Fysiikan numeeriset menetelmät
FYSP120

Vesa Apaja
Jyväskylän Yliopisto / Fysiikan laitos
kevät 2012

vesa.apaja@jyu.fi

kurssin kotisivut

https://koppa.jyu.fi/kurssit/116361

Sisältö

- Luennot:
  - 10 x 2 tuntia
- Ohjatut harjoitukset: Janne Kauttonen (janne.kauttonen@jyu.fi)
  - 4 kpl ; 2 ryhmää, 2 tuntia/viikko, n. 5 tehtävää
  - Palautetaan parin viikon kuluessa
  - saa tehdä pareittain
- Tavoitteena oppia:
  - Numeerisen fysiikan perusteita
  - Matlab-ohjelmiston käyttöä (samalla octave'n käyttöä)

Mihin numeerisia menetelmiä tarvitaan?

Kokeellinen fysiikka

- Mittausdatan käsittely
  - Statistinen analyysi
  - Virhearviot
  - Vertailu teorioihin
  - Tietokannat, lajittelu
- Mittalaitteistojen suunnittelu
  - Tulosten simulointi esim. LHC testit

Teoreettinen fysiikka

- Mekaniikka ja termodynamiikka on yksinkertaista vain leluesimerkeissä
- Kvanttimekaniikka on differentiaaliyhtälöiden ratkomista, simulointia ja matriisien diagonalisointia

Mikä työvälineeksi?

- Monitaitoiset työhevoset; usein mukana integroitu grafiikka
  - Matlab (numeriikkaa) tai octave (ilmainen, matkii Matlabia)
  - Mathematica (symbolista laskentaa ja numeriikkaa)
  - root (ilmainen, CERNin C++ paketti)
  - Python (ilmainen, numpy = numerical python, visual python ...)
- Valitse ohjelmointikieli ja käytä kirjastoja
  - GSL (Gnu Scientific Library, C-kielinen)
  - lapack ja blas kirjastot ym. (www.netlib.org, kirjastojen esikuva)
  - Numerical recipes (fortran, C, C++)
  - Boost (C++ kirjasto)
  - NAG (fortran kääntäjä *nagfor* ja kirjasto - laitoslisenssi!)
  - Kääntäjävalmistajien omat kirjastot: optimoitua tehoa
    - ACML (*AMD Core Math Library*)
    - MKL (*Intel Math Kernel Library*)

- Käännettäviä kieliä; rajat eivät tule heti vastaan
  - C
    - nopeaa numeriikkaa
    - sallii tehdä melkein mitä vain – myös outoja virheitä
  - C++
    - Rakenteellisempi kuin C
    - Template library - helpottaa työtäsi
    - Root sisältää C++ tulkin
  - Fortran
    - Numeronmurskaukseen (ja vain siihen)
  - Uudempia kieliä: Java, C# ... ovat ottaneet opikseen ”vanhojen kielten” virheistä

Ilmaiset kääntäjät ovat jo melko hyviä

Kannattaa opetella joku kieli *hyvin*.

Kiusallisia tilanteita numeriikassa

- Laskentamenetelmän tarkkuus on 0.1 eV ja tuloksen tulkinta perustuu 0.01 eV:n energiaeroon - tulkinta on luultavasti pötyä
- Saan Masan koneella eri tuloksen kuin omallani – ohjelmassa on luultavasti numeerinen ongelma, ja se satunnaisesti tuottaa puppua.
- Sain 16 prosessorilla eri tuloksen kuin neljällä – rinnakkaistus on ohjelmoitu väärin
- Pomo käskee käyttää yleisesti jaossa olevaa ohjelmistopakettia, enkä ymmärrä siitä mitään. Lasken sillä luultavasti väärin.
- GIGO (garbage in garbage out) : syöttötiedot väärät

Aina tulee ulos numeroita – mutta voiko niihin luottaa?

Ulos tulee *paljon* numeroita – mikä on oleellista ?

Laskentatarkkuus ja konvergenssi

- Millä luvulla *tarkkuus* koneen mielestä *1+ tarkkuus = 1*
  - Yksinkertainen tarkkuus (single precision): noin 10^{-7}
  - Kaksinkertainen tarkkuus (double precision): noin 10^{-16}
- Älä vertaa kahden reaaliluvun yhtäsuuruutta ilman toleranssia
  - Ei : jos x=y niin tee jotain
    ongelma: tämä tekee satunnaisesti jotain!
    syy: kaksinkertaisella tarkkuudella
    1.1111111111111111 ei ole 1.1111111111111112
  - Vaan : jos $|x-y|<10^{-10}$ niin tee jotain
- Kokonaislukuja on turvallista verrata
- Laskenta on lopetettava jollain konvergenssikriteerillä
  Jos raja on liian tiukka *(”lopeta, jos tulos muuttui edellisestä alle 10^{-16} ”)* ei lasku kenties valmistu koskaan

Yksiköt

- Kannattaa valita laskettavan systeemin yksiköt niin, että tarvittavat numerot ovat suuruusluokkaa 1 – tietokoneen numeroiden esitystapa on parhaimmillaan
- Ole tarkkana, yksikkömuunnokset voivat olla paha juttu
  - Kanada 1984: Boeing 767 joutui pakkolaskuun polttoaineen käytyä vähiin kesken lennon
  - USA/Mars 1999: Mars Climate Orbiter tuhoutui Marsin ilmakehässä
  - USA 1999: Potilas sai 0.5 grammaa fenobarbitalia (piti olla 0.5 *grainia (grain = 0.065 grammaa*))

Algoritmeja

Mikään laskentatehon lisäys ei korvaa hyvää algoritmia.

- FFT (Fast Fourier Transform; Cooley & Tukey)
  N datapistettä: $O(N \log_2(N))$ (usein sanotaan vain ”N log N”) vs. suoraviivainen muunnos $O(N^2)$
  Ero on huikea!
  Esim. $N=10^4$, $N \log_2(N) = 132877$ *vs.* $N^2 = 10^8$
  millisekunteina: FFT 133 sekuntia , suoraviivainen 27 tuntia
  (osta 6 kertaa nopeampi kone: suhde on 22 sekuntia vs. vajaat 5 tuntia)

- Matriisitulon laskeminen
  Gaussin menetelmä $O(N^3)$ Strassen 1969 $O(N^{2.81})$ mainio!
  (Coppersmith-winograd *1990* $O(N^{2.38})$ on useimmille liian mutkikas)

- RSA salaus

- Satunnaislukualgoritmit
  Outo idea, panna nyt deterministinen kone tuottamaan satunnaisuutta
  Eivät tuota todella satunnaisia lukuja, vaan pseudosatunnaislukuja
  Riski on aina olemassa, joskus ei-satunnaisuus kasautuu ja tuottaa vääriä tuloksia

- Simplex algoritmi: funktion minimointi tai optimointi reunaehdoilla
  2D ongelmaan 3-verteksinen (”3-jalkainen”) simplex (kolmio), 3D ongelmaan 4-jalkainen simplex (tetraedri) jne.; huonoa jalkaa liikutetaan parempaan paikkaan.

- Peruskysymys on aina *kauanko suoritus kestää verrattuna systeemin kokoon/dimensioon?*
  Polymoninen aika (kuten N^3) vai eksponentiaalinen aika (kuten $e^{0.34N}$)
  Jälkimmäisessä tapauksessa suurta ongelmaa ei voi ratkaista järkevässä ajassa.

Datan approksimointi

- Approksimointia tarvitaan aina, kun ei haluta käsitellä suurta ja mahdollisesti häiriöistä alkuperäistä dataa, vaan helpommin siirrettävää ja laskettavaa esitystä datasta.
  Esim. funktion sovittaminen mittausdataan.

- Selvitä miltä data näyttää saadaksesi mielikuvan tehtävän luonteesta. Käytä esim. plot, loglog, semilogx, semilogy kuvia apuna.

- Valitse miten mittaat approksimoinnin hyvyyttä – valitse ***normi***
  Lue: NMK sivu 76

  - Esim. funktio kuvaa dataa hyvin jos neliösumma on pieni – pienimmän neliösumman sovitus (pns)
  - Esim. Milloin kaksi funktiota ovat (melkein) samat tai onko yksi funktio lähempänä vertailufunktiota kuin toinen ?

$$||f||_{L_2} = ||f||_2 = \left[\int_\Omega [f(x)]^2 dx\right]^{1/2}$$ L_2 normi

Datan approksimointi: pienimmän neliösumman sovitus dataan

- Valitaan joukko kantafunktioita (*basis functions*) $\{\phi_1(x), \phi_2(x), ..., \phi_n(x)\}$

  Erikoistapaus: kanta on ortonormaali, jos $\int dx\ \phi_i(x)\phi_j(x) = \delta_{i,j}$

- Olkoot tunnetut datapisteet $y_1, y_2, ..., y_N$
- Haluamme löytää luvut c_i, joille

neliösumma $\sum_{i=1}^{N}[f(x_i) - y_i]^2$ on pienin, missä $f(x) = \sum_{j=1}^{n} c_j\phi_j(x)$

Minimoidaan neliösumma ***g(c)***

$$g(\mathbf{c}) = \sum_{i=1}^{N}\left|\sum_{j=1}^{n} c_j\phi_j(x_i) - y_i\right|^2 = \left[\sum_{j=1}^{n} c_j\phi_j(\mathbf{x}) - \mathbf{y}\right]\cdot\left[\sum_{j=1}^{n} c_j\phi_j(\mathbf{x}) - \mathbf{y}\right]$$

$$\frac{\partial g(\mathbf{c})}{\partial c_k} = 2\phi_k(\mathbf{x})\cdot\left[\sum_{j=1}^{n} c_j\phi_j(\mathbf{x}) - \mathbf{y}\right] = 2\phi_k(\mathbf{x})\cdot[f(\mathbf{x}) - \mathbf{y}] = 0$$

Eli: virhe ***f(x)***-***y*** on ortogonaalinen kaikkien kantafunktioiden kanssa

Kirjoitetaan minimiehto yhtälöryhmänä, rivi = *k*, sarake = *j*:

$$\sum_{j=1}^{n}\left[\sum_{i=1}^{N}\phi_k(x_i)\phi_j(x_i)\right]c_j = \sum_{i=1}^{N}\phi_k(x_i)y_i$$

$$\sum_{j=1}^{n}\langle\phi_k|\phi_j\rangle c_j = \langle\phi_k|y\rangle$$

Jos tunnettuja pisteitä on ääretön määrä korvataan summa integraalilla:

$$\langle\phi_k|\phi_j\rangle = \int dx\phi_k(x)\phi_j(x)$$

Erikoistapaus: kanta on ortonormaali, jos $\langle\phi_i|\phi_j\rangle = \delta_{ij}$ ja tällöin ratkaisu on suoraan $c_k = \langle\phi_k|y\rangle$

Pienimmän neliösumman menetelmä matriisien ja vektorien avulla

$A_{ij} = \phi_j(x_i)$ kantafunktiot tunnetuissa pisteissä x_i *(design matrix)*

$\mathbf{c} = \begin{pmatrix} c_1 & c_2 & \dots & c_n \end{pmatrix}^T$ tuntemattomat kertoimet

$\mathbf{y} = \begin{pmatrix} y_1 & y_2 & \dots & y_N \end{pmatrix}^T$ tunnetut arvot

Kirjoitetaan haluttu ehto ilman optimointimatikkaa →

$A\mathbf{c} \approx \mathbf{y}$ haluttu sovitus $(A\mathbf{c})_i = \sum_j \phi_j(x_i) c_j = f(x_i)$

$(N \times n)(n, 1) \approx (N, 1)$ sovitusyhtälön matriisien koot

!!!

Esim. suoran sovitus 100 pisteen joukkoon: A on (100 x 2) matriisi

Matlab ratkaisee ongelman näin : **c= A\y** *Backslash* \ *"left matrix divide"*

- *N=n* voi olla yksikäsitteinen ratkaisu (riippuu kantafunktioista)
- *n>N* enemmän tuntemattomia kuin yhtälöitä (alimääräytynyt)
- *n<N* enemmän yhtälöitä kuin tuntemattomia (ylimääräytynyt)

Matlabin kenoviivaoperaatio **A\y** hakee:

- yhtälöryhmän ratkaisun jos *n=N* ja yhtälöt riippumattomia
- Pienimmän neliösumman sovituksen jos *n<N tai n>N*

P.S. Sama operaatio löytyy octavesta

Osoitetaan, että aiemmin johdettu pienimmän neliösumman yhtälö on sama kuin matriisiyhtälö $A^TAc=A^Ty$

Pienimmän neliösumman minimille saatiin ehto

$$\sum_{j=1}^{n}\left[\sum_{i=1}^{N}\phi_k(x_i)\phi_j(x_i)\right]c_j=\sum_{i=1}^{N}\phi_k(x_i)y_i$$

määritelmä: $A_{ij}=\phi_j(x_i)$

$$\Leftrightarrow\sum_{j=1}^{n}\left[\sum_{i=1}^{N}A_{ik}A_{ij}\right]c_j=\sum_{i=1}^{N}A_{ik}y_i$$

transponointi: $(A^T)_{ij}=A_{ji}$

$$\Leftrightarrow A^TA\mathbf{c}=A^T\mathbf{y}$$

---

kantafunktioista

Yksi yleinen tapa saada kantafunktiojoukko on ratkaista ominaisarvoyhtälö $M\boldsymbol{x}=\lambda\boldsymbol{x}$, vektorit $\boldsymbol{x}$ eri ominaisarvoille λ ovat sopivia kantafunktioiksi (matriisin M valinta riippuu ongelmasta)

Esim. Legendren polynomit $P_L(x)$, Hermiten polynomit $H_k(x)$, trigonometriset funktiot

Jos kantafunktiojoukko on *täydellinen*, voi minkä hyvänsä funktion esittää k.o. kannassa:

$$\sum_i|\phi_i\rangle\langle\phi_i|=1$$

$$\Rightarrow|\psi\rangle=1|\psi\rangle=\sum_i|\phi_i\rangle\langle\phi_i|\psi\rangle=\sum_i\langle\phi_i|\psi\rangle|\phi_i\rangle=\sum_i c_i|\phi_i\rangle$$

Esim. pienimmän neliösumman sovitus suoraan

```
% ------------------------------------
% linear_fit.m
% Linear least squares fit to data
% ------------------------------------
clear all ; clf;
% known data points; column vectors
N=5;
x=linspace(2,7,N)';
y=[2.1,6.8,12.2,17.4,21.6]';
% basis functions {1,x}, so n=2,linear fit
% (N,n) = (5,2) = design matrix
A= [ones(N,1),x];
% least square fit coefficients c
c=A\y;                                  tässä tehdään sovitus
%define inline function f(x)
% f=@(x) c(1)+c(2)*x; % coefficient one by one
f=@(x)  [ones(N,1),x]*c;
plot(x,y,'rx',x,f(x));hold on;
title 'linear fit';xlabel 'x';ylabel 'y';
% same problem using built-in polyfit
cpoly=polyfit(x,y,1);   % fit to 1st order polynomial (=line)
ypoly=polyval(cpoly,x); % evaluate polynomial at points x
plot(x,ypoly,'go');
```

Matlabin inline-funktiot:

f(x)=sin(x) määritellään
joko

```
f=@(x) sin(x)
```

tai

```
f = inline('sin(x)','x')
```

Esim.
f(x,a) = acos(x)+(1-a)sin(x)
joko

```
f = @(x,a) a*cos(x)+(1-a)*sin(x)
```

tai

```
f = inline('a*cos(x)+(1-a)*sin(x)','x','a')
```

Spline interpolointi (Matlab: spline ja ppval)

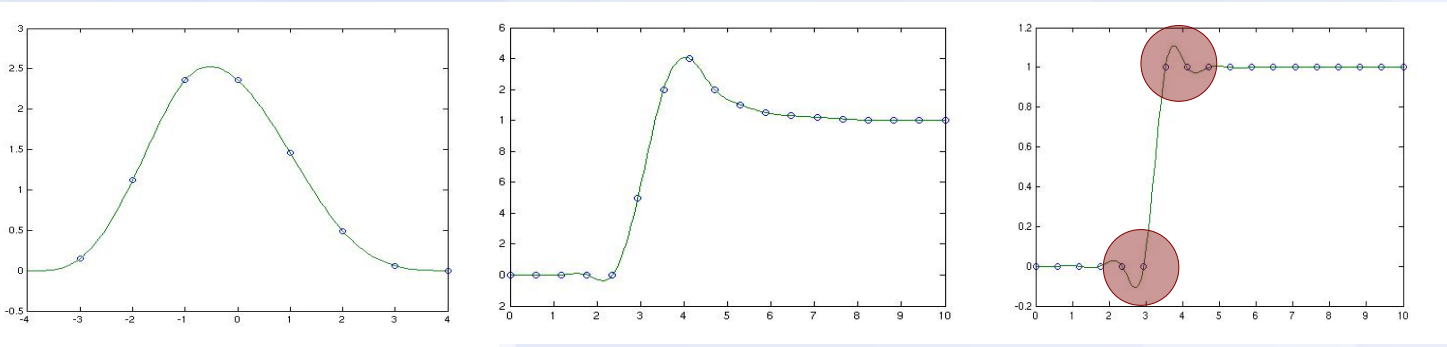
Cubic spline interpolointi: sovitetaan aina kolmeen peräkkäiseen datapisteeseen *(solmupisteet, knots)* kolmannen asteen polynomi niin, ettei datapisteiden kohdalle tule kulmaa.

```
% ---------------------------------------
% cubic spline interpolation of data
% ---------------------------------------
% knots
x = -4:4; y = [0 .15 1.12 2.36 2.36 1.46 .49 .06 0];
% spline coefficients
c = spline(x,[0 y 0]);
% dense set of x-points
xx = linspace(-4,4,101);
plot(x,y,'o',xx,ppval(c,xx),'-');
```

lasketaan spline-kertoimet c
määritellään lisäksi, että päätepisteissä funktion derivaatta on nolla

lasketaan funktion arvoja pisteissä xx spline-kertoimilla c

Huom: spline-kertoimet lasketaan vain kerran, funktion arvoja voidaan laskea mielivaltainen määrä



Tiukat mutkat tuottavat 3. asteen polynomeille vaikeuksia, seurauksena on ylilyöntejä

Spline interpolointi
- miten syntyy kuutiollinen spline (cubic spline)

- n+1 pistettä (x_i,y_i), n väliä, kullakin välillä 3. asteen polynomi, jotka yhdistyvät sileästi toisiinsa
- Ratkaistaan välin [x_i,x_{i+1}] polynom $S_i(x) = a_i(x - x_i)^3 + b_i(x - x_i)^2 + c_i(x - x_i) + d_i$
- 1. ehto: spline menee solmupisteiden kautta $S_i(x_i) = y_i \; ; \quad S_i(x_{i+1}) = y_{i+1}$
- 2. ehto: 1. ja 2. derivaatat ovat jatkuvia $S'_{i-1](x_i)=S'_i(x_i)} \; ; \; S''_{i-1](x_i)=S''_i(x_i)}$

  - tuntemattomia on 4 per polynomi, polynomeja n, kaikkiaan 4n tuntematonta
  - jatkuvuusehto antaa 2n ehtoa
  - derivaattojen jatkuvuusehto antaa 2(n-1) ehtoa

  Tähän mennessä 4n-2 ehtoa

- Puuttuvat 2 ehtoa saadaan sopimalla päätepisteiden 2. derivaatat
  - Luonnollinen spline *(natural spline)* $S''(x_0) = 0 \; , \quad S''(x_n) = 0$
  - Tunnetut/arvatut derivaatat *(”clamped spline”)* $S''(x_0) = f''(x_0) \; , \quad S''(x_n) = f''(x_n)$
- Ratkaistavaksi jää **lineaarinen yhtälöryhmä**, josta saadaan polynomien kertoimet
- Huom:
  - 3. asteen polynomista ei ole järkevää yrittää laskea 3. derivaattaa, se ei ole sileä
  - Älä ekstrapoloi; pisteen x_n jälkeen polynomi on viimeisen tunnetun välin 3. asteen polynomi ja se menee nopeasti ylös tai alas – kuten x^3

Padé-approksimointi

- Polynomin sovittaminen singulaariseen dataan ei onnistu tyydyttävästi
- Yleistetään Taylorin sarja: kirjoitetaan funktio kahden polynomin osamääränä jolloin $Q_M(x)$:n nollakohdat ovat singulariteetteja

$$f(x) \approx \frac{P_L(x)}{Q_M(x)}$$

- *L* on polynomin *P* aste ja *M* polynomin Q aste, usein Padé-approksimaation tasoa merkitään *[L/M]*, eli *[3/2]* on 3. asteen polynomi jaettuna 2. asteen polynomilla
- *Q(x)*:n vakio voidaan valita; usein *Q(x)=1+...* (saadaan yksikäsitteinen tulos)
- Valitaan polynomien asteet ja lasketaan erotus

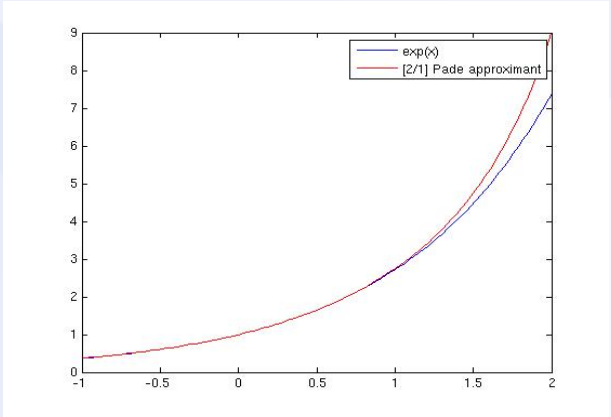
$$E(x) = f(x)Q(x) - P(x)$$

Esim.

$$[2/1] = \frac{a_0 + a_1 x + a_2 x^2}{b_0 + b_1 x} \qquad f(x) = e^x = 1 + x + \frac{1}{2}x^2 + \frac{1}{6}x^3 + ... = f_0 + f_1 x + f_2 x^2 + ...$$

$$\begin{aligned} E(x) &= (f_0 + f_1 x + f_2 x^2 + f_3 x^3 ...)(b_0 + b_1 x) - (a_0 + a_1 x + a_2 x^2) \\ &= [f_0 b_0 - a_0] + [f_0 b_1 + f_1 b_0 - a_1]x + [f_1 b_1 + f_2 b_0 - a_2]x^2 + [f_2 b_1 + f_3 b_0]x^3 + ... \\ &= [b_0 - a_0] + [b_1 + b_0 - a_1]x + [b_1 + b_0 - a_2]x^2 + [b_1 + \frac{1}{6}b_0]x^3 + ... \\ &= [1 - 1] + [b_1 + 1 - a_1]x + [b_1 + -a_2]x^2 + [b_1 + \frac{1}{6}]x^3 + ... \end{aligned}$$

$$\Rightarrow [2/1] = \frac{6 + 4x + x^2}{6 - 2x}$$



singulaarinen kun x=3

- Valitaan polynomien asteet ja lasketaan erotus

- Kehitetään *f(x)* Taylorin sarjaksi halutun pisteen ympärillä
- Asetetaan kunkin *x*:n potenssin kerroin nollaksi, saadaan polynomien *P* ja *Q* kertoimet

Lue: NMK s. 98, esimerkki 4.6.1

Fourier-sarjat ja FFT *(Fast Fourier Transform)*

- Funktion approksimointi Fourier-sarjana

$$f(x) = \frac{a_0}{2} + \sum_{j=1}^{N} a_j \cos(jx) + b_j \sin(jx)$$ kantafunktiot {1,sin(jx),cos(jx)}

- Matlabin **fft** laskee diskreetin Fourier-muunnoksen (DFT)

$$\tilde{f}(k) = \sum_{n=1}^{N} f(n) e^{-i\frac{2\pi(k-1)(n-1)}{N}}$$ kertoimet (Fourier-muunnos) $\tilde{f}(k)$ kompleksilukuja

$$1 \leq k \leq N$$

DFT

Mihin Fourier-muunnos ei sovellu hyvin ?

- Datassa on äärellisen kestoinen signaali

  Esim. hiljaisuuden keskellä puhetta

  - 000000000päläpäläpälä0000000000000000000

    Fourier-muunnos ei sisällytä tietoa siitä missä kohtaa datajonossa signaali on => tehoton kuvaus

- Vaihtoehtoja:

  - Ikkunoitu Fourier-muunnos
  - Wavelet-muunnos (”aaltoset”)

$$s(a,b) = \int dx \psi_{ab}(x) f(x) \qquad \psi_{ab}(x) = \frac{1}{\sqrt{a}} \psi(\frac{x-b}{a}) \qquad \psi(x) \text{ on oskilloiva funktio}$$

perusaallon translaatiota (*b*) ja venyttelyä (*a*) , *b* sisältää tiedon tapahtuman paikasta

---

Cooley, James W.; Tukey, John W. "An algorithm for the machine calculation of complex Fourier series". Math. Comput. 19 (90): 297–301 (1965)

Fourier-sarjat ja FFT: miten FFT toimii ?

$$\tilde{f}(k) = \sum_{n=0}^{N} f(n) e^{i\frac{2\pi k n}{N}} \qquad n = \begin{cases} 2m & n\ parillinen \\ 2m+1 & n\ pariton \end{cases} \qquad 0 \le k \le N-1$$

(Indeksointi in nyt helpoin aloittaa nollasta)

$$\tilde{f}(k) = \sum_{m=0}^{N/2-1} f(2m) e^{i\frac{2\pi k 2m}{N}} + e^{i\frac{2\pi k}{N}} \sum_{m=0}^{N/2-1} f(2m+1) e^{i\frac{2\pi k 2m}{N}} = E_k + e^{i\frac{2\pi k}{N}} O_k$$

Parillisten DFT Parittomien DFT

Danielson ja Lanczos lemma (1942)

Ongelma hajosi kahtia, välissä on kompleksinen vaihekerroin. Tätä voidaan jatkaa, hajotetaan E_k kahtia Parillisiin ja parittomiin alkioihin, samoin O_k. Saadaan

$$E_k + e^{i\cdots} O_k = (EE_k + e^{i\cdots} OE_k) + e^{i\cdots}(EO_k + e^{i\cdots} OO_k)$$

Jatketaan pilkkomista, kunnes jäljellä on vain yksittäisiä alkuperäisen taulukon ***f*** lukuja, vaikkapa

$$EOOOOEOOOEOEEEO_k = f(p) \quad , \text{ jokin alkuperäisen datan indeksi } p$$

Yhden numeron *f(p)* diskreetti Fourier muunnos on luku itse: $\sum_{n=0}^{0} f(p)\, e^{i\frac{2\pi k n}{N}} = f(p)$ $(f(p) \text{ on luku indeksillä } n=0)$

Tästä saisi jo toimivan FFT ohjelman, joka voi helppo toteuttaa käyttäen rekursiota (Wikipediassa on pseudokoodi) – olettaen, että lukuja on 2^p kappaletta, jotta jako parilliset/parittomat menee lopulta tasan.
Edellä oli ”jokin indeksi ***p***”, mutta mikä? Eli monesko alkuperäisen taulukon alkio on esimerkiksi

E	O	O	O	O	E	O	O	O	E	O	E	E	E	O
0	1	1	1	1	0	1	1	1	0	1	0	0	0	1

Luetaan oikealta vasemmalle ja muutetaan desimaaliluvuksi:

$$100010111011110 = 2^{14} + 2^{10} + 2^8 + 2^7 + 2^6 + 2^4 + 2^3 + 2^2 + 2^1 = 17886$$

Tulos: $EOOOOEOOOEOEEEO_k = f(17886)$

Bit Reversal

Diskreetin Fourier muunnoksen laskemiseen tarvitaan siis vain alkuperäisten taulukon alkioiden järjestelyä ja yhdistämistä vaihetekijöillä.

B-spline kantafunktiot

- polynomeja, aste on yhtä pienempi kuin **kertaluku *k***
  k = 4: 3. asteen polynomi, eli muotoa ax^3+bx^2+cx+d
- valitaan sopivat **solmupisteet t_i** (valinta vaikuttaa kantafunktioiden jatkuvuuteen, ks. seuraava sivu)
- positiivisia, summautuvat ykköseen kussakin *x* (normitettu)
- mitä suurempi *k*, sitä korkeammat derivaatat ovat jatkuvia
- helppoja laskea
  B-spline kantafunktiot pisteessä *x* saadaan rekursiokaavasta (deBoor)

$$B_i^k(x) = \frac{x - t_i}{t_{i+k-1} - t_i} B_i^{k-1}(x) + \frac{t_{i+k} - x}{t_{i+k} - t_{i+1}} B_{i+1}^{k-1}(x) \qquad B_1^1(x) = 1, \ \text{ jos } t_i \le x < t_{i+1} \ , \ \text{muulloin } 0$$

- tehokkainta laskea kerralla **kaikki** *k* kappaletta pisteessä *x* vaikuttavaa kantafunktiota
- kussakin pisteessä *x* on vain *k* kappaletta nollasta poikkeavaa B-spline funktiota
  => kehitelmissä on vain k kerrointa per *x* , minimimäärä operaatioita (*minimal support)*

$$f(x) = \sum_{i=1^N} c_i B_i^k(x) = \sum_{i=left}^{left+k-1} c_i B_i^k(x)$$

”vasen indeksi” *left* riippuu x:n arvosta

- Matlabissa on hyvä B-spline funktioita havainnollistava ohjelma **bspligui**

Sanastoa:

breakpoints	x:n arvot, joissa on yksi tai useampi solmupiste
knot points, knot sequence	solmupisteet, *x*:n arvoja kasvavassa järjestyksessä; sama arvo voi toistua
multiplicity	montako kertaa sama solmupiste toistuu listalla

B-spline kantafunktiot

- Solmupisteisiin on helppo tehdä esim. epäjatkuvuus tai epäjatkuva derivaatta
  – lisätään vain solmupisteiden toistoa (*multipilicity)*

Esim: B-spline aste *k=3*

- polynomit 2. astetta
- kullakin *x*:n arvolla on 3 nollasta poikkeavaa B-spline funktiota
- yksinkertaisissa solmupisteissä on 1. ja 2. derivaatta jatkuvia
- esim. solmupisteet ovat $t = (x_1, x_2, x_3, x_3, x_3, x_4, x_5, x_5, x_6..)$

k eli 3 samaa: *f, f', f''*
epäjatkuvia pisteessä x_3

2 samaa: *f' ja f''* epäjatkuvia pisteessä x_5

kuvan solmupisteet: $t = (0, 0, 0, 1, 2, 3, 4, 5, 5, 5)$

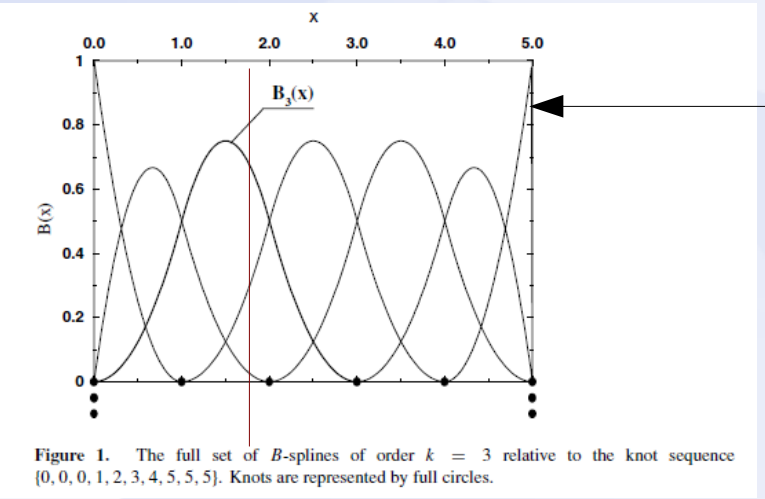


Figure 1. The full set of *B*-splines of order $k = 3$ relative to the knot sequence {0, 0, 0, 1, 2, 3, 4, 5, 5, 5}. Knots are represented by full circles.

päätepisteessä vain yksi B-spline funktio on nollasta poikkeava

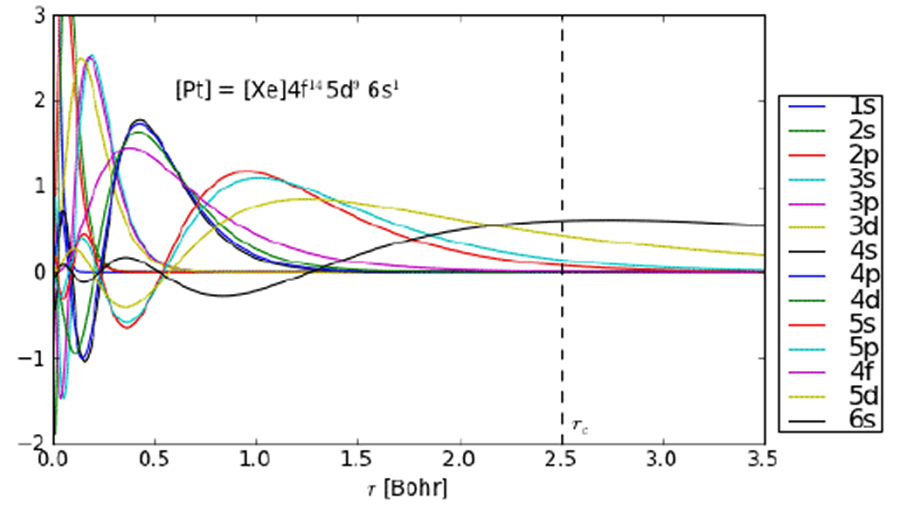
H Bachau[1], E Cormier[1], P Decleva[2], J E Hansen[3] and F Martín[4]
Rep. Prog. Phys. **64** (2001) 1815–1942

Elektronirakennelaskujen kantafunktioista

Atomin ytimen lähellä aaltofunktio oskilloi voimakkaasti, mutta mielenkiinto kohdistuu sidoksiin, joiden alueella aaltofunktio ei muutu nopeasti.

- Ongelma helpottuu, jos oletetaan kuorialue muuttumattomaksi (*frozen core approximation*)
- **Tasoaaltokanta:** suoraviivainen, mutta kuorialueen kuvaamiseen tarvittaisiin hyvin suuri tasoaaltokanta.
    - Käytetään ns. *pseudopotentiaaleja*, joilla saadaan samat olennaiset tulokset kuin koko Coulombin potentiaalilla, mutta joka on slleämpi; tarvitaan vähemmän tasoaaltoja ja laskenta nopeutuu
    - Kullekin atomilajille on generoitava oma pseudopotentiaalinsa
        - Esim. *Vanderbilt ultrasoft pseudopotentials;* useita parametreja, testattava erikseen
    - Tasoaallot kuvaavat hyvin rakenteeltaan toistuvia systeemejä – huonoja kuvaamaan esim. atomiklustereita
- **PAW (Projector Augmented Wave)** -menetelmä: atomikeskisiä ”augmentation” palloja, joiden sisällä aaltofunktiot ovat atomien osa-aaltofunktioita, ulkopuolella ”pehmeitä” – liitetään toisiinsa rajalla
    - Käyttää laskentaa tehostavia, pehmeitä aaltofunkioita, joista voi koota koko *all electron* -aaltofunktion
    - Laskentapisteiden valinta (”*grid*”) vaikuttaa tuloksiin – eikä muutos ole aina samaan suuntaan gridiä muutettaessa
    - Lisätietoja: *"From ultrasoft pseudopotentials to the projector augmented-wave method"*, G. Kresse and D. Joubert, Physical Review B 59, 1758 (1999)

PAW: Platinan kuorialueen tilat



Carsten Rostgaard (2010)

- Vain 5d ja 6s poikkevat nollasta sidosalueella

Bulkki platina: atomien väli 4.27 Bohr (2.23 Å)

Title: **SOFT SELF-CONSISTENT PSEUDOPOTENTIALS IN A GENERALIZED EIGENVALUE FORMALISM**
Author(s): VANDERBILT D
Source: PHYSICAL REVIEW B Volume: **41** Issue: **11** Pages: **7892-7895** DOI: **10.1103/PhysRevB.41.7892** Published: **APR 15 1990**
Times Cited: **9,025** (from Web of Science)

63. Title: **PROJECTOR AUGMENTED-WAVE METHOD**
Author(s): BLOCHL PE
Source: PHYSICAL REVIEW B Volume: **50** Issue: **24** Pages: **17953-17979** DOI: **10.1103/PhysRevB.50.17953** Published: **DEC 15 1994**
Times Cited: **7,414** (from Web of Science)
Links

Lineaarisen yhtälöryhmän ratkaisu: Gaussin eliminointimenetelmä

- Lineaarinen yhtälöryhmä:

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

- Matriisimuodossa

$$\begin{pmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ a_{21} & a_{22} & \ldots & a_{2n} \\ & \ldots & & \\ a_{n1} & a_{n2} & \ldots & a_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \ldots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \ldots \\ b_n \end{pmatrix} \quad \text{eli} \quad Ax = b$$

- Muokkaa kerroinmatriisi yläkolmiomuotoon:
  eliminoi x_1 ensimmäisestä
  yhtälöstä ja sijoita alempiin,
  x_2 toisesta ja sijoita alempiin jne.

$$\begin{pmatrix} c_{11} & c_{12} & c_{13} & \ldots & c_{1n-1} & c_{1n} \\ 0 & c_{22} & c_{23} & \ldots & c_{2n-1} & c_{2n} \\ 0 & 0 & c_{33} & \ldots & c_{3n-1} & c_{3n} \\ & & \ldots & & & \\ 0 & 0 & 0 & \ldots & 0 & c_{nn} \end{pmatrix}$$

Nyt muuttuja x_n voidaan heti ratkaista,
sijoittaa edelliseen yhtälöön, ratkaista x_{n-1}, sijoittaa ylempään, jne. ja lopulta ratkaista muuttuja x_1.

$$x_n = b'_n / c_{nn}$$

Huom! muuttunut yhtälön oikea puoli

Lineaarisen yhtälöryhmän ratkaisu: Gaussin eliminointimenetelmä

gauss_elim.m

```
function [x] = gauss_elim(A,b)
% Solves Ax=b using gaussian elmination
% Returns: vector x

n  = max(size(A));
% Transform A to upper triangle matrix
for j=2:n,
    for i=j:n,
       m = A(i,j-1)/A(j-1,j-1);
       A(i,:) = A(i,:) - A(j-1,:)*m;
       b(i) = b(i) - b(j-1)*m;
    end
end

% Back substitution

x = zeros(n,1);
x(n) = b(n)/A(n,n);

for j=n-1:-1:1,
  x(j) = (b(j)-A(j,j+1:n)*x(j+1:n))/A(j,j);
end
```

!!!

Alkuperäinen A

0.9572	0.1419	0.7922
0.4854	0.4218	0.9595
0.8003	0.9157	0.6557

x_1 ratkaistu 1. yhtälöstä ja sijoitettu 2. yhtälöön

0.9572	0.1419	0.7922
0.0000	0.3498	0.5578
0.8003	0.9157	0.6557

x_1 sijoitettu 3. yhtälöön

0.9572	0.1419	0.7922
0.0000	0.3498	0.5578
0.0000	0.7971	-0.0066

x_2 ratkaistu 2. yhtälöstä ja sijoitettu 3. yhtälöön

0.9572	0.1419	0.7922
0.0000	0.3498	0.5578
0.0000	0.0000	-1.2776

Valmis takaisinsijoitukseen.
Myös yhtälön oikea puoli eli b muuttuu!

Kohta !!! on vaarallinen:
- Jos diagonaalille on nolla tulee jakolaskusta NaN
- Jos diagonaalilla on hyvin pieni luku muihin verrattuna on menetelmä helposti numeerisesti epästabiili

Lineaarisen yhtälöryhmän ratkaisu: Gaussin eliminointimenetelmä

- Pienten diagonaalielementtien ongelma

Pivoting: muokataan matriisia A niin että diagonaalilla on mahdollisimman isoja lukuja ("pivot" = tukialkio, diagonaalialkio)

Mitä saa tehdä:

- Vaakarivejä saa vaihtaa keskenään (yhtälöiden järjestys on yhdentekevä)
- Lisätä vakio kertaa vaakarivi johonkin toiseen vaakariviin
- Vaakarivin saa kertoa nollasta poikkeavalla vakiolla

Partial pivoting: haetaan vaakarivejä vaihtamalla diagonaalille suurin luku
Useimmiten riittävä

Complete pivoting: haetaan koko matriisin suurimmat alkiot diagonaalille vaihtamalla sekä vaaka- että pystyrivejä

Esim: Sama tehtävä kahdessa muodossa

$$\begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$ ratkaisu $$x = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$, mutta ohjelmassa tulee nollalla jakaminen ellei pivoting'ia tehdä

$$\begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$ ratkaisu $$x = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$ ohjelma toimii

- **Yhtälöryhmän ratkaisu suoralla matriisin inversiolla : älä tee.** $Ax = b \Leftrightarrow x = A^{-1}b$
- **Yhtälöryhmän ratkaisu LU-hajotelmalla:** *Neliömatriisin A voi jakaa tuloksi*
  A= LU
  L on alakolmiomatriisi (diagonaalilla ykkösiä)
  U on yläkolmiomatriisi

[L,U] = lu(A) L voi tarvita rivien permutointeja tullakseen alakolmiomatriisiksi
tai
[L,U,P]=lu(A) partial pivoting - P on permutaatiomatriisi; $A=P^{-1}LU$

- Miksi LU-hajotelma?

Ax=b => Ax=LUx=L(Ux)=b , ratkaistaan ensin helppo yhtälöryhmä Ly=b ja sitten toinen helppo yhtälöryhmä Ux=y

Helppoja tehtäviä koska Ly=b ratkeaa takaisinsijoittamalla ylhäältä alaspäin
ja Ux=y ratkeaa takaisinsijoittamalla alhaalta ylöspäin
forward and backward substitution

- Oleellinen etu: kerran saatua LU-hajotelmaa voi käyttää monta kertaa

Esim. Ratkaistava 5000 yhtälöryhmää Ax=b, joissa on sama A, mutta eri b
- Laske ensin A:n LU hajotelma A=LU *vain yhden kerran*
- Ratkaise Ly=b ja Ux=y kullekin vektorille b

- Käänteismatriisin laskeminen (A:n käänteismatriisi on A^{-1} jos $A^{-1}A=I$, I=yksikkömatriisi)
  $A^{-1} = (LU)^{-1} = U^{-1}L^{-1}$ *huomaa järjestys*
- Determinantin laskeminen
  det(A) = det(LU) = det(L) det(U) = det(U) = U:n diagonaalialkioiden tulo – helppo !!

Lineaarisen yhtälöryhmän ratkaisu: Matlab *linsolve*

```
>> help linsolve
 LINSOLVE Solve linear system A*X=B.
    X = LINSOLVE(A,B) solves the linear system A*X=B using
    LU factorization with partial pivoting when A is square,
    and QR factorization with column pivoting otherwise.
    Warning is given if A is ill conditioned for square matrices
    and rank deficient for rectangular matrices.

    [X, R] = LINSOLVE(A,B) suppresses these warnings and returns R
    the reciprocal of the condition number of A for square matrices,
    and the rank of A if A is rectangular.

    X = LINSOLVE(A,B,OPTS) solves the linear system A*X=B,
    with an appropriate solver determined by the properties of
    the matrix A as described by the structure OPTS. The fields of OPTS
    must contain logicals. All field values are defaulted to false.
    No test is performed to verify whether A possesses such properties.
```

QR factorization (QR decomposition) :
qr(A), matriisin hajotelma ortogonaalimatriisin Q ja yläkolmiomatriisin R tuloksi voidaan tehdä muillekin kuin neliömatriiseille

Condition number of A: *cond(A),* matriisin A kuntoluku eli häiriöalttius

A is ill conditioned: matriisin A kuntoluku eli häiriöalttius on suuri – ratkaisu Ax=b on virhealtis

Rank of A: *rank(A)*, matriisin A aste eli ranki eli säännöllisyysaste eli rangi
matriisin A lineaarisesti riippumattomien vaaka tai pystyrivien luku
yhtälöryhmässä niiden yhtälöiden luku jotka tuovat ”uutta tietoa” muuttujasta x
(esim. yhtälöistä 2x+5y=3 ja 4x+10y=6 on toinen turha)

jatkuu...

Below is the list of all possible field names and
their corresponding matrix properties.

Field Name : Matrix Property

LT : Lower Triangular
UT : Upper Triangular
UHESS : Upper Hessenberg
SYM : Real ***Symmetric*** or Complex ***Hermitian***
POSDEF : ***Positive Definite***
RECT : General Rectangular
TRANSA : (Conjugate) Transpose of A

Here is a table containing all possible combinations of options:

LT	UT	UHESS	SYM	POSDEF	RECT	TRANSA
T	F	F	F	F	T/F	T/F
F	T	F	F	F	T/F	T/F
F	F	T	F	F	F	T/F
F	F	F	T	T/F	F	T/F
F	F	F	F	F	T/F	T/F

Symmetric: matriisi A on symmetrinen diagonaalin suhteen eli $a_{ij} = a_{ji,}$ eli $A^T=A$

Positive definite: matriisille A kaikki $x^TAx>0$ jos x ei ole nollavektori; erityisesti kaikki ominaisarvot > 0

Hermitian: Hermiten matriisille A

$$A^\dagger = A$$

eli matriisin hermiten konjugaatti (transponointi ja kompleksikonjugointi) on matriisi itse
Tärkeitä fysiikassa: ominaisarvot ovat reaaliset (myöhemmin puhutaan paljon ominaisarvoista)
Mitattavia suureita vastaa Hermiten operaattori, joka voidaan kirjoittaa kantatilojen avulla
Hermiten matriisiksi.

Kokeile: yhtälöryhmän Ax=b ratkaisua kirjoittamalla **x=A\b** (\ on ”left matrix divide”)

Muita hajotelmia

- LU hajotelma on aika hidas laskea
- $S=LDL^T$ hajotelma symmetriselle matriisille S
  - n. 2x nopeampi kuin LU
  - D on diagonaalimatriisi, L alakolmiomatriisi
  - Jos D:n alkiot ovat >0 on matriisi positiivisesti definiitti
- Cholesky hajotelma $S=LL^T$ symmetriselle, positiivisesti definiitille matriisille
  - n. 2x nopeampi kuin LU
- QR-hajotelma A=QR, missä $Q^{-1}=Q^T$
  - Kätevä yhtälöryhmän ratkaisussa

$$Ax = (QR)x = b \Leftrightarrow Rx = Q^{-1}x = Q^T x$$

Lisää matriisityyppejä

- Tridiagonaalimatriisi: nollasta poikkeavia alkioita vain diagonaalilla ja sen vieressä

$$\begin{pmatrix} 5 & 6 & 0 & 0 & 0 \\ 1 & 3 & 4 & 0 & 0 \\ 0 & 2 & 9 & 6 & 0 \\ 0 & 0 & 4 & 5 & 5 \\ 0 & 0 & 0 & 4 & 7 \end{pmatrix}$$

  - Tulee vastaan esim. diskretisoitaessa toista derivaattaa (ks. seuraava sivu)

$$f''(x_i) \approx \frac{f(x_{i-1}) - 2f(x_i) + f(x_{i+1})}{h^2} \quad \text{eli} \quad \begin{pmatrix} f''(x_1) \\ f''(x_2) \\ f''(x_3) \\ \dots \\ f''(x_n) \end{pmatrix} \approx \frac{1}{h^2} \begin{pmatrix} -2 & 1 & 0 & 0 & 0 & 0 & \dots & 0 \\ 1 & -2 & 1 & 0 & 0 & 0 & \dots & 0 \\ 0 & 1 & -2 & 1 & 0 & 0 & \dots & 0 \\ & & \dots & & & & & \\ 0 & 0 & 0 & 0 & \dots & 0 & 1 & -2 \end{pmatrix} \begin{pmatrix} f(x_1) \\ f(x_2) \\ f(x_3) \\ \dots \\ f(x_n) \end{pmatrix}$$

- Nauhamatriisi *(band matrix)* edellisen laajennus, nollasta poikkeavia alkioita vain symmetrisesti diagonaalin kahta puolta
- Harva matriisi *(sparse matrix)*: paljon nollia
  - Isoille harvoille matriiseille on omat tehokkaat menetelmänsä
    - Pakataan matriisi: kannattaa tallentaa tieto siitä missä nollasta poikkeavia alkioita on ja mitä ne ovat
      - pienempi muistintarve

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

$$\Bigg\} \quad f''(x_i) \approx \frac{f(x_{i-1}) - 2f(x_i) + f(x_{i+1})}{h^2}$$

Iteraatiomenetelmät

- Yhtälöryhmä voi olla niin suuri, ettei sitä kannata ratkaista konetarkkuuteen asti – kestää liian kauan tai muisti loppuu kesken
- Mahdollisesti paljon suoria menetelmiä nopeampia isoissa tehtävissä
- Iteroidaan kohti ratkaisua lähtien pisteestä ***x¹*** kaavalla $\mathbf{x}^{k+1} = \mathbf{x}^k + \alpha_k \mathbf{p}^k$

  menetelmästä riippuva
- Idea: funktion minimi on samalla yhtälöryhmän ratkaisu
  - Jos A on symmetrinen ($A^T=A$), positiividefiniitti ($\mathbf{x}^T A\mathbf{x}>0$, kun $\mathbf{x}$ ei ole 0), niin

$$f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T A\mathbf{x} - \mathbf{b}^T\mathbf{x} + c \qquad \text{minimikohta on gradientin nollakohta} \qquad \nabla f(\mathbf{x}) = A\mathbf{x} - \mathbf{b} = 0$$

Esim. kahden muuttujan tapaus: $\mathbf{x} = \begin{pmatrix} x \\ y \end{pmatrix}$

$$f(x,y) = \frac{1}{2}\begin{pmatrix} x & y \end{pmatrix}\begin{pmatrix} 5 & 6 \\ 6 & 3 \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix} - \begin{pmatrix} 7 & 9 \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix} + 12$$

$$= \frac{1}{2}(5x^2 + 12xy + 3y^2) - 7x - 9y + 12$$

$$\nabla f(x,y) = (5x + 6y - 7, 3y + 6x - 9)$$

$$A\mathbf{x} - \mathbf{b} = \begin{pmatrix} 5 & 6 \\ 6 & 3 \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix} - \begin{pmatrix} 7 \\ 9 \end{pmatrix} = \nabla f(x,y)$$

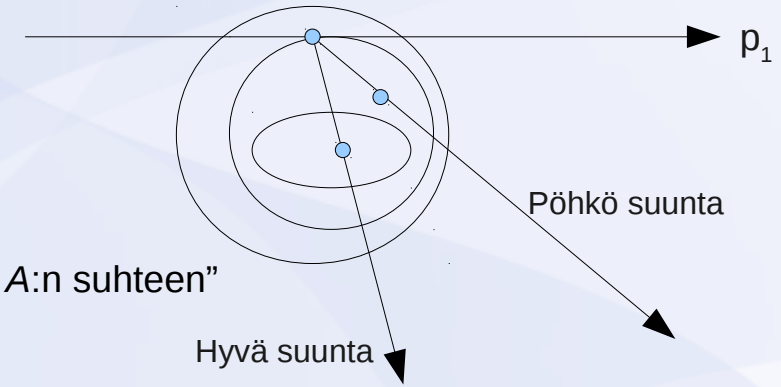
minimi

$$(x,y) = (-1.5714, 0.1429)$$

Kysymys: Mitä tapahtuu jos A ei ole symmetrinen. Miksi A:n pitää olla positiividefiniitti ?

Liittogradienttimenetelmä *(conjugate gradient method)*

- Tunnetuimmat versiot Fletcher-Reeves ja Polak-Ribiere
- Paljon käytetty, tehokas optimointimenetelmä
- Voidaan käyttää myös yhtälöryhmien iteratiiviseen ratkaisuun
- Valitaan seuraava optimointisuunta siten, ettei edellisiä optimointeja pilata
  - Jos on jo löydetty minimi suuntaan $\mathbf{p}_1$, ei kannata valita seuraavaa suuntaa niin, että tämä työ hukataan



- Hyvin valitut suunnat $\{\mathbf{p}_i\}$ ovat ”konjugoituja A:n suhteen”

$$\langle \mathbf{p}^i | A\mathbf{p}^j \rangle = 0; i \neq j \qquad (*)$$

Kokeile: lookfor 'conjugate gradient'
Matlabissa ja octavessa on menetelmät pcg, bicg, cgs jne.

(*) Pistetulo kvanttimekaniikan bra-ket merkinnöillä, tarkoittaa $\mathbf{p}^i \cdot (A\mathbf{p}^j)$

Liittogradienttimenetelmä

Funktio ratkaisee yhtälön $A\mathbf{x}=\mathbf{b}$

```
function [x] = conj_grad(A,b,x)
%  Conjugate gradient steps starting from input point x
%  Returns: vector x, the point of convergence

acc  = 10^-13;   % accuracy goal
% initial values
r    = b - A*x; % residual
p    = r;
k    = 0;
% iteration till convergence
dot  = r'*r;
while dot > acc,
    pp      = A*p;          % A used only here
    alpha   = dot/(p'*pp);
    x       = x + alpha*p;
    r       = r - alpha*pp;
    dot2    = r'*r;
    beta    = dot2/dot;
    p       = r + beta*p;
    dot     = dot2;
    k       = k + 1;
end
disp(['converged after ' num2str(k) ' steps']);
```

Algoritmi: NMK sivu 57-58
ja Wikipedia

$$
\begin{aligned}
&\mathbf{r}^0 = \mathbf{b} - A\mathbf{x}^0 \\
&\mathbf{p}^0 = \mathbf{r}^0 \\
&k = 0 \\
&\textbf{do} \\
&\quad \alpha_k = \frac{\langle \mathbf{r}^k, \mathbf{r}^k \rangle}{\langle \mathbf{p}^k, A\mathbf{p}^k \rangle} \\
&\quad \mathbf{x}^{k+1} = \mathbf{x}^k + \alpha_k \mathbf{p}^k \\
&\quad \mathbf{r}^{k+1} = \mathbf{r}^k - \alpha_k A\mathbf{p}^k \\
&\quad \beta_k = \frac{\langle \mathbf{r}^{k+1}, \mathbf{r}^{k+1} \rangle}{\langle \mathbf{r}^k, \mathbf{r}^k \rangle} \\
&\quad \mathbf{p}^{k+1} = \mathbf{r}^{k+1} + \beta_k \mathbf{p}^k \\
&\quad k = k + 1 \\
&\textbf{until } \|\mathbf{r}^{k+1}\| < \epsilon
\end{aligned}
$$

- Vain vektori $A\mathbf{x}$ tarvitaan, ei koko matriisia A
- Algoritmi ottaa konjugaattisuunnan vain edelliseen suuntaan nähden ja käyttää sitä sekä edellistä residuaalia seuraavan suunnan laskemiseen; usein ei tarvita kaikkia kaikki konj. suuntia.

Numeerinen integrointi

- Integraali esitetään summan raja-arvona (Riemannin integraali)

$$\int_a^b dx f(x) = \lim_{N\to\infty} \sum_{i=1}^{N} h f(x_i) = \lim_{N\to\infty} h \sum_{i=1}^{N} f(x_i) \ , \quad h = \frac{b-a}{N}$$

jos solmupisteet x_i valitaan tasavälein ja kunkin painoarvo summassa on sama 1.

- Äärellisessä määrässä pisteitä *N* saadaan tarkempi tulos jos solmupisteet x_i ja niiden painot w_i valitaan sopivasti, eli lasketaan summa

$$\int_a^b dx f(x) \approx \sum_{i=1}^{N} w_i f(x_i)$$ **kvadratuuri**

Funktiosta *f(x)* riippuu **miten pisteet ja painot kannattaa valita**.

Esim.

Riemannin summa
Helppo koodata, hyvin epätarkka.

$$\int_a^b dx f(x) \approx h \sum_{i=1}^{N} f(x_i)$$ åårellinen N

- ei integroi tarkasti mitään muuta kuin vakiofunktion

Newton-Cotes kvadratuurit

Integroitava tarkasti korkeintaan tiettyä astetta oleva polynomi

1. asteen polynomi - trapezoidisääntö painot avoimella 3.pisteen välillä h/2 *(1,2,1)

h/2 *

x1	x2	x3	x4	x5	x6	x7
1	2	1				
		1	2	1		
				1	2	1
1	2	2	2	2	2	1

← lopulliset painot w_i

esitetään usein muodossa $\int_a^b dx f(x) = \frac{h}{2}(f_1 + 2f_2 + f_3)$

kelpaa sekä parittomille että parillisille määrälle pisteitä

2. ja 3. (!) asteen polynomi - Simpsonin sääntö painot h/3*(1,4,1)

h/3*

x1	x2	x3	x4	x5	x6	x7
1	4	1				
		1	4	1		
				1	4	1
1	4	2	4	2	4	1

← lopulliset painot w_i

$$\int_a^b dx f(x) = \frac{h}{3}(f_1 + 4f_2 + f_3)$$

Lisää simpsonin säännöstä:

- päätepisteet painolla 1, välipisteet painoilla 4,2,4,...,2,4

sarjan 4,2,4 tuottaminen silmukassa:

```
% h=x_2-x_1
% tulos = f(x_1)+f(x_N))
w = 2
for i=2:N-1
        w = 6-w
        % tulos = tulos +w*f(x_i)
 end
% tulos = h*tulos/3
```

Varoitus: pisteitä **pitää** olla pariton määrä !
(muuten painot ovat 1,4,2,4,....,4,2,1

väärä paino

3. asteen polynomi - Simpsonin 3/8 sääntö painot 3h/8*(1,3,3,1)

$$\int_a^b dx f(x) = \frac{3h}{8}(f_1 + 3f_2 + 3f_3 + f_4)$$

4. asteen polynomi - Boden sääntö 2h/45*(7,32,12,32,7)

X1	X2	X3	X4	X5	X6	X7	X8	X9	X10	X11	X12	X13
7	32	12	32	7								
				7	32	12	32	7				
								7	12	32	12	7
7	32	12	32	14	32	12	32	14	12	32	12	7

$$\int_a^b dx f(x) = \frac{2h}{45}(7f_1 + 32f_2 + 12f_3 + 32f_4 + 7f_5)$$

Pulmia integroitaessa n. asteen polynomeja kun n on suuri:
• painokertoimet voivat olla suuria ja erimerkkisiä, summaus tulee herkäksi virheille

Spline-integrointi

jos datan arviointiin on käytetty esim. kuutiollista splinea, voidaan tämä paloittainen 3.asteen polynomi tietenkin myös integroida (ja derivoida)

Newton-Cotes kvadratuurien johtamisesta

Esitietona tarvitaan
Lagrangen interpolaatiopolynomi *L(x)* ja Lagrangen (kanta)polynomit $l_i(x)$:

n+1 tunnettua pisteet (x_i,y_i), pisteiden kautta kulkee n. asteen polynomi

$$L_n(x) = \sum_{j=0}^{n} y_j \prod_{i=1;i\neq j}^{n+1} \frac{x - x_i}{x_j - x_i} = \sum_{j=0}^{n} y_j l_j(x)$$

- polynomi $L_n(x)$ menee pisteen x_k kautta, koska tulo menee nollaksi ellei x_k ole tulosta puuttuva arvo x_j => $j=k$. Summaan jää siis vain k.s termi:

$$L_n(x_k) = \sum_{j=1}^{n+1} y_j \prod_{i=1;i\neq j}^{n+1} \frac{x_k - x_i}{x_j - x_i} = \sum_{j=1}^{n+1} y_j \delta_{jk} = y_k$$

Lagrangen interpolaatiopolynomi on yksikäsitteinen, pienintä mahdollista astetta oleva pisteiden (x_i,y_i) kautta kulkeva polynomi

- n. asteen polynomin kertoimet määrää n+1 pistettä
  - jos halutaan integroida n. asteen polynomia tarvitaanko tarkkaan tulokseen n+1 pistettä ? Ei !
- kvadratuuri laskee jopa 2n+1 pisteen polynomin n+1 pisteellä

Adaptiiviset integrointiohjelmat

- Halutaan tihentää pistejoukkoa kunnes haluttu tarkkuus saavutetaan
- Ei haluta laskea funktiota samoissa pisteissä yhä uudelleen

Esim adaptiivinen trapezoidisääntö h/2*(1,2,1)

1. lasku 1 2 1
lisäpisteet 2 2
lisäpisteet 2 2 2 2
... 2 2 2 2 2 2 2 2

jatketaan kunnes integraali muuttuu alle halutun tarkkuuden verran askeleesta toiseen

Jokainen uusi tihennys tuo vain uusia pisteitä, painot aina 2

Matlab: quad adaptiivinen Simpsonin säännön integrointi

$$\int_0^2 \frac{1}{x^3 - 2x - 5} = -0.460502$$

```
f = @(x)1./(x.^3-2*x-5);
quad(f,0,2)
```

Numeerinen integrointi - Gaussin kvadratuurit

Edellä Newton-Cotes kvadratuureissa haettiin vain hyviä painoja w_i
Gaussin kvadratuureissa haetaan **hyvät painot w_i *ja* hyvät pisteet x_i**

Esim Halutaan mahdollisimman tarkka tulos mahdollisimman korkea-asteisen polynomin integraalille käyttäen vain kahta pistettä, eli

$$\int_{-1}^{1} dx f(x) \approx w_1 f(x_1) + w_2 f(x_2)$$ (väliksi voi aina tehdä [-1,1] muuttujanvaihdolla)

4 tuntematonta, tarvitaan 4 ehtoa eli asetetaan x:n potenssit 0,1,2,3 integroitumaan tarkasti

$$x^0 : w_1 + w_2 = 2$$
$$x^1 : w_1 x_1 + w_2 x_2 = 0$$
$$x^2 : w_1 x_1^2 + w_2 x_2^2 = \frac{2}{3}$$
$$x^3 : w_1 x_1^3 + w_2 x_2^3 = 0$$

ratkaisu: $$w_1 = w_2 = 1 \ , \quad x_1 = -\frac{1}{\sqrt{3}} \quad , \quad x_2 = \frac{1}{\sqrt{3}}$$

eli 2 pisteen Gaussin kvadratuuri on $$\int_{-1}^{1} dx f(x) \approx f(-\frac{1}{\sqrt{3}}) + f(\frac{1}{\sqrt{3}})$$

samoin saadaan 3 pisteen Gaussin kvadratuuri, $$\int_{-1}^{1} dx f(x) \approx \frac{5}{9} f(-\frac{3}{\sqrt{5}}) + \frac{8}{9} f(0) + \frac{5}{9} f(\frac{3}{\sqrt{5}})$$

$$\int_{-1}^{1} dx e^{-x^2} = 1.4964$$

2 pisteen kaavalla 1.43306
3 pisteen kaavalla 1.49868

Numeerinen integrointi - Gaussin kvadratuurit

Yleinen kaava polynomeille:

Haetaan polynomi p(x), joka on ortogonaalinen kaikkien x:n potenssien suhteen n. potenssiin asti.

$$\int_{-1}^{1} dx p(x) x^k = 0 \quad \forall\ k = 0...n-1$$

ja sovitaan, että korkeimman x:n potenssin kerroin on 1.
saadaan n+1 yhtälön lineaarinen yhtälöryhmä, josta polynomin kertoimet määräytyvät.

Esim: 4. asteen polynomiksi tulee

$$x^4 - \frac{6}{7}x^2 + \frac{3}{35}$$

polynomin *p(x)* nollakohdat ovat kvadratuurin pisteet x_1, x_2, x_3 ja x_4

nollakohdat Matlabilla

analyyttisesti:

```
>> solve('x^4-6/7*x^2+3/35')

ans =

 -(3/7 - 2/35*5^(1/2)*6^(1/2))^(1/2)
 -(2/35*5^(1/2)*6^(1/2) + 3/7)^(1/2)
(3/7 - (2*5^(1/2)*6^(1/2))/35)^(1/2)
((2*5^(1/2)*6^(1/2))/35 + 3/7)^(1/2)
```

numeerisesti:

```
>> roots ([1,0,-6/7,0,3/35])

ans =

  -0.8611
   0.8611
  -0.3400
   0.3400
```

Numeerinen integrointi - Gaussin kvadratuurit

Yleinen kaava painofunktiolle *w(x)*:

Haetaan polynomit $p_n(x)$, jotka ovat ortogonaalisia painon *w(x)* suhteen, t.s. niille pätee

$$\int_a^b dx\ w(x)p_n(x)p_m(x) = 0 \quad \text{jos } n \neq m \qquad \text{t.s.} \qquad \langle p_n | p_m \rangle = 0 \quad \text{jos } n \neq m$$

Olkoon $p_{n+1}(x)$:n nollakohdat pisteissä $x_1, x_2, \ldots x_n$

1) voidaan osoittaa, että nollakohdat ovat yksinkertaisia ja kaikki ovat välillä *(a,b)*
2) olkoon $l_i(x)$ nollakohtien kautta kulkeva Lagrangen kantapolynomi, eli $l_i(x_j) = \delta_{ij}$

Silloin $\int_a^b dx w(x)f(x) \overset{!}{=} \sum_{i=0}^{n} w_i f(x_i)$ missä $w_i = \int_a^b dx l_i(x)w(x)$

jos *f(x)* on korkeintaan astetta 2n+1 oleva polynomi

Polynomit osataan integroida tarkasti

Erityisesti

$$\int_a^b dx\ w(x)p_n(x)x^k = 0 \quad \forall k = 0, 1, ..., n-1$$

tämä seuraa siitä, että ortogonaalipolynomit $p_m(x),\ m = 0, 1, ..., k$ ovat myös kantafunktioita,

$$x^k = \sum_{i=1}^{k} c_i P_i(x)$$

$$\int_a^b dx\ w(x)p_n(x)x^k = \sum_{i=1}^{k} c_k \int_a^b dx\ w(x)p_n(x)p_i(x) = 0 \quad \forall k = 0, 1, ..., n-1$$

Numeerinen integrointi - Gaussin kvadratuurit

Todistus:

kirjoitetaan enintään 2n+1 astetta oleva polynomi *f(x)* muotoon

$$f(x) = p_{n+1}(x)q_n(x) + r_n(x)$$

$\begin{cases} q_n(x) \\ r_n(x) \end{cases}$ korkeitaan astetta n olevia polynomeja

quotient
residual

Silloin

$$\int_a^b dx w(x) f(x) = \underbrace{\int_a^b dx w(x) p_{n+1}(x) q_n(x)}_{0} + \int_a^b dx w(x) r_n(x) = \int_a^b dx w(x) r_n(x)$$

Tämän takia valittiin ortogonaalipolynomeja !

Koska $r_n(x)$ on korkeintaan astetta *n* oleva polynomi, se voidaan kirjoittaa tarkasti Lagrangen kantapolynomien avulla:

$$r_n(x) = \sum_{i=0}^{n} r_n(x_i) l_i(x)$$

joten

$$\int_a^b dx w(x) r_n(x) = \int_a^b dx w(x) \sum_{i=0}^{n} r_n(x_i) l_i(x) = \sum_{i=0}^{n} \left[\int_a^b dx w(x) l_i(x) \right] r_n(x_i) = \sum_{i=0}^{n} w_i r_n(x_i)$$

mutta koska

$$f(x_i) = \underbrace{p_{n+1}(x_i)}_{0} q_n(x_i) + r_n(x_i) = r_n(x_i)$$

saadaan väite todistettua.

Tämän takia haettiin ortogonaalipolynomin nollakohtia !

Numeerinen integrointi - Gaussin kvadratuurit

Jos haluat löytää kvadratuurin omaan painofunktioosi *w(x)*, lukaise esim. *Numerical Recipes* kirjan osasta ”Integration of functions” miten se tehdään (löytyy myös netistä).

1) Haetaan rekursiivinen yhtälö, josta ortogonaalipolynomit $p_n(x)$ lasketaan

$$p_{-1}(x) = 0 \quad p_0(x) = 1$$
$$p_{i+1}(x) = (x - a_i)p_i(x) - b_i p_{i-1}(x) \quad i = 0, 1, 2...$$

$$b_i = \frac{\langle p_i | p_i \rangle}{\langle p_{i-1} | p_{i-1} \rangle} \quad i = 1, 2, ... \quad \text{valitaan } b_0 = 0$$
$$a_i = \frac{\langle x p_i | p_i \rangle}{\langle p_i | p_i \rangle} \quad i = 0, 1, 2, ...$$

painofunktio *w(x)* on näissä sisätuloissa

2) Haetaan painot, esim. ratkaisemalla yhtälöryhmä

$$\begin{pmatrix} p_0(x_1) & p_0(x_2) & ... & p_0(x_n) \\ p_1(x_1) & p_1(x_2) & ... & p_1(x_n) \\ & \vdots & & \\ p_{n-1}(x_1) & p_{n-1}(x_2) & ... & p_{n-1}(x_n) \end{pmatrix} \begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{pmatrix} = \begin{pmatrix} \int_a^b w(x)p_0(x) \\ \int_a^b w(x)p_1(x) \\ \vdots \\ \int_a^b w(x)p_n(x) \end{pmatrix}$$

tai kaavasta (hankalahko todistaa)

$$w_i = \frac{\langle p_{n-1} | p_{n-1} \rangle}{p_{n-1}(x_i)p'_{n-1}(x_i)}$$

polynomin $p_{n-1}(x)$ derivaatta

Numeerinen integrointi - Gaussin kvadratuurit

Tavallisia painofunktioita ja kvadratuureja

$\int_{-1}^{1} dx f(x)$	$\mathrm{w(x)} = 1$	Gauss-Legendre
$\int_{-1}^{1} dx \frac{1}{\sqrt{1-x^2}} f(x)$	$\mathrm{w(x)} = \frac{1}{\sqrt{1-x^2}}$	Gauss-Chebyshev
$\int_{-\infty}^{\infty} dx e^{-x^2} f(x)$	$\mathrm{w(x)} = e^{-x^2}$	Gauss-Hermite
$\int_{-1}^{1} dx (1-x)^{\alpha}(1+x)^{\beta} f(x)$	$\mathrm{w(x)} = (1-x)^{\alpha}(1+x)^{\beta}$	Gauss-Jacobi
$\int_{0}^{\infty} dx x^{\alpha} e^{-x} f(x)$	$\mathrm{w(x)} = x^{\alpha} e^{-x}$	Gauss-Laguerre

Huom: integraalin arvoa lasketaan summana $\sum_{i=0}^{n} w_i f(x_i)$

- funktion *f(x)* arvoja lasketaan summaan **vain** sopivan ortogonaalipolynomin nollakohdissa

  Gauss-Chebyshev kvadratuurissa on singulaarinen painofunktio, silti integraalin laskemiseen tarvitaan vain äärellisen funktion *f(x)* arvoja välipisteissä.

- *Jos integrandi on muotoa painofunktio kertaa polynomi käytä Gaussin kvadratuuria*
- *Jos integrandi ei muistuta polynomia*
  - *kokeile adaptiivista integrointirutiinia*
  - *älä käytä kvadratuuria (aina voi kokeilla)*

Reaalifunktion nollakohtien haku

- **Newtonin puolitusmenetelmä**
  - Arvataan kaksi pistettä x_1 ja x_2 *nollakohdan kahta puolta* =>
  - Otetaan uusi piste näiden puolivälistä : $x = (x_1 + x_2)/2$
  - Testataan onko

$$f(x)f(x_1) < 0 \quad ? \qquad x_2 = x$$

$$f(x)f(x_2) < 0 \quad ? \qquad x_1 = x$$

  - Testataan uudelleen, kunnes $|x_1 - x_2| < haluttu\ tarkkuus$

Tarvitsee vain funktion arvoja

$$f(x_1)f(x_2) < 0$$

Esim. $f(x)=x^2-2=0$

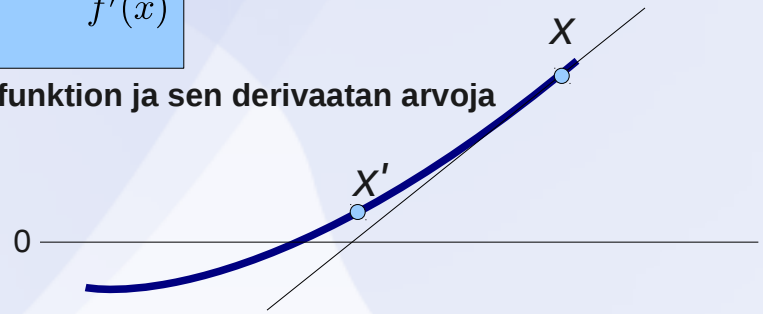
k	zero	deviation
1	1.500000000000000	0.500000000000000
2	1.250000000000000	0.250000000000000
3	1.375000000000000	0.125000000000000
...		
31	1.414213561918586	0.000000000465661
32	1.414213562151417	0.000000000232831
33	1.414213562267832	0.000000000116415
34	1.414213562326040	0.000000000058208

- **Newton (-Raphson) menetelmä**
  - Jos x_0 on funktion *f(x)* nollakohta, niin lähellä sitä

$$0 = f(x_0) = f(x+h) = f(x) + f'(x)h + ... \Rightarrow h = -\frac{f(x)}{f'(x)}$$

siis $\boxed{x' = x - \frac{f(x)}{f'(x)}}$ on parempi arvaus nollakohdaksi

Tarvitsee funktion ja sen derivaatan arvoja

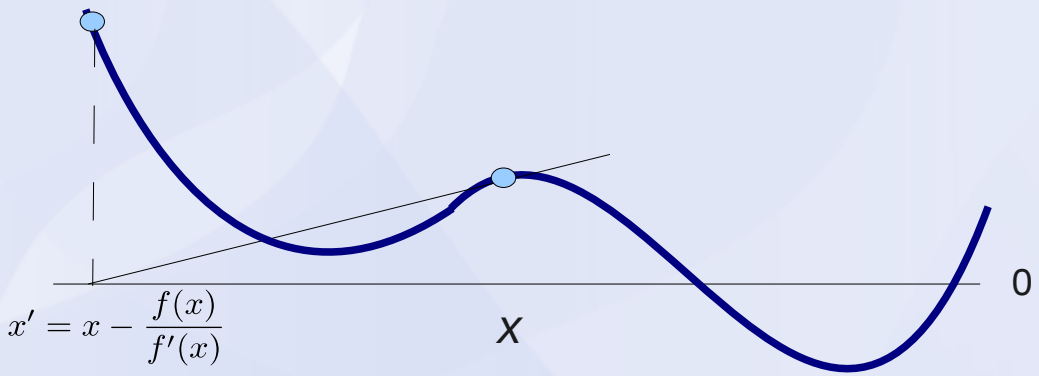


Ideaalisessa tilanteessa erittäin nopea

Esim. $f(x)=x^2-2=0$

k	zero	deviation
1	1.000000000000000	-0.500000000000000
2	1.500000000000000	0.083333333333333
3	1.416666666666667	0.002450980392157
4	1.414215686274510	0.000002123899820
5	1.414213562374690	0.000000000001595

Newtonin menetelmän epästabiilisuus



Jos piste sattuu lähelle lokaalia maksimia/minimiä, niin hyppäys voi kasvaa isoksi ja viedä täysin hakoteille.

Optimointi

- Optimointi on periaatteessa derivaatan (gradientin) nollakohdan hakua
- Optimointi voi toimia
  - ilman gradienttien tuntemusta
    - geneettiset algoritmit
    - hilahakumenetelmät
    - stokastiset menetelmät; *simulated annealing, muurahaispolku optimointi...*
    - epälineaariset simplex-menetelmät
  - gradienttien perusteella
    - tavoitefunktiolla on ainakin jatkuva ensimmäinen (osittais)derivaatta, mieluiten
      myös korkeammat derivaatat
    - suppenee yleensä nopeammin kuin menetelmät joissa gradienttia ei käytetä
    - suppeneminen vain paikalliseen minimiin taattu
    - Yksinkertaiset, vain gradientin käyttöön perustuvat menetelmät käyttävät n muuttujan optimointiin noin n^2 iteraatioaskelta
    - Quasi-Newton menetelmät (perustuvat Hessin matriisin aproksimointiin) suppenevat n muuttujan tapauksessa usein n iteraatioaskeleella

Lineaarinen optimointi: yleinen tehtävänasettelu

maksimoi $c^T x$
ehdoilla $Ax + w = b$
$x, c \geq 0$

(minimoinnissa vaihda ***c***:n merkki)

$$f(x_1, x_2, ..., x_N) = c_1 x_1 + c_2 x_2 + , , , + c_N x_N = \sum_{i=1}^{N} c_i x_i = c^T x$$

Lineaarinen, vain x:n ensimmäisiä potensseja

N optimoitavaa parametria $x_1, x_2, ..., x_N$

$x_i \geq 0$ Sopimus, ei rajoita tulosta

$w_i \geq 0$ *Slack variables;* ratkaistaan myös – otetaan yleisyyden vuoksi

Optimointi: alamäki simplex algoritmi (*downhill simplex; Nelder-Mead)*

(myös lineaarisessa optimoinnissa tunnetaan simplex algoritmi, tämä ei ole läheistä suku sille)

- Minimoi $f(x_1, x_2, ..., x_N)$
- Ei tehokkain, mutta helppo koodata - ja hauska
- Sijoitetaan parametriavaruuteen *N+1* verteksinen polytooppi, simplex:
  valitaan *N+1* pistettä $X_1, X_2, ..., X_{N+1}$ ja lasketaan funktion arvo niissä $f(X_i)$
- Järjestetään pisteet siten, että $f(X_1) < f(X_2) < ... < f(X_{N+1})$, eli viimeinen on huonoin
- Simplex liikkuu parametriavaruudessa seuraavilla säännöillä:
  - PEILAUS : peilataan huonoin piste kaikkien muiden pisteiden keskipisteen ***K*** suhteen
  - LAAJENEMINEN: Jos heijastettu piste on paras kaikista, niin simplex laajenee heijastussuuntaan
  - 1D-KUTISTUS: Jos heijastettu piste on ei ole muita parempi, niin simplex kutistuu yhdessä ulottuvuudessa pisteiden $\boldsymbol{X_{N+1}}$ ja ***K*** välisen suoran suhteen
  - KUTISTUS: Jos heijastettu piste on huonompi kuin mikään muu, kutistuu simplex kokonaisuudessaan kohti parasta pistettä $\boldsymbol{X_1}$

Nelder, J.A., and Mead, R. 1965, "A simplex Method for Function Minimization", Computer Journal, Vol. 7, pp. 308-313.

Optimointi: alamäki simplex algoritmi (*downhill simplex; Nelder-Mead)*

Octave: simplex menetelmiä
- nelder_mead_min
- Nmsmax

Yleismenetelmä:
- fmins

```
f=@(x) (x(1)-5).^2+(x(2)-8).^4 ;
fmins(f,[0;0])
ans =

   4.9998
   7.9918
```

```
-- Function File: [X] = fmins(F,X0,OPTIONS,GRAD,P1,P2, ...)
    Find the minimum of a funtion of several variables.  By default
    the method used is the Nelder&Mead Simplex algorithm

    Example usage:   fmins(inline('(x(1)-5).^2+(x(2)-8).^4'),[0;0])
```

esimerkki ei toimi :^)

Hauskoja Matlab-demoja optimoinnista:

traveling_salesman_demo kauppamatkustajalle lyhin reitti kaupunkien välille; geneettinen algoritmi

samultiprocessordemo jaetaan moniprosessorikoneelle töitä mahd. tehokkaasti; simulated annealing

optimtool Optimization Toolbox Graphical User Interface.

Optimointi: alamäki simplex algoritmi (*downhill simplex; Nelder-Mead)*

Wikipedia:

- **1. Order** according to the values at the vertices:
  $$f(\mathbf{x}_1) \leq f(\mathbf{x}_2) \leq \cdots \leq f(\mathbf{x}_{n+1})$$
- **2.** Calculate x_o, the center of gravity of all points except x_{n+1}.
- **3. Reflection**

  Compute reflected point $\mathbf{x}_r = \mathbf{x}_o + \alpha(\mathbf{x}_o - \mathbf{x}_{n+1})$

  If the reflected point is better than the second worst, but not better than the best, i.e.: $f(\mathbf{x}_1) \leq f(\mathbf{x}_r) < f(\mathbf{x}_n)$,

  then obtain a new simplex by replacing the worst point $\mathbf{x}_{n+1}$ with the reflected point $\mathbf{x}_r$, and go to step 1.
- **4. Expansion**

  If the reflected point is the best point so far, $f(\mathbf{x}_r) < f(\mathbf{x}_1)$,

  then compute the expanded point $\mathbf{x}_e = \mathbf{x}_o + \gamma(\mathbf{x}_o - \mathbf{x}_{n+1})$

  If the expanded point is better than the reflected point, $f(\mathbf{x}_e) < f(\mathbf{x}_r)$

  then obtain a new simplex by replacing the worst point $\mathbf{x}_{n+1}$ with the expanded point $\mathbf{x}_e$, and go to step 1.

  Else obtain a new simplex by replacing the worst point $\mathbf{x}_{n+1}$ with the reflected point $\mathbf{x}_r$, and go to step 1.

  Else (i.e. reflected point is not better than second worst) continue at step 5.
- **5. Contraction**

  Here, it is certain that $f(\mathbf{x}_r) \geq f(\mathbf{x}_n)$

  Compute contracted point $\mathbf{x}_c = \mathbf{x}_o + \rho(\mathbf{x}_o - \mathbf{x}_{n+1})$

  If the contracted point is better than the worst point, i.e. $f(\mathbf{x}_c) < f(\mathbf{x}_{n+1})$

  then obtain a new simplex by replacing the worst point $\mathbf{x}_{n+1}$ with the contracted point $\mathbf{x}_c$, and go to step 1.

  Else go to step 6.
- **6. Reduction**

  For all but the best point, replace the point with

  $x_i = x_1 + \sigma(x_i - x_1)$ for all i $\in \{2, \ldots, n+1\}$. go to step 1.

Note: α, γ, ρ and σ are respectively the reflection, the expansion, the contraction and the shrink coefficient. Standard values are $\alpha = 1, \gamma = 2, \rho = -1/2$ and $\sigma = 1/2$.

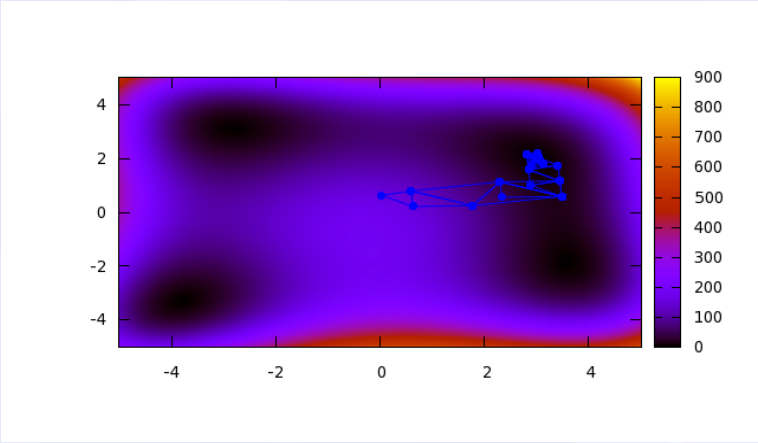
Ohjelma: neldel_mead_simplex.m

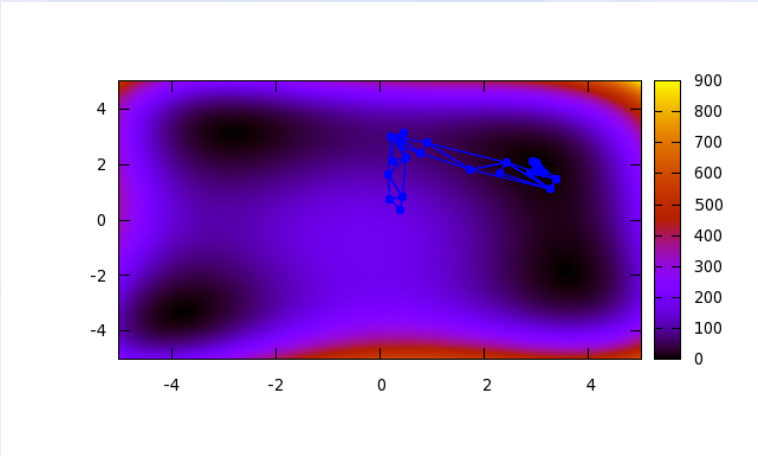
Optimointi: alamäki simplex algoritmi (*downhill simplex; Nelder-Mead)*

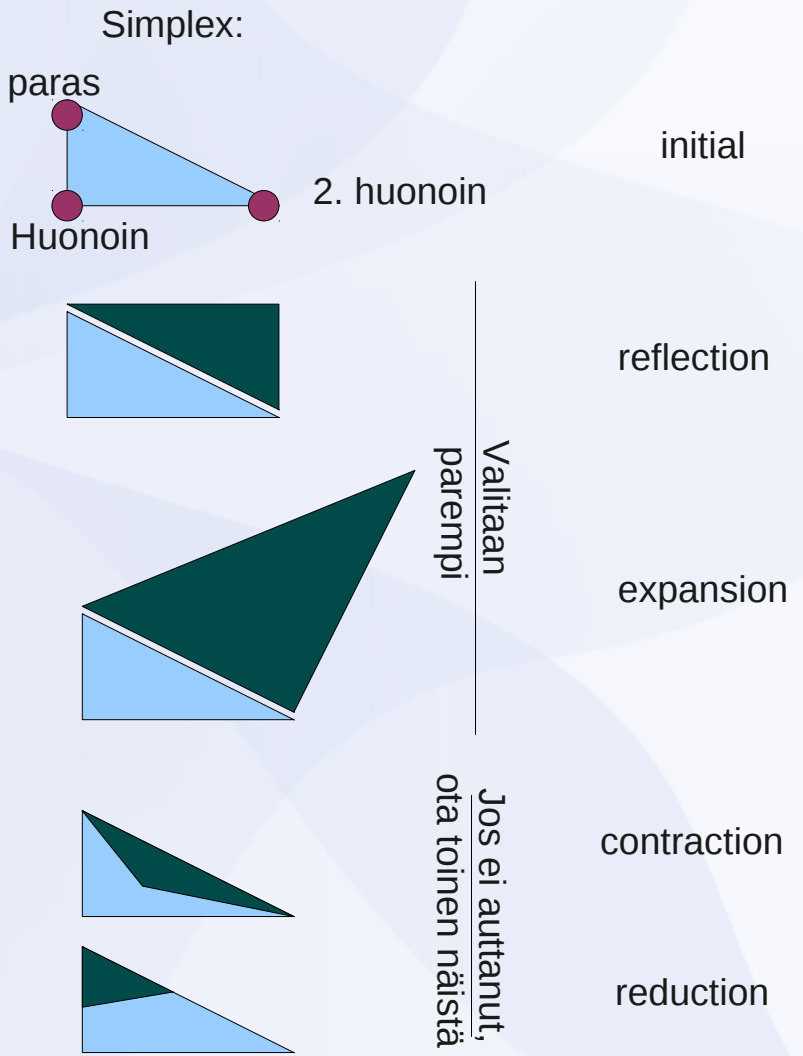
Wikipedian algoritmilla optimoitu Himmelblaun funktio

$$f(x, y) = (x^2 + y - 11)^2 + (x + y^2 - 7)^2$$

Ohjelma: neldel_mead_simplex.m







Steepest descent -menetelmä

- Optimointimenetelmä, jossa mennään jyrkimmän alamäen suuntaan,

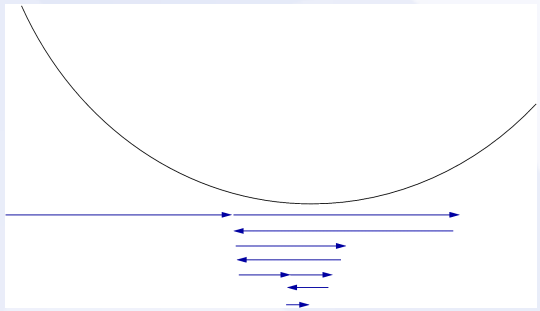
$$\mathbf{x}^{k+1} = \mathbf{x}^k - \alpha_k \nabla f(\mathbf{x_k})$$

- Kuinka pitkälle mennään samaan suuntaan, eli mikä on vakio α_k ?
  Lopetetaan, kun funktion gradientti tähän suuntaan (=suunnattu gradientti) on nolla, eli

$$\underbrace{\nabla f(\mathbf{x}^{k+1}))}_{\text{grad. uudessa pisteessa}} \cdot \underbrace{\nabla f(\mathbf{x}^k))}_{\text{menosuunta}} = 0$$

- Jos gradientteja ei haluta laskea voi gradientin suuntaan edetä myös vakioaskelin, ja palailla takaisin jos mennään ohi minimin: *Backtracking*

1) lasketaan gradientti
2) valitaan sopiva aloitusaskel a
3) otetaan a mittainen askel gradientin suuntaan
4) pieneneekö funktion arvo?
   - kyllä : jatketaan samaan suuntaan
   - ei : siirrytään askel a taaksepäin
     puolitetaan a
     - onko a liian pieni ? on: VALMIS !
       ei: siirry kohtaan 3)



Kamalan tehoton tapa, mutta helppo koodata.

Newtonin optimointimenetelmä (Newton's method)

- Oletetaan, että funktio on **likimain kvadraattinen** (= toista astetta ~ paraabeli)

➡ funktio on likimain muotoa (katkaistu Taylorin sarja moniulotteiselle funktiolle)

$$f(\mathbf{x}+\mathbf{d}) = f(\mathbf{x}) + \nabla f(\mathbf{x}) \cdot \mathbf{d} + \frac{1}{2}\mathbf{d} \cdot H \cdot \mathbf{d} \qquad , \quad H = \nabla\nabla f(\mathbf{x})$$

matriisimuodossa

$$f(\mathbf{x}+\mathbf{d}) = f(\mathbf{x}) + \nabla f(\mathbf{x})^T \mathbf{d} + \frac{1}{2}\mathbf{d}^T H \mathbf{d}$$

Hessin matriisi (*Hessian*) $H_{ij} = \dfrac{\partial^2 f(\mathbf{x})}{\partial x_i \partial x_j}$

- Haetaan gradientin nollakohtaa $\mathbf{x}^* = \mathbf{x} + \mathbf{d}$

$$\nabla f(\mathbf{x}+\mathbf{d}) = \nabla f(\mathbf{x}^*) = 0 = \nabla\left[f(\mathbf{x}) + \nabla f(\mathbf{x})^T \mathbf{d} + \frac{1}{2}\mathbf{d}^T H \mathbf{d}\right]$$

$$= \nabla\left[f(\mathbf{x}) + \nabla f(\mathbf{x})^T (\mathbf{x}^* - \mathbf{x}) + \frac{1}{2}(\mathbf{x}^* - \mathbf{x})^T H (\mathbf{x}^* - \mathbf{x})\right]$$

$$= \nabla f(\mathbf{x}) + \frac{1}{2}H^T \mathbf{d} + \frac{1}{2}H\mathbf{d}$$

$$= \nabla f(\mathbf{x}) + H\mathbf{d}$$

(jos tämä näyttää epäilyttävältä on välivaiheet varminta laskea komponenteittain)

$$\Rightarrow H\mathbf{d} = -\nabla f(\mathbf{x})$$

lineaarinen yhtälöryhmä, ratkaisu on siirtymävektori ***d***

- jos funktio ei ole likikään kvadraattinen tulee ongelmia
- menestys riippuu alkupisteestä ***x***
- geometrisesti Hessin matriisi kuvaa funktion kaareutumista (oletetun minimin lähellä)

Hessin matriisin korvaaminen helpommin laskettavalla?

- Hessin matriisi (*Hessian)* on *N* muuttujan tapauksessa *NxN* matriisi, kukin elementti on toinen derivaatta

  jonka laskeminen voi olla työlästä (numeerisesti: tarvitsee monta funktion arvoa)

  Jos toiset derivaatat voi laskea analyyttisesti se kannattaa tehdä.

- Voiko Hessin matriisin korvata?

  Esim. *Steepest descent* -menetelmä on itse asiassa Newtonin menetelmä jossa Hessin matriisi on yksikkömatriisi;

  $$H\mathbf{d} = \mathbf{d} = -\nabla f(\mathbf{x})$$

  $$H\mathbf{d} = -\nabla f(\mathbf{x}) \Leftrightarrow \mathbf{d} = -H^{-1}\nabla f(\mathbf{x})$$

- Tarvitaan oikeastaan Hessin matriisin *käänteismatriisi*

- **Sekanttimenetelmät:**
  Ei lasketa Hessin matriisia, vaan rakennetaan matriisi joka iteraatio iteraatiolta lähestyy Hessin käänteismatriisia

  Esim. **Broyden-Fletcher-Goldfarb-Shanno (BFGS)** algoritmi

Levenberg-Marquardt menetelmä

(1944) (1963)

- Hyvin yleinen, tunnetaan parhaiten menetelmänä tehdä **epälineaarinen** pienimmän neliösumman sovitus
- Aloittaa kuten *steepest descent*, kääntyy Newtonin menetelmäksi iteroinnin edetessä

$$\mathbf{d} = -\nabla f(\mathbf{x}) \longrightarrow H\mathbf{d} = -\nabla f(\mathbf{x})$$

Tämä saavutetaan lisäämällä aluksi Hessin matriisin diagonaalille vakio jota hiljalleen pienennetään

Levenberg: $(H + aI)\mathbf{d} = -\nabla f(\mathbf{x})$ *suuri a: ~ steepest descent*
pieni a: ~ Newton

Verrataan tätä pienimmän neliösumman ehtoon sovitusparametreille ***c***

$A^T A\mathbf{c} = A^T\mathbf{y}$ $A_{ij} = \phi_j(x_i)$ kantafunktiot tunnetuissa pisteissä *(design matrix)*

Levenberg korvasi vasemman puolen matriisin A^TA hiukan muunnetulla versiolla

$$(A^T A - aI)\mathbf{c} = A^T\mathbf{y}$$

”damping”

Jos iteraatioaskel ei johda parempaan tulokseen niin *damping*-vakiota *a* kasvatetaan (siirrytään enemmän *steepest descent*-iteraation tyyliin eli ”varman päälle”)

- Suuri *a* tarkoittaa samalla myös sitä, että askeleen pituus on lyhyempi: $d \sim \frac{1}{a}$

⇨ Levenbergin menetelmässä askelta voi muuttaa vain kaikkiin suuntiin saman verran

Levenberg-Marquardt menetelmä

Mitä Levenbergin algoritmissa

$$(H + aI)\mathbf{d} = -\nabla f(\mathbf{x})$$

tapahtuu, kun *a* on suuri? Hessin matriisiahan ei silloin käytetä lainkaan hyväksi ja törmätään ns. *error valley* ongelmaan: jyrkimmän alamäen menetelmässä eteneminen sahaa zigzag-rataa kohti laakson pohjaa.

Marquardt esitti 1963 oleellisen parannuksen: koska meillä on Hessin matriisi *H* joka tapauksessa jo käytössä, kannattaa ottaa siitä hyöty irti. Hessin matriisi kertoo, miten funktio *f(x)* käyristyy eri suuntiin. **Kannattaa siirtyä paljon loivan alamäen suuntaan ja vähän jyrkän.** Näin *error valley* ongelmaa ei tule, vaan laaksossa edetään alaspäin tehokkaasti.

$$\begin{aligned} Hc &= \lambda c \\ D &= diag(\lambda) \\ (H + aD)\mathbf{d} &= -\nabla f(\mathbf{x}) \end{aligned}$$

ratkaistaa Hessin matriisin ominaisarvot ja kootaan ne diagonaalimatriisiksi *D*

pieni ominaisarvo – suuri *d*
suuri ominaisarvo – pieni *d*

Askelvektori *d* saadaan Levenberg-Marquardt menetelmässä siis laskemalla käänteismatriisi $(H+aD)^{-1}$

$$\mathbf{d} = (H + aD)^{-1}\nabla f(\mathbf{x})$$

Jacobin matriisi ja Hessin matriisi

- Jacobin matriisi (*jacobian)* on koottu eri pisteissä lasketuista gradienteista:

$$\mathrm{J}=\begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots \frac{\partial f_2}{\partial x_n} \\ & \cdots & \\ \frac{\partial f_N}{\partial x_1} & \frac{\partial f_N}{\partial x_2} & \cdots \frac{\partial f_m}{\partial x_n} \end{pmatrix} \qquad f_1 = f(x_1) \ldots f_m = f(x_m)$$

Joten *gradienttifunktion* Jacobin matriisi on Hessin matriisi

Funktion nollakohtien haussa käytetään Jacobin matriisia
Funktion optimoinnissa käytetään Hessin matriisia

Luottoalue (Trust Region) algoritmit

- *Dog Leg:* nimi viittaa uuden pisteen valintaan ”koiramaisesti”; nimi tulee golf termistä

  Usein tehokkaampi kuin Levenberg-Marquardt

- Myös *steepest descent* + Newton, mutta käyttää ns. *trust region* – ideaa (Powell):
  Trust region : Funktion oletetaan olevan luotettava vain nykyisen pisteen lähellä
  => optimointiin tulee reunaehto, että ollaan luotettavan alueen sisällä

- Usein tarvitaan ylimääräinen suure, *objective or model function*, joka jotenkin mittaa sitä onko uusi piste parempi kuin vanha, eli voidaanko luottoaluetta suurenta vai pitääkö sitä kutistaa.

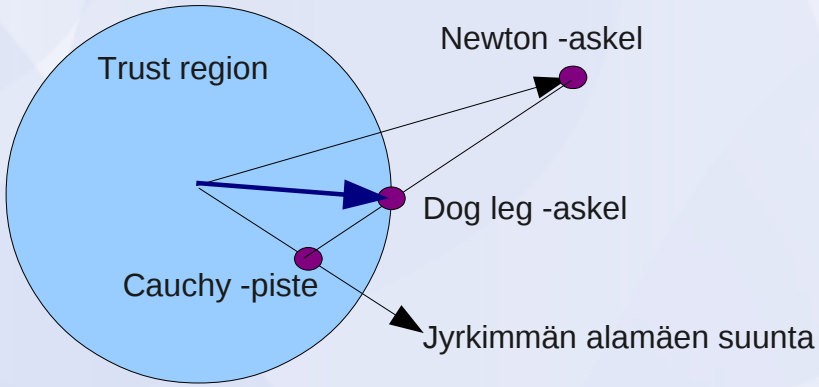
- *trust region* pyrkii parantamaan optimoinnin stabiilisuutta.

>> help fsolve
FSOLVE solves systems of nonlinear equations of several variables.
FSOLVE attempts to solve equations of the form:
F(X) = 0 where F and X may be vectors or matrices.

kuvaus Mathworks'in www-sivuilla:

The fsolve function has four algorithms for solving systems of nonlinear multidimensional equations:
- Trust-region dogleg
- Trust-region-reflective
- Levenberg-Marquardt
- Gauss-Newton

Dog leg -algoritmi (Powell 1970)



1) Valitaan mallifunktio, joka käyttäytyy nykyisen pisteen lähellä kunte *f(x)*
2) Cauchy pisteessä *mallifunktio* on minimissä jyrkimmän alamäen suuntaan
3) Jos Cauchy-piste on luottoalueen ulkopuolella, seuraava askel on alamäkeen luottoalueen rajalle
4) Jos Newton-askel (Gauss-Newton) on luottoalueen sisällä, otetaan se.
5) jos Cauchy-piste on sisällä ja Newton ulkona, lasketaan askel kuvan tavalla

Mukava lisä on, että luottoalueen rajan ja suoran leikkauskohta voidaan laskea analyyttisesti, eikä se vie aikaa.

Differentiaaliyhtälöt

- Numeerinen ratkaisu ei koskaan ole kokonainen ratkaisujoukko vaan yksittäinen käyräparven edustaja

  $$f(x) = A\sin(kx) + B\cos(kx) \quad \text{yksille } A,\ B,\ \text{ja } k$$

  => tarvitaan alkuehdot, joilla ratkaisu määräytyy yksikäsitteisesti numeriikkaa varten

  Esim. $\begin{matrix} f(0) = 0 \\ f'(0) = 1 \end{matrix}$ => kaksi vakiota määräytyy, riittää 2. asteen differentiaaliyhtälön numeeriseen ratkaisuun

- Eulerin menetelmä: mennään lyhyt matka siihen suuntaan mihin derivaatta (gradientti) osoittaa
  - hyvin epätarkka – virhe on $O(h^2)$ Esim. **euler.m**
  - yksinkertainen kirjoittaa pikaista testiä varten – vaikea tehdä virhettä *koodauksessa*
- n. asteen differentiaaliyhtälö hajotetaan n:ksi kytketyksi 1. asteen differentiaaliyhtälöksi

  Esim.

$$\frac{d^3}{dx^3}f(x) + \frac{d^2}{dx^2}f(x) + 5x = 0 \quad <=> \quad \begin{cases} p(x) = f'(x) \\ g(x) = f''(x) = p'(x) \\ \frac{d^3}{dx^3}f + \frac{d^2}{dx^2}f + 5x = g'(x) + g(x) + 5x = 0 \end{cases}$$

kolme kytkettyä 1. asteen differentiaaliyhtälöä

Kytketyt differentiaaliyhtälöt (*coupled diff. equations)*

Edetään piste pisteeltä ja pidetään kirjaa senhetkisistä funktion ja derivaattojen arvoista.

Edellisen esimerkin 3. asteen yhtälön ratkaisu voisi tapahtua Eulerin kaavalla näin:

1. Aloitetaan pisteestä *x,* missä *f(x)*, *f'(x)=p(x)* ja *f''(x)=g(x)* tunnetaan
2. ratkaistaan g(x+h) integroimalla differentiaaliyhtälöä

$$g'(x) + g(x) + 5x = 0$$

   pisteestä *x* pisteeseen *x+h*

$$g(x+h) \approx g(x) + g'(x)h = g(x) + [-g(x) - 5x]h$$

3. ratkaistaan *p(x+h)* yhtälöstä

$$p(x+h) \approx p(x) + p'(x)h = p(x) + g(x)h$$

4. ratkaistaan *f(x+h)* yhtälöstä

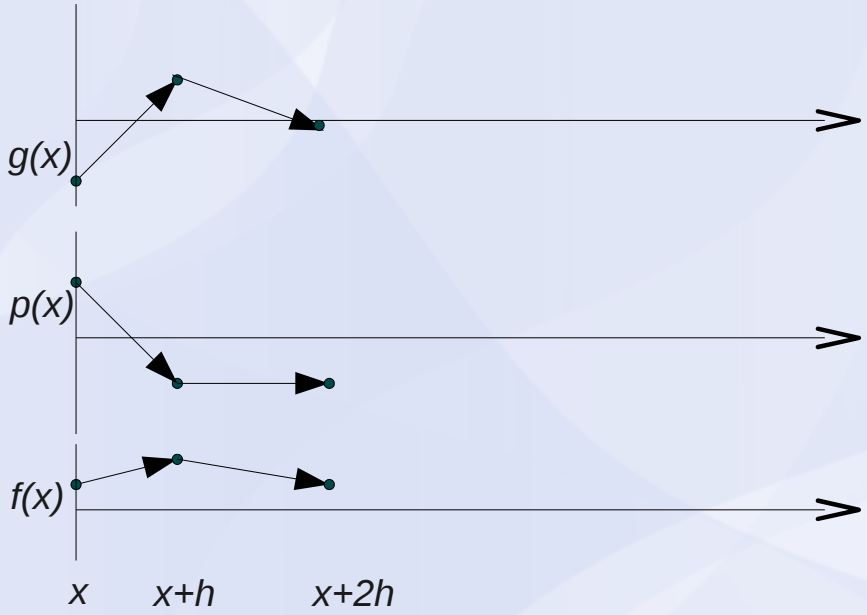
$$f(x+h) \approx f(x) + f'(x)h = f(x) + p(x)h$$

5. edetään kirjanpidossa seuraavaan pisteeseen ja asetetaan $x \to x + h$
6. siirry kohtaan 2

Esim. **euler2.m**

Parannettu versio: Korvataan Eulerin menetelmä paremmalla kunkin 1.asteen yhtälön ratkaisussa

Kytketyt differentiaaliyhtälöt



Toisen kertaluvun differentiaaliyhtälön ratkaiseminen:
kullakin *x* pidetään kirjaa funktiosta *f(x)* ja sen derivaatoista *p(x)* ja *g(x)*

Mikä on sopiva askeleen pituus *h*? Valitaan menetelmä, jolla ratkaistaan väli *x*-> *x*+h ja lyhennetään askelta kunnes *f(x+h)* on kyllin tarkka.

Ongelmia: tarvittava *h* voi olla aivan liian pieni. Silloin
1.) Etsitään parempi menetelmä
2.) Keksitään tyystin eri tapa edetä *x*-pisteiden välillä.

Kytkettyjen yhtälöiden stabiilius ja kankeat yhtälöt
(stability, stiff set of equations)

- Differentiaaliyhtälöryhmä on stabiili, jos kaksi lähellä olevaa alkuarvoa antavat lähellä olevan tuloksen
  vrt. kaoottinen systeemi: pieninkin muutos alkuarvoon aiheuttaa valtavan eron lopputulokseen

- Miten saadaan selville, onko kytketty yhtälö stabiili?

  1) Kirjoitetaan yhtälöryhmä matriisimuotoon $F'(x) = AF(x) + q(x)$

Esim:

$$f_1'(x) = a_{11}f_1(x) + a_{12}f_2(x) + q_1(x)$$
$$f_2'(x) = a_{21}f_1(x) + a_{22}f_2(x) + q_2(x)$$

on matriisimuodossa $F'(x) = AF(x) + q(x)$ $(q(x)$ tunnettu$)$

missä $F(x) = \begin{pmatrix} f_1(x) \\ f_2(x) \end{pmatrix}, A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}, \mathbf{q}(x) = \begin{pmatrix} q_1(x) \\ q_2(x) \end{pmatrix},$

---

2) Puretaan yhtälöiden kytkentä; haetaan apufunktiot, jotka eivät ole kytkettyjä.
   Diagonalisoidaan matriisi ***A*** : $A = PDP^{-1}$ missä ***D*** on diagonaalimatriisi
   Silloin

$$F'(x) = AF(x) + q(x) = PDP^{-1}F(x) + q(x)$$
$$\Rightarrow \; P^{-1}F'(x) = DP^{-1}F(x) + P^{-1}q(x)$$
$$\Rightarrow \; G'(x) = DG(x) + Q(x)$$

ei ole enää kytketty, koska ***D*** on diagonaalimatriisi

Esim. $g_1'(x) = d_{11}g_1(x) + q_1(x)$
$g_2'(x) = d_{22}g_2(x) + q_2(x)$

Kytkettyjen yhtälöiden stabiilius ja kankeat yhtälöt
(stability, stiff set of equations)

3) Kukin yhtälö on nyt muotoa

$$g'(x) = \lambda g(x) + q(x)$$

missä λ on A:n ominaisarvo.
Unohdetaan hetkeksi funktio ***q(x)***; silloin ratkaisu on $g(x) = e^{\lambda x}$

Selvästi ratkaisu menee suurilla ***x***:n arvoilla
- nollaan, jos $\lambda < 0$
- äärettömiin, jos $\lambda > 0$

Tämän perusteellä näyttäisi, että yhtälöryhmä voi olla stabiili suurilla ***x*** vain, jos **kaikki *A*:n ominaisarvot ovat negatiivisia.**

Esim. Matriisin ominaisarvot ovat -10 ja 10, eli ratkaisut ovat $Ae^{-10x} + Be^{10x}$
Oletetaan, että haluttu ratkaisu on suurilla ***x*** nollaan menevä exponenttifunktio.
Lähdettäessä pienestä ***x***:n arvosta numeerinen ratkaisu saattaa alkaa hyvin, mutta väistämättä pyöristysvirheet kasautuvat ja toinen komponentti pääsee vallalle:

$e^{-10x} + \epsilon e^{10x}$ => numeerinen ratkaisu karkaa käsistä ennemmin tai myöhemmin, olipa ϵ kuinka pieni hyvänsä

Seuraavaksi tarkastellaan esimerkkiä, jossa ominaisarvot ovat negatiiviset, mutta silti yhtälöryhmä on numeeriesti vaikea ratkaista.

Kytkettyjen yhtälöiden stabiilius ja kankeat yhtälöt
(stability, stiff set of equations)

- Kankeat yhtälöt (*stiff set*) vaativat tavanomaisella piste pisteeltä etenevältä ratkaisumenetelmältä liikoja: jokin yhtälöistä on epästabiili ellei askelpituutta on lyhennetä hyvin lyhyeksi, paljon lyhyemmäksi kuin mitä muiden yhtälöiden ratkaiseminen vaatisi.

Esim.

$$f'(x) = 264f(x) - 2128g(x)$$
$$g'(x) = 133f(x) - 1066g(x)$$

$$A = \begin{pmatrix} 264 & -2128 \\ 133 & -1066 \end{pmatrix}$$

ominaisarvot : $\begin{pmatrix} -2 \\ -800 \end{pmatrix}$

molemmat <0, näyttää stabiililta ensi silmäyksellä

=> ratkaisufunktiot ovat lineaarikombinaatioita funktioista e^{-2x}, e^{-800x}

mutta: numeerinen ratkaisu on epästabiili, ellei askel ole alle 1/800
- vaikka tuloksen kannalta eksponentin -800 osuus on yhdentekevä jos x>0 !

Kankeat yhtälöt: Implisiittinen Eulerin menetelmä

Yksi ratkaisutapa:

Ei oteta Eulerin askelia eteenpäin derivaatalla pisteessä *x*, vaan lasketaan derivaatta *seuraavassa pisteessä x+h*. Tämä on ns. **implisiittinen Eulerin menetelmä**

(eksplisiittinen) Eulerin menetelmä: $f(x+h) \approx f(x) + f'(x)h$

implisiittinen Eulerin menetelmä: $f(x+h) \approx f(x) + f'(x+h)h$ (*)

Tällä saadaan numeerinen ratkaisu stabiiliksi

Esim. Yhtälöryhmä $\mathbf{F}'(x) = A\mathbf{F}(x)$ A:n ominaisarvot $\lambda < 0$ ($-A$ on positiividefiniitti)

$$\mathbf{F}(x+h) = \mathbf{F}(x) + Ah\mathbf{F}(x+h) \Leftrightarrow \mathbf{F}(x+h) = (1-Ah)^{-1}\mathbf{F}(x)$$

Suurella argumentin arvolla ratkaisu on $\mathbf{F}(x+nh) = [(1-Ah)^{-1}]^n\mathbf{F}(x)$

tämän pitää olla stabiili

(I-Ah)$^{-1}$:n ominaisarvot *A*:n ominaisarvojen avulla:

$$(1-Ah)^{-1}x = \lambda' x \Leftrightarrow x = \lambda'(1-Ah)x = \lambda'(x-hAx) = \lambda'(x-h\lambda x) = \lambda'(1-h\lambda)x \Leftrightarrow \lambda' = (1-h\lambda)^{-1}$$

siis

$\lambda < 0 \Rightarrow \lambda' < 1$

$[(1-Ah)^{-1}]^n \to 0, \text{ kun } n \to \infty$

, joten **ratkaisu on stabiili riippumatta askeleen pituudesta *h* !**

(mutta epätarkka suurella *h* ja matriisin kääntö tarvitaan joka askeleella ...)

(*) $f(x+h) \approx f(x) + f'(x+h)h = f(x) + (f'(x) + f''(x)h + ...)h = f(x) + f'(x)h + f''(x)h^2 + ...$

sama kuin Taylorin sarja 1. kertalukuun

1. asteen differentiaaliyhtälön ratkaiseminen

- Edetään pisteestä *x* pisteeseen *x+h* mahdollisimman tarkasti, lasketaan funktio ja derivaatta tarpeellisessa määrässä välipisteitä
- Käytettävissä on funktion derivaatan arvo missä tahansa pisteessä
- Algoritmin kertaluku kertoo mihin *h*:n potenssiin verrannollinen virhe on.
  - Esim. 2. kertaluvun Runge Kutta : virhe on $O(h^3)$
- Standardi on käyttää vähintään 4. kertaluvun algoritmeja, virhe $O(h^5)$
- Ongelman voi tulkita ns. **Greenin funktion** *G* avulla
  - määritelmä $f(x') = \int dx G(x', x) f(x) \quad , \; x' = x + h$

    Greenin funktio *G(x',x)* ”vie” funktion pisteestä *x* pisteeseen *x'* .

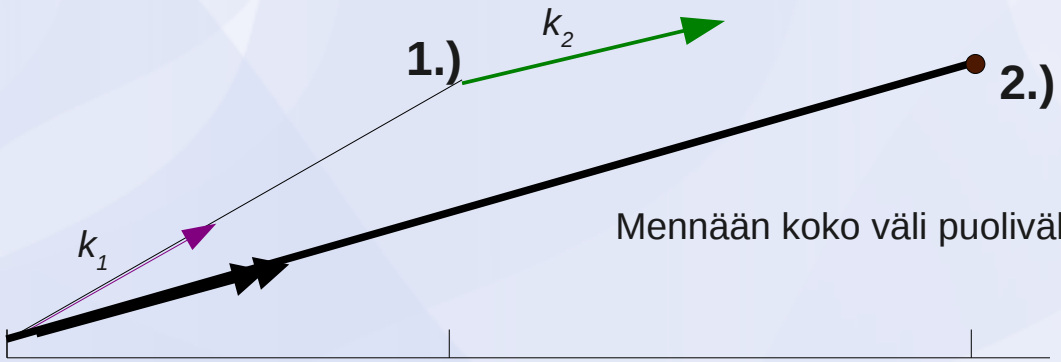
    Esim. 4. kertaluvun algoritmin taustalla on 4. kertaluvun Greenin funktio.
- Esim.

  4. kertaluvun Runge-Kutta algoritmi (lyhyeltä nimeltään RK4)

  1) laske derivaatta pisteessä *x* : olkoon se k_1
  2) Mene derivaatan k_1 suuntaan pisteeseen *x+h/2* ja laske siellä uusi derivaatta k_2
  3) Mene derivaatan k_2 suuntaan pisteeseen *x+h/2* ja laske siellä uusi derivaatta k_3
  4) Mene derivaatan k_3 suuntaan pisteeseen *x+h* ja laske siellä uusi derivaatta k_4
  5) Kokoa kaikki tieto yhteen: mene derivaatan $(k_1+2k_2+2k_3+k_4)/6$ suuntaan pisteeseen *x+h.* Olet perillä.

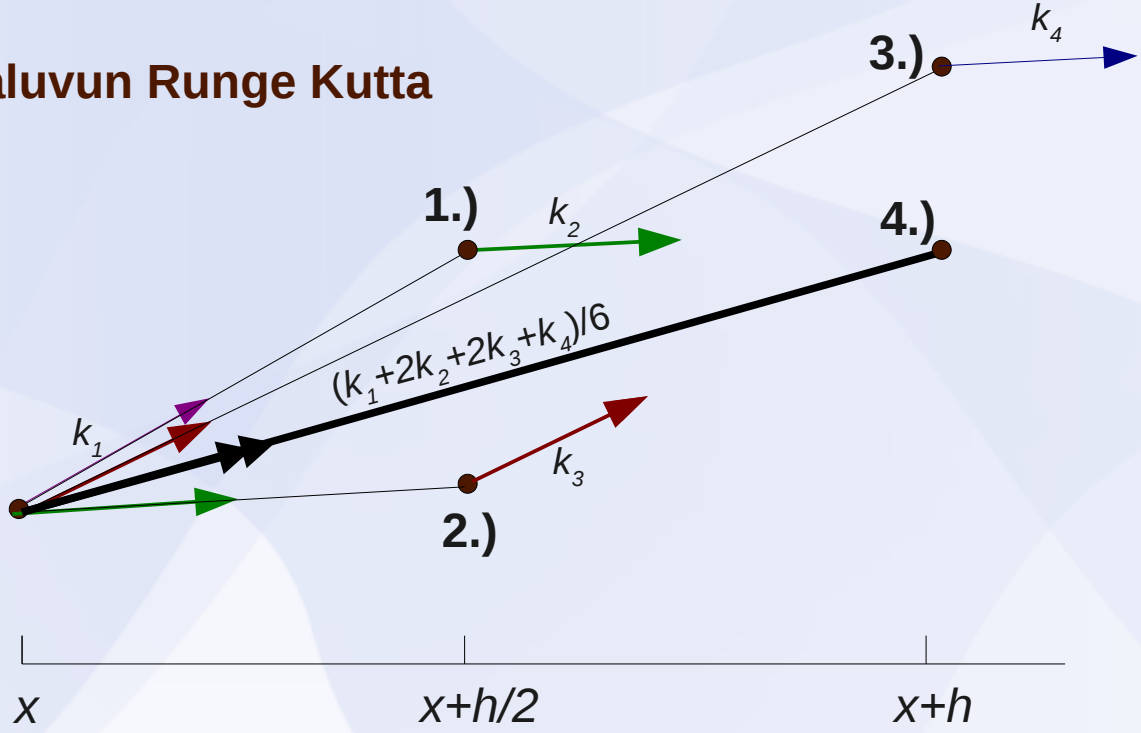
  Tämä olettaa yleisesti, että *y'=f(x,y)*, eli derivaatta voi riippua myös funktion arvosta *y* itsestään. Jos *y'=f(x)* ovat derivaatat $k_2 = k_3$.

2. kertaluvun Runge Kutta



Mennään koko väli puolivälissä lasketulla derivaatalla

4. kertaluvun Runge Kutta



Reunaehdot

- Esim.
  Sidotun tilan aaltofunktion pitää mennä kaukana nollaan (jotta todennäköisyys päätyä äärettömän kauas on nolla, eli hiukkanen on sidottu). Tämä ehto johtaa **energiatilojen kvantittumiseen**.

  - **Tähtäysmenetelmä** (*shooting method):* arvataan energia jolla lähdetään alkupisteestä ja edetään kunnes ollaan *äärettömyydessä* (valitaan sopivan kaukainen kohde)
    Jos aaltofunktio ei mennyt nollaan korjataan energiaa – kaikki energian arvot eivät kelpaa

  - **Välipisteen derivaatan sovitus :** kaksi ratkaisua, toinen lähtee alkupisteestä oikealle, toinen äärettömyydestä vasemmalle. Sovitussa välipisteessä katsotaan miten paljon ratkaisujen derivaatat eroavat – jos pisteeseen tulee mutka korjataan energiaa.
    Tavoite on saavutettu kun aaltofunktion derivaatta on jatkuva myös kohtauspisteessä.
    (aaltofunktion arvo saa olla eri, kerro toinen ratkaisun osista vakiokertoimella)

- Usein joutuu ratkaisemaan ongelman analyyttisesti alkupisteen lähellä saadakseen alkuarvot oikein

  - Esim. radiaalisen Schrödingerin yhtälön ratkaisun pisteen *r=0* lähellä määrää usein hiukkasten välinen otentiaali, koska se on useimmiten singulaarinen – äärettömyydessä potentiaali on nolla.

Ominaisarvoyhtälöt

- Ominaisarvoyhtälö on $A\mathbf{x} = \lambda\mathbf{x}$
- Yleistetty ominaisarvoyhtälö on $A\mathbf{x} = \lambda B\mathbf{x}$, missä myös *B* on matriisi
- Operaattoreista siirrytään matriiseihin näin:

1) Valitse täydellinen joukko kantafunktioita $\{\phi_i\}$ Esim. Tasoaaltokanta $\phi_i(\mathbf{x}) = e^{i\mathbf{k}_i\cdot\mathbf{x}}$
   (kaikki funktiot voidaan esittää näiden kantafunktioiden avulla)
2) Laske operaattorin esitys tässä kannassa,
   eli kokoa matriisi, jonka matriisielementit ovat $A_{ij} = \langle\phi_i|\hat{A}|\phi_j\rangle$

Esim. operaattori $\hat{A} = -\nabla^2$

$$A_{ij} = \int d\mathbf{x} e^{-i\mathbf{k}_i\mathbf{x}}(-\nabla^2)e^{i\mathbf{k}_j\mathbf{x}} = -\mathbf{k}_j^2\int d\mathbf{x} e^{-i\mathbf{k}_i\mathbf{x}}e^{i\mathbf{k}_j\mathbf{x}} = -\mathbf{k}_j^2\delta_{ij}$$

eli tämä operaattori on diagonaalinen tasoaaltokannassa

Operaattoriyhtälöstä $\hat{A}|x\rangle = \lambda|x\rangle$ päästään kannan valinnalla matriisiyhtälöön $A\mathbf{x} = \lambda\mathbf{x}$

- Kvanttimekaniikassa pulmana on kantatilojen ääretön määrä, ääretönulotteinen matriisi
  Käytännössä siis kantatilojen joukko on pakko katkaista jostakin,
  usein pidetään esim. alimpia energiatiloja vastaavat kantafunktiot.
- Ominaisarvoyhtälön ratkaisuun kannattaa käyttää valmiita ohjelmia, *musta laatikko* -periaatteella.

Ominaisarvoyhtälöt: Menetelmän valinta

- Tarvitsetko kaikki ominaisarvot ja kenties myös niitä vastaavat ominaisvektorit ?
  - ominaisarvot on laskettava aina, ominaisvektorit saa haluttaessa lisätyöllä
  - ominaisvektorit tarvitaan jos haluat esim. muunnoksen, joka ***diagonalisoi* matriisin *A***

$A\mathbf{x} = \lambda\mathbf{x}$ ratkaisu on {ominaisarvo,ominaisvektori}-pari $\{\lambda_i, \mathbf{x}_i\}$

kokoa ominaisvektoreista ja ominaisarvoista matriisit *P* ja *D* seuraavasti:

eli ominaisvektorit pystyriveiksi. Tällä saavutetaan tulos

$$P = \begin{pmatrix} x_1 & x_2 & \dots & x_n \end{pmatrix} \qquad D = diag(\lambda_1, \lambda_2, ..., \lambda_n)$$

$$A = PDP^{-1}$$ Sanotaan, että P diagonalisoi A: n $P^{-1}AP = D$

Esim. Haluat laskea matriisin *k*:nnen potenssin,

$$A^k = (PDP^{-1})^k = (PDP^{-1})(PDP^{-1})(PDP^{-1})...(PDP^{-1}) = PDDD...DP^{-1} = PD^kP^{-1}$$

$$D^k = diag(\lambda_1^k, \lambda_2^k, ..., \lambda_n^k)$$

Nyt osaat laskea myös mitä *exp(A)* on !

$$exp(A) = I + A + A^2 + ...$$

- Jos tarvitset vain muutamia ominaisarvoja, niin käytettävissä on paljon nopeampi menetelmä, etenkin jos matriisi on **harva ja/tai rakenteellinen:** tulon ***Ax*** laskeminen on nopeaa kaikille ***x.***
  - Lanczos menetelmä
  - Arnoldi menetelmä

  } **Krylovin aliavaruusmenetelmiä**

  ...kuten liittogradienttimenetelmäkin!

Esim. Matlabin menetelmä **eigs** - tavallinen ominaisarvoyhtälön ratkaisija on **eig** ”eig+sparse = **eigs**”

Ominaisarvoyhtälöt: potenssimenetelmä (power method)

- Mitä tapahtuu, jos kerrot matriisia itsellään ? Eli lasket mitä on A^k, kun k on suuri.
  Vastaus liittyy varmasti ominaisarvoihin, koska matriisin potenssin voi laskea ominaisarvojen avulla:

$$A^k = Pdiag(\lambda_1^k, \lambda_2^k, ..., \lambda_n^k)P^{-1}$$

  jos ominaisarvot on järjestetty suurimmasta pienimpään, eli 1. on suurin, niin lopulta sen k's potenssi $\lim_{k\to\infty} A^k = Pdiag(\lambda_1^k, 0, ..., 0)P^{-1}$

  on ylivoimaisesti suurin. Siis

$$\{A[A^k\mathbf{b}]\}_i = [A^{k+1}\mathbf{b}]_i \to [Pdiag(\lambda_1^{k+1}, 0, ...0)P^{-1}\mathbf{b}]_i = P_{i1}\lambda_1^{k+1}P_{1n}^{-1}b_n = \lambda_1[P_{i1}\lambda_1^k P_{1n}^{-1}b_n] = \lambda_1[A^k\mathbf{b}]_i$$

$$\Rightarrow A[A^k\mathbf{b}] = \lambda_1[A^k\mathbf{b}]$$, joten hakasuluissa on suurinta ominaisarvoa vastaava ominaisvektori

- Operoidaan matriisilla *johonkin* vektoriin monta kertaa:

$$\mathbf{x}_1 = \lim_{k\to\infty} \frac{A^k\mathbf{b}}{||A^k\mathbf{b}||}$$

  (Jotta lasku ei karkaa numeerisesti, kannattaa vektori normittaa)

- Tämä ei aina suppene...

Löysimme siis ainakin yhden menetelmän jolla laskea suurin ominaisarvo ja sitä vastaava ominaisvektori. Oletus, että A on diagonalisoituva matriisi helpotti todistusta.

Esim. **poww.m**

Ominaisarvoyhtälöt: Krylovin aliavaruus (Krylov subspace) $\{b,A^1b,A^2b...A^kb\}$

- Edellä laskettiin $\{b,A^1b,A^2b...A^kb\}$, mutta heitettiin kaikki muut paitsi viimeinen vektori menemään. Pidetään kaikki – tämä joukko vektoreita virittää ns. Krylovin aliavaruuden K_{k+1}

  (eli lineaarikombinaatiot ovat aliavaruuden vektoreita)

- Ortogonalisoidaan vektorit $\{b,A^1b,A^2b...A^kb\}$ jotta saadaan aliavaruuden kanta. Nämä kantavektorit oletettavasti kuvaavat hyvin suurimpia ominaisarvoja vastaavia om. vektoreita.

- Yksi ortogonalisointimenetelmä on Gram-Schmidt, joka Krylovin aliavaruudessa näyttää tältä:

  1. vektori on *b*
  2. vektori on *Ab – (Ab:n projektio b:lle)*
  3. vektori on A^2b - *(A^2b:n projektio Ab:lle) - (A^2b:n projektio b:lle) jne.*

  ”klassinen” algoritmi numeerisesti epästabiili, parempi versio on yhtä helppo ks. seuraaava sivu !

- Tärkeimmät Krylovin aliavaruusmenetelmät ovat
  - **Arnoldi** iterointi - 1951 alkuperäinen, paljon parannuksia 90-luvulla ja myöhemminkin mm. ARPACK ohjelmapaketti, alalaji *implicitly restarted Arnoldi method*
  - **Lanczos** iterointi

- Hakevat ***osan ominaisarvoista*** ja ominaisvektoreista (jos haluat ne)

- Tarvitsevat alkuarvauksen *b*, jonka ”hyvyys” ratkaisee suppenemisnopeuden
  Alkuarvaus ei saa olla ortogonaalinen lopullisen ominaisvektorin kanssa

- **Tarvitsevat vain tulon *Ab* laskemisen**, matriisi *A* voi olla valtava, kunhan se on harva ja hyvin rakenteellinen (Esim. QR-iteroinnillakin voi hakea ominaisarvoja, mutta se tuhoaa matriisin *A* rakenteen ja on siksi liian hidas isoille matriiseille)

Reunahuomautus: parannettu Gram-Schmidt *(Modified Gram-Schmidt)*

Miten tehdä ortogonalisointi numeerisesti stabiilisti (kvanttimekaniikan merkinnöin)

Muunnos ei-ortogonaalisesta kannasta $\{|v_1\rangle, |v_2\rangle, ..., |v_n\rangle\}$ ortogonaaliin kantaa $\{|u_1\rangle, |u_2\rangle, ..., |u_n\rangle\}$

- Klassisessa Gram-Schmidt (**CGS**) ortogonalisoinnissa pyöristysvirheet kasautuvat. Lopulta virhe on niin paha, etteivät vektorit enää ole lainkaan ortogonaalisia

$$|u_1\rangle = |v_1\rangle$$

$$|u_k\rangle = |v_k\rangle - \sum_{j=1}^{k-1} \frac{\langle u_j|v_k\rangle}{\langle u_j|u_j\rangle}|u_j\rangle$$

Älä käytä numeriikassa!
”Loss of orthogonality”

- Parannetussa Gram-Schmidt ortogonalisoinnissa projektio otetaan *edelliseen* ortogonalisoituun vektoriin, jolloin siihen tulleet pyöristysvirheet otetaan ortogonalisoinnissa huomioon eivätkä ne kasaudu

MGS Algoritmi: korvaa alkuperäisen vektorijoukon ortonormaalilla

for j=1...n
  for i =1...j-1

$$|v_j\rangle = |v_j\rangle - \frac{\langle v_i|v_j\rangle}{\langle v_i|v_i\rangle}|v_i\rangle$$

  end

$$|v_j\rangle = \frac{|v_j\rangle}{\sqrt{\langle v_j|v_j\rangle}}$$ normitus

end

”Reorthogonalization”:
Joskus ortogonalisointi pitää tehdä uudelleen,
Jos tulos ei ole
riittävän tarkasti ortogonaalinen:
MGSR algoritmi

Muita: Householder, Givens

Tuumi mitä näissä algoritmeissa tapahtuu, jos yrität ortogonalisoida vektoreita, jotka *eivät* ole lineaaarisesti riippumattomia.

Ominaisarvoyhtälöt: Arnoldi iteraatio

- Periaate:
  1. Lasketaan matriisin *A* alimmat *m* Krylovin vektoria
  2. Ortogonalisoidaan ne => Kantavektorit $\boldsymbol{q}_k$ niistä koottu matriisi *Q*
  3. Projisoidaan matriisi *A* Krylovin aliavaruuteen, saadaan pienempi ns. Hessenbergin matriisi *H* (jos ajattelee alkuperäisen matriisin olevan operaattorin esitys täydellisessä kannassa, niin tämä on operaattorin matriisiesitys hyvin valitussa, epätäydellisessä kannassa)

$$H = \begin{pmatrix} h_{11} & h_{12} & \dots & & h_{1n} \\ h_{21} & h_{22} & \dots & & h_{2n} \\ 0 & h_{32} & \dots & h_{3\,n-1} & h_{3n} \\ & \vdots & & & \\ 0 & 0 & \dots & h_{n\,n-1} & h_{nn} \end{pmatrix} \qquad A \approx QHQ^{\dagger}$$

  4. Lasketaan *H*:n ominaisarvot ja -vektorit (Ritzin arvot) tavanomaisilla menetelmillä

- Tyypillisesti Ritzin ominaisarvot suppenevat kohti *A*:n ominaisarvojen ääripäiden arvoja
- Käyttö lineaaristen yhtälöryhmien ratkomiseen: *GMRES* (G*eneralized Minimal RESidual method)*

Ominaisarvoyhtälöt: Lanczos iteraatio

- Helpotetaan Arnoldin iteraatiota ja tehdään algoritmi erityisesti **Hermiten matriiseille**
- Välitulos on *A*:ta pienempi tridiagonaalimatriisi (lävistäjällä ja sen vieressä nollasta poikkeavia elementtejä), jonka ominaisarvot on nopea ja helppo löytää tavallisilla menetelmillä

Arnoldi algoritmi

Wikipedia:

- Start with an arbitrary vector q_1 with norm
- Repeat for $k = 2, 3, \ldots$
  - $q_k \leftarrow A q_{k-1}$
  - **for** j from 1 to $k - 1$
    - $h_{j,k-1} \leftarrow q_j^* q_k$
    - $q_k \leftarrow q_k - h_{j,k-1} q_j$
  - $h_{k,k-1} \leftarrow \| q_k \|$
  - $q_k \leftarrow \frac{q_k}{h_{k,k-1}}$

iteraatio lopetetaan, kun $\boldsymbol{q_k}$ on nollavectori

Tästä kootaan ns. ylempää Hessenbergin muotoa oleva matriisi *H*

$$H = \begin{pmatrix} h_{11} & h_{12} & \ldots & & h_{1n} \\ h_{21} & h_{22} & \ldots & & h_{2n} \\ 0 & h_{32} & \ldots & h_{3\,n-1} & h_{3n} \\ & \vdots & & & \\ 0 & 0 & \ldots & h_{n\,n-1} & h_{nn} \end{pmatrix}$$

tämän ominaisarvoista **osa**
approksimoi *A*:n ominaisarvoja

Lanczos algoritmi

Wikipedia:

Algorithm Lanczos

$v_1 \leftarrow$ random vector with norm 1.

$v_0 \leftarrow 0$

$\beta_1 \leftarrow 0$

Iteration: for $j = 1, 2, \cdots, m$

$w_j \leftarrow Av_j - \beta_j v_{j-1}$

$\alpha_j \leftarrow (w_j, v_j)$

$w_j \leftarrow w_j - \alpha_j v_j$

$\beta_{j+1} \leftarrow \|w_j\|$

$v_{j+1} \leftarrow w_j / \beta_{j+1}$

return

- "←" is a loose shorthand for "changes to". For instance, "*largest* ← *item*" means that the value of *largest* changes to the value of *item*.
- "**return**" terminates the algorithm and outputs the value that follows.

Note that (x,y) represents the dot product of vectors x and y here.

After the iteration, we get the α_j and β_j which constructs a tridiagonal matrix

$$T_{mm} = \begin{pmatrix} \alpha_1 & \beta_2 & & & & 0 \\ \beta_2 & \alpha_2 & \beta_3 & & & \\ & \beta_3 & \alpha_3 & \ddots & & \\ & & \ddots & \ddots & \beta_{m-1} & \\ & & & \beta_{m-1} & \alpha_{m-1} & \beta_m \\ 0 & & & & \beta_m & \alpha_m \end{pmatrix}$$

tämän ominaisarvoista **osa** approksimoi *A*:n ominaisarvoja

Lanczos algoritmi

Ongelma:

Kantavektorien ortogonaalisuus häviää ennemmin tai myöhemmin numeeristen virheiden kasautuessa

- Vääriä ominaisarvoja *(spurious eigenvalues)*
  jotkut Ritz-ominaisarvot eivät vastaa mitään oikeaa ominaisarvoa
  katoavat usein iteraation edetessä

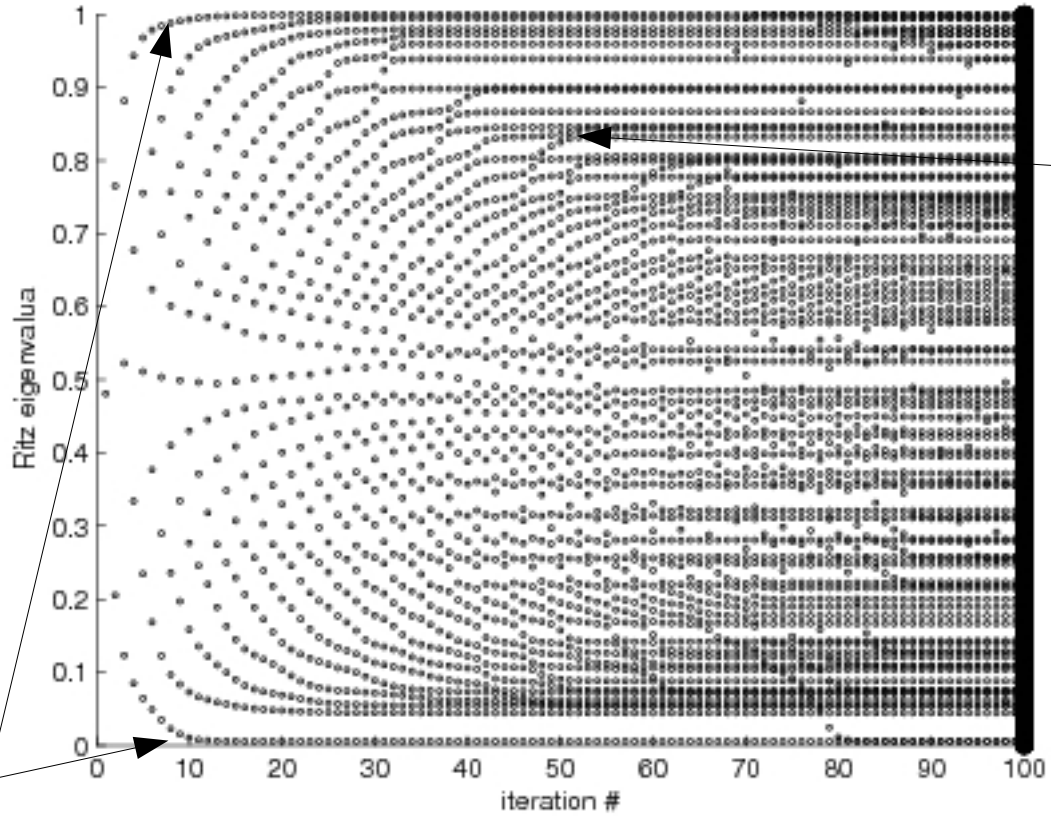
On siis pakko joko

1) estää ortogonaalisuuden kato
2) palauttaa ortogonaalisuus kun kantavektorit on luotu
3) poistaa väärät ominaisarvot - kun ne on tunnistettu

Ortogonalisointi saattaa olla hidas operaatio (esim. elektronirakennelaskuissa)

Ominaisarvoyhtälöt: Ritzin ominaisarvot

- Arnoldi ja Lanczos tuottavat ns. Ritz'in ominaisarvoja, joista osa suppenee kohti todellisia ominaisarvoja.
  Ritz'in ominaisarvojen joukossa on katoavia ominaisarvoja, jotka lopulta sulautuvat muihin



Seassa on
vääriä ominaisarvoja

suurin ja pienin ominaisarvo suppenevat nopeimmin

- Vektoreiden ortogonaalisuus katoaa iteraation edetessä ellei ortogonalisointia tehdä uudelleen

Schrödingerin yhtälö: B-spline ratkaisu

Oletus: potentiaali riippuu vain etäisyydestä, *V(r)*
Halutaan alimmat energian ominaisarvot ja vastaavat ominaisfunktiot

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

Sijoitetaan $\psi = \mathcal{R}(r)Y(\theta,\phi)$
Osoittautuu, että kulmaosan ratkaisuna on tunnetut palloharmoniset funktiot $Y_{lm}(\theta\phi)$
l, m ovat kvanttilukuja (kokonaislukuja, kullakin lukuparilla saadaan yksi ratkaisu)
Radiaalinen (*r* -riippuva osa) on differentiaaliyhtälön

$$-\frac{\hbar^2}{2m}[\mathcal{R}''(r) + \frac{2}{r}\mathcal{R}'(r)] + \left[V(r) + \frac{\hbar^2}{2m}\frac{l(l+1)}{r^2}\right]\mathcal{R}(r) = E\mathcal{R}(r)$$

ratkaisu. Valitaan sopiva yksikköjärjestelmä $\hbar = 1; m = 1$ ja haetaan ratkaisua

$$\mathcal{R}(r) = \frac{R(r)}{r}$$

$$-R''(r) + \left[V(r) + \frac{l(l+1)}{r^2}\right]R(r) = ER(r)$$

reunaehto: $R(0) = 0$

Etsitään ratkaisu käyttäen funktiokantana **B-spline kantaa**
tehtävänä on siis ratkaista kertoimet c_i , joilla

$$R(r) = \sum_{i=1}^{n} c_i B_i(r)$$

Schrödingerin yhtälö: B-spline ratkaisu

Differentiaaliyhtälöstä matriisiyhtälöksi

sijoitetaan $R(r) = \sum_{i=1}^{n} c_i B_i(r)$ radiaaliseen Schrödingerin yhtälöön,

$$\sum_j \left\{ -B_j''(r) + \left[V(r) + \frac{l(l+1)}{r^2} \right] B_j(r) \right\} c_j = E \sum_j B_j(r) c_j$$

kerrotaan molemmat puolet vasemmalta $B_i(r)$:llä ja integroidaan yli ***r***:n,
summan ja integroinnin järjestystä saa vaihtaa (integroidaan summa termi termiltä)

$$\sum_j \underbrace{\int_0^{r_{max}} dr B_i(r) \left\{ -B_j''(r) + \left[V(r) + \frac{l(l+1)}{r^2} \right] B_j(r) \right\}}_{H_{ij}} c_j = E \sum_j \underbrace{\int_0^{r_{max}} dr B_i(r) B_j(r) c_j}_{S_{ij}}$$

eli

$$\sum_j H_{ij} c_j = E \sum_j S_{ij} c_j$$

Ratkaistava yhtälö matriisimuodossa on siis ominaisarvoyhtälö

$$Hc = ESc$$

r_{max} on laskennallista syistä valittava äärellinen katkaisuetäisyys.

Sen vaikutus tuloksiin nähdään kohta.

Schrödingerin yhtälö: B-spline ratkaisu

Matriisielementit

Lasketaan radiaalisen yhtälön matriisielementit B-spline kannassa, eli integraalit

$$H_{ij} = \int_0^{r_{max}} dr B_i(r) \left\{ -B_j''(r) + \left[V(r) + \frac{l(l+1)}{r^2} \right] B_j(r) \right\}$$

$$S_{ij} = \int_0^{r_{max}} dr B_i(r) B_j(r)$$

Hamiltonin matriisi H_{ij} kannattaa laskea paloittain,

	integrandi
$(H_1)_{ij} = -\int_0^{r_{max}} dr B_i(r) \frac{d^2}{dr^2} B_j(r)$	polynomi
$(H_2)_{ij} = \int_0^{r_{max}} dr B_i(r) V(r) B_j(r)$	ei polynomi
$(H_3)_{ij} = \int_0^{r-max} dr B_i(r) \frac{l(l+1)}{r^2} B_j(r)$	ei polynomi
$S_{ij} = \int_0^{r_{max}} dr B_i(r) B_j(r)$	polynomi

Tästä näkyy yksi B-spline kannan vahvuus: puolet integraaleista on polynomin integrointia, joka voidaan laskea erittäin tarkasti* käyttäen **Gauss-Legendre kvadratuuria**.
Myös muut integraalit voi laskea hyvin tarkasti samalla tavalla.

* Konetarkkuudella, eli kaksinkertaisen tarkkuuden luvuilla virhe on luokkaa 10^{-16}

Schrödingerin yhtälö: B-spline ratkaisu

Miten korkean asteen kvadratuuri tarvitaan eli montako Gauss-Legendre pistettä ?
Tarkastellaan lähemmin integraalin

$$(H_1)_{ij} = -\int_0^\infty dr B_i(r) \frac{d^2}{dr^2} B_j(r)$$

laskemista.

- *r*-alue on jaettu solmupisteiden t_i välisiin segmentteihin
- kertaluvun *k* B-spline kantapolynomeista $B_i(r)$ vain *k* kappaletta on nollasta poikkeavia pisteessä *r.*
- kukin $B_i(r)$ ulottuu vain *k* peräkkäiseen segmenttiin

Solmupisteiden t_i ja t_{i+1} välissä on vain tietyt nollasta poikkeavat kantapolynomit $B_{m+1}(r)$ ... $B_{m+k}(r)$.

- polynomin $B_i(r)$ aste on *k-1*, missä *k* on valitun B-spline kannan kertaluku

$\frac{d^2}{dr^2} B_j(r)$ on astetta *k-3* oleva polynomi

$B_i(r) \frac{d^2}{dr^2} B_j(r)$ on astetta *2k-4* oleva polynomi

tarvitaan siis kvadratuuri, joka osaa integroida astetta *2k-4* olevia polynomeja.

n pisteen Gauss-Legendre kvadratuuri integroi tarkasti astetta *2n+1* olevia polynomeja

Schrödingerin yhtälö: B-spline ratkaisu

polynomin aste

$$(H_1)_{ij} = -\int_0^{r_{max}} dr B_i(r) \frac{d^2}{dr^2} B_j(r)$$ *2k-4*

$$S_{ij} = \int_0^{r-max} dr B_i(r) B_j(r)$$ *2k-2*

Kummankin saa siis tarkasti *n=k* -pisteen Gauss-Legendre kvadratuurilla.
Integraalissa

$$(H_3)_{ij} = \int_0^{r-max} dr B_i(r) \frac{l(l+1)}{r^2} B_j(r)$$

integroidaan polynomien osamäärää. Voidaan kuitenkin osoittaa, että tämäkin voidaan integroida tarkasti kun käytetään hieman useampaa Gauss-Legendre pistettä. Sama pätee potentiaalitermille

$$(H_2)_{ij} = \int_0^{r_{max}} dr B_i(r) V(r) B_j(r)$$

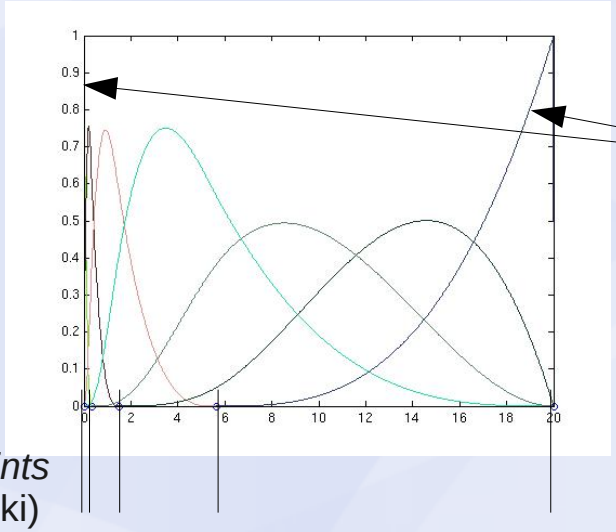
jos esimerkiksi *V(r)* on Coulombin potentiaali, eli *~1/r* .

=> Matriisielementit voidaan laskea hyvin tarkasti

Reunaehto: *R(0)=0* voidaan toteuttaa jättämällä pois ensimmäinen ja viimeinen $B_i(r)$, sillä vain ne ovat nollasta poikkeavia ensimmäisessä ja viimeisessä pisteessä. Käytännössä matriiseista *H* ja *S* jätetään silloin ensimmäinen ja viimeinen rivi ja sarake pois (esim. silloin kun ominaisarvoyhtälöä ratkaistaan).

Schrödingerin yhtälö: B-spline ratkaisu

Eksponentiaalisesti jakautuneet pisteet: pisteitä tiheämmässä *r=0* lähellä
breakpoints = 5 pisteellä tulee tällaiset B-spline kantafuntiot
k=4, kukin funktio $B_i(x)$ poikkeaa nollasta vain pisteiden t_i ja t_{i+k-1} välissä



breakpoints
(esimerkki)

1. ja viimeinen B-spline kantafunktio
ei mene nollaan rajoilla, toisin kuin *R(r)*
=> älä käytä niitä (tai aseta solmupisteitä)

Schrödingerin yhtälö: B-spline ratkaisu

Esimerkki: Vetyatomi

Vilkaistaan aluksi miten Matlabin B-spline toimii. Tässä muutamia funktioita Spline Toolbox kokoelmasta:

SPMAK(KNOTS,COEFS) puts together a spline from the knots and coefficients.

[KNOTS,COEFS,N,K,D] = SPBRK(SP) breaks the B-form in SP into its parts and returns as many of them as are specified by the output arguments.

SPAPI Spline interpolation.

exponentiaalisesti jakautuneet solmupisteet:

n	energia	tarkka $-1/n^2 * 0.5$
1	-0.500000000000004363176	-0.5
2	-0.125000000000000915934	-0.125
3	-0.055555555555555805741	-0.055555555...
4	-0.031249999999958201144	-0.03125
5	-0.019999997150367706714	-0.002
6	-0.013868477731681715 80	-0.013888888...
7	-0.009596374993026447 06	-0.010204081...
8	-0.004662836937221919 93	-0.0078125...
9	0.001656069944172800 09	-0.00617283...

väärät desimaalit punaisella

Tuloksen huononeminen suuremmilla energioilla ei johdu laskentatarkkuuden huononemisesta, vaan siitä, että ”äärettömyysraja” rmax ei riitä suuren energian aaltofunktioille.

Matriisiyhtälö pakottaa kaikki aaltofunktiot nollaan rajalla rmax, koska mikään kantafunktio ei ulotu sen kauemmas.

Rajaa voi siirtää kauemmas, mutta samalla matriisien koko kasvaa ja laskenta hidastuu.

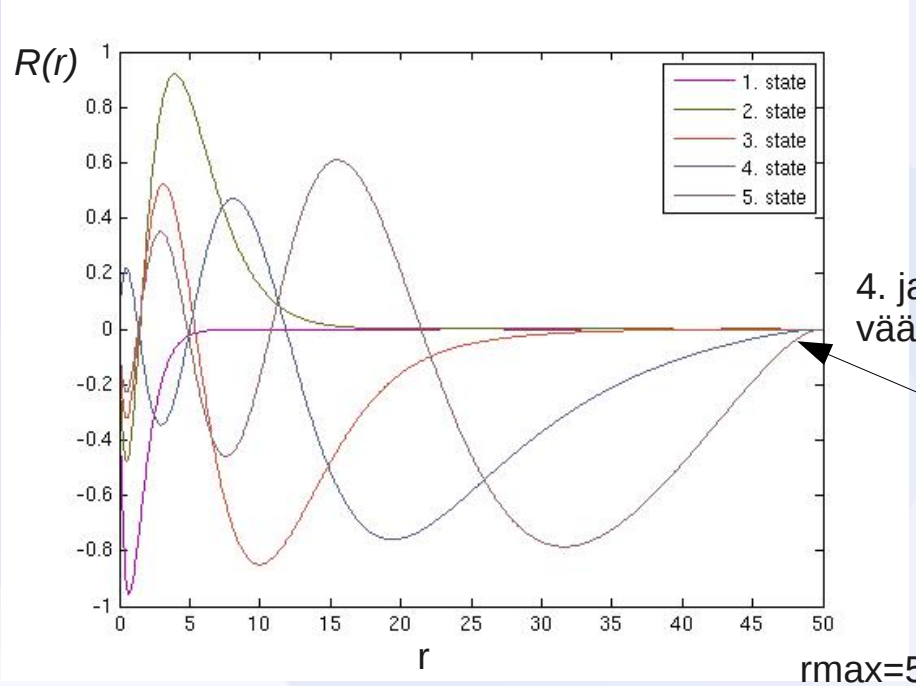
Matriisia ei voi pienentää harventamalla solmupisteitä tai laskentatarkkuus huononee kauttaaltaan.

Schrödingerin yhtälö: B-spline ratkaisu

k=9 , rmax = 50 , nbreak = 20 , eksponentiaaliset pisteet

```
i=   1  E =  -0.49999999999406219420  Exact =  -0.50000000000000000000  error = 5.93781e-12
i=   2  E =  -0.12499999996774111000  Exact =  -0.12500000000000000000  error = 3.22589e-11
i=   3  E =  -0.05555554778885649081  Exact =  -0.05555555555555555247  error = 7.76670e-09
i=   4  E =  -0.03120433749952353925  Exact =  -0.03125000000000000000  error = 4.56625e-05
i=   5  E =  -0.01786476394349107225  Exact =  -0.02000000000000000042  error = 2.13524e-03
i=   6  E =  -0.00226589537948422397  Exact =  -0.01388888888888888812  error = 1.16230e-02
i=   7  E =   0.01850662010568454238  Exact =  -0.01020408163265306041  error = 2.87107e-02
i=   8  E =   0.04411072241323033688  Exact =  -0.00781250000000000000  error = 5.19232e-02
i=   9  E =   0.07467049717184717772  Exact =  -0.00617283950617283916  error = 8.08433e-02
i=  10  E =   0.11097078778181203307  Exact =  -0.00500000000000000010  error = 1.15971e-01
```

OK kelvottomia



rmax=50

4. ja 5. *etc.* tilat menevät pahasti väärin, koska *rmax* on liian pieni

aaltofunktio leikkautuu nollaksi liian aikaisin

Simulaatiot

- Simulaatiotyypit
  (Juha Merikosken mukaan) :
  - Deterministiset simulaatiot – liikeyhtälöiden numeerinen ratkaisu, eli lasketaan ratoja
    - klassinen molekyylidynamiikka (MD) – atomien klassinen liike
      - biomolekyylien liike liuoksessa
      - tähti- ja galaksijoukkojen dynamiikka (relativistinen)
    - kvantti-MD – elektronitiheys Shcrödingerin yhtälöstä
  - Stokastiset simulaatiot
    - Monte Carlo menetelmät
      - stokastinen rajapintamalli
      - ferromagneetin tasapainotilan simulaatio
      - (alkeishiukkasten) sirontaprosessien simulaatio
      - kvantti Monte Carlo (*QMC)* aaltofunktion stokastinen kuvaus
    - Langevin yhtälö ym. semideterministiset menetelmät
      - lämpökylpy; stokastinen termostaatti MD:ssä
- Esim.
  FYSA241/k1 on Ising-mallin Matlab simulaatio, koodin saat sivulta
  http://users.jyu.fi/~veapaja/Ising-simulaatio

Molekyylidynamiikka (Molecular Dynamics, MD)

- Asetetaan hiukkasten välinen vuorovaikutus, pannaan ne simulaatiokoppiin ja aletaan laskea ratoja Newtonin liikeyhtälöistä (Hamiltonin liikeyhtälöistä)

- Ratojen tarkka laskeminen on mahdotonta

  - Pyöristysvirheet saavat lopulta differentiaaliyhtälöiden ratkaisut vääristymään

  - Ei tarvitakaan tarkkoja ratoja, kunhan hiukkaset liikkuvat *keskimäärin* oikein
    Esim. systeemin keskim. energia ei saa karata, se voi kyllä hiukan heilua

- Integraattori: algoritmi joka ”integroi” liikeyhtälöitä, eli ratkoo likimäärin k.o. differentiaaliyhtälöitä
  Symplektinen integraattori (*symplectic int.*): integraattori, joka pitää Hamiltonin liikeyhtälöt voimassa

- Algoritmeja: - **Leap frog** Lasketaan nopeudet ja kiihtyvyydet eri ajanhetkillä, vuoron perään
  - **Verlet** Kirjoita liikeyhtälö ajassa eteen- ja taaksepäin meneville liikkeille ja laske ne yhteen

$$\mathbf{r}_i(t+\Delta t) = \mathbf{r}_i(t) + \mathbf{v}_i(t)\Delta t + \frac{1}{2}\mathbf{a}_i(t)(\Delta t)^2$$

$$+ \quad \mathbf{r}_i(t-\Delta t) = \mathbf{r}_i(t) - \mathbf{v}_i(t)\Delta t + \frac{1}{2}\mathbf{a}_i(t)(\Delta t)^2$$

$$\mathbf{r}_i(t+\Delta t) = 2\mathbf{r}_i(t) - \mathbf{r}_i(t-\Delta t) + \mathbf{a}_i(t)(\Delta t)^2 \qquad + O(\Delta t^4)$$

huono puoli: ei nopeuksia

- Velocity Verlet

Kullekin hiukkaselle

$$\mathbf{r}(t+\Delta t) = \mathbf{r}(t) + \mathbf{v}(t)\Delta t + \frac{1}{2}\mathbf{a}(t)(\Delta t)^2$$

$$\mathbf{v}(t+\Delta t/2) = \mathbf{v}(t) + \frac{1}{2}\mathbf{a}(t)$$

$$\mathbf{a}(t+\Delta t) = F(r(t+\Delta t))/m = -\nabla V(\mathbf{r}(t+\Delta t)/m$$

$$\mathbf{v}(t+\Delta t) = \mathbf{v}(t+\Delta t/2) + \frac{1}{2}\mathbf{a}(t+\Delta t)$$

Monte Carlo simulaatio: Variaatio Monte Carlo (VMC) (*Variational Monte Carlo)*

- Arvataan monenkappaleen aaltofunktio ja halutaan laskea sen tuottamat fysikaaliset ominaisuudet
  Esim. ^{4}He nesteen perustila on likimain muotoa

$$\psi(\mathbf{R}) = \psi(\mathbf{r}_1, ...\mathbf{r}_N) = \prod_{(i\neq j)=1}^{N} e^{-\frac{1}{2}\left(\frac{\beta}{r_{ij}}\right)^{\mu}} \qquad \beta \approx 3.2 \quad \mu = 4$$

Tässä aaltofunktion päätehtävä on pitää atomit etäisyyden ~2.5 Å päässä, koska niillä on kova kuori.
Perustilan energia *E* on *Nd* -ulotteinen integraali, *N* hiukkasta, *d* koordinaattia:

$$E = \int d\mathbf{R}\psi(\mathbf{R})\hat{H}\psi(\mathbf{R}) / \int d\mathbf{R}|\psi(\mathbf{R})|^2 \qquad \hat{H} = -\frac{\hbar^2}{2m}\sum_{i=1}^{N} -\nabla_i^2 + V(r)$$

Pannaan *N* atomia satunnaisesti laatikkoon, jonka koon voi laskea halutusta tiheydestä.
Käytetään ***Metropolis algoritmia*** tuottamaan aaltofunktion todennäköisyysjakaumaa vastaavia koordinaatteja ***R*** ja ”mitataan” kunkin tilanteen ns. *lokaali energia $E_L(R)$;*

$$E_L(\mathbf{R}) = \hat{H}\psi(\mathbf{R})/\psi(\mathbf{R})$$ simulaatiosta mitattava suure

Helposti nähdään, että lokaalin energian keskiarvo jakaumasta $|\psi(\mathbf{R})|^2$ on energia.
Aaltofunktiota ei tarvitse normittaa!

- Oleellista kvantti Monte Carlossa on, ettei hiukkasten klassisia ratoja - tietenkään- lasketa, vaan tartutaan aaltofunktion todennäköisyystulkintaan ja tuotetaan koordinaatteja ***R*** aaltofunktiosta.

Metropolis algoritmi

(Metropolis N, Rosenbluth A W, Rosenbluth M N, Teller A H and Teller E, J. Chem. Phys. 21, 1087. 1953)

- Katsotaan johdannoksi mitä tarkoitetaan ns. ***detailed balance*** -ehdolla.
  Detailed balance on riittävä, muttei välttämätön, ehto sille, että systeemi on tasapainossa. Se kertoo miten kahden tilan todennäköisyydet ja näiden tilojen välisten siirtymien todennäköisyydet riippuvat toisistaan. Periaate on, että siirtymäprosessit ovat vastakkaisia ja tasapainossa kumoavat toisensa.

  Tilan *i* paino (normittamaton todennäköisyys) on *W(i)* ja tilan *j* paino on *W(j)*.
  Todennäköisyys, että siirrytään tilasta *i* tilaan *j* on *P(i->j)* ja päinvastaisen siirron todennäköisyys on *P(j->i).* Tasapainoehto on *W(i) P(i->j) = W(j) P(j->i)*.

  Muista: tässä on prosessi ja ***täsmälleen vastakkainen*** prosessi,
  eli jos molemmat tehdään päädytään samaan tilaan mistä aloitettiin.

- *P(i->j)* on siirtymisen todennäköisyys ja koostuu yleensä kahdesta osasta:
  todennäköisyys, että siirtoa *yritetään* (*"Try"*) on *T(i->j*
  todennäköisyys, että siirto *hyväksytään* (*"Accept"*) o*n A(i->j*)
  silloin *P(i->j) = T(i->j)A(i->j*) todennäköisyys, että siirtoa *yritetään ja se hyväksytään*

- *A(i->j)* on se minkä me haluaisimme tietää: olemme tilassa i, ehdotamme tilaa j, ja kysymme siirrymmekö vai emme? Selvästi pitäisi olla voimassa

$$\frac{A(i \to j)}{A(j \to i)} = \frac{T(j \to i)W(j)}{T(i \to j)W(i)}$$

  detailed balance-ehdolla päästään tähän asti

  mutta tämä antaa vain kahden *A*:n suhteen.

- Metropolis (tarkemmin: Metropolis-Hastings) algoritmin valinta:

$$A(i \to j) = min\left[1, \frac{T(j \to i)W(j)}{T(i \to j)W(i)}\right]$$

(Hastings lisäsi kaavaan yritystodennäköisyyden *T*)

Todistetaan, että Metropolis-Hastings valinta

$$A(i \to j) = min\left[1, \frac{T(j \to i)W(j)}{T(i \to j)W(i)}\right]$$

toteuttaa *detailed balance* -ehdon

tapaus (i) *T(j->i)W(j)>T(i->j)W(i)* , suhde *>1*

$$A(i \to j) = min\left[1, \frac{T(j \to i)W(j)}{T(i \to j)W(i)}\right] = 1$$

$$A(j \to i) = min\left[1, \frac{T(i \to j)W(i)}{T(j \to i)W(j)}\right] = \frac{T(i \to j)W(i)}{T(j \to i)W(j)}$$

$$\Rightarrow \frac{A(i \to j)}{A(j \to i)} = \frac{T(j \to i)W(j)}{T(i \to j)W(i)}$$ OK

tapaus (ii) *T(j->i)W(j)<T(i->j)W(i)* , suhde *<1*

$$A(i \to j) = min\left[1, \frac{T(j \to i)W(j)}{T(i \to j)W(i)}\right] = \frac{T(j \to i)W(j)}{T(i \to j)W(i)}$$

$$A(j \to i) = min\left[1, \frac{T(i \to j)W(i)}{T(j \to i)W(j)}\right] = 1$$

$$\Rightarrow \frac{A(i \to j)}{A(j \to i)} = \frac{T(j \to i)W(j)}{T(i \to j)W(i)}$$ OK

- Muitakin vaihtoehtoja *detailed balance* -ehdon toteuttamiseen on, mm. ***lämpökylpy:***

$$A(i \to j) = \frac{T(j \to i)W(j)}{T(j \to i)W(j) + T(i \to j)W(i) +}$$

- Miksen käyttänyt todennäköisyyttä olla tilassa *i*, *P(i)*, vaan puhuin tilan ”painosta” *W(i)* ? Koska todennäköisyydet *P(i)* saadaan normittamalla painot *W(i)* ja se on turha operaatio!

  Esim. Variaatio Monte Carlossa *(VMC)* aaltofunktiota ei normiteta, se antaa tilan ”painon”.

- Yritystodennäköisyys *T* on hyvin tarpeellinen otus, koska usein kaikkia vaihtoehtoja ei yritetä yhtä todennäköisesti.

  Esim. Molekyyli ABC on pinnalla ja se voi hajota joko ABC -> AB + C tai – erittäin harvoin - ABC -> A + BC. Simulaatiossa pitää päättää kumpaa yritetään sillä kertaa, vaikkapa yhdeksän kertaa kymmenestä yritetään ensimmäistä reaktiota ja kerran toista, *T(1. reaktio) = 0.9* ja *T(2. Reaktio) = 0.1* - nämä ovat täysin mielivaltaisia ”tehokkuutta” parantavia valintoja, jotka pitää ottaa Metropolis-Hastings -kaavan mukaisesti huomioon päätettäessä hyväksymisestä *A*.

- Metropolis-algoritmia on helppo soveltaa esim. termodynaamisten keskiarvojen laskemiseen

$$\langle x \rangle = \frac{\sum_i x_i e^{-E_i/(k_B T)}}{\sum_i e^{-E_i/(k_B T)}}$$

  Tässä tilan *i* paino on $W(i) = \exp(E_i/(k_B T))$

  Itse jakauman normitus on partitiofunktio *Z*, joka on vaikea laskettava – mutta sitä ei *tarvitsekaan* laskea ! Painoja ei tarvitse normittaa !

Aaltofunktiot: Metropolis algoritmi ja Markovin ketju

- Metropolis algoritmilla tuotettu Markovin ketju on vastaus kysymykseen

  ”***Miten tuottaa satunnaislukuja, joilla on haluttu jakauma”***

  Markovin ketju on sellainen pisteiden ketju, joka *asymptoottisesti* (=ääretön ketju) tuottaa halutun jakauman.

- Kaava satunnaiskävelyyn *(random walk),* joka tuottaa ns. **Markovin ketjun** (*Markov chain) :*
  (kirjoitettu aaltofunktiota ajatellen)

  1) Valitaan aloituskoordinaatit ***R***
  2) Lasketaan ehdotus uusiksi koordinaateiksi Gaussin jakaumalla aiemmista paikoista lähtien
  $$\mathbf{R'} = \mathbf{R} + \boldsymbol{\xi}$$
  missä $\boldsymbol{\xi}$ on vakio kertaa Gaussin jakaumasta otettu satunnaisvektori

  3) Hyväksytään siirros, jos aaltofunktioiden suhde $|\psi(\mathbf{R}')|^2/|\psi(\mathbf{R})|^2 > 1$, jos ei, tulkitaan suhde todennäköisyytenä ja verrataan sitä satunnaislukuun $r = U[0,1]$: jos suhde on suurempi hyväksytään siirros. Jos siirrosta ei hyväksytä $\mathbf{R}' = \mathbf{R}$, eli vanha paikka ”lasketaan uudelleen” .

  4) Aseta ***R=R'*** .

  5) Mittaa halutut suureet koordinaateista ***R***
  - energia, tiheys, parikorrelaatiofunktio, ...

  6) Jos haluat koota enemmän dataa (eli pienentää stokastista virhettä) palaa kohtaa 2)

  7) Laske mitattujen suureiden keskiarvot ja stokastiset virhearviot.

- Parannus arvattuun aaltofunktioon: esim. diffuusio Monte Carlo (DMC) - kirjoitetaan Schrödingerin yhtälö imaginaariselle ajalle => diffuusioyhtälö, josta saadaan todellinen perustilan energia. Periaate lyhyesti: aaltofunktiota projisoidaan

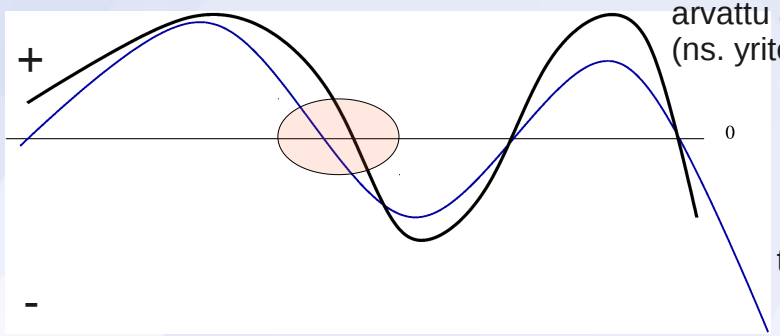
$$e^{-\tau H}\psi(\mathbf{R}) \rightarrow \phi_0(\mathbf{R}), \quad \text{kun } \tau \rightarrow \infty$$

projisointi tehdään statistisesti

Projisoitua aaltofunktiota ei enää osata kirjoittaa suljettuun muotoon, mutta lasku tuottaa koordinaatteja ***R*** , joiden jakauma on perustilan aaltofunktio. Niistä voidaan laskea odotusarvoja.

- Diffuusio Monte Carlo ei ole käyttökelpoinen ilman yriteaaltofunktiota $\varphi_T(\mathbf{R})$; se tuottaa pisteitä ***R*** todennäköisyysjakaumasta $\varphi_T(\mathbf{R})\phi_0(\mathbf{R})$ Importance sampling

- Pulma: bosonien perustila $\phi_0(\mathbf{R})$ on positiivinen, mutta fermionien ei – sillä on pakko olla nollakohtia ja erimerkkisiä alueita (paulin kieltosääntö!). Näissä tilanteissa tuloa $\varphi_T(\mathbf{R})\phi_0(\mathbf{R})$ ei voi suoraan tulkita todennäköisyytenä.
  Osittainen parannus: arvataan (taas pitää arvata...) missä aaltofunktio on negatiivinen, pikku kirjanpidolla saadaan tehtyä positiivisena pysyvä $\varphi_T(\mathbf{R})\phi_0(\mathbf{R})$
  Mutta: huono arvaus tuottaa liian korkean energian, luonto arvasi paremmin.



väärin arvattu nollakohta tuottaa virhettä

Statistinen riippumattomuus ja virhearvio

Markovin ketjussa seuraava piste riippuu edellisestä – tulee korreloitunut pistejoukko, eikä virheen arvioinnissa peräkkäisiä otoksia voi olettaa satunnaisiksi.

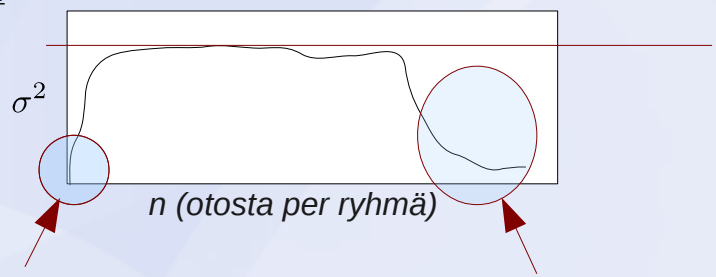
- Pisteet x_i, jotka ovat kyllin monen askeleen päässä toisistaan eivät ole korreloituneita. Niistä saaduista otoksista $f(x_i)$ voi suoraan laskea statistisen virhearvion kaavalla

$$\sigma_I^2 \approx \sigma_{I_N}^2 = \frac{1}{N}\left[\frac{1}{N}\sum_{i=1}^{N} f(x_i)^2 - \left(\frac{1}{N}\sum_{i=1}^{N} f(x_i)\right)^2\right]$$

paljon otoksia menee hukkaan!

- ***Block averaging:*** Jakamalla otokset *n* otoksen ryhmiin ovat ryhmien keskiarvot statistisesti riippumattomia, kunhan *n* on kyllin suuri. Otoksia ei hukata, kaikki käytetään.
- Paljonko on ”kyllin”? Raja on pakko hakea kokeilemalla ja vertaamalla saatua virhearviota. Jonkin *n*:n arvon yläpuolella σ^2 ei enää kasva: se on todellinen statistinen virhe.

Esim.



Turhan optimistinen virhearvio; otoksia on paljon, mutta ne eivät ole statistisesti riippumattomia

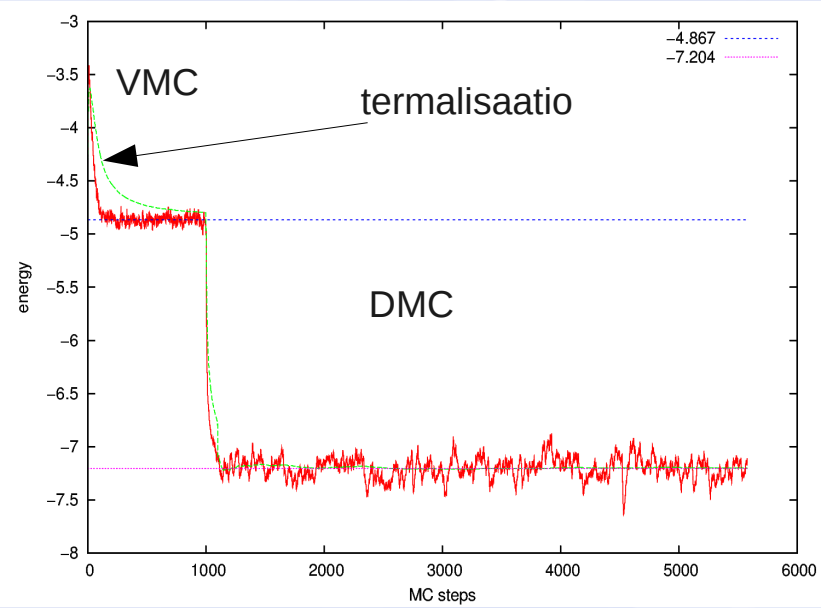
Täällä ryhmien määrä romahtaa, koska otoksia on liian vähän. Lopulta kaikki otokset ovat yhdessä ryhmässä: yksi keskiarvo ja sen ”statistinen virhe” on nolla.

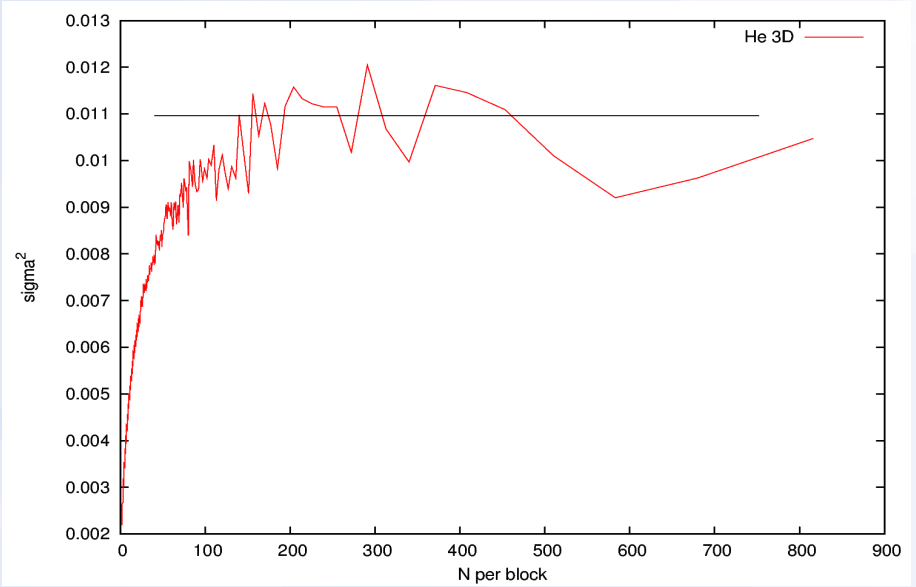
Mitä tehdä jos tasannetta ei ole? Systemaattinen vika tai pitää laskea enemmän otoksia.

- ***Jackknife:*** jätetään satunnaiset *m* (esim *m=1*) otosta pois ja lasketaan keskiarvo, toistetaan sama monta kertaa ja lasketaan virhearvio.

Tämä on esimerkki ns. ***resampling*** algoritmista. Erittäin suosittu.

VMC ja DMC tuloksia: He^4 kvanttineste (T=0)





- Aloitetaan 1000 ”kävelijää” (*walkers*, kukin oma simulaationsa), kussakin 256 atomia
- Annetaan kunkin kehittyä hetken erikseen VMC-algoritmin mukaan ; alun laskeva energiakäyrä
- Mitataan variaatioenergia ja muut halutut suureet
- Kytketään imaginaarinen aikakehitys päälle 1000 askeleen jälkeen
  Imaginaarinen aikakehitys toteutetaan laskemalla kullekin kävelijälle paino, joka on $e^{-\tau(\hat{T}+\hat{V})}$, jota on ja hajotettu osiin $e^{-\tau\hat{T}}$ ja $e^{-\tau\hat{V}}$
  ja paranneltu *importance sampling* -menetelmällä
  => ne kävelijät joilla on pieni paino kuolevat ja ne joilla on suuri monistuvat
  koetetaan pitää keskimäärin 1000 kävelijää elossa