Noter til elementær numerisk regning

Dieter Britz
Kemisk Institut, Aarhus Universitet

19. juli 2010

Foreord

Dette hæfte er vokset fra noter oprindeligt skrevet af forskellige forfattere til kurset DatA, som senere blev til DatK, hovedsageligt Poul Jørgensen (PJ), Niels Christian Nielsen (NCN) og undertegnede. PJ og NCN er de oprindelige forfattere af kapitlet om lineære systemer, mens NCN skrev om Fouriertransformationen. Undertegnede skrev resten, og har siden lagt mere stof til, såsom funktionsevaluering og digital signalbehandling, som forekom mig interessant, selvom der aldrig har været øvelser i disse emner (endnu). Jeg har modificeret kapitlerne skrevet af andre i nogen grad, men de bærer fortsat præg af PJ og NCN.

Der er intet stikordsregister, men det er mit håb, at Indholdsfortegnelsen er tilstrækkelig til at man kan finde det man leder efter uden besvær.

Denne udgave er ret kraftigt renset for fejl og generelt revideret. Den gør derfor tidligere udgaver forældet.

Dieter Britz
Juli 2010

Indhold

Kapitel 1

Introduktion

1.1 Sprog

I kurset **DatK** gives en elementær introduktion til de vigtigste hovedpunkter inden for numerisk regning. På nuværende tidspunkt er der tre computersprog, der er velegnede til videnskabelige beregninger: `Fortran`, `Pascal` og `C++`.

`Fortran` (nu den moderniserede `Fortran 90/95`) har været standardsproget i mange år, og der findes en stor mængde effektive underprogrammer, der er skrevet (og afprøvet) af fagfolk. `Fortran` er det sprog, vi benytter i DatK. `Pascal` og `C++` har det fortrin, at deres oversættere til en PC er meget billigere end dem for `Fortran 90/95`. Dog kan man, hvis man arbejder under `Linux`, helt gratis downloade Intels `Fortran 90/95` oversætter, som virker upåklageligt. `Pascal` vælges ofte til grænsefladecomputere i forbindelse med instrumenter, og `Pascal` anses af nogle for at være et bedre struktureret sprog; men de overser ofte de seneste drastiske forbedringer i `Fortran 90/95`, som nu er næsten lige så elegant et sprog som nogle andre. I den seneste tid er `C++` blevet populær, men efter forfatterens mening er `C++` ikke så let at lære som de to andre sprog. Der findes nu også sproget `Java`, men det er ikke tænkt til numerisk regning. Konklusionen er, at vi bruger `Fortran 90/95` i DatK. Det forholder sig ligesom med talte sprog: Når man har lært det første fremmedsprog, går det nemmere med det næste. Så hvis der er behov for det, kan en `Fortran`-programmør let selv lære sig et andet programmeringssprog.

1.2 Præcision og afrunding

I `Fortran 90/95` (og andre computersprog) er der mange forskellige former for tal eller variabler; der er *f.eks.* hele tal og reelle tal. Ved anvendelse af hele tal er der særlige problemer ved division. Således udregnes 1/2 som 0, det nærmeste, lavere hele tal. Multiplikation af hele tal er altid eksakt, forudsat at begrænsningen til et helt tals størrelse ikke overskrides. Når det angår reelle tal, er der almindeligvis mindre fejl i en computers repræsentation af et tal. Ligesom tallet, der angiver 1/3, eller 0,333... udtrykt med et begrænset antal

decimaler i decimalsystemet, har en lille fejl, således er der almindeligvis også en lille fejl i en computers binære system med dens begrænsede antal binære cifre. Når to sådanne tal påvirker hinanden (addition, multiplikation osv.), vil resultatet også indeholde en lille fejl. Denne kaldes en afrundingsfejl. Et givet reelt tals præcision er bestemt af antallet af binære cifre tilknyttet det pågældende tal. Almindelige maskiner har en præcision på ca. 6-7 decimaltal eller 1 i $10^6 - 10^7$. Afrundingsfejlen er således ca. 10^{-6} eller 10^{-7} gange selve den egentlige værdi. Dette er det vigtigt at huske. *F.eks.* vil to reelle tal næppe nogensinde være ganske ens, selv om vi måske kunne tro det. Hvis vi *f.eks.* adderer 0,01 til sig selv mange gange, vil vi forvente, at summen på et tidspunkt bliver 1 (100 additioner begyndende ved 0). Men på grund af afrunding vil summen imidlertid gå forbi 1 ved en eller anden lille fejl. Så en sløjfe, der skal terminere, når summen = 1, vil aldrig gøre det. Betingelsen må udtrykkes anderledes, *f.eks.* $|sum - 1| < \epsilon$, med ϵ værende et lille tal (mindre end 0,01 i dette tilfælde). Der findes teknikker til at minimere akkumuleringen af afrundingsfejl, og disse vil blive diskuteret i `Fortran`-delen af kurset.

En udmærket gennemgang af tal og deres præcision findes i rapporten af Løfstedt [1].

1.3 Fejlfinding

Det er sjældent, at programmer kører korrekt første gang. Når et program er kommet gennem oversætteren - *dvs.* når det er syntaktisk korrekt - kan der stadig være fejl, eller "lus". Nogle vil dukke op, når programmet kører. Den `Fortran 90/95` compiler, der bruges i DatK fortæller omtrentlig, hvad problemet er, men siger ikke noget om, hvor i programmet fejlen er indtruffet.

En ganske simpel måde at finde fejl på er at indsætte udskriftsmeddelelser på strategiske steder i programmerne. Det kan blot være meddelelser som `"Er nu ved pkt A"` og/eller udskrivning af mistænkte variablers værdier. Dette vil groft indkredse det område, hvori fejlen ligger, og man kan så indsætte flere udskriftsmeddelelser i det afsnit osv. Det er (sædvanligvis) hurtigt at lokalisere en fejl på den måde. Udskriftssætningerne fjernes selvfølgelig igen bagefter, eller - hvis man er usikker - kan man lave dem om til kommentarer, som er lette at reaktivere.

1.4 Taylor-rækker

Dette er en påmindelse om den meget nyttige Taylor-udvikling. Hvis vi har en funktion og kender dens værdi og dens afledte ved x, så er funktionens værdi ved $x + h$ (h normalt lille i forhold til x) givet ved

$$f(x+h) = f(x) + hf'(x) + \frac{h^2}{2!}f''(x) + \frac{h^3}{3!}f'''(x) + \ldots \tag{1.1}$$

hvor $f'(x)$, $f''(x)$ betyder funktionens første, anden, ... afledte med hensyn til x. Ved $x-h$ har vi tilsvarende

$$f(x-h) = f(x) - hf'(x) + \frac{h^2}{2!}f''(x) - \frac{h^3}{3!}f'''(x) + \ldots \tag{1.2}$$

Disse to udtryk fører til nogle nyttige resultater. Bemærk, at disse uendelige serier i praksis altid er trunkeret (skåret af) et sted. Sandsynligvis er h lille, og fejlen, der skyldes trunkering, vil således være lille.

