17);
            cout<<y<<setw(20)<<f-y<<endl;
        }
    }
}

```

```

    }
    NR::polcoe(xa,ya,coef);
    cout<<"Coefficients c_1 found:\n";
        cout<<coef ;
        cout<<"Coefficients c_2 of the data:\n";
    cout<<AGL;
    for (int i=0;i<coef.size();i++) coef[i]-=AGL[i];
    cout<<"norm(c_1-c_2)= "<<vnormp(coef,2.0)<<endl;
    // getchar();
}
return 0;
}
// FILE: mypolin5.cpp ends

```

## Ohjelman tulostusta:

Polyn. interpolation using 3 base points on (0, xmax):

x	f(x)	interpolated	error
0	2.60047	2.60047	0
0.5	3.95788	3.95788	0
1	6.54417	6.54417	-8.88178e-16
1.5	10.3593	10.3593	1.77636e-15
2	15.4034	15.4034	7.10543e-15

```

Coefficients c_1 found:
    2.6004693 1.4859573 2.4577481
Coefficients c_2 of the data:
    2.6004693 1.4859573 2.4577481
norm(c_1-c_2)= 1.7901808e-15

```

.....

Johtopäätöksenä ohjelman tulosteesta on, että polynomien oikeat kertoimet löytyivät (samat kuin myf:ssä).

**3.2. Interpolointi ja ekstrapolointi rationaalifunktioilla.** Rationaalifunktio  $R_{i(i+1)\dots(i+m)}$  on muotoa

$$R_{i(i+1)\dots(i+m)} = \frac{P_\mu(x)}{Q_\nu(x)} = \frac{p_0 + p_1x + \dots + p_\mu x^\mu}{q_0 + q_1x + \dots + q_\nu x^\nu}$$

$$m + 1 = \mu + \nu + 1.$$

Rationaali-interpolointi soveltuu myös tilanteisiin, jolloin polynomi-interpolointi ei käy (nimittäjän nollakohdat  $\rightarrow R$ :n  $\infty$ -kohdat). Algoritmi NR::ratint käyttää hyväksi Neville'n kaavan tapaista palautuskaavaa.

**3.3. Kuutiosplini-interpolointi.** Olkoon  $x_1 < x_2 < \dots < x_N$  ja vastaavat funktion arvot  $y_1, \dots, y_N$ . Halutaan approksimoida funktiota välillä  $[x_1, x_N]$  käyttäen approksimanttina funktiota  $g$ , jolla

(a)  $g|_{[x_i, x_{i+1}]}$  on kuutiopolynomi (kertoimet tavallisesti riippuvat  $i$ :stä)

(b)  $\lim_{x \rightarrow x_i^-} g''(x) = \lim_{x \rightarrow x_i^+} g''(x)$  kaikissa liitospisteissä  $x_i, i, 2 \leq i \leq N - 1$ .

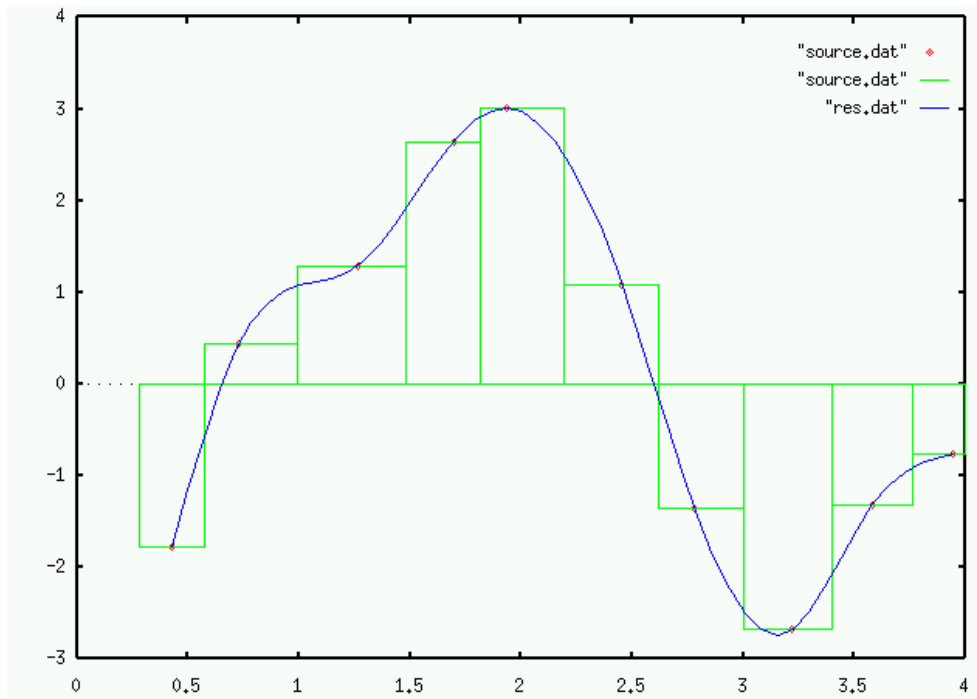
(c) Kuten (b), mutta  $g'$ :lle.

Yhtälöt (b) johtavat  $N - 2$ :een yhtälöön  $N$ :lle tuntemattomalle  $g''(x_i), i = 1, \dots, N$ . Jotta tuntemattomat määräytyisivät 1-käsitt. on asetettava lisäehdot päätepisteissä  $x_1$  ja  $x_N$ . Tällöin on kaksi mahdollisuutta:

(a) aset.  $g''(x_1) = 0$  tai  $g''(x_N) = 0$  tai molemmat  $= 0$  (ns. *luonnollinen splini*)

(b) aset.  $g'(x_1)$  tai  $g'(x_N)$  tai molemmat halutun suuruiseksi  
Tällaista funktiota  $g$  kutsutaan *kuutiospliniksi*.

Graafisesti splini-interpoloinnin vaikutusta voidaan havainnollistaa oheisella kuvalla, jossa laatikot ja käyrällä olevat pisteet kuvaavat alkuperäistä dataa. Splini-interpolointi tuottaa pisteiden kautta kulkevan sileän käyrän.



Algoritmin NR::spline toiminta

**input:**  $x_1 < x_2 < \dots < x_N$ ,  $y_1, \dots, y_N$  eli  $x_a$ ,  $y_a$  ja lisäksi  $g'$ :n arvot pist.  $x_1, x_N$  : yp1, ypn Asettamalla derivaattojen arvot yp1, ypn luvuksi 1E30 saadaan luonnollinen splini.

**output**  $g''$ :n arvot pist.  $x_1, \dots, x_N$  : y2a

Algoritmin NR::splint toiminta

**input:**  $x$ ,  $x_i, y_i, y''$  eli  $x$ ,  $x_a$ ,  $y_a$ ,  $y2a$

**output:** splini-approksimaation arvo pist.  $x$

**Reunaehto 1.derivaatan avulla.** Seuraava ohjelma osoittaa, että siinä annetut ehdot ensimmäiselle derivaatalle ko. välin päätepisteissä toteutuvat, vrt. kuvio.

```

/* FILE: myspbr2.cpp */
/* g++ -Wall myspbr2.cpp -L../lib -I../gnuplot02 -o a -lm -lnr */

/* This program carries out spline interpolation to data.
   If the parameter BRYCOND ==1 the difference quotient
   of the data at the end points gives bry conditions
   for the derivative otherwise constant values specified

```

```

    in SplineWithBryVal are used.
*/

#include <cstdlib> // Used in putmat2
#include <cstdio> // Used in putmat2
#include <ctime>
#include <string>
#include <fstream>
#include <iostream>
#include <iomanip>
using namespace std;

#define PRINT 1
#include "nr.h"
#include "matut102.h"
#include "plot.h"

double pwlin(Vec_DP &dat, double x)
{
    double x1=dat[1],x2=dat[2],y1=dat[3], y2=dat[4],
           c1=dat[5],c2=dat[6], leikpis;
    if (dat[1] >dat[2]-1.0e-3) {x1=dat[2]; x2=dat[1]+1e-3;}
    if (fabs(c1-c2) <1.0e-3){
        if (x <0.5*(x1+x2)) return y1+c1*(x-x1);
        if (x >0.5*(x1+x2)) return y2+c2*(x-x2);}
    else {
        leikpis =(y1-y2+c2*x2-c1*x1)/(c2-c1);
        if (x <leikpis) return y1+c1*(x-x1);
        if (x >leikpis) return y2+c2*(x-x2);
    } /* pwlin has slopes given by boundary */
    /* values of the derivative */
    return 1.0;
}

void SplineWithBryVal(Mat_DP a,Vec_DP &xa,
                    Vec_DP &ya,Vec_DP &y2a,Vec_DP &dat, int brycond )
{
    int m=a.nrows();
    for (int j =0;j<a.nrows();j++)
        {xa[j]=a[j][0]; ya[j]=a[j][1];}
    double yp1, ypn;
    /* Set derivatives at end points */
    if (brycond == 0)
        {
            yp1=0.5;    ypn=-1.0;

```

```

    }
    else{
        yp1= (ya[2-1]-ya[1-1])/(xa[2-1]-xa[1-1]);
        ypn= (ya[m-1]-ya[m-2])/(xa[m-1]-xa[m-2]); }
        dat[1]=xa[1-1];      dat[2]=xa[m-1];
        dat[3]=ya[1-1];      dat[4]=ya[m-1];
        dat[5]=yp1; dat[6]=ypn; /* dat[0] not used! */
        NR::spline(xa,ya,yp1,ypn,y2a);
    }

void PlotDataSplineBryVal(Vec_DP &xa, Vec_DP &ya,
    Vec_DP &y2a,Vec_DP &dat, char *mydata)
{

    const char *fname = "z1.dat", *fnameb = "z2.dat";
    fnameb =getfname(fnameb);
    fname =getfname(fname);
    FILE *fp, *fpb;
    fp =fopen(fname,"w");
    fpb =fopen(fnameb,"w");
    if ((fp==NULL)|| (fpb==NULL))
        {cout<<"File error in PlotDataSplineBryVal"<<endl;
        abort();}
    int N=xa.size();
    if (N<40) N=40;
    double x,y;
    for (int j=0;j<=N-1;j++){
        x=xa[1-1]+(xa[xa.size()-1]-xa[1-1])*(j)/(N-1.0);
        NR::splint(xa,ya,y2a,x, y);
        fprintf(fp," %12.5lf %12.6lf\n",x,y);
        y=pwlin(dat,x);
        fprintf(fpb," %12.5lf %12.6lf\n",x,y);
    }
    fclose(fp);      fclose(fpb);
    plot(fname,"r-3",fnameb,"b-2",mydata,"ks4",NULL);
}

int main(int argc, char *argv[])
{
    int brycond;
    char *fname=argv[1];
    if (argc !=2){
        printf("Usage: ./myspbr fname1.dat  \n");
        exit(1);}
    Mat_DP a(1,1);

```

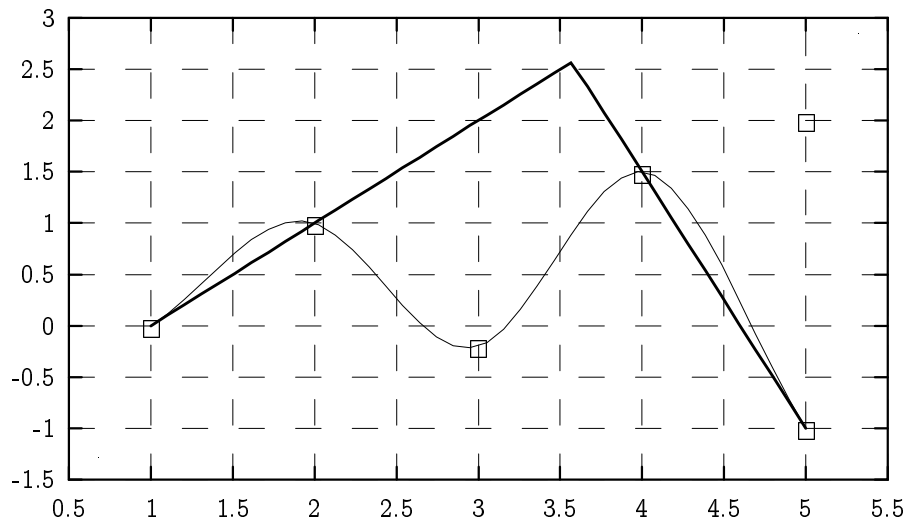


```

a=getmat(fname);
showmat2(a," %10.4lf");
int m=a.nrows();
Vec_DP xa(m),ya(m), y2a(m), dat(7);
for (int j=0;j<2;j++)
  { brycond=j;
    SplineWithBryVal(a,xa,ya,y2a,dat,brycond);
    PlotDataSplineBryVal(xa,ya,y2a,dat,fname);
  }
}
/* FILE: myspbr2.cpp */

/* Tested with: a.dat:
5 2
1 0
2 1
3 -0.2
4 1.5
5 -1
*/

```



Mon Feb 04 00:51:57 2002

Fysikaalinen tulkinta : Pisteiden  $(x_i, y_i)$  kautta asetetun kimmoisan sauvan muoto on splini (sauvan "energia"  $\int_{x_0}^{x_n} g''(s)ds$  pyrkii minimiinsä ko. rajoite-ehdoin).

```
// FILE: mysplint.cpp begin
```

```

/* g++ -Wall mysplint.cpp -L../lib -I../utils -I../gnuplot02 -I../democpp02 -o a -lm -l...

#include <cstdlib> // Used in putmat2
#include <cstdio> // Used in putmat2
#include <ctime>
#include <string>
#include <fstream>
#include <iostream>
#include <iomanip>
using namespace std;

#include <cmath>

#define PRINT 1
#include "nr.h"

#include "matutl02.h"
#include "gnuplt1.h"

using namespace std;

#define N 20

int n_global;
DP yp1, ypn;
Vec_DP xa(N), ya(N), y2a(N);
DP a[N+1], b[N+1];
int M;

DP myfunc(DP x)
// myfunc gives the values of the sine sums
{
    DP s=0.0; int j;
    for (j=1;j<=M; j++)
        s-=a[j]*cos(b[j]*x)/b[j];
    return s;
}

DP dmyfunc(DP x)
// dmyfunc2 gives the derivative function of myfunc
{
    DP s=0.0; int j;
    for (j=1;j<=M; j++)
        s+=a[j]*sin(b[j]*x);
}

```

```

    return s;
}

DP fsint(DP x)
{
    DP y;
    NR::splint(xa,ya, y2a, x, y);
    return y;
}

DP deviat(DP x)
{
    DP y;
    NR::splint(xa,ya, y2a, x, y);
    return y-myfunc(x);
}

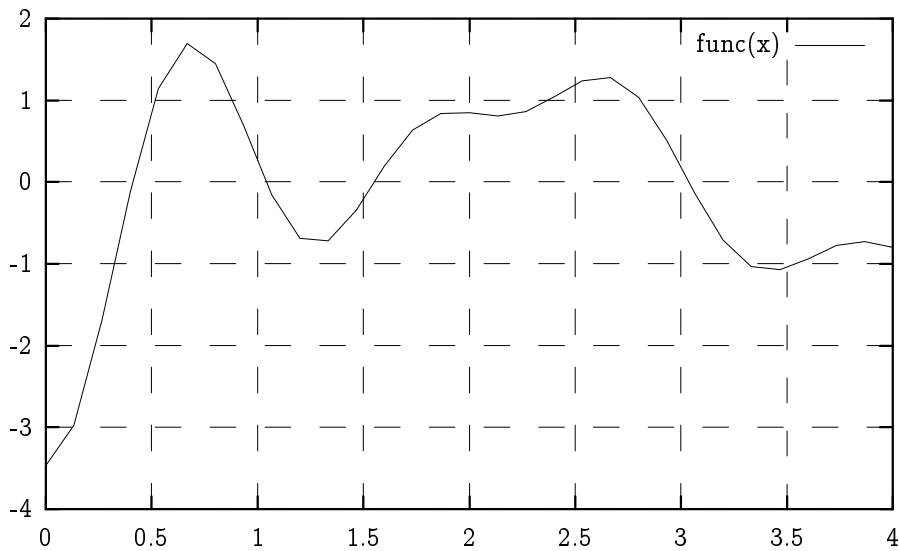
int main()
{
    int i;
    DP x,y;
    M=7;                // Generate random coefficients
    init_srand();
    for (i=1;i<=M;i++)
    { // rdm is from matutl
        a[i]= rdm(0.0,3.0);
        b[i]=rdm(0.3,6.0);
    }
    // Generate array for interpolation
    for (i=1;i<=20;i++)
    {
        xa[i-1]=i*4.0/N;
        ya[i-1]=myfunc(xa[i-1]);
    }
    // calculate 2nd derivative with spline
    yp1=dmyfunc(xa[0]);
    ypn=dmyfunc(xa[N-1]);
    NR::spline(xa,ya,yp1,ypn,y2a);
    // test result
    cout<<setw(23)<<"spline"<<setw(17)<<"actual";
    cout<<setw(17)<<"error"<<endl;
    cout<<setw(11)<<"angle"<<setw(15)<<"value";
    cout<<setw(17)<<"value"<<endl;
    for (i=0;i<=15;i++) {
        x=4.0*i/15;

```

```

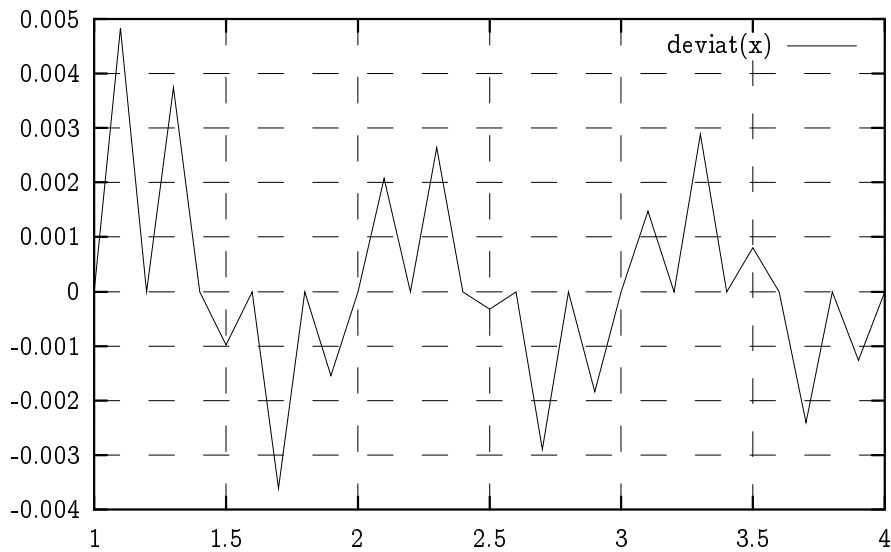
NR::splint(xa,ya, y2a, x, y);
cout.precision(2);
cout<<setw(10)<<x;
cout.precision(6);
cout<<setw(17)<<myfunc(x)<<setw(17);
cout<<y<<setw(17)<<y-myfunc(x)<<endl;
}
gnuplt1(myfunc,"myfunc(x)",0, NULL);
gnuplt1(deviat,"deviat(x)",0, NULL);
return 0;
}
// FILE: mysplint.cpp ends

```



Yllä on funktion myfunc kuvaaja. Alla osa ohjelman tulostuksesta sekä vielä virhefunktion deviat kuvaaja.

	spline	actual	error
angle	value	value	
0	-3.87375	-3.79202	0.0817389
0.27	-2.6557	-2.65475	0.000948754
0.53	-0.0229636	-0.0236019	-0.000638306
0.8	1.98268	1.98268	0
1.1	2.242	2.24182	-0.000173306
1.3	1.3447	1.34536	0.000658708
1.6	0.548351	0.548351	0
.....			



**3.4. Järjestetyn vektorin läpikäynti.** Oletetaan  $x_1 < x_2 < \dots < x_N$  ja  $x \in \mathbb{R}$ . Asetetaan  $x_0 = -\infty$  ja  $x_{N+1} = \infty$ .

**Tehtävä:** Etsi  $j$  s.e.  $x_j \leq x < x_{j+1}$ . Silloin  $0 \leq j \leq N$  ja jos  $j = 0$  niin  $x < x_1$  ja jos  $j = N$  niin  $x \geq x_N$ .

Algoritmi `NR::locate` s.118 toteuttaa tämän.

Algoritmi `NR::hunt` toimii kuten `NR::locate`, mutta voidaan antaa alkuarvaus indeksille  $j$ .

**3.5. Interpolointipolynomin kertoimet.** Interpoloinnissa ei yleensä tarvita interpolointipolynomin kertoimia, halutaan vain laskea `int.pol`:n arvo annetussa pisteessä  $x$  (huomaa esitys on tavallisesti joko  $y = c_0 + c_1x + \dots + c_Nx^N$  tai  $y = c_0x^N + c_1x^{N-1} + \dots + c_N$ .)

**NR:n suositus:** Jos tarvitaan vain interpol. polyn. arvoja, ei kertoimia ole syytä määrittää (ikäänkuin polynomin arvon laskemisen alkuvaiheena) vaan tulee käyttää aikaisemmin tässä luvussa esiteltyjä menetelmiä.

**Perustelu:** Kertoimien määrittäminen johtaa yhtälöryhmään

$$\begin{bmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^{N-1} \\ 1 & x_2 & x_2^2 & \cdots & x_2^{N-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_N & x_N^2 & \cdots & x_N^{N-1} \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_N \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}$$

jonka matriisi (ns. *Vandermonden matriisi*) on häiriöaltis.

Jos  $c_i$ :t ratkaistaan tästä ja sen jälkeen lasketaan polynomin arvoja, ei yleensä saada samaa (oikeata) vastausta kuin NR::polint:sta.

Algoritmi NR::polcoe s.121 laskee kertoimet, NR::polcof s.121 samoin, tosin eri menetelmällä. Ohjelmassa mypolin5.cpp oli esimerkki NR::polcoe:n käytöstä.

### 3.6. 2-ulotteinen interpolointi. Olkoot

$$x1a[j], j=1, \dots, m, x1a[j] < x1a[j+1]$$

$$x2a[k], k=1, \dots, n, x2a[k] < x2a[k+1]$$

ja vastaavasti

$$ya[j][k] = y(x1a[j], x2a[k])$$

taulukoituja funktion arvoja. Haluamme määrittää funktion arvon interpoloinnilla taulukoitujen pisteiden ulkopuolella, pisteessä  $(x_1, x_2)$ . Valitaan  $j$  ja  $k$  siten, että

$$x1a[j] \leq x_1 \leq x1a[j+1], x2a[k] \leq x_2 \leq x2a[k+1].$$

Merkitään

$$y_1 = ya[j][k], y_2 = ya[j+1][k],$$

$$y_3 = ya[j+1][k+1], y_4 = ya[j][k+1],$$

ja

$$t = (x_1 - x1a[j]) / (x1a[j+1] - x1a[j]) \in [0, 1]$$

$$u = (x_2 - x2a[k]) / (x2a[k+1] - x2a[k]) \in [0, 1].$$

Bilineaarinen interpolointi-kaava  $y(x_1, x_2)$ :lle on

$$y(x_1, x_2) = (1 - t)(1 - u)y_1 + t(1 - u)y_2 + tuy_3 + (1 - t)uy_4.$$

Parannusmahdollisuuksia:

- 1) korvataan lineaarinen funktio polynomilla (korkeampien osittaisderivaattojen mahdollinen epäjatkuvuus)
- 2) pyritään "sileään" interpolointifunktioon

### 1. Polynomi-interpolointi

Idea:  $-M - 1$ :n kertal. interpolointi  $x_1$ -suunnassa

$-N - 1$ :n kertal. interp.  $x_2$ -suunnassa

**Toteutus:** Etsitään ensin  $M \times N$  taulukkopistettä, annetun pisteen ympäriltä, ( $M$   $x_1$ -suun. ja  $N$   $x_2$ -suunn.) Tehdään  $M$  interpol.  $x_2$  suunnassa, saadaan funktion arvot pist.  $(x_{1a}[j], x_2)$ ,  $j=1, \dots, m$ . Näiden avulla tehdään vielä yksi interpol.  $x_1$ -suuntaan, saadaan funktion arvo etsityssä pisteessä.

### 2. Bikuubinen interpolointi

Bikuubisessa (bicubic) interpoloinnissa kutakin solmupistettä kohden annetaan

$$(*) \quad y, \partial y / \partial x_1, \partial y / \partial x_2, \partial^2 y / \partial x_1 \partial x_2.$$

Funktion arvojen lisäksi tarvitaan siis myös em. derivaatat.

Oheisena tilannetta havainnollistava kuva 3.6.1 NR:n sivulta 124.

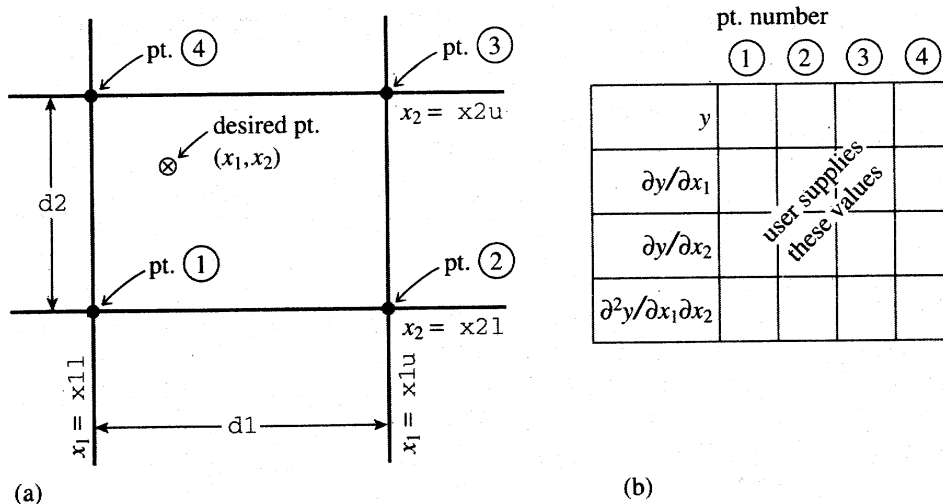


Figure 3.6.1. (a) Labeling of points used in the two-dimensional interpolation routines `bcuint` and `bcucof`. (b) For each of the four points in (a), the user supplies one function value, two first derivatives, and one cross-derivative, a total of 16 numbers.

Bikuubisen interpoloinnin ominaisuuksia:

- (a) interpolointi antaa oikein takaisin em. arvot solmupisteissä
- (b) interpoloinnin antama funktio on em. derivaattoineen jatkuva myös siirryttäessä ”ruudusta” toiseen.

Bikuubisen interpolointipolynomin arvon laskeminen ruudun sisäpisteessä tapahtuu seuraavasti:

1) Annetaan kussakin ruudun kulmapisteessä 4 funktion tai derivaatan arvoa kuten yllä kaavassa (\*) (siis yhteensä 16 arvoa).

2) Olk.  $t, u \in [0, 1]$  kuten bilin. interpoloinnissa.

Silloin bikuubisen interpoloinnin kaavat ovat

$$y(x_1, x_2) = \sum_{i=1}^4 \sum_{j=1}^4 c_{ij} t^{i-1} u^{j-1}$$

$$y'_1(x_1, x_2) = \sum_{i=1}^4 \sum_{j=1}^4 (i-1) c_{ij} t^{i-2} u^{j-1}$$

$$y'_2(x_1, x_2) = \sum_{i=1}^4 \sum_{j=1}^4 (j-1) c_{ij} t^{i-1} u^{j-2}$$

$$y''_{12}(x_1, x_2) = \sum_{i=1}^4 \sum_{j=1}^4 (i-1)(j-1) c_{ij} t^{i-2} u^{j-2}$$

vakiot  $c_{ij}$  saadaan algoritmista NR::bcucof (laskennassa tarvitaan myös ruudun pituus D1 ja korkeus D2).

3) Varsinaisen interpoloinnin tekee NR::bcuint.

```
// FILE: mybcu.cpp begins
#include <iostream>
#include <iomanip>
#include <cmath>
#include "nr.h"
#include "gnusurf.h"

#define SCALE 1

using namespace std;
```



```

// Driver for routine bcuint

const DP xx_d[4]={0.0,2.0,2.0,0.0};
const DP yy_d[4]={0.0,0.0,2.0,2.0};

DP f(DP x, DP y)
{
    return tan(cos(x)+sin(x))*sin(x+y);
}

DP ff(DP u, DP v)
{
    int i;
    DP ansy,ansy1,ansy2;
    DP x1,x1l,x1u,x2,x2l,x2u,xxyy;
    Vec_DP y(4),y1(4),y12(4),y2(4);
    Vec_DP xx(xx_d,4),yy(yy_d,4);

    x1l=xx[0];
    x1u=xx[1];
    x2l=yy[0];
    x2u=yy[3];
    for (i=0;i<4;i++)
    {
        xxxy=xx[i]*yy[i];
        y[i]=xxyy*xxyy;
        y1[i]=2.0*yy[i]*xxyy;
        y2[i]=2.0*xx[i]*xxyy;
        y12[i]=4.0*xxyy;
    }
    x1=u; x2=v;
    NR::bcuint(y,y1,y2,y12,x1l,x1u,x2l,x2u,
              x1,x2,ansy,ansy1,ansy2);
    return ansy;
}

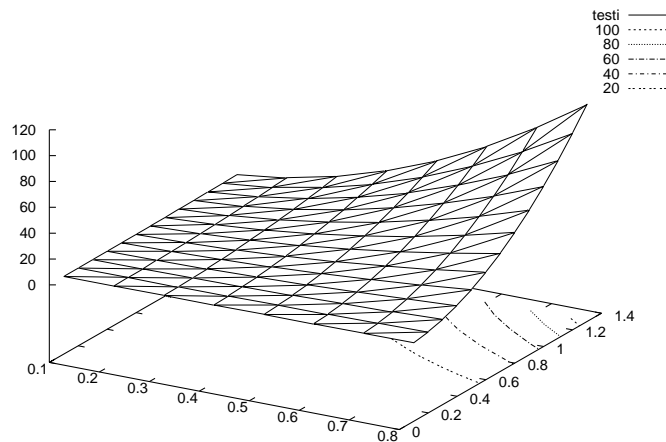
int main(void)
{
    int i;
    DP ansy,ansy1,ansy2,ey,ey1,ey2;
    DP x1,x1l,x1u,x1x2,x2,x2l,x2u,xxyy;
    Vec_DP y(4),y1(4),y12(4),y2(4);
    Vec_DP xx(xx_d,4),yy(yy_d,4);

```

```

x1l=xx[0];
x1u=xx[1];
x2l=yy[0];
x2u=yy[3];
for (i=0;i<4;i++)
{
  xxyy=xx[i]*yy[i];
  y[i]=xxyy*xxyy;
  y1[i]=2.0*yy[i]*xxyy;
  y2[i]=2.0*xx[i]*xxyy;
  y12[i]=4.0*xxyy;
}
cout << endl << setw(6) << "x1" << setw(9) << "x2";
cout << setw(8) << "y" << setw(12) << "expect";
cout << setw(7) << "y1" << setw(11) << "expect";
cout << setw(7) << "y2" << setw(11) << "expect" << endl << endl;
cout << fixed << setprecision(4);
for (i=0;i<10;i++) {
  x2=(x1=0.2*(i+1));
  NR: :bcuint(y,y1,y2,y12,x1l,x1u,x2l,x2u,x1,x2,ansy,ansy1,ansy2);
  x1x2=x1*x2;
  ey=x1x2*x1x2;
  ey1=2.0*x2*x1x2;
  ey2=2.0*x1*x1x2;
  cout << setw(8) << x1 << setw(9) << x2 << setw(9) << ansy;
  cout << setw(9) << ey << setw(9) << ansy1 << setw(9) << ey1;
  cout << setw(9) << ansy2 << setw(9) << ey2 << endl;
}
double xxx[]={0.,2.},yyy[]={0,2.};
gnusurf(ff,xxx,yyy,SCALE,0,"ff","testi");
cout << "Program ended normally." << endl;
return 0;
}
// FILE: mybcu.cpp ends

```



### 3. Muita mahdollisuuksia

Bikuubinen spline: Algoritmit NR::splie2, splin2 s. 128.

**3.7. Polynomien sovelluksia. Numeerinen derivointi.** Seuraava esimerkki valaisee hieman numeerisen derivoinnin implementoinnin mahdollisuuksia teoksen [AS, 25.3.6] mukaisesti. Lähtökohtana on tasavälisesti taulukoidut funktion arvot. Haluamme löytää approksiimaatiot derivaatan arvoille.

Arvoja on oltava vähintään kaksi kappaletta. Jos arvoja on tasan kaksi, voidaan derivaatan likiarvona käyttää erotusosamäärää  $(y_2 - y_1)/(x_2 - x_1)$ .

Tarkastamme seuraavaksi teoksen [AS], kohdan 25.3.6 mukaisesti tilannetta, kun arvoja on viisi. Olkoon  $h > 0$ ,  $p = -2, -1, 0, 1, 2$  ja taulukoidut funktion arvot  $f_p = f(x_0 + ph)$ . Silloin  $f$ :n numeerisella derivaatalla on lauseke (Abramowitz-Stegun, 25.3.6)

$$\begin{aligned}
 f'(x_0 + ph) &= [a * f_{-2} - b * f_{-1} + c * f_0 - d * f_1 + e * f_2]/h, \\
 a &= (2 * p^3 - 3 * p^2 - p + 1)/12, \\
 b &= (4 * p^3 - 3 * p^2 - 8 * p + 4)/6, \\
 c &= (2 * p^3 - 5 * p)/2, \\
 d &= (4 * p^3 + 3 * p^2 - 8 * p - 4)/6, \\
 e &= (2 * p^3 + 3 * p^2 - p - 1)/12, \quad p = -2, -1, 0, 1, 2.
 \end{aligned}$$

Havaitaan, että derivaatan lauseke pisteessä  $x_p, p = -2, -1, 0, 1, 2$  on "painotettu erotusosamäärä", jonka painokertoimet  $a, b, c, d, e$  riippuvat  $p$ :stä. Kertoimien numeeriset arvot ilmenevät seuraavasta kaaviosta:

$p$	$a$	$-b$	$c$	$-d$	$e$
-2	-25/12	24/6	-6/2	8/6	-3/12
-1	-3/12	-5/6	3/2	-3/6	1/12
0	1/12	-4/6	0/2	4/6	-1/12
1	-1/12	3/6	-3/2	5/6	3/12
2	3/12	-8/6	6/2	-24/6	25/12

Siinä tapauksessa, että taulukoituja arvoja on  $m$  kpl, ( $m \geq 5$ ), menetellään seuraavasti. Kohdassa  $x_j, 2 \leq j \leq m-2$  sovelletaan yo. kaavaa arvoille  $f_{j-2}, f_{j-1}, f_j, f_{j+1}, f_{j+2}$  tapauksessa  $p = 0$ . Kohdassa  $x_j, j = 1, 2$  käytetään arvoja  $f_1, \dots, f_5$  ja kaavaa tapauksessa  $p = -2$  tai vastaavasti  $p = -1$ . Pisteessä  $x_j, j = m-1, m$  käytetään arvoja  $f_{m-4}, \dots, f_m$  ja kaavaa tapauksessa  $p = 1$  tai vastaavasti  $p = 2$ .

```
// FILE: mynumber.cpp begins
// g++ mynumber.cpp -L../lib -I../utils -I../democpp02 -o a -lm -lnr

#include <cstdlib> // Used in putmat2
#include <cstdio> // Used in putmat2
#include <ctime>
#include <string>
#include <fstream>
#include <iostream>
#include <iomanip>
#include <cmath>

using namespace std;
#include "nr.h"
#include "matutl02.h"

#define MAXD 50
#define PI M_PI

using namespace std;
```

```

void number(const Vec_DP seq1, Vec_DP &seq2, double h, int n)
/* A numerical approximation of the derivative of a function
   f is computed, given evenly spaced function values f[a+i h],
   i = 1,...,n, stored in seq1 in this order. The values of the
   derivative Df[a+i,h], i = 1,...,n are returned in seq2. The
   computation is based on the five-point formula in
   Abramowitz-Stegun 25.3.6. It is required that n >= 5. */
{
  int i, ilow = 1, ihigh = n, p, p2, p3;
  double a, b, c, d, e;

  if ((ihigh - ilow + 1) < 5)
    fprintf(stderr, "\nToo few points for numerical differentiation.\n");
  else
    for(i = ilow; i <= ihigh; i++) {
      if (i <= ilow + 2)
        p = i - ilow - 2;
      if ((i > ilow + 2) && (i <= ihigh - 2))
        p = 0;
      if (i > ihigh - 2)
        p = i - ihigh + 2;
      p2 = p * p;
      p3 = p * p2;
      a = ((2.0 * p3) - (3.0 * p2) - p + 1.0) / 12.0;
      b = ((4.0 * p3) - (3.0 * p2) - (8.0 * p) + 4.0) / 6.0;
      c = ((2.0 * p3) - (5.0 * p)) / 2.0;
      d = ((4.0 * p3) + (3.0 * p2) - (8.0 * p) - 4.0) / 6.0;
      e = ((2.0 * p3) + (3.0 * p2) - p - 1.0) / 12.0;
      seq2[i] = (a * seq1[i - p - 2]) -
        (b * seq1[i - p - 1]) +
        (c * seq1[i - p]) -
        (d * seq1[i - p + 1]);
      seq2[i] = (seq2[i] + (e * seq1[i - p + 2])) / h;
    }
}

double f(double x)
{
  return sin(x);
}

int main()
{
  double h=1e-5, x0 =1.1;

```

```

Vec_DP y(6),dy(6);
int i;
for (i=1;i<=5;i++) y[i]= f(x0+(i-3)*h);
numder(y,dy,h,5);
printf("    x          df(x)          virhearvio \n");
//cout.setf(ios::scientific);
cout.precision(8);
for (i=1;i<=5;i++)
{
    cout<<setw(10)<<x0+(i-3)*h<<setw(12)<<dy[i];
    cout<<setw(20)<<dy[i]- cos(x0+(i-3)*h)<<endl;
}
return 0;
}
// FILE: mynumder.cpp ends

```

Ohjelman tulostus on seuraava:

x	df(x)	virhearvio
1.09998	0.45361395	-6.2296244e-11
1.09999	0.45360503	4.0718258e-12
1.1	0.45359612	8.7911403e-12
1.10001	0.45358721	-6.5398544e-13
1.10002	0.4535783	2.3879404e-11

Seuraava ohjelma numdf tarjoaa hieman joustavamman kutsutavan ja joitakin lisäominaisuuksia ohjelmaan numder verrattuna, jota numdf kutsuu. Laskemme välillä  $(a, b)$  määritellyn funktion numeeriselle toiselle derivaatalle likiarvon pisteessä  $x$ . Tarkoitusta varten laskemme ensin funktion arvot pisteissä  $x + ph$ ,  $p = -2, -1, 0, 1, 2$ , ja huolehdimme siitä, että  $a < x - 2h$  ja  $x + 2h < b$ . Sovellamme ohjelmaa numder funktion arvojen muodostamaan taulukkoon.

```

// FILE: mynumdf.cpp begins
// g++ mynumdf.cpp -L../lib -I../utils -I../democpp02 -o a -lm -lnr

#include <cstdlib> // Used in putmat2
#include <cstdio> // Used in putmat2
#include <ctime>
#include <string>
#include <fstream>

```

```

#include <iostream>
#include <iomanip>
#include<cmath>

using namespace std;
#include "nr.h"

#include "numder.cpp"

using namespace std;

double numdf(double f(double),double a,double b,double x,double h,int n)
/* A numerical approximation of the derivative of a function
   f defined on (a,b) is computed at x. The
   computation is based on the five-point formula in
   Abramowitz-Stegun 25.3.6 with step h. If x <a+2h or x>b-2h
   the step length is reduced. */
{
    int i;
    Vec_DP seq1(6),seq2(6);
    double dx =h;
    if (( x <= a) || (x>=b))
    {
        cerr<<"\n Argument error in numdf, x = "<<x<<endl;
        exit(1);
    }
    if (x-2*h < a) dx= (x-a)/3;
    if (x+2*h > b) dx= (b-x)/3;
    for (i=1;i<=5;i++) {seq1[i] = f(x+(i-3)*dx);}
    numder(seq1,seq2,dx,5);
    if (n == 2) {
        numder(seq2,seq1,dx,5);
        return( seq1[3]);
    }
    else if (n==1) return(seq2[3]);
    else return 0.0;
}

double f(double x)
{
    return sin(x);
}

double tst(double x)
{

```

```

    return (f(x) +numdf( f, x-1.0, x+1.0, x, 1E-5, 2));
}

int main()
{
    double x;
    int i;
    cout<<"          x          D(D(sin(x)))          virhearvio \n";
    // cout.setf(ios::scientific);
    cout.precision(6);
    for (i=1;i<=5;i++) {
        x=0.1*i;
        cout<<setw(10)<<x<<setw(14)<< numdf( f, x-1.0, x+1.0, x, 1E-5, 2);
        cout<<setw(18)<<tst(x)<<endl;
    }
    return 0;
}
// FILE: mynumdf.cpp ends

```

x	D(D(sin(x)))	virhearvio
0.1	-0.0998333	1.1473e-07
0.2	-0.198669	2.74021e-07
0.3	-0.29552	-6.09202e-08
0.4	-0.389419	-8.96643e-07
0.5	-0.479426	-4.26042e-07

**3.8. Käyrä annettujen pisteiden kautta.** Annetun  $n$ :nnän tasopisteen  $(x[i], y[i]), i = 1, \dots, n$ , kautta voidaan piirtää jokin tasokäyrä seuraavasti. Olkoon  $t[i] =$  pisteestä 1 pisteeseen  $i$  välipisteiden kautta piirretyn murtoviivan pituus. Etsitään käyrän parametri-muotoista yhtälöä muodossa  $(p(t), q(t))$  (huomaa, että tasokäyrää ei yleisessä tapauksessa voi antaa muodossa  $y = f(x)$ ), missä  $(p(t[i]), q(t[i])) = (x[i], y[i]), i = 1, \dots, n + 1$  ja  $p$  ja  $q$  ovat sopivia interpolointifunktioita. Käyttäen ohjelmaa gnuplt1 teemme nyt ohjelman, joka merkitsee pisteet kuvaruudulle ja piirtää niiden kautta splini-interpoloinnin antaman käyrän. Jos haluttaisiin, että käyrä olisi umpeutuva, olisi asetettava viimeinen piste samaksi kuin en-



simmäinen. Voitaisiin myös kokeilla esim. polynomi-interpolaatiota samaan tarkoitukseen.

```
// FILE: mysptrp2.cpp begins

// Spline through points. Given a set of points
// in the plane, a curve is drawn through these points.
// Points are in the file myspt.dat
/* g++ -Wall mysptrp2.cpp -L../lib -I../gnuplot02 -o a -lm -lnr */

#include <cstdlib> // Used in putmat2
#include <cstdio> // Used in putmat2
#include <ctime>
#include <string>
#include <fstream>
#include <iostream>
#include <iomanip>
using namespace std;

#include <cmath>
#include "nr.h"
#define PRINT 1

#include "matut102.h"
#include "plot.h"

#define NP 20
#define MP 20
#define MAXSTR 80

static void tabf(Vec_DP &xx,Vec_DP &yy ,int n,const char *name)
{
    ofstream df(name);
    int i;
    if (!df)
        abort();
    for (i = 0; i < n; i++) {
        df<<xx[i]<<" "<< yy[i]<<endl;
    }
    df.close();
}
```

```

DP dist(DP x1, DP y1, DP x2, DP y2)
{
    return pow( (x1-x2)*(x1-x2)+(y1-y2)*(y1-y2), 0.5);
}

void getpoints(Vec_DP &Xa, Vec_DP &Ya1, Vec_DP &Ya2, int &number)
{
    int j;
    const char *fname= "mysptrp3.dat";
    Mat_DP a(MP,NP);
    a=getmat(fname);
    showmat(a);
    int n = a.nrows();
    number=n;
    cout<<"number = "<<n<<endl;
    Vec_DP ya1(n);
    Vec_DP ya2(n);
    Vec_DP xa(n);
    for (j=0;j<n; j++)
        {
            (ya1)[j]=a[j][0];
            (ya2)[j]=a[j][1];
        }
    (xa)[0]=0.0;
    for (j=0;j<n-1; j++) {
        (xa)[j+1]=(xa)[j]+dist((ya1)[j], (ya2)[j], (ya1)[j+1], (ya2)[j+1]);
        cout<<setw(10)<<j<<setw(16)<<(xa)[j+1]<<endl;
    }
    Xa=xa;
    Ya1=ya1;
    Ya2=ya2;
    cout<<"Press enter:";
    getchar();
}

int main()
{
    DP a,b,x,y,yp1, ypn;
    int n, j, m;
    Vec_DP xa(1),ya1(1),ya2(1),xx(100),yy(100);
    // yp1 =0.3; ypn=1/0.3;
    // yp1 =1e30; ypn =1e30 Luonnollinen splini
    getpoints(xa,ya1,ya2,n);
    // n=xa.size();
    Vec_DP y2(n),y3(n);
}

```

```

tabf(ya1,ya2,n,"tmp1.dat");
yp1 = ((ya1)[1] - (ya1)[0]) / ((xa)[1] - (xa)[0]);
ypn = ((ya1)[n-1] - (ya1)[n-2]) / ((xa)[n-1] - (xa)[n-2]);
yp1 = ypn = yp1 + ypn;
NR::spline(xa, ya1, yp1, ypn, y2);
yp1 = ((ya2)[1] - (ya2)[0]) / ((xa)[1] - (xa)[0]);
ypn = ((ya2)[n-1] - (ya2)[n-2]) / ((xa)[n-1] - (xa)[n-2]);
yp1 = ypn = yp1 + ypn;
NR::spline(xa, ya2, yp1, ypn, y3);

a=(xa)[0]; b=(xa)[n-1];
m=100;
for (j=0;j<100;j++) {
    x=a+(b-a)*(j-1)/99;
    NR::splint(xa,ya1,y2,x,y);
    xx[j]=y;
    NR::splint(xa,ya2,y3,x,y);
    yy[j]=y;
    cout<<j<<setw(14)<<xx[j]<<setw(14)<<yy[j]<<endl;
}
tabf(xx, yy,100, "tmp2.dat");
plot("tmp1.dat","b-3",
      "tmp2.dat","r-2",NULL);
return 0;
}
// FILE: mysptrp2.cpp ends

/*
TIEDOSTO mysptrp2.dat:

5 2
0 0
1 0
1 1
0 1
0 0
*/

/*
TIEDOSTO mysptrp.dat:

5 2
0 0
1 0
1 1

```

```
0 1
0 0
*/
```

```
/*
```

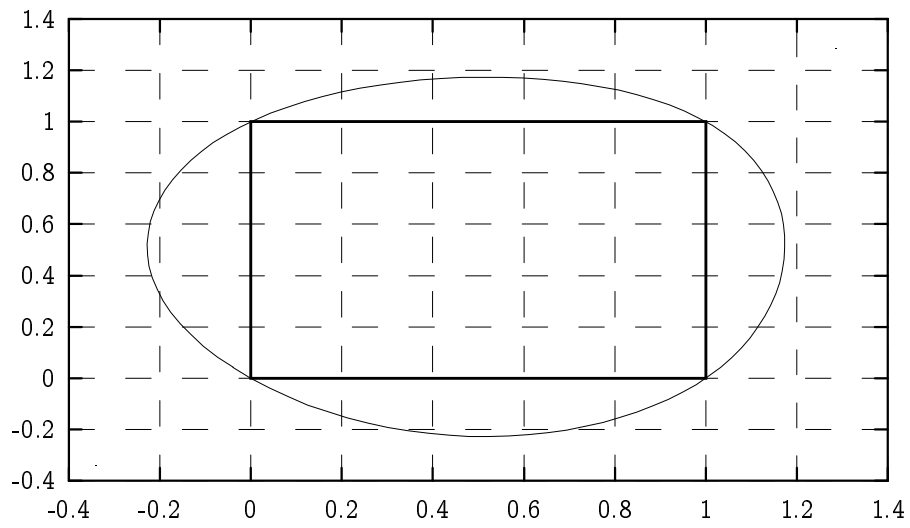
```
TIEDOSTO mysptrp3.dat:
```

```
5 2
0 0
1 0
1 1
0 1
0 0
*/
```

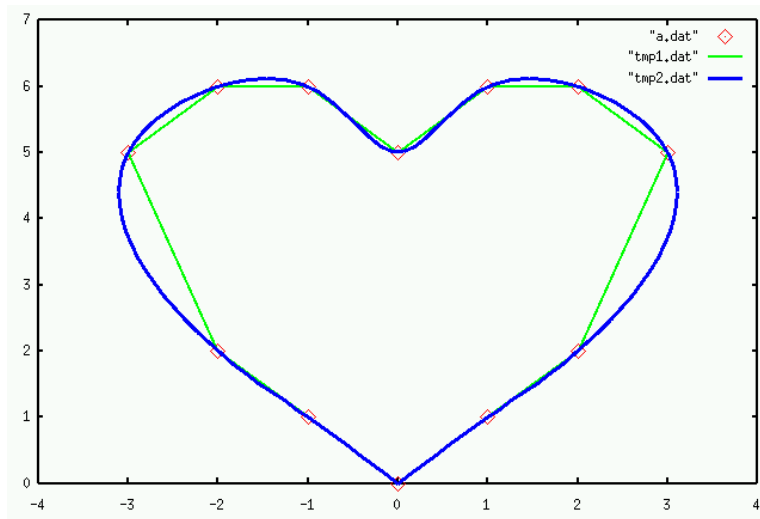
```
Generoitu spline-tiedosto:
```

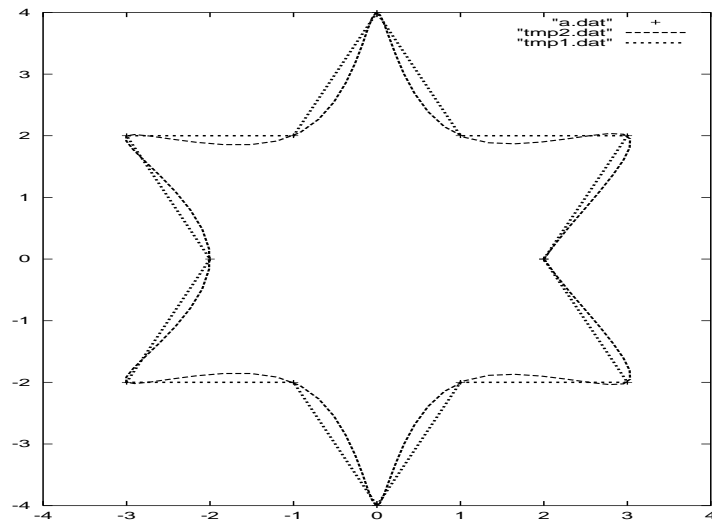
```
0.000000 0.000000
0.040908 -0.038492
0.082737 -0.073206
0.125362 -0.104214
0.168655 -0.131586
(.....)
-0.155392 0.212488
-0.131586 0.168655
-0.104214 0.125362
-0.073206 0.082737
-0.038492 0.040908
0.000000 0.000000
```

Samalla menetelmällä voidaan helposti generoida muita vastaavia kuvia.



Mon Feb 04 01:21:02 2002





## 4 NUMEERINEN INTEGROINTI

Pinta-alojen ja tilavuuksien määritysongelmat ovat eräitä matematiikan vanhimpia sovelluskohteita. Nämä ongelmat voidaan palauttaa integraalien laskemiseen. Koska vain suhteellisen harvat integraalit pystytään laskemaan eksaktisti, käytetään tavallisesti likiarvoja ja numeerista integrointia. Numeerisen integroinnin synonyymi on kvadratuuri. Kvadratuurimenetelmiä käytetään mm. osana ns. elementtimenetelmää, joka on osittaisdifferentiaaliyhtälöiden numeerinen ratkaisumenetelmä.

**Numeerisen integroinnin perustehtävä:** Laske  $\int_a^b f(x)dx$ , kun  $f$  on jatkuva ja  $-\infty < a < b < \infty$ .

Mahdollisia ongelmatilanteita:

- a) epäoleellinen integraali:  $f$ :llä äärettömyyspisteitä,
- b)  $f$ :n raju heilahtelu, terävät huiput.

Tarpeen vaatiessa tehdään lisärajoituksia. Esimerkiksi, jos integroitava funktio on luokassa  $C^4$ , niin kvadratuurikaavoille voidaan johtaa virhearvioita (vrt. allaolevat kaavat).

**Tavoite:** Pyritään saavuttamaan mahdollisimman hyvä approksimaatio määrätylle integraalille mahdollisimman harvoilla funktion arvojen määrityksillä.

Avoin kvadratuuri ei käytä päätepistearvoja  $f(a)$ ,  $f(b)$ .

Suljettu kvadratuuri käyttää päätepistearvoja.

Avoin kvadratuuri voi soveltua myös tapauksiin ” $|f(a)| = \infty$ , tai  $|f(b)| = \infty$ .”

Suljetulla välillä jatkuvaa funktiota numeerisesti integroitaessa voidaan muodostaa välin  $[a, b]$  tasavälinen jako kiinteään määrään osavälejä ja kullakin osavälillä käyttää pinta-alojen approksimointiin suorakulmioita tai puolisuunnikkaita. Toinen mahdollisuus on soveltaa saman idean kehittyneempiä versioita ns. kvadratuurikaavoja. Ne voivat perustua approksimoivien jakojen tihennyksiin tai funktion kasvua tai heilahtelua huomioivien painofunktioiden käyttöön.

Piirtämällä kuvaaja saadaan visuaalinen vaikutelma funktion kulusta. Jos funktio heilahtelee rajusti, voidaan varautua vaikeuk-

siin numeerisessa integroinnissa. Analyysin kielellä ilmaistuna heilahtelu näkyy siinä, että funktion korkeammat derivaatat kasvavat, jolloin myös virhearvio kvadratuurikaavoissa huononee.

**4.1. Tasaväliset jaot.** Merkitään  $x_i = x_0 + ih$ ,  $i = 0, 1, \dots, N + 1$ ,  $f_i = f(x_i)$ . Tärkeimpiä kvadratuurikaavoja ovat

1) Puolisuunnikaskaava (trapetsikaava):

$$\int_{x_1}^{x_2} f(x)dx = h\left[\frac{1}{2}(f_1 + f_2)\right] + o(h^3 f'').$$

2) Simpsonin kaava:

$$\int_{x_1}^{x_3} f(x)dx = h\left[\frac{1}{3}f_1 + \frac{4}{3}f_2 + \frac{1}{3}f_3\right] + O(h^5 f^{(4)}).$$

3) Simpsonin  $\frac{3}{8}$  kaava:

$$\int_{x_1}^{x_4} = h\left[\frac{3}{8}f_1 + \frac{9}{8}f_2 + \frac{9}{8}f_3 + \frac{3}{8}f_4\right] + O(h^5 f^{(4)}).$$

**N-pisteen kaavoja. (suljettu tapaus)** Tihentämällä jakoa ja soveltamalla tihennettyyn jakoon em. kaavoja saadaan seuraavat kaavat.

a) Puolisuunnikaskaava, N-pistettä: Algoritmi NR: :qtrap

$$\int_{x_1}^{x_N} f(x)dx = h\left[\frac{1}{2}f_1 + f_2 + \dots + f_{N-1} + \frac{1}{2}f_N\right] + O\left(\frac{(b-a)^3 f''}{N^2}\right).$$

b) Simpsonin kaava, N-pistettä: Algoritmi NR: :qsimp

$$\int_{x_1}^{x_N} f(x)dx = h\left[\frac{1}{3}f_1 + \frac{4}{3}f_2 + \frac{2}{3}f_3 + \frac{4}{3}f_4 + \dots + \frac{2}{3}f_{N-2} + \frac{4}{3}f_{N-1} + \frac{1}{3}f_N\right] + O\left(\frac{1}{N^4}\right)$$



Eräs menetelmä kaavojen 1)-3) ja niiden erilaisten varianttien johtamiselle perustuu määräämättömien kertoimien menetelmään kuten edellä jo nähtiin ohjelman mynumint.cpp yhteydessä. Kaavoissa esiintyvät kertoimet saatiin testaamalla kaavaa monomifunktioille ja ratkaisemalla muodostuva lineaarinen yhtälöryhmä.

**4.2. Jaon tihennys.** Edellä mainittuja integrointikaavoja voidaan käyttää myös iteratiivisesti siten, että jakopisteitten väliin asetetaan uusia jakopisteitä. Tällöin voidaan jo laskettuja funktion arvoja käyttää hyväksi, joten laskennan aikaisemmassa vaiheessa tehty työ ei mene hukkaan. Jaon tihentämistä jatketaan kunnes haluttu tarkkuus saavutetaan. Algoritmi NR::trapzd, qtrap.

**4.3. Rombergin metodi.** Tasavälisessä jaossa, jossa jakopisteiden välimatka on  $h$ , integraalille saatu approksimaatio  $I(h)$  riippuu parametrasta  $h$ . Jakoa tihennettäessä muodostetaan pienenevä äärellinen jono  $h_j$  lukuja  $h_j > 0$ , ja kullekin lasketaan vastaava approksimaatio  $I(h_j)$ . Jaon tihennystä voidaan käyttää systemaattisesti hyväksi. Ns. Rombergin menetelmässä tämä tapahtuu seuraavasti. Lasketaan puolisuunnikaskaavan antamia arvoja  $I(h_i)$  integraaleille  $\int_a^b f(x)dx$  vastaten välinpituuksia  $h_i$ ,  $i = 1, \dots, n$ , missä  $h_i > h_{i+1}$ . Ekstrapoloidaan Neville'n algoritmin avulla tilanteeseen  $h = 0$ . Asiasta on perusteellinen selostus kirjassa Stoer-Bulirsch [SB]. Algoritmit NR::qromb, qromo liittyvät Rombergin menetelmään.

Alla on esitetty ohjelmien käyttöesimerkki tapauksissa, joissa integraali voidaan laskea myös eksaktisti.

```
// FILE: myqromo.cpp begins
// g++ myqromo.cpp -L../lib
#include<iostream>
#include<iomanip>
#include<cmath>
#include "nr.h"

using namespace std;

#define X1 0.0
#define X2 1.0
#define X3 10000
```

```

#define AINF 1.0E20
#define PI M_PI

DP globalp=2.0;

static DP ff(DP x)
{
    return (pow(x, -1.0/globalp)) ;
}

int main()
{
    DP result,res2;
    int p;
    cout<<"\nImproper integrals:\n\n";
    cout<<"\n int_0^1 x^{-1/p} dx = p/(p-1): \n \n";
    cout<<"  p   Numerical      Exact          Error \n";
    for (p=2;p<10;p++) {
        globalp=p*1.0;
        res2=(DP)(p/(p-1.0));
        result=NR::qromo(ff,X1,X2,NR::midsql);
        cout<<setw(2)<<p<<setw(12)<<result;
        cout<<setw(12)<<res2<<setw(14)<<result-res2<<endl;
    }
    cout<<"\nImproper integrals:\n\n";
    cout<<"\n int_1^{infy} x^{-p} dx = 1/(p-1): \n \n";
    cout<<"  p   Numerical      Exact          Error \n";
    for (p=2;p<10;p++) {
        globalp=1.0/p;
        res2=(DP)(1.0/(p-1.0));
        result=NR::qromo(ff,X2,X3,NR::midinf);
        cout<<setw(2)<<p<<setw(12)<<result;
        cout<<setw(12)<<res2<<setw(14)<<result-res2<<endl;
    }
    return 0;
}
// FILE: myqromo.cpp ends

```

Improper integrals:

int\_0^1 x^{-1/p} dx = p/(p-1):

p	Numerical	Exact	Error
2	2	2	0
3	1.5	1.5	2.24537e-06
4	1.33334	1.33333	2.76962e-06

5	1.25	1.25	1.16561e-06
6	1.2	1.2	3.88171e-06
7	1.16667	1.16667	2.53182e-06
8	1.14286	1.14286	1.7985e-06
9	1.125	1.125	1.35669e-06

Improper integrals:

$\text{int}_1^{\infty} x^{-p} dx = 1/(p-1):$

p	Numerical	Exact	Error
2	0.9999	1	-1e-04
3	0.5	0.5	-5e-09
4	0.333333	0.333333	-3.33344e-13
5	0.25	0.25	5.55112e-17
6	0.2	0.2	5.55112e-17
7	0.166667	0.166667	5.55112e-17
8	0.142857	0.142857	8.32667e-17
9	0.125	0.125	8.32667e-17

Esimerkkejä epäoleellisista integraaleista ovat  $\int_{-\infty}^{-1} \frac{dx}{x^2}$ ,  $\int_0^1 \frac{dx}{\sqrt{x}}$ . Nämä tapaukset voidaan käsitellä em. ohjelmien mukautetuilla versioilla. Näihin tapauksiin liittyviä apuohjelmia ovat NR::midpnt, midinf, midsql, midsqu ja NR::midexp.

NR::midsql (NR::midsqu) soveltuu käytettäväksi yhdessä NR::qromo:n kanssa tilanteissa, joissa integraalilla on integroitava singulariteetti alarajalla (yläraajalla), kuten seuraavissa tapauksissa:

$$\int_0^1 \frac{dx}{\sqrt{x}} \quad \left( \int_0^1 \frac{dx}{\sqrt{1-x}} \right).$$

Edellä ohjelmassa myqromo.c annettiin esimerkki aliohjelman NR::qromo, midsql yhteistoiminnasta. Vastaavasti voidaan käsitellä tilanne, jossa sekä ylä- että alarajalla on singulariteetti.

**Esim.**

$$\int_0^1 \frac{dx}{\sqrt{x(1-x)}} \approx \int_s^{0.5} + \int_{0.5}^{1-s} = I_1 + I_2,$$

missä  $s$  on sopiva pieni positiivinen luku. Edelleen

$$I_1 = \text{NR}::\text{qromo}(f, s, 0.5, \text{midsql}),$$

$$I_2 = \text{NR}::\text{qromo}(f, 0.5, 1.0-s, \text{midsqu})$$

Monipuolinen kokoelma kvadratuurikaavoja on esim. teoksessa Abramowitz-Stegun [AS], luku 25.

**4.4. Gaussin kvadratuuri** Vektoriavaruuksien teoriasta tuttu ortogonaalisuuden käsite voidaan yleistää eri tavoin. Pehdymme nyt erääseen tällaiseen numerikaan kannalta varsin hyödylliseen yleistykseen, jossa vektoriavaruuden muodostavat välillä  $[a, b]$  määritellyt jatkuvat funktiot. Olkoot  $f$  ja  $g$  tällaisia funktioita ja  $W$  jokin kolmas funktio, jota kutsutaan *painofunktioksi*. Siinä tapauksessa, että em. funktiot ovat riittävän säännöllisiä (jolloin ko. integraalit ovat hyvin määriteltäviä; tavallisesti  $W(x) > 0 \quad \forall x \in [a, b]$  ja jatkuva), määrittelemme funktioiden  $f$  ja  $g$  *sisätulon* kaavalla

$$\langle f|g \rangle \equiv \int_a^b W(x)f(x)g(x)dx.$$

Funktio  $f$  ja  $g$  ovat *ortogonaalisia*, jos niiden sisätulo on 0. Funktiota, jonka sisätulo itsensä kanssa on 1 sanotaan *normitetuksi*. Jos normitetun funktiojoukon funktiot ovat ortogonaalisia keskenään, joukkoa sanotaan *ortonormaaliksi*.

Voimme konstruoida ortonormitetun joukon polynomeja, jossa on täsmälleen yksi polynomi  $p_j(x)$  kutakin astetta  $j = 0, 1, 2, 3, \dots$  Rekursiivinen konstruktio etenee seuraavasti. Asetetaan

$$p_{-1}(x) \equiv 0, \quad p_0(x) \equiv 1,$$

$$p_{j+1}(x) = (x - a_j)p_j(x) - b_j p_{j-1}(x), \quad j = 0, 1, 2, \dots,$$

missä

$$a_j = \frac{\langle xp_j|p_j \rangle}{\langle p_j|p_j \rangle}, \quad j = 0, 1, 2, \dots, \quad b_j = \frac{\langle p_j|p_j \rangle}{\langle p_{j-1}|p_{j-1} \rangle}, \quad j = 1, 2, 3, \dots$$

Kerroin  $b_0$  asetetaan mielivaltaisesti; valitsemme sen nolllaksi.

Ortogonaalisilla polynomeilla on tärkeitä sovelluksia mm. numeeriseen integrointiin. Integraalin

$$\int_a^b W(x)f(x)dx \approx \sum_{j=1}^N w_j f(x_j)$$

approksimoinnissa voidaan abskissat  $x_j$  ja painot  $w_j$  valita niin, että kaava pätee yhtälönä kaikille enintään astetta  $(2N - 1)$  oleville polynomeille. Tämä johtuu siitä, että kaavassa on  $2N$  parametria, nimittäin painot  $w_j$  ja abskissat  $x_j$ .

Kirjamme esittelee sivulla 151 useita numeerisen integroinnin yhteydessä esiintyviä ortogonaalisia polynomeja. Mainittakoon näistä *Gaussin-Legendren polynomi*:

$$W(x) = 1, -1 < x < 1, (j + 1)P_{j+1} = (2j + 1)xP_j - jP_{j-1}.$$

Algoritmi NR::gauleg nojautuu tämän integrointipolynomin käyttöön.

Gaussin kvadratuurikaavat pohjautuvat ortogonaalisiin polynomeihin. Mm. pisteet  $x_j$  valitaan tarkoitukseen parhaiten sopivalla tavalla ja mahdollisesti ei-tasavälisesti. Jakopisteet  $x_j$  määräytyvät ortogonaalipolynomien ominaisuuksista. Etuna on se, että saadaan lisää vapausasteita ja tarkkuutta.

Seuraava ohjelma tarkastelee algoritmin NR::gauleg mukaista numeerista integrointia. Algoritmin osana on pisteiden  $x_j$  paikallistaminen Gaussin kvadratuurin mukaisesti. Sovellamme algoritmia tapaukseen, missä integrandina on funktio  $\sum_{j=1}^M a[j] * \sin(b[j] * x)$  ja missä  $a$  ja  $b$  ovat satunnaisvektoreita. Tarkkuusarviota varten taulukoimme virhefunktion ff3(x), joka ilmoittaa eksaktin integraalin ja numeerisen integraalin välisen erotuksen integroinnin ylärajan funktiona, kun integrointivälinä on  $[0, x]$ .

```
// FILE: mygleg.cpp begins
// Integrand is \sum_{j=1}^M a[j]*sin(b[j]*x)
// with a and b random numbers and exact integral
// \sum_{j=1}^M -(a[j]/b[j])*cos(b[j]*x)
/* g++ mygleg.cpp -L../lib -I../utils -I../democpp02
-I../gnuplot02 -o a -lm -lnr
```

```

*/

#include <cmath>
#include <cstdlib>
#include <cstdio>
#include <ctime>
#include <string>
#include <fstream>
#include <iostream>
#include <iomanip>

#define PRINT 1
using namespace std;
#include "nr.h"
#include "gnuplt1.h"
#include "matutl02.h"

#define NPOINT 10
#define MTERM 10
#define X1 0.0
#define X2 1.0
#define X3 5.0

// X1, X3 are the lower, upper bounds of integration

double a[MTERM], b[MTERM]; // MTERM is an upper bound for M
int M; // M is the number of sine terms

double myfunc(double x)
// myfunc gives the values of the sine sums
{
    double s=0.0;
    for (int j=0;j<M; j++) s+=a[j]*sin(b[j]*x);
    return s;
}

double myfunc2(double x)
// myfunc2 gives the integral function of myfunc
{
    double s=0.0;
    for (int j=0;j<M; j++) s+=a[j]*cos(b[j]*x)/b[j];
    return -s;
}

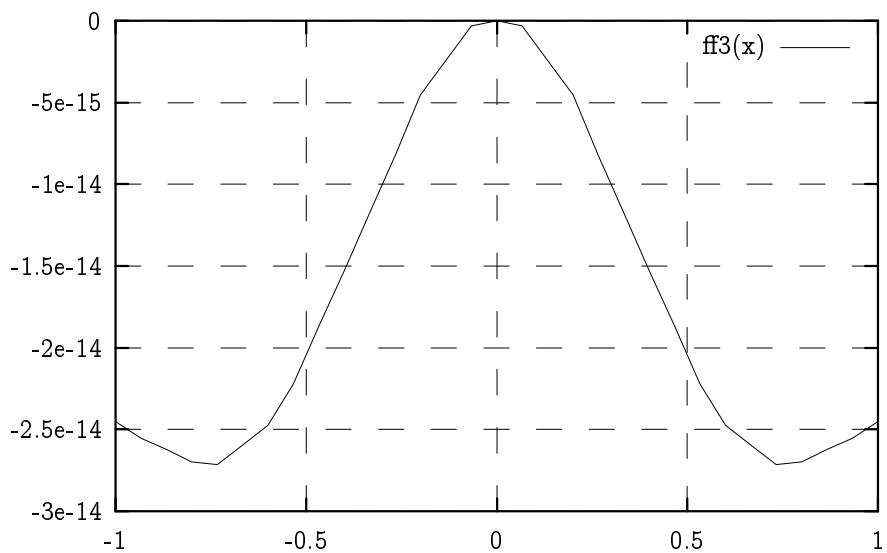
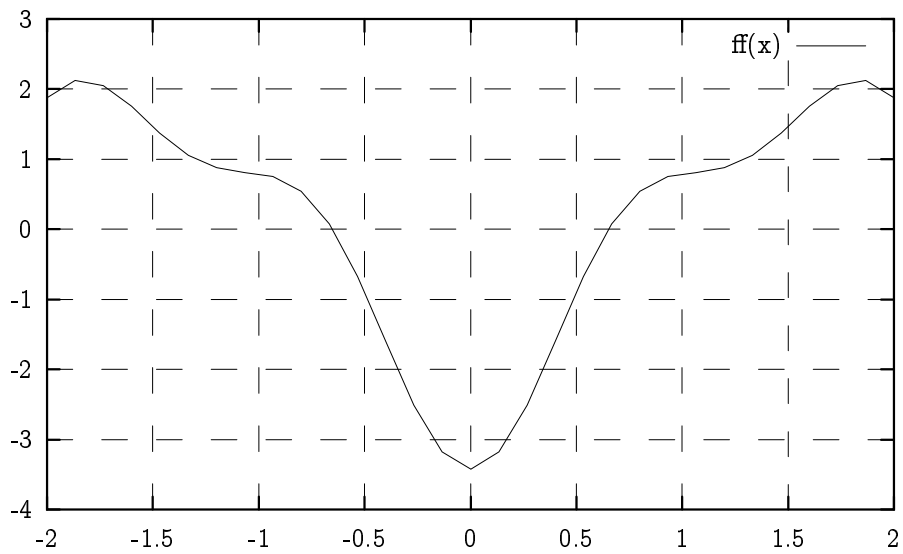
```

```

double ff3(double r)
// This is the error in the gauleg algorithm for
// numerical integration of myfunc
{
    double a=0.0,b=r,c=0.0;
    Vec_DP x(NPOINT),w(NPOINT);
    NR::gauleg(a,b,x,w);
    for (int i=0;i<NPOINT;i++) c+=(w[i]*myfunc(x[i]));
    return c - ( -myfunc2(a)+myfunc2(b));
}

int main()
{
    double xx=0.0, zz;
    Vec_DP x(NPOINT),w(NPOINT);
// Generate random coefficients
    for (int i=0;i<MTERM;i++) {a[i]=0.0; b[i]=0.0;}
    M=(4<MTERM)? 4:MTERM ; // M must be smaller than MTERM
    init_srand();
    for (int i =0;i<M;i++)
    { // rdm is from matutl
        a[i]=rdm(0.2,3.0);
        b[i]=rdm(0.5,6.0) ;
    }
    cout<<"\na = ";
    for (int i =0;i<M;i++) cout<<setw(14)<<a[i];
    cout<<"\nb = ";
    for (int i =0;i<M;i++) cout<<setw(14)<<b[i];
    NR::gauleg(X1,X3,x,w);
    for (int i=0;i<NPOINT;i++) xx+=(w[i]*myfunc(x[i]));
    cout<<"\nIntegral from GAULEG: "<<xx<<endl;
    zz= -myfunc2(X1)+myfunc2(X3);
    cout<<"Actual value: "<< zz<<endl;
    cout<<"Error = "<< zz-xx<<endl;
    gnuplt1(myfunc2,"ff(x)",0, NULL);
    gnuplt1(ff3,"ff3(x)",0, NULL);
    return 0;
}
// FILE: mygleg.cpp ends

```



Ylempi kuva esittää integroitavaa funktiota ja alempi algoritmin NR::gauleg avulla tapahtuvan numeerisen integroinnin virhettä ylärajan funktiona, kun integrointi tehdään yli välin  $[0, x]$ . Virhe voidaan laskea koska integrointi palautuu trigonometrisiin funktioihin.



## 5 FUNKTIOIDEN APPROKSIMOINTI

Kaksi erilaista, mutta matemaattisesti identtistä funktioiden laskutapaa voivat numeerisilta ominaisuuksiltaan olla täysin erilaisia (pyöristysvirheiden vaikutus, epästabiilit menetelmät). Tunnetun matemaattisen identiteetin nojalla  $\cos(x)$  ja  $\sqrt{1 - \sin^2(x)}$  ovat samoja, kun  $x \in (0, \pi/2)$ , mutta ei ole takeita siitä, että tietokoneitse laskettaessa niille saataisiin täsmälleen samat arvot. Vastauksissa todennäköisesti esiintyy pyöristysvirheen suuruusluokkaa olevia eroja, kuten olemme Luvussa 1 nähneet.

Funktioiden arvojen laskemiseen käytetään useita erilaisia approksimoiteja, joista NR esittää katsauksen Luvussa 5. Tuttu Taylorin sarja on yleensä numeerisiin tarkoituksiin huono. Jos Taylorin sarjasta käytetään  $p$  ensimmäistä termiä funktion  $\exp(-x)$  arvojen laskemiseen pisteessä  $x = -5$ , ovat tulokset surkeita, jos  $p \leq 10$ . Parempia vaihtoehtoja ovat esimerkiksi muut polynomi- ja rationaalifunktiokehitykset, Chebyshevin ja Padén menetelmät.

Trigonometrinen funktioiden arvoja laskettaessa voidaan menetellä seuraavasti. Ensin johdetaan jollakin välillä, esim.  $[-1, 1]$  pätevä approksimaatio, jonka virheelle tiedetään arvio. Sen jälkeen palautetaan yleinen tapaus muunnoskaavojen avulla tähän tapaukseen.

Polynomin arvon määrittämistä ei pidä tehdä näin:

```
p=c[1]+c[2]*x+c[3]*pow(x,2)+c[4]*pow(x,3)+c[5]*pow(x,4);
```

vaan ns. *Hornerin kaavan* mukaisesti:

```
p=c[n];  
for (j=1;j <= n-1;j++) p=p*x+c[n-j];
```

Myös polynomin derivaatta voidaan laskea samaan tapaan.

Olemme toistaiseksi tarkastelleet reaalisen lukualueen matriiseja vektoreita jne. Vastaavat tarkastelut voidaan ulottaa myös

kompleksiseen tapaukseen tarkastelemalla näitä reaalityyppien parejaina. Koska C-kieli ei tue kompleksilukujen käyttöä (C++ tukee), on käyttäjän implementoitava kompleksilukujen aritmeettiset operaatiot, kuten yhteenlasku, kertominen, juurenotto jne. NR esittää sivuilla 948-950 kompleksilukujen aritmeettisiin operaatioihin soveltuvia algoritmeja C-kielissä. Seuraava C++-ohjelma valottaa algoritmien käyttöä ja antaa arvion NR:n kompleksilaskutoimitusten tarkkuudesta.

Palautamme mieleen luvusta 1, että hypergeometrinen sarja määritellään kaavalla

$${}_2F_1(a, b; c; z) = \sum_{n=0}^{\infty} \frac{(a, n)(b, n)}{(c, n)n!} z^n,$$

missä  $(a, 0) = 1$ ,  $(a, n + 1) = (a, n)(a + n)$ . Kirjamme esittää sivuilla 272-273 algoritmin funktion laskemiseksi. Seuraava ohjelma osoittaa oikeaksi identiteetin  $F(\pi/2, \sqrt{k}) = (\pi/2)F(1/2, 1/2, 1, k)$ , missä

$$F(\pi/2, k) \equiv \int_0^{\phi} \frac{d\theta}{\sqrt{1 - k^2 \sin^2 \theta}}.$$

```
// FILE: myhypser2.cpp begins
// g++ myhypser2.cpp -L../lib -I../utils -o a -lm -lnr

#include <iostream>
#include <iomanip>
#include <cmath>
#include <complex>
#include "nr.h"
using namespace std;

// Driver for routine hypser

DP ff2(DP x)
{
    DP xx= x, y=NR::ellf(0.5*M_PI,xx);
    return y;
}

int main()
{
    int p;
```

```

DP x,r,s,kk;
complex<DP> a(0.5,0.0),b(0.5,0.0),c(1.0,0.0);
complex<DP> z,series,deriv;

cout<<" a= 1/2, b=1/2, c =1 \n";
cout<<"      k          F(a,b;c;k)      (2/PI)*F(PI/2,sqrt(k))      Error\n";
for (p=1;p<=9; p++) {
    x=((DP)p)*0.1;
    z=complex<DP>(x,0.0);
    NR: :hypser(a,b,c,z,series,deriv);
    r=series.real(); s=series.imag();
    kk=(2/M_PI)*((ff2(pow(x, 0.5)))));
    cout<<setw(7)<<x<<setw(14)<<series.real();
    cout<<setw(20)<<kk<<setw(20)<<fabs(r-kk)<<endl;
}
cout<<"This seems to show that F(PI/2,sqrt(k))=(PI/2) F(1/2,1/2,1,k)\n";
cout<<"which holds by AS, 17.3.9.\n";
return 0;
}
// FILE: myhypser2.cpp ends

```

```

a= 1/2, b=1/2, c =1
  k          F(a,b;c;k)      (2/PI)*F(PI/2,sqrt(k))      Error
 0.1          1.02651          1.02651          4.44089e-16
 0.2          1.05655          1.05655           0
 0.3          1.0911           1.0911          4.44089e-16
 0.4          1.1316           1.1316          2.22045e-16
 0.5          1.18034          1.18034          1.11022e-15
 0.6          1.24113          1.24113          2.22045e-16
 0.7          1.32122          1.32122          4.44089e-16
 0.8          1.43698          1.43698          2.22045e-16
 0.9          1.64126          1.64126          4.44089e-16

```

This seems to show that  $F(\pi/2, \sqrt{k}) = (\pi/2) F(1/2, 1/2, 1, k)$   
which holds by AS, 17.3.9.

## Chebyshevian approksimointi

Astetta  $n$  oleva Chebyshevian polynomi  $T_n(x)$  määritellään kaavalla

$$T_n(x) \equiv \cos(n \arccos(x)), \quad x \in [-1, 1].$$

Silloin

$$T_0(x) = 1, T_1(x) = x, T_2(x) = 2x^2 - 1, \\ T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x), n \geq 1.$$

Chebyshevin polynomit ovat ortogonaalisia:

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

Approksimointi Chebyshevin polynomeilla perustuu seuraavaan lauseeseen.

**Lause.** Olkoon  $f$  välillä  $[-1, 1]$  määritelty jatkuva funktio ja kertoimet  $c_j, j = 0, 1, 2, \dots, N - 1$ , määritelty seuraavasti

$$c_j = \frac{2}{N} \sum_{k=1}^N f(x_k)T_j(x_k), \quad x_k = \cos\left(\frac{\pi(k - \frac{1}{2})}{N}\right).$$

Silloin kaava

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

pätee yhtälönä, kaikissa  $T_N(x)$ :n nollakohdissa  $x_k, k = 1, 2, \dots, N$ .

Eräs Chebyshevin polynomien etu on kertoimien  $c_k$  nopea pieneeminen. Jos yo. approksimaation asemasta käytetäänkin osasummaa, jossa ylärajana on  $m \leq N$ , niin muodostuva ero on enintään  $\sum_{k=m+1}^N |c_k|$ , koska  $|T_n(x)| \leq 1$ .

Seuraava ohjelma antaa esimerkin *Chebyshevin approksimaation* käytöstä. Esimerkkifunktiona on samankaltainen funktio kuin Luvun 4 lopussa tarkastellussa esimerkissä. Ohjelma generoi testifunktiolle Chebyshevin approksimaation ja piirtää kuvan alkuperäisestä funktiosta ja approksimaatiosta.

```
/* FILE: mychebft.cpp begin                                     */
// Plots a function and its Chebyshev approximation
// g++ mychebft.cpp -L../lib -I../utils -I../democpp02 -I../gnuplot02 -o a -m -lnr
/* Modified from driver for routine chebft */
```

```

#include <cstdlib> // Used in putmat2
#include <cstdio> // Used in putmat2
#include <ctime>
#include <string>
#include <fstream>
#include <iostream>
#include <iomanip>
using namespace std;
#include "nr.h"
#include "matutl02.h"
#define PRINT 1

#include "gnuplt1.h"

#define NVAL 40
#define PI02 (0.5*M_PI)
#define MTERM 15

Vec_DP ccoef(NVAL), acoef(MTERM), bcoef(MTERM);
int M, mval; /* M is the number of sine terms */

double func(double x)
/* myfunc gives the values of the sine sums */
{
    float s=0.0; int j;
    for (j=1;j<=M; j++)
        s+=acoef[j]*sin(bcoef[j]*x);
    return s;
}

double ff2(double xx)
{
    DP f= NR::chebev((-PI02),PI02,ccoef, mval, xx);
    return f;
}

int main()
{
    double a=(-PI02),b=PI02,f, x;
    int i;
    init_srand();
    M=5; /* Generate random coefficients; M<MVAL */
    for (i=0;i<=M-1;i++) { /* rdm is from matutl02.cpp */
        acoef[i]=rdm(0.5,5.0);
        bcoef[i]=rdm(-0.5,5.0) ;};
}

```

```

printf("\na = ");
for (i =0;i<=M-1;i++) {printf(" %8.5f", acoef[i]);};
printf("\nb = ");
for (i =0;i<=M-1;i++) {printf(" %8.5f", bcoef[i]);};
Vec_DP c(NVAL);
NR::chebft(a,b,c,func);
for (i=0 ;i<=NVAL-1;i++)
{
    if (fabs(c[i]) > 1e-9) printf("\nc[ %2d] = %12.5e \n",i, c[i]);
    ccoef[i]=c[i];
}
cout<<"\nFunction depends on "<<2*M<<" parameters\n";
/* test result */
printf("\nHow many terms in Chebyshev evaluation?\n");
printf("Enter n between 6 and %2d. (n=0 to end).\n",NVAL);
scanf("%d",&mval);
printf("You entered: %2d\n",mval);
if ((mval<2)|| (mval>NVAL-1)) abort();
Vec_DP c2(mval);
for (i=0 ;i<=mval-1;i++) {
    c2[i]= ccoef[i]; }
printf("\n%9s %14s %16s\n","x","actual","Chebyshev fit");
for (i=-8;i<=8;i++) {
    x=i*PI02/10.0;
    f=NR::chebev(a,b,c2,mval,x);
    printf("%12.6f %12.6f %12.6f \n",x,func(x),f);
}
gnuplt1(func,"func(x)",0,ff2,"ff2(x)",2,NULL);
return 0;
}
/* FILE: mychebft.cpp end */

```

```

a = 1.45756 4.89216 2.31717 1.39905 2.39889
b = 4.52844 3.69437 2.09250 4.54659 -0.03627
c[ 1] = -2.02306e+00
c[ 3] = -2.79034e+00
c[ 5] = 4.63701e+00
c[ 7] = -1.80785e+00
c[ 9] = 3.50729e-01
c[ 11] = -4.24137e-02
c[ 13] = 3.57098e-03
c[ 15] = -2.23239e-04
c[ 17] = 1.07587e-05
c[ 19] = -3.26085e-07

```

```
c[ 21] = -5.11194e-08
```

```
.....
```

```
c[ 37] =  2.88002e-08
```

```
c[ 39] = -9.61879e-09
```

Function depends on 10 parameters

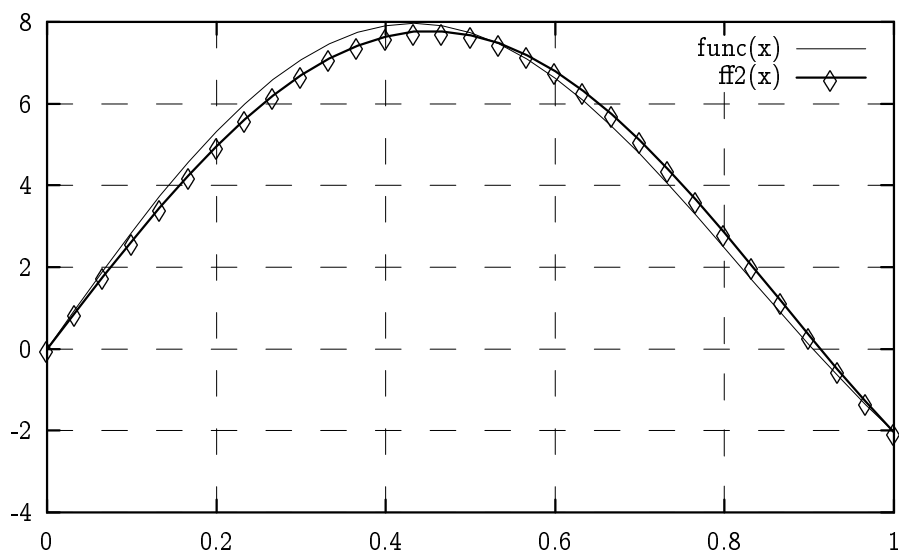
How many terms in Chebyshev evaluation?

Enter n between 6 and 40. (n=0 to end).

8

You entered: 8

x	actual	Chebyshev fit
-1.256637	4.608825	4.886785
-1.099557	3.604818	3.860278
-0.942478	0.854915	0.722675
-0.785398	-2.823468	-3.193391
-0.628319	-6.158212	-6.389697



### Padén approksimointi

Astetta  $N$  oleva rationaalifunktio on lauseke

$$R(x) = \frac{p(x)}{q(x)},$$

missä  $p$  ja  $q$  ovat polynomeja, joiden astelukujen summa =  $N$ . Erikoistapauksena  $q(x) \equiv 1$  saadaan polynomit. Näinollen on selvää, että astetta  $N$  olevalla rationaalifunktioapproksimoinnilla saavutetaan suurempi tarkkuus kuin samaa astetta olevalla polynomiapproksimaatiolla. Lisäetuna rationaalifunktioilla on, että niitä voidaan käyttää myös approksimoitaessa funktioita, joilla on singulariteetti (eli äärettömyyspiste) lähellä approksimointiväliä, mutta kuitenkin välin ulkopuolella.

Lähtökohtana approksimaatiolle

$$f(x) = \sum_{k=0}^{\infty} c_k x^k \approx \frac{\sum_{k=0}^M a_k x^k}{1 + \sum_{k=1}^N b_k x^k} \equiv R(x) = \frac{p(x)}{q(x)}$$

ovat vaatimukset

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

Laventamalla edellinen yhtälö kertomalla puolittain oikeanpuolen nimittäjällä, ja siirtämällä sopivasti termejä saamme lausekkeen

$$f(x)q(x) - p(x).$$

Vaativalla, että lausekkeen sarjakehitelmästä häviävät kaikki astetta  $\leq M + N$  olevat termit, saadaan rekursiivinen ehto kertoimien määrittämiselle. Muodostuva lineaarinen yhtälöryhmä voidaan ratkaista Luvun 2 menetelmin. Padé-approksimointi on MacLaurin-kehityksen yleistys.

Algoritmi NR: :pade (s. 202) antaa Padé-approksimoinnin, jossa  $M = N$ .

```
// FILE: mypade.cpp begins
// g++ mypade.cpp -L../lib -I../utils -o a -lm -lnr
// This program computes Pade approximation for
// 2^x in terms of p(x)/q(x) where both p and q
// are polynomials of degree n.
```

```
#include <iostream>
#include <iomanip>
#include <cmath>
```



```

#include "nr.h"
using namespace std;

int main(void)
{
    int j,k,n;
    DP resid,b,d,fac,x;

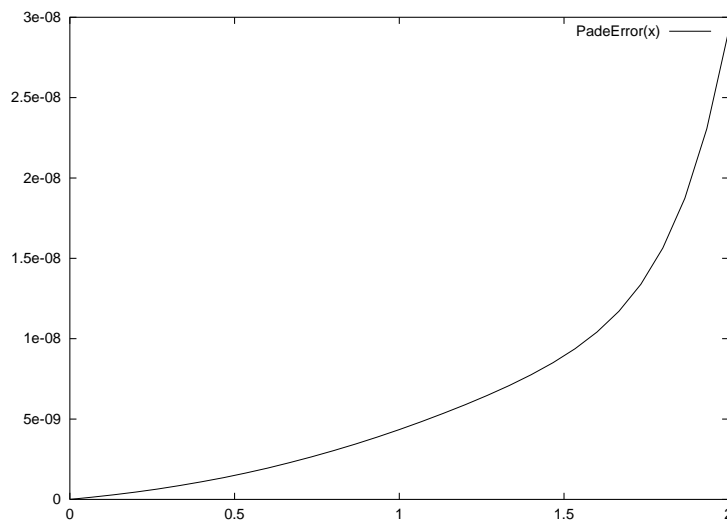
    for (;;) {
        cout << "Enter n for PADE routine (or 0 to stop): ";
        cin >> n;
        cout << endl;
        if (n < 1) break;
        Vec_DP c(2*n+1),cc(2*n+1);
        fac=1;
        for (j=0;j<2*n+1;j++) {
            c[j]=1.0/fac;
            cc[j]=c[j];
            fac =(fac/log(2.0))*((double)j+1.0);
        }
        NR::pade(c,resid);
        cout << "Norm of residual vector= " << scientific;
        cout << setw(8) << resid << endl;
        cout << setw(14) << "point" << setw(19) << "func. value";
        cout << setw(16) << "pade series" << setw(16) << "power series";
        cout << setw(16) << "pade-func" << endl;
        cout << fixed << setprecision(8);
        for (j=0;j<21;j++) {
            x=j*0.25;
            for (b=0.0,k=2*n;k>=0;k--) {
                b *= x;
                b += cc[k];
            }
            d=NR::ratval(x,c,n,n);
            cout << setw(8) << x << setw(12) << pow(2.0,x);
            cout << setw(16) << d << setw(16) << b;
            cout << setw(15) << pow(2.0,x)-d << endl;
        }
        cout << endl;
    }
    return 0;
}
// FILE: mypade.cpp ends

```

Enter n for PADE routine (or 0 to stop): 6

Norm of residual vector= 5.7246106e-17

point	func. value	pade series	power series	pade-func
0	1	1	1	0
0.25	1.1892071	1.1892071	1.1892071	-2.220446e-16
0.5	1.4142136	1.4142136	1.4142136	2.220446e-16
0.75	1.6817928	1.6817928	1.6817928	0
1	2	2	2	3.1086245e-15
1.25	2.3784142	2.3784142	2.3784142	6.5281114e-14
1.5	2.8284271	2.8284271	2.8284271	8.3311136e-13
1.75	3.3635857	3.3635857	3.3635857	7.4069639e-12
2	4	4	4	5.0424109e-11
2.25	4.7568285	4.7568285	4.7568284	2.8004532e-10
2.5	5.6568542	5.6568542	5.656854	1.3249197e-09
2.75	6.7271713	6.7271713	6.7271705	5.5069762e-09
3	8	8	7.9999974	2.0572496e-08
3.25	9.5136569	9.5136568	9.5136496	7.0277862e-08
3.5	11.313708	11.313708	11.313689	2.2250681e-07
3.75	13.454343	13.454342	13.454294	6.5991298e-07



Padé-approksimointi on yo. esimerkissä yllättävän tarkka. Chebyshevin approksimoinnissa saadaan aikaan jollakin välillä hyvä tarkkuus, kun taas Padé-approksimoinnissa tarkkuusalueesta ei voida sanoa juuri mitään.

## 6 ERIKOISFUNKTIOT

Usein esiintyvää tai muuten hyödyllistä funktiota sanotaan erikoisfunktioiksi. Näiden funktioiden merkitys ja erityisyys on siinä, että ne mahdollistavat monissa sovelluksissa esiintyvien ääriarvot tehtävien ratkaisemisen analyttisessä muodossa. Suuri osa erikoisfunktioista toteuttaa jonkin toisen kertaluvun differentiaaliyhtälön. Matemaattinen fysiikka ja kombinatoriikka ovat tieteenaloja, joissa erikoisfunktioita tavantakaa käytetään.

Toisinaan erikoisfunktiot jaetaan ns. alkeisfunktioihin ja korkeampiin transkendenttifunktioihin. Esimerkkejä alkeisfunktioista ovat polynomit, trigonometriset funktiot, hyperboliset funktiot ja eksponenttifunktiot sekä näiden käänteisfunktiot.

Korkeampiin transkendenttifunktioihin kuuluvat Eulerin  $\Gamma$ -funktio, useat todennäköisyyslaskennan funktiot, Besselin funktiot, elliptiset integraalit ja elliptiset funktiot sekä Gaussin hypergeometrisen funktio. Erikoisfunktioiden määrittelyalue on tavallisesti jokin kompleksitason osa-alue. Ohjelmakirjastot tarjoavat vaihtelevan kokoelman algoritmeja erikoisfunktioille: vaihtelua esiintyy erityisesti algoritmien pätevyysalueen ja virherajojen suhteen.

Kansainvälisesti merkittävimpiä perusteoksia erikoisfunktioiden alalla on [AS], joka on myös erittäin hyödyllinen hakuteos numeriiikan alalla. Se lienee eniten siteerattuja matematiikan käsikirjoja.

**6.1. Gamma-funktio.**  $\Gamma$ -funktiolle on useita ekvivalentteja esityksiä. Tavallisin näistä on Eulerin integraali

$$\Gamma(z) = \int_0^{\infty} t^{z-1} e^{-t} dt, \operatorname{Re}\{z\} > 0.$$

Funktion määrittelyaluetta voidaan laajentaa käyttäen iteratiivisesti kaavaa

$$\Gamma(z + 1) = z\Gamma(z).$$

Kaavan avulla gammafunktion määrittelyjoukko voidaan laajentaa joukoksi  $\mathbb{C} \setminus \{0, -1, -2, \dots\}$ . Poikkeuspisteissä funktiolla on napa eli

äärettömyyspiste, katso alempana olevaa kuvaa  $Re\Gamma(z)$ :sta. Eräitä perusominaisuuksia on se, että  $\Gamma(z)$  on kertomafunktion yleistys:

$$\Gamma(n+1) = n!, \quad n = 1, 2, 3, \dots, \Gamma(1) = \Gamma(2) = 1.$$

Silloin pätee kehitelmä

$$\Gamma(z+1) = \left(z + \gamma + \frac{1}{2}\right)^{z+\frac{1}{2}} e^{-(z+\gamma+\frac{1}{2})} \sqrt{2\pi} \left[ C_0 + \frac{C_1}{z+1} + \dots + \frac{C_N}{z+N} + \epsilon \right].$$

Huomioiden yo. monimutkaisen määritelmän, on seuraava tieto melko yllättävä, nimittäin, että jos  $\gamma = 5$ ,  $N = 6$ , niin on olemassa sopivat vakiot  $C_0, \dots, C_6$ , joille virhetermille pätee  $|\epsilon| < 2 \cdot 10^{-10}$  (alueessa  $Re z > 0$ ).

NR tarjoaa algoritmeja seuraaville funktioille, jotka ovat läheisessä yhteydessä  $\Gamma$ -funktioon:

NR::gammln(x),  $x > 0$ , laskee  $\log \Gamma(x)$ :n arvoja

NR::factrl(n) laskee  $n!$ :n arvoja

NR::bico(N,K), laskee  $\binom{N}{K}$ :n arvoja

NR::beta(p,q) laskee  $B(p,q) = \frac{\Gamma(p)\Gamma(q)}{\Gamma(p+q)}$ :n arvoja.

```
// FILE: mybico2.cpp begins
// Driver for routine bico
```

```
#include <cmath>
```

```
#define SCALE 1
```

```
#define PRINT 1
```

```
#include "nr.h"
```

```
#include "gnusurf.h"
```

```
double f(double x, double y)
{
```

```

    int n=(int)floor(x), p=(int)floor(y);
    if (n<=p) n=p+1;
    return NR::bico(n,p);
}

int main()
{
    int k,n,sum;
    double xx[]={1.,6.},yy[]={1.,6.};
    cout<<"Pascal triangle (and row sum):"<<endl;
    for (n=1;n<=8; n++)
    {
        sum=0;
        for (k=0;k<=n;k++)
        {
            sum+=(int)NR::bico(n,k);
            cout<<" "<<NR::bico(n,k) <<" ";
            if (k==n) cout << " ( "<<sum<<" )\n";
        }
    }
    gnusurf(f,xx,yy,SCALE,0,"bico(x,y)","testi");
    return 0;
}
// FILE: mybico2.cpp ends

```

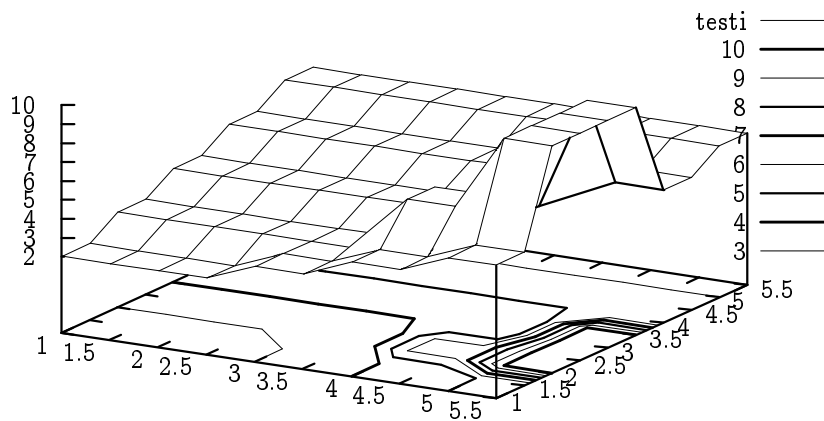
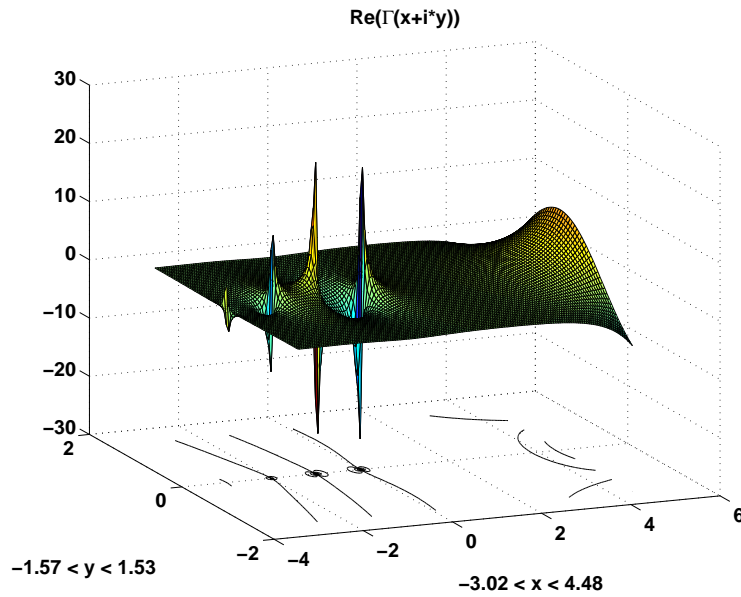
```
[vuorinen@mat-148]~/nrcpp02/democpp02 ./a
```

```
Pascal triangle (and row sum):
```

```

1 1 ( 2 )
1 2 1 ( 4 )
1 3 3 1 ( 8 )
1 4 6 4 1 ( 16 )
1 5 10 10 5 1 ( 32 )
1 6 15 20 15 6 1 ( 64 )
1 7 21 35 35 21 7 1 ( 128 )
1 8 28 56 70 56 28 8 1 ( 256 )

```



NR:ssä on implementoitu mm. seuraavia funktioita:

- Epätäydellinen gamma-funktio: NR:::gammp(A, X)

$$P(a, x) = \frac{\gamma(a, x)}{\Gamma(a)} = \frac{1}{\Gamma(a)} \int_0^x e^{-t} t^{a-1} dt \quad (a > 0).$$

- Virhefunktio  $\operatorname{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$  ja muita todennäköisyyslaskennan funktioita.

- Besselin funktiot  $J_\nu(z) = \left(\frac{1}{2}z\right)^\nu \sum_{k=0}^{\infty} \frac{(-\frac{1}{4}z^2)^k}{k! \Gamma(\gamma+k+1)}$ ,  $\nu, \gamma \in \mathbb{R}$  (ks. kuva 6.5.1/NR s.231).

```

// FILE: mybess.cpp begins
// Bessel functions

#include <fstream>
#include <iostream>
#include <iomanip>
#include "nr.h"
#define PRINT 1
#include "gnuplt1.h"

using namespace std;
double ff1(double x)
{
    return NR::bessj0(x);
}

double ff2(double x)
{
    return NR::bessy0(x);
}

double ff3(double x)
{
    return NR::bessj1(x);
}

double ff4(double x)
{
    return NR::bessy1(x); // for positive x
}

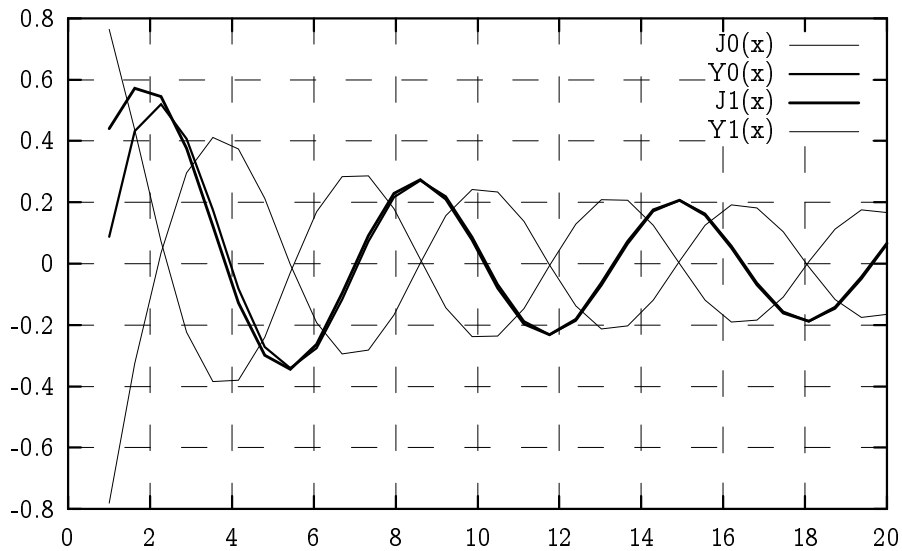
int main()
{
    int p;
    double x,a,b,c,d;
    cout<<"  x          J0(x)          Y0(x)          J1(x)          Y1(x) \n";
    for (p=1;p<=9; p++)
    {
        x=((double)p)*1.0;
        a= NR::bessj0(x);
        b= NR::bessy0(x);
        c= NR::bessj1(x);
        d= NR::bessy1(x);
        cout<<setw(12)<<x<<setw(12)<<a<<setw(12)<<b;
        cout<<setw(12)<<c<<setw(12)<<d<<endl;
    }
}

```

```

}
gnuplot1(ff1,"J0(x)",13,ff2,"Y0(x)",13,ff3,"J1(x)",\
13,ff4,"Y1(x)",13,NULL);
return 0;
}
// FILE: mybess.cpp ends

```



**6.2. Elliptiset integraalit.** Monet mekaniikan ja tähtitieteen ongelmat johtivat 1700- ja 1800-luvuilla integraaleihin, joita ei voitu ilmaista alkeisfunktioiden avulla. Näin johduttiin uusiin funktioluokkiin mm. elliptisiin integraaleihin, joiden asema asteittain vakiintui ja joilla edelleen on tärkeä asema sähkömagneettikan kentälaskuissa. Monet näistä transkendenttifunktioista osoittautuivat niin tarpeelliseksi, että niiden arvot taulukoitiin käsikirjoihin. Nuo 1800-luvulla vuosikausien laskutyön vaatineet taulukot voitaisiin nykyään laskea muutamassa sekunnissa.

Esimerkkejä funktioluokista, jotka esiintyvät useissa eri yhteyksissä, ovat Gaussin hypergeometrisen funktion sekä elliptiset integraalit ja elliptiset funktiot. Mainittakoon, että mm. heilurin heilahdusajan kaava voidaan ilmaista elliptisten integraalien avulla, samoin ellipsin kaaren pituus. Elliptisten funktioiden teorian merkittävä kehittäjä oli Legendre, jonka tutkimukset johtivat mm. hänen nimeään kantavien esitysmuotojen löytymiseen.



Legendren ns. epätäydellisiä elliptisiä integraaleja ovat

$$E(\phi, k) = \int_0^\phi \sqrt{1 - k^2 \sin^2 \phi} d\phi = \int_0^\phi \frac{\sqrt{1 - k^2 y^2}}{\sqrt{1 - y^2}} dy$$

$$F(\phi, k) = \int_0^\phi \frac{d\phi}{\sqrt{1 - k^2 \sin^2 \phi}} = \int_0^y \frac{dy}{\sqrt{(1 - y^2)(1 - k^2 y^2)}} =$$

$$\int_0^x \frac{dx}{\sqrt{(1 + x^2)(1 + (1 - k^2)x^2)}},$$

$$y = \sin \phi, x = \tan \phi, 0 < k < 1, 0 < \phi \leq \pi/2.$$

Numeerisiin tarkasteluihin soveltuvat edellisiä paremmin ns. *Carlsonin symmetriset integraalit*

$$R_F(x, y, z), R_J(x, y, z, p), R_D(x, y, z), R_C(x, y),$$

1970-luvulta. Kirjamme toisessa painoksessa on Carlsonin algoritmit NR::rf, rj, rd, rc näiden laskemiseksi. Monet ohjelmakirjastot käyttävät juuri näitä algoritmeja myös elliptisten integraalien laskemiseen. Tähän tarkoitukseen tarvitaan seuraavat palautuskaavat Legendren integraaleille

$$F(\phi, k) = (\sin \phi) \text{NR}::\text{rf}(\cos^2 \phi, 1 - k^2 \sin^2 \phi, 1),$$

$$E(\phi, k) = (\sin \phi) \text{NR}::\text{rf}(\cos^2 \phi, 1 - k^2 \sin^2 \phi, 1) -$$

$$\frac{1}{3} k^2 (\sin^3 \phi) \text{NR}::\text{rd}(\cos^2 \phi, 1 - k^2 \sin^2 \phi, 1).$$

Erityisen tärkeitä ovat Legendren *ensimmäisen kertaluvun täydellinen elliptinen integraali*

$$K(k) = F\left(\frac{\pi}{2}, k\right) = \text{rf}(0, 1 - k^2, 1), k \in [0, 1],$$

ja *toisen kertaluvun täydellinen integraali*

$$E(k) = E\left(\frac{\pi}{2}, k\right) = \text{rf}(0, 1 - k^2, 1) -$$

$$\frac{1}{3} k^2 \text{rd}(0, 1 - k^2, 1), k \in [0, 1].$$

Gauss (1799) on todistanut, että  $K(k)$  voidaan laskea lukujen 1 ja  $k$  ns. aritmeettis-geometrisen keskiarvon  $ag(1, k)$  avulla seuraavasti. Asetetaan

$$a_0 = 1, b_0 = k, a_{n+1} = \frac{1}{2}(a_n + b_n), b_{n+1} = \sqrt{a_n b_n}, n = 0, 1, 2, \dots$$

Silloin lukujonojen  $(a_n)$  ja  $(b_n)$  yhteinen raja-arvo  $ag(1, k)$  toteuttaa Gaussin identiteetin

$$(*) \quad K(\sqrt{1 - k^2}) = \frac{\pi}{2ag(1, k)}.$$

B. C. Carlsonin töissä 1970-luvulla on Gaussin identiteetin yleistyksiä. Carlsonin algoritmit on otettu elliptisten integraalien numeerisen laskemisen perustaksi useissa ohjelmakirjastoissa, kuten NAG, IMSL ja SLATEC.

Seuraavasta ohjelmasta ilmenee miten em. funktioita käytetään. Tulosten oikeellisuuden tarkistamiseksi ohjelmassa tulostetaan viimeisessä sarakkeessa testisuure

$$E(k)K(k') + E(k')K(k) - K(k')K(k) - \frac{\pi}{2}; \quad k' = \sqrt{1 - k^2},$$

joka on identtisesti nolla *Legendren identiteetin* perusteella.

```
// FILE: myrf.cpp begins
/* g++ -Wall myrf.cpp -L../lib -I../utils -I../gnuplot02 -o a -lm -lnr */
// This program computes the values of the complete elliptic
// integrals K(k) and E(k). To check the correctness of
// the result, a test quantity is printed (it should vanish by
// Legendre's identity)
#include <cstdlib>
#include <cstdio>
#include <ctime>
#include <string>
#include <fstream>
#include <iostream>
#include <iomanip>
using namespace std;
#include "nr.h"

#define PRINT 1
#include "gnuplt1.h"
```

```

double ff1(double x)
{ return NR::elle(0.5*M_PI,x);}

double ff2(double x)
{ return NR::ellf(0.5*M_PI,x);}

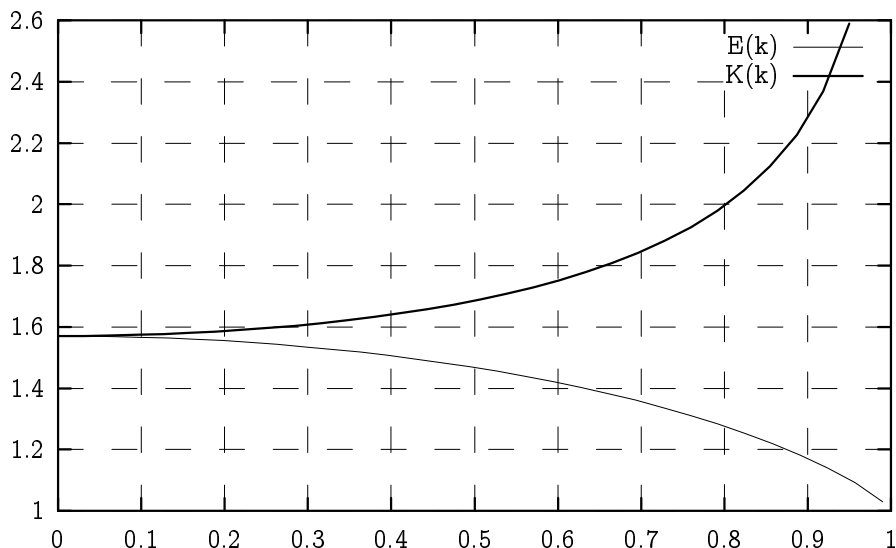
double LegId(double &x)
// Legendre's identity says that
//  $E K' + E' K - K K' - \pi/2 = 0$ 
{
    if (abs(x)>1-1e-10) x=(x/abs(x))*(1-1e-11);
    double E=ff1(x), E1=ff1(pow(1-x*x, 0.5)),
           K=ff2(x), K1=ff2(pow(1-x*x, 0.5));
    return E*K1+E1*K-K*K1-0.5*M_PI;
}

int main()
{
    int p;
    double x,r1,r2;
    cout<<"      k          E(k)          K(k) ";
    cout<<"      Legendre(k) \n";
    for (p=1;p<=9; p++)
    {
        x=p*0.1;
        r1=NR::ellf(0.5*M_PI,x);
        r2=LegId(x);
        cout<<setw(6)<<x<<setw(12)<<NR::elle(M_PI/2,x);
        cout<<setw(12)<<r1<<setw(16)<<r2<<endl;
    }
    gnuplt1(ff1,"E(k)",0,ff2,"K(k)",0,NULL);
    return 0;
}
// FILE: myrf.cpp ends

```

k	E(k)	K(k)	Legendre(k)
0.1	1.56686	1.57475	-9.51496e-16
0.2	1.55497	1.58687	7.5764e-16
0.3	1.53483	1.60805	-7.5287e-16
0.4	1.50594	1.64	4.13732e-16
0.5	1.46746	1.68575	3.49113e-16
0.6	1.41808	1.75075	9.15067e-17
0.7	1.35566	1.84569	-3.5757e-16
0.8	1.27635	1.9953	1.4615e-16

0.9      1.1717      2.28055      1.07119e-16



Erikoisfunktioiden alalta on laaja kirjallisuus. Eräs arvovaltainen alan esittely on teos G. Andrews, R. Askey, R. Roy: Special Functions, Encyclopedia of Mathematics and its Applications, Vol. 71, Cambridge U. Press, 1999, ISBN 0-521-78988-5. Erikoisfunktioiden numeeriseen laskentaan suuntautuneita ovat seuraavat teokset S. L. Moshier: Methods and Programs for Computing Mathematical Functions, Prentice Hall 1989, W. J. Thompson: Atlas for Computing Mathematical Functions, J. Wiley & Sons 1997, ISBN 0-471-00260-7. D. Lozier ja F. Olver ovat tehneet laajan katsauksen erikoisfunktioiden numeerisiin algoritmeihin, joka löytyy Internetistä,

<http://math.nist.gov/mcsd/Reports/2001/nesf/paper.pdf> .

Hyödyllisiä linkkejä on myös S. Finchin koostamalla Favorite Mathematical Constants www-sivulla

<http://pauillac.inria.fr/algo/bsolve/constant/constant.html> .

## 7 SATUNNAISLUVUISTA

Tietokoneella voidaan generoida ns. pseudosatunnaislukuja, joita jatkossa hieman epätarkasti kutsumme satunnaisluvuiksi. Oikeista satunnaisluvuihin ne eroavat sikäli, että ne ovat täysin deterministisesti tuotettuja jollakin satunnaislukugeneraattorilla. Niiden tilastolliset ominaisuudet noudattavat jotakin etukäteen annettua todennäköisyysjakaumaa tai tämä ainakin on tavoitteena. Satunnaislukugeneraattorin laatuksiteereitä:

- tehokkuus,
- miten hyvin generoitujen lukujen jakauma vastaa tavoitteena olevaa jakaumaa.

Satunnaislukujen avulla voimme tuottaa edustavia näytteitä jonkin systeemin syöteavaruudesta. Sellaista menettelyä, jossa tällaisia kokeiluja systemaattisesti tuotetaan ja jossa tuloksien pohjalta tehdään johtopäätöksiä tarkasteltavasta ilmiöstä, kutsutaan simuloinniksi. Simulointia käytetään erityisesti silloin, kun ilmiö on niin monimutkainen, ettei sitä pystytä toivotulla tarkkuudella kuvaamaan matemaattisella mallilla. Simulointia käytetään myös, jos halutaan tehdä numeerisia kokeita systeemeistä, joita ei todellisuudessa voida toteuttaa esim. kalliiden kustannusten vuoksi. Konkreettisenä esimerkkinä voidaan mainita puhelinverkkoliikenteen simulointi. Mielenkiinnon kohteena voi silloin toisaalta olla palvelun ja kapasiteetin parametrien tarkkailu, toisaalta systeemin mitoitukseen liittyvät parametrit.

Luvussa 1 esitimme muutamia pieniä simulointiohjelmia, mm. nopanheiton simulointiin. Satunnaisluvut generoitiin siellä C-kielen standardikirjastoon kuuluvan satunnaislukugeneraattorin avulla. NR esittää kritiikkiä satunnaislukugeneraattoreista ja luettelee joitakin perusteellisesti testattuja generaattoreita.

Yleistoteamuksena voi sanoa, että

- Generaattoreiden laatu vaihtelee merkittävästi.
- Generaattoreita kehitetään edelleen vilkkaasti.
- Implementaatiot riippuvat ohjelmointikielestä ja kääntäjästä.

- Useat generaattorit tuottavat jaksollisen lukujonon (jakso voi olla pitkäkin).

Esimerkkinä uudesta laajaa huomiota saaneesta generaattorista, ks. Mersenne twister

<http://www.math.keio.ac.jp/~matumoto/emt.html>

Todennäköisyyslaskennan perustulosten nojalla satunnaismuuttujan funktion jakauma voidaan palauttaa alkuperäiseen jakaumaan muuttujanvaihtokaavan avulla. Tämän vuoksi tasaisesta jakaumasta voidaan generoida sopivan muunnoksen avulla kaikki käytännössä tarvittavat jakaumat. Ks. erityisesti A. M. Law - W. D. Kelton: Simulation modeling and analysis, McGraw Hill, 2000, ISBN 0-07-059292-6, Luku 8.

**Lause.** Olkoot  $X, Y$  jatkuvia satunnaismuuttujia tiheysfunktiona  $f_{XY}(x, y)$  sekä  $G, H : R^2 \rightarrow R$  funktioita ja  $U = G(X, Y), V = H(X, Y)$  satunnaismuuttujia. Oletetaan, että yhtälöryhmästä

$$\begin{cases} G(x, y) = u \\ H(x, y) = v \end{cases}$$

voidaan  $x$  ja  $y$  ratkaista yksikäsitteisesti  $u$ :n ja  $v$ :n funktiona ja että joukossa, jossa  $f_{XY}(x, y) > 0$  pätee

$$\frac{\partial(G, H)}{\partial(x, y)} = \frac{\partial G}{\partial x} \frac{\partial H}{\partial y} - \frac{\partial G}{\partial y} \frac{\partial H}{\partial x} \neq 0.$$

Silloin satunnaismuuttujien  $U$  ja  $V$  yhteisjakauman tiheysfunktio on

$$f_{UV}(u, v) = f_{XY}(x, y) \left| \frac{\partial(G, H)}{\partial(x, y)} \right|^{-1}$$

missä  $(x, y)$  on em. yhtälöryhmän ratkaisu.

**Boxin-Müllerin muunnos.** Olkoot  $x_1, x_2$  riippumattomia satunnaismuuttujia, jotka noudattavat välin  $[0, 1]$  tasaista jakaumaa. Silloin muuttujat  $z_1, z_2$

$$z_1 = \sqrt{-2 \log x_1} \cos(2\pi x_2), \quad z_2 = \sqrt{-2 \log x_1} \sin(2\pi x_2)$$

ovat riippumattomia  $N(0, 1)$ -normaalijakautuneita satunnaismuuttujia. Tämä seuraa edellisestä valinnalla  $U = z_1, V = z_2$ .

NR:n ohjelma NR: : gasdev perustuu Boxin-Müllerin menetelmään. Demo-ohjelma xgavdev.cpp tuottaa seuraavan normaalijakautuman profiilin.

x	p(x)	graph:
0	0.0115	****
0.2	0.0176	*****
0.4	0.0213	*****
0.6	0.0291	*****
0.8	0.0403	*****
1	0.0479	*****
1.2	0.0591	*****
1.4	0.0674	*****
1.6	0.0769	*****
1.8	0.073	*****
2	0.0795	*****
2.2	0.0799	*****
2.4	0.0751	*****
2.6	0.0696	*****
2.8	0.0573	*****
3	0.0476	*****
3.2	0.0391	*****
3.4	0.0293	*****
3.6	0.0208	*****
3.8	0.0154	*****
4	0.0101	****

Seuraava ohjelma on xran.cpp /NR hieman muokattuna.

```

/* FILE: myrantst.cpp begins.                                     */
/* g++ -Wall myrantst.cpp -L../lib -I../gnuplot02 -o a -lm -lnr */

#include <cstdlib> // Used in putmat2
#include <cstdio> // Used in putmat2
#include <ctime>
#include <string>
#include <fstream>
#include <iostream>

```

```

#include <iomanip>
#include "nr.h"
#include <cmath>
#include "matut102.h"

using namespace std;

DP myrdm(int &seed)
{
    static int init=1;
    unsigned seed2=(seed>0)? seed:(-seed);
    if(init) { srand(seed2); init=0; }
    DP t=((DP)rand())/((DP)0x7fffffff); // 231
    return t;
}

// Driver for routines ran0, ran1, ran2, ran3
DP fnc(const DP x1, const DP x2, const DP x3, const DP x4)
{
    return sqrt(x1*x1+x2*x2+x3*x3+x4*x4);
}

void integ(DP func(int &))
{
    const unsigned long twotoj[16]={0x1L,0x2L,0x4L,0x8L,0x10L,
        0x20L,0x40L,0x80L,0x100L,0x200L,0x400L,0x800L,0x1000L,
        0x2000L,0x4000L,0x8000L};
    const DP PI=3.141592653589793238;
    int i,idum=(-1),j,jpower,k;
    DP x1,x2,x3,x4;
    Vec_INT iy(3);
    Vec_DP yprob(3);

    // Calculates pi statistically using volume of unit n-sphere
    for (i=0;i<3;i++) iy[i]=0;
    cout << "volume of unit n-sphere, n = 2, 3, 4" << endl;
    cout << "# points      pi      (4/3)*pi  (1/2)*pi^2";
    cout << endl << endl;
    cout << fixed << setprecision(6);
    for (j=0;j<15;j++) {
        for (k=twotoj[j];k>=0;k--) {
            x1=func(idum);
            x2=func(idum);
            x3=func(idum);
            x4=func(idum);

```



```

        if (fnc(x1,x2,0.0,0.0) < 1.0) ++iy[0];
        if (fnc(x1,x2,x3,0.0) < 1.0) ++iy[1];
        if (fnc(x1,x2,x3,x4) < 1.0) ++iy[2];
    }
    jpower=twotoj[j+1];
    for (i=0;i<3;i++)
        yprob[i]=DP(twotoj[i+2])*iy[i]/jpower;
    for (i=0;i<3;i++) yprob[i]=DP(twotoj[i+2])*iy[i]/jpower;
    cout << setw(6) << jpower << setw(12) << yprob[0];
    cout << setw(12) << yprob[1] << setw(12) << yprob[2] << endl;
}
cout << endl << "actual" << setw(12) << PI;
cout << setw(12) << 4.0*PI/3.0 << setw(12) << 0.5*PI*PI << endl;
cin.get();
}

```

```

int main(void)
{
    cout << endl << "Testing ran0:" << endl; integ(NR::ran0);
    cout << endl << "Testing ran1:" << endl; integ(NR::ran1);
    cout << endl << "Testing ran2:" << endl; integ(NR::ran2);
    cout << endl << "Testing ran3:" << endl; integ(NR::ran3);
    cout << endl << "Testing rand:" << endl; integ(myrdm);
    return 0;
}

```

/\* FILE: myrantst.cpp ends.

\*/

Testing ran0:

volume of unit n-sphere, n = 2, 3, 4

# points	pi	(4/3)*pi	(1/2)*pi <sup>2</sup>
2	2	0	0
4	3	4	8
8	2.5	4	6
.....			
4096	3.1084	4.10352	5
8192	3.10547	4.09766	4.96484
16384	3.11987	4.13916	4.97656
32768	3.12891	4.15674	4.92285
actual	3.14159	4.18879	4.9348

2	2	0	0
4	3	4	8
8	2.5	4	6

4096	3.1084	4.10352	5
8192	3.10547	4.09766	4.96484
16384	3.11987	4.13916	4.97656
32768	3.12891	4.15674	4.92285

4096	3.1084	4.10352	5
8192	3.10547	4.09766	4.96484
16384	3.11987	4.13916	4.97656
32768	3.12891	4.15674	4.92285

4096	3.1084	4.10352	5
8192	3.10547	4.09766	4.96484
16384	3.11987	4.13916	4.97656
32768	3.12891	4.15674	4.92285

4096	3.1084	4.10352	5
8192	3.10547	4.09766	4.96484
16384	3.11987	4.13916	4.97656
32768	3.12891	4.15674	4.92285

4096	3.1084	4.10352	5
8192	3.10547	4.09766	4.96484
16384	3.11987	4.13916	4.97656
32768	3.12891	4.15674	4.92285

4096	3.1084	4.10352	5
8192	3.10547	4.09766	4.96484
16384	3.11987	4.13916	4.97656
32768	3.12891	4.15674	4.92285

4096	3.1084	4.10352	5
8192	3.10547	4.09766	4.96484
16384	3.11987	4.13916	4.97656
32768	3.12891	4.15674	4.92285

4096	3.1084	4.10352	5
8192	3.10547	4.09766	4.96484
16384	3.11987	4.13916	4.97656
32768	3.12891	4.15674	4.92285

4096	3.1084	4.10352	5
8192	3.10547	4.09766	4.96484
16384	3.11987	4.13916	4.97656
32768	3.12891	4.15674	4.92285

Testing ran1:

volume of unit n-sphere, n = 2, 3, 4

# points	pi	(4/3)*pi	(1/2)*pi <sup>2</sup>
----------	----	----------	-----------------------

2	2	4	0
4	3	2	0
.....			
16384	3.12866	4.20264	4.93066
32768	3.14172	4.18262	4.90234
actual	3.14159	4.18879	4.9348

Testing ran2:  
volume of unit n-sphere, n = 2, 3, 4  
# points      pi      (4/3)\*pi      (1/2)\*pi^2

2	4	4	8
4	5	6	12
.....			
16384	3.1499	4.13135	4.79297
32768	3.15149	4.16797	4.87695
actual	3.14159	4.18879	4.9348

Testing ran3:  
volume of unit n-sphere, n = 2, 3, 4  
# points      pi      (4/3)\*pi      (1/2)\*pi^2

2	4	4	0
4	5	4	4
.....			
16384	3.15186	4.14941	4.82031
32768	3.13989	4.13086	4.854
actual	3.14159	4.18879	4.9348

Testing rand:  
volume of unit n-sphere, n = 2, 3, 4  
# points      pi      (4/3)\*pi      (1/2)\*pi^2

2	4	4	0
4	5	6	0
.....			
16384	3.14331	4.2085	4.91309
32768	3.13098	4.17725	4.92578
actual	3.14159	4.18879	4.9348

Ylläolevassa ohjelmassa vertaillaan eri generaattorien tuottamia

lukuja ja viimeisenä on mukana kurssin alusta käytössä generaattori rand.

Internetissä on saatavilla useita satunnaislukujen generointiohjelmia. Mainittakoon esim. RANLIB-kirjasto, joka löytyy osoitteesta

<http://netlib.bell-labs.com/netlib/random/index.html> .

**file: ranlib.c.tar.gz**

*for: random number generation from the distributions: beta, chi-square, exponential, F, gamma, multivariate normal, noncentral chi-square, noncentral F, univariate normal, uniform, binomial, negative binomial, multinomial, Poisson; also: random permutations of an integer array. by: Barry W. Brown, James Lovato, Kathy Russell alg: (base) L'Ecuyer and Cote, ACM TOMS 17:98-111 (1991) lang: C*

**file: dcdflib.c.tar.gz**

*for: cumulative distribution functions, inverses, and parameters for: beta, binomial, chi-square, noncentral chi-square, F, noncentral F, gamma, negative binomial, normal, Poisson, Student's t. alg: various TOMS algorithms and Abramowitz and Stegun; Bux Dekker zero-finding by: Barry W. Brown, James Lovato, Kathy Russell lang: C*

Em. RANLIB-kirjastoa ei pidä sekoittaa Unixin samannimiseen käskyyn jota käytetään ohjelmakirjastojen laatimisessa.

Muuta kirjallisuutta:

L. Devroye: Non-uniform random variate generation, Springer-Verlag, 1986

D. Knuth: The art of computer programming, Vol. II.

## 8 LAJITTELUMENETELMISTÄ

NR:n käsittelemiä aiheita:

- Lukujonon järjestys suuruusjärjestykseen.
- Lukujonon indeksitaulun laatiminen: ts. monesko alkio on pienin, toiseksi pienin jne.
- Lukujonon rankkeeraus: etsitään kunkin alkion arvojärjestys.

### 8.1. Algoritmeja.

Suora lomitus (merge) on toteutettu algoritmina NR::piksrt. Käytetään tavanomaista korttipakan järjestysmenetelmää.

-järjestetään jonon 2 ensimmäistä alkioita

-lisätään 3. alkio mukaan oikealle paikalleen

-jatketaan kunnes koko jono käsitelty. Kompleksisuus  $O(N^2)$

Kekolajittelu (heapsort, perustuu prioriteettijonon eli keon käyttöön)

NR::sort.

Indeksitaulun muodostus

Ohjelma xindexx.cpp tarvitsee syötetiedoston arrin

Tuloste: NR::indx jolla NR::arrin(indx(J)),  $J=1, \dots, N$  kasvavassa järj.

Pikalajittelu NR::qcksrt (Bubble sort: NR:n mukaan tehoton ja mielenkiinnoton)

```
/* FILE: xindexx.cpp begins */
/* Driver for routine indexx */
#include <string>
#include <fstream>
#include <iostream>
#include <iomanip>
#include "nr.h"
#include "print_array.h"
using namespace std;
// Driver for routine indexx
int main(void)
```

```

{
    const int NP=100;
    string txt;
    int i,j;
    Vec_INT indx(NP);
    Vec_DP a(NP);
    ifstream fp("tarray.dat");
    if (fp.fail())
        NR::nrerror("Data file tarray.dat not found");
    getline(fp,txt);
    for (i=0;i<NP;i++) fp >> a[i];
    NR::indexx(a,indx);
    cout << endl << "original array:" << endl;
    cout << fixed << setprecision(2);
    print_array(a,10,7);
    cout << endl << "sorted array" << endl;
    for (i=0;i<10;i++) {
        for (j=0;j<10;j++) cout << setw(7) << a[indx[10*i+j]];
        cout << endl;
    }
    return 0;
}

```

```

}
original array:
  30   72   3.3   87   53   63   89   26   93   28
  72   48   53   18   27   60   83   23   67   53
  53   15    8   53   76   79   68   38   25   73
  13   52   35  1e+02  38   82   62   80   93   3.2
  99   92   94    7   6.7   89   83    9   13   62
    3   85   96   74   49   78   37   3.5   49   72
  1.4  9.5   32   90   28   79   54   46   12   38
  77   74   10   4.6   50   68   19   34   4.1   98
  42   64   89   53   72   3.9   20   45   71   59
  28   16   68   56   26   25   82   90   57   38

```

```

sorted array
  1.4    3   3.2   3.3   3.5   3.9   4.1   4.6   6.7    7
    8    9   9.5   10   12   13   13   15   16   18
   19   20   23   25   25   26   26   27   28   28
   28   30   32   34   35   37   38   38   38   38
   42   45   46   48   49   49   50   52   53   53
   53   53   53   53   54   56   57   59   60   62
   62   63   64   67   68   68   68   71   72   72
   72   72   73   74   74   76   77   78   79   79
   80   82   82   83   83   85   87   89   89   89
   90   90   92   93   93   94   96   98   99  1e+02

```

## 9 EPÄLINEAARISET YHTÄLÖRYHMÄT

Algebran peruslauseen mukaan astetta  $p$  olevalla polynomilla on täsmälleen  $p$  juurta kompleksitasossa. Jos  $p = 2, 3, 4$ , niin juuret voidaan esittää algebrallisina lausekkeina kertoimista, mutta vastaava esitys ei enää ole mahdollista millekään asteelle  $p \geq 5$ . Yhtälön ratkaisu johtaa tavallisesti tilanteeseen, jossa voidaan keilla numeerisen ratkaisumenetelmän soveltuvuutta. Tässä luvussa luodaan katsaus niihin menetelmiin, joita oppikirjamme NR tarjoaa epälineaarisen yhtälön tai yhtälöryhmän ratkaisemiseen.

Aihepiiriä on tutkittu satoja vuosia. 1600-luvulta on peräisin Newtonin tai Newtonin-Raphsonin menetelmä, jonka suppenemisaralue on viime vuosiin saakka pysynyt mysteerinä. Vasta ns. kompleksidynamiikan tutkimus 1980-luvulta alkaen on tuonut asiaan uutta näkemystä. On osoittautunut, että suppenemisaralue on geometriseltä struktuuriltaan varsin monimutkainen, sen reuna on ns. fraktaalijoukko. Kysymys siitä suppeneeko iteraatio annetulla alkuarvolla on edelleen avoin.

Olkoon  $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$  jatkuva funktio ja  $m, n \geq 1$ .

**Perustehtävä:** Etsi  $x \in \mathbb{R}^n$  s.e.  $f(x) = 0$ .

**Mahdollisia vaikeuksia:**

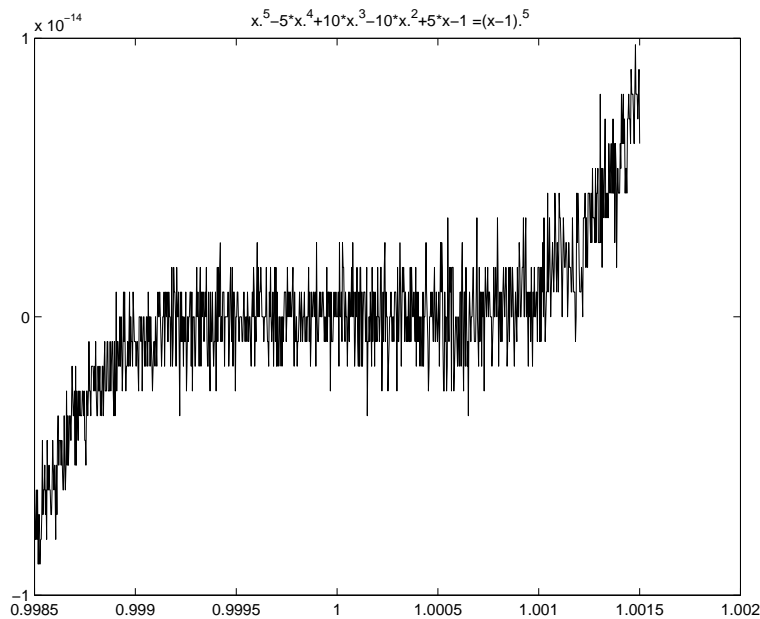
-Reaalisen ratkaisun olemassaolo epävarmaa.

-Yksikäsitteisyys voi puuttua tai olla vain lokaalia. Mistä juurta etsitään? Mistä saadaan hyvä alkuarvaus?

-Tapaus  $m, n \geq 2$  on vaikeampi kuin tapaus  $m = n = 1$ . Jos  $n = m = 1$ , niin juuri voidaan löytää paikallistamalla (bracket) se ensin välille  $(x_1, x_2)$  s.e.  $f(x_1)f(x_2) < 0$ , ja sitten iteroimalla esim. välinpuolitusmenetelmällä.

-Jos  $f$  on epälineaarinen niin ratkaisumenetelmä on aina iteratiivinen. Miten löytyy alkuarvaus?

-Juurien multiplisiteetti. Jos  $n = m = 1$  ja multiplisiteetti parillinen, niin juurien paikallistaminen merkinvaihtovälille tulee ongelmalliseksi. Esim. funktiolla  $F(x) = x^2$  on juuri välillä  $(-1, 1)$ , vaikka se ei vaihdakaan merkkiään tällä välillä.



-Juuren "sumeus" numeerisessa mielessä muistettava tarkkuuskriteereitä määrättäessä.

**Huom.** Yhteys minimointiongelmaan: Yhtälön  $f(x) - y = 0$  ratkaisu on sama kuin funktion  $(f(x) - y)^2 = (f(x) - y)^T (f(x) - y)$  minimi.

**Menetelmät:**

(a) **Brentin menetelmä:** Annettu yhden muuttujan funktio, tiedetään nollakohdan sijaintiväli, etsitään juuri. Ei tarvita derivaattaa. (1970-luku)

(b) **Newton(-Raphson):** Kuten yllä, mutta tarvitaan derivaattaa.

(c) **Muita menetelmiä:**

- Välinpuolitusmenetelmä, arvojen taulukointi
- Sekanttimenetelmä
- Polynomifunktiolle sopivat menetelmät

(d) **Graafinen ratkaisu:** Kätevä esim. alkuarvauksen saamiseen muille menetelmille. Algoritmi NR: :scrsho.

**9.1. Juurien paikallistaminen ja välinpuolitusmenetelmä.** Analyysin peruskurssilla todistetaan seuraava lause.

**Rolle'n lause.** Olkoon  $f$  jatkuva välillä  $[a, b]$  ja  $f(a)f(b) < 0$ . Tällöin  $f$ :llä on välillä  $[a, b]$  nollakohta.

Juuren haun kannalta hankala testiongelmaksi on seuraava: Ratkaise  $f(x) = 0$ ;  $f(x) = 3x^2 + \frac{1}{\pi^4} \log[(\pi - x)^2] + 1$  (2 juurta välillä  $\pi \pm 10^{-667}$ ).

NR:n algoritmit NR: :zbrac ja NR: :zbrak on tarkoitettu paikallistamaan juuri. (Tuskin nekään toimivat yo. funktion tapauksessa.)

NR: :zbrac laajentaa alkuperäistä väliä s. 352 (NR: :zbrak pienentää).

**Välinpuolitusmenetelmä.** Olkoon  $f$  jatkuva välillä  $[a, b]$  ja  $f(a)f(b) < 0$ ,  $c = (a + b)/2$ . Jos  $f(a)f(c) < 0$  niin korvataan  $b$   $c$ :llä, muuten  $a$   $c$ :llä. Iteroidaan.

Tämä menettely oli esillä jo Luvun I algm:ssa mybisect.cpp. Palautetaan sieltä mieleen, että algm soveltui myös käänteisfunktion arvojen laskemiseen.

**9.2. Sekanttimenetelmä ja regula falsi.** Regula falsi menetelmää kutsutaan joskus myös false position menetelmäksi.

Sekanttimenetelmä että regula falsi kumpikin olettavat, että funktion nollakohta on paikallistettu välille  $(a, b)$  ja vielä funktiota approksimoidaan lineaarisella funktiolla, jonka leikkauspiste  $x$ -akselin kanssa antaa juurelle likiarvon  $c$ . Kunkin iteraation aikana yksi aikaisempi piste "unohdetaan".

*Ainoa ero:* Regula falsi säilyttää aikaisemmista pisteistä sen, jolla on erimerkkinen arvo  $f(c)$ :n kanssa [jolloin nollakohta on paikallistettu välille  $(a, c)$  tai  $(c, b)$ ], kun taas sekanttimenetelmä säilyttää



viimeksi löydetyn pisteistä  $a$  ja  $b$  [sekanttimenetelmässä juuren ei siis tarvitse olla paikallistettu uudella välillä].

**Esim.** Oletetaan, että  $f$  on kuten NR:n s. 355 olevassa kuvassa 9.2.2 ja sekanttimenetelmä on tuottanut pisteet  $x_1 < x_2$  (ts. ensin löydetty  $x_1$ , sitten  $x_2$ )

Tällöin sekanttimenetelmä antaa uudeksi väliksi  $(x_3, x_2)$ , joka ei sisällä nollakohtaa (suppeneminen epävarmaa).

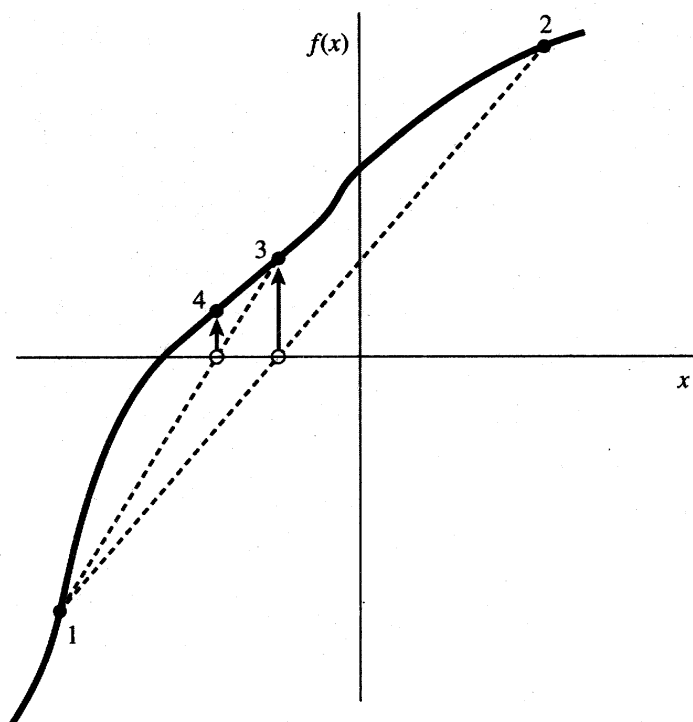


Figure 9.2.2. False position method. Interpolation lines (dashed) are drawn through the most recent points that bracket the root. In this example, point 1 thus remains "active" for many steps. False position converges less rapidly than the secant method, but it is more certain.

Algoritmi NR::rtflsp s. 356 ja NR::rtsec.

Supetessaan sekä sekanttimenetelmä että regula falsi ovat hitaita, mutta yleensä molemmat ovat parempia kuin välinpuolitusmenetelmä.

Esimerkki, jossa regula falsi on hidas on annettu kirjan sivulla 356 olevassa Kuvassa 9.2.3.

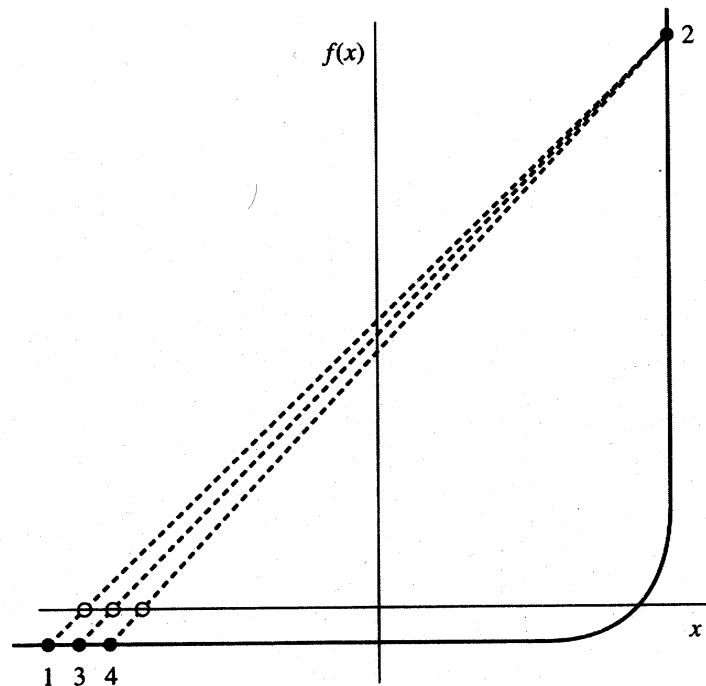


Figure 9.2.3. Example where both the secant and false position methods will take many iterations to arrive at the true root. This function would be difficult for many other root-finding methods.

**9.3. Brentin menetelmä.** On esimerkkejä, joissa sekanttimenetelmä ja regula falsi ovat hitaampia kuin välinpuolitusmenetelmä (mm. eräät sileät funktiot, joiden 2. derivaatta muuttuu sopivasti nollakohdan lähellä). Ns. **Brentin menetelmä** (v. 1973) yhdistää näiden kolmen menetelmän hyvät puolet. Brentin menetelmän piirteet:

- juuren paikallistus
- välinpuolitus
- *käänteinen kvadraattinen interpolointi*

Sekanttimenetelmä ja regula falsi approksimoivat funktiota lineaarisesti kahden juurta rajoittavan luvun välillä, kun taas käänteinen kvadraattinen interpolointi käyttää kolmea aikaisempaa pistettä kvadraattisen funktion sovittamiseen niiden kautta ja uuden likiarvon löytämiseen.

Jo löydettyihin pistepareihin  $a, f(a), b, f(b)$  ja  $c, f(c)$ , sovitetaan Lagrange'n interpolointipolynomi  $g$  vaihtamalla  $x, y$ -akselit

keskenään. Silloin  $g(0)$  on uusi approksimaatio juurelle ja Lagran-  
gen kaavan nojalla sille saadaan eksplisiittinen lauseke lukujen  $a$ ,  $f(a)$ ,  
 $b$ ,  $f(b)$  ja  $c$ ,  $f(c)$  avulla.

Brentin menetelmä antaa juuren ”likiarvoiksi”  $g(0)$ :n mikäli  $g(0)$   
on sillä välillä johon juuri on suljettu ja mikäli väli kutistuu kyllin  
nopeasti. Jos ei, otetaan  $g(0)$ :n asemasta välinpuolitusaskel.

#### 9.4. Newtonin menetelmä. Käytetään approksimointia

$$f(x + \delta) = f(x) + f'(x)\delta + \frac{f''(x)}{2}\delta^2.$$

Kun  $\delta^2$ :n sisältävä termi unohdetaan, niin vaatimus  $f(x + \delta) = 0$   
johtaa lausekkeeseen  $\delta = -\frac{f(x)}{f'(x)}$ , jolloin  $x + \delta$  on juuren approksi-  
maatio.

**Newtonin algoritmi.** Kiinnitetään  $x_0$  ja asetetaan

$$x_{i+1} = x_i - f(x_i)/f'(x_i), \quad i = 1, 2, 3, \dots$$

Oletetaan, että  $f'(x) \neq 0$ . Algoritmi NR::rtnewt .

**Newtonin algoritmi vaimennuksella.** Kuten edellä, mutta hyväksytään  
 $x_{i+1}$  vain, jos  $|f(x_{i+1})| < |f(x_i)|$ . Muussa tapauksessa puolitetaan  
askelpituutta  $\delta$  kunnes tämä ehto tulee voimaan.

Newtonin menetelmää ja sen erästä pulmatilannetta havainnol-  
listavat NR:n sivulla 363 olevat kuvat 9.4.1 ja 9.4.2 jotka ovat myös  
oheisina.

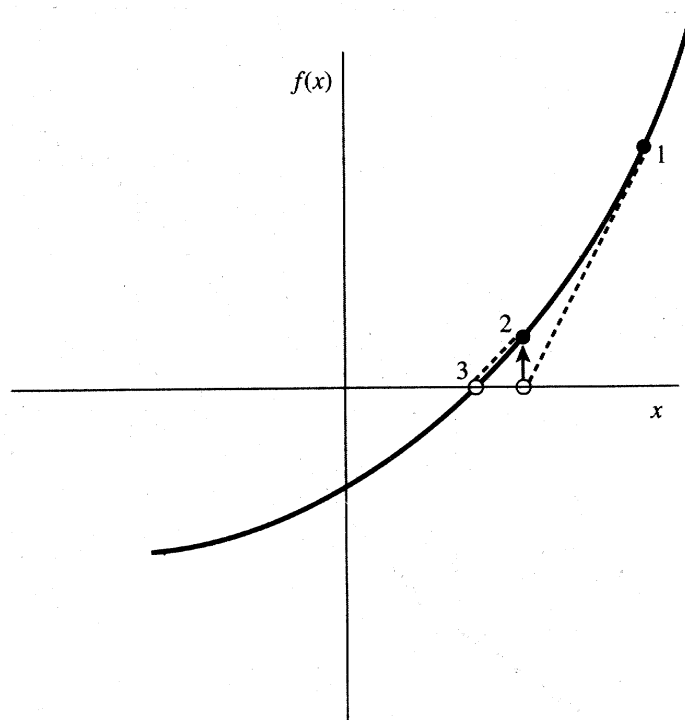


Figure 9.4.1. Newton's method extrapolates the local derivative to find the next estimate of the root. In this example it works well and converges quadratically.

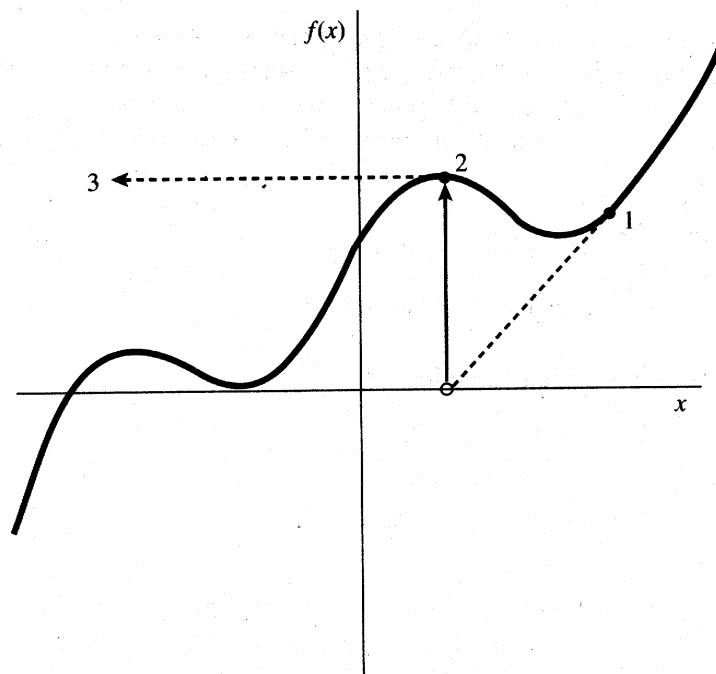


Figure 9.4.2. Unfortunate case where Newton's method encounters a local extremum and shoots off to outer space. Here bracketing bounds, as in `rtsafe`, would save the day.

Algoritmi NR::rtsafe sisältää lisäpiirteitä: se estää uusien askeleiden "karkaamisen" väliltä, jolla nollakohtaa etsitään.

```
// FILE: mysafe.cpp begins
#include <iostream>
#include <iomanip>
#include "nr.h"
using namespace std;

// Driver for routine rtsafe

DP fx(const DP x)
{
    return exp(x)-3*x*x;
}

void funcd(const DP x,DP &fn, DP &df)
{
    fn=exp(x)-3*x*x;
    df=exp(x)-6*x;
}

int main(void) {
    const int N=100,NBMAX=20;
    const DP X1=1.0,X2=50.0;
    int i,nb=NBMAX;
    DP xacc,root;
    Vec_DP xb1(NBMAX),xb2(NBMAX);

    NR::zbrak(fx,X1,X2,N,xb1,xb2,nb);
    cout << endl << "Roots of f(x):" << endl;
    cout << setw(20) << "x" << setw(16) << "f(x)" << endl;
    cout << fixed << setprecision(6);
    for (i=0;i<nb;i++)
    {
        xacc=(1.0e-6)*(xb1[i]+xb2[i])/2.0;
        root=NR::rtsafe(funcd,xb1[i],xb2[i],xacc);
        cout << "root" << setw(4) << (i+1) << setw(15) << root;
        cout << setw(15) << fx(root) << endl;
    }
    return 0;
}
// FILE: mysafe.cpp ends
```

Roots of  $f(x)$ :

		x	f(x)
root	1	3.73308	5.73975e-11

Keskeinen kysymys Newtonin iteraatiossa on, miten löydetään sellainen alkupiste  $x_0$ , josta iteraatio suppenee kohti juurta.

Hyvä alkuarvaus saadaan esim. graafisesti. Joskus voidaan soveltaa seuraavaa lausetta.

**Lause.** Olkoon  $f$  kahdesti jatkuvasti derivoituva suljetulla välillä  $[a, b]$  ja olkoot seuraavat ehdot voimassa:

- (1)  $f(a)f(b) < 0$ ,
- (2)  $f'(x) \neq 0$ , kun  $x \in [a, b]$ ,
- (3)  $f''(x)$  on joko  $\geq 0$  tai  $\leq 0$ , kun  $x \in [a, b]$ ,
- (4)

$$\frac{|f(a)|}{|f'(a)|} < b - a \quad \text{ja} \quad \frac{|f(b)|}{|f'(b)|} < b - a.$$

Tällöin Newtonin menetelmä suppenee kohden yhtälön  $f(x) = 0$  yksikäsitteistä juurta kaikilla alkuarvoilla  $x_0 \in [a, b]$ .

**Esim.** Olkoon  $f(x) = \arctan x$ . Silloin yhtälöllä  $f(x) = 0$  on juuri  $x = 0$ . Newtonin iteraatio on nyt  $x_{k+1} = x_k - (1 + x_k^2) \arctan x_k$ . Jos  $x_0$  valit. s.e.  $\arctan |x_0| > \frac{2|x_0|}{1+|x_0|^2}$ , niin jono  $\{x_k\}$  hajaantuu. Allaolevasta kaaviosta ilmenee Newtonin algoritmin käyttäytyminen eri alkuarvoilla:

0	8.0000e-01	1.0000e+00	1.2000e+00	1.4000e+00	1.6000e+00
1	-3.0658e-01	-5.7080e-01	-9.3758e-01	-1.4136e+00	-2.0034e+00
2	1.8862e-02	1.1686e-01	4.7772e-01	1.4501e+00	3.5509e+00
3	-4.4737e-06	-1.0610e-03	-6.9652e-02	-1.5506e+00	-1.4090e+01
4	5.9690e-17	7.9631e-10	2.2505e-04	1.8471e+00	2.8520e+02
5	0.0000e+00	0.0000e+00	-7.5990e-12	-2.8936e+00	-1.2720e+05
6	0.0000e+00	0.0000e+00	0.0000e+00	8.7103e+00	2.5415e+10
7	0.0000e+00	0.0000e+00	0.0000e+00	-1.0325e+02	-1.0146e+21

Taulukon ensimmäinen sarake ilmoittaa iteraatiokierroksen numeron 0 – 7 ja ylin vaakarivi antaa alkuarvot. Havaitaan, että alkuarvoilla 0.8, 1.0, ja 1.2 algoritmi suppenee kohti haluttua juurta, kun taas alkuarvoilla 1.4 ja 1.6 se hajaantuu.

**Esim.** Seuraava esimerkki selvittää Newtonin iteraation suppenemista ko. funktioille koodista ilmenevin alkuarvoin.

```
// FILE : mynewt.cpp begins
// Solves f(x)=0 by Newton's method for several
// functions f.
#include<cmath>
#include<iostream>
#include<iomanip>

using namespace std;

double fx(int n, double x)
{
    double t;
    switch (n)
    {
        case 1:{t= sin(x)-x*x*x-1.0; break;}
        case 2:{t= x-log(x)-3.0; break;}
        case 3:{t= x*x*x-x-1.0; break;}
        case 4: {t= sin(x)-x-2.0; break;}
        case 5: {t= pow(x+1.0,5.0)*(x-1.0); break;}
        default: t=1.0;
    }
    return t;
}

double dfx(int n, double x)
{
    double t;
    switch (n)
    {
        case 1: {t= cos(x)-3*x*x; break;}
        case 2:{t= 1.0-1.0/x; break;}
        case 3: {t= 3*x*x-1.0; break;}
        case 4: {t= cos(x)-1.0; break;}
        case 5:{t= pow(x+1.0,5.0)+(x-1.0)*5*pow(x+1.0,4.0); break;}
        default: t=1.0;
    }
}
```

```

    return t;
}

int main()
{
    int i,n;
    double x, xsta[] ={-1.0, 2.0, 1.5, 2.5, 0.5};
    for (n=0;n<5;n++) {
        x=xsta[n];
        cout<<"Function "<<n+1<<" :-----\n";
        for (i=0;i<6;i++) {
            cout.setf(ios::scientific);
            cout<<setw(14)<<x<<" "<<setw(14)<<fx(n+1,x)<<endl;
            x=x-fx(n+1,x)/dfx(n+1,x);
        }
    }
    return 0;
}
// FILE : mynewt.cpp ends

```

```

Function 1 :-----
-1.000000e+00  -8.414710e-01
-1.342103e+00  4.434889e-01
-1.256439e+00  3.246611e-02
-1.249104e+00  2.279121e-04
-1.249052e+00  1.151122e-08
-1.249052e+00  1.218643e-16
Function 2 :-----
2.000000e+00  -1.693147e+00
5.386294e+00  7.024367e-01
4.523714e+00  1.438074e-02
4.505252e+00  8.350548e-06
4.505241e+00  2.837730e-12
4.505241e+00  4.440892e-16
Function 3 :-----
1.500000e+00  8.750000e-01
1.347826e+00  1.006822e-01
1.325200e+00  2.058362e-03
1.324718e+00  9.243778e-07
1.324718e+00  1.866847e-13
1.324718e+00  1.376937e-16
Function 4 :-----
2.500000e+00  -3.901528e+00

```



```

3.338608e-01 -2.006168e+00
-3.599939e+01 3.499109e+01
-4.994674e+00 3.955095e+00
4.874778e-01 -2.019079e+00
-1.684621e+01 1.575411e+01
Function 5 :-----
5.000000e-01 -3.796875e+00
-2.500000e-01 -2.966309e-01
-4.204545e-01 -9.286774e-02
-5.466622e-01 -2.961456e-02
-6.429758e-01 -9.530584e-03
-7.176249e-01 -3.083615e-03

```

Kirjoittamalla  $\varphi(x) = x - f(x)/f'(x)$  huomataan, että Newtonin iterointi on esimerkki kiintopisteiteroinneista, joista jo edellä Luvuissa 1 ja 2 oli puhetta. Lisäksi  $\varphi'(x) = f(x)f''(x)/(f'(x))^2$ .

**Lause.** Olkoon  $f : [a, b] \rightarrow \mathbb{R}$  kahdesti jatkuvasti differentioituva. Oletamme, että on olemassa vakio  $c \in (0, 1)$ , jolle kaikilla  $x \in [a, b]$  pätee

$$\left| \frac{f(x)f''(x)}{(f'(x))^2} \right| \leq c < 1.$$

Silloin Newtonin iteraatio suppenee kaikilla alkuarvoilla  $x_0 \in [a, b]$ .

**Lause.** Olkoon  $f : [a, b] \rightarrow \mathbb{R}$  kahdesti jatkuvasti differentioituva. Oletamme, että  $f''(x) > 0$  välillä  $[a, b]$  ja lisäksi  $f(a) > 0, f(b) \leq 0$ . Silloin Newtonin iteraation muodostama pistejono alkuarvolla  $x_0 = a$  suppenee monotonisesti kohti juurta.

### Yhteenveto Newtonin menetelmästä.

- Iteraation askeleet määräytyvät käyrälle piirrettyjen tangenttien leikkauspisteestä  $x$ -akselin kanssa.
- Hyvä alkuarvaus on tarpeen. Muuten iteraatio voi hajaantua tai supeta kohden lukua, joka ei ole juuri. Jopa pieni muutos alkuarvoon voi vaikuttaa oleellisesti algoritmin käyttäytymiseen.
- Jos menetelmä suppenee, niin konvergenssi on tavallisesti varsin nopeaa.

- Soveltuu myös kompleksijuuriin etsimiseen. Luokittelu sup-peneviin ja hajaantuviin alkuarvoihin on vaikeaa, vrt. NR:n sivulla 368 oleva Kuva 9.4.4.

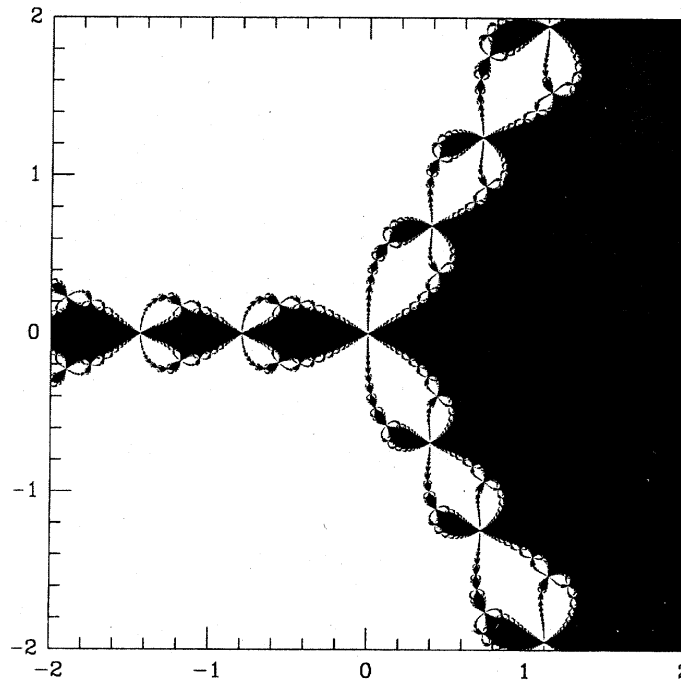


Figure 9.4.4. The complex  $z$  plane with real and imaginary components in the range  $(-2, 2)$ . The black region is the set of points from which Newton's method converges to the root  $z = 1$  of the equation  $z^3 - 1 = 0$ . Its shape is fractal.

**9.5. Polynomien juuret.** Perusideoita polynomiyhtälön ratkaisussa ovat seuraavat.

- $n$  asteen polynomilla on  $n$  juurta (multiplisiteetti).
- Jos  $a + ib$  on juuri, niin myös  $a - ib$  on juuri.
- Jos on löydetty polynomien  $P$  juuri  $x_0$ , niin  $P = (x - x_0)Q(x)$  ja tekijä  $(x - x_0)$  voidaan jakaa pois.

Algoritmi NR::zroots etsii polynomien  $\sum_{i=1}^{n-1} A(i)x^{i-1}$  kaikki nol-lakohdat.

Algoritmi NR::zroots:n toiminnassa on 2 vaihetta

- Etsitään alustavat juuret.
- Parannetaan alustavien juurien tarkkuutta.

**Juurien sijainti kompleksitasossa.** Jos polynomi  $P_n(x) = x^n + a_{n-1}x^{n-1} + \dots + a_1x + a_0$  on reaalikertoiminen ja  $|a_j| \leq M$  kaikilla  $j$ , niin  $P_n$ :n nollakohdat sijaitsevat tason  $\mathbb{C}$  ympyrässä  $\{z : |z| < M + 1\}$ . Jos lisäksi tiedetään, että kaikki 0-kohdat ovat reaalisia niin ne kaikki ovat yhtälön  $nx^2 + 2a_{n-1}x + (2(n-1)a_{n-2} - (n-2)a_{n-1}^2) = 0$  ratkaisujen määrämällä reaaliakselin välillä (ns. Laguerren lause).

## 9.6. Newtonin menetelmä usealle yhtälölle.

Yhtälöryhmän

$$(1) \quad \begin{cases} f(x, y) = 0 \\ g(x, y) = 0 \end{cases}$$

ratkaisua varten on etsittävä pistejoukkojen  $f^{-1}(0)$  ja  $g^{-1}(0)$  kaikki topologiset komponentit. Samoin kuin yhden muuttujan tapauksessa, hyvä alkuarvaus on tärkeä Newtonin algoritmille myös korkeammassa dimensioissa. Newtonin algoritmia koskeva tutkimus on laajaa. Eräs tutkimuksen suunta liittyy algoritmin suppenemisminimaisuuksien parantamiseen, algoritmia sopivasti modifioimalla. Tätä ideaa on noudatettu ns. globaalisti suppenevassa Newtonin algoritmissa, joka tulee alempana esille.

Yleinen epälineaarinen yhtälöryhmä on muotoa

$$f_i(\bar{X}) = 0, \quad \bar{X} = (x_1, \dots, x_N), \quad i = 1, \dots, N$$

Taylorin kaava antaa approksimaation

$$f_i(X + \delta X) = f_i(X) + \sum_{j=1}^N \frac{\partial f_i}{\partial x_j} \delta x_j + O(\delta X^2).$$

Unohdetaan  $O(\delta X^2)$ -termi ja vaaditaan  $f_i(X + \delta X) = 0$ . Saadaan:

$$(2) \quad \begin{cases} \sum_{j=1}^N \alpha_{ij} \delta x_j = \beta_i = -f_i, \\ x_i^{(p+1)} = x_i^{(p)} + \delta x_i, \quad i = 1, \dots, N. \end{cases}$$

Tässä  $p$  on iteraatioaskeleen numero. Luvut  $\delta x_i$  saadaan ratkaisemalla (2) LU-menetelmällä. Algoritmi NR::mnewt (s. 381).

Kuten Kohdan 9.4 lopussa annettu esimerkki osoittaa, hyvä alkuarvaus on oleellinen Newtonin menetelmälle.

**Lause** (Newton-Kantorovitsh) Olkoon  $B^n(r) = \{x \in \mathbb{R}^n : |x| < r\}$  ja  $f : B^n(1) \rightarrow \mathbb{R}^n$  jatkuvasti differentioituva ja seuraavat ehdot voimassa

$$(a) \|Df(x) - Df(y)\| \leq \gamma \|x - y\| \quad \forall x, y \in B^n(1)$$

$$(b) \|Df(0)^{-1}f(0)\| \leq \alpha,$$

$$(c) \|Df(0)^{-1}\| \leq \beta.$$

Merkitään

$$h = \alpha\beta\gamma \quad r_{1,2} = \frac{1 \mp \sqrt{1 - 2h}}{h}\alpha$$

Jos  $h \leq 1/2$  ja  $r_1 < 1$  niin jono

$$x_{k+1} = x_k - Df(x_k)^{-1}f(x_k), \quad x_0 = 0, \quad k = 0, 1, 2, \dots$$

pysyy joukossa  $B^n(r_1)$  ja konvergoi kohti  $f$ :n 1-käsitteistä juurta joukossa  $B^n(r_2)$ .

Tod. Ks. esim. Stoer-Bulirsch.

**9.7. Esimerkkejä Newton-iteroinnista.** Soveltamalla Newtonin menetelmää  $x_{n+1} = x_n - f(x_n)/f'(x_n)$  tapauksiin  $f(x) = 1/x - a$ ,  $x^2 - a$ ,  $x^p - a$ , saadaan käänteisluvun laskemiselle ja juurenotolle seuraavat algoritmit:

$$x_{n+1} = x_n(2 - ax_n) \rightarrow 1/a, \quad 0 < x_0 < 2/a,$$

$$x_{n+1} = \frac{1}{2}(x_n + a/x_n) \rightarrow \sqrt{a}, \quad x_0 > 0,$$

$$x_{n+1} = \left(1 - \frac{1}{p}\right)x_n + \frac{a}{px_n^{p-1}} \rightarrow^p \sqrt[p]{a}, \quad x_0 > 0 \quad .$$

**9.8. Lopetuskriteereistä.** Virheraja, toleranssi, voidaan valita esim. vaatimalla, että suhteellinen virhe  $\leq 10^{-6}$ .

Huomaa, että nollan lähellä suhteellinen virhe voi olla ongelmallinen nimittäjässä mahdollisesti olevan nollan vuoksi.

Muita mahdollisia kriteereitä: virhe  $< \varepsilon \cdot \frac{|x_1|+|x_2|}{2}$ , missä  $x_1, x_2$  paikallistavat juuren ja  $\varepsilon = 10^{-6}$ .

Lopetusehdoksi voidaan valita myös yläraja iteraatioiden lukumäärälle.

**9.9. Globaalisti suppeneva Newtonin menetelmä.** Edellä on jo yhden muuttujan tapauksessa tullut esille Newtonin menetelmän virhealtius iteraation alkupisteen valinnalle. Newtonin menetelmää voidaan hieman korjata niin, että saatu menetelmä suppenee lähes kaikilla alkuarvoilla. Oppikirjassamme on tällainen korjattu Newtonin menetelmä, ns. *globaalisti suppeneva* menetelmä. Lähtökohtana on korjata Newtonin menetelmää niin, että otetaan Newton-askel vain mikäli tarkasteltavan funktion normi vähenee siirryttäessä uuteen pisteeseen ja muulloin edetään Newtonin askeleen suunnassa lyhempi matka siten, että uudessa pisteessä ko. normi on pienempi. Idea on täsmälleen sama kuin yhdenmuuttujan Newtonin menetelmän yhteydessä mainitulla vaimennetulla Newtonin menetelmällä.

Tämä idea on toteutettu algoritmossa NR::newt.c s. 386, jonka käytöstä on esimerkki alla.

```
// FILE: mynewt2.cpp begins
// Solves a system f(x,y)=0 by improved Newton method
// with several initial values
#include<iostream>
#include<iomanip>
#include<cmath>

#include "nr.h"
#include "gnusurf.h"

#define SCALE 1
#define N 2

using namespace std;

void funcv(Vec_I_DP &x,Vec_O_DP &f)
```

```

{
    f[0]=SQR(x[0])+SQR(x[1])-2.0;
    f[1]=exp(x[0]-1.0)+x[1]*SQR(x[1])-2.0;
}

double ff(double x, double y)
{
    Vec_DP u(N), v(N);
    DP tmp;
    u[0]=x; u[1]=y;
    funcv(u,v);
    tmp=v[0]*v[0]+v[1]*v[1];
    return pow(tmp, 0.5);
}

int main()
{
    int i, p,k;
    bool check;
    Vec_DP x(N),f(N),x0(N);
    double xx[]={-2.,10.},yy[]={-2.,10.};

    for (k=0;k<=3;k++)
        for (p=0;p<=3;p++)
            {
                x[0]=0.5+k; x0[0]=x[0];
                x[1]=0.5+p; x0[1]=x[1];
                cout<<" \n ----- \n";
                cout<<"Before iteration: \n";
                for (i=0;i<N;i++)
                    printf("%5d %12.6f %12.6f %12.4e\n",i,x0[i],x[i],f[i]);
                NR::newt(x,check,funcv);
                funcv(x,f);
                if (check) cout<<"Convergence problems.\n";
                cout<<"After iteration: \n";
                printf("%5s %12s %12s %12s\n","Index","xstart", "x","f");
                for (i=0;i<N;i++)
                    printf("%5d %12.6f %12.6f %12.4e\n",i,x0[i],x[i],f[i]);
            }
    cout<<"Enter x0:\n";
    cin>>x[0];
    cout<<"Enter y0:\n";
    cin>>x[1];
    x0[0] = x[0];
    x0[1] = x[1];
}

```

```

NR::newt(x,check,funcv);
printf("%5s %12s %12s %12s\n","Index","xstart", "x","f");
for (i=0;i<N;i++)
    printf("%5d %12.6f %12.6f %12.4e\n",i,x0[i],x[i],f[i]);
gnusurf(ff,xx,yy,SCALE,0,"ff(x,y)","|ff(x,y)|");
return 0;
}
// FILE: mynewt2.cpp ends

```

```
/*
```

```
After iteration:
```

Index	xstart	x	f
0	3.500000	1.000000	6.4812e-10
1	1.500000	1.000000	1.6371e-09

```
-----
```

```
Before iteration:
```

0	3.500000	3.500000	6.4812e-10
1	2.500000	2.500000	1.6371e-09

```
After iteration:
```

Index	xstart	x	f
0	3.500000	-0.713747	9.9195e-13
1	2.500000	1.220887	2.5931e-12

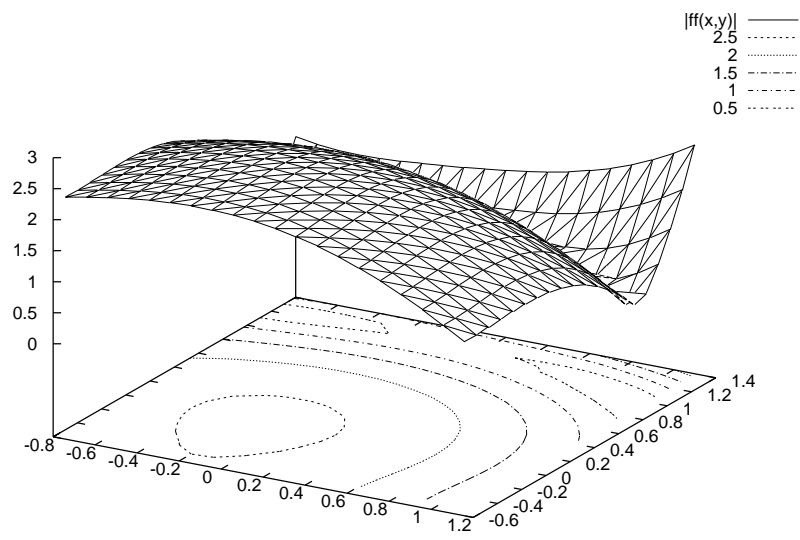
```
-----
```

```
Before iteration:
```

0	3.500000	3.500000	9.9195e-13
1	3.500000	3.500000	2.5931e-12

```
.....
```

```
*/
```





## 10 MINIMOINTI JA MAKSIMOINTI

### Johdantoa

Reaaliarvoisen funktion maksimointi palautuu minimointiin, sillä  $f$ :n maksimointi on sama kuin  $(-f)$ :n minimointi.

Pisteet  $X, Y, Z$  ”sulkevat” (tai paikallistavat, engl. “bracket”) minimin, jos funktion arvo  $Y$ :ssä on pienempi kuin  $X$ :ssä t.  $Z$ :ssa, ts.  $f(Y) < \min\{f(X), f(Z)\}$ .

*Optimointi*. Perustehtävänä kohdefunktion (object function) minimointi (tavallisesti useamman muuttujan funktio).

**Rajoitettu optimointi** (engl. constrained optimization). Tällöin muuttujilla on a priori-rajoituksia, esim. ratkaisun on oltava tietyssä alueessa.

**Lineaarinen ohjelmointi (LO)** (engl. linear programming (LP)). Nyt minimoitava funktio ja myös rajoite-ehdot lineaarisia. Ratkaisumenetelmänä on ns. simpleksialgoritmi.

NR:n suositukset optimointialgoritmeista ovat dimension mukaan jaoteltuina seuraavat.

#### **Yksiulotteinen minimointi.**

- Ilman derivaattojen käyttöä: *Brentin menetelmä* tai *kultaisen leikkauksen* haku.
- Derivaattojen avulla: Brentin menetelmän parannus, jossa käytetään derivaattoja.

#### **Moniulotteinen minimointi.**

Muistitilan tarve  $N$  tai  $N^2$  voi olla pullonkaulana, jos avaruuden dimensio  $N$  on suuri.

- Nelder-Meadin simpleksihaku -menetelmä: derivaattoja ei tarvita. (Ei pidä sekoittaa lineaarisen ohjelmoinnin simplex menetelmään.)

- Suuntajoukko-menetelmä (Powellin menetelmä): ei tarvita derivaattoja. Tarvitaan 1-ulott. minimoinnin alialgoritmi, kuten Brent.

Derivaattojen käyttöön perustuu 2. algoritmiluokkaa, joista kumpikin tarvitsee 1-ulotteisen minimimoinnin alialgoritmin.

a) konjugaattigradienttimenetelmät (esim. Fletcher-Reeves ja Polak-Ribiere menet.) (muistitilan tarve  $N$ )

b) kvasi-Newton menetelmät (esim. Davidson-Fletcher-Powell (DFP) tai Broyden-Fletcher-Goldfarb-Shanno (BFGS) (muistitilan tarve  $N^2$ )).

### **Yleishuomautuksia.**

- 1-ulott.tapauksessa  $f$ :n minimin haku  $\Leftrightarrow f'$ :n 0-kohtien haku
- Moniulotteisessa tapauksessa käytetään usein  $f$ :lle minimipisteen läheisyydessä kvadraattista approksimaatiota. Minimoinnin kannalta hankala esimerkkifunktio saattaa olla esim.  $f(x, y) = x^{10} + y^{10}$ .

Algoritmien toimintaa ei NR:ssä kuvata yksityiskohtaisesti juuri missään. Tämä koskee myös optimointialgoritmejä. Myöskään konvergenssitodistuksia ei esitetä.

### **Jyrkimmän laskun menetelmä.**

Kirjamme ei esittele ns. jyrkimmän laskun (engl. steepest descent) menetelmää, koska se on em. menetelmiä paljon heikompi. Koska tämä menetelmä on kuitenkin geometriseen intuitioon vetoava ja erittäin yksinkertainen toteuttaa, on algoritmi kurssin www-sivulla saatavissa mysteep5.cpp. Ajatuksena on muodostaa murtoviiva, siten että se lähestyisi funktion minimikohtaa. Valitaan ensin lähtöpiste ja edetään siitä siihen suuntaan, jossa funktio voimakkaimmin vähenee, kunnes löydetään funktion lokaali minimi tällä suunnalla. Toistetaan sama uudessa pistessä.

Palautetaan mieleen, että em. voimakkaimman vähenemisen suunta on funktion negatiivisen gradientin suunta. Gradientin laskuun käytämme algoritmia numjf.cpp. Kohdefunktion pitää siis olla de-

rivoituva. On helppo osoittaa, että kahden peräkkäisen murtoviivan osana olevan janan välinen kulma on suora.

**10.1 Kultaisen leikkauksen haku.** Olkoon  $a < b < c$ . Sanotaan, että pisteet  $a, b, c$  rajoittavat (t. paikallistavat t. sulkevat t. haarukoivat) minimin (engl. bracket) jos,  $f(b) < \min\{f(a), f(c)\}$ .

Uusi minimin rajoittava kolmikko löydetään seuraavasti:

1) Tapaus  $b - a > c - b$ . Valitaan  $x \in (a, b)$ .

Jos  $f(a) < f(x) \Rightarrow x, b, c$  rajoittavat minimiin ( $a \leftarrow x$ ).

Jos  $f(x) < f(b) \Rightarrow a, x, b$  rajoittavat minimiin ( $c \leftarrow b, b \leftarrow x$ )

2) Tapaus  $b - a < c - b$  kuten yllä mutta valitaan  $x \in (b, c)$ .

Iteroimalla vaiheita 1) & 2) kunnes  $|a - c| < tol$  löydetään minimi.

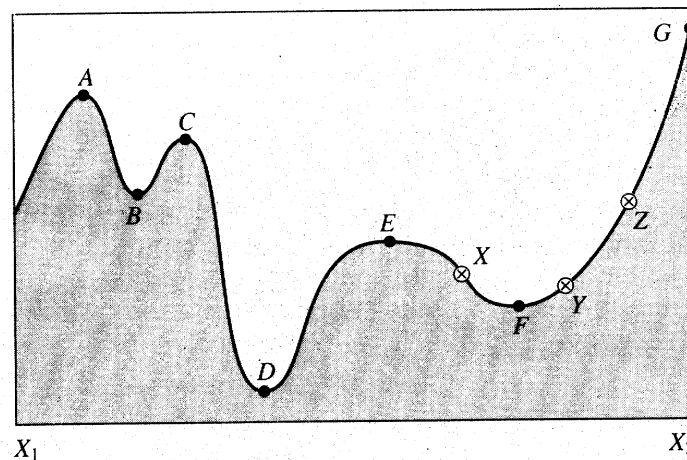


Figure 10.0.1. Extrema of a function in an interval. Points  $A$ ,  $C$ , and  $E$  are local, but not global maxima. Points  $B$  and  $F$  are local, but not global minima. The global maximum occurs at  $G$ , which is on the boundary of the interval so that the derivative of the function need not vanish there. The global minimum is at  $D$ . At point  $E$ , derivatives higher than the first vanish, a situation which can cause difficulty for some algorithms. The points  $X$ ,  $Y$ , and  $Z$  are said to "bracket" the minimum  $F$ , since  $Y$  is less than both  $X$  and  $Z$ .

Miten suuri tarkkuus void. saavuttaa? Olkoon  $b$  minimikohta, jolloin  $f'(b) = 0$ .

$$\text{Taylor} \Rightarrow f(x) \sim f(b) + \frac{1}{2}f''(b)(x - b)^2$$

$$\text{Vaatimus: } \left| \frac{1}{2} f''(b)(x - b)^2 \right| < \epsilon |f(b)|$$

$$\Leftrightarrow |x - b| < \sqrt{\epsilon} \sqrt{\frac{2|f(b)|}{|f''(b)|}} = \sqrt{\epsilon} |b| \sqrt{\frac{2|f(b)|}{|f''(b)||b|^2}}$$

$\epsilon = \text{kone-epsilon} \sim 10^{-16}$ .

**Peukalosääntö.** Yo. tarkastelun nojalla NR suosittaa, että valitaan 1-ulotteisessa minimoinnissa  $\text{tol} \sim \sqrt{\epsilon} \sim 10^{-5}$ .

Miten yo. vaiheessa 1) & 2) valitaan uusi piste  $x$ ?

**Kultaisen leikkauksen haku (Fibonacci haku).** Merk.  $\delta = \frac{3-\sqrt{5}}{2} = 0.38197$ .

1) Jos  $b - a > c - b$  valit.  $x = b - \delta(b - a)$

2) Jos  $b - a < c - b$  valit.  $x = b + \delta(c - b)$

NR: :mnbrak s.281 hoitaa minimin rajoittamisen

NR: :golden s.282 tekee kultaisen leikk.haun

```
// FILE: mygold.cpp begins
// g++ -Wall mygold.cpp -L../lib -I../utils -I../gnuplot02 -o a -lm -lnr
#include <iostream>
#include <iomanip>
#include <cmath>
#include "nr.h"
using namespace std;

// Driver for routine golden

DP func(const DP x)
{
    return NR::bessj0(x);
}

int main(void)
{
    const DP TOL=1.0e-6,EQL=1.0e-3;
    bool newroot;
```

```

int i,j,nmin=0;
DP ax,bx,cx,fa,fb,fc,xmin;
Vec_DP amin(20);

cout << "Minima of the function bessj0" << endl;
cout << setw(10) << "min. #" << setw(9) << "x";
cout << setw(18) << "bessj0(x)" << setw(13) << "bessj1(x)" << endl;
cout << fixed << setprecision(6);
for (i=0;i<100;i++)
{
    ax=DP(i);
    bx=DP(i+1);
    NR::mnbrak(ax,bx,cx,fa,fb,fc,func);
    NR::golden(ax,bx,cx,func,TOL,xmin);
    if (nmin == 0)
    {
        amin[nmin++]=xmin;
        cout << setw(7) << nmin << setw(16) << xmin;
        cout << setw(13) << NR::bessj0(xmin);
        cout << setw(13) << NR::bessj1(xmin);
        cout << endl;
    }
    else
    {
        newroot=true;
        for (j=0;j<nmin;j++)
            if (fabs(xmin-amin[j]) <= EQL*xmin) newroot=false;
        if (newroot) {
            amin[nmin++]=xmin;
            cout << setw(7) << nmin << setw(16) << xmin;
            cout << setw(13) << NR::bessj0(xmin);
            cout << setw(13) << NR::bessj1(xmin);
            cout << endl;
        }
    }
}
return 0;
}
// FILE: mygold.cpp ends
/* Output:
Minima of the function bessj0
min. #      x      bessj0(x)  bessj1(x)
1          3.83171  -0.402759  2.84232e-07
2          10.1735  -0.249705  1.34685e-07
3          16.4706  -0.196465  2.37059e-07

```

```

...
14          85.604   -0.0862347  5.51502e-07
15          91.8875  -0.0832343 -2.19822e-06
16          98.171   -0.0805267 -4.9987e-07
*/

```

**10.2 Parabolinen interpolointi ja Brentin menetelmä.** Kultaisen leikkauksen haku toimii kaikissa tilanteissa. ”Hyvissä” tilanteissa on mahdollista edetä tehokkaammin käyttäen hyväksi sitä, että ääriarvokohdissa funktio on ”parabelin näköinen”. Pisteiden  $(a, f(a))$ ,  $(b, f(b))$ ,  $(c, f(c))$  kautta kulkevan parabelin ääriarvokohdan abskissa on

$$(1) \quad x = b + \frac{1}{2} \frac{(b-a)^2(f(b)-f(c)) - (b-c)^2(f(b)-f(a))}{(b-a)(f(b)-f(c)) - (b-c)(f(b)-f(a))}.$$

Huomaa kuitenkin, että parabelin ääriarvon laatua (min vai max) pistessä  $x$  ei tiedetä.

**Idea.** Käytetään kaavaa (1), jos se näyttää johtavan hyviin tuloksiin, muuten hidasta mutta varmaa kultaisen leikkauksen hakua.

**Mahdollisia vaikeuksia.**

- (a) Valinta em. kahden strategian välillä voi olla vaikeaa.
- (b) ”Loppupeli” ehkä vaikea (pyöristysvirheet).
- (c) Luokittelun ”hyviin” ja ”huonoihin” tilanteisiin täytyy olla robusti (toimintavarma).

**Brentin menetelmä.** Toteuttaa em. ideat: Kussakin vaiheessa pidetään kirjaa kuudesta pisteestä  $a, b, u, v, w, x$ , jotka määrit. seur.: funktion minimi on rajoitettu  $a$ :n ja  $b$ :n väliin;  $x$  on piste, jossa funktiolla on pienin arvo tähän asti tutkituista pisteistä;  $w$ :ssä funktiolla on toiseksi pienin arvo ja  $v$  on  $w$ :n edellinen arvo sekä  $u$  on piste, jossa funktion arvo laskettiin viimeksi. Lisäksi merk.  $x_m = (a + b)/2$ . Tilannetta havainnollistavan kuvan merkinnät:

- 1, 2, 3 alkupisteet

- 4 = pisteiden 1, 2, 3 kautta kulkevan parabelin minimikohta
- 5 = pisteiden 1, 4, 2 kautta kulkevan parabelin minimikohta

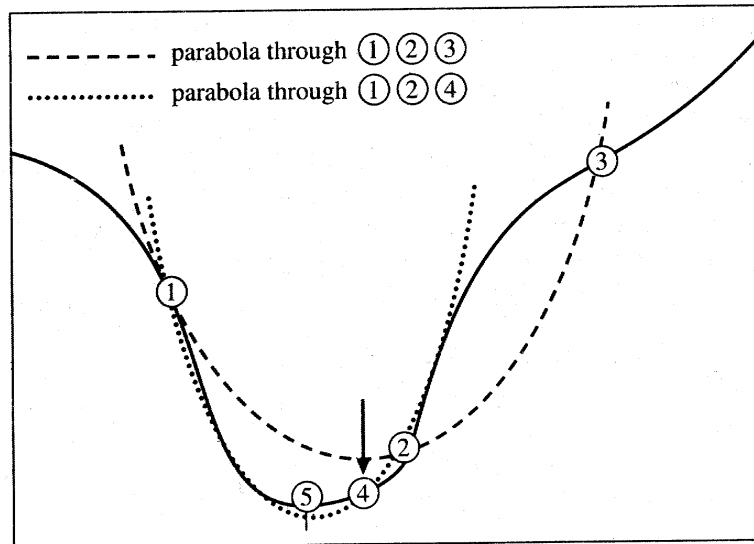


Figure 10.2.1. Convergence to a minimum by inverse parabolic interpolation. A parabola (dashed line) is drawn through the three original points 1,2,3 on the given function (solid line). The function is evaluated at the parabola's minimum, 4, which replaces point 3. A new parabola (dotted line) is drawn through points 1,4,2. The minimum of this parabola is at 5, which is close to the minimum of the function.

**Huom.** Brentin menetelmä olettaa, että funktio on määritelty *koko* reaaliakselilla. Jos näin ei ole asianlaita, on funktio jatkettava sopivalla tavalla koko reaaliakselilla määritellyksi funktioksi.

NR:n algm:ssa on lisävaatimuksia parabelin käytölle, esim. vaatimus, että sen aiheuttama korjaus on enintään puolet edellisen iteraatiokierroksen korjauksesta. (Tästä seuraa, että parab. käyttö ei estä konvergenssia.)

Algm NR: :brent s.404

Käytetään myös osana korkeampiulotteisessa minimoinnissa, etsittäessä minimiä suorilla. Näin ollen NR: :brent on eräs NR:n tärkeimpiä optimointialgoritmeja.

```
// FILE: mybrent.cpp begins
// g++ -Wall mybrent.cpp -L../lib -I../utils -I../gnuplot02 -o a -lm -lnr
#include <iostream>
#include <iomanip>
#include <cmath>
```

```

#include "nr.h"
using namespace std;

// Driver for routine brent

DP func(const DP x)
{
    return NR::bessj0(x);
}

int main(void)
{
    const DP TOL=1.0e-6,EQL=1.0e-4;
    bool newroot;
    int i,j,nmin=0;
    DP ax,bx,cx,fa,fb,fc,xmin;
    Vec_DP amin(20);

    cout << endl << "Minima of the function bessj0" << endl;
    cout << setw(10) << "min. #" << setw(9) << "x";
    cout << setw(18) << "bessj0(x)" << setw(13) << "bessj1(x)" << endl;
    cout << fixed << setprecision(6);
    for (i=0;i<100;i++) {
        ax=DP(i);
        bx=DP(i+1);
        NR::mnbrak(ax,bx,cx,fa,fb,fc,func);
        NR::brent(ax,bx,cx,func,TOL,xmin);
        if (nmin == 0)
        {
            amin[nmin++]=xmin;
            cout << setw(7) << nmin << setw(16) << xmin;
            cout << setw(13) << NR::bessj0(xmin);
            cout << setw(13) << NR::bessj1(xmin);
            cout << endl;
        }
        else
        {
            newroot=true;
            for (j=0;j<nmin;j++)
                if (fabs(xmin-amin[j]) <= (EQL*xmin)) newroot=false;
            if (newroot) {
                amin[nmin++]=xmin;
                cout << setw(7) << nmin << setw(16) << xmin;
                cout << setw(13) << NR::bessj0(xmin);
                cout << setw(13) << NR::bessj1(xmin);
            }
        }
    }
}

```



```

        cout << endl;
    }
}
return 0;
}
// FILE: mybrent.cpp ends
/* Output:
Minima of the function bessj0
   min. #      x      bessj0(x)  bessj1(x)
     1         3.8317  -0.402759  4.85347e-07
     2         10.1735 -0.249705  1.14457e-07
     3         16.4706 -0.196465  -7.6138e-07
...
    14         85.604  -0.0862347 -2.17754e-08
    15         91.8875 -0.0832343  3.69073e-07
    16         98.1709 -0.0805267  2.66599e-06
*/

```

**10.3 1-ulott. haku ja ensimmäiset derivaatat.** Yksioikoinen menettely olisi hakea deriv.:n 0-kohdat ja näistä edelleen ääriarvopisteet. Vaikeutena olisi tällöin erottaa maksimit minimeistä ja tästä syystä NR ei sitä suosittele.

NR esittää sen sijaan algoritmin `NR::dbrent`.

Oletetaan, että  $a < b < c$  sulkevat minimin.

Jos  $f'(b) > 0 \Rightarrow$  haetaan minimiä väliltä  $(a, b)$  seuraavasti. ensin aset.  $b$ :n ja kahden muun parhaan pisteen kautta parabeli. Parabelin deriv.:n 0-kohta etsitään sekanttimenet. Varmistus kuten Brent.

Jos  $f'(b) < 0 \Rightarrow$  haetaan minimiä väliltä  $(b, c)$ .

Algm `NR::dbrent` s. 406

## 10.4 Simpleksihakumenetelmä.

Nelder ja Mead 1965

(simpleksi-haku = polytooppihaku = downhill simplex method)

**Idea.** Työnnetään ”kivi” liikkelle laaksoa ympäröivältä harjanteelta. Alaspäin vyöryvä kivi löytää reittinsä matalimman kohdan.

Simpleksihaku-menetelän simpleksi on kiven asemassa, simpleksi tosin muuttaa haun aikana muotoaan kuten ”ameeba”. Ei tarvita derivaattaa eikä 1-ulotteista minimointia.

Simpleksi tasossa on kolmio. Yleisesti  $\mathbb{R}^N$ :n simpleksi on monitahokas, jolla on  $N + 1$  kärkeä. Simpleksihaku menet.:n aloitus vaatii  $N + 1$  kärkipistettä. Jos yksi näistä on  $P_0$ , niin muiksi void. valita  $P_0 + \lambda_i e_i$  ( $e_i = \mathbb{R}^N$ :n yksikkövektori), missä  $\lambda_i$  on skaalaustekijä. Mahdollisesti  $\lambda_i = \lambda \forall i$ .

**Menetelmä.** Valitaan ”lähtösimpleksin” kärjet  $x_1, \dots, x_{n+1}$  alueelta, jolla minimin ajatellaan sijaitsevan. Merkitään

$$f_i = f(x_i)$$

$$x_h = \text{kärki, jossa } f_h = \max\{f_i\}$$

$$x_l = \text{kärki, jossa } f_l = \min\{f_i\}$$

$x_0 = \frac{1}{n} \sum_{i \neq h} x_i$  eli huonoimman kärjen  $x_h$  vastaisen sivun painopiste.

Kiinnitetään parametrit  $\alpha > 0$ ,  $0 < \beta < 1$ ,  $\gamma > 1$ . Toistetaan seuraavia perusoperaatiota:

(1) **Peilaus.** Muodostetaan uusi kärki peilaamalla huonoin pisteen vastaisen sivun suhteen. Merkitään  $x_r = (1 + \alpha)x_0 - \alpha x_h$ .

Jos  $f_r > f_i \forall i \neq h$  mennään kohtaan (3).

Jos  $f_i \leq f_r < f_h$ , korvataan  $x_h$   $x_r$ :llä ja saadaan uusi simpleksi. Toistetaan peilaus.

(2) **Laajennus.** Jos  $f_r < f_l$  niin edetään pitemmälle samaan suuntaan ja merk.  $x_e = \gamma x_r + (1 - \gamma)x_0$ , missä  $\gamma > 1$  on ns. laajennuskerroin. Jos  $f_e < f_l$ , korvataan  $x_h$   $x_e$ :llä ja saadaan uusi polytooppi. Jos  $f_e \geq f_l$ , korvataan  $x_h$   $x_r$ :llä (laajennus ei kannata). Palataan (1):een.

(3) **Pienennys.** Jos  $f_r < f_h$ , korvataan  $x_h$   $x_r$ :llä. Merkitään  $x_c = \beta x_h + (1 - \beta)x_0$ , missä  $0 < \beta < 1$  on kutistuskerroin. Jos  $f_c <$

$\min(f_h, f_r)$ , korvataan  $x_h$   $x_c$ :llä. Jos  $f_c \geq \min(f_h, f_r)$ , korvataan jokainen  $x_i$  pisteellä  $(x_i + x_l)/2$  l. ”puolitetaan” simpleksi. Palataan (1):een.

**Lopetuskriteeri.** Lopetetaan kun simpleksin sivut kyllin pienet tai kun funktion vaihtelu kärjissä on riittävän vähäistä (esim. kun  $(\frac{1}{n+1} \sum_{i=1}^{n+1} (f_i - f_0)^2)^{1/2} < \epsilon$ .)

Menetelmän yo. kuvaus on monisteesta Neittaanmäki-Mäkelä-Parviainen: Epälineaarinen optimointi (JY, Matem. laitos), jossa on monipuolinen katsaus optimointimenetelmiin. Niin ollen yo. kuvaus ei välttämättä vastaa NR:n algoritmiin NR : : amoeba implementointia (s.292), mutta antaa kuitenkin yleiskuvan simpleksimenetelmästä (NR ei tässäkään kohdassa kovin tarkkaan selosta algoritmin toimintaa). Algoritmin perusteellinen tarkastelu on esitetty tutkimuksessa Lagarias, Reeds, Wright, Wright, SIAM J. Optim. 9 (1998), 112-147.

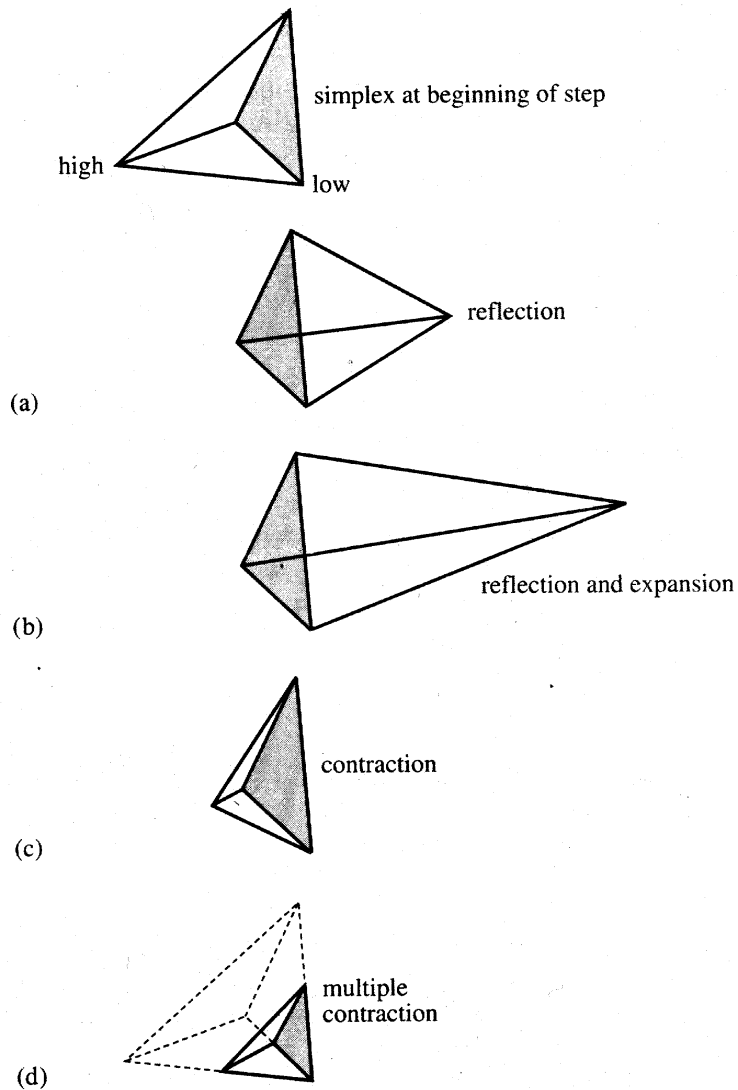


Figure 10.4.1. Possible outcomes for a step in the downhill simplex method. The simplex at the beginning of the step, here a tetrahedron, is shown, top. The simplex at the end of the step can be any one of (a) a reflection away from the high point, (b) a reflection and expansion away from the high point, (c) a contraction along one dimension from the high point, or (d) a contraction along all dimensions towards the low point. An appropriate sequence of such steps will always converge to a minimum of the function.

```
// FILE: myamoeba.cpp begins
// g++ -Wall myamoeba.cpp -L../lib -I../utils -I../gnuplot02 -o a -lm -lnr
#include <iostream>
#include <iomanip>
#include <cmath>
#include "nr.h"
using namespace std;
```

```

// Driver for routine amoeba

DP func(Vec_I_DP &x)
{
    return 0.6-NR::bessj0(SQR(x[0]-0.5)+SQR(x[1]-0.6)+SQR(x[2]-0.7));
}

int main(void)
{
    const int MP=4,NP=3;
    const DP FTOL=1.0e-10;
    int i,nfunc,j;
    Vec_DP x(NP),y(MP);
    Mat_DP p(MP,NP);

    for (i=0;i<MP;i++) {
        for (j=0;j<NP;j++)
            x[j]=p[i][j]=(i == (j+1) ? 1.0 : 0.0);
        y[i]=func(x);
    }
    NR::amoeba(p,y,FTOL,func,nfunc);
    cout << endl << "Number of function evaluations: " << nfunc << endl;
    cout << "Vertices of final 3-d simplex and" << endl;
    cout << "function values at the vertices:" << endl << endl;
    cout << setw(3) << "i" << setw(10) << "x[i]";
    cout << setw(12) << "y[i]" << setw(12) << "z[i]";
    cout << setw(14) << "function" << endl << endl;
    cout << fixed << setprecision(6);
    for (i=0;i<MP;i++) {
        cout << setw(3) << i;
        for (j=0;j<NP;j++) cout << setw(12) << p[i][j];
        cout << setw(12) << y[i] << endl;
    }
    cout << endl << "True minimum is at (0.5,0.6,0.7)" << endl;
    return 0;
}
// FILE: myamoeba.cpp ends

```

```

Number of function evaluations: 64
Vertices of final 3-d simplex and
function values at the vertices:

```

i	x[i]	y[i]	z[i]	function
0	0.498932	0.600845	0.700729	-0.4

1	0.501374	0.598983	0.70052	-0.4
2	0.501484	0.599711	0.698687	-0.4
3	0.498624	0.598268	0.699555	-0.4

True minimum is at (0.5,0.6,0.7)

Seuraava ohjelma havainnollistaa simpleksin kulkureittiä. Se on pienin muutoksin sama kuin edellinen ohjelma. Muutokset liittyvät lähinnä piirtämistä tukeviin täydennyksiin ohjelmassa amoeba2.cpp.

```

/* FILE: myamoe2.cpp begins. */
/* Driver for routine amoeba */
/* g++ -Wall myamoe2.cpp -L../lib -I../utils -I../gnuplot02 -o a -lm -lnr */

#include <cstdlib> // Used in putmat2
#include <cstdio> // Used in putmat2
#include <ctime>
#include <string>
#include <fstream>
#include <iostream>
#include <iomanip>
using namespace std;
#include "nr.h"
#include "matutl02.h"
#include "amoeba2.cpp"

#define MP 3
#define NP 2
#define FTOL 1e-2

double func(Vec_I_DP &x)
{
    return 2*x[0]*x[0]-2*x[1]+3*x[1]*x[1]-x[1]-x[0]*x[1];
    // nice: return x[0]*x[0]-2*x[1]+2*x[1]*x[1]-x[0]-x[0]*x[1];
}

int main(void)
{
    void amoeba2(Mat_IO_DP &p, Vec_IO_DP &y, const DP ftol,
                DP funk(Vec_I_DP &), int &it );
    int i,nfunc,j;

```

```

Vec_I_DP x(NP);
Vec_DP y(MP);
Mat_DP p(MP,NP);
int ITERR=14;
FILE *fp;
init_srand();

for (int k=1;k<=10;k++) /* Iteration with ten different
                        initial simplexes */
{

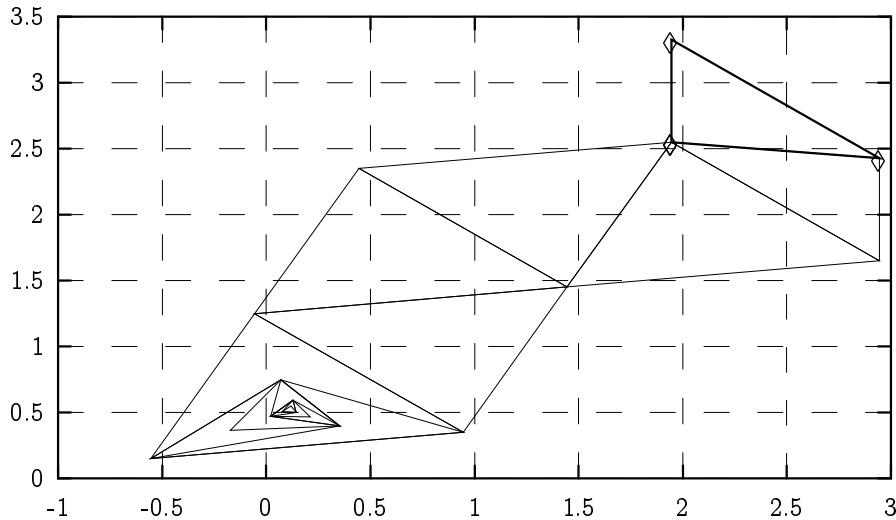
for (i=0;i<MP;i++) {
  for (j=0;j<NP;j++)
    x[j]=p[i][j]=(i == (j+1) ? 1.0 : 0.0)+(3.0*tan(k)+j*rdm(0.1,0.9));
    y[i]=func(x);
}

amoeba2(p,y,FTOL,func, nfunc, ITERR);
printf("\nNumber of function evaluations: %3d\n",nfunc);
printf("Vertices of final simplex and\n");
printf("function values at the vertices:\n\n");
printf("%3s %10s %12s %14s\n\n",
        "i", "x[i]", "y[i]", "function");
for (int i=1-1;i<=MP-1;i++) {
  printf("%3d ",i);
  for (int j=1-1;j<=NP-1;j++) printf("%12.6f ",p[i][j]);
  printf("%12.6f\n",y[i]);
}
fp=fopen("amoplot.cmd","w");
if(fp) {
  fprintf(fp,"set grid\n");
  fprintf(fp,"set timestamp\n");
  /* amoeba2 writes amosimp.dat */
  // fprintf(fp,"plot ");
  fprintf(fp,
"plot 'amosimp.dat' notitle w l lw 3, 'amosimp0.dat' notitle w l lw 4, 'amosimp0.dat' not

  fprintf(fp,"pause -1");
  fclose(fp);
  system("gnuplot amoplot.cmd");
}
else printf("Cannot open file\n");
getchar();
}
}

```

```
/* FILE: myamoe2.cpp ends. */
```



Mon Mar 04 01:04:22 2002

### Rajoitetusta optimoinnista.

NR:n eräät juurenhaku- ja minimointialgoritmit edellyttävät, että funktio on määritelty koko  $\mathbb{R}$ :ssä, ei pelkästään osavälillä. Jos algm käytetään kuitenkin vain osavälillä määriteltyihin funktioihin, niin voidaan joutua virhetilanteisiin.

**Esim.** Merk.  $f(x) = 1/\sqrt{x^3(1-x)}$ ,  $0 < x < 1$ .

1) Ratkaise  $f(x) = 10$ ,  $0 < x < 1/4$ .

2) Etsi  $f$ :n minimi,  $0 < x < 1$ .

Nämä molemmat ovat esimerkkejä tehtävistä, jotka saattavat johtaa vaikeuksiin, jos NR:n algoritmeja yritetään käyttää ilman esivalmisteluja. Mahdollinen esivalmistelu voisi olla esim. muuttujanvaihto, joka estää argumentin karkaamisen luvattomalle alueelle.

Jos halutaan funktion  $z = f(x, y)$  suurin ja pienin arvo joukossa  $x_1 \leq x \leq x_2$ ,  $y_1 \leq y \leq y_2$  voidaan tehdä apusijoitus  $g(x, y) = f(h(x), k(y))$ , missä  $h$ :n arvoalue on  $[x_1, x_2]$  ja  $k$ :n  $[y_1, y_2]$ .

Tähän aihepiiriin liittyy myös NR:n. algm NR::linmin, joka minimoi annetun  $n$ :nnän muuttujan funktion annetun pisteen  $P$  kautta kulkevalla suoralla, jonka suunta on annettu.

### Sakkofunktio (penalty function)-menetelmä.



Olkoon  $f : \mathbb{R}^n \rightarrow \mathbb{R}$  jatkuva. Etsitään  $f$ :n minimiä  $\mathbb{R}^n$ :n kompaktissa joukossa  $K$ . Jatkossa oletetaan, että int  $K$  on alue.

Sakkofunktio-menetelmä yo. rajoitetun minimointitehtävän ratkaisuun perustuu funktion

$$f(x) + g(x)$$

minimointiin ilman rajoituksia, missä  $g(x) \rightarrow \infty$ , kun  $x \rightarrow \partial K$ , ja  $g(x) > 0 \forall x \in K$ . Selvästi funktion  $g$  valinta vaikuttaa ratkaisuun, joten tällä menetelmällä voidaan saavuttaa vain likimääräisiä tuloksia.

Sakkofunktio-menetelmä soveltuu tilanteisiin, joissa  $K$  on rakenteeltaan yksinkertainen esim. suorakulmainen särmiö tai  $\{x \in \mathbb{R}^n : |x - x_0| < r\}$  (kuula). Näihin tilanteisiin liittyen voidaan sakkofunktio eksplisiittisesti konstruoida. NR:ssä ei sakkofunktio-menetelmää kuitenkaan esitellä eikä käytetä.

**Esim.** Tunnettu esimerkkifunktio optimointiteoriassa on ns. Rosenbrockin funktio  $F(x, y) = 100(y - x^2)^2 + (1 - x)^2$ , jonka minimikohta on  $(1, 1)$ . Jos halutaan  $f$ :n minimi joukossa  $\{(x, y) : x^2 + y^2 \leq \frac{1}{4}\}$  niin void. sakkofunktioksi  $g$  valita esim.

$$g(x, y) = \ln \frac{4}{1 - 4(x^2 + y^2)}.$$

Nyt pitää vielä huolehti siitä, että logaritmissa ei tule argumentin virhettä laskennan aikana. (Ns. yleistetty Rosenbrockin funktio on

$$F(x_1, \dots, x_n) = 1 + \sum_{i=2}^n [100(x_i - x_{i-1}^2)^2 + (1 - x_{i-1})^2].$$

$$(x_0 = (\frac{1}{11}, \frac{2}{11}, \dots, \frac{10}{11}))$$

### 10.5 Suuntajoukko-menetelmät.

$N$ :nnän muuttujan funktio suoralle rajoitettuna on esitettävissä yhden muuttujan funktiona. Näin ollen  $N$ :nnän muuttujan funktion pienin arvo suoralla void. löytää aikaisempien menetelmien avulla (jos on olemassa). Tällaiselle algoritmille sopii nimeksi NR: :linmin (s.419).

Algm NR::linmin: Annetulle pisteelle  $P$  ja vektorille  $n$  on etsittävä  $\lambda \in \mathbb{R}$  s.e.  $f(P + \lambda n)$  minimoituu.

Korvaa  $P$   $P + \lambda n$ :llä.

Tällaista minimoivaa pistettä  $P + \lambda n$  kutsutaan  $f$ :n  $(P, n)$ -*minimiksi* l. *minimiksi suuntaan*  $n$  pisteestä  $P$ .

**Huom.** NR::linmin vastakkaisiin suuntiin tuottaa saman uuden pisteen  $P + \lambda n$ , ts.  $\lambda$  voi olla  $< 0$ .

Monet minimointialgoritmit käyttävät osanaan NR::linmin:in kaltaista 1-ulott. minimointia sopivasti valituissa suunnissa. Pääasiallinen ero eri minimointimenetelmissä on juuri suuntien valintamenetelystä, jossa voidaan mm. käyttää hyväksi aikaisemmin laskettuja funktion tai sen gradientin arvoja jne.

Eräs yksinkertainen menetelmä on käyttää yksikkövektorien suuntia  $e_1, \dots, e_N$ , mutta tämä on tavallisesti hidasta. Muut suuntia käyttävät menetelmät pyrkivät parempaan suuntajoukkoon kuin  $\{e_i\}$  käyttämällä laskennan aikana kertynyttä tietoa funktion käyttäytymisestä.

### Liittosuuntamenetelmä (l. konjugoitujen suuntien menet.)

Jos funktio  $f$  saavuttaa pienimmän arvonsa suoralla pisteessä  $P_0$ , niin tällöin  $(\nabla f)(P_0) \perp$  suora (muuten funktion deriv. suoran suuntaan  $\neq 0$  k.o. pisteessä, joka olisi ristiriidassa  $P_0$ :n minimiominaisuuden kanssa).

Taylor-kehitelämä voidaan kirjoittaa seuraavasti

$$\begin{aligned} f(P + x) &= f(P) + \sum_i \frac{\partial f}{\partial x_i} x_i + \frac{1}{2} \sum_{i,j} \frac{\partial^2 f}{\partial x_i \partial x_j} x_i x_j + \dots \\ &\approx c - b \cdot x + \frac{1}{2} x \cdot A \cdot x \end{aligned}$$

missä

$$c = f(P), \quad b = -\nabla f|_P, \quad A_{ij} = \frac{\partial^2 f}{\partial x_i \partial x_j}|_P$$

matriisia  $A$  kutsutaan  $f$ :n *Hessen matriisiksi* pisteessä  $P$ .

Välttämätön ehto ääriarvolle pisteessä  $P$  on  $\nabla f|_P = 0$ , joten minimipisteen läheisyydessä  $f$ :ää approksimoi  $f(P) +$  kvadraattinen termi joka riippuu Hessen matriisista. Sanotaan, että  $u$  ja  $v$  ovat toistensa *liittosuuntia* l. *kongujoituja suuntia*, jos pätee

$$0 = u \cdot A \cdot v \quad (u \text{ ja } v \text{ liittosuuntia})$$

### Powellin menetelmä

Aloitukset:  $u_i = e_i, i = 1, \dots, N$ .

Toistetaan seuraavaa proseduuria niin kauan kuin funktion arvot vähenevät

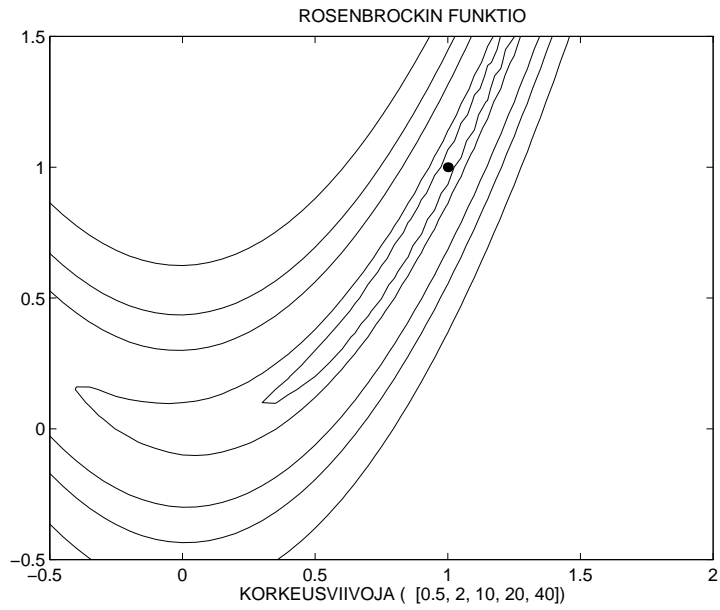
- Talleta alkupiste  $P_0$
- Kun  $i = 1, \dots, N$  etene  $P_{i-1}$ :stä  $(P_{i-1}, u_i)$ -minimiin  $P_i$
- Kun  $i = 1, \dots, N - 1$  aseta  $u_i \leftarrow u_{i+1}$ .
- Aseta  $u_N = P_N - P_0$ .
- Etene  $P_N$ :stä  $(P_N, u_N)$ -minimiin, merk. tätä pistettä  $P_0$ :lla.

Powell osoitti v. 1964 että kvadraattiselle funktiolle  $A$  yo. proseduurin  $k$  iteraatiota tuottavat vektorijoukon, jonka  $k$  viimeistä vektoria ovat keskenään konjugoituja ts.  $u \cdot Av = 0$ . Voidaan osoittaa, että yo. algm  $N$  kertaa toteutettuna minimoi  $N$ :n muuttujan kvadraattisen funktion (kullakin iteraatiokierroksella tehdään  $N + 1$  NR::linmin-kutsua).

Kokemus on osoittanut, että Powellin algoritmin tuottamalla vektorijoukolla on taipumus surkastua niin, että muodostuukin lähes lineaarisesti riippuva vektorijoukko. Surkastumista voidaan yrittää torjua esim. seuraavin tavoin.

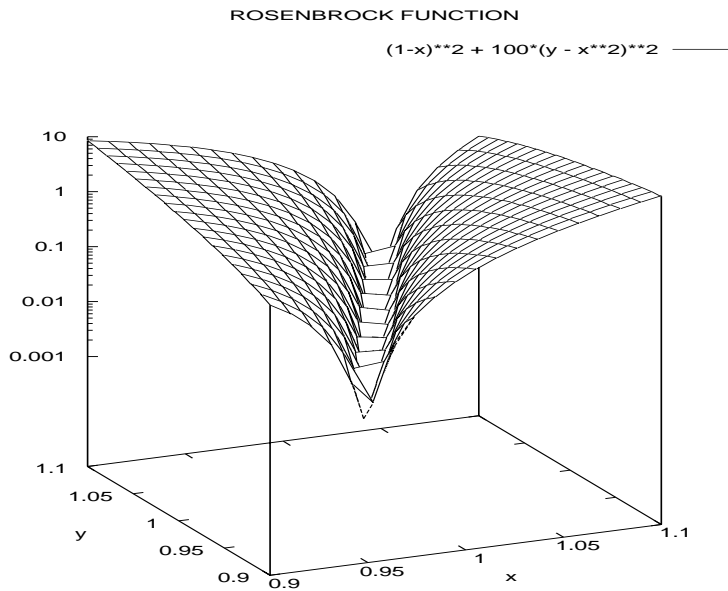
1. Kun algoritmi on toteutettu  $N$  t.  $N + 1$  kertaa asetetaan uudelleen  $u_i = e_i$ .

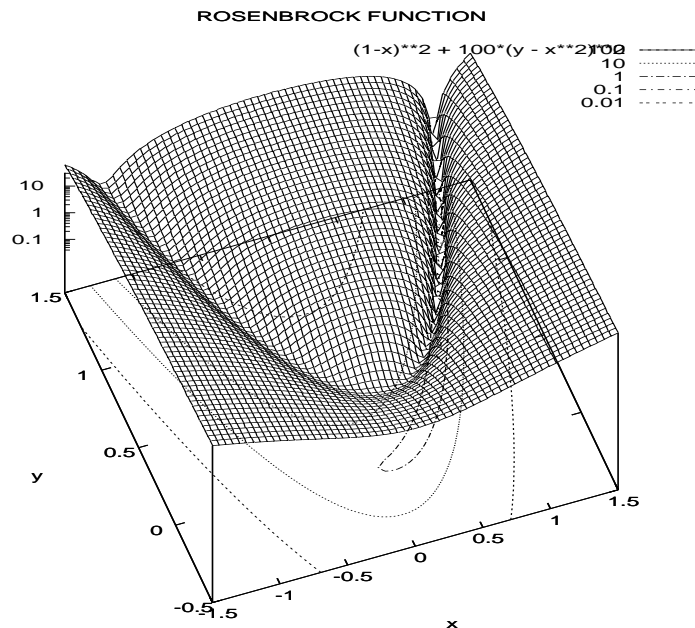
2. Voidaan valita muutamia hyviä suuntia  $N$ :n konjug. suunnan asemasta.



### Minimien geometriaa

Oheisessa kuvassa on kahden muuttujan reaaliarvoisen funktion nivookäyriä. Funktion minimi sijaitsee ”banaanilaakson” pohjalla.





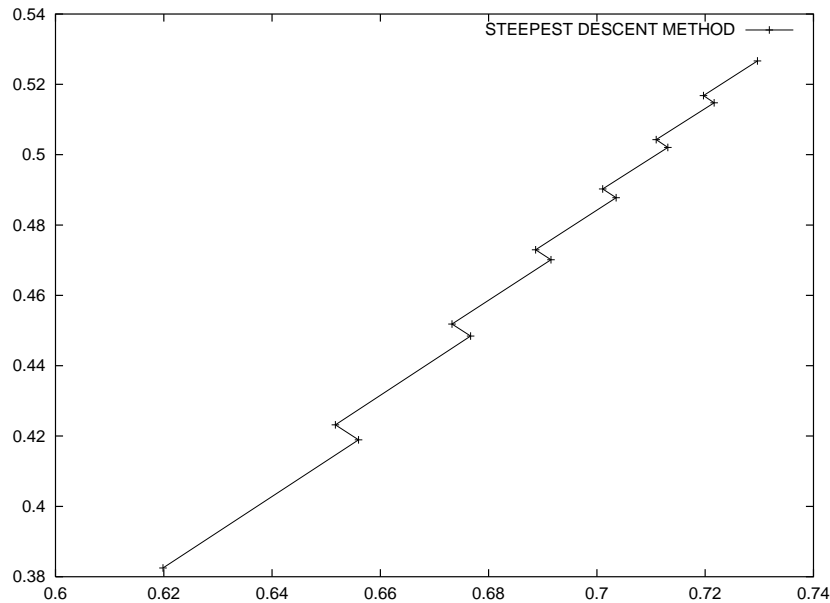
Funktion voimakkaimman kasvun suunta on  $\nabla f$  ja voimakk. vähenemisen suunta  $-\nabla f$ . (vrt. 1-ulott. tapaus). Jos  $P$ :n kautta käyvän nivookäyrän tangentin suunta  $P$ :ssä on  $t$ , niin  $t \cdot \nabla f = 0$  (seuraa nivookäyrän määritelmästä).  $N$ -ulott. tapauksessa nivoo-pinnalla on vastaava ominaisuus.

Funktion minimointi voidaan periaatteessa suorittaa seuraavalla Cauchyn esittämällä *jyrkimmän laskeuman* (*t. suunnan*) menetelmällä (*gradienttimenetelmä*, steepest descent method)

1. Valitaan alkupiste  $P_0$  ja tehdään NR::linmin suuntaan  $-\nabla f|_{P_0}$  ja saadaan piste  $P_1$ .

2. Jos  $|P_0 - P_1| > \text{tol aset.}$   $P_0 \leftarrow P_1$  ja toistetaan askel 1. Muuten lopetus.

Voidaan osoittaa, että gradienttimenet. suppenee aina kun alkuarvaus on kyllin lähellä minimiä. Konvergenssi voi tosin "baanilaakson" tyyppisissä tilanteissa olla piinallisen hidasta (menetelmän kulkema reitti muodostaa sik-sakkia). Ks. www-sivun ohjelma mysteep5.cpp .



Olkoon  $f$  funktio, jolla on 1-käsitt. minimi 0:ssa ja joukko  $\{z \in \mathbb{R}^n : |f(z)| \leq 1\}$  on rajoitettu. Oletamme, että jollakin menetelmällä on generoitu pisteistö  $p_k$  s.e.  $p_k \rightarrow p_0$ ,  $f(p_k) \rightarrow f(p_0)$  ja  $f(p_{k+1}) < f(p_k)$  kaikilla  $k = 1, 2, \dots$ . On huomattava, että tällöin voi olla  $f(p_0) > f(0)$  ts.  $p_0$  ei ole minimipiste. (Näin on esimerkiksi jos valit.  $p_0 \neq 0$  s.e.  $f(p_0) > f(0)$  ja jono  $p_k \rightarrow p_0$  s.e.  $f(p_k) > f(p_{k+1})$ ).

Minimointimenetelmiä, joissa  $f(p_{k+1}) < f(p_k)$  toteutuu kutsutaan *laskevien suuntien menetelmiksi*.

### Powellin menetelmän modifikaatio

Kuten edellä mainittiin, Powellin menetelmässä voi esiintyä generoitujen suuntien luhistumisilmiö ts. niiden muodostama joukko tulee (lähes) lineaarisesti riippuvaksi. Tämä ikävä ilmiö aiheuttaa komplikaatioita algoritmin toimintaan. Komplikaatioita voidaan yrittää hoitaa modifioimalla Powellin menetelmää seuraavasti.

- Jätetään pois se suunta, johon funktio eniten vähenee, otetaan  $P_N - P_0$  edelleenkin mukaan.

Heuristinen perustelu: Em. suunta on luultavasti lähellä  $P_N - P_0$ :aa, joten sen poisjättö toivottavasti vähentää suuntien ”yhteenluhistumista”.

-Poikkeus yo. sääntöön: Uutta suuntaa  $P_N - P_0$  ei oteta mukaan suuntajoukkoon vaan pidet. edelliset suunnat, jos joko (a) tai (b) toteutuu. Merk.

$$f_0 = f(P_0), f_N = f(P_N), f_E = f(2P_N - P_0)$$

$$\nabla f = \max\{(\nabla f)_i\}, (\nabla f)_i = |f(P_i) - f(P_{i-1})|$$

(a)  $f_E \geq f_0$

(b)  $2(f_0 - 2f_N + f_E)[(f_0 - f_N) - \nabla f]^2 \geq (f_0 - f_N)^2 \nabla f$

**Perusteluja:**

Tapauksessa (a) uuteen suuntaan  $P_N - P_0$  eteneminen pisteestä  $P_N$  ei näytä johtavan funktion arvojen pienenemiseen. Kriteerille (b) on kirjassa esitetty perusteluja (s. 299).

```
// FILE: mypowell.cpp begins
// g++ -Wall mypowell.cpp -L../lib -I../utils -I../gnuplot02 -o a -lm -lnr

#include <cstdlib> // Used in putmat2
#include <cstdio> // Used in putmat2
#include <ctime>
#include <string>
#include <fstream>
#include <iostream>
#include <iomanip>
#include <cmath>
using namespace std;
#include "nr.h"
#include "matut102.cpp"

//const int NDIM=7;

Vec_DP truemin(1);

// Driver for routine powell

DP func(Vec_I_DP &x)
{
    int n=x.size();
    DP t=SQR(x[0]-truemin[0]);
    for (int i=1;i<n;i++) t+= SQR(x[i]-truemin[i]);
    return 1.0+t; //-NR::bessj0(t);
```

```

}

int main(void)
{
    const DP FTOL=1.0e-6;
    int iter;
    DP fret;

    init_srand();
    const int NDIM=(int)rdm(3.0,7.0);
    Vec_DP p(NDIM);
    truemin=p;
    for (int i=0;i<NDIM;i++)
{
    truemin[i]=2.0*rdm(0.2, 1.8);
    p[i]=truemin[i]+5.0*rdm(0.2 , 2.0);}
    cout<<"Iterations started at: \n"<<p<<endl;
    Mat_DP xi(NDIM,NDIM);
    for (int i=0;i<NDIM;i++)
        for (int j=0;j<NDIM;j++)
            xi[i][j]=(i == j ? 1.0 : 0.0);
    NR::powell(p,xi,FTOL,iter,fret,func);
    cout << "Iterations: " << iter << endl << endl;;
    cout << "Minimum found at: " << endl;
    cout << fixed << setprecision(6);
    for (int i=0;i<NDIM;i++) cout << setw(12) << p[i];
    cout << endl << endl << "Minimum function value = ";
    cout << setw(12) << fret << endl << endl;
    cout << "True minimum of function is at:" << endl;
    cout << truemin<<endl;
    return 0;
}
// FILE: mypowell.cpp ends
Iterations started at:
    10.54074 10.867299 11.819706 11.198338 4.0820065

Iterations: 1

Minimum found at:
    3.56817      2.078      2.47333      1.29719      1.99635

Minimum function value =          1

True minimum of function is at:
    3.5681672 2.0780049 2.4733272 1.2971873 1.9963494

```



## 10.6 Liittogradientti- I. konjugaattigradianttimenetelmä

Kaksi yksioikoista funktion minimointitapaa, joissa kummassakin on osana 1- ulott. miniminhaku, ovat

- a) minimointi koord. akselien suunnissa, muuttuja kerrallaan
- b) jyrkimmän laskeuman menetelmä (Cauchy)

Kokemus on osoittanut b):n varsin tehottomaksi ja näin ollen NR ei esitä sille omaa algoritmia.

Aikaisemmin todettiin: Jos  $P$  on  $f$ :n  $(P_0, n)$ -minimi, niin  $n \cdot (\nabla f)(P) = 0$ . Erityisesti tämä pätee (b) tapauksessa. Siis molemmissa tapauksissa (a) ja (b) kaksi peräkk. NR: :linmin:iä suoritetaan suuntiin, jotka kohtisuoria toisiaan vastaan.

Näitä menetelmiä voidaan pyrkiä tehostamaan valitsemalla uusi suunta s.e. se on  $\perp$  mahdollisimman montaa aikaisempaa suuntaa vastaan. Samantapainen idea esiintyy lin.algebrasta tunnetussa Gram-Schmidtin ortogonalisointi-menetelmässä. Tähän filosofiaan pohjautuvia minimointimenetelmiä: kutsutaan *liittogradientti* I. *konjugaatti-gradientti-menetelmiksi*. NR esittelee näistä *Fletcherin-Reevesin* ja *Polakin-Ribieren* menetelmät.

Jatkossa oletetaan, että

$$(1) \quad f(x) \approx c - b \cdot x + \frac{1}{2} x \cdot Ax \quad (b = -\nabla f)$$

2. **Lause** Olkoon  $A$  symmetrinen pos. def.  $n \times n$  matriisi,  $g_0 \in \mathbb{R}^n$  mielivaltainen ja  $h_0 = g_0$ . Kun  $i = 0, 1, 2, \dots$  niin määritellään

$$(3) \quad g_{i+1} = g_i - \lambda_i A h_i, \quad h_{i+1} = g_{i+1} + \gamma_i h_i$$

missä vakiot  $\lambda_i, \gamma_i$  valitaan s.e.  $g_{i+1} \cdot g_i = 0$  ja  $h_{i+1} \cdot A h_i = 0$  eli

$$(4) \quad \lambda_i = \frac{h_i \cdot g_i}{g_i \cdot A h_i}, \quad \gamma_i = \frac{g_{i+1} \cdot A h_i}{h_i \cdot A h_i}$$

(jos nimittäjä = 0 niin asetetaan vakiot = 0). Silloin

$$(5) \quad g_i \cdot g_j = 0, \quad h_i \cdot Ah_j = 0 \quad i \neq j.$$

**Tod.** Sivutetaan.

Lauseen 2 nojalla nähdään, että jos *kaksi perättäistä* vektoria ovat konjugoituja (ts.  $h_{i+1} \cdot Ah_i = 0$ ) tai kohtisuoria niin em. prosessin tuottamissa jonoissa samat ominaisuudet periytyvät koko jonoon.

Kaavojen (4) nojalla saadaan (HT)

$$(6) \quad \gamma_i = \frac{g_{i+1} \cdot g_{i+1}}{g_i \cdot g_i} = \frac{(g_{i+1} - g_i) \cdot g_{i+1}}{g_i \cdot g_i}$$

$$(7) \quad \lambda_i = \frac{g_i \cdot h_i}{h_i \cdot Ah_i}.$$

**Johtopäätös:** Jos matriisi  $A$  tunnetaan, niin kaavojen (3), (4) määrittelemä rekursio tuottaa jonon konjugoituja suuntia, joita käytetään peräkkäisissä NR::linmin:eissä. Kun on tehty  $N$  kpl NR::linmin:ejä löydetään minimi kvadraattisen kohdefunktion tapauksessa (sama seikka mainittiin jo Powellin menetelmän yhteydessä edellä). Huomaa kuitenkin, ettei matriisia  $A$  tunneta, jos  $f$  on epälineaarinen.

**8. Lause.** Olkoot  $g_i$  ja  $h_i$  konstruoituja kuten kaavoissa (3) ja (4). Oletetaan, että  $g_i = -\nabla f(P_i)$ , missä  $f$ :llä Taylor-kehitelmä (1), ja että pisteestä  $P_i$  edetään  $f$ :n  $(P_i, h_i)$  -minimiin  $P_{i+1}$  ja merk.  $g_{i+1} = -\nabla f(P_{i+1})$ . Silloin  $g_{i+1}$  on sama kuin (3):n antama  $g_{i+1}$ .

**Huom.** Lauseen 8 tilanteessa  $g_{i+1}$  konstruoitiin tuntematta  $A$ :ta.

**Tod.** Sivutetaan (kirjan todistus kovin heuristinen).

Lause 8 antaa algoritmin suuntien konstruoinnille, joka ei vaadi Hessen matriisin  $A$  tuntemista. Suuntajonon  $h_i$  konstruointiin tarvitaan vain NR::linmin:iä ja gradientin määrittämiä sekä viimeisimpien  $g_i$ :den varastointia.

Kaikki yllä sanottu soveltuu NR:n molempiin liittogradienttimenetelmiin sekä Fletcher-Reeves että Polak-Ribieren menetelmään.

Ainoa ero näissä menet. on, että FR käyttää kaavan (6) ensimmä. yhtälöä, PR taas jälkimm. yhtälöä  $\gamma_i$ :n laskemiseen. Vaikka ne algebrallisesti ovat samoja, laskennallisesti PR on osoittautunut käytännössä paremmin toimivaksi.

Algm NR: : frprmn s.423.

Liittogradientti-menetelmällä on sovelluksia harvojen yhtälöryhmien ratkaisuun (NR: : linbcg).

NR: : linmin ei vaadi derivaattaa, mutta jos derivaattaa käytetään NR: : frprmn:n yhteydessä, jossa joka tapauksessa tarvitaan gradienttia, niin tällöin olisi luontevaa käyttää NR: : linmin:in sellaista versiota, jossa myös on derivaatta mukana (s.424).

## 10.7 Kvasi-Newton menetelmät

Kvasi-Newton (muuttuvan metriikan, variable metric) menetelmien tavoite on sama kuin liittogradienttimenetelmillä, nimitt. tiedon kerääminen minimoitavasta funktiosta siten, että  $N$  peräkkäistä NR: : linmin:iä johtaa  $N$ :n muuttujan kvadraattisen funktion tapauksessa minimiin.

Muistitilan tarve  $N \times N$ . Eri variantteja:

Davidon-Fletcher-Powell (DFP)

Broyden-Fletcher-Goldfarb-Shanno (BF65)

**Idea:** Yritetään löytää jono matriiseja  $H_i$  s.e.  $H_i \rightarrow H = A^{-1}$ .

Olk.  $f(p+x) \approx f(p) - b \cdot x + \frac{1}{2}x \cdot Ax$ ,  $b = -\nabla f(p)$ . Minimipisteessä  $x_m = p$  pätee

$$(1) \quad Ax_m = b$$

ja piste  $x_i$  tot.

$$(2) \quad Ax_i = \nabla f(x_i) + b$$

(1)& (2)  $\Rightarrow$

$$(3) \quad x_m - x_i = A^{-1}[-\nabla f(x_i)].$$

Jos  $A^{-1}$  tunnettaisiin, päästäisiin kaavalla (3) pisteestä  $x_i$  suoraan minimiin. Mutta miten löydetään  $A^{-1}$  tai sen approksimaatio?

(3)  $\Rightarrow$

$$\mathbf{x}_{i+1} - \mathbf{x}_i = A^{-1}[\nabla f_{i+1} - \nabla f_i], \quad \nabla f_i = \nabla f(\mathbf{x}_i)$$

Jos  $H_{i+1}$  on  $A^{-1}$ :n likiarvo niin halutaan

$$\mathbf{x}_{i+1} - \mathbf{x}_i = H_{i+1}(\nabla f_{i+1} - \nabla f_i).$$

Menetelmien ideana on etsiä  $H_{i+1}$ :tä muodossa  $H_i +$  korjaus.

Korjaustermin lauseke DFP:ssä ja BFGS:ssä hieman erilainen.

Algm NR: :dfpmin s. 428

### 10.8 Lineaarinen ohjelmointi ja simpleksialgoritmi

Lin. ohjelmointi on synonyymi lin. optimoinnille. Lin. optimoinnin perustehtävä on, kuten alan nimestäkin ilmenee, tutkia lin. funktioiden ääriarvoja. Tehtävän standardimuoto on maksimoida lineaarinen funktio

$$(1) \quad z = a_{01}x_1 + \dots + a_{0N}x_N$$

primaariehdoin

$$(2) \quad x_i \geq 0, \quad i = 1, \dots, N$$

ja samanaikaisin lisäehdoin, joita  $M = m_1 + m_2 + m_3$  kpl

$$(3) \quad a_{i1}x_1 + \dots + a_{iN}x_N \leq b_i, \quad i = 1, \dots, m_1,$$

$$(4) \quad a_{j1}x_1 + \dots + a_{jN}x_N \geq b_j (\geq 0), \quad j = m_1 + 1, \dots, m_1 + m_2,$$

$$(5) \quad a_{k1}x_1 + \dots + a_{kN}x_N = b_k, \quad k = m_1 + m_2 + 1, \dots, m_1 + m_2 + m_3.$$

Yhteensä  $M + N$  ehtoa (2)-(5), jotka määritt. "simpleksin" tai monitahokkaan  $R^N$ :ssä. Sallitaan  $a_{ij} < 0, = 0, > 0$ . Ehto  $b_i \geq 0$  on tehty mukavuussyistä, se on normalisointi johon päästään aina tarvittaessa kertomalla  $-1$ :llä. Ei ole merkitystä sillä onko  $M < N, M = N$ , vai  $M > N$ .

Vektori  $x = (x_1, \dots, x_N)$ , joka tot. ehdot (2)-(5), on nimeltään *käypä vektori* ja maksimoitava funktio (1) on *kohdefunktio* (kuten aikaisemminkin). Kohdefunktion maksimoiva käypä vektori on *optimaalinen käypä vektori*. Optim. käypää vektoria ei löydy, jos (i) käypä vektoreita ei ole ollenkaan (ts. rajoitukset yhteensopimattomia) tai jos (ii) maksimia ei ole ts. on olemassa suunta, jossa muuttujat  $x_i$  voivat kasvaa  $+\infty$ :een s.e. ehdot toteutuvat, jolloin myös kohdefunktio  $\rightarrow +\infty$ . Jos ratkaisu on olemassa, niin se ei yleensä ole 1-käsitt. Erityisesti seuraavissa tapauksissa ratkaisun ei tarvitse olla yksikäsitteinen: jos

- (1) kohdefunktio ei rajoitettu,
- (2) ehdot ristiriitaisia ts. ratkaisua ei ole olemassa,
- (3) ratkaisu saavutetaan simpleksin kärkipisteessä,
- (4) ratkaisujoukko on simpleksin sivu.

Rajoituksista seuraa, että käyvät vektorit ovat rajoite-ehtojen määrittelemässä simpleksissä.

Tarkastellaan jatkossa seuraavaa ongelmaa:

$$(6) \quad \text{maksimoi } z = x_1 + x_2 + 3x_3 - \frac{1}{2}x_4$$

kun  $x_i \geq 0$ ,  $i = 1, \dots, 4$  ja

$$(7) \quad \begin{cases} x_1 + 2x_3 \leq 740 \\ 2x_2 - 7x_4 \leq 0 \\ x_2 - x_3 + 2x_4 \geq \frac{1}{2} \\ x_1 + x_2 + x_3 + x_4 = 9 \end{cases}$$

Ratkaisuvektori on  $x_1 = 0, x_2 = 3.33, x_3 = 4.73, x_4 = 0.95$  kuten myöhemmin tullaan näkemään.

Ratkaisuun edetään vaiheittain.

**Alustavia huomautuksia.** Koska kohdefunktio on lin. niin optimaalinen käypä vektori on simpleksin (7) reunalla. (Jos näin ei olisi, niin voitaisiin kohdefunktion gradientin suuntaan etenemällä kasvattaa kohdefunktiota.) Optimaalisen käyvän vektorin määrittelee  $N$  koordinaattia, joten sen täytyy toteuttaa  $N$  kpl ehdoista (2)-(5) s.e. = pätee. Sanotaan, että käypä vektori on *käypä perusvektori*

(feasible basic vector), jos se toteuttaa  $N$  kpl ehdoista (2)-(5) s.e. = pätee. Jos  $N > M$  niin käyvällä perusvektorilla on ainakin  $N - M$  kpl komponenteista = 0 (muuta ehtoja kuin (2) oli vain  $M$  kpl). Huomaa: Käyvän perusvektorin komponenteista enint.  $M$  kpl  $\neq 0$ .

**Lin.ohjelmoinnin peruslause.** (LOP) Jos on olem. optim. käypä vektori niin on olem. käypä perusvektori, joka on optim.

LOP palauttaa perusongelman (1) kombinatoriseen ongelmaan: Mitkä  $N$  kpl rajoituksista (2)-(5) (yhteensä  $M + N$  kpl) ovat ne, jotka liittyvät optim. käypään perusvektoriin?

Eri mahdollisuuksia kokeilemalla ja sijoittamalla kohdefunktioon (1) void. maksimointi (periaatteessa) suorittaa. Laskutoimitusten määrä kasvaa kuitenkin rajusti  $N$ :n mukana (esim. jos rajoite-ehdot määräävät suorakulmaisen särmiön  $\mathbb{R}^N$ :ssä, niin kärkipisteitä on  $2^N$ , ts. vaativuus kasvaa vähintään eksponentiaalisesti).

**Simpleksimenetelmä** (Dantzig 1948) (1):n ratkaisemiseksi perustuu kokeilun organisointiin siten, että

(a) kokeillaan joukkoa kombinaatioita s.e. kohdefunktio kasvaa joka askeleella

(b) optim. käypä ratkaisu saavutetaan ”yleensä” enint.  $\max\{M, N\}$  askeleella.

Ominaisuuden (b) todistus: 1982 S. Smale.

Simpleksimenetelmää käsittelevä kirjallisuus on laajaa, esim. mainittakoon H. A. Taha: Operations research, Macmillan 1992.

### **Simpleksimenet. rajoitetulle normaalimuot. tehtävälle**

LO-ongelma (1) on *normaalimuodossa*, jos ehtoja (3) ja (4) ei ole vaan ainoat rajoitukset ovat ei-neg. ehto (2) ja yhtälöt (5), ts.  $m_1 = m_2 = 0$ .

**Rajoitettu normaalimuoto:** Vaadit. lisäksi että kussakin yhtälössä (5) on ainakin yksi muuttuja *posit.* kertoimella ja ettei se esiinny muissa rajoiteyhtälöissä (5). Tällöin kukin yhtälöistä (5) void. ratk. ko. muuttujan suhteen. Näin valittuja muuttujia kutsutaan *perusmuuttujiksi* tai vasemman puolen muuttujiksi. Näitä on  $M (= 0 + 0 + m_3 = m_3)$  kpl. Muita muuttujia (oikeanpuoleisia muuttujia)

on  $N - M$  kpl. Rajoitettu normaalimuoto voidaan saavuttaa vain jos  $M \leq N$  kuten jatkossa oletetaan.

Myöhemmin osoitetaan miten jokainen LO-ongelma voidaan palauttaa rajoitettuun normaalimuotoon (mm. muuttujien lukumäärää joudutaan mahdollisesti keinotekoisesti lisäämään).

Kun perusmuuttujat ratkaistaan rajoiteyhtälöistä (5) ja oikeanpuol. muuttujat aset. = 0 niin saadaan heti eräs rajoitetussa normaalimuodossa olevan tehtävän käypä perusvektori (jonka ei tarvitse olla optimaalinen)

**Esim.** Maksimoi  $z = 2x_2 - 4x_3$  ( $x_i \geq 0$ ) rajoiteyhtälöin

$$\begin{cases} x_1 = 2 - 6x_2 + x_3 \\ x_4 = 8 + 3x_2 - 4x_3. \end{cases}$$

$x_1, x_4$  perusmuuttujat

$x_2, x_3$  oikeanpuoleiset muuttujat

Tehtävä taulumuodossa:

		$x_2$	$x_3$	
$z$	0	2	-4	
$x_1$	2	-6	1	
$x_4$	8	3	-4	

Edetään neljässä askeleessa.

1) **Askel 1** Etsitään ne  $z$ -rivin alkioit, joissa positiivinen kerroin. Jos tällaista positiiviseen alkioon liittyvää oikeanpuoleista muuttujaa kasvatetaan, niin myös kohdefunktio kasvaa.

2) **Askel 2** Pivotin valinta. Tutkitaan edellä valittujen oikeanpuoleisten muuttujien sarakkeita yo. taulussa. Etsitään vastausta kysymykseen: Paljonko tällaista valittua oikeanpuoleista muuttujaa voidaan kasvattaa (kun muut oikeanpuol. muuttujat = 0) ennen kuin jonkin vas. puolen muuttuja tulee negatiiviseksi (mikä on kiellettyä (2):n nojalla). Jos kaikki kertoimet  $> 0$  ko. oikeanpuol. muuttujan sarakkeessa niin kohdefunktio ei ole rajoitettu ja maksimia ei ole ( $\Rightarrow$  done), Jos taas ko. oikeanpuoleisen muuttujan

sarakkeessa on yksi tai useampia negat. alkioita, niin on tutkittava, mikä tällaisista ensimmäiseksi rajoittaa ko. sarakkeen oik.puol. muuttujan kasvua. Rajoitus ko. oik.puolen muuttujan kasvulle saadaan osamäärästä:

ko. vaakarivin vakiosarakkeen luku

---

ko. oikeanpuol. muuttujan kerroin (ko. vaakarivillä)

Ko. oikeanpuoleisen muuttujan sarakkeen pienin näin saatu osamäärä antaa tiukimman rajoitteen  $r_j$ . Ko oikeanpuol. muuttujan kohdefunktion aiheuttama kasvu on  $r_j \times z$ -rivin ko. alkio. Toistetaan tämä kaikille sell. oikeanpuol. muuttujille, joille  $z$ -rivin kerroin  $> 0$ . Näistä suurimman kasvun kohdefunktioon antava on "pivotti".

Yo. esimerkin tapauksessa ainoa negatiivinen kerroin  $x_2$ :n sarakkeessa on  $-6$ , joka sallii  $x_2$ :n kasvaa  $2/6$  :ksi, mikä puolestaan antaa kohdefunktioon kasvun  $2 \times 2/6 = 2/3$ .

3) **Askel 3** Pivot informaation käyttö. Ratk. pivot alkioita vast. yhtälö (toteutetaan kasvatus)

$$x_1 = 2 - 6x_2 + x_3 \Rightarrow x_2 = \frac{1}{3} - \frac{1}{6}x_1 + \frac{1}{6}x_3.$$

Seuraavaksi sijoitetaan tämä  $z$ -riviin

$$z = 2x_2 - 4x_3 = 2\left(\frac{1}{3} - \frac{1}{6}x_1 + \frac{1}{6}x_3\right) - 4x_3 = \frac{2}{3} - \frac{1}{3}x_1 - \frac{11}{3}x_3$$

sekä muihin vas.puolen muuttujien riveihin (esimerkin tapauksessa vain  $x_4$ -rivi). Saadaan uusi taulu

		$x_1$	$x_3$
$z$	$\frac{2}{3}$	$-\frac{1}{3}$	$-\frac{11}{3}$
$x_2$	$\frac{1}{3}$	$-\frac{1}{6}$	$\frac{1}{6}$
$x_4$	9	$-\frac{1}{2}$	$-\frac{7}{2}$

4) **Askel 4** Toisto. Toistetaan yo. menettely ja yritetään kasvattaa kohdefunktiota (toistetaan kunnes kaikki  $z$ -rivillä olevat oikean



puoleisten muuttujien kertoimet  $< 0$ , joka osoittaa, ettei enää voida kasvattaa). Yo. esim. tapauksessa näin on jo asianlaita ja vastaus on:

$z$ :n maksimi on  $\frac{2}{3}$ , jonka antaa

$$x_1 = x_3 = 0, \quad x_2 = \frac{1}{3}, \quad x_4 = 9.$$

**Huom.** Kun  $n = 2$  voidaan LO-ongelmia ratkaista myös graafisesti. Esim. Maksimoi  $x_1 + x_2$  kun  $x_1, x_2 \geq 0$  ja  $\begin{cases} x_1 + 3x_2 \leq 5 \\ 2x_1 + x_2 \leq 3 \end{cases}$

$$-\frac{1}{3}x_1 + \frac{5}{3} = -2x_1 + 3$$

$$\frac{5}{3}x_1 = \frac{4}{3} \Rightarrow x_1 = \frac{4}{5}$$

$$\Rightarrow x_2 = -\frac{8}{5} + 3 = \frac{7}{5}$$

$$x_1 + x_2 = \frac{4}{5} + \frac{7}{5} = \frac{11}{5}$$

### Kertausta: LO ja simpleksialgoritmi

**Teht.** Maksimoi

$$(1) \quad z = a_{01}x_1 + \dots + a_{0N}x_N \quad (\text{kohdeftio})$$

kun

$$(2) \quad x_1 \geq 0, \dots, x_N \geq 0$$

ja samanaik. toteutuvat seur.  $M = m_1 + m_2 + m_3$  ehto

$$(3) \quad a_{i1}x_1 + \dots + a_{iN}x_N \leq b_i, \quad (b_i \geq 0) \quad i = 1, \dots, m_1,$$

$$(4) \quad a_{j1}x_1 + \dots + a_{jN}x_N \geq b_j, \quad j = m_1 + 1, \dots, m_1 + m_2,$$

$$(5) \quad a_{k1}x_1 + \dots + a_{kN}x_N = b_k, \quad k = m_1 + m_2 + 1, \dots, m_1 + m_2 + m_3$$

$x = (x_1, \dots, x_N)$  on käypä vektori, jos se tot. (2)-(5).

**Normaalimuoto:**  $m_1 = m_2 = 0$ .

**Rajoitettu norm.muoto:**  $m_1 = m_2 = 0$  ja lisäksi kukin yhtälöstä (5) sisältää vähintään yhden muuttujan posit. kertoimella, joka ei esiinny muissa yhtälöissä ( $\Rightarrow M \leq N$ ). Tällaisia muuttujia sanotaan *perusmuuttujiksi* l. *vas.puolen muutt.*

**Esim.** Maksimoi  $z = 2x_2 - 4x_3$ ,  $x_i \geq 0$ , kun

$$x_1 = 2 - 6x_2 + x_3$$

$$x_4 = 8 + 3x_2 - 4x_3$$

Sama taulumuodossa:

		$x_2$	$x_3$	
$z$	0	2	-4	
$x_1$	2	-6	1	
$x_4$	8	3	-4	

### Simpleksialgoritmi rajoitetulle normaalimuotoiselle tehtävälle

4. vaihetta:

1. **Vaihe**  $z$ -rivin posit. kertoimien valinta [Millä oikean puolen muuttujilla on kasvattavaa vaikutusta kohdefunktioon?]

2. **Vaihe** Pivotin (1. navan) valinta

Tutkitaan ed. valittujen (posit. kertoimisten) muuttujien vaikutus kohdefunktion rajoite-ehtojen vallitessa ja etsitään eniten vaikuttava seuraavasti. Kiinnitetään oikean puolen muuttuja  $x_j$ . Etsitään  $x_j$ :n maksim. arvo "kasvatusraja" asettamalla muut  $x_i$ :t = 0 (löydetään tutkimalla  $x_j$ -sarakkeen alla olevien negat. lukuja vastaavia rivejä: pienin näin lyödetty raja on  $x_j$ :n "kasvatusraja"). Saadaan  $x_j$ :n kontribuutio kohdefunktioon = "kasvatusraja",  $z$ -rivin kerroin Valit. suurimman kontribuution antava  $x_j$ . Sen "kasvatusrajan" antava kerroin on *pivotti*. (Esim. -6)

3. **Vaihe** Pivotti-informaation käyttö

Ratkaistaan pivottia vastaava rajoiteyhtälö

$$x_1 = 2 - 6x_2 + x_3 \Rightarrow x_2 = \frac{1}{3} - \frac{1}{6}x_1 + \frac{1}{6}x_3$$

Sitten sijoitetaan tämä  $z$ -riviin

$$z = 2x_2 - 4x_3 = 2\left(\frac{1}{3} - \frac{1}{6}x_1 + \frac{1}{6}x_3\right) - 4x_3 = \frac{2}{3} - \frac{1}{3}x_1 - \frac{11}{3}x_3$$

ja muihin (kuin pivointiin) vasemman puoleisten muuttujien riveihin: (tässä esim. vain  $x_4$ )

$$x_4 = 8 + 3\left(\frac{1}{3} - \frac{1}{6}x_1 + \frac{1}{6}x_3\right) - 4x_3 = 9 - \frac{1}{2}x_1 - \frac{7}{2}x_3$$

Saadaan uusi taulu

		$x_1$	$x_3$
$z$	$\frac{2}{3}$	$-\frac{1}{3}$	$-\frac{11}{3}$
$x_2$	$\frac{1}{3}$	$-\frac{1}{6}$	$\frac{1}{6}$
$x_4$	9	$-\frac{1}{2}$	$-\frac{7}{2}$

4. **Vaihe.** Toisto. Toistetaan yo. menettely saatuun tauluun ja yritetään kasvattaa kohdeftiota (toistetaan kunnes kaikki  $z$ -rivin alkio  $< 0$ , joka osoittaa ettei enää voida kasvattaa). Esim. tapuksessa näin on jo asianlaita ja vastaus on: Kohdeftion  $z$  maksimi on  $\frac{2}{3}$ , jonka tuottavat  $x_1 = x_3 = 0$ ,  $x_2 = 1/3$ ,  $x_4 = 9$ .

**Huom.** Tapaus, missä vakiosarakkeen alkio  $= 0 \Leftrightarrow x \equiv 0$  käypä vektori) voi vaatia erikoiskäsittelyn (esim. vas. ja oik. puolen muuttujien vaihto)

**Kertaus päättyy tähän.**

### Simpleksialgoritmi yleisessä tapauksessa

**Tavoite:** Yleisen LO-ongelman (1)-(5) palautus rajoitettuun normaalimuotoon.

**Esim.** Maksimoi (6)  $z = x_1 + x_2 + 3x_3 - \frac{1}{2}x_4$ , kun  $x_i \geq 0$  ja

$$(7) \quad \begin{cases} x_1 + 2x_3 \leq 740 \\ 2x_2 - 7x_4 \leq 0 \\ x_2 - x_3 + 2x_4 \geq \frac{1}{2} \\ x_1 + x_2 + x_3 + x_4 = 9 \end{cases}$$

Ensin halutaan päästä eroon tyyppiä (3)& (4) olevista rajoituksista (yo. esim.:ssä (7):n kolme ensimmäistä rajoitusta). Tämä toteutetaan ottamalla käyttöön uusia muuttujia.  $y_i$  ns. *välismuuttujia*,

jotka toteuttavat  $y_i \geq 0$  ja jotka poistavat epäyhtälössä olevan ”väljyyden” ja muuntavat ey:t yhtälöiksi (slack variable). Näitä on  $m_1 + m_2$  kpl ja laskujen aikana niitä käsitellään kuten muita muuttujia; lopuksi ne kuitenkin jätetään huomiotta. Näin yo. esim. tulee muotoon

$$(8) \quad \begin{cases} x_1 + 2x_3 + y_1 = 740 \\ 2x_2 - 7x_4 + y_2 = 0 \\ x_2 - x_3 + 2x_4 - y_3 = \frac{1}{2} \\ x_1 + x_2 + x_3 + x_4 = 9 \end{cases}$$

Toiseksi halutaan muuttaa em. tavalla muokattu ongelma rajoitettuun normaalimuotoon. Tätä varten tarvitaan kaksi seikkaa vrt. määritelmä (1) kerroin  $> 0$ , (2)  $N \geq M$ . Molemmat seikat saadaan voimaan ottamalla käyttöön *keinomuuttujat*  $z_i$  (artificial variable). Nyt (8) saa muodon

$$(9) \quad \begin{cases} z_1 = 740 - x_1 - 2x_3 - y_1 \\ z_2 = -2x_2 + 7x_4 - y_2 \\ z_3 = \frac{1}{2} - x_2 + x_3 - 2x_4 + y_3 \\ z_4 = 9 - x_1 - x_2 - x_3 - x_4 \end{cases}$$

Esim. on nyt rajoitetussa normaalimuodossa. Tämä  $\Leftrightarrow$  alkuperäisen kun  $z_i = 0$  kaikilla  $i$ . Vielä 2 askelta:

1. **Askel** Korvataan kohdeftio (yo. esim.(6)) apukohdeftiolla

$$(10) \quad z' = -z_1 - z_2 - z_3 - z_4 = -\left(749\frac{1}{2} - 2x_1 - 4x_2 - 2x_3 + 4x_4 - y_1 - y_2 + y_3\right)$$

Suoritetaan simpleksialgoritmi kohdeftioon (10) rajoituksin (9). Kohdeftio maksimoituu ei-negatiivisilla  $z_i$ :n arvoilla jos  $z_i = 0$ . Odotamme, että simpleksialgoritmi antaa  $x_i, y_i$ :n vas. puolen ja  $z_i$ :t oikeanpuol. muuttujiksi. Unohdetaan  $z_i$ :t, jolloin jää  $x_i, y_i$ :t sisältävä probleema rajoitetussa normaalimuodossa.

2. **Askel.** Ratk. 1. askeleen tuottama ongelma mutta nyt alkuperäisen kohdeftiolla.

	$x_1$	$x_2$	$x_3$	$x_4$	$y_1$	$y_2$	$y_3$	
$z$	0	1	1	3	$-\frac{1}{2}$	0	0	0
$z_1$	740	-1	0	-2	0	-1	0	0
$z_2$	0	0	-2	0	7	0	-1	0
$z_3$	$\frac{1}{2}$	0	-1	1	-2	0	0	1
$z_4$	9	-1	-1	-1	-1	0	0	0
$z'$	$-749\frac{1}{2}$	2	4	2	-4	1	1	-1

Yo. tauluun liittyvä LO-tehtävä on raj.norm. muodossa, joten se voidaan ratkaista edellä esitetyllä menettelyllä. Saadaan seuraava taulu kirjan algoritmin NR::simplex avulla.

		$x_1$	$y_2$	$y_3$
$z$	17.03	-.95	-.05	-1.05
$x_2$	3.33	-.35	-.15	.35
$x_3$	4.73	-.55	.05	-.45
$x_4$	.95	-.10	.10	.10
$y_1$	730.55	.10	-.10	.90

Algoritmin syötetietoina on tämän sivun ensimmäisen taulun paksunnetun viivoituksen sisään jäävä osa.

Algm NR::simplx s.439

**Input:** tehtävä taulumuodossa (paksunnettujen viivojen sisälle jäävä osa taulusta) matriisina  $A$

**Output:**  $A$  uudelleen indeksoitava  $A$ :n  $J + 1$ :llä rivillä on nyt muuttuja  $x(IPOS)V(J)$ ,  $J = 1, \dots, M$

$POSV(J) > N \Rightarrow y$ -muuttuja ts.  $x_{N+J} = y_j$

$A$ :n  $J + 1$ :ssä sarakkeessa nyt muuttuja  $x(IZROV(J))$ ,  $J = 1, \dots, N$ . Jos  $IZROV(J) > N$  niin  $x_{N+j} = y_j$  ja, jos  $IZROV(J) > N + m_1 + m_2$ , niin ko.  $A$ :n  $J + 1$ :s sarake pitää jättää vaille huomiota.

```
// FILE: mysimplx.cpp begins
#include <cstdio>
```

```

#include <cstdlib>
#include <string>
#include <iostream>
#include <iomanip>
#include "nr.h"
using namespace std;

// Driver for routine simplex

int main(void)
{
    const int N=4,M=4,NP=N+1,MP=M+2;
    const int M1=2,M2=1,M3=1;        // M1+M2+M3 = M
    const int NM1M2=N+M1+M2;
    int i,j,icase;
    DP c_d[MP*NP]=
    {0.0,1.0,1.0,3.0,-0.5,
     740.0,-1.0,0.0,-2.0,0.0,
     0.0,0.0,-2.0,0.0,7.0,
     0.5,0.0,-1.0,1.0,-2.0,
     9.0,-1.0,-1.0,-1.0,-1.0,
     0.0,0.0,0.0,0.0,0.0};
    string txt[NM1M2]=
    {"x0","x1","x2","x3","y0","y1","y2"};
    Vec_INT izrov(N),iposv(M);
    Mat_DP a(c_d,MP,NP);

    NR::simplex(a,M1,M2,M3,icase,izrov,iposv);
    if (icase == 1)
        cout << endl << "unbounded objective function" << endl;
    else if (icase == -1)
        cout << "no solutions satisfy constraints given" << endl;
    else {
        cout << endl << setw(11) << " ";
        for (i=0;i<N;i++)
            if (izrov[i] < NM1M2) cout<<"          "<<txt[izrov[i]];
        cout << endl << endl;
        cout << fixed << setprecision(3);
        for (i=0;i<=M;i++) {
            if (i == 0 || iposv[i-1] < NM1M2) {
                if (i > 0)
                    cout << txt[iposv[i-1]];
                else
                    cout << " ";
                cout << setw(10) << a[i][0];
            }
        }
    }
}

```

```

        for (j=1;j<=N;j++)
            if (izrov[j-1] < NM1M2)
                cout << setw(10) << a[i][j];
            cout << endl;
        }
    }
}
return 0;
}
// FILE: mysimplx.cpp ends
/* Output:
           x0          y1          y2
y0         17      -0.95      -0.05      -1.05
x1         731       0.1       -0.1       0.9
x3         3.33     -0.35     -0.15     0.35
x2         0.95     -0.1       0.1       0.1
x2         4.72     -0.55     0.05     -0.45
*/

```

Esitämme seuraavaksi mysimplx.cpp:n muokatun version mysix4.cpp, joka on pyritty tekemään joustavammaksi käyttää. Komennolla

./mysix4 <mysix4.inp >mysix4.out  
ohjelma ratkaisee saman tehtävän kuin mysimplx.cpp. Tiedosto mysix4.inp on seuraava:

```

4 2 1 1
1 1 3 -0.5
740 0 0.5 9
1 0 2 0
0 2 0 -7
0 1 -1 2
1 1 1 1
n
// FILE: mysix4.cpp begins
// g++ mysix4.cpp -L../lib -I../utils -I../democpp02 -o a -lm -lnr
// Usage: ./a < file.inp where file.inp is at the end of program
/* This program solves:
Maximize z = \sum_{p=1}^N a_{0p}x_p
under
x_p >= 0, p =1,...,N
\sum_{p=1}^N a_{ip}x_p <= b_i, i =1,...,m_1

```

```

\sum_{p=1}^N a_{jp}x_p \geq b_j, j =m_1+1,\dots,m_1+m_2
\sum_{p=1}^N a_{kp}x_p = b_k, k =m_1+m_2+1,\dots,m_1+m_2+m_3
where all b_i \geq 0.
*/

#include <cstdlib> // Used in putmat2
#include <cstdio> // Used in putmat2
#include <ctime>
#include <string>
#include <fstream>
#include <iostream>
#include <iomanip>
#include <cmath>
#include "nr.h"
#include "matut102.cpp"

using namespace std;

#define u1 10
#define u2 11
#define u3 12
#define gln u1
#define glm u1
#define glnp u2 // np \geq n+1
#define glmp u3 // mp \geq m+2
#define glnm1m2 u3

void writemat(Mat_DP &a, int p1, int p2)
{
    char c='y';
    int i,j;
    cout<<"Problem in tabular form is:"<<endl;
    cout<<setw(17)<<" ";
    for(i=1;i<=p1;i++) cout<<"          x"<<i;
    cout<<endl;
    for(i=1;i<=p1;i++) {
        if(i==1) cout<<"z          ";
        if(i>1) cout<<"z"<<i-1<<"          ";
        for(j=1;j<=p2;j++) {
            cout.precision(3); // *****
            cout << scientific << setw(11) <<a[i][j];
        }
        cout<<endl;
    }
    cout<<"Type y if you want to correct an element"<<endl;
}

```



```

cin>>c;
while(c=='y') {
    cout<<"Enter indices of the element"<<endl;
    cin>>i>>j;
    cout<<"Enter now the new element"<<endl;
    cin>>a[i][j];
    cout<<"Matrix A is:"<<endl;
    for(i=1;i<=p1;i++) {
        for(j=1;j<=p2;j++) {
            cout.precision(2);
            cout <<scientific <<a[i][j] << " ";
        }
        cout<<endl;
    }
    cout<<"Type y if you want to correct an element"<<endl;
    cin>>c;
}
}

void enterproblem(Mat_DP &a, int *n, int *m, int *np, int *mp,
                 int *nm1m2, int *m1, int *m2, int *m3)
{
    int i,j;
    cout<<"This program solves the LP-problem on p. 430 of NR."<<endl;
    cout<<"Please specify now the problem (n+m1+m2+m3<11):"<<endl;
    cout<<"Enter first integers n,m1,m2,m3 >=0:"<<endl;
    cin>>*n>>*m1>>*m2>>*m3;
    *m=*m1+ *m2+ *m3;
    *np=*n+1; *mp= (*m)+2;
    *nm1m2=*n + *m1 + *m2;
    cout<<"Enter now the "<<*n<<" coefficients of the object";
    cout<<"function:"<<endl;
    cout<<"Constant term must be zero as on p. 430."<<endl;
    for(i=1;i<=*n;i++) cin >>a[1][i+1];
    a[1][1]=0;
    for(i=*n+2;i<=u2;i++) a[1][i]=0;
    cout<<"Enter now the "<<*m<<" non-negative coefficients b[i]"<<endl;
    for(i=1;i<=*m;i++) cin>>a[i+1][1];
    for(i=*m+2;i<=u3;i++) a[i][1]=0.0;
    for(i=2;i<=(*m1+1);i++) a[i][*n+2]=-1;
    for(i=*m1+2;i<=(*m1+*m2+2);i++) a[i][*n+2]=1;
    for(i=*m1+*m2+3;i<= *m1+*m2+*m3+3;i++) a[i][*n+2]=0;
    if(*m1>0) {
        cout<<"Enter first coefficients in restrictions of the form <="<<endl;
        for(i=2;i<= *m1+1;i++) {

```

```

        for(j=2;j<= *n+1;j++) {
            cin>>a[i][j];
            a[i][j]=-a[i][j];
        }
    }
}
if(*m2>0) {
    cout<<"Enter next coefficients in restrictions of the form >="<<endl;
    for(i=*m1+2;i<= *m1+m2+1;i++) {
        for(j=2;j<= *n+1;j++) {
            cin>>a[i][j];
            a[i][j]=-a[i][j];
        }
    }
}
if(*m3>0){
    cout<<"Enter finally coefficients in restrictions of the form =:"<<endl;
    for(i=*m1+m2+2;i<= *m1+m2+m3+1;i++) {
        for(j=2;j<=(*n+1);j++) {
            cin>>a[i][j];
            a[i][j]=-a[i][j];
        }
    }
}
}

int main()
{
    int i,icase,j,n,m,np,mp,nm1m2;
    Vec_INT izrov(gln);
    Vec_INT iposv(glm);
    Mat_DP aa(glmp+1,glnp+1);
    Vec_DP coefz(glmp);
    DP s=0.0;
    const char *ch2;
    int m1,m2,m3;          // m1+m2+m3=m
    enterproblem(aa,&n,&m,&np,&mp,&nm1m2,&m1,&m2,&m3);
    writemat(aa,m+1,n+1);
    Mat_DP a(m+2,n+1);
    for (i=0;i<m+2;i++)
        for (j=0;j<n+1;j++)
            a[i][j]=aa[i+1][j+1];
    for (i=0;i<gln;i++) coefz[i]=a[0][i];
    NR::simplx(a,m1,m2,m3,icase,izrov,iposv);
}

```

```

if (icase == 1)
    cout << endl << "unbounded objective function" << endl;
else if (icase == -1)
    cout << "no solutions satisfy constraints given" << endl;
else {
    cout << endl << setw(15) << " ";
    for (i=0;i<n;i++)
    {
        ch2=" x";
        if (izrov[i] < nm1m2) {
            if (izrov[i]<n)
                cout<<setw(8)<<ch2<<izrov[i]+1<<" ";
            else {
                ch2=" y";
                cout<<setw(8)<<ch2<<izrov[i]+1-n<<" ";
            }
        }
    }
    cout << endl;
    cout << fixed << setprecision(3);
    for (i=0;i<=m;i++)
    {
        if (i == 0 || iposv[i-1] < nm1m2) {
            ch2=" x";
            if (i > 0)
            {
                if (iposv[i-1]<n)
                    cout<<ch2<<iposv[i-1]+1<<" ";
                else {
                    ch2=" y";
                    cout<<ch2<<iposv[i-1]+2-n<<" ";
                }
            }
        }
        else
            cout << " ";
        cout << setw(10) << a[i][0];
        for (j=1;j<=n;j++)
            if (izrov[j-1] < nm1m2)
                cout << setw(10) << a[i][j];
        cout << endl;
    }
}
}
}

cout<<"\niposv:";

```

```

for (i=0;i<glm;i++) cout<<setw(2)<<iposv[i];
cout<<"\nizrov:";
for (i=0;i<gln;i++) cout<<setw(2)<<izrov[i];
s=0.0;
cout<<"\nMax. value of object function:\n z= ";

for (i=0;i<m;i++) {
    if ((iposv[i]>-1) &&(iposv[i]<n)) {
        if (i>0) cout<<" ";
        cout<<"("<<coefz[iposv[i]+1]<<")*("<<a[i+1][0]<<")";
        s += coefz[iposv[i]+1]*a[i+1][0];
    }
}
printf("\n = %10.4lf\n",s);
showmat2(a, " %8.4lf");
return 0;
}
// FILE: mysix4.cpp ends.
// Example input file mysix4.inp:
// to solve problem (10.8.7) on p. 432 /NR
4 2 1 1
1 1 3 -0.5
740 0 0.5 9
1 0 2 0
0 2 0 -7
0 1 -1 2
1 1 1 1
n

```

Tulos kirjoitetaan tiedostoon NR::mysix4.out :

```

This program solves the LP-problem on p. 430 of NR.
Please specify now the problem (n+m1+m2+m3<11):
Enter first integers n,m1,m2,m3 >=0:
Enter now the 4 coefficients of the objectfunction:
Constant term must be zero as on p. 430.
Enter now the 4 non-negative coefficients b[i]
Enter first coefficients in restrictions of the form <=:
Enter next coefficients in restrictions of the form >=:
Enter finally coefficients in restrictions of the form =:
Problem in tabular form is:

```

		x1	x2	x3	x4	x5
z	0	1	1	3	-0.5	
z1	740	-1	-0	-2	-0	
z2	0	-0	-2	-0	7	
z3	0.5	-0	-1	1	-2	

```

z4          9          -1          -1          -1          -1
Type y if you want to correct an element

```

```

          x1          y2          y3
          17         -0.95        -0.05        -1.05
y2        731          0.1         -0.1         0.9
x2        3.33        -0.35        -0.15         0.35
x4        0.95        -0.1         0.1         0.1
x3        4.72        -0.55         0.05        -0.45

```

```
iposv: 4 1 3 2 0 0 0 0 0 0
```

```
izrov: 0 5 7 6 0 0 0 0 0 0
```

```
Max. value of object function:
```

```

z= +(1)*(3.33)+(-0.5)*(0.95)+(3)*(4.72)
= 17.0250

```

```
6x5 matrix:
```

```

17.0250  -0.9500  -0.0500  -1.9500  -1.0500
730.5500  0.1000  -0.1000  1.1000  0.9000
 3.3250  -0.3500  -0.1500  -0.3500  0.3500
 0.9500  -0.1000  0.1000  -0.1000  0.1000
 4.7250  -0.5500  0.0500  -0.5500  -0.4500
 0.0000  0.0000  0.0000  -1.0000  0.0000

```

## 10.9 Kombinatorinen minimointi

Kombinatorinen minimointi tarkastelee minimointia tilanteessa, jossa tapahtuma-avaruus on diskreetti. Tyypillisiä esimerkkejä ovat erilaiset verkkoihin liittyvät ongelmat. Ainoa tällainen ongelma, jota tässä luvussa tarkastellaan on kauppamatkustajan ongelma. Jatkuvan tapahtuma-avaruuden tilanteessa oli mielekäästä puhua vähenemissuunnista, mutta diskreetissä tapauksessa ei tälle käsitteelle ole luontevaa geometrista tulkintaa. Muutenkin geometrisen mielikuvan muodostaminen kohdefunktiosta on hankalampaa.

**Kauppamatkustajan ongelma.** (Travelling salesman problem (TSP).) On etsittävä annetun  $n$ :nnän kaupungin kautta käyvä lyhin reitti.

Koska jonon  $\{1, \dots, n\}$  permutiointeja on  $n! \approx \sqrt{2\pi n} n^{n+\frac{1}{2}} e^{-n}$  kappaletta ja permutioinnit voidaan samaistaa kulkureittien kanssa, saadaan tästä yläraja TSP:n vaativuudelle. Alarajan löytäminen

taitaa edelleenkin olla avoin ongelma. TSP on vaativuusluokkaa NP oleva ongelma, sille ei tunneta polynomista algoritmia.

TSP on erittäin tärkeä käytännössä. Esimerkiksi voidaan ajatella teollisuusrobotin toimintaa, joka iskee peltilevyyn tiettyihin kohtiin niitit. Työn tehokkuus riippuu suoraan TSP:n ratkaisusta. Samoin puhelinliikenteen reititys. Sovellusten vuoksi TSP on tullut keskeiseksi alan ongelmaksi. Etsittäessä tehokkaampia algoritmeja globaalin minimin löytämiseksi on menestyksellisesti kokeiltu fysiikasta peräisin olevia ideoita. Laskennallisessa fysiikassa on luotu matemaattisia malleja kappaleen jähmettymiselle, joita voidaan soveltaa TSP:hen.

Kuumien metallien jähmettymisessä ja kiteytymisessä lopputulos riippuu oleellisesti siinä mekanismista, jolla lämpötilaa laskeaan. Jos lämpötilaa lasketaan hitaasti molekyylit ”löytävät oman paikkansa” ja muodostuu kiteitä (potentiaalienergian minimiä vastaava konfiguraatio). Jos taas lasketaan liian nopeasti, näin ei käy (karkaisu).

Normaalit minimointimenetelmät pyrkivät mahdollisimman nopeasti minimiin (nopea jäähtyminen) ja ne pysähtyvät tavallisesti lokaaliin minimiin. Yo. hitaan jäähtymisen (annealing) ideaa numeerikkaan on soveltanut Metropolis et al. (1953).

Kirjan `algm NR::anneal` s.448 ratkaisee kauppamatkustajan ongelman yo. termodynaamisen filosofian hengessä. Kirjamme nykyinen 2. laitos esittää myös miten annealing menetelmää voidaan soveltaa myös tilanteessa, jossa tapahtuma-avaruus on jatkuva. Toinen ohjelma osoittaa miten annealing-henkisesti voidaan muokata `NR::amoeba` algoritmia. Tehtävänä on minimoida funktio, joten tapahtuma-avaruus on ko. tapauksessa jatkuva.

```
// FILE: myann2.cpp begins
// g++ myann2.cpp -L../lib -I../utils -I../democpp02 -I../gnuplot02 -o a -lm -lnr
// This program generates random city coordinates and finds
// for the Travelling Salesman Problem (TSP) the shortest route
// that passes through all of them. Method: anneal.
// This procedure is repeated several times and the histogram
```

```

// of route lengths is displayed.
#include <cstdlib> // Used in putmat2
#include <cstdio> // Used in putmat2
#include <ctime>
#include <string>
#include <fstream>
#include <iostream>
#include <iomanip>
using namespace std;
#include "nr.h"
#include "matut102.h"
// #define PRINT 1
#include "plot.c"
using namespace std;
// Driver for routine anneal

void hist(Vec_DP x, int nrbins, const char* mytitle)
{
    FILE *fp;
    Mat_DP xy(nrbins,2);
    int i,j, tiedostoon=0;
    double xmin=x[(1-1)], xmax=x[(1-1)],s=0.0, dx=0.0,count=0.0;
    for(i=(1-1);i<x.size();i++) {
        if(x[i]<xmin) xmin=x[i];
        if(x[i]>xmax) xmax=x[i];
        s+=x[i];
    }
    for(i=(1-1);i<(nrbins);i++) {
        // xy[i][0]=i*1.0;
        xy[i][0]=xmin+(i+0.5)*(xmax-xmin)/nrbins;
        xy[i][1]=0.0; }
    dx=xy[1][0]-xy[0][0];
    dx=(dx>0)? dx: 1.0;
    for(i=(1-1);i<x.size();i++)
        for(j=0;j<nrbins;j++)
            {
                if ((j<nrbins-1)&&
                    ((x[i]-xmin)>=((xmax-xmin)/(nrbins))*j*1.0)&&
                    (x[i]-xmin)<(((xmax-xmin)/(nrbins))*(j+1.0)))) xy[j][1]+=1.0;
                else
                    if ((j==nrbins-1)&&
                        ((x[i]-xmin)>=((xmax-xmin)/(nrbins))*j*1.0)&&
                        (x[i]-xmin)<(((xmax-xmin)/(nrbins))*(j+1.0)))) xy[j][1]+=1.0;
            }
    for(j=0;j<nrbins;j++) count = (count<xy[j][1])? xy[j][1]: count;
}

```

```

fp = fopen("z.dat","w");
for (i=(1-1);i<(nrbins);i++)
    fprintf(fp,"%lf    %lf\n",xy[i][(1-1)],xy[i][1]);
fclose(fp);
// gnuplt2("z.dat","z.dat",5,NULL);
fp = fopen("x1x2.dat","w");
fprintf(fp,"%lf10.5  0.000 \n %lf10.5  %lf \n",xmin-1.0*dx,xmax+1.0*dx,
        (double)count+1.0); // Extreme corners of plotting area
fclose(fp);
fp=fopen("gnuplt.cmd","w");
if (tiedostoon==1)
    {
        fprintf(fp,"set terminal postscript\n");
        fprintf(fp,"set output \"a.ps\"\n");}
fprintf(fp,"set title \"%s/hist3 z.dat ,Average=%5.1f\" \n",\
        mytitle,s/x.size());
fprintf(fp,"set timestamp\n");
fprintf(fp,"set grid\n");
fprintf(fp,"plot \"%s\" w boxes lw 4, \"x1x2.dat\" w p\n","z.dat");
fprintf(fp,"pause -1");
fclose(fp);
cout<<xy<<endl;
cout<<x<<endl;
system("gnuplot gnuplt.cmd");
}

int MyPlot(Vec_DP x0,Vec_DP y0, Vec_DP x1,Vec_DP y1,
          Vec_DP x2,Vec_DP y2)
/*
Plots three curves x0,y0, x1,y1,  x2, y2.
Here x0, y0 have equal length likewise for x1,y1 and x2,y2.
But the lengths of x0 and x1 need not be the same.
*/
{
    int n0=x0.size(), n1=x1.size(),n2=x2.size();
    const char *fname[10];
    fname[0]="z0.dat";
    fname[1]="z1.dat";
    fname[2]="z2.dat";
    for (int j=0;j<3;j++)
        {
            ofstream fp(fname[j]);
            if (fp.fail())
                {cout << "File " << fname[j]
                 <<" could not be opened.\n"<<endl;
                }
        }
}

```



```

        abort();}
    if (j==0) for (int i=0;i<n0;i++)
        fp<<x0[i]<<" "<<y0[i]<<endl;
    if (j==1) for (int i=0;i<n1;i++)
        fp<<x1[i]<<" "<<y1[i]<<endl;
    if (j==2) for (int i=0;i<n2;i++)
        fp<<x2[i]<<" "<<y2[i]<<endl;
    fp.close();}
plot(fname[0],"b-3",fname[1],"b-3",fname[2],"ks3",NULL);
return 0;
}

double length(Vec_DP x,Vec_DP y, Vec_INT iorder)
// This is the length of the closed polygonal line
// with vertices at (x,y)
{
    DP s=pow(pow(x[iorder[0]] -x[iorder[x.size()-1]],2.0)+
        pow(y[iorder[0]] -y[iorder[x.size()-1]],2.0), 0.5);
    for (int i=0;i<x.size()-1;i++)
        s+=pow(pow(x[iorder[i]] -x[iorder[i+1]],2.0)+
            pow(y[iorder[i]] -y[iorder[i+1]],2.0),0.5);
    return s;
}

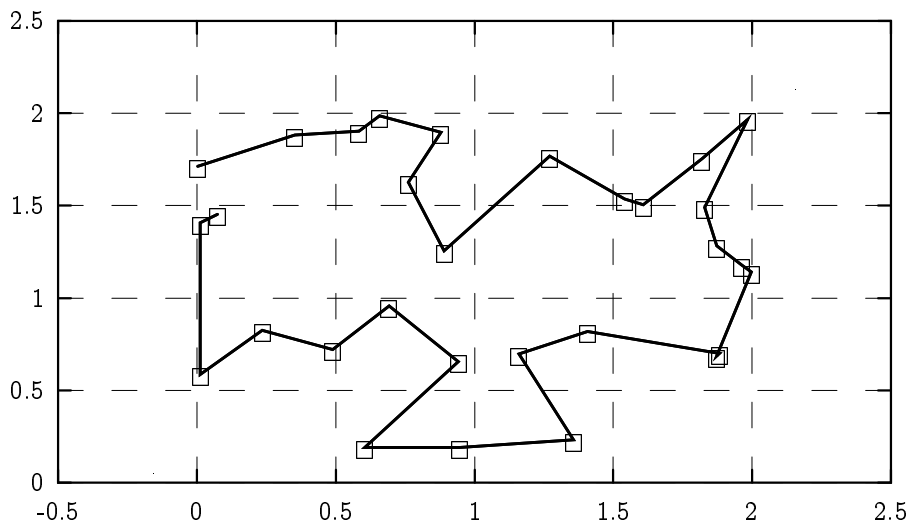
DP MinRoute(Vec_DP x, Vec_DP y, Vec_INT &iorder)
{
    cout<<"Initial length = "<<length(x,y,iorder)<<endl;
    // MyPlot(x,y,x,y,x,y);
    NR::anneal(x,y,iorder);
    cout << endl << "*** System Frozen ***" << endl;
    cout << "Final path:" << endl;
    cout << setw(5) << "city" << setw(8) << "x";
    cout << setw(11) << "y" << endl;
    cout << fixed << setprecision(4);
    Vec_DP xx(x), yy(y);
    for (int i=0;i<x.size();i++)
        {
            int ii=iorder[i];
            xx[i]=x[ii]; yy[i]=y[ii];
            cout << setw(4) << ii << setw(11) << x[ii];
            cout << setw(11) << y[ii] << endl;
        }
    DP t= length(x,y,iorder);
    cout<<"****Final length = "<<t<<endl;
    MyPlot(xx,yy,xx,yy,xx,yy);
}

```

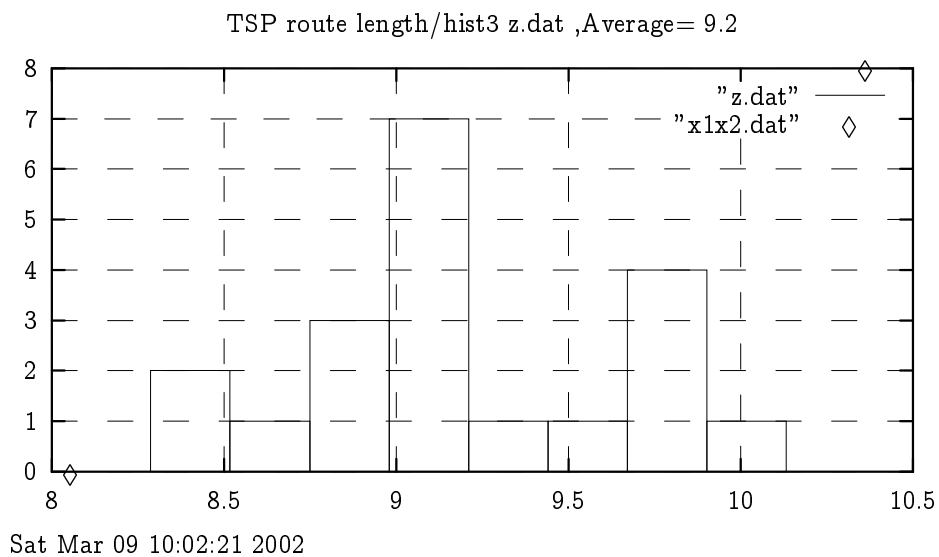
```

    return t;
}
int main(int argc, char* argv[])
{
    if (argc<3) {cout<<"Usage: ./myann2 30 15"<<endl; abort();}
    const int NCITY=atoi(argv[1]), PMAX=atoi(argv[2]);
    init_srand();
    Vec_INT iorder(NCITY);
    Vec_DP x(NCITY),y(NCITY),len(PMAX);
    for (int p=0;p<PMAX;p++)
    {
        for (int i=0;i<NCITY;i++)
        {
            x[i]= rdm(0.0,2.0);
            y[i]=rdm(0.0,2.0);
            iorder[i]=i;
            cout << setw(4) << i << setw(11) << x[i];
            cout << setw(11) << y[i] << endl;
        }
        len[p]=MinRoute(x,y,iorder);
    }
    cout<< len<<endl;
    hist(len,8,"TSP route length");
    return 0;
}
// FILE: myann2.cpp ends

```



Sat Mar 09 10:02:19 2002



Yo. kuvat on tuotettu komentoriviargumentein 30 20 ohjelmalla myann2.cpp.

```
// FILE: myamebsa.cpp begins
// Driver for routine amebssa
// g++ myamebsa.cpp -I../utils -I../democpp02 -L../lib -o a -lm -lnr

#include <cstdio>
#include <cstdlib>
#include <string>
#include <fstream>
#include <iostream>
#include <iomanip>
#include <cmath>
using namespace std;

#include "nr.h"

int idum=(-64);

DP tfunk(Vec_I_DP &p)
{
    const int N=4;
    const DP RAD=0.3,AUG=2.0;
    const DP wid_d[N]={1.0,3.0,10.0,30.0};
    int j;
    DP q,r,sumd=0.0,sumr=0.0;
    Vec_DP wid(wid_d,N);
```

```

for (j=0;j<N;j++)
{
    q=p[j]*wid[j];
    r=DP(q >= 0 ? int(q+0.5) : -int(0.5-q));
    sumr += q*q;
    sumd += (q-r)*(q-r);
}
return 1+sumr*(1+(sumd > RAD*RAD ? AUG : AUG*sumd/(RAD*RAD)));
}

```

```

int main(void)
{
    const int NP=4, MP=5;
    const DP FTOL=1.0e-6;
    const DP xoff[NP]={10.0,10.0,10.0,10.0};
    int i,iiter,iter,j,jiter,nit;
    DP temptr,yb,ybb;
    Vec_DP x(NP),y(MP),pb(NP);
    Mat_DP p(MP,NP);
    for (i=0;i<MP;i++)
        for (j=0;j<NP;j++) p[i][j]=0.0;
    cout << fixed << setprecision(6);
    for (;;)
    {
        for (j=1;j<MP;j++) p[j][j-1]=1.0;
        for (i=0;i<MP;i++)
        {
            for (j=0;j<NP;j++) x[j]=(p[i][j] += xoff[j]);
            y[i]=tfunk(x);
        }
        yb=1.0e30;
        cout << "Input t, iiter (t=0 to end):" << endl;
        cin >> temptr >> iiter;
        if (temptr <= 0.0) break;
        ybb=1.0e30;
        nit=0;
        for (jiter=0;jiter<100;jiter++)
        {
            iter=iiter;
            temptr *= 0.8;
            NR::amebsa(p,y,pb,yb,FTOL,tfunk,iter,temptr);
            nit += iiter-iter;
            if (yb < ybb)
            {
                ybb=yb;
            }
        }
    }
}

```

```

        cout << setw(6) << nit << setw(14) << temptr;
        for (j=0;j<NP;j++) cout << setw(10) << pb[j];
        cout << setw(15) << yb << endl;
    }
    if (iter > 0) break;
}
cout << endl << "Vertices of final 3-D simplex and";
cout << " function values at the vertices:" << endl;
cout << setw(3) << "i" << setw(10) << "w[i]" << setw(12) << "x[i]";
cout << setw(12) << "y[i]" << setw(12) << "z[i]";
cout << setw(14) << "function" << endl << endl;
for (i=0;i<MP;i++)
{
    cout << setw(3) << i;
    for (j=0;j<NP;j++) cout << setw(12) << p[i][j];
    cout << setw(15) << y[i] << endl;
}
cout << setw(3) << 99;
for (j=0;j<NP;j++) cout << setw(12) << pb[j];
cout << setw(15) << yb << endl << endl;
}
cout << "Normal completion" << endl;
return 0;
}
// FILE: myamebsa.cpp ends

/*

Input t, iiter (t=0 to end):
1e5
100
    101          80000    2.9378    13.9933    10.1153   -1.83195        24821.9
    202          64000    8.41456   -10.9117    7.32063    0.635657        20596.6
.....

Vertices of final 3-D simplex and function values at the vertices:
    i      w[i]          x[i]          y[i]          z[i]          function

    0 -0.00285134  0.000400207-0.000144002 -8.0757e-05        1.00002
    1  0.00126823  0.000377812-0.000397716  6.58558e-06        1.00002
    2-0.000657412  0.000688685  1.22072e-05  4.21202e-05        1.00001
    3  0.00202642  0.00046798  0.000108441-3.94758e-05        1.00001
    4  0.000182318  0.00144037-0.000298595  3.91761e-05        1.00003
    99 2.59937e-05  0.000326425-6.89748e-05  7.43448e-06        1

```

\*/

Lisää annealing-menetelmästä: P.J.M. von Laarhoven and E.H.L. Aarts: Simulated annealing: Theory and applications. Kluwer Acad.Publ.1987.

---

### Conjugate gradient algorithm

---

$\mathbf{x}_0$  = initial guess

$\mathbf{g}_0 = \nabla f(\mathbf{x}_0)$

$\mathbf{s}_0 = -\mathbf{g}_0$

for  $k = 0, 1, 2, \dots$

    Choose  $\alpha_k$  to minimize  $f(\mathbf{x}_k + \alpha_k \mathbf{s}_k)$  {perform line search }

$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{s}_k$  {update solution }

$\mathbf{g}_{k+1} = \nabla f(\mathbf{x}_{k+1})$  { compute new gradient }

$\beta_{k+1} = (\mathbf{g}_{k+1}^T \mathbf{g}_{k+1}) / (\mathbf{g}_k^T \mathbf{g}_k)$

$\mathbf{s}_{k+1} = -\mathbf{g}_{k+1} + \beta_{k+1} \mathbf{s}_k$  { modify gradient }

end

---

The formula for  $\beta_{k+1}$  in this algorithm is due to Fletcher and Reeves. An alternative formula due to Polak and Ribiere suggests the choice

$$\beta_{k+1} = ((\mathbf{g}_{k+1} - \mathbf{g}_k)^T \mathbf{g}_{k+1}) / (\mathbf{g}_k^T \mathbf{g}_k),$$

which sometimes is better.

Algm on toteutettu esimerkkihjelmassa mycgrad2.cpp.

## 11 OMINAISARVOT

$N \times N$  matriisilla  $A$  on *ominaisvektori*  $x$  ja vastaava *ominaisarvo*  $\lambda$  jos

$$(1) \quad Ax = \lambda x \text{ ja } x \neq 0.$$

$$(1) \Rightarrow (2) \quad \det |A - \lambda I| = 0.$$

Astetta  $n$  olevan polynomiyhtälön (2) juuret antavat ominaisarvot. Ko. polynomia kutsutaan *karakteristiseksi polynomiksi*. Kaikkien ominaisarvojen joukkoa kutsutaan matriisin *spektriiksi*.

Periaatteessa ominaisarvot voidaan aina laskea ratkaisemalla (2). Tässä luvussa esitellään useita huomattavasti tehokkaampia menetelmiä ominaisarvojen laskuun.

Ominaisarvon määritelmästä (1) seuraa heti, että jos  $x$  on matriisin  $A$  ominaisarvoon  $\lambda$  liittyvä ominaisvektori ja  $\tau \in \mathbb{R}$ , niin silloin  $x$  on myös matriisin  $A_\tau \equiv A + \tau I$  ominaisarvoon  $\lambda + \tau$  liittyvä ominaisvektori. Kuten myöhemmin tulemme näkemään, tällä ominaisarvojen siirtotempulla on tärkeä merkitys osana ominaisarvojen numeerista laskentaa.

**Nimityksiä:**

$$A \text{ symmetrinen} \Leftrightarrow A = A^\top \Leftrightarrow a_{ij} = a_{ji} \quad \forall i, j$$

$$A \text{ hermiittinen l. itseadjungoitu} \Leftrightarrow A = A^\dagger \Leftrightarrow a_{ij} = a_{ji}^* \quad (a_{ji}^* \text{ on } a_{ji} \text{:n kompleksikonjugaatti})$$

$$A \text{ ortogonaalinen} \Leftrightarrow A \cdot A^\top = A^\top A = 1 \Leftrightarrow A^\top = A^{-1}$$

$$A \text{ unitaarinen} \Leftrightarrow A^\dagger = A^{-1}$$

$$A \text{ normaali} \Leftrightarrow AA^\dagger = A^\dagger A$$

**Huom.** Diagonaalisen matriisin diagonaali-alkiot ovat ominaisarvoja.

**Perusidea.** Ominaisarvojen laskemiseksi  $A$  yritetään muuntaa, ominaisarvot säilyttäen, diagonaalimuotoon.

$$A \rightarrow P_1^{-1} A P_1 \rightarrow P_2^{-1} \cdot P_1^{-1} \cdot A \cdot P_1 \cdot P_2$$

Tällöin

$$\det |A - \lambda I| = \det |P_1^{-1}AP_1 - \lambda I| = \det |P_1^{-1}(A - \lambda I)P_1| = \\ \det |P_1^{-1}| \det |A - \lambda I| \det |P_1| = \det |A - \lambda I|,$$

joten yo. muunnokset säilyttävät ominaisarvot.

Yo. perusidea voidaan toteuttaa kahdella eri tavalla

- 1) alkeismuunnosten (Jacobi, Householder) iterointi
- 2) faktorisointi  $A = F_L F_R$  + iterointi (QR-hajotelma)

**Yleispiirteitä.**

- Jos halutaan vain ominaisarvot tai vain ominaisvektorit, mutta ei molempia, on tehtävä helpompi.

- Erityistä tyyppiä (reaalinen, symmetrinen) oleville matriiseille löytyy tehokkaampia algoritmeja.

Kirjallisuutta: Ks. lista Golub-van Laan (3. painos v.1989)

Coleman-van Laan

R.T. Gregory - D.L. Karney (1969), A collection of matrices for testing computational algorithms (J.Wiley)

Internetissä on Matrix Market, joka sisältää ehkäpä satoja esimerkkimatriiseja.

G. Strang: Linear algebra and its applications ( v.1989)

### 11.1 Jacobin muunnokset symmetriselle matriisille

Jacobin kierto on matriisi

$$P_{pq} = \begin{bmatrix} 1 & & & & 0 \\ & \ddots & & & \\ & c & \dots & s & \\ & \vdots & 1 & \vdots & \\ & -s & \dots & c & \\ & & & \ddots & \\ 0 & & & & 1 \end{bmatrix}$$

Diagonaali-alkiot = 1 paitsi riveillä  $p$  ja  $q$  on alkiona  $c$ .

Diagonaalin ulkopuolella alkiot = 0 paitsi (ks. kaavio)  $s$  ja  $-s$ .



$c$  ja  $s$  ovat kiertokulman  $\phi$  sini ja kosini, joten  $c^2 + s^2 = 1$ .

**Idea:** Muodostet.  $A' = P_{pq}^\top A P_{pq}$ . lasket.  $A'_{pq}$  ja valit.  $\phi$  s.e.  $A'_{pq} = 0$  (ks.NR s.464)

Toistetaan tämä eri indeksille, kunnes  $A$  on konetarkkuuden puitteissa diagonaalinen. (Huom. Kerran jo nollattu alkio voi tulla uudelleen erisuureksi kuin 0!)

Algm NR: : jacobi s.467

## 11.2 Givensin ja Householderin reduktiot

### Givens

- Tehdään Jacobin kierrot diagonaalien alapuolisiin sarakkeen alkioihin edeten sarakkeittain vasemmalta oikealle, alkioittain ylhäältä alas,
- Yhteensä noin  $n^2/2$  kiertoa

### Householder

- Peruselementtinä  $\mathcal{H}$ :n muunnos: Olkoot  $x, y \in \mathbb{R}^n \setminus \{0\}$ ,  $|x| = |y|$ . Silloin on olemassa  $(n-1)$ -ulotteinen taso  $T$ ,  $0 \in T$  s.e.  $Px = y$ , missä  $P$  on peilaus  $T$ :ssä. Selvästi  $Pz = z$  kun  $z \in T$ . ( $P$  on ortogonaalinen lineaarikuvaus)

- $\mathcal{H}$ :n algoritmi redusoi symmetrisen  $N \times N$  matriisin  $A$  tridiag. muotoon  $(N-2)$ :lla ortogonaalisella muunnoksella seuraavaan tapaan:

$$A = \begin{bmatrix} a_{11} & \dots & a_{1N} \\ a_{21} & \dots & a_{2N} \\ \vdots & \vdots & \\ a_{N1} & \dots & a_{NN} \end{bmatrix}$$

Sovelletaan em. muunnosta tälle  $x$ :lle kun  $y = \begin{bmatrix} k \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$ ,  $|k| = (\sum_{j=2}^N a_{j1}^2)^{1/2}$ .

Olkoon  $P_1$  pariin  $x, y$  liittyvä em. muunnos. Silloin

$$P_1 A = \begin{bmatrix} 1 & 0 & \dots & 0 \\ 0 & (n-1)P_1 & & \\ \vdots & & & \\ 0 & & & \end{bmatrix} \begin{bmatrix} a_{11} & \dots & a_{1N} \\ \vdots & \vdots & \\ a_{N1} & \dots & a_{NN} \end{bmatrix} = \begin{bmatrix} a_{11} & \dots & a_{1N} \\ k & \dots & \\ \vdots & \dots & \\ 0 & \dots & \end{bmatrix}$$

$$A^1 = P_1 A P_1^T = \begin{bmatrix} a_{11} & k & 0 & \dots & 0 \\ k & \dots & & & \\ 0 & \dots & & & \\ \vdots & \dots & & & \\ 0 & \dots & & & \end{bmatrix}$$

Seuraavassa vaiheessa valitaan edelliseen lauseen  $x$ :ksi  $A$ :n 2. sarakkeen  $n - 2$  alinta alkioita ja  $y = \begin{bmatrix} h \\ 0 \\ \vdots \\ 1 \end{bmatrix}$  ( $(n - 3)kpl$ ,  $|h|^2 = \sum_{j=3}^N a_{j2}^2$ ).

Tällöin  $P_2$ :lla on muoto

$$P_2 = \begin{bmatrix} 1 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & \dots & & \\ \vdots & & \dots & & \\ 0 & 0 & \dots & & \end{bmatrix}$$

Saadaan  $A^{(2)} = P_2 P_1 A P_1^T P_2^T$ . Jatketaan kunnes kaikki sarakkeet on käsitelty yo. tavalla. Prosessi tuottaa  $A$ :sta toisen matriisin  $B$ , jolla samat ominaisarvot kuin  $A$ :lla ja joka on tridiagonaalisessa muodossa.

Algm NR::tred2 s.474

### 11.3 Tridiagonaalimatriisin ominaisarvot ja -vektorit

Karakterisen polynomin juuret löydetään esim. Newtonin menetelmällä.

### QR- ja QL-algoritmit

Reaalinen matriisi  $A$  voidaan esittää muodossa  $A = QR$  missä  $Q$  on ortogonaalinen ja  $R$  yläkolmiomatriisi.  $Q$  löydetään esim. peräkkäisillä Householderin muunnoksilla. Vaihtoehtoisesti voidaan kirjoittaa  $A = QL$  missä  $Q$  on ortogonaalinen ja  $L$  alakolmiomatriisi.

Nimitys:  $QR$ -,  $QL$ -esitys.

$QL$ -algoritmi koostuu jonosta ortogonaalimuunnoksia:  $A_s = Q_s L_s$ ,  $A_{s+1} = L_s Q_s (= Q_s^T A_s Q_s)$ .

**Lause.** 1) Jos  $A$ :lla on ominaisarvot  $\lambda_i$  s.e.  $|\lambda_i| \neq |\lambda_j|$  kaikilla  $i \neq j$  niin  $A_s \rightarrow$  alakolmio kun  $s \rightarrow \infty$ . ominaisarvot diagonaalilla.

2) Jos  $A$ :lla on  $p$  kpl ominaisarvoja  $\lambda_i$ , joiden itseisarvot samoja, niin  $A_s \rightarrow$  alakolmio kun  $s \rightarrow \infty$  paitsi diagon. olevaa  $p \times p$  blokkia, jonka ominaisarvot  $\rightarrow \lambda_i$ .

Todistus sivuutetaan. Ks. [BS] Sivutuote todistuksesta on että  $a_{ij}^{(s)} \sim (\frac{\lambda_i}{\lambda_j})^s$ , huomaa  $j > i \Rightarrow \frac{|\lambda_i|}{|\lambda_j|} \leq 1$ . Konvergenssi kohti 0:aa voi olla hidasta, jos  $\lambda_i \sim \lambda_j$ .

Konvergenssia voidaan kiihdyttää siirtotekniikalla: Jos  $A$ :n ominaisarvot ovat  $\lambda_i$ , niin  $A - kI$ :n ominaisarvot ovat  $\lambda_i - k$ . Kirjoittamalla

$$A_s - k_s I = Q_s L_s \Leftrightarrow A_{s+1} = L_s Q_s + k_s I = Q_s^T A_s Q_s$$

huomataan, että suppenemisvauhdin määräävä ominaisarvojen suhde on nyt

$$\frac{\lambda_i - k_s}{\lambda_j - k_s}.$$

Toimintastrategiana on valita  $k_s$  kullakin  $s$  siten että suppenemisvauhti on maksimaalinen.

Vielä tehokkaampi on ns. implisiittinen siirto, jota NR suosittaa.

Algm NR::tqli s. 480 toteuttaa reaalisen *symmetrisen* tridiagonaalimatriisin QL-hajoitelman implisittisin siirtein. Se soveltuu myös NR::tred2:n tuottaman symmetrisen (ehkä ei tridiagonaalisen) matriisin ominaisarvojen laskuun.

## 11.5 Yleisen matriisin reduktio Hessenbergin muotoon

Reaalisen symmetrisen matriisin ominaisarvot löydetään `alm:n NR::tqli` ja edeltävän `NR::tred2:n` avulla varsin tehokkaasti. Epäsymmetrisen matriisin tapaus on osoittautunut vaikeammaksi. `NR:n` resepti epäsymm. tapaukseen on kolmivaiheinen a) tasapainotus (balancing) b) reduktio Hessenbergin muotoon ts. ( $a_{ij} = 0 \forall i \geq j + 2$ ) c) QR-algoritmi.

a) **Tasapainotus:** Muokataan (pyöristysvirheiden vaikutuksen pienentämiseksi) reaalinen matriisi  $A$  ortogonaalisilla muunnoksilla  $Q$  muotoon  $Q \cdot A \cdot Q^T$  s.e. ominaisarvot säilyvät mutta tasapainotetaan rivi- ja sarakesummia niin, että ne tulevat likimain samoiksi. `Algm NR::balanc` s. 483.

b) **Reduktio Hessenbergin muotoon:** Toteutetaan Gaussin eliminoinnin kaltaisella menetelmällä. `Algm NR::elmhes` s.485.

c) **QR-algoritmi.** Vaiheiden a) & (b) jälkeen jatkokäsittelynä on QR-menetelmä. `Algm NR::hqr` s.491.

```

/* FILE: mych11.cpp begins */
/* gcc mych11.cpp -L../lib -I../util -o a -lm -lnr      */
/* g++ -Wall beg.cpp -L../lib -I../utils -I../gnuplot02 -o a -lm -lnr */

#include <cstdlib> // Used in putmat2
#include <cstdio> // Used in putmat2
#include <ctime>
#include <string>
#include <fstream>
#include <iostream>
#include <iomanip>
using namespace std;
#include "nr.h"
#include "matut102.h"
#define N 5
int main()
{
    int i, j;
    Mat_DP rand_mat(N,N), temp_mat(N,N), prod_mat(N,N), diff_mat(N,N);
    Vec_DP d(N), e(N);
    init_srand();
    /* Create random symmetric matrix */
    ranmat2(rand_mat,-10.0, 10.0);
    transp(rand_mat,temp_mat);

```

```

prod_mat=temp_mat+rand_mat;
rand_mat=prod_mat; // Now symmetric
temp_mat=rand_mat;
/* Householder reduction */
NR::tred2(temp_mat, d, e);
/* QL algorithm with implicit shifts */
NR::tqli(d, e, temp_mat);
printf("\n\nSymmetric random matrix:\n");
showmat2(rand_mat, " %10.6lf");
printf("\n\nEigenvalues:\n\n ");
for (i = 0; i <= N-1; i++)
    printf("%-12.5lf ",d[i]);
matmul(rand_mat, temp_mat, prod_mat );
for (i = 0; i <= N-1; i++)
    for (j = 0; j <= N-1; j++)
        temp_mat[j][i] *= d[i];
for (i = 0; i <= N-1; i++)
    for (j = 0; j <= N-1; j++)
        diff_mat[i][j] = prod_mat[i][j] - temp_mat[i][j];
printf("\nnorm(A*x-lambda*x) = %12.4e\n",mnormp(diff_mat,2.0));
printf("\n\nThe matrix A*x-lambda*x (should be 0):\n");
showmat2(diff_mat, " %12.4e");
return 0;
}

/* FILE: mych11.cpp ends */
/*
Symmetric random matrix:
5x5 matrix:
  1.237173   1.691847 -11.165518  -7.440216   9.466231
  1.691847 -18.371782  12.643289  10.716048  -0.357420
 -11.165518  12.643289 -14.454560   6.337935  -0.327107
  -7.440216  10.716048   6.337935  16.455957  -1.977455
   9.466231  -0.357420  -0.327107  -1.977455   5.702360
Eigenvalues:
 -3.49092   -11.15502   10.15300   -32.32196   27.38405
norm(A*x-lambda*x) =  2.7593e-14
The matrix A*x-lambda*x (should be 0):
5x5 matrix:
  4.4409e-16  -8.8818e-16  -8.8818e-16   7.1054e-15   1.7764e-15
  4.3854e-15   1.7764e-15   2.8866e-15  -1.0658e-14  -2.6645e-15
  2.4425e-15   0.0000e+00  -5.1001e-16  -1.0658e-14   0.0000e+00
  7.3275e-15   3.5527e-15   6.2172e-15  -1.3323e-14  -7.1054e-15
 -3.5527e-15  -3.5527e-15   3.5527e-15   7.9936e-15   0.0000e+00
*/

```

Seuraava demo-ohjelma antaa lisää esimerkkejä ominaisarvojen laskemisesta.

```
/* FILE myeigen2.cpp begins */
#include <iostream>
#include <iomanip>
#include "nr.h"

#include "matut102.h"
using namespace std;

void seigen(const Mat_DP &a, Mat_DP &eigvec, Vec_DP &eigval)
// a must be symmetric real square matrix
// column j of the matrix eigvec is the jth eigenvector
{
    int NP=a.nrows(), nrot;
    Vec_DP d(NP);
    Mat_DP v(NP,NP), e(NP,NP);
    /* Check that a is symmetric */
    DP s=0.0;
    for (int i=0; i<NP;i++)
        for (int j=0; j<i-1;j++) s+=abs(a[i][j]-a[j][i]);
    if ((s>1.0e-15)|| (a.ncols()!=NP))
        {cout<<"Argument error in seigen"<<endl; abort();}
    e=a;
    NR::jacobi(e,d,v,nrot);
    NR::eigsrt(d,v);
    eigval=d;
    eigvec=v;
}

void eigen(Mat_DP &a, Vec_CPLX_DP &w)
{
    NR::balanc(a);
    NR::elmhes(a);
    NR::hqr(a,w);
}

int main()
{
    const int NP=5;
    Mat_DP e(NP,NP), eT(NP,NP), eigvec(NP,NP);
    Vec_CPLX_DP wri(NP);
    Vec_DP eigval(NP);
```

```

init_srand();
ranmat2(e, -2.0,2.0);
eigen(e,wri);
cout << "eigenvalues:" << endl;
cout << setw(11) << "real" << setw(12) << "imag."
    << setw(11) << " abs " << endl;
for (int i=0;i<wri.size();i++)
    cout << setw(25) << wri[i] << setw(12)
        <<abs(wri[i])<< endl;
ranmat2(e, -2.0,2.0);
transp(e,eT);
e=e+eT; // This is symmetric
seigen(e, eigvec,eigval);
cout<<"Symmetric matrix:\n " <<endl;
showmat2(e, "% 11.6f");
cout<<"Eigenvalues:\n" << eigval << endl;
return 0;
}
/* FILE myeigen2.cpp ends */

```

```

eigenvalues:
    real      imag.      abs
(0.0630519,-1.43255)  1.43394
(0.0630519,1.43255)  1.43394
(-2.65948,0)        2.65948

Symmetric matrix:
3x3 matrix:
 1.376873  0.710549 -2.418001
 0.710549  2.953847 -0.358019
-2.418001 -0.358019  3.740742

Eigenvalues:
 5.4386207 2.8212785 -0.18843683

Eigenvector j is column j of:
-0.52701874 -0.028247942  0.84938407
-0.26696004  0.95435978  -0.13390202
 0.80683554  0.29732048  0.51050656

```

## 11.6 Hermiittinen matriisi

Em. algm konvertointi kompleksimatriiseille ei ole triviaalia. Tästä syystä on paikallaan todeta, että hermiittiselle  $n \times n$  matriisille  $A = A_r + iA_i$ , missä  $A_r, A_i$ , ovat  $n \times n$  reaalmatriiseja, ominaisarvojen lasku voidaan palauttaa  $2n \times 2n$  reaalmatriisiin  $B = [A_r - A_i; A_i A_r]$

ominaisarvojen laskuun. Jos  $B$ :llä on ominaisvektori  $z$  niin merk.  
 $z = (q_r, q_i)$  missä sekä  $q_r$  että  $q_i$  ovat  $n$ -komponenttisia reaalivekto-  
reita. Silloin myös  $z = (q_i, q_r)$  on samaan ominaisarvoon liittyä  $B$ :n  
ominaisvektori ja  $z = q_r + i * q_i$  on samaan ominaisarvoon liittyä  $A$ :n  
ominaisvektori. Huomaa, että  $B$  on symmetrinen, joten sen ominai-  
sarvot ovat reaalisia. Jos  $\lambda_1, \dots, \lambda_n$  ovat  $A$ :n ominaisarvot, niin  $B$ :n  
ominaisarvot ovat  $\lambda_1, \lambda_1, \dots, \lambda_n, \lambda_n$ .



## 12 FOURIER-MUUNNOS

### 12.0. Johdanto

-Fourier-menetelmät soveltuvat datan käsittelyyn ja analyysiin (suuret datamäärät) esim. signaalianalyysissä ja sähkötekniikassa

-nopealla Fourier-muunnoksella on lukuisia sovelluksia esim. osittaisdiff. yhtälöiden numeriikkaan

Aloitamme lyhyellä Fourier-muunnosten ominaisuuksien luettelolla (jatkuva tapaus, tässä luvussa jatkossa lähinnä diskreetti tapaus).

$$H(f) = \int_{-\infty}^{\infty} h(t)e^{2\pi ift} dt \quad t = \text{"time"}$$

$$(1) \quad h(t) = \int_{-\infty}^{\infty} H(f)e^{-2\pi ift} df \quad f = \text{"frekvenssi"}$$

Merkintä:  $h(t) \Leftrightarrow H(f)$  (Muunnosparit)

Jos...	niin...
$h(t)$ reaal.	$H(-f) = H(f)^* \leftarrow$
$h(t)$ imag.	$H(-f) = -H(f)^*$
$h(t)$ parillinen ( $h(t) = h(-t)$ )	$H(-f) = H(f)$
$h(t)$ pariton ( $h(t) = -h(-t)$ )	$H(-f) = -H(f)$
$h(t)$ reaalinen ja parillinen	$H(f)$ reaal. ja parillinen
$h(t)$ – –" – –	$H(f)$ imag. ja pariton
$h(t)$ imag. ja parillinen	$H(f)$ imag. ja parillinen
$h(t)$ imag. ja pariton	$H(f)$ reaal. ja pariton

$$h(at) \Leftrightarrow \frac{1}{|a|} H\left(\frac{f}{a}\right)$$

$$\frac{1}{|b|} h\left(\frac{t}{b}\right) \Leftrightarrow H(bf)$$

$$h(t - t_0) \Leftrightarrow H(f)e^{2\pi i f t_0}$$

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

$(g * h)(t)$  on  $g$ :n ja  $h$ :n konvoluutio,  $g * h = h * g$ . Tavallisesti  $h, g$  reaalisia.

**Konvoluutiolause:**  $g * h \Leftrightarrow G(f)H(f)$ .

$Corr(g, h) = \int_{-\infty}^{\infty} g(\tau + t)h(\tau)d\tau$  ( $g$ :n ja  $h$ :n korrelaatio)

**Korrelaatiolause:**  $Corr(g, h) \Leftrightarrow G(f)H^*(f)$ .

**Wiener-Khinchinin lause:**  $Corr(g, g) \Leftrightarrow |G(f)|^2$ .

**Parsevalin lause:**  $\int_{-\infty}^{\infty} |H(f)|^2 df = \int_{-\infty}^{\infty} |h(t)|^2 dt$ .

Historiallisia huomautuksia: J. Fourier esitti v.1807 (21-vuotiaana), että mieliv. jatkuva ftio voidaan esittää sinien ja kosinien summana. Hänen ajatuksensa herättivät vastustusta mm. koska todistukset puuttuivat. F. sovelsi ideoitaan mm. lämmönjohtodiff. yhtälön ratkaisuun kellarien rakentamisessa.

## 12.1 Diskreetti Fourier-muunnos

Funktion  $h$  arvoja on mitattu tasavälein  $h_n = h(n\Delta)$ ,  $\Delta > 0$ ,  $n = 0, 1, 2, \dots, N - 1$  ( $N$  parillinen). Oletetaan, että  $h(t) = 0$  välin  $(0, \Delta(N - 1))$  ulkopuolella. Halutaan löytää approksimaatio  $H(f_n)$ :lle,  $f_n = \frac{n}{N\Delta}$ ,  $n = -\frac{N}{2}, \dots, \frac{N}{2}$ . Sijoitus 12.0 (1):een antaa

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

$\sum_0^{N-1}$  on nimeltään pisteiden  $h_k$ ,  $n = 0, 1, 2, \dots, N - 1$ , *diskreetti Fourier-muunnos*, ts.

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

Diskreetille Fourier-muunnokselle pätevät monen jatkuvan Fourier-muunnoksen ominaisuudet kuten NR:n demo-ohjelmista ilmenee. Käänteismuunnos on

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

## 12.2 Nopea Fourier-muunnos (FFT)

Karkea arvio 12.1 (2):n vaatimalle laskutyölle saadaan seuraavasti. Merk.  $W = e^{2\pi i/N}$ , jolloin 12.1 (2) saa muodon

$$(1) \quad H_n = \sum_{k=0}^{N-1} W^{nk} h_k, \quad n = 0, 1, \dots, N-1.$$

Laskutoimitusten lukum.  $H_0, \dots, H_{N-1}$ :n laskemiseksi  $\approx N^2$ . (Matriisi  $W \times [h_0, \dots, h_{N-1}]^T$ ).

Nopea Fourier-muunnos tekee saman  $O(N \log_2 N)$  operaatiolla, Gauss 1800-luku, Danielsson-Lanczos 1940-luku, Cooley-Tukey 1960-luku. Perusidea ilmenee seuraavasta ( $W = e^{2\pi i/N}$ ):

$$F_k = \sum_{j=0}^{N-1} e^{2\pi ijk/N} f_j = \sum_{j=0}^{N/2-1} e^{2\pi ik(2j)/N} f_{2j} + \sum_{j=0}^{N/2-1} e^{2\pi ik(2j+1)/N} f_{2j+1} =$$

$$\sum_{j=0}^{N/2-1} e^{2\pi ikj/(N/2)} f_{2j} + W^k \sum_{j=0}^{N/2-1} e^{2\pi ikj/(N/2)} f_{2j+1} = F_k^e + W^k F_k^o$$

$F_k^e$  = parilliset komponentit  $f_j$ :stä.

$F_k^o$  = parittomat komponentit  $f_j$ :stä.

Siis  $F_k$ :n lasku palautui  $F_k^e$ :n ja  $F_k^o$ :n laskuksi.

Sovelletaan samaa rekursiivisesti yhteensä  $\log_2 N$  kertaa  $\Rightarrow$  laskutyö  $N \times \log_2 N$ . Divide et impera!

Algm NR:: four1 s. 507.

## 12.3 Reaalifunktioiden FFT

FFT soveltuu kompleksilukudatoille  $h_k$ , siis myös reaaliluvuille. Miten reaalilukujen tapauksessa voitaisiin toimia tehokkaammin?

**Ideota:** a) Pakataan syötteeseen kaksi reaalifunktiota siten, että niiden Fourier-muunnokset voidaan erottaa toisistaan, jolloin saadaan yhdellä kertaa kahden funktion FFT.

b) Pakataan kompl. syötevektori ilman turhia nollia puolta lyhympään kaavioon ja suoritetaan FFT tähän lyhyempään vektoriin.

a) Algm NR:: twofft.  $f$  reaalinen  $\Rightarrow F_{N-n} = (F_n)^*$

$f$  imaginaarinen  $\Rightarrow G_{N-n} = -(G_n)^*$

b) Annettu reaalinen  $f_j$ . Muodostetaan kompleksiluvut  $h_j = f_{2j} + i f_{2j+1}$ ,  $j = 0, 1, \dots, N/2 - 1$ . Syötetään tämä kompleksidata NR:: four1:lle, jolloin saadaan  $H_n = F_n^e + i F_n^o$ ,  $n = 0, \dots, N/2 - 1$ . Tästä voidaan  $F_n$ :t laskea (NR s.512 (12.3.5)).

Algm NR:: realft s.513

### Nopea sini/kosini muunnos

Datan  $f_j$ ,  $j = 0, \dots, N-1$  *sini- ja kosinimuunnokset* määritellään kaavoin

$$(1) \quad F_k = \sum_{j=1}^{N-1} f_j \sin(\pi j k / N), \quad F_k = \sum_{j=0}^{N-1} f_j \cos(\pi j k / N).$$

Miten nämä lasketaan FFT:n avulla?

**Idea:** Tehdään uusi funktio, jonka FFT on sinimuunnos.

Määritellään uusi laajennettu funktio

$$(2) \quad f_{2N-j} \equiv -f_j, \quad j = 0, 1, \dots, N-1.$$

Laajennetun ftion FFT

$$(3) \quad F_k = \sum_{j=0}^{2N-1} f_j e^{2\pi i j k / (2N)}.$$

Osasumma  $j = N \rightarrow 2N - 1$  voidaan muokata seuraavasti

$$(4) \quad \sum_{j=N}^{2N-1} f_j e^{2\pi i j k / (2N)} = \sum_{j=1}^N f_{2N-j} e^{2\pi i (2N-j) k / (2N)} = - \sum_{j=0}^{N-1} f_j e^{-2\pi i j k / (2N)}$$

josta

$$(5) \quad F_k = \sum_{j=1}^{N-1} f_j [e^{2\pi i j k / (2N)} - e^{-2\pi i j k / (2N)}] = 2i \sum_{j=0}^{N-1} f_j \sin(\pi j k / N)$$

Näin on sinimuunnoksen lasku palautettu FFT:hen.

Yo. temppu on sama, jolla annetulle ftiolle löydetään pelkästään sin-termejä sisältävä Fourier-sarja. Vastaavasti voidaan myös kosi-nimuunnos palauttaa FFT:hen.

NR:n algoritmit NR:: snft ja NR:: cosft tekevät yo. muunnokset kuitenkin tehokkaammin kuin yllä ks. NR s. 516-518.

Muita FFT:hen palautuvia. Konvoluutio, korrelaatio.

FFT:n eräs sovellus on signaalin pelkistäminen kohinaisesta datasta. Tähän liittyvä demo-ohjelma myfrexa.cpp on www-sivulla.

## 13 DATAN TILASTOLLINEN ANALYYSI

-keskiarvo, moodi, mediaani

-jakaumien samuuden testaus, korrelaatio

-datan silyys

### 13.1. Jakauman momentit

Keskiarvo  $\bar{x} = \frac{1}{N} \sum_{j=1}^N x_j$

Varianssi  $Var(x_1, \dots, x_N) = \frac{1}{N-1} \sum_{j=1}^N (x_j - \bar{x})^2$ ;  $\sigma(x_1, \dots, x_N) = \sqrt{Var(x_1, \dots, x_N)}$

ADev( $x_1, \dots, x_N$ ) =  $\frac{1}{N} \sum_{j=1}^N |x_j - \bar{x}|$

Skew( $x_1, \dots, x_N$ ) =  $\frac{1}{N} \sum_{j=1}^N \left(\frac{x_j - \bar{x}}{\sigma}\right)^3$

Kurt( $x_1, \dots, x_N$ ) =  $-3 + \frac{1}{N} \sum_{j=1}^N \left(\frac{x_j - \bar{x}}{\sigma}\right)^4$

**Huom.**  $Var(x_1, \dots, x_N) = \frac{1}{N-1} [\sum_{j=1}^N x_j^2 - N\bar{x}^2]$ , mutta tämä kaava ei pyöristysvirheiden vuoksi ole yhtä hyvä kuin yo. määritelmä (ks. Knuth II s.216)

Algm NR: :moment s.613

### 13.2 Mediaani

**Määr.**

$$\int_{-\infty}^{x_{med}} p(x) dx = \frac{1}{2} = \int_{x_{med}}^{\infty} p(x) dx$$

Jos havainnot  $x_1, \dots, x_N$ , niin  $x_{med}$  on näistä keskimäinen.

### 13.3 Varianssit, keskiarvot

-Studentin  $t$ -testi kahden eri datan keskiarvojen eroavuuden mittaamiseen (kun varianssi ajatellaan samaksi). Algm NR: :ttest

- $t$ -testi keskiarvojen eroavuuden mittaamiseen kun varianssit eivät samoja. Algm NR: :tutest, NR: :tptest, NR: :ftest

### 13.4 Ovatko jakaumat samat?

- ryhmitetty data -  $\chi^2$  testi
- jatkuva data - Kolmogorov-Smirnov

#### $\chi^2$ -testi

- $N_i$  havaintojen määrä  $i$ :nnessä ryhmässä,  $N_i \in N$ ,  $n_i$  odotettu lukumäärä  $i$ :nnessä ryhmässä (esim. tunnetun jakauman antamana),  $n_i \in N$

$$\chi^2 = \sum_i \frac{(N_i - n_i)^2}{n_i}$$

-jos  $\chi^2$  on suuri, on hypoteesi  $H_0$ : "havainnot jakautuneet em. tunnetun jakauman mukaan" väärä

-vapausasteiden lukumäärä = ryhmien lukumäärä

Algm NR: :chsone s. 621

Yllä verrattiin havaintoja tunnettuun jakaumaan.

Jos halutaan tutkia kahta eri dataa ja selvittää ovatko kummankin jakaumat keskenään samat, void. myös käyttää  $\chi^2$ -testiä. Algm NR: :chstwo s.622.

#### Kolmogorov-Smirnov

Algm NR: :ksone, kstwo

### 13.5 Datan silitys

Jos data on sekavaa tai vaihtelut suuria, voi datan yleisten muodon tai "tendenssin" näkeminen olla vaikeaa. Näissä tilanteissa saattaa olla eduksi kokeilla erilaisia visualisointitekniikoita, joita ovat mm. erilaisia data approksimoivien käyrien piirtäminen (esim. PNS-käyrät).

## 14 DATAN MALLINTAMINEN

**Perustehtävä:** Annettu pisteparit  $(x_i, y_i)$ ,  $i = 1, \dots, N$ , jossa  $y_i$ :n ajatellaan olevan muotoa  $f(x_i, a_1, \dots, a_M)$ . Miten parametrit  $a_i$  on valittava, jotta  $y_i$ :t sopivat mahdollisimman hyvin malliin  $f(x_i, a_1, \dots, a_M)$ ?

Mittausvirheiden vaikutus voidaan myös huomioida.

Malli on *lineaarinen*, jos se riippuu lineaarisesti parametreista  $a_i$ , muuten *epälineaarinen*.

**Esim.** Lineaarinen malli  $f(x, a_1, \dots, a_M) = \sum_{i=1}^M a_i x^i$ .

Epälineaarinen malli  $f(x, a_1, a_2) = a_1 \exp(-a_2 x) + \exp(-x)$ .

Sovituksen hyvyyskriteerinä on tavallisesti neliösumma

$$S = \sum_{i=1}^N (y_i - f(x_i, a_1, \dots, a_M))^2$$

tai sen jokin variantti (Gaussin pns-menetelmä).

### 14.1 Normaaliyhtälöt

Olkoon annettu pisteparit  $(x_i, y_i)$ ,  $i = 1, \dots, N$ . Tällöin ajatellaan tavallisesti, että  $x_i$  on lukuarvona tarkka, kun taas  $y_i$  on ”mittattu” suure, joka voi olla epätarkka esim. mittausjärjetelyistä johtuen. Olkoon  $\sigma_i$   $y_i$ :n stand. poikkeama. Mallin  $y(x_i, a_1, \dots, a_M)$  sovittamiseksi mittauspisteistöön tarkastellaan suuretta

$$S(a) = \sum_{i=1}^N \left( \frac{y_i - y(x_i, a_1, \dots, a_M)}{\sigma_i} \right)^2,$$

joka halutaan minimoida. Tämä suure noudattaa (perustelu:tn. laskenta)  $\chi^2$ -jakaumaa vap.astein  $N - M$ . Välttämätön ehto suureen  $S$  minimoinnille on  $\frac{\partial S}{\partial a_i} = 0$ ,  $i = 1, \dots, M$  eli *normaaliyhtälöt*

$$(1) \quad \begin{cases} \sum_{i=1}^N \frac{y_i - y(x_i, a_1, \dots, a_M)}{\sigma_i^2} \frac{\partial y(x_i, a_1, \dots, a_M)}{\partial a_k} = 0 \\ k = 1, \dots, M \end{cases}$$



## 14.2. PNS-suora

Nyt mallina on  $y = a + bx$  missä  $a$  ja  $b$  halutaan estimoida datasta  $(x_i, y_i)$ ,  $i = 1, \dots, N$ . Normaalilyhtälöt 14.1 (1) antavat nyt helposti  $a$ :n ja  $b$ :n (ks. NR s.662).

Algm NR::fit s.665

## 14.3 Yleinen lin. PNS

Mallina on  $y = \sum_{k=1}^M a_k X_k(x)$  missä  $X$ :t ovat mieliv. funktioita (ei tarvitse olla lin.). Normaalilyhtälöt 14.1 (1) antavat nyt vaatimuksen

$$0 = \sum_{i=1}^N \frac{1}{\sigma_i^2} [y_i - \sum_{j=1}^M a_j X_j(x_i)] X_k(x_i), \quad \forall k = 1, \dots, M$$

tai lyhyesti  $\sum_{j=1}^M \alpha_{kj} a_j = \beta_k$ , missä  $\alpha_{kj} = \sum_{i=1}^N \frac{X_j(x_i) X_k(x_i)}{\sigma_i^2}$ ,  $\beta_j = \sum_{i=1}^N \frac{y_i X_k(x_i)}{\sigma_i^2}$  tai

$$[\alpha] = A^\top \cdot A, [\beta] = A^\top \cdot \beta, A_{ij} = X_j(x_i)/\sigma_i$$

Sama matriisimuodossa  $[\alpha]a = [\beta]$  eli  $(A^\top \cdot A)a = A^\top b$

$$a_j = \sum_{k=1}^n [\alpha]_{jk}^{-1} \beta_k = \sum_{k=1}^n C_{jk} \left[ \sum_{i=1}^N \frac{y_i X_k(x_i)}{\sigma_i^2} \right]$$

$[\alpha]^{-1}$  löydet. LU-hajoittelman avulla.

Algm NR::lfrit s.674

**Huom.** Algm NR::lfrit sovittaa dataan  $(x[i], y[i])$  annettujen kantafunktioiden  $X_j(x)$ ,  $j = 1, \dots, MA$ , lineaarikombinaation  $u = \sum A_j X_j(x)$  (lin. malli), jossa on MA kpl kertoimia  $A_j$ . Näistä osa voidaan pitää kiinteinä. Varioivat parametrit annetaan taulukon IA avulla, jos  $IA[j]=0$ , niin  $A_j$  on kiinteä, muulloin se varioi. (NR, s.674)

## SVD pns-sovituksessa

Merk.  $A = [X_j(x_i)/\sigma_i]$  kuten edellä. Pns-sovituksen perustehtävä on seuraava:

Etsi  $a$ , joka minimoi suureen  $\chi^2 = [A \cdot a - b]^2$ .

Olk.  $A$ :lla SVD ( $A$  on  $N \times M$ ,  $W$  on  $M \times M$  ja  $V$  on  $N \times N$  matr.)

$$A = UWV^T$$

Olk.  $U_{(i)}$ ,  $V_{(i)}$ ,  $i = 1, \dots, M$ ,  $U$  ja  $V$ :n sarakeet.

Palautetaan mieleen (NR s.65) yhtälön  $A \cdot a = b$  SVD-ratk.

$$a = V \cdot \text{diag}(1/w_j) \cdot U^T(b).$$

Tämä voidaan ilmaista  $A$ :n pseudoinverssin  $A^+$  avulla myös muodossa  $a = A^+b$ . Yo. merkinnöin

$$a = \sum_{i=1}^n \left( \frac{U_{(i)} \cdot b}{w_i} \right) V_{(i)} \pm \frac{1}{w_1} V_{(1)} \pm \dots \pm \frac{1}{w_n} V_{(n)}.$$

Singulaariarvojen editointimenettelyä, jota jo luvussa 2 tarkasteltiin, voidaan käyttää myös tässä yhteydessä. Jos  $w_i = 0$  niin aset.  $1/w_i = 0$  (vrt. NR s.62, 677). Samoin tehdään jos  $\text{abs}(w_i) < N \times \text{kone-eps}$ .

Algm NR::svdfit s.678.

**Huom.** Luvussa 3 esitelty algoritmi NR::mypolfit käyttää osanaan algoritmia NR::svdfit.

**Huom.** Joskus epälineaarinen malli voidaan linearisoida ts. muuntaa lineaariseksi. Esim. malli  $y = x^a$ ,  $a$  estimoitava parametri, muuntuu muotoon  $z = \log y = a \log x$  joka on lineaarinen.

Yleisemmin, jos  $f$  on homeomorfismi niin malli  $y = f(ax)$ ,  $a$  parametri, void. linearisoida ottamalla  $F^{-1}: z = f^{-1}(y) = ax$ . Jos epälin. malli on laskennallisesti vaikea, voi yrittää mallin korvaamista yksinkertaisemmalla (vrt. kohdefunktion kvadraattinen approksimaatio minimin läheisyydessä optimointialgoritmeissa). Ma-

temaattisesti tämä merkitsee funktion korvaamista sopivalla esim. Taylor-approksimaatiolla.

#### 14.4 Marquardt'n menetelmä

Dataan  $(x_i, y_i)$ ,  $i = 1, \dots, N$  sovit. malli  $y(x_i, a)$ , joka on epälin. param. vektorin  $a = (a_1, \dots, a_n)$  suhteen. Toiminta-ajatus on, kuten tavallista, kohdefunktion

$$\chi^2(a) = \sum_{i=1}^N (y_i - y(x_i, a))^2 / \sigma_i^2$$

minimointi. Koska malli on nyt epälineaarinen, normaaliyhtälöitä 14.1 (1) ei voida suoraan ratkaista. Näin ollen käytetään jotakin sopivaa iteratiivista miniminhakualgoritmia. Periaatteessa menettellään samoin kuin optimointiteoriassa, mutta pyrkien käyttämään hyväksi kohdefunktion ylläolevaan muotoon sisältyvää informaatiota (Hessen matriisi).

Marquardt'n (-Levenbergin) menetelmässä ideana on vaihdella joustavasti tilanteen mukaan kahden minimointimenetelmän, jyrkimmän vähenemisen ja Newton-tyyppisen menetelmän välillä.

Merkitään

$$\beta_k = -\frac{1}{2} \frac{\partial \chi^2}{\partial a_k} = \sum_{i=1}^N \frac{\partial y(x_i, a)}{\partial a_k} [y_i - y(x_i, a)] / \sigma_i^2$$

$$\alpha_{kl} = \frac{1}{2} \frac{\partial^2 \chi^2}{\partial a_k \partial a_l} = \sum_{i=1}^N \frac{1}{\sigma_i^2} \left[ \frac{\partial y(x_i, a)}{\partial a_k} \frac{\partial y(x_i, a)}{\partial a_l} - (y_i - y(x_i, a)) \frac{\partial^2 y(x_i, a)}{\partial a_l \partial a_k} \right]$$

Newton-tyyppinen menetelmä: Annetaan parametrille  $a$  korjaus  $\delta a = (\delta a_1, \dots, \delta a_M)$ , joka määräytyy yhtälöstä

$$\sum_{l=1}^M \alpha_{kl} \delta a_l = \beta_k.$$

Jyrkimmän vähenemisen menetelmä: Annetaan  $a$ :lle korjaus

$$\delta a_l = \text{vakio} \times \beta_k.$$

Jatkoa varten muutetaan  $\alpha_{kl}$ :n määritelmää niin, että jätetään pois 2. kertaluvun osittaisderivaatat (stabiliteetin parannusyritys) ts. asetetaan

$$\alpha_{kl} = \sum_{i=1}^N \frac{1}{\sigma_i^2} \frac{\partial y(x_i, a)}{\partial a_k} \frac{\partial y(x_i, a)}{\partial a_l}.$$

Määritellään myös

$$\alpha'_{jj} = \alpha_{jj}(1 + \lambda), \alpha'_{jk} = \alpha_{jk}, j \neq k.$$

### Marquardt'n algoritmi (1963)

1. Annetaan alkuarvaus  $a$ :lle, valitaan  $\lambda = 0.001$
2. Ratkaistaan  $\delta a$ :n suhteen  $\sum_{i=1}^n \alpha'_{kl} \delta a_l = \beta_k$
3. Jos  $\chi^2(a + \delta a) \geq \chi^2(a)$ , niin  $\lambda \leftarrow 10 * \lambda$  ja palataan 2.een
4. Jos  $\chi^2(a + \delta a) < \chi^2(a)$ , niin  $\lambda \leftarrow \lambda/10$ ,  $a \leftarrow a + \delta a$  ja palataan 3:een mikäli lopetusehto ei toteudu.

**Huom.** Tapauksessa  $\lambda \sim 0$  Marquardt'n menetelmä toimii lähes samoin kuin Newtonin menetelmä, ja tapauksessa  $\lambda$  suuri kuten steepest descent.

Algm NR: :mrqmin s.685

**Huom.** Tässä algoritmossa on mahdollista, kuten NR::lfit:ssä kiinnittää eräitä parametrejä ja varioida haluttuja.

```

/* FILE: ch14ex1.c begins */
/*
gcc ch14ex1.c -L../lib -I../util -o a -lm -lnr
*/

#include <stdio.h>
#include <math.h>
#include <stdlib.h>

#define NRANSI

#include "../util/nr.h"
#include "../util/nrutil.h"
#include "../util/matutl6.c"

```

```

#include "../gnuplot/gnuplt2t.c"

#define NP 100
#define P1 5

/* compute p:th power of x for lfit */
void powers(float x, float pows[], int p)
{
    int i;
    float xp = 1.0;
    for (i = 1; i <= p; i++)
    {
        xp *= x;
        pows[i] = xp;
    }
}

/* compute value and derivatives (w.r.t. coefficients) of function
 * lambda_1*exp(-x) + lambda_2*exp(-lambda_3*x)
 * for mrqmin.
 */
void exps(float x, float *coefs, float *fx, float *dyda, int ma)
{
    *fx = coefs[1]*exp(-x)+coefs[2]*exp(-coefs[3]*x);
    dyda[1] = exp(-x);
    dyda[2] = exp(-coefs[3]*x);
    dyda[3] = -coefs[2]*x*exp(-coefs[3]*x);
}

int main()
{
    int i, j, ia[P1+1], iaa[4];
    float chisq1, chisq2, chisq2a, alambda, sum, x, xtop;
    float *xi, *yi1, *yi2, *yi2a, *li1, *li2, *yi1data, *yi2data,
        *li1fit, *li2fit, *li2fitc, *sig;
    float **covar, **alpha;
    FILE *tmpfile1, *tmpfile2;

    /* initialize random number generator */
    unsigned int seed = time(NULL);
    srand(seed);

```

```

xi = vector(1, NP);
yi1 = vector(1, NP);
yi2 = vector(1, NP);
yi2a = vector(1, NP);
yi1data = vector(1, NP);
yi2data = vector(1, NP);
sig = vector(1, NP);
li1 = vector(1, P1);
li2 = vector(1, 3);
li1fit = vector(1, P1);
li2fit = vector(1, 3);
li2fitc = vector(1, 3);
covar = matrix(1, NP, 1, NP);
alpha = matrix(1, NP, 1, NP);

/* make random coefficients for the functions */
for (i = 1; i <= P1; i++)
    li1[i] = 100*(float)rand()/(float)RAND_MAX;
for (i = 1; i <= 3; i++)
    li2[i] = 100*(float)rand()/(float)RAND_MAX;

/* compute function values */
/* Now  $f_1(x) = \sum_{j=1}^p \lambda_j x^j$  */
/* and  $f_2(x) = \lambda_1 e^{-x} + \lambda_2 e^{-\lambda_3 x}$  */
for (i = 1; i <= NP; i++)
{
    x = (float)i*0.01;
    xi[i] = x;
    xtop = 1;
    sum = 0;
    for (j = 1; j <= P1; j++)
    {
        xtop *= x;
        sum += li1[j]*xtop;
    }
    yi1[i] = sum;
    yi2[i] = li2[1]*exp(-x)+li2[2]*exp(-li2[3]*x);

    /* std. deviations unknown, set = 1, see NRC, p. 671 */
    sig[i] = 1;
}

/* add random noise to function values, use this as data */
for (i = 1; i <= NP; i++)

```

```

{
  yi1data[i] = yi1[i]*(1+0.1*((float)rand()/(float)RAND_MAX-0.5));
  yi2data[i] = yi2[i]*(1+0.1*((float)rand()/(float)RAND_MAX-0.5));
}

/* ia[] tells lfit which coefficients to fit */
for (i = 1; i <= P1; i++)
  ia[i] = 1;

/* fit polynomial */
lfit(xi, yi1data, sig, NP, li1fit, ia, P1, covar, &chisq1, &powers);

alambda = -1;
for (i = 1; i <= 3; i++)
  {
  /* fit all coefficients */
  iaa[i] = 1;
  /* initial guess for coefficient */
  li2fit[i] = li2fitc[i] = 100*(float)rand()/(float)RAND_MAX;
  }

/* compute chi-square for random coefficients */
chisq2a = 0.0;
for (i=1; i<= NP; i++)
  chisq2a += SQR((yi1[i] - li2fit[1]*exp(-xi[i])+
                 li2fit[2]*exp(-li2fit[3]*xi[i]))/sig[i]);

/* fit nonlinear function, see description for mrqmin */
mrqmin(xi, yi2data, sig, NP, li2fit, iaa, 3, covar, alpha,
       &chisq2, &exps, &alambda);

while (alambda > 0.00001)
  mrqmin(xi, yi2data, sig, NP, li2fit, iaa, 3, covar, alpha,
       &chisq2, &exps, &alambda);

alambda = 0;
mrqmin(xi, yi2data, sig, NP, li2fit, iaa, 3, covar, alpha,
       &chisq2, &exps, &alambda);

printf("\n\nPOLYNOMIAL:\n\n");
printf("Original coefficients:\n");
for (i = 1; i <= P1; i++)
  printf("%10.4f ", li1[i]);
printf("\n\nCoefficients from lfit:\n");

```

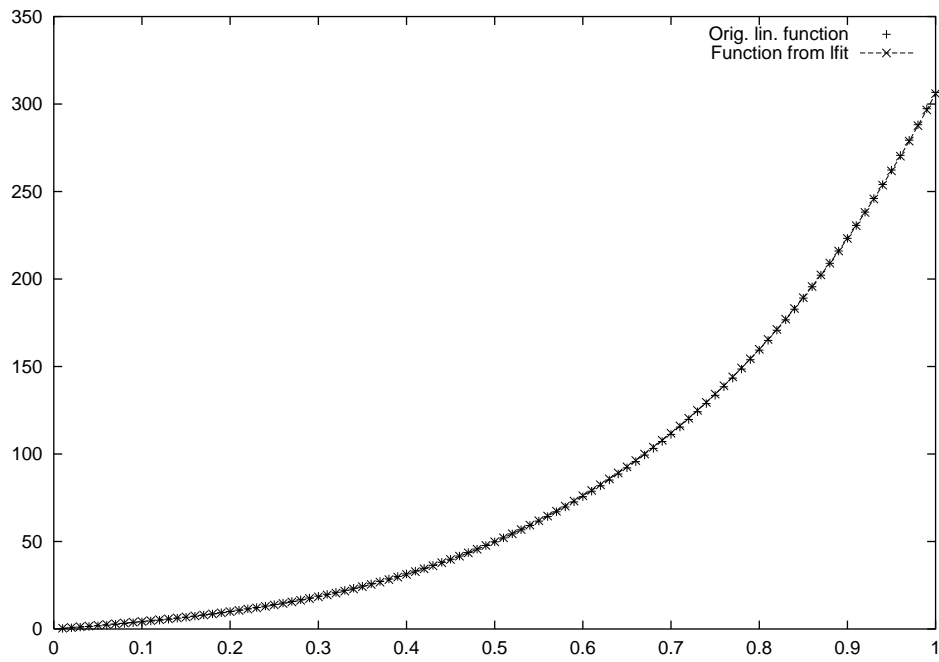
```

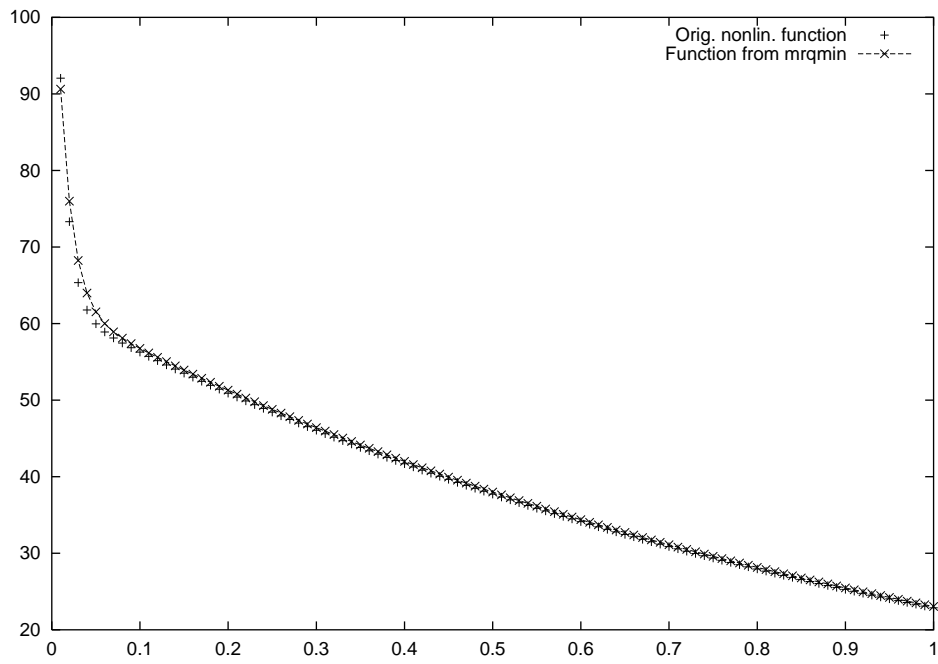
for (i = 1; i <= P1; i++)
    printf("%10.4f ", li1fit[i]);
printf("\n\nChi-square = %10.4f\n", chisq1);
printf("\n\nNON-LINEAR FUNCTION:\n\n");
printf("Original coefficients:\n");
for (i = 1; i <= 3; i++)
    printf("%10.4f ", li2[i]);
printf("\n\nRandom coefficients:\n");
for (i = 1; i <= 3; i++)
    printf("%10.4f ", li2fitc[i]);
printf("\n\nChi-square for random coefficients= %10.4f\n", chisq2a);
printf("\nCoefficients from mrqmin:\n");
for (i = 1; i <= 3; i++)
    printf("%10.4f ", li2fit[i]);
printf("\n\nChi-square = %10.4f\n\n\n", chisq2);
tmpfile1 = fopen("tmp1.dat","w");
tmpfile2 = fopen("tmp2.dat","w");
for (i=1; i<=NP; i++)
    {
        fprintf(tmpfile1,"%g %g\n",xi[i],yi1[i]);
        sum = 0;
        x = xi[i];
        xtop = 1;
        for (j = 1; j <= P1; j++)
            {
                xtop *= x;
                sum += li1fit[j]*xtop;
            }
        fprintf(tmpfile2,"%g %g\n",xi[i],sum);
    }
fclose(tmpfile1);
fclose(tmpfile2);
gnuplt2t(3, "tmp1.dat", "Orig. lin. function", 1,
        "tmp2.dat", "Function from lfit", 2,NULL);
tmpfile1 = fopen("tmp1.dat","w");
tmpfile2 = fopen("tmp2.dat","w");
for (i=1; i<=NP; i++)
    {
        x = xi[i];
        fprintf(tmpfile1,"%g %g\n",x,yi2[i]);
        fprintf(tmpfile2,"%g %g\n",x,
            li2fit[1]*exp(-x)+li2fit[2]*exp(-li2fit[3]*x));
    }
fclose(tmpfile1);
fclose(tmpfile2);

```



```
gnuplt2t(3, "tmp1.dat", "Orig. nonlin. function", 1,  
           "tmp2.dat", "Function from mrqmin", 2, NULL);  
  
return 0;  
}  
  
/* FILE: ch14ex1.c ends */
```





## 14.6 Robustia estimointia

Perusajatus robustissa estimoinnissa: Ei anneta ”pienien” poikkeamien sovitettavasta mallista häiritä.

Kirjan kuva s.700 osoittaa pns- ja robustin estimoinnin eron.

Esimerkkinä on suoran  $y = a + bx$  sovitus pisteistöön  $(x_i, y_i)$  minimoimalla absoluuttiset poikkeamat ts. summa

$$(1) \quad \sum_{i=1}^N |y_i - a - bx_i| = \Delta$$

**Huom.** Pns-sovituksessa kohdefunktio oli  $\chi^2$ .

Summa  $\sum_{i=1}^N |c_i - \alpha|$  minimoituu kun  $\alpha = c_n$ , lukujen  $c_i$  mediaani. Näin ollen (1) minimoituu kun

$$(2) \quad a = \text{mediaani}\{y_i - bx_i\},$$

(b kiinteä). Param.  $b$  puolestaan valit. s.e.  $\frac{\partial \Delta}{\partial b} = 0$  ts.

$$(3) \quad \sum_{i=1}^N N x_i \text{sgn}(y_i - a - bx_i) = 0.$$

Asetetaan tässä  $a$ :ksi (2):n antama luku, jolloin  $a = a(b, x_i, y_i)$  ja ratk. tämän jälkeen (3) välinpuolit. menetelmällä  $b$ :n suhteen. Saatu  $b$  sijoit.  $a$ :n lausekkeeseen.

Algm NR: :medfit s.704

## 15 TAVALLISET DIFFERENTIAALIYHTÄLÖT

### 15.0 Johdanto

**Perusongelma:** Annettu jatkuvat  $f_i(x, y_1, \dots, y_N)$   $i = 1, \dots, N$ . Etsi  $y_i$ ,  $i = 1, \dots, N$  s.e. (huom. tässä ei tarkastella deriv. suorittamista)

$$(1) \quad \frac{dy_i(x)}{dx} = f_i(x, y_1, \dots, y_N) \quad \forall i = 1, \dots, N.$$

**Esim.**

$$\begin{cases} \frac{dy_1}{dx} = x + y_1 + y_2^2 & \text{tai } y'(x) = \frac{\sin x}{x} \\ \frac{dy_2}{dx} = x^2 + y_1^2 + y_2 \end{cases}$$

Tavallisesti (1):n ratk. ei määräydy 1-käsitt. vaan on asetettava lisäehtoja, esim. annetaan  $y_i(x_0)$ ,  $i = 1, \dots, N$  (ns. **alkuarvottehtävä** AAT) tai  $y_i(x_0)$  ja  $y_i(x_1)$ ,  $i = 1, \dots, N$  ns. **reuna-arvottehtävä** RAT). Olem. olo ja 1-käsitt. lauseilla on keskeinen merkitys myös numeerisessa diff.yht. teoriassa.

AAT:n  $y' = f(x, y)$ ,  $y(x_0) = y_0$ , numeerinen ratkaisu välillä  $[x_0, \bar{x}_0]$ ,  $x_0 < \bar{x}_0$ , voidaan tehdä **Eulerin menetelmällä** seuraavasti. Valit. kok.luku  $n \geq 3$ . Merk.  $x_i = x_0 + hi$ ,  $h = (\bar{x}_0 - x_0)/n$  ja aset.  $y_{i+1} = y_i + hy'_i = y_i + hf(x_i, y_i)$ ,  $i = 0, 1, 2$ . Geom. tulkinta:

Saadaan ratkaisulle likiarvot pisteissä  $x_i$ .

Eulerin menetelmä on historiallisesti tärkeä mutta tehoton.

**Nykyiset reseptit** a) Runge-Kutta

b) Richardsonin ekstrapolointi (Bulirsch-Stoer)

a) Euler-tyylisiä askeleita (lisäpiirteitä tarkkuuden parant.)

b) lasket. useilla askelpituuksilla  $h$  ja ekstrapoloidaan rationaalilla approksimaatiolla tapaukseen  $h \sim 0$

NR erottaa algoritmien 3 tasoa:

→ eteneminen yhden tai useamman askeleen valitulla algoritmilla (NR::rk4;mmid)

→ askelpituuden säätö (NR::rkqc,bsstep)

→ ajuriohjelma (NR::rkdumb, odeint)

## 15.1. Runge-Kutta

Lähtökohtana Eulerin menetelmä

$$x_{n+1} = x_n + h,$$

$$y_{n+1} = y_n + hf(x_n, y_n) = y_n + k_1.$$

Otetaan keskelle koeaskel

$$k_2 = hf\left(x_n + \frac{h}{2}, y_n + \frac{k_1}{2}\right),$$

$$(1) \quad y_{n+1} = y_n + k_2 + O(h^3).$$

Kaava (1) antaa 2. kertaluvun R-K menetelmän. Yleensä sanotaan että menetelmä on kertalukua  $n$ , jos virhe  $= O(h^{n+1})$ .

Jatketaan yo. kaavoja

$$k_3 = hf\left(x_n + \frac{h}{2}, y_n + \frac{k_2}{2}\right),$$

$$k_4 = hf(x_n + h, y_n + k_3).$$

Määrit.

$$(2) \quad y_{n+1} = y_n + \frac{k_1}{6} + \frac{k_2}{3} + \frac{k_3}{3} + \frac{k_4}{6} + O(h^5).$$

Tämä on 4. kertaluvun R-K menetelmä. Kertoimien  $\frac{1}{6}, \frac{1}{3}, \frac{1}{3}, \frac{1}{6}$  valintaa (2):ssa voidaan perustella Taylor-kehitelmiä tutkimalla.

Algm NR: :rk 4 s.712 Ajuri NR: :rkdumb

## 15.2 R-K adaptiivisella askelpituudella

**Tavoite:** Saavuttaa ennalta annettu tarkkuus mahdollisin vähin laskuin. Tavoitteeseen pyritään askelpituuden säädöllä [virhearvio]  
Askeleen puolitus 4. kertaluvun R-K:n kanssa.

Olk. tarkka ratk.  $y$  ja sille approksimaatio  $y_1$  kun edet.  $x$ :stä  $x + 2h$ :hon ja appr.  $y_2$  kun edet. ensin  $x + h$ :hon ja sitten  $x + h$ :sta  $x + 2h$ :hon. Koska menetelmä on kertalukua 4, saadaan

$$y(x + 2h) = y_1 + (2h)^5(\phi) + O_1(h^6),$$

$$y(x + 2h) = y_2(x + h) + h^5\phi + O_2(h^6)$$

$$= y_2(x) + h^5\phi + O_3(h^6) + h^5\phi + O_2(h^6) = y_2 + 2h^5\phi + O_4(h^6).$$

Lauseke

$$\Delta = y_2 - y_1 = 30\phi h^5 + O(h^6)$$

antaa virhearvion. Olk.  $h_0$  toinen askelpituus ja sitä vastaava virhe  $\Delta_0$ . Silloin

$$h_0 \simeq h_1 \left| \frac{\Delta_0}{\Delta_1} \right|^{1/5}.$$

Jatkossa  $\Delta_0$  on *haluttu tarkkuus*. Kaava (1) kertoo:

-jos  $\Delta_1 > \Delta_0$ : paljonko vähennettävä askelpituutta  $h_1$  kun yritetään saavuttaa haluttu tarkkuus

-jos  $\Delta_1 < \Delta_0$ : paljonko voidaan kasvattaa askelpit.  $h_1$  ja silti saavuttaa haluttu tarkkuus

**Johtopäätös:** Kaava (1) soveltuu diff. yht. ratkaisun ”laaduntarkkailuun”. Askelpituus valit. kussakin vaiheessa mahdollisimman suureksi niin että haluttu tarkkuus saavutetaan:

Algm NR: :rkqs s. 719

Algm NR: :odeint s.721

```
// FILE: myod1.cpp begins
// g++ myod1.cpp -I../utils -I../democpp02 -L../lib -o a -lm -lnr
```

```
/*
```

```
The hypergeometric function F(a,b;c;x) satisfies
```

$dF(a,b;c;x)/dx = (ab/c)F(a+1,b+1;c+1;x)$  and  $F(a,b;c;0) = 1$ .

(See Chapter 6.)

For  $x$  in  $(0,1)$  and  $a,b$  in  $(0,1)$

$dF(a-1,b;c;x)/dx = (a-1)[F(a,b;c;x) - F(a-1,b;c;x)]/x$

$dF(a,b;c;x)/dx = [(c-a)F(a-1,b;c;x) + (a-c+bx)F(a,b;c;x)]/(x(1-x))$

Hence the IVP

$dy_1(x)/dx = (a-1)[y_2(x) - y_1(x)]/x$

$dy_2(x)/dx = [(c-a)y_1(x) + (a-c+bx)y_2(x)]/(x(1-x))$

$y_1(x_1) = F(a-1,b;c;x_1)$

$y_2(x_1) = F(a,b;c;x_1)$

has solution  $y_1 = F(a-1,b;c;x)$   $y_2 = F(a,b;c;x)$

```
*/
// Driver for routine odeint

#include <cstdlib> // Used in putmat2
#include <cstdio> // Used in putmat2
#include <ctime>
#include <string>
#include <fstream>
#include <iostream>
#include <iomanip>

#include <cmath>
#include "nr.h"
#include "matut102.cpp"

using namespace std;

#define N 2
#define KMAX 100

DP dxsav; // defining declarations
int kmax,kount;
Vec_DP *xp_p;
Mat_DP *yp_p;

int nrhs; // counts function evaluations

DP a= 0.7, b=0.5, c=1.5;

void derivs(DP x,const Vec_DP &y,Vec_DP &dydx)
{
```

```

    nrhs++;
    dydx[0] =(a-1.0)*(y[1]-y[0])/x;
    dydx[1]=((c-a)*y[0]+(a-c+b*x)*y[1])/(x*(1.0-x));
}

DP hyperg(DP a, DP b,DP c, DP x)
{
    complex<DP> ser,der,aa(a,0.0),bb(b,0.0),cc(c,0.0),xx(x,0.0);
    if (x>0.99) x=0.99;
    NR::hypser(aa,bb,cc,xx,ser,der);
    return ser.real();
}

int main(void)
{
    int i,nbad,nok;
    DP eps=1.0e-4,h1=0.1,hmin=0.0,x1=0.01,x2=0.99;
    DP tmp; // eps is the desired accuracy
    complex<DP> tmp2;
    Vec_DP ystart(N);
    xp_p=new Vec_DP(KMAX);
    yp_p=new Mat_DP(N,KMAX);
    Vec_DP xp=*xp_p;
    Mat_DP yp=*yp_p;
    ystart[0]=hyperg(a-1.0,b,c,x1);
    ystart[1]=hyperg(a,b,c,x1);
    nrhs=0;
    kmax=100;
    dxsav=(x2-x1)/20.0;
    NR::odeint(ystart,x1,x2,eps,h1,hmin,nok,nbad,derivs,NR::rkqs);
    cout<<"successful steps:"<<setw(15)<<nok<<endl;
    cout<<"bad steps:"<<setw(22)<<nbad<<endl;
    cout<<"function evaluations:"<<setw(11)<<nrhs<<endl;
    cout<<endl<<"stored intermediate values: " <<kount<<endl;
    cout<<endl;
    cout<<"          x          integral          F(a,b;c;x)          Error";
    cout<<endl;
    for (i=0;i<kount-1;i++){
        tmp=hyperg(a,b,c,(*xp_p)[i]);
        cout<<setw(10)<<(*xp_p)[i]<<setw(17)<<(*yp_p)[1][i]<<setw(15);
        cout<<tmp<<setw(20)<<tmp-(*yp_p)[1][i]<<endl;
    }
    return 0;
}
// FILE: myod1.cpp ends

```



```

successful steps:          3
bad steps:                9
function evaluations:     117

stored intermediate values: 9

```

x	integral	F(a,b;c;x)	Error
0.01	1.00235	1.00235	0
0.11	1.02722	1.02722	1.64666e-06
0.324758	1.09176	1.09177	2.64781e-06
0.641458	1.23791	1.23795	4.09625e-05
0.702466	1.27996	1.28	3.73756e-05
0.778566	1.34513	1.34516	3.33739e-05
0.865942	1.45088	1.45091	2.85362e-05
0.932215	1.58244	1.58247	2.48521e-05

NR::odeint soveltuu myös määrättyjen integraalien laskuun erit. jos integrandissa on rajuja vaihteluja.

**Huom.** Algm NR::rkqs:ssa kaavaa (1) käytetään muokatussa muodossa.

```

// FILE: myod2.cpp begins
// We use odeint to compute
// int_{x1}^{x2} 100*sin(pi*x)
// Driver for routine odeint
// g++ myod2.cpp -I../utils -I../democpp02 -L../lib -o a -lm -lnr

#include <cstdlib> // Used in putmat2
#include <cstdio> // Used in putmat2
#include <ctime>
#include <string>
#include <fstream>
#include <iostream>
#include <iomanip>

#include <cmath>
#include "nr.h"
#include "matut102.cpp"

using namespace std;

#define N 1

```

```

#define KMAX 100

DP dxsav; // defining declarations
int kmax,kount;
Vec_DP *xp_p;
Mat_DP *yp_p;

int nrhs; // counts function evaluations

void derivs(DP x,const Vec_DP &y,Vec_DP &dydx)
{
    nrhs++;
    dydx[0] =100.0*sin(M_PI*x);
}

int main(void)
{
    int i,ii,nbad,nok;
    DP eps=1.0e-4,h1=0.1,hmin=0.0,x1=0.0,x2=1.0;
    DP tmp; // eps is the desired accuracy
    Vec_DP ystart(N);
    xp_p=new Vec_DP(KMAX);
    yp_p=new Mat_DP(N,KMAX);
    for (ii=1;ii<=6;ii++){
        eps =eps/5.0;
        ystart[0]=0.0;
        nrhs=0;
        kmax=100;
        dxsav=(x2-x1)/20.0;
        NR::odeint(ystart,x1,x2,eps,h1,hmin,nok,nbad,derivs,NR::rkqs);
        cout<<endl<<endl<<"eps = "<<eps;
        cout<<endl<<"successful steps:"<<setw(15)<<nok<<endl;
        cout<<"bad steps:"<<setw(22)<<nbad<<endl;
        cout<<"function evaluations:"<<setw(9)<<nrhs<<endl;
        cout<<endl<<"stored intermediate values: " <<kount<<endl;
        cout<<endl;
        cout<<"          x          integral          Error"<<endl;
        Vec_DP xp=*xp_p;
        Mat_DP yp=*yp_p;
        for (i=0;i<kount;i++) {
            tmp=100*(cos(M_PI*x1)-cos(M_PI*xp[i]))/M_PI;
            cout<<setw(10)<<xp[i]<<setw(17)<<yp[0][i];
            cout<<setw(16)<<yp[0][i]-tmp<<endl;}
    }
}

```

```

    return 0;
}
// FILE: myod2.cpp ends

```

```

eps = 2e-05
successful steps:          17
bad steps:                1
function evaluations:     153

```

```

stored intermediate values: 6

```

x	integral	Error
0	0	0
0.0952162	1.41352	-5.10758e-08
0.258262	9.91483	-4.3075e-06
0.475795	29.4128	-1.60266e-05
0.736166	53.3397	9.5355e-06
1	63.6621	9.03394e-05

```

eps = 4e-06
successful steps:          20
bad steps:                1
function evaluations:     196

```

.....

### 15.3 Modifioitu keskipistemennetelmä

**Idea:** Edetään pisteestä  $x$  pisteeseen  $x + H$  jonolla  $n$  kpl peräkk. osa-askelia, joiden pituus on  $h = H/n$ .

$$\begin{cases} z_0 = y(x) \\ z_1 = z_0 + hf(x, z_0) \\ z_{m+1} = x_{m-1} + 2hf(x + mh, z_m), \quad m = 1, 2, \dots, n-1 \\ y(x + H) \approx y_n \equiv \frac{1}{2}[z_n + z_{n-1} + hf(x + H, z_n)] \end{cases}$$

Vrt. Euler R-K.

Algm NR: :mmids.723 on tarkoitettu lähinnä käytettäväksi Bulirsch-Stoerin menetelmä kanssa (myös)

#### 15.4 Richardsonin ekstrapolointi ja Bulirsch-Stoerin menetelmä

Menetelmä ei sovellu tapaukseen, missä  $dy$  sisältää ei-sileitä funktioita (esim. taulukoituja tai mitattuja) tai singulariteettejä integrointivälillä.

Bulirsch-Stoerin menetelmän 3 piirrettä:

1) Richardsonin ekstrapolointi (vrt. Rombergin menetelmä numeerisessa integroinnissa): Numeerisen laskun vastaus  $v(h)$  riippuu parametrinä  $h$  analyttisesti ( $h$  esim. askelpituus). Tunnetaan  $v(h_i)$ ,  $i = 1, \dots, p$ , ekstrapoloidaan arvoon  $h = 0$  sovittamalla dataan  $(h_i, v(h_i))$ ,  $i = 1, \dots, p$ , jokin ”luonnollinen” käyrä.

2) Sovitettava käyrä on rationaalifunktio (Bulirsch-Stoer)

3) Käytet. menetelmää, jonka virheftio on parillinen (vrt. NR s.723, Gragg)

Parametriä  $h$  pienennetään käyttämällä tark. välin  $(x, x + H)$  kulkemiseen  $n_j$  askelta, ts.  $h_j = H/n_j$ , missä  $n_j = 2n_{j-2}$ ,  $n = 2, 4, 6, 8, 12, 16, 24, 32, 48, 64, 96$ .

Algm NR: :bsstep

```
// FILE: myod3.cpp begins
// Driver for routine odeint
// Example from Sewell ISBN 0-12-637475-9 p. 63
// g++ myod3.cpp -I../utils -I../democpp02 -L../lib -o a -lm -lnr
#include <cstdlib> // Used in putmat2
#include <cstdio> // Used in putmat2
#include <ctime>
#include <string>
#include <fstream>
#include <iostream>
#include <iomanip>

#include <cmath>
#include "nr.h"
```

```

#include "matut102.cpp"

using namespace std;

#define N 1
#define KMAX 100

DP dxsav; // defining declarations
int kmax,kount;
Vec_DP *xp_p;
Mat_DP *yp_p;

int nrhs; // counts function evaluations

void derivs(DP x,const Vec_DP &y,Vec_DP &dydx)
{
    nrhs++;
    dydx[0] = 100.0 / (1.0 + 10000.0 * x * x);
}

int main(void)
{
    int i, nbad, nok;
    DP eps = 1.0e-4, h1 = 0.1, hmin = 0.0,
        x1 = -1.0, x2 = 2.0;
    DP sol;
    Vec_DP ystart(N);
    xp_p=new Vec_DP(KMAX);
    yp_p=new Mat_DP(N,KMAX);
    ystart[0] = atan(-100.0);
    nrhs = 0;
    kmax = 100;
    dxsav = (x2 - x1) / 20.0;
    NR::odeint(ystart, x1, x2, eps, h1, hmin, nok, nbad, derivs,
        NR::rkqs);
    printf("\n%s %13s %3d\n", "successful steps:", " ", nok);
    printf("%s %20s %3d\n", "bad steps:", " ", nbad);
    printf("%s %9s %3d\n", "function evaluations:", " ", nrhs);
    printf("\n%s %3d\n", "stored intermediate values: ", kount);
    printf("\n%8s %18s %15s %10s\n",
        "x", "integral", "atan(100*x)", "Error");
    Vec_DP xp=*xp_p;
    Mat_DP yp=*yp_p;

    for (i = 0; i < kount; i++)

```

```

{
  sol = atan(100.0 * xp[i]);
  printf("%10.4f %16.6f  %14.6f  %12.4e\n",
        xp[i], yp[0][i], sol, yp[0][i] - sol);
}
return 0;
}
// FILE: myod3.cpp ends

```

```

successful steps:          12
bad steps:                 13
function evaluations:     215

```

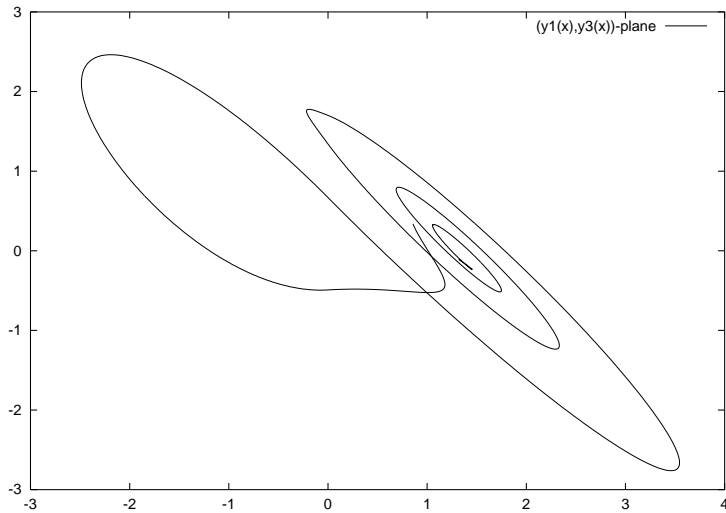
```

stored intermediate values: 9

```

x	integral	atan(100*x)	Error
-1.0000	-1.560797	-1.560797	0.0000e+00
-0.4000	-1.545827	-1.545802	-2.5712e-05
-0.1863	-1.517212	-1.517185	-2.7585e-05
-0.0301	-1.249923	-1.249887	-3.6121e-05
0.1262	1.491761	1.491712	4.8518e-05
0.3511	1.542385	1.542325	5.9654e-05
0.6179	1.554678	1.554613	6.4309e-05
1.1340	1.562047	1.561978	6.8324e-05
2.0000	1.565866	1.565796	6.9805e-05

Seuraava esimerkki havainnollistaa kappaleen liikerataa 3-ulotteisessa avaruudessa, kun nopeus on annettu. Riippuen nopeuden lausekkeesta, liikerata voi olla hyvinkin kaaottinen, vrt. kuva.



```
// FILE: myod4.cpp begins
// Driver for routine rkdump
// Studies the Silnikov system of ODE's.
// See Kocak, p. 148
/* g++ -Wall myod4.cpp -L../lib -I../utils -I../gnuplot02 -o a -lm -lnr */

#include <cstdlib> // Used in putmat2
#include <cstdio> // Used in putmat2
#include <ctime>
#include <string>
#include <fstream>
#include <iostream>
#include <iomanip>
#include <cmath>

using namespace std;

#include "nr.h"
#include "matut102.cpp"

#define PRINT 1
#include "gnuplt2.h"

#define NVAR 3
#define NSTEP 2000

DP agl=0.33, bgl=0.75, cgl=0.63, dgl=0.0; // Define global variables
```

```

DP f(DP x)
{
    DP r =1.0-bgl*x;
    if (x <=0) r=1.0+cgl*x;
    return r;
}

void derivs(DP x,const Vec_DP &y,Vec_DP &dydx)
{
    dydx[0] = y[1];
    dydx[1] = y[2];
    dydx[2] =-y[1]-agl*y[2]+f(y[0]);
}

void plot(Vec_DP &x, Mat_DP &y, int n, int m, const char *name)
{
    int i;
    ofstream fp("z1.dat");
    for (i=1;i<n;i++)
        fp<<x[i]<<" "<<y[m][i]<<endl;
    fp.close();
    gnuplt2("z1.dat",name,0,NULL);
}

Vec_DP *xx_p;    // defining declaration
Mat_DP *y_p;

int main(void)
{
    int i,j,ii,yn =1;
    DP x1=12.0,x2=40.0;
    Vec_DP vstart(NVAR);
    DP y1=0.0, y2=1.0, y3= 1.0, tmp;
    const char *name="(y1(x),y3(x))-plane";

    // Note: The arrays xx and y must have indices up to NSTEP
    xx_p=new Vec_DP(NSTEP+1);
    y_p=new Mat_DP(NVAR,NSTEP+1);

    vstart[0]=y1;
    vstart[1]=y2;
    vstart[2]=y3;
    for (ii=1;ii<10;ii++)
    {

```



```

if (ii>1)
{
  vstart[0]=rdm(0.8,1.5);
  vstart[1]=rdm(-0.5,0.5);
  vstart[2]=rdm(-0.2,0.5);
}
NR::rkdumb(vstart,x1,x2,derivs);
Vec_DP &xx=*xx_p;
Mat_DP &y=*y_p;
printf("%8s %17s %10s %10s\n","x","y1(x)","y2(x) ", "y3(x)");
for (j=10;j<NSTEP;j+=10)
  printf("%10.4f %14.6f %12.6f %12.6f\n",
    xx[j],y[0][j], y[1][j], y[2][j]);
for (i=1;i<NSTEP;i++) xx[i]=y[0][i];
plot(xx,y,NSTEP-1, 2, name);
}
return 0;
}
// FILE: myod4.cpp ends

```

## 16 REUNA-ARVOTEHTÄVÄT

### 16.0 Johdanto

Perustehtävä: Etsi funktiot  $y_i$  s.e  $\forall x \in (x_1, x_2)$

$$(1) \quad \begin{cases} \frac{dy_i(x)}{dx} = g_i(x, y_1, \dots, y_N), & i = 1, \dots, N \\ (1a) \quad B_{1j}(x_1, y_1, \dots, y_N) = 0, & j = 1, \dots, n_1 \\ (1b) \quad B_{2j}(x_2, y_1, \dots, y_N) = 0, & j = 1, \dots, N - n_1 \end{cases}$$

**Ratkaisumenetelmät:** a) tähtäysmenetelmä

b) relaksaatiomenetelmä

c) differenssimenetelmä

a) **Tähtäysmenetelmä:** Kiinnitetään kaikkien  $y_i$ :iden arvot  $x_1$ :ssä se. (1a) toteutuu ( $n_1$  kpl ehtoja). Ratkaistaan diff.yht. näillä alkuarvoilla ja tutkit. ratk. arvoja  $x_2$ :ssa, jolloin (1b) ei välttämättä toteudu. Säättämällä  $N - n_1$  kpl parametreja pakotetaan myös (1b) voimaan. (Myös void. lähteä liikkeelle molemm. päistä  $x_1, x_2$  ja vaatia, että saadut ratkaisut liittyvät toisiinsa ”sileästi” ennalta valituksa liitospisteessä.) Newtonin moniulotteinen juurenhakumenetelmä tulee käyttöön.

b) **Relaksaatiomenetelmä:**

-valitaan jakopisteet

-diff.yht. korvataan jakopisteissä pätevillä differenssiyhtälöillä

-ratkaisuyritteen jakopisteissä annetut arvot ovat tuntemattomia, differenssiyhtälöiden eikä reunaehtojen tarvitse toteutua

-säädetään ratkaisuyritteen arvoja jakopisteissä niin että differenssiyhtälöt ja reunaehdot lähenevät askel askeleelta haluttuja.

c) **Differenssimenetelmä**

-valitaan jakopisteet

-muodostetaan kuhunkin pisteeseen liittyvä yhtälö huomioiden erityisesti reunaehdot

-ratkaistaan saatu harva lineaarinen yhtälöryhmä

-soveltuu lähinnä lineaariseen tapaukseen

## 16.1 Tähtäysmenetelmä

16.0 (1):n ratkaisufunktiot  $y_i$ ,  $i = 1, \dots, N$ , toteuttavat  $n_1$  kpl ehtoja  $x_1$ :ssä, joten jää  $n_2 = N - n_1$  vapaasti valittavaa alkuarvoa  $x_1$ :ssä.

Voidaan kirjoittaa

$$y_i(x_1) = y_i(x_1; V_1, \dots, V_{n_2}) \quad i = 1, \dots, N.$$

Annetulle  $\bar{V} = (V_1, \dots, V_{n_2})$  saadaan  $\bar{y}(x_1) = (y_1(x_1), \dots, y_N(x_1))$ , josta edelleen näillä alkuarvoilla saadaan (1):n AAT:n ratkaisu  $x_2$ :ssa (integ.  $x_1$ :sta  $x_2$ :een esim. odeint). Määritellään poikkeamavektori  $\bar{F} = (F_1, \dots, F_{n_2}) = (B_{21}(x_2, \bar{y}), \dots, B_{2n_2}(x_2, \bar{y}))$

Siis etsitään  $\bar{V}$ :n arvoa s.e.  $\bar{F} = 0$ , joten haluttu  $\bar{V}$ :n korjaus on  $\delta\bar{V}$  (vrt. Newton)

$$[\alpha] \cdot \delta V = -F, \quad [\alpha]_{ij} = \frac{\partial F_i}{\partial V_j}$$

ja siis  $\bar{V}_{new} = \bar{V}_{old} + \delta\bar{V}$ .

Tässä käytetään approksimaatiota

$$\frac{\partial F_i}{\partial V_j} \approx \frac{F_i(V_1, \dots, V_j + \Delta V_j, \dots) - F_i(V_1, \dots, V_j, \dots)}{\Delta V_j}$$

Alg. shoot s.758

## 16.2. Tähtäys liitospisteeseen

Jos tähtäysmenetelmässä alkuarvot ovat huonosti valittuja, niin voi olla, ettei vastaava numeerinen ratkaisu ole edes määritelty koko välillä  $[x_1, x_2]$ . Tällöin voi olla mahdotonta edetä pisteestä  $x_1$  pisteeseen  $x_2$ .

Vaikeutta voidaan yrittää torjua valitsemalla jokin liitospiste  $x_f \in (x_1, x_2)$  s.e.  $x_1$ :stä voidaan edetä  $x_f$ :ään ja  $x_2$ :sta (toiseen suuntaan)  $x_f$ :ään. Pisteessä  $x_i$  pätee  $n_i$ ,  $i = 1, 2$ , ehtoa, joten  $x_1$ :stä

aloitettaessa voidaan  $N - n_i$  param. valita vapaasti ( $N = n_1 + n_2$ ).  
Merkitään

$$y_i(x_1) = y_i(x_1; V_{(1)1}, \dots, V_{(1)n_2}), \quad i = 1, \dots, N,$$

$$\bar{y}_i(x_2) = \bar{y}_i(x_2; V_{(2)1}, \dots, V_{(2)n_1}), \quad i = 1, \dots, N,$$

missä  $y_i$  tot. RAT:n  $x_2$ :ssä ja  $\bar{y}_i$   $x_1$ :ssa. Yhteensopivuusvaatimus, joka ratkaistaan Newtonin menetelmällä

$$y_i(x_f; V_{(1)}) = \bar{y}_i(x_f; V_{(2)}), \quad i = 1, \dots, N,$$

Alg. shootf s. 761

### 16.3 Relaksaatiomenetelmä

Korvataan ODE (ordinary diff.eq) FDE:llä (finite diff.eq.)

$$(1) \quad \text{ODE} : \frac{dy}{dx} = g(x, y),$$

$$(2) \quad \text{FDE} : y_k - y_{k-1} = (x_k - x_{k-1})g\left[\frac{x_k + x_{k-1}}{2}, \frac{y_k + y_{k-1}}{2}\right].$$

Yleensä  $N$  yhtälöä  $M$  jakopistettä  $\Rightarrow N \times M$  muuttujaa.

Huomaa, että FDE kytkee kaksi vierekkäistä pisteparia nim.  $(x_{k-1}, y_{k-1})$  ja  $(x_k, y_k)$ .

Etsitään ratkaisua välillä  $x_1, x_M$   $x_1 < x_2 < \dots < x_M$   $N$ :nnän  $dy$ :n reuna-arvot tehtävälle,  $n_1$  ehtoa  $x_1$ :ssa,  $n_2 = N - n_1$  pist.  $x_M$ . Merkitään  $\bar{y}_k = (y_1, \dots, y_N)|_{x=x_k}$ . Tässä tapauksessa kaavan (2) yleistys on seuraava

$$(3) \quad 0 = \bar{E}_k = \bar{y}_k - \bar{y}_{k-1} - (x_k - x_{k-1})\bar{g}_k(x_k, x_{k-1}\bar{y}_k, \bar{y}_{k-1}), \quad k = 2, 3, \dots, M.$$

FDE (3) kytkee  $2N$  muuttujaa pist.  $k - 1, k$  ja sen komponentit antavat  $N$  yhtälöä. Yhtälöt soveltuvat indeksin arvoilla  $k = 2, \dots, M$ ,

yhteensä  $M - 1$  kpl yhtälöryhmiä. Siis yht.  $N(M - 1)$  yhtälöä  $MN$  tuntemattomalle. Puuttuvat  $N$  yhtälöä saadaan reunaehdoista.

Pist.  $x_1$  saadaan

$$(4) \quad 0 = E_1 = B(x_1, \bar{y}_1).$$

Toisessa päätepisteessä  $x_M$  saadaan

$$(5) \quad 0 = E_{M+1} = C(x_M, \bar{y}_M).$$

Vektorilla  $E_1$  ja  $B$  on vain  $n_1$  vapaasti muuttuvaa komponenttia (reunaehdot  $x_1$ :ssä). Oletetaan että ne ovat viimeiset  $n_1$  komponenttia ts.  $E_{j,1} \neq 0$  kun  $j = n_2 + 1, \dots, N$ . Toisessa reunapisteessä oletetaan, että vain ensimmäiset  $n_2$  komponenttia  $E_{j,M+1}$ ,  $j = 1, \dots, n_2$ , muuttuvat vapaasti.

**FDE:n (3)-(5) ratkaisu:** muuttujia ovat  $y_{j,k}$ , jotka ovat  $N$ :n muuttujien  $y_j$  arvot  $M$  pisteissä  $x_k$ . Etsitään muuttujaan  $y_{j,k}$  liittyvä korjaus  $\Delta y_{j,k}$  s.e.  $y_{j,k} + \Delta y_{j,k}$  on parannettu ratkaisu.

Korjaustermien tot. yhtälö saadaan Taylor-sarjasta

$$(6) \quad E_k(\bar{y}_k + \Delta \bar{y}_k, \Delta \bar{y}_{k-1} + \Delta \bar{y}_{k-1}) \sim \\ E_k(\bar{y}_k, \bar{y}_{k-1}) + \sum_{n=1}^N \frac{\partial E_k}{\partial y_{n,k-1}} \Delta y_{n,k-1} + \sum_{n=1}^N \frac{\partial E_k}{\partial y_{n,k}} \Delta y_{n,k}.$$

Vaativuus  $E(\bar{y} + \Delta \bar{y}) = 0$  johtaa yhtälöihin

$$(7) \quad \sum_{n=1}^N S_{j,n} \Delta y_{n,k-1} + \sum_{n=N+1}^{2N} S_{j,n} \Delta y_{n-N,k} = -E_{j,k}; \quad j = 1, \dots, N,$$

$$(8) \quad S_{j,n} = \frac{\partial E_{j,k}}{\partial y_{n,k-1}}, \quad S_{j,n+N} = \frac{\partial E_{j,k}}{\partial y_{n,k}}, \quad n = 1, \dots, N.$$

$[S_{j,n}]$  on  $N \times 2N$  matriisi kullakin indeksillä  $k$ .

Siis kukin jakopiste antaa  $N$  yhtälöryhmän, joka kytkee  $2N$  korjaustermiä pisteissä  $k, k - 1$ .

Samoin voidaan tehdä reunapisteissä:

$$(9) \quad \sum_{n=1}^N S_{j,n} \nabla y_{n,1} = -E_{j,1}, \quad j = n_2 + 1, \dots, N,$$

missä

$$(10) \quad S_{j,n} = \frac{\partial E_{j,1}}{\partial y_{n,1}}, \quad n = 1, \dots, N,$$

$$(11) \quad \sum_{n=1}^N S_{j,n} \Delta y_{n,M} = -E_{j,M+1}, \quad j = 1, \dots, n_2,$$

$$(12) \quad S_{j,n} = \frac{\partial E_{j,M+1}}{\partial y_{n,M}} \quad n = 1, \dots, N.$$

NR:n kuva 17.3.1 s. 765 esittää ko. matriisin rakenteen. Korjaukset saadaan ratkaisemalla ko. yhtälöryhmä. Ratkaisun välivaiheena Gaussin eliminoinnin variantti ks. kirja s. 764, joka käyttää hyväkseen matriisin lohkostruktuuria, "harvuutta", muistintarpeen minimoimiseksi.

## 16.4 Differenssimenetelmistä

Lineaarisen differentiaaliyhtälön

$$(1) \quad L(y) = y'' + p_1(x)y' + p_2(x) = q(x)$$

*reuna-arvot tehtävän (RAT)*

$$(2) \quad \begin{cases} L(y) = q(x) \\ \alpha_1 y(a) + \alpha_2 y'(a) = A \\ \beta_1 y(b) + \beta_2 y'(b) = B \end{cases}$$

ratkaisemiseksi voidaan käyttää ns. *differenssimenetelmää*.

Derivaatat korvataan tässä approksimaatioilla

$$(3) \quad y'(x) = \frac{y(x+h) - y(x-h)}{2h}, \quad h > 0,$$

$$(4) \quad y''(x) = \frac{y(x+h) - 2y(x) + y(x-h)}{h^2}, \quad h > 0.$$

Differenssiyhtälöiden käyttöä RAT:n 16.4 (2) ratkaisuun esitellään Luvussa 20. Pääpiirteisssään menetelmä on seuraava:

1) **Diskretointi**  $x_i = x_0 + ih, x_0 = a, h = \frac{b-a}{N}, i = 0, 1, \dots, N, y_i = y(x_i)$ . (3)&(4)  $\Rightarrow$

$$(5) \quad y_{i+1} - 2y_i + y_{i-1} + p_1(x_i)(y_{i+1} - y_{i-1})\frac{h}{2} + p_2(x_i)y_i h^2 = h^2 q(x_i)$$

2) **Lineaaristen yhtälöiden muodostus** muuttujille  $y_0, y_1, \dots, y_N$ . Huomioidaan reuna-arvot. Saadaan muotoa

$$\begin{bmatrix} a_1 & c_1 & \cdots & 0 \\ b_2 & a_2 & c_2 & \cdots \\ \vdots & & & c_{n-1} \\ 0 & \cdots & b_{n-1} & a_n \end{bmatrix} \begin{bmatrix} z_1 \\ \vdots \\ z_n \end{bmatrix} = \begin{bmatrix} w_1 \\ \vdots \\ w_n \end{bmatrix}$$

oleva lineaarinen yhtälöryhmä, jossa  $n = N + 1, z_i = y_{i-1}, i = 1, \dots, n$  ja  $w_i, i = 1, \dots, n$  määräytyy kaavoista (5) ja RAT:stä (2).

### 3. Lin. yhtälöiden ratkaisu

Yo. yhtälöryhmä on tridiagonaalinen, joten se voidaan ratkaista käyttäen alg. tridag .

## 17 DIFFERENSSIMENETELMISTÄ

### 17.0 Johdanto

Differentiaaliyhtälöiden numeeriseen käsittelyyn on kehitetty ns. *differentssimenetelmä*, joka perustuu määrittelyalueen tasaväliseen diskretointiin ja derivaattojen approksimointiin erotusosamäärillä. Tarkastelemme jatkossa vain 1- tai 2-ulotteisia tilanteita. Jälkimmäisessä tapauksessa diskretointi tapahtuu koordinaattiakselien suuntaisilla suorakulmioilla ja oletamme että määrittelyalue on monikulmio, jonka sivut ovat koordinaattiakselien suuntaisia. Jos suorakulmiot ovat neliöitä, voidaan sallia myös sivuja, joiden päätepisteet ovat neliöiden kärkipisteitä, ja jotka muodostavat  $45^\circ$  kulman koordinaattiakselien suuntien kanssa.

Taylorin kaavasta saadaan yhden muuttujan funktiolle  $y$ :

$$y(x+h) = y(x) + hy'(x) + \frac{1}{2}h^2y''(x) + \dots$$

$$y(x-h) = y(x) - hy'(x) + \frac{1}{2}h^2y''(x) - \dots$$

$$y(x+h) + y(x-h) = 2y(x) + h^2y''(x) + O(h^3)$$

$$(17.1) \quad \Rightarrow y'(x) = \frac{y(x+h) - y(x-h)}{2h} + O(h^2),$$

$$(17.2) \quad \Rightarrow y'(x) = \frac{y(x+h) - y(x)}{h} + O(h^2),$$

$$(17.3) \quad \Rightarrow y''(x) = \frac{y(x+h) - 2y(x) + y(x-h)}{h^2} + O(h^3).$$

Oletamme nyt, että  $u$  on kahden muuttujan funktio ja jaamme tason yhteneviin suorakulmioihin, kärkipisteinä  $(x_i, y_j)$ , missä  $x_i =$



$ih, y_j = jk, i, j = 0, \pm 1, \pm 2, \pm 3, \dots$  ja  $h(k)$  on suorakulmion leveys (korkeus). Merkitsemme vielä

$$(17.4) \quad u_{i,j} = u(x_i, y_j).$$

Ylläolevien yhden muuttujan tapausta koskevien tarkastelujen nojalla saadaan

$$(17.5) \quad \frac{\partial^2 u}{\partial x^2}(x_i, y_j) \approx \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{h^2},$$

$$(17.6) \quad \frac{\partial^2 u}{\partial y^2}(x_i, y_j) \approx \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{k^2}.$$

Näiden kaavojen soveltaminen kahden muuttujan differentiaaliyhtälön numeeriseen ratkaisuun reuna-arvotehtävien tapauksessa etenee periaatteessa seuraavasti.

1) Numeroimme alueen sisällä olevat verkkopisteet  $(x_i, y_j)$  yhdellä indeksillä  $r = 1, \dots, m$ . Tuntemattomista  $u_{i,j}$  muodostuu nyt vektori  $p = (p_1, \dots, p_m)$ . Tavallisesti käytämme jatkossa  $u_{i,j}$ :lle verkkopisteen mukaista indeksiä  $r$ . Alueen reunalla olevissa verkkopisteissä etsittävän ratkaisun arvot saadaan reuna-arvotehtävästä.

2) Muodostamme  $m$  kpl yhtälöitä, yhden kussakin pisteessä  $(x_i, y_j)$ , kaavojen (17.5) ja (17.6) mukaisesti. Pisteessä  $(x_i, y_j)$  saadaan yhtälö, jossa esiintyvät tuntemattomat  $u_{i,j}, u_{i\pm 1,j}, u_{i,j\pm 1}$ , tai reuna-arvot jos jokin naapuripisteistä  $u_{i\pm 1,j}, u_{i,j\pm 1}$  sattuu olemaan reunapiste.

3) Ratkaistaan saatu yhtälöryhmä. Tarkastellemme jatkossa vain tilanteita, joissa ryhmä on lineaarinen. Silloin ratkaisu voidaan tehdä esim. Gaussin eliminoinnin tai LU-hajoituksen avulla.

On selvää, että ratkaisun tarkkuus riippuu diskretoinnin parametreista  $h$  ja  $k$ . Ei kuitenkaan ole mitenkään itsestään selvää, että virhe  $\rightarrow 0$  kun  $h, k \rightarrow 0$ . Virhearvioita on voitu johtaa eräissä erikoistapauksissa. Ne sisältävät riippuvuuden mm. tarkasteltavasta yhtälöstä, ratkaisufunktioiden 4. kertaluvun osittaisderivaatoista ja luvuista  $h$  ja  $k$ .

Jatkossa sovellamme yo. diskretointimenettelyä tavallisen diff-yhtälön reuna-arvotehtävän ratkaisuun:

$$D(y) = 0; y(0) = a, y(1) = b,$$

missä  $D(y) = 0$  on differentiaaliyhtälö (esim. toisen kertaluvun vakiokertoiminen diff. yhtälö) tai osittaisdifferentiaaliyhtälön reuna-arvot tehtävän ratkaisuun tason monikulmiossa.

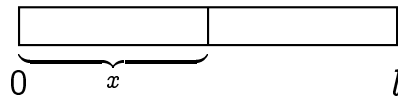
Tärkeitä ”prototyyppejä” osittaisdifferentiaaliyhtälöistä ovat ( $u : D \rightarrow R$ )

$$(1) \quad \frac{\partial u}{\partial t} = \alpha^2 \frac{\partial^2 u}{\partial x^2} \quad \text{lämpöyhtälö}$$

$$(2) \quad \frac{\partial^2 u}{\partial t^2} = c^2 \frac{\partial^2 u}{\partial x^2} \quad \text{aaltoyhtälö}$$

$$(3) \quad \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0 \quad \text{Laplacen yhtälö}$$

$$(4) \quad \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = p(x, y); \quad u|_{\partial D} = g \quad \text{Poissonin yhtälö}$$



(1):n ratkaisu  $u(x, t)$  kuvaa (esim.) lämpötilaa homogeenisen sauvan kohdassa  $x$ , ajanhetkellä  $t$ ,  $0 < x < l$ ,  $\alpha^2$  kerroin.

(2) esiintyy akustisten, vesi-, ja sähkömagneettisten aaltojen tutkimuksessa (myös värähtelevä kieli).

(3) esiintyy sähkö- ja magneettikenttien potentiaalien tutkimuksessa  $\rightarrow$  toinen nimitys *potentiaaliyhtälö*.

(4) esiintyy mm. nestedynamiikassa.

Kuten tavallisten DY:den tapauksessa ODY:den (1)–(4) ratkaisut eivät yleensä ole yksikäsitteisiä. Tarvittaessa yhtälöihin (1)–(4) liitetään erilaisia täydentäviä ehtoja (esim. reunaehtoja) jotka takaavat ratkaisujen yksikäsitteisyyden.

Yhtälön (1) tapauksessa reunaehdot voivat esim. olla (i) lämpötilajakauma hetkellä  $t = 0$  ts.  $u(x, 0) = f(x)$  ja lisäksi (ii)  $u(0, t) = u(l, t) = 0 \quad \forall t \geq 0$  tai  $u_x(0, t) = u_x(l, t) = 0$ .

Yhtälön (2) tapauksessa reunaehto voi olla, kun tarkastellaan värähtelevää kieltä, (i) kielen asema kun  $t = 0$  (ii) kielen nopeus hetkellä  $t = 0$  (iii) päätepisteet paikallaan. Ts.  $u(x, 0) = f(x)$ ,  $u_t(x, 0) = g(x)$ ,  $u(0, t) = u(l, t) = 0$ .

Yhtälön (3) tapauksessa ei ole aikaparametria  $t$ . Luonnollinen lisäehto (3):een on esim. se että annetaan  $u$ :n arvot alueen reunalla. Näin täydennetty (3) on kuuluisa "Dirichletin probleema". Yhtälön (3) ratkaisuja sanotaan harmonisiksi funktioiksi. Esimerkkejä harmonisista funktioista ovat muotoa  $x^2 + Axy - y^2 + Bx + Cy$ , missä  $A, B, C$ , ovat vakioita, olevat funktiot.

Ratkaisun yksikäsitteisyys/olemassaolo voi olla selvää esim. fysiikkalisin perustein. Jatkossa oletamme että näin on asianlaita.

ODY-teorian klassisena lähtökohtana on luokittelu yhtälöiden (1)–(3) mukaisesti prototyyppisiin ja kunkin luokan ominaispiirteiden selvittely. Täsmällisemmin ODY

$$A \frac{\partial^2 u}{\partial x^2} + B \frac{\partial^2 u}{\partial x \partial y} + C \frac{\partial^2 u}{\partial y^2} = f(x, y, \frac{\partial u}{\partial x}, \frac{\partial u}{\partial y}, u)$$

on *elliptinen parabolinen* t. *hyperbolinen* sen mukaan onko  $B^2 - 4AC < 0, = 0, > 0$ . Tämä kolmijako muodostaa toisen kertaluvun ODY-teorian perustan.

### 17.1. Tavallisen diff. yhtälön reuna-arvotettava

Lineaarisen differentiaaliyhtälön

$$(1) \quad L(y) = y'' + p_1(x)y' + p_2(x)y = q(x)$$

*reuna-arvotettävän* (RAT)

$$(2) \quad \begin{cases} L(y) = q(x) \\ \alpha_1 y(a) + \alpha_2 y'(a) = A \\ \beta_1 y(b) + \beta_2 y'(b) = B \end{cases}$$

ratkaisemiseksi voidaan käyttää ns. *differenssi-menetelmää*.

Derivaatat korvataan tässä approksimaatioilla (17.1) ja (17.3).

Differenssi-menetelmä lineaarisen differentiaaliyhtälön (1) RAT:n  $y(a) = A, y(b) = B$  ratkaisemiseksi on seuraava:

1) **Diskretointi.**  $x_i = x_0 + ih, x_0 = a, h = \frac{b-a}{N}, i = 0, 1, \dots, N, y_i = y(x_i)$ . (17.1)&(17.3)  $\Rightarrow$

$$(3) \quad y_{i+1} - 2y_i + y_{i-1} + p_1(x_i)(y_{i+1} - y_{i-1})\frac{h}{2} + p_2(x_i)y_i h^2 = h^2 q(x_i)$$

(4)

$$y_{j-1}(1 - p_1(x_j)\frac{h}{2}) + y_j(-2 + p_2(x_j)h^2) + y_{j+1}(1 + p_1(x_j)\frac{h}{2}) = h^2 q(x_j),$$

$$j = 1, \dots, N - 1.$$

Ensimmäisessä yhtälössä  $y_0(1 - p_1(x_1)\frac{h}{2})$  ja viimeisessä yhtälössä  $y_N(1 - p_1(x_{N-1})\frac{h}{2})$  ovat vakioita, jotka määräytyvät reuna-arvoista ja ne siirretään oikealle puolelle.

2) **Lineaaristen yhtälöiden muodostus.** Tuntemattomia ovat  $y_1, \dots, y_{N-1}$ . Huomioidaan reuna-arvot. Saadaan muotoa

$$\begin{bmatrix} a_1 & c_1 & \cdots & 0 & & \\ b_2 & a_2 & c_2 & \cdots & & \\ \vdots & & & & c_{n-1} & \\ 0 & \cdots & & b_n & a_n & \end{bmatrix} \begin{bmatrix} z_1 \\ \vdots \\ z_n \end{bmatrix} = \begin{bmatrix} w_1 \\ \vdots \\ w_n \end{bmatrix}$$

oleva lin.yht. ryhmä, jossa  $n = N - 1, z_i = y_i, i = 1, \dots, n$  ja  $w_i, i = 1, \dots, n$  määräytyy kaavoista (3) ja RAT:stä.

### 3. Lin. yhtälöiden ratkaisu

Yo. yhtälöryhmä on tridiagonaalinen, joten se voidaan ratkaista käyttäen algoritmia tridag.

#### Esimerkki.

Ratkaise reuna-arvot tehtävä  $y'' = x + (1 - x/5)y, y(1) = 2, y(3) = -1$ , diskretoinnilla

$$y''(x) = (y(x-h) + y(x+h) - 2y(x))/(h * h), h = 2/10.$$

Ratkaisu tehdään seuraavalla algoritmilla.

```

// FILE: mybvp2.cpp begins
/* g++ mybvp2.cpp -L../lib -I../utils -o a -lm -lnr */
// Diagonal entries a[1]..a[N]
// superdiagonal entries c[1]..c[N-1]
// subdiagonal entries b[2]..b[N]
/* Solve BVP
   y(a) = y0, y(b) = y1
   y'' + p1(x)y' + p2(y)y = q(x)
   at points x_j = a+j h ; h =(b-a)/n, j =1,...,n-1 */

#include <iostream>
#include <iomanip>
#include "nr.h"

using namespace std;

#define NP 10

DP p1(DP x)
{
    return 0.0;
}
DP p2(DP x)
{
    return -1.0+(x/5.0);
}
DP q(DP x)
{
    return x;
}

int main()
{
    int k,n=NP;
    Vec_DP diag(NP-1),superd(NP-1),subd(NP-1),rhs(NP-1), u(NP-1);
    DP a,b, h, x, y0, y1;
    a=1.0; b=3.0; h=(b-a)/(DP)n;
    y0 =2.0; y1=-1.0;
    x=a+h;
    for (k=0;k<n-2;k++)
    {
        diag[k]=-2.0+h*h*p2(x);
        superd[k]=1.0+p1(x)*h/2.0;
        subd[k+1]=1.0-p1(x)*h/2.0;
        rhs[k]=h*h*q(x);
    }
}

```

```

    x=x+h;
}
diag[n-2]=-2.0+h*h*p2(b-h);
rhs[0]=rhs[0]-y0*(1.0-p1(a+h)*h/2.0);
rhs[n-2]=rhs[n-2]-y1*(1.0+p1(b-h)*h/2.0);
subd[0]=0.0;
NR::tridag(subd,diag,superd,rhs,u);

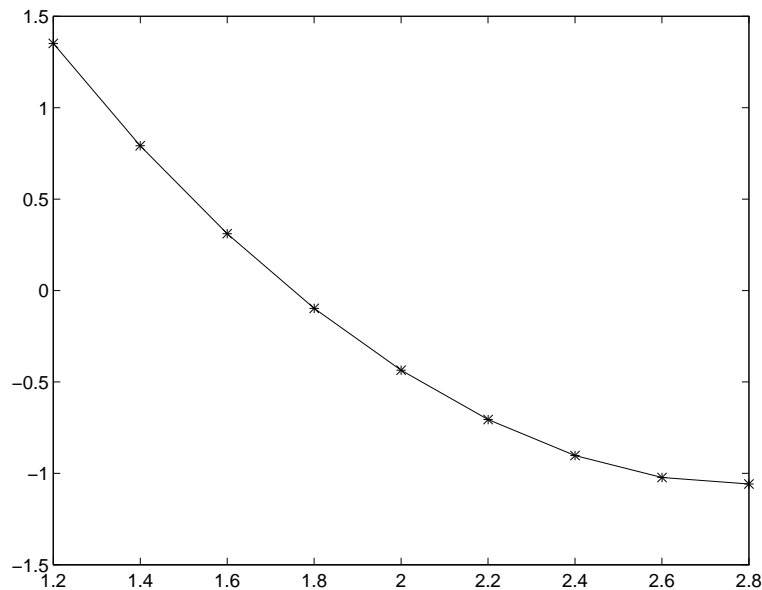
cout<<"\nThe solution vector is:\n";
cout<<setw(16)<<"x"<<setw(16)<<"u(x)"<<endl;
for (k=0;k<n-1;k++)
    cout<<setw(16)<<a+(k+1.0)*h<<setw(16)<<u[k]<<endl;
return 0;
}
// FILE: mybvp2.cpp ends

```

Tulokseksi saadaan:

The solution vector is:

x	u(x)
1.2	1.35909
1.4	0.807501
1.6	0.335166
1.8	-0.0640526
2	-0.392911
2.2	-0.651199
2.4	-0.836074
2.6	-0.94234
2.8	-0.962698



## 17.2. Osittaisdifferentiaaliyhtälöiden reuna-arvotekävät

Differenssimenetelmä soveltuu myös osittaisdifferentiaaliyhtälöiden numeeriseen ratkaisuun monikulmioalueissa. Edellä kohdassa 17.1 tavallisen differentiaaliyhtälön reuna-arvotekävä palautettiin tri-diagonaalisen lineaarisen yhtälöryhmän ratkaisuun. Osittaisdifferentiaaliyhtälöiden tapauksessa ratkaisu palautuu vastaavien väli-vaiheiden jälkeen sellaisen harvan lineaarisen yhtälöryhmän ratkaisuun, missä  $\neq 0$  alkioita on vain diagonaalilla ja eräillä sivudiagonaaleilla.

**Laplace-yhtälön numeerinen ratkaisu.** Laplace-yhtälön

$$\nabla^2 u = u_{xx} + u_{yy} = 0$$

diskretointiin käytämme approksimointia

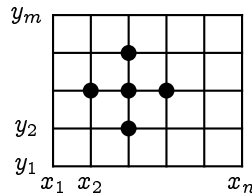
$$f''(x) = \frac{f(x+h) - 2f(x) + f(x-h)}{h^2} + O(h^2)$$

joten

$$\nabla^2 u(x, y) =$$

$$(1) \quad \frac{u(x+h, y) + u(x-h, y) + u(x, y+h) + u(x, y-h) - 4u(x, y)}{h^2} + O(h^2).$$

Jaetaan suorakulmio  $R = \{(x, y) : 0 \leq x \leq a, 0 \leq y \leq b, b/a = m/n\}$   $(n-1) \times (m-1)$  neliöön sivunpituus  $= h$  ( $a = nh, b = mh$ ).



Merkitään

$$x_i = (i-1)h, \quad i = 1, \dots, n, \quad y_j = (j-1)h, \quad j = 1, \dots, m.$$

(1)  $\Rightarrow$

$$(2) \quad \nabla^2 u_{i,j} = \frac{u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1} - 4u_{i,j}}{h^2} = 0.$$

(2) on nimeltään *viiden pisteen differenssikaava* Laplace-yhtälölle. Kaava ilmaisee eräänlaisen keskiarvo-ominaisuuden arvoille  $u_{i,j}$ . Tämä on diskreetti analogia harmonisten funktioiden tunnetulle keskiarvo-ominaisuudelle.

Oletamme, että arvot  $u_{i,j}$  tunnetaan reunalla (kun  $i = 1$  tai  $n+1$ ,  $j = 1$  tai  $m+1$ ), kuten on laita esim. Dirichletin ongelmassa.

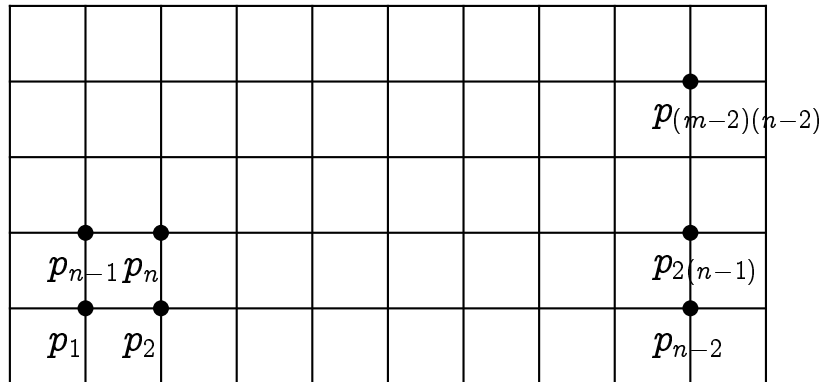
Verkon sisäpisteitä ( $2 \leq i \leq n-1$ ,  $2 \leq j \leq m-1$ ) on  $(n-2) \times (m-2)$  kpl ja tuntemattomia  $u_{i,j}$  samoin.

Muodostetaan tuntemattomista vektori

$$p = (p_1, \dots, p_{n-1}, p_{n-1}, \dots, p_{(n-2) \times (m-2)})$$

seuraavasti



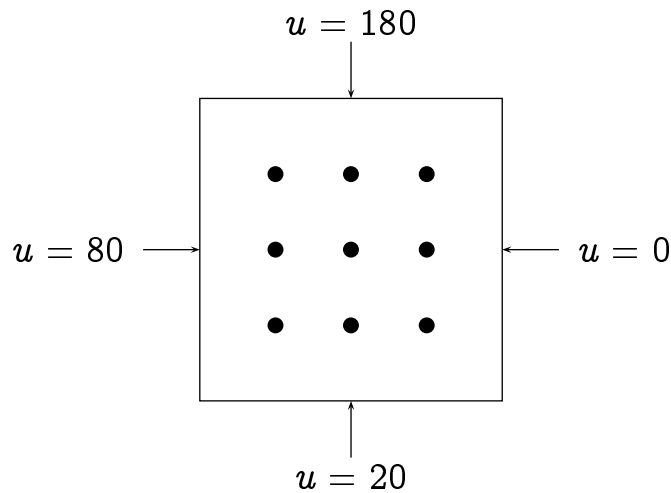


Yhtälöt kirjoitetaan  $p_i$ :lle indeksin  $i$  mukaisessa järjestyksessä käyttäen kaavaa (2) ja huomioiden reuna-arvot.

**Esimerkki.** Etsi Dirichletin probleeman  $\nabla^2 u = 0$ ,

$$\begin{cases} u(x, 0) = 20, & u(x, 4) = 180, & 0 < x < 4, \\ u(0, y) = 80, & u(4, y) = 0, & 0 < y < 4, \end{cases}$$

ratkaisu alueen  $R = \{(x, y) : 0 \leq x \leq 4, 0 \leq y \leq 4\}$  pisteissä  $(i, j)$ ,  $i = 1, 2, 3, j = 1, 2, 3 (h = 1)$ .



Tuntemattomien indeksointi on seuraavan taulukon mukainen:

```

7 8 9
4 5 6
1 2 3

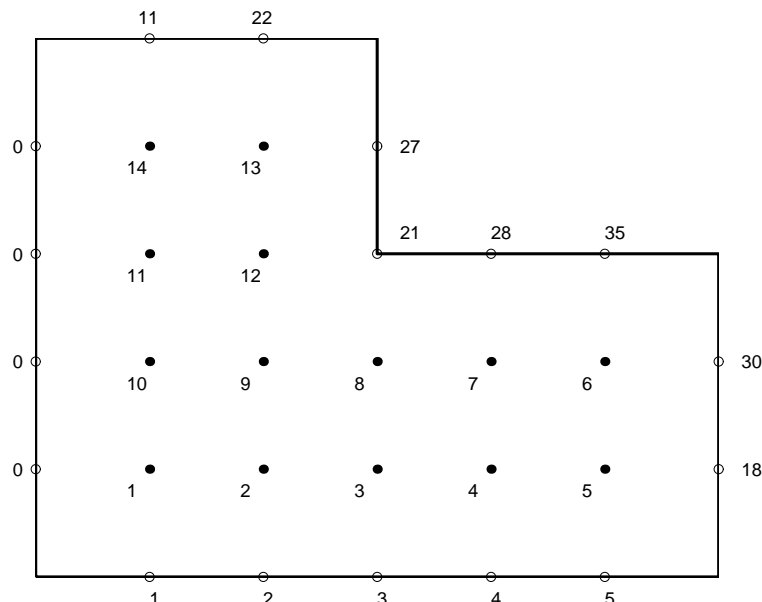
```

1. yhtälö on  $80 + p_2 + 20 + p_4 - 4p_1 = 0$

2. yhtälö on  $p_1 + p_3 + 20 + p_5 - 4p_2 = 0$   
 Saadaan yhtälöryhmä (Mathews s. 524)

$$\begin{array}{rcccccccl}
 -4p_1 + p_2 & & + p_4 & & & & & = -100 \\
 p_1 - 4p_2 + p_3 & & & + p_5 & & & & = -20 \\
 & p_2 - 4p_3 & & & + p_6 & & & = -20 \\
 p_1 & & - 4p_4 + p_5 & & + p_7 & & & = -80 \\
 p_2 & + p_4 - 4p_5 + p_6 & & + p_8 & & & & = 0 \\
 & p_3 & + p_5 - 4p_6 & & & + p_9 & & = 0 \\
 & & p_4 & & - 4p_7 + p_8 & & & = -260 \\
 & & & p_5 & + p_7 - 4p_8 + p_9 & & & = -180 \\
 & & & & + p_6 & + p_8 - 4p_9 & & = -180
 \end{array}$$

**Esimerkki.** Seuraava ohjelma ratkaisee kuvanmukaisen Dirichletin ongelman. Ongelmassa on valmiiksi muodostettu diskretointi, tuntemattomien numerointi alueen sisäpisteissä ja annettu reuna-arvot reunalla olevissa verkon kulmapisteissä.



Ratkaisussa voidaan erottaa seuraavat vaiheet.

(1) Alueen ja reuna-arvojen kuvailu jatkokon sopivassa muodossa.

(2) Verkkopisteiden numerointi ja yhtälöiden muodostus.

(3) Syötetiedoston muodostus. Jos on  $m$  kpl yhtälöitä, niin muodostetaan  $m \times 6$  matriisi, jonka 1 sarake on rivin indeksi, sarakkeet 2-5 ovat 4:n lähimmän naapuripisteen indeksit jos naapuripisteitä puuttuu, niin viimeistä indeksiä toistetaan niin, että saadaan 4 indeksiä ja 6 sarake on naapuripisteiden joukossa mahdollisesti olevien reuna-arvojen summa. Kuvan tapauksessa syötetiedosto mylu2.inp on

```
14 6
1 2 10 10 10 1
2 1 3 9 9 2
3 2 4 8 8 3
4 3 5 7 7 4
5 4 6 6 6 23
6 5 7 7 7 65
7 4 6 8 8 28
8 3 7 9 9 21
9 2 8 10 12 0
10 1 9 11 11 0
11 10 12 14 14 0
12 9 11 13 13 21
13 12 14 14 14 49
14 11 13 13 13 11
```

(4) Ratkaistaan yhtälöryhmä ja tulostetaan ratkaisu.

```
// FILE: mylu2.cpp
/* g++ mylu2.cpp -I../utils -I../democpp02 -L../lib -o a -lm -lnr */

#include <cstdlib> // Used in putmat2
#include <cstdio> // Used in putmat2
#include <ctime>
#include <string>
#include <fstream>
#include <iostream>
#include <iomanip>
#include <cstdint>
#include "nr.h"
#include "matutl02.cpp"
#include "solve.cpp"
#include <cmath>
```

```

#define NP 14

using namespace std;

void showintmat(Mat_DP &a)
{
    int i,j;
    for (i=0;i<a.nrows();i++)
    {
        for (j=0;j<a.ncols();j++)
            cout<<setw(4)<<floor(a[i][j]);
        cout<< endl;
    }
}

int main(int argc, char *argv[])
{
    int j,p,k;
    Mat_DP struc;
    const char *fnamea="mylu2.inp";
    if ( argc >= 2) {
        fnamea = argv[1];
    }
    struc = getmat(fnamea);
    Mat_DP a(struc.nrows(),struc.ncols());
    unitmat(a);
    Vec_DP b(struc.nrows(), x(struc.nrows()));
    if (struc.nrows() > NP) cout<<"Dimension error in matrix \n";
    for (j=0;j<a.nrows();j++) a[j][j]=-4.0*a[j][j];
    for (j=0;j<a.nrows();j++)
    {
        b[j]=-struc[j][struc.ncols()-1];
        for (p=1; p<struc.ncols()-1;p++)
        {
            k =(int) floor(struc[j][p]);
            if (j!=k-1) a[j][k-1]= 1.0;
        }
    }
    showintmat(a);
    for (p=0;p<struc.nrows();p++) cout<<" " <<b[p]<<endl;
    for (p=0;p<struc.nrows();p++) x[p] =0.0;
    LUsolve(a,b,x);
    cout<<"\nLU-solution: \n";
    for (j=0;j<a.nrows();j++)

```

```
        cout<<" x["<<j<<"] = "<<x[j]<<endl;
    return 0;
}
// FILE: mylu2.cpp
```

```
/* mylu2.inp :
```

```
14 6
1 2 10 10 10 1
2 1 3 9 9 2
3 2 4 8 8 3
4 3 5 7 7 4
5 4 6 6 6 23
6 5 7 7 7 65
7 4 6 8 8 28
8 3 7 9 9 21
9 2 8 10 12 0
10 1 9 11 11 0
11 10 12 14 14 0
12 9 11 13 13 21
13 12 14 14 14 49
14 11 13 13 13 11
*/
```

Käännetyn ohjelman ajo voidaan tehdä esim. komennolla  
mylu2 mylu2.inp > mylu2.dat jolloin ohjelma lukee tiedos-  
tosta mylu2.inp syötteet ja kirjoittaa tulokset tiedostoon mylu2.dat.  
Alla tiedosto mylu2.dat.

```

-4  1  0  0  0  0  0  0  0  1  0  0  0  0
 1 -4  1  0  0  0  0  0  1  0  0  0  0  0
 0  1 -4  1  0  0  0  1  0  0  0  0  0  0
 0  0  1 -4  1  0  1  0  0  0  0  0  0  0
 0  0  0  1 -4  1  0  0  0  0  0  0  0  0
 0  0  0  0  1 -4  1  0  0  0  0  0  0  0
 0  0  0  1  0  1 -4  1  0  0  0  0  0  0
 0  0  1  0  0  0  1 -4  1  0  0  0  0  0
 0  1  0  0  0  0  0  1 -4  1  0  1  0  0
 1  0  0  0  0  0  0  0  1 -4  1  0  0  0
 0  0  0  0  0  0  0  0  0  1 -4  1  0  1
 0  0  0  0  0  0  0  0  1  0  1 -4  1  0
 0  0  0  0  0  0  0  0  0  0  1 -4  1  0
 0  0  0  0  0  0  0  0  0  0  1  0  1 -4

```

```

-1
-2
-3
-4
-23
-65
-28
-21
-0
-0
-0
-21
-49
-11

```

LU-solution:

```

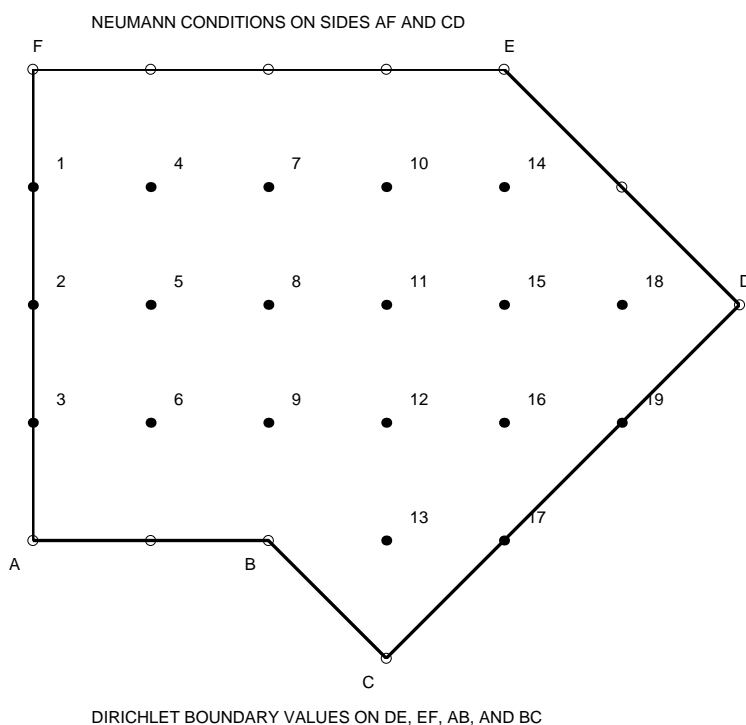
x[0] = 3
x[1] = 6
x[2] = 9
x[3] = 12
x[4] = 15
x[5] = 25
x[6] = 20
x[7] = 15
x[8] = 10
x[9] = 5
x[10] = 7
x[11] = 14
x[12] = 18
x[13] = 9

```

**Esimerkki.** Tarkastamme seuraavaksi monikulmiota  $G$  jonka kärkipisteinä ovat ABCDEF, ja  $G$ :ssä Poissonin yhtälöä Neumannin-Dirichletin reuna-arvoin

$$\begin{cases} u_{xx}(x, y) + u_{yy}(x, y) = g(x, y), & (x, y) \in G, \\ u(x, y) = f(x, y), & (x, y) \text{ sivuilla AB, BC tai DE, EF}, \\ \frac{\partial u}{\partial n}(x, y) = 0, & \text{kun } (x, y) \text{ sivuilla AF tai CD}. \end{cases}$$

Jälkimmäinen, reunan normaaliderivaattaehto, on nimeltään Neumannin ehto. Ruudun sivunpituus on  $h = 1/4$ . Tuntemattomia on nyt kahta eri tyyppiä, nimittäin sellaisia, jotka vastaavat alueen pisteitä (muut kuin indeksejä 1, 2, 3, 17, 19 vastaavat) ja alueen reunalla olevia pisteitä (indeksit 1, 2, 3, 17, 19).



Tarkastamme aluksi ensimmä. tyyppiä olevia pisteitä  $P(x_i, y_j)$ . Näihin pisteisiin liittyvä diskretointi antaa yhtälön

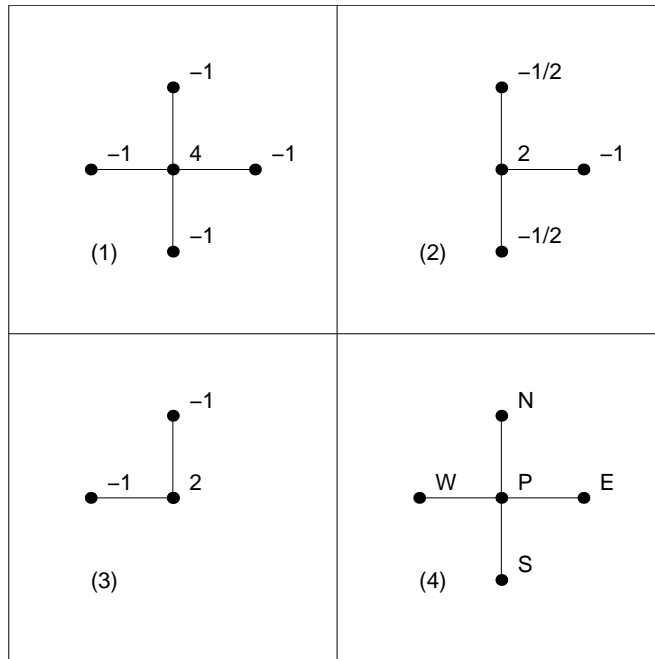
$$\frac{u_E - 2u_P + u_W}{h^2} + \frac{u_N - 2u_P + u_S}{h^2} = g_P \equiv g(x_i, y_j),$$

josta edelleen saadaan

$$(1) \quad 4u_P - u_N - u_W - u_S - u_E + h^2 g_P = 0.$$

Ilmansuuntien mukainen indeksointi ilmenee oheisen kuvan kohdasta (4) ja niihin liittyvät painot kohdasta (1).

(1): INTERIOR POINT (2): NEUMANN BRY CONDITION (VERTICAL SIDE)



(3): NEUMANN BRY CONDITION (45 DEGREE)

Seuraavaksi tarkastamme pystysuoralla päätysivulla olevia pisteitä 1, 2, 3, joissa vaaditaan Neumannin reunaehto. Jos  $P$  on tällainen piste, kuvitellaan  $P$ :n länsipuolelle symmetrinen apupiste, ja ehto

$$\frac{\partial u}{\partial n} \approx \frac{u_W - u_E}{2h} = 0$$

antaa  $u_E = u_W$ . Huomioimalla tämä kaavassa (1) saadaan

$$4u_P - u_N - 2u_W - u_S + h^2 g_P = 0,$$

$$(2) \quad 2u_P - \frac{1}{2}u_N - u_W - \frac{1}{2}u_S + \frac{1}{2}h^2 g_P = 0.$$

Pystysuoran sivun Neumann-reunaehdon diskreetointiin liittyvät painot ilmenevät kuvan kohdasta (2).



Jäljellä on vielä  $45^\circ$  kulmassa oleva päätysivu CD, ja siellä olevat pisteet 17, 19 joissa myöskin vaaditaan Neumannin ehto. Jos  $P$  on tällainen piste, niin oletamme kuten edellä, symmetriasyistä, että  $u_E = u_W, u_S = u_N$ , jolloin (1) yksinkertaistuu muotoon

$$4u_P - 2u_N - 2u_W + h^2 g_P = 0,$$

ja edelleen muotoon

$$(3) \quad 2u_P - u_N - u_W + \frac{1}{2}h^2 g_P = 0,$$

$45^\circ$  kaltevuuskulmassa olevan sivun Neumann-reunaehdon diskretointiin liittyvät painot ilmenevät kuvan kohdasta (3).

Tuntemattomiin 1, 2, 3 liittyvät tyyppiä (2) ovat yhtälöt, tuntemattomiin 4, .., 16 ja 18 tyyppiä (1), ja tuntemattomiin 17, 19 tyyppiä (3) olevat yhtälöt.

Ratkaisuvaiheessa kirjoitetaan yhtälöryhmä muotoon  $Ax = b$  ja ratkaistaan se esim. LU-menetelmällä.

Alla esitämme miten ratkaisu tehdään erikoistapauksessa  $h = 1, g(x, y) \equiv 0, f(x, y) = 0$ , kun  $(x, y) \in DE, EF$   $f(x, y) = 1$ , kun  $(x, y) \in AB, BC$ .

Syötetiedosto tehdään samaan tyyliin kuin ohjelmassa mylu2.c. Kun tiedosto on luettu sisään, luettuun matriisiin tehdään korjauksia, jotka aiheutuvat Neumannin reuna-arvoja vastaavien yhtälöiden korjaamisesta.

```
FILE: myneum.inp begins
19 6
1 4 2 2 2 0
2 5 3 3 3 0
3 2 6 6 6 0
4 1 5 7 7 0
5 2 4 6 8 0
6 3 5 9 9 1
7 4 8 8 10 0
8 5 7 9 11 0
9 6 8 12 12 1
```

```

10 7 11 14 14 0
11 10 8 15 12 0
12 9 11 13 16 0
13 12 17 17 17 1
14 10 15 15 15 0
15 11 14 16 18 0
16 12 15 17 19 0
17 13 16 16 16 0
18 15 19 19 19 0
19 16 18 18 18 0

```

FILE: myneum.inp ends

```

// FILE: myneum.cpp begins
/* Solve  $u_{xx}+u_{yy} = 0$  in G
   where G is bounded by polygonal domain ABCDEF,
   Dirichlet condition 0 on DE and EF
   Dirichlet condition 1 on AB and BC
   Neumann condition on AF and CD.
   FILE: myneum.inp gives input assuming h =1
*/
#include<iostream>
#include<iomanip>
#include<cmath>
#include<cstdio>
#include "nr.h"
#include "matut102.h"
#include "solve.h"

void showmyamat(Mat_DP &a,Vec_DP &b)
{
    int i,j;
    for (i=0;i<a.nrows();i++)
    {
        for (j=0;j<a.ncols();j++)
        {
            if ((j==0) && (i != 1)) cout<<" ";
            if ((j==0) && (i == 1)) cout<<" ";
            if ((i>2) && (j == 2)) cout<<" ";
            if ( fabs(a[i][j] +4.0) <1e-10)
                printf("%- 3d", (int) floor(a[i][j] + 1e-10));
            else if ( fabs(a[i][j] -1.0) <1e-10)
                printf("%- 3d", (int) floor(a[i][j] + 1e-10));
            else if ( fabs(a[i][j] +2.0) <1e-10)
                printf("%- 3d", (int) floor(a[i][j] + 1e-10));
        }
    }
}

```

```

        else if ( fabs(a[i][j] -0.5) <1e-10) cout<<"1/2 ";
        else if ( fabs(a[i][j] ) <1e-10)
            printf("%- 3d", (int) floor(a[i][j] + 1e-10));
        if (j==a.ncols()-1) cout<<setw(6)<< b[i];
    }
    cout << endl;
}
}

int main()
{
    int j, p, k;
    Mat_DP struc;
    const char *fnamea="myneum.inp";
    struc = getmat(fnamea);
    Mat_DP a(struc.nrows(),struc.nrows());
    unitmat(a);
    Vec_DP b(struc.nrows()), x(struc.nrows());
    a=-4.0*a;
    for (j=0;j<struc.nrows();j++)
    {
        b[j]=-struc[j][struc.ncols()-1];
        for (p=1; p<struc.ncols()-1;p++)
        {
            k =(int) floor(struc[j][p]);
            if (j!=k-1) a[j][k-1]= 1.0;
        }
    }
    // Set a[j][k] =1,k not j,
    // if x_k occurs on j:th row
    // Vertical side Neumann: AF
    for (j=0;j<3;j++)
        for (p=0;p<19;p++) a[j][p]=0.0;
    a[0][0]=-2.0; a[0][3]=1.0; a[0][1]=0.5; b[0]=-0.5*0;
    a[1][1]=-2.0; a[1][0]=0.5; a[1][2]=0.5; a[1][4]=1.0; b[1]=-0.5*0;
    a[2][2]=-2.0; a[2][1]=0.5; a[2][5]=1.0; b[2]=-0.5*1.0;
    // 45 degree side Neumann: CD
    for (p=0;p<19;p++) a[16][p]=0.0;
        for (p=0;p<19;p++) a[18][p]=0.0;
    a[16][16]=-2.0; a[16][12]=1.0; a[16][15]=1.0; b[16]=0.0;
    a[18][18]=-2.0; a[18][15]=1.0; a[18][17]=1.0; b[18]=0.0;
    showymat(a, b);
    for (p=0;p<struc.nrows();p++) x[p] =0.0;
    LUsolve(a,b,x);
    cout<<"\nLU-solution: "<<endl;
    for (j=0;j<struc.nrows();j++)

```

```

        cout<<" x["<<j<<"] = "<<x[j]<<endl;
    return 0;
}
// FILE: myneum.c ends

```

```

/* myneum.inp:

```

```

19 6
1 4 2 2 2 0
2 1 5 3 3 0
3 2 6 6 6 0
4 1 5 7 7 0
5 2 4 6 8 0
6 3 5 9 9 1
7 4 8 8 10 0
8 5 7 9 11 0
9 6 8 12 12 1
10 7 11 14 14 0
11 10 8 15 12 0
12 9 11 13 16 0
13 12 17 17 17 1
14 10 15 15 15 0
15 11 14 16 18 0
16 12 15 17 19 0
17 13 16 16 16 0
18 15 19 19 19 0
19 16 18 18 18 0

```

```

*/

```

```

-2 1/2 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 -0.00
1/2 -2 1/2 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 -0.00
0 1/2 -2 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 -0.50
1 0 0 -4 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 -0.00
0 1 0 1 -4 1 0 1 0 0 0 0 0 0 0 0 0 0 0 -0.00
0 0 1 0 1 -4 0 0 1 0 0 0 0 0 0 0 0 0 0 -1.00
0 0 0 1 0 0 -4 1 0 1 0 0 0 0 0 0 0 0 0 -0.00
0 0 0 0 1 0 1 -4 1 0 1 0 0 0 0 0 0 0 0 -0.00
0 0 0 0 0 1 0 1 -4 0 0 1 0 0 0 0 0 0 0 -1.00
0 0 0 0 0 0 1 0 0 -4 1 0 0 1 0 0 0 0 0 -0.00
0 0 0 0 0 0 0 1 0 1 -4 1 0 0 1 0 0 0 0 -0.00
0 0 0 0 0 0 0 0 1 0 1 -4 1 0 0 1 0 0 0 -0.00
0 0 0 0 0 0 0 0 0 0 1 -4 0 0 0 1 0 0 0 -1.00
0 0 0 0 0 0 0 0 0 1 0 0 0 -4 1 0 0 0 0 -0.00

```

```

0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 -4 1 0 1 0 -0.00
0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 -4 1 0 1 -0.00
0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 -2 0 0 0.00
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 -4 1 -0.00
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 -2 0.00

```

LU-solution:

```

x[1] = 2.12684e-01
x[2] = 4.42430e-01
x[3] = 7.03894e-01
x[4] = 2.04154e-01
x[5] = 4.26570e-01
x[6] = 6.86573e-01
x[7] = 1.77360e-01
x[8] = 3.73124e-01
x[9] = 6.15829e-01
x[10] = 1.32162e-01
x[11] = 2.72737e-01
x[12] = 4.03619e-01
x[13] = 4.41643e-01
x[14] = 7.85515e-02
x[15] = 1.82044e-01
x[16] = 2.84265e-01
x[17] = 3.62954e-01
x[18] = 9.26218e-02
x[19] = 1.88443e-01

```

## 18 MINIMOINTI- LISÄYKSIÄ

Paraabeliapproksimointi on joskus hyödyllinen etsittäessä yhden muuttujan funktion minimiä, sillä paraabelin minimi on helppo löytää. Taylorin kaavan nojalla saadaan

$$f(x+h) \approx f(x) + f'(x)h + \frac{1}{2}f''(x)h^2.$$

Minimikohdassa  $h = -f'(x)/f''(x)$  kuten todetaan derivoimalla  $h$ :n suhteen. Algm on sama kuin Newtonin menetelmä sovellettuna  $f'$ :uun.

---

### Newton's method for 1D minimization

---

```
 $x_0 =$  initial guess  
for  $k = 0, 1, 2, \dots$   
     $x_{k+1} = x_k - f'(x_k)/f''(x_k)$   
end
```

---

Moniulotteisessa tapauksessa lähtökohtana on

$$f(\mathbf{x} + \mathbf{s}) \approx f(\mathbf{x}) + \nabla f(\mathbf{x})^T \mathbf{s} + \frac{1}{2} \mathbf{s}^T \mathbf{H}_f(\mathbf{x}) \mathbf{s},$$

missä  $\mathbf{H}_f(\mathbf{x})$  on toisten osittaisderivaattojen muodostama Hessen matriisi  $\mathbf{H}_f(\mathbf{x})_{i,j} = \partial^2 f(\mathbf{x}) / \partial x_i \partial x_j$ . Tämä kvadraattinen funktio saa minimiarvon kun

$$\mathbf{H}_f(\mathbf{x}) \mathbf{s} = -\nabla f(\mathbf{x})^T \mathbf{s}.$$

Tämä tarkastelu johtaa seuraavaan algoritmiin.

---

**Newton's method for minimization**

---

$\mathbf{x}_0$  = initial guess  
for  $k = 0, 1, 2, \dots$   
    Solve  $\mathbf{H}_f(\mathbf{x}_k)\mathbf{s}_k = -\nabla f(\mathbf{x}_k)$  for  $\mathbf{s}_k$    { Compute Newton step }  
     $\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{s}_k$                            { update solution }  
end

---

---

**Conjugate gradient algorithm**

---

$\mathbf{x}_0$  = initial guess  
 $\mathbf{g}_0 = \nabla f(\mathbf{x}_0)$   
 $\mathbf{s}_0 = -\mathbf{g}_0$   
for  $k = 0, 1, 2, \dots$   
    Choose  $\alpha_k$  to minimize  $f(\mathbf{x}_k + \alpha_k \mathbf{s}_k)$    { perform line search }  
     $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{s}_k$                            { update solution }  
     $\mathbf{g}_{k+1} = \nabla f(\mathbf{x}_{k+1})$                            { compute new gradient }  
     $\beta_{k+1} = (\mathbf{g}_{k+1}^T \mathbf{g}_{k+1}) / (\mathbf{g}_k^T \mathbf{g}_k)$   
     $\mathbf{s}_{k+1} = -\mathbf{g}_{k+1} + \beta_{k+1} \mathbf{s}_k$                    { modify gradient }  
end

---

The formula for  $\beta_{k+1}$  in this algorithm is due to Fletcher and Reeves. An alternative formula due to Polak and Ribiere suggests the choice

$$\beta_{k+1} = ((\mathbf{g}_{k+1} - \mathbf{g}_k)^T \mathbf{g}_{k+1}) / (\mathbf{g}_k^T \mathbf{g}_k),$$

which sometimes is better.

Newtonin menetelmään pohjautuva minimointi on toteutettu esimerkkiohjelmassa mynewmin.cpp ja konjugaattigradienttimenetelmä ohjelmassa mycgrad2.cpp. Newtonin algm käyttää osanaan numeerisen Hessen matriisin laskentaa, ks. mynumhes3.cpp.