Lige som med en enkelt funktion, kan Taylorrækken anvendes til at udtrykke forskydning af et helt sæt funktioner. Lad sådan et sæt være ligninger for sættet af variablerne $x_1, x_2, \ldots, x_N$ (eller, i vektornotation, $\mathbf{x}$), og vi har N funktioner $f_i, i = 1 \ldots N$. Lad $h_1, h_2, \ldots, h_N$ (eller vektoren $\mathbf{h}$) være et sæt afstande fra et givet sæt $\mathbf{x}$, dvs. sættet $\mathbf{x}+\mathbf{h}$. Der findes nu en Taylorekspansion for hele funktionssættet $\mathbf{x}+\mathbf{h}$ med udgangspunkt fra $\mathbf{x}$:

$$\begin{aligned}
f_1(\mathbf{x}+\mathbf{h}) &= f_1(\mathbf{x}) + h_1\frac{\partial f_1}{\partial x_1} + h_2\frac{\partial f_1}{\partial x_2} + h_3\frac{\partial f_1}{\partial x_3} + \ldots \\
f_2(\mathbf{x}+\mathbf{h}) &= f_2(\mathbf{x}) + h_1\frac{\partial f_2}{\partial x_1} + h_2\frac{\partial f_2}{\partial x_2} + h_3\frac{\partial f_2}{\partial x_3} + \ldots \\
&\vdots \\
f_N(\mathbf{x}+\mathbf{h}) &= f_N(\mathbf{x}) + h_1\frac{\partial f_N}{\partial x_1} + h_2\frac{\partial f_N}{\partial x_2} + h_3\frac{\partial f_N}{\partial x_3} + \ldots
\end{aligned} \tag{1.3}$$

hvor vi kun gik så langt som de første afledte. Det ovenstående kan skrives i den mere kompakte vektor-matrixform,

$$f(\mathbf{x}+\mathbf{h}) = f(\mathbf{x}) + \mathbf{J}\cdot\mathbf{h} \tag{1.4}$$

hvor $\mathbf{J}$, Jacobian er givet ved

$$\mathbf{J} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots \frac{\partial f_1}{\partial x_N} \\ \vdots & & \\ \frac{\partial f_N}{\partial x_1} & \frac{\partial f_N}{\partial x_2} & \cdots \frac{\partial f_N}{\partial x_N} \end{bmatrix} . \tag{1.5}$$

1.5 Fejlorden

Når man approksimerer noget (et integral, en interpolation, et derivat eller en løsning af en differentialligning *mm.*), deler man en kontinuært variabel i diskrete intervaller oftest betegnet med h. Approksimationen er så behæftet med en fejl e, og den er en funktion af h's størrelse. Funktionen er en række af potenser i h

$$e = a\,h^p + b\,h^q + \ldots \tag{1.6}$$

hvor p er den laveste potens hvis koefficient er forskellig fra nul. Eftersom h er normalt en lille størrelse, har de højere potenser i rækken mindre betydning, så p er dominerende. En god approksimation for fejlen er derfor

$$e \approx a\, h^p \ . \tag{1.7}$$

Vi kender normalt ikke koefficienterne, og de er heller ikke vigtige. Det der er vigtigt er den såkaldte **orden** af fejlen, eller potensen der dominerer den, her 2. Man udtrykker fejlen i ligning (1.7) som

$$e = O(h^p) \ . \tag{1.8}$$

Det er denne orden, der er af største interesse i approksimationer. Hvis orden er lig med p, og man prøver at forbedre approksimationen ved *fx.* at halvere h, så ved vi fra (1.8), at det deler fejlen, hvad ellers den var, med 2^p, *osv.*.

Estimering af orden

Sommetider er det usikkert, men godt at vide, hvilken orden en vis fejl har med hensyn til en variabel, som for eksempel et interval. Der er flere metoder for at bestemme denne orden med.

Approksimationer som derivater eller integraler beregnes ofte ud fra datapunkter med visse afstande eller intervaller h. Ordenen af fejlen er så $O(h^p)$, og det gælder at finde p. Lad os kalde den beregnede mængde $u(x, h)$, *dvs.* værdien u ved x beregnet med intervallerne h.

Hvis vi har en eksakt løsning, kender vi fejlene $e(h)$. I så fald beregner vi $e(h)$ for dette interval og igen for αh, som giver den nye fejl $e(\alpha h)$. Oftest vælger man $\alpha = 2$. Vi kan skrive løsningen i formen

$$u = \hat{u} + a\, h^p + b\, h^q + \ldots \tag{1.9}$$

eller, da p dominerer, mere enkelt

$$u = \hat{u} + a\, h^p \tag{1.10}$$

hvor $\hat{u}$ er den sande løsning.

Hvis vi kender $\hat{u}$, er det nemt. Vi udfører to beregninger; den første med interval h og derefter en med $2h$. Så har vi to resultater

$$\begin{aligned} u_h &= \hat{u} + a\, h^p \\ u_{2h} &= \hat{u} + a\, 2^p\, h^p \end{aligned} \tag{1.11}$$

og kan kombinere dem til

$$\frac{u_{2h} - \hat{u}}{u_h - \hat{u}} = \frac{a\, 2^p\, h^p}{a\, h^p} = 2^p \tag{1.12}$$

hvilket giver os p,

$$p = \ln(2^p)/ln2 \ . \tag{1.13}$$

Hvis vi ikke kender den eksakte løsning, er der en anden metode, fundet af Østerby [2], der benytter sig af tre estimater. Ud over beregning med h og $2h$, udfører vi nu en tredje med $4h$. Så gælder

$$u_{4h} = \hat{u} + a\,4^p\,h^p \tag{1.14}$$

og vi kan beregne forholdet

$$\frac{u_{4h} - u_{2h}}{u_{2h} - u_h} = \frac{(4^p - 2^p)\,a\,h^p}{(2^p - 1)\,a\,h^p} \tag{1.15}$$

som atter giver os 2^p.

Har man fundet p, kan man forbedre metoden, og eliminere denne fejlterm, så at fejlordenen rykker op til den næste potens, her q, og derved mindsker man fejlen, ofte betydeligt. Se senere, under Romberg integration, s. 21 og ekstrapolation, s. 33.

1.6 Approksimationer til derivater

Numerisk differentiering er tit af interesse, når vi har at gøre med en kompliceret evaluering af en funktion, som ikke umiddelbart kan differentieres algebraisk. Det kan være en sum af terme eller punkter i en series, som er givet. Vi kan få forskellige approksimationer for både første og anden derivater ud fra Taylorrækkene (1.1) og (1.2). Der er tre forskellige approksimationer til $\mathrm{d}f/\mathrm{d}x$. Bruger vi (1.1), så får vi

$$\frac{\mathrm{d}f}{\mathrm{d}x} \approx \frac{f(x+h) - f(x)}{h} - \frac{h}{2}f''(x) - \ldots \tag{1.16}$$

hvilket af indlysende grunde kaldes en fremad differens. Bruger man i stedet (1.2), får man den baglænse differens

$$\frac{\mathrm{d}f}{\mathrm{d}x} \approx \frac{f(x) - f(x-h)}{h} + \frac{h}{2}f''(x) - \ldots\ . \tag{1.17}$$

Begge er $O(h)$, som kan ses. Bruger vi begge ligninger, og trækker (1.2) fra (1.1), får vi

$$\frac{\mathrm{d}f}{\mathrm{d}x} \approx \frac{f(x+h) - f(x-h)}{2h} + \frac{h^2}{3}f'''(x) - \ldots \tag{1.18}$$

hvilket ses at være $O(h^2)$, da termen i h dropper ud. Denne form er den centrale differens og er den mest nøjagtige af de tre.

Ved at lægge de to Taylorrækker sammen, får vi et nyt resultat, nemlig en approksimation for andet derivat:

$$\frac{\mathrm{d}^2 f}{\mathrm{d}x^2} \approx \frac{f(x+h) - 2f(x) + f(x-h)}{h^2} + \frac{h^2}{4}f'''(x) - \ldots \tag{1.19}$$

som er den centrale formel for andet derivat og også er $O(h^2)$.

Det er muligt at få bedre, *dvs.*, højre-orden, approksimationer for disse derivater ved at bruge flere punkter; man kan endda beregne sådanne flerpunktsderivater på punkter med forskellige intervaller imellem dem; men det går ud over rammen her.

Kapitel 2

Rodsøgning

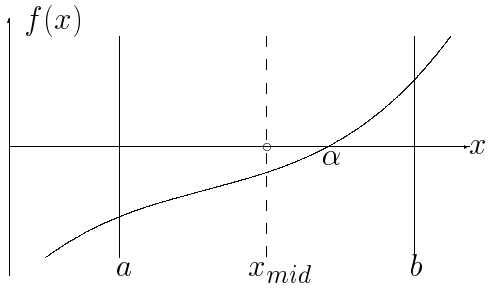
Opgaven består i at finde, for en funktion $f(x)$, den x-værdi (rod) eller de x-værdier, hvor $f(x) = 0$. Løsningsmetoden afhænger af, hvad man ved om $f(x)$. Af de eksisterende metoder bliver de følgende fire beskrevet: binær søgning, regula falsi, Newtons metode samt en iterativ metode.

2.1 Binær søgning

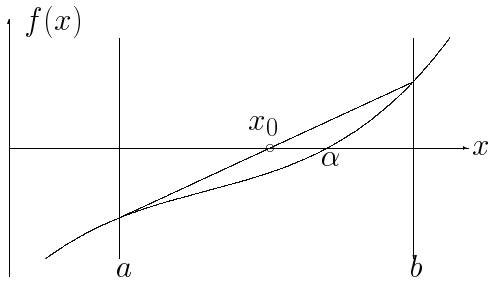
Først skal man, som vist i Fig. 2.1, finde de to punkter a og b, som ligger på hver sin side af løsningen, $x = \alpha$, således at $f(a) < 0$ og $f(b) > 0$. Dette er ofte uden besvær. Så begynder man at indsnævre området $[a, b]$. Proceduren er at beregne funktionen ved midtpunktet mellem a og b:

$$x_{mid} = \tfrac{1}{2}(a + b). \tag{2.1}$$

Hvis $f(x_{mid})$ ligger under nul-linjen, *dvs.* $f(x_{mid}) < 0$, så flyttes a til x_{mid}; hvis $f(x_{mid}) > 0$, så flyttes b til x_{mid}. Denne proces (dette trin) gentages indtil området $[a, b]$ (som bliver ved med at omgive $x = \alpha$) er tilstrækkelig snæver. Man skal definere sig en passende størrelse, ε, man er tilfreds med, og den definerer så den nøjagtighed, hvormed α er blevet fundet. *Dvs.* man ender processen, når $|a - b| \leq \varepsilon$ og afleverer roden $\alpha \pm \varepsilon$.



Figur 2.1: Binær søgning



Figur 2.2: Regula falsi

Ovennævnte beskrivelse kan formaliseres med følgende algoritme:

1. Find a og b, hvor $f(a) < 0$ og $f(b) > 0$
2. Beregn $x_{mid} = \frac{1}{2}(a + b)$
3. Hvis $f(x_{mid}) < 0$, $a \leftarrow x_{mid}$
   ellers $b \leftarrow x_{mid}$
4. Hvis $|a - b| > \varepsilon$, gentag fra 2.

Eftersom man halverer området $[a, b]$ ved hvert trin, er det forudsigeligt, hvor mange *iterationer* (gentagelsestrin) man skal udføre for at opnå et interval ε: det er $\log_2(|a - b|/\varepsilon)$. Denne metode konvergerer forholdsvis langsomt, men for "pæne" $f(x)$ er den meget robust; *dvs.* den vil altid finde løsningen, α.

2.2 Regula falsi

Denne er en variant af binær søgning og konvergerer (lidt) hurtigere. Der startes med et par, a og b, ligesom med binær søgning. Som vist i Fig. 2.2, finder man nu punktet x_0, hvor $f(x_0) = 0$ på den rette linje trukket fra $f(a)$ til $f(b)$. Simpel geometri giver, at

$$x_0 = \frac{af(b) - bf(a)}{f(b) - f(a)} \; . \tag{2.2}$$

Denne x_0 bliver nu behandlet på samme måde som under binær søgning. Algoritmen er:

1. Find a og b, $f(a) < 0$ og $f(b) > 0$.
2. Beregn x_0
3. Hvis $f(x_0) < 0$, $a \leftarrow x_0$
   ellers $b \leftarrow x_0$

4. Hvis $|a - b| \geq \varepsilon$, gentag fra 2.

Metoden er lige så robust som binær søgning (for pæne funktioner), men noget hurtigere til at konvergere.

2.3 Newtons metode

Denne metode er en af de hurtigst konvergerende - når den virker, hvilket ikke altid er tilfældet. Reglen er, at man nok burde kende en rimelig approksimation til den rod, der skal findes, og at funktionen skal være pæn i dette område. Hvis dette er givet, konvergerer metoden meget hurtigt. I øvrigt skal man kunne differentiere $f(x)$ mht. x, men det kan man altid (se afsnit 2.4).

Der er to måder at forklare metoden på; en geometrisk og en algebraisk. I Fig. 2.3 har vi et foreløbigt bud på roden, $x = x_n$, som vi har fundet efter n trin. Vi vil nu forbedre den til den nye værdi x_{n+1}. Vi tegner tangenten til funktionen ved x_n og den skærer x-aksen ved x_{n+1}. Det er indlysende, at

$$f'(x_n) = \frac{f(x_n)}{x_n - x_{n+1}} \tag{2.3}$$

som giver igen

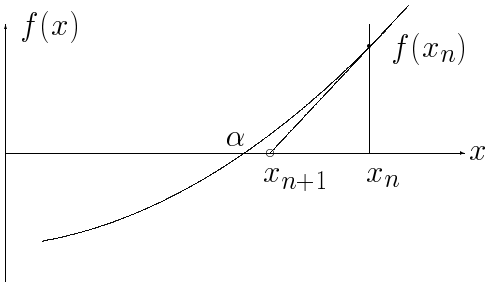
$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \; . \tag{2.4}$$

Det tilsvarende algebraiske argument går ud fra Taylorrækken: Vi har x_n og vil finde $x_n + \delta x = x_{n+1}$, således at $f(x_{n+1}) = 0$. Taylorrækken giver

$$f(x_n + \delta x) = f(x_n) + \delta x f'(x_n) + \frac{\delta x^2}{2!} f''(x_n) + \ldots \tag{2.5}$$

Vi sætter $f(x_n + \delta x) = 0$ og ser bort fra led med højere potenser end δx:

$$\begin{aligned} 0 &= f(x_n) + \delta x f'(x_n) \\ \delta x &= -f(x_n)/f'(x_n) \end{aligned} \tag{2.6}$$



Figur 2.3: Newtons metode

hvilket igen giver os den samme formel,

$$x_{n+1} = x_n + \delta x = x_n - \frac{f(x_n)}{f'(x_n)} \,. \tag{2.7}$$

Denne måde at udlede metoden på har den fordel, at den lettere kan udvides til funktioner af flere variabler, *dvs.* et sæt funktioner.

Newtons metode med et sæt funktioner

I afsnit 1.4 beskrives Taylorrækken for et sæt funktioner $f_i(\mathbf{x}),\ i = 1 \ldots N$, for hvilket vi har et sæt bud på rødderne, $x_i,\ i = 1 \ldots N$. Vi kom til approksimationen

$$\mathbf{f}(\mathbf{x} + \mathbf{h}) = \mathbf{f}(\mathbf{x}) + \mathbf{J} \cdot \mathbf{h} \tag{2.8}$$

hvor $\mathbf{f}$ er vektoren $[f_1\ f_2\ \ldots\ f_N]$, $\mathbf{h}$ er vektoren $[\delta x_1\ \delta x_2\ \ldots\ \delta x_N]^T$ og $\mathbf{J}$ er Jacobimatricen som defineret i afsnit 1.4. Analog med proceduren for en enkelt variabel sætter vi $\mathbf{f}(\mathbf{x} + \mathbf{h})$ lig nul og får forbedringsvektoren $\mathbf{h}$ med

$$\mathbf{h} = -J^{-1} \cdot \mathbf{f}(\mathbf{x}) \tag{2.9}$$

og derfor

$$\mathbf{x_{n+1}} = \mathbf{x_n} + \mathbf{h} = \mathbf{x_n} - \mathbf{J}^{-1} \cdot \mathbf{f}(\mathbf{x}) \tag{2.10}$$

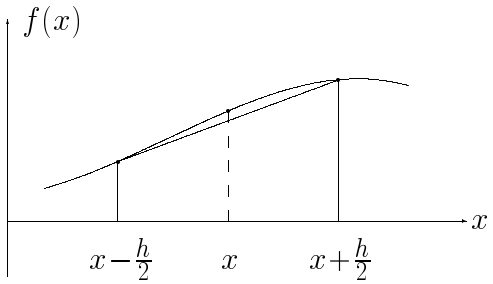
Her, eftersom $\mathbf{J}$ er en matrix, skal den (i princippet) inverteres og inversen ganges med $\mathbf{f}(\mathbf{x})$ (men i praksis løser man ligningen $\mathbf{J} \cdot \mathbf{h} = -\mathbf{f}(\mathbf{x})$ direkte for $\mathbf{h}$).

2.4 "Besværlige"funktioners derivater

Newtons metode benytter derivater. Man har ofte med funktioner $f(x)$ at gøre, som ikke så let bare kan evalueres. Det kan *f.eks.* være, at $f(x)$ i sig selv er et resultat af et indviklet regnestykke og at der ikke foreligger et udtryk for dets derivat $f'(x)$. I så fald kan man approksimere $f'(x)$ numerisk. Man vælger et lille interval h og bruger approksimationen

$$f'(x) \approx \frac{f(x + \frac{h}{2}) - f(x - \frac{h}{2})}{h} \,. \tag{2.11}$$

hvilket er illustreret i Fig. 2.4. Valget af h kræver lidt "fingerfølelse"; det skulle helst være så lille som muligt, men hvis man vælger det så lille, at differencen mellem de to funktionsværdier nærmer sig deres præcision, bliver $f'(x)$ unøjagtig. Her er et eksempel. Vi approksimerer $f'(x)$ ved $x = 1$ for funktionen $f(x) = \mathrm{e}^{-x}$. Vi kender $f'(x)$: det er $-\mathrm{e}^{-1}$, eller -0.3678794. Her er en tabel over approksimationerne for en række h-værdier, ved at bruge enkelpræcision i regnemaskinen (6-7 decimalers præcision).



Figur 2.4: Numerisk differentiering

h	$f'(approx)$	fejl
10^{0}	-0.383400	-0.015521
10^{-1}	-0.368033	-0.000153
10^{-2}	-0.367880	-0.000000
10^{-3}	-0.367880	-0.000000
10^{-4}	-0.367761	0.000119
10^{-5}	-0.366569	0.001311
10^{-6}	-0.357628	0.010252
10^{-7}	-0.000000	0.367879

Bemærk, at fra $h > 10^{-3}$ bliver approksimationen dårligere og til sidst nul. Normalt skal man eksperimentere sig frem, og muligvis bruge en højere regnepræcision.

2.5 Invertering af funktioner

Mange funktioner kan let vendes om. For eksempel, hvis

$$y = f(x) = \mathrm{e}^{-x} \tag{2.12}$$

og man vil finde x for en given y, er dette uproblematisk, fordi funktionen kan omvendes til

$$x = -\ln(y)\ . \tag{2.13}$$

Dette er imidlertid ikke altid tilfældet. I elektrokemien finder man *f.eks.* en funktion af formen

$$y = ae^{ux} + be^{vx} \tag{2.14}$$

og den kan ikke vendes om uden videre. Hvis vi nu alligevel gerne vil finde en x-værdi, der passer til en given y-værdi (a, b, u og v antages som kendte

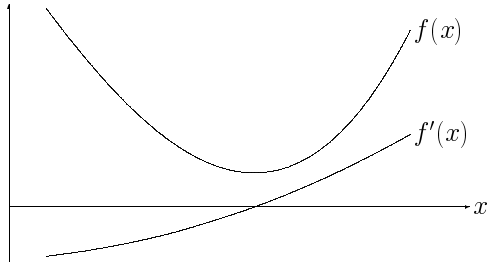
parametre), skal vi bruge en søgemetode. Det gør vi ved at omdanne opgaven til et rodfindingsproblem: Vi skriver

$$f(x) = ae^{ux} + be^{vx} - y \tag{2.15}$$

og finder funktionens rod, hvor $f(x) = 0$.

2.6 Min/max søgning

Et relateret problem er at finde en funktions minimum eller maximum, se Fig. 2.5. Den nemmeste metode er at finde roden til $f'(x)$, fordi min/maximummet jo vil ligge et sted, hvor $f'(x) = 0$. Man skal altså bare evaluere $f'(x)$ og bruge en af de før omtalte rodfindingsmetoder.



Figur 2.5: Funktion og dens derivat

Hvis funktionen er en multivariatsfunktion, $f(x_1, x_2, \ldots, x_N)$ så får man et sæt : $f'_1, f'_2, \ldots, f'_N$:

$$f'_i = \frac{\partial}{\partial x_i} f\left(x_1, x_2, \ldots, x_N\right) \qquad\qquad i = 1 \ldots N \tag{2.16}$$

for hvilket vi også kan finde et sæt rødder som beskrevet.

2.7 "g-Metoden" - iterativ løsning

Denne metode, også kaldt iterationsmetode, er nogle gange nyttig til at finde roden af en ikkelineær funktion. Lad funktionen (problemet) være

$$f(x) = 0 \tag{2.17}$$

og lad os sige, at vi kan omordne funktionen til en anden form,

$$x = g(x)\ . \tag{2.18}$$

Her har vi en implicit form af den første. Hvis vi er heldige, kan følgende iterative formel bruges,

$$x_{n+1} = g(x_n) \tag{2.19}$$

dvs., vi sætter det nuværende x_n ind i funktionen og får et forbedret gæt x_{n+1} ud af den.

En kendt anvendelse af denne metode forekommer i kemien, ved beregning af koncentrationen af H^+, når en svag syre (syrekonstant K) er opløst med koncentration c i vand. Det eksakte udtryk der skal bruges er

$$K = \frac{x^2}{c - x} \tag{2.20}$$

hvor $x = [H^+]$. Ofte tillader man sig at sige, at x nok er forsvindende i forhold til c (for små K-værdier), og så forenkles udtrykket til

$$K = \frac{x^2}{c} \,, \tag{2.21}$$

dvs. direkte

$$x = \sqrt{Kc} \,. \tag{2.22}$$

Men hvis den antagelse ikke gælder, kan man anvende den iterative formel

$$x_{n+1} = \sqrt{K\,(c - x_n)} \tag{2.23}$$

startende med $x_0 = 0$. Et konkret eksempel: lad $pK_a = 2$ ($K = 0.01$) og $c = 0.1M$. Sekvensen er:

$$\begin{aligned}
x_1 &= \sqrt{0.01\,(0.1 - 0)} = \sqrt{0.001} && = 0.03162 \\
x_2 &= \sqrt{0.01\,(0.1 - 0.03162)} && = 0.02615 \\
x_3 &= \sqrt{0.01\,(0.1 - 0.02615)} && = 0.02718 \\
x_4 &= \sqrt{0.01\,(0.1 - 0.02718)} && = 0.02699 \\
&\vdots \\
x_7 & && = 0.0270156 \,.
\end{aligned} \tag{2.24}$$

Det ses, at vi her undgår at løse en andengrads ligning. Metoden er desuden meget "lommeregnervenlig" - prøv selv; man behøver ikke indtaste mellemværdierne, kun konstanterne. Læg også mærke til, at en andengrads ligning har to rødder, og at vi her har set bort fra den anden rod, som er meningsløs i denne sammenhæng.

Denne metode bliver nævnt igen i kapitel 7 i forbindelse med løsning af lineære systemer (Gauss-Seidel).

2.8 Polynomier

Polynomier er et emne af særlig interesse, og der er særlige metoder for at finde deres rødder.

Op til den fjerde grad findes der direkte analytiske metoder for at finde rødderne. Der er en velkendt formel for et andengrads polynomium, og Abramowitz & Stegun [3] viser metoderne for tredje og fjerde graderne. Derefter bliver det for svært, og man griber til numeriske metoder. Der er dog undtagelser, i de tilfælde, hvor man let kan finde en faktor (en rod), som funktionen kan divideres med, som måske derefter kan løses analytisk. Division med en faktor er kaldt "deflation".

I bogen Press & al. [4] er der præsenteret flere metoder for at finde polynomiumsrødder, hvoraf der er to, som er særligt anbefalet. Vi beskriver dem her.

Laguerre-metoden

Lad polynomiet være $P_n(x)$, og det har rødderne $x_i, i = 1, \ldots n$:

$$P_n(x) = (x - x_1)(x - x_2)\ldots(x - x_n)\ . \tag{2.25}$$

Vi kender ingen af rødderne, men vi har et startgæt på en af dem. Logaritmen af funktionen er

$$\ln|P_n(x)| = \ln|x - x_1| + \ln|x - x_2| + \cdots + \ln|x - x_n| \tag{2.26}$$

som efter differentiering bliver til

$$\frac{\mathrm{d}\ln|P_n(x)|}{\mathrm{d}x} = \frac{1}{x - x_1} + \frac{1}{x - x_2} + \cdots + \frac{1}{x - x_n} = \frac{P_n'}{P_n} \equiv G \tag{2.27}$$

hvor P_n' står for det første afled af P_n. Læseren kan selv overbevise sig om denne ligning. Vi differentierer én gang til:

$$-\frac{\mathrm{d}^2\ln|P_n(x)|}{\mathrm{d}x^2} = \frac{1}{(x - x_1)^2} + \frac{1}{(x - x_2)^2} + \cdots + \frac{1}{(x - x_n)^2} = \left(\frac{P_n'}{P_n}\right)^2 - \frac{P_n''}{P_n} \equiv H \tag{2.28}$$

hvor P_n'' er andet afled. Både G og H kan evalueres ved den nuværende x-værdi.

Vi mener (håber) , at x_1 ikke er langt fra vores nuværende gæt x. Vi kalder afstanden $x - x_1 = a$, og vi vil derfor gerne finde a. Mærkeligvis indebærer metoden den antagelse, at alle andre rødder har samme afstand b fra vores gæt, *dvs.*, $x - x_i = b, i = 2, 3, \ldots, n$. Med disse antagelser formulerer vi ligningerne (2.27) og (2.28) til de to nye

$$G = \frac{1}{a} + \frac{n - 1}{b} \tag{2.29}$$

og

$$H = \frac{1}{a^2} + \frac{n-1}{b^2} \tag{2.30}$$

som vi kan løse for a:

$$a = \frac{n}{G \pm \sqrt{(n-1)(nH - G^2)}} \tag{2.31}$$

hvor man vælger fortegnet således, at nævneren får den størst mulig værdi. Så har vi afstanden a, og kan korrigere vores gæt dermed. Bemærk, at termen indenfor kvadratroden kan være imaginær, så vi kan finde en kompleks rod med metoden.

Det er klart, at vi ikke finder den rigtige rod ved første gennemgang, og at vi altså må iterere. Vi gør det et antal gange, indtil $a = 0$, eller så tæt på at vi er tilfreds. Så har vi fundet en af rødderne, og kan deflate P_n til P_{n-1} med faktoren $(x - x_1)$. Derefter kan vi starte med samme procedure for at finde en anden rod, *osv.*

Efter vi har samlet et antal rødder, som vi mener, ikke er præcise nok, kan vi benytte hvad Press & al. [4] kalder rodpolering (eng. *root polishing*) til at gøre dem mere præcise. Se sidste afsnit i dette kapitel.

Et eksempel

Vi vil finde rødderne af polynomiet

$$P_4(x) = x^4 - 8x^3 + 17x^2 + 2x - 24 = 0 \tag{2.32}$$

(den har faktisk rødderne -1, 2, 3, 4), og vi gætter på $x = 1$ som start. Vi benytter de ovenstående ligninger og de første tre iterationer giver os henholdsvis løsningerne 1.7882, 1.9964 og 2.000, som ikke ændrer sig derefter. Det efterlader kun tre rødder, som kunne findes analytisk efter opskriften i Abramowitz & Stegun [3], efter deflation, ved at dividere $P_4(x)$ med $(x - 2)$. Det resulterer i

$$P_3(x) = x^3 - 6x + 5x + 12 \; . \tag{2.33}$$

Men, da vi nu har et program der benytter Laguerres metode, kan vi jo fortsætte med det. Vi gætter igen at den nye rod er $x = 1$, og nu får vi talrækken 2.3703, 2.9506 og 3.0000. Hvis vi deflatere igen med $(x - 3)$, ender vi med en andengrads ligning

$$P_2(x) = x^2 - 3x - 4 \; , \tag{2.34}$$

som vi let kan løse for de to sidste rødder.

Egenværdimetode

Press & al. [4] beskriver følgende metode. Den vender problemet at finde egenværdier på hovedet. Som nævnt i kapitel 7 på side 74, fører definitionen af eigenværdierne naturligt til et polynomiumsudtryk for disse. Det er dog den mindst effektive måde at finde egenværdier på, da man har mere effektive iterative metoder til det. Disse metoder kan man tilpasse problemet polynomiumsrødder. Vi kan udtrykke et polynomium i formen

$$P_n(x) = a_0 + a_1 x + \cdots + a_{n-1}x^{n-1} + x^n \tag{2.35}$$

(man kan jo dividere det hele med a_n hvis $a_n \neq 1$). Man opstiller en såkaldt *companion matrix*

$$\mathbf{A} = \begin{bmatrix} -a_{n-1} & -a_{n-2} & \cdots & -a_1 & -a_0 \\ 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 \\ \vdots & & & & \vdots \\ 0 & 0 & \cdots & 1 & 0 \end{bmatrix} . \tag{2.36}$$

Denne matrices egenværdier er også rødderne af polynomiet.

Et eksempel

Hvis vi igen tager polynomiet (2.32), så bliver companion matricen til

$$\mathbf{A} = \begin{bmatrix} 8 & -17 & -2 & 24 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{2.37}$$

og et passende underprogram for eigenværdier finder alle fire rødder på én gang.

Rodpolering

Begge metoder nævnt ovenfor er iterative metoder, og derfor kan det ske, at vi tvivler på værdiernes præcision. Med "root polishing" kan vi forbedre dem. Det baserer på Newtons metode. Man tager udgangspunkt i hver af de rødder man har, og iterer frem med Newton. Funktionen er selve polynomiet, samt dets første afledte. Normalt tager det ikke mere end 1-2 trin inden man konvergerer til 16 decimaler på den præcise rod. Formlen er altså

$$x_{n+1} = x_n - \frac{P_n(x_n)}{P_n'(x_n)} . \tag{2.38}$$

Metoden virker også, hvis en rod er kompleks - man skal blot sørge for, at programmet er skrevet passende, med variabler af typen `complex`.

Kapitel 3

Integration

Det hænder, at vi er interesseret i at beregne et bestemt integral af en vanskelig funktion:

$$I = \int_a^b f(x)\mathrm{d}x \tag{3.1}$$

som vi ikke umiddelbart kan integrere algebraisk. Vi deler området $[a, b]$ der skal integreres i lige store dele h (se Fig. 3.1).

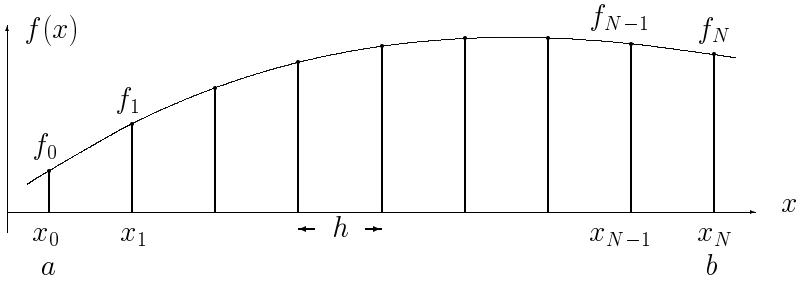
Vi behandler først en funktion som vi kan evaluere ved hver given position x. Senere kommer vi til funktioner som er repræsenteret som et diskret sæt værdier. De tilsvarende $f(x)$-værdier er $f_0, f_1, f_2, \ldots, f_N$. Integralet I er arealet af $f(x)$ i området $[a, b]$.

Der er et antal forskellige måder at integrere på, som skal beskrives nedenfor.

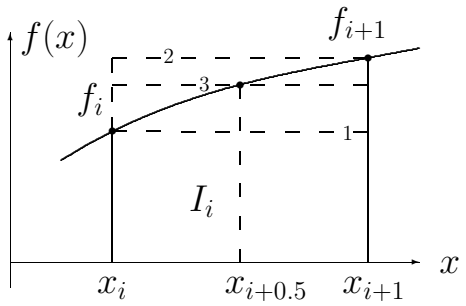
3.1 Blokintegration

Begrebet blokintegration betegner her en approksimation der benytter en konstant over hvert givet interval.

I Fig. 3.2 vises der et af de N intervaller i Fig. 3.1. Vi kan vælge tre forskel-



Figur 3.1: Inddeling af en funktion



Figur 3.2: Blockintegration, enkelt element

lige konstanter, vist i figuren af de tre stiplede linjer 1, 2 og 3. Linjerne 1 og 2 er klart utilfredsstillende, men linje 3 forekommer intuitivt som den bedste. Bruger vi linje 1, så er integralet I_i lig med

$$I_i = f_i \ (x_{i+1} - x_i) = h f_i \ , \tag{3.2}$$

mens hvis vi bruger linje 2, er approksimationen

$$I_i = h f_{i+1} \ . \tag{3.3}$$

Det er let at vise, at begge disse valg giver en approksimation af $O(h)$; *dvs*, hvis vi prøver at forbedre vores approksimation ved at doble antallet af skridt (og halvere h), får vi blot den halve fejl. Det kan gøres bedre. Et oplagt valg er linje 3, og approksimationen bliver så

$$I_i = h f(x_{i+0.5}) \ . \tag{3.4}$$

Denne approksimation, midtpunktsmetoden, er $O(h^2)$, hvilket er meget bedre.

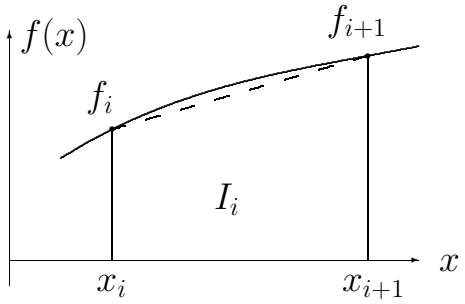
Det totale integral I er nu summen af alle I_i, og hvis vi bruger den sidstnævnte metode, bliver det til den udvidede formel

$$I = \sum_{i=0}^{N-1} I_i = h \sum_{i=0}^{N-1} f(x_{i+0.5}) \tag{3.5}$$

hvor integralsymbolet $\int$ er blevet erstattet med sumsymbolet $\sum$.

3.2 Trapezformler

I Fig. 3.3 gør vi det indlysende at bruge trapezet, hvilket vi får ved at trække en ret linje mellem (x_i, f_i) og (x_{i+1}, f_{i+1}). Nu er betingelsen for at fejlarealet er minimal, kun at funktionen $f(x)$ ikke krummer meget i området. Ideelt skulle den være en ret linje. Formlen er:



Figur 3.3: Trapezintegration, enkelt element

$$I_i = \tfrac{h}{2}\,(f_i + f_{i+1}) \tag{3.6}$$

og når vi lægger alle I_i sammen:

$$\begin{aligned} I &= I_0 + I_1 + \cdots + I_{N-1} \\ &= \frac{h}{2}(f_0 + f_1) \\ &\quad + \frac{h}{2}(f_1 + f_2) \\ &\quad + \ldots \\ &\quad + \frac{h}{2}(f_{N-2} + f_{N-1}) \\ &\quad + \frac{h}{2}(f_{N-1} + f_N) \end{aligned} \tag{3.7}$$

får vi

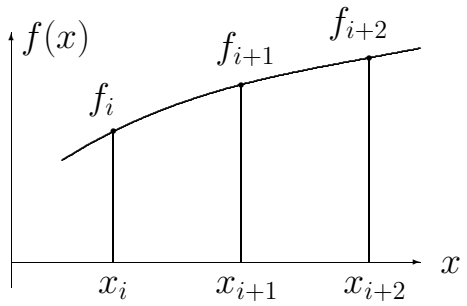
$$I = h\,\left\{\tfrac{1}{2}f_0 + f_1 + f_2 + \cdots + f_{N-1} + \tfrac{1}{2}f_N\right\}\ . \tag{3.8}$$

Lige som (3.5), er denne formel $O(h^2)$.

3.3 Simpsons regel

I blokintegrationen brugte vi en grov blok som approksimation og ser helst, at $f(x)$ er konstant i intervallet. Ved trapezformlen behøver $f(x)$ ikke at være konstant, men gerne en ret linje. Hvis vi nu går over til flere end to x-værdier, så kan vi endog tage krumningen af $f(x)$ med. I Fig. 3.4 ser vi et dobbeltinterval, givet af tre punkter. Det er let at tilpasse en parabel til de tre punkter, og integralet af denne parabel over de to intervaller giver så

$$I = \frac{h}{3}\,(f_i + 4f_{i+1} + f_{i+2})\ . \tag{3.9}$$



Figur 3.4: Simpsonintegration, dobbelt element

Når vi sætter $N/2$ af disse dobbeltintervaller sammen, får vi

$$I = h\left\{\tfrac{1}{3}f_0 + \tfrac{4}{3}f_1 + \tfrac{2}{3}f_2 + \cdots + \tfrac{2}{3}f_{N-2} + \tfrac{4}{3}f_{N-1} + \tfrac{1}{3}f_N\right\} \ , \tag{3.10}$$

hvilket er den velkendte Simpsons regel. Læg mærke til at for at få sekvensen af koefficienterne, $\frac{1}{3}, \frac{4}{3}, \frac{2}{3}, \frac{4}{3}, \ldots, \frac{2}{3}, \frac{4}{3}, \frac{1}{3}$ til at passe, skal N være et lig tal; *dvs.* der skal være et **lige** antal **intervaller** eller et **ulige** antal x-**værdier**.

Den ovenstående Simpsonformel kan udtrykkes som

$$I = \frac{h}{3}\left(f_0 + f_N + 4U + 2L\right) \tag{3.11}$$

hvor $U = f_1 + f_3 + \cdots + f_{N-1}$ (de uliges sum), og $L = f_2 + f_4 + \cdots + f_{N-2}$ (de liges sum). Denne opdeling bliver meget nyttig i et følgende afsnit.

3.4 Fejl

For en god ordens skyld nævner vi her, hvad vi ved om fejlen i den samlede I for de forskellige approksimationer. Vi har:

blokintegration: $O(N^{-1})$ eller $O(h)$

midtpunktsintegration: $O(N^{-2})$ eller $O(h^2)$

trapezformlen: $O(N^{-2})$ eller $O(h^2)$

Simpsons regel: $O(N^{-4})$ eller $O(h^4)$.

Det vil for eksempel sige, at hvis vi for Simpsons regel forøger antallet af intervallerne N til $2N$, deler vi fejlen med $2^4 = 16$. Oplysningen af ordenen kan vi bruge til at optimere eller forbedre en given approksimation, se det følgende afsnit, samt afsnit 3.6.

3.5 Romberg integration

Denne metode er beskrevet meget klart af Østerby [5], og artiklen kan skaffes. Se igen på blokintegrationen ved brug af midtpunkterne. Det kan vises, at det approksimerede integral I_h svarende til en intervalstørrelse h kan udtrykkes af rækken

$$I_h = \hat{I} + a\,h^2 + b\,h^4 + c\,h^6 + \ldots \tag{3.12}$$

hvor $\hat{I}$ er den sande værdi, som vi gerne vil approksimere, og de andre terme er fejltermene. Den laveste potens, i h^2, vil dominere, hvilket er grunden til at man siger at metoden er $O(h^2)$. Hvis vi kunne eliminere denne dominerende term, ville vi opnå en højere orden $O(h^4)$; det kan lade sig gøre.

Generelt, uden at specificere metoden, er ligningen

$$I_h = \hat{I} + a\,h^p + b\,h^q + c\,h^r + \ldots \tag{3.13}$$

hvor $a, b, c, \ldots$ er ukendte konstanter, og $p, q, r, \ldots$ er her uspeciferede potenser. Dem kan vi enten beregne, eller hvis vi ikke kan, kan vi numerisk bestemme dem, se afsnit 1.5. Metoden benytter to estimater af integralet; først med N skridt af længde h i intervallet med resultatet I_h, og så med et nyt estimat med $N/2$ skridt af længde $2h$, med resultatet I_{2h}. Så kan vi skrive to ligninger:

$$\begin{aligned} I_h &= \hat{I} + a\,h^p + b\,h^q + c\,h^r + \ldots \\ I_{2h} &= \hat{I} + a\,2^p\,h^p + b\,2^q\,h^q + c\,2^r\,h^r + \ldots \end{aligned} \tag{3.14}$$

og hvis vi ganger den første af de to med 2^p og trækker den anden fra resultatet får vi, efter noget omrokkering,

$$I_h + \frac{I_h - I_{2h}}{2^p - 1} = \hat{I} + d\,h^q + \ldots \tag{3.15}$$

hvor d er en ny konstant. Det ses, at approksimationen nu er af den højere $O(h^q)$. Vi kan blive ved, ved at beregne endnu et integral med $N/4$ og skridtlængde $4h$, og beregn endnu en ny approksimation, også af $O(h^q)$, og så kan vi gentage Rombergprocessen og også eliminere termen i h^q, og igen øge vores orden til $O(h^r)$, osv. Det er en meget effektiv måde at forbedre integrationen på.

Et numerisk eksempel

Vi vil beregne integralet $\int_0^1 exp(-x)\mathrm{d}x$. Vi kender resultatet, $1 - exp(-1)$. Vi bruger midtpunktsintegration, og forsøger os med N som vi dobler, indtil fejlen er tilstrækkelig lille, her defineret ved at den er $< 10^{-6}$. Resultatet er:

```
   N    fejl uden Romberg   fejl med Romberg
   4      -0.025590            -0.000186
   8      -0.006537            -0.000012
  16      -0.001643            -0.000001
  32      -0.000411             0.000000
  64      -0.000103             0.000000
 128      -0.000026             0.000000
 256      -0.000006             0.000000
 512      -0.000002             0.000000
1024       0.000000             0.000000
```

Vi ser, at uden Romberg tager det 1024 intervaller, inden vi opnår den ønskede fejl, mens med Romberg er blot 16 intervaller nok. Rombergintegration er en meget effektiv metode.

3.6 Dynamisk intervaldeling

Vi vil gerne integrere en funktion $f(x)$ i et givet interval, men vi ved ikke, hvilken N-værdi, der er passende. Vi ved selvfølgelig, hvor stor en fejl i integralet vi kan acceptere. Lad os sige, at vi bruger Simpson. Vi kender ikke fejlen selv, kun dens afhængighed af N (eller h). Lad denne ukendte fejl være e_N for en given N-værdi. Hvis vi nu *f.eks.* dobler N, ved vi, at fejlen bliver sekstendelt, *dvs.* $e_{2N} = \frac{1}{16}e_N$. Hvis vi altså beregner integralet for N intervaller, I_N , og derefter I_{2N} for $2N$ intervaller, kan vi konkludere, at forskellen mellem de to værdier er et udtryk for selve fejlen. Vi kan det, fordi den sande værdi for I, $\hat{I}$, jo er den samme hver gang, så vi kan opstille de to ligninger

$$\begin{aligned} I_N &= \hat{I} + e_N \\ I_{2N} &= \hat{I} + e_{2N} \end{aligned}$$

og

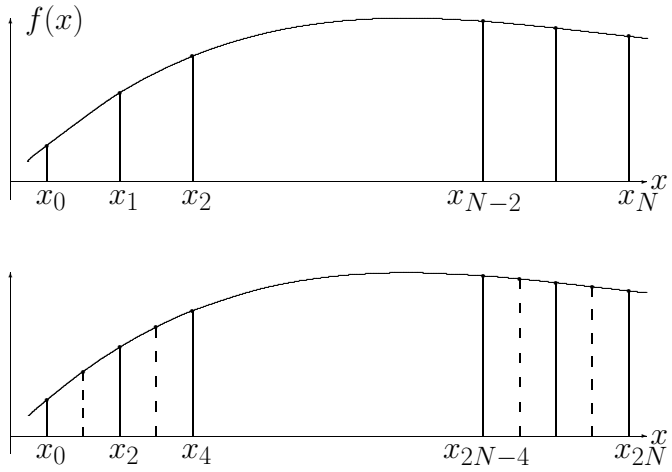
$$e_{2N} = \tfrac{1}{16}e_N \ .$$

Vi får

$$\begin{aligned} I_N - I_{2N} &= e_N - e_{2N} \\ &= e_N \left(1 - \tfrac{1}{16}\right) \\ &= \tfrac{15}{16}e_N \approx e_N \ . \end{aligned} \tag{3.16}$$

Normalt ved vi, hvilken værdi vi er tilfreds med. Proceduren er altså, at gentagne gange doble antallet af intervallerne N, se på differensen $|I_N - I_{2N}|$, og når den er tilstrækkelig lille, at standse processen. Så ved man, at den sidste I_{2N}-værdi har en fejl der er mindre end den højst acceptable.

Denne procedure er temmelig nem at anvende med Simpsons regel: Fig. 3.5 viser en dobling af N. I den øvre del ses N intervaller, og vi har, at



Figur 3.5: Dobling af antal intervaller N

$$I_N = \frac{h}{3}\{f_0 + f_N + 4U + 2L\} \tag{3.17}$$

med definitionerne for U og L som ovenstående. Vi skal altså beregne tre summer:

enderne, $f_0 + f_N$

de ulige, $f_1 + f_3 + \cdots + f_{N-1}$

de lige, $f_2 + f_4 + \cdots + f_{N-2}$

og lægge dem sammen til I_N som i formlen. Når vi går over til $2N$ intervaller, kan vi bruge alle tre summer igen; vi behøver kun at beregne de nye f'er ind imellem (de stiplede linjer, alle ulige). Enderne er de samme for alle N. Det er tydeligt fra Fig. 3.5, at for $2N$, alle de nye lige punkter er både de lige og ulige for N, så vi kan sige, at

$$\text{(sum af lige) } (2N) = \text{(sum af ulige) } (N) + \text{(sum af lige) } (N)\ . \tag{3.18}$$

Dette er meget nemt at programmere. Algoritmen er:

1. Start med $N = 1$ ($\rightarrow f_0, f_1$)
2. Endesum $E = f_0 + f_1$ (konstant herefter)
3. Sum af ulige $U = 0$ (der er ingen ulige endnu)
4. Sum af lige $L = 0$ (ingen lige endnu)
5. Evaluer integralet, I_N

6. Dobl $N \to 2N$ (halver h); $L_{2N} = U_N + L_N$
7. Beregn den nye $U = f_1 + f_3 + \ldots$
8. Beregn den nye I_{2N}
9. Evaluer $|I_N - I_{2N}| = e$
10. Sæt $I_N := I_{2N}$
11. Hvis $|e| >$ tolerancen, vend tilbage til 6, ellers er I_N det ønskede integral.

3.7 Diskrete funktionsværdier

Vi vil tit beregne integralet af en funktion vi kun kender som et fast antal punkter, måske målet af et instrument, eller beregnet på en eller anden måde. Her kan vi ikke beregne funktionsværdier ved vilkårlige argumenter, men må holde os til de punkter, vi har. Det kunne for eksempel være de punkter set i Fig. 3.1. Vi kan altså ikke direkte bruge midtpunktsmetoden. Men vi kan approksimere midtpunkterne ved at tage gennemsnittet af de kendte nabopunkterne:

$$f_{i+0.5} \approx \frac{1}{2}(f_i + f_{i+1}) \; . \tag{3.19}$$

Når vi lægger dem alle sammen som i (3.5), ender vi med præcist det samme som med trapezmetoden. Hvad er mere, kan vi også anvende Romberg her, ved at beregne integralet igen, ved brug af hver andet punkt, og Romberg-formeln. En anden mulighed, hvis antallet af punkterne er ulig er, at anvende Simpsons regel. Der er altså ikke specielle probleme med integralet af et sæt værdier ved faste punkter.

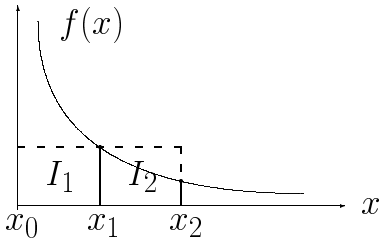
3.8 Åbne formler

Der forekommer funktioner, hvor man ikke kan evaluere $f(a)$ eller/og $f(b)$, *f.eks.*:

$$\int_0^1 x^{-\frac{1}{2}} \, \mathrm{d}x \; . \tag{3.20}$$

Hvis det drejer sig om en funktion vi kan evaluere som (3.20), kan vi bruge midtpunktsmetoden, fordi den vil aldrig berøre endepunkterne, så singulariteterne ikke dukker op. Her finder man ikke-heltals ordener. Integrerer man funktionen i (3.20) med midtpunktsmetoden, og måler man ordenen, finder man ud af, at integralet er givet ved

$$I_h = \hat{I} + a\, h^{\frac{1}{2}} + b\, h^2 + \ldots \tag{3.21}$$



Figur 3.6: Åben form, blok integration

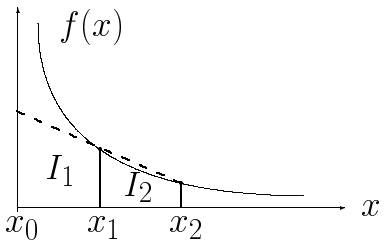
og andre åbne funktioner har andre "mærkelige" potenser i rækken. Når man kender disse potenser, kan man igen anvende Rombergmetoden, for at forbedre effektiviteten.

Hvis, på den anden side, funktionen er et sæt diskrete værdier, er der et problem ved enden. Her kan man approksimere den "vanskelige" ende med en slags ekstrapolation. Fig. 3.6 viser sådan en funktion. Ved blokintegration kan vi her lade både I_1 og I_2 være lig med hf_1, og derfor ikke få brug for f_0. Det er klart, at en bedre approksimation for I_1 ville være, som vist i Fig. 3.7, at bruge trapezformlen med en f_0-værdi fra en ekstrapolation tilbage til f_0 ved x_0. Da vi har ens intervaller x_0, er ekstrapolationen

$$\begin{aligned} f_0 &= f_1 + (f_1 - f_2) \\ &= 2f_1 - f_2 \ . \end{aligned} \tag{3.22}$$

Trapezformlen giver så

$$I_1 = \tfrac{1}{2}\,(f_1 - f_2) \ . \tag{3.23}$$



Figur 3.7: Åben form, trapezintegration

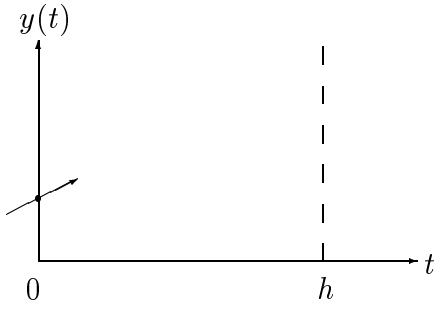
Kapitel 4

Differentialligninger

Mange fænomener i naturvidenskab kan bedst beskrives matematisk i form af differentialligninger af forskellig orden, enten som ordinære (**ode**) eller partielle differentialligninger (**pde**). Det kan også blive til systemer af sådanne ligninger. Eksempler er radioaktiv henfald, kemisk reaktionskinetik og massetransport (diffusion m.m.). Selvom man i nogle tilfælde kan finde analytiske løsninger, er det dog oftest svært eller umuligt, og man er nødt til at bruge numeriske metoder. Vi holder os her til ordinære (ikke-partielle) differentialligninger, ode's, og først til enkle typer. Disse skrives generelt som

$$\frac{\mathrm{d}y}{\mathrm{d}t} = f(y,t) \tag{4.1}$$

I mange tilfælde har funktionen f kun én variabel (y eller t). Man skriver også y' i stedet for $\mathrm{d}y/\mathrm{d}t$. At løse sådan en ligning er ensbetydende med at finde den underliggende funktion $y(t)$. Da $f(y,t)$ udtrykker en gradient, har vi brug for en yderligere oplysning for at løse ligningen: en værdi for y ved en t-værdi eller en såkaldt **randbetingelse**. Fig. 4.1 viser, hvad man har at starte løsningen



Figur 4.1: Begyndelse af løsning af en ode

med: en værdi ved $t = 0$ og hældningen y' dér. Nu beregner vi nye værdier ved de diskrete t-værdier h, $2h$, , . . . *osv.*

Følgende notation bliver brugt her. Vi antager at vores funktion som skal løses for bliver løst i form af et antal punkter ved tider $h, 2h, \ldots,$ hvor h er det tidsinterval vi har valgt. Vi beregner således et antal y-værdier, $y_1, \ldots, y_N$, og oftest beregner vi et nyt punkt y_{n+1} ved t_{n+1} ud fra et kendt, y_n ved t_n.

Som eksempel benyttes der en differentialligning, hvis løsning vi kender:

$$y' = -y\,; \qquad y(0) = 1 \tag{4.2}$$

som har løsningen $y(t) = \exp(-t)$.

4.1 Eulermetoden

Denne simple metode går ud på - grafisk set - at "bevæge sig" hen ad gradienten ved et givet, allerede kendt, punkt ved tiden t_n til det næste ved $t + h$ eller t_{n+1}. Matematisk udtrykt erstatter vi y' med diskretiseringen

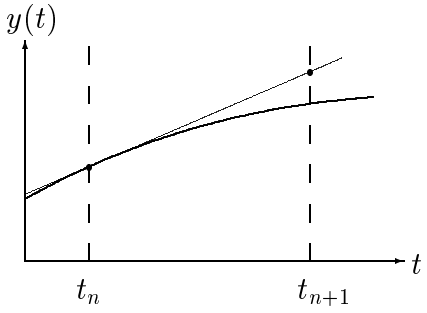
$$y' \approx \frac{y(t+h) - y(t)}{h} \tag{4.3}$$

hvilket vi ved er en første-ordens approksimation, hvis det skal gælde for t (fremaddifferens) eller også for $t+h$ (baglæns). Her er det en fremad-differens. Ved at sætte y' lig med $f(y,t)$ får vi

$$y(t+h) = y(t) + h\, f(y,t) \tag{4.4}$$

eller

$$y_{n+1} = y_n + h\, f(y_n, t)\ . \tag{4.5}$$



Figur 4.2: Et trin med Eulermetoden

For vores eksempel (4.2) giver det

$$y_{n+1} = y_n(1 - h) \; . \tag{4.6}$$

Processen gentages nu for flere t-værdier, indtil vi når til den sidste ønskede t-værdi, $t + Nh$ eller t_N. Fig. 4.2 viser den eksakte $y(t_n)$-værdi sammen med den, man får ved t_{n+1} fra Eulermetoden, og det er klart, at denne metode indebærer en vis fejl. Den kan formindskes ved at tage flere trin med mindre h. Man finder, at fejlen efter N trin er proportional med h, *dvs.* $O(h)$. Der er metoder, der er bedre end dette.

4.2 Brug af Taylorrækken

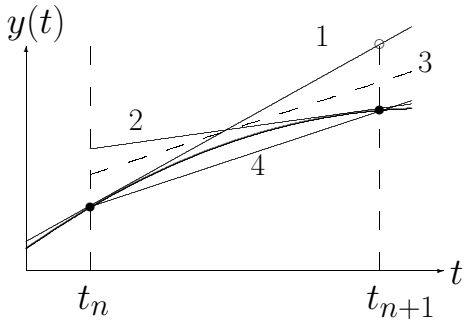
Når vi bruger Eulermetoden, benytter vi faktisk en kort stump af Taylorrækken. Hvis vi har $y(t)$ og vil frem til $y(t + h)$, så siger Taylor

$$y(t + h) = y(t) + hy'(t) + \frac{h^2}{2!}y''(t) + \ldots \tag{4.7}$$

og ved Euler bruger vi kun de to første led på den højre side, hvilket giver (4.5). Hvis vi kan differentiere y', kan vi dog forbedre approksimationen en orden højere ved at tage det tredje led med $\frac{h^2}{2}y''(t)$. Metoden kan dog ikke nemt automatiseres til en generel metode, fordi man jo algebraisk skal differentiere funktionen y'; et computerprogram baseret på denne metode vil altså altid være specifik for en bestemt differentialligning. Metoden anvendes ikke så tit.

4.3 Runge Kutta (RK)

Fig. 4.3 viser en måde at se på RK-metoden: Først tager vi et Euler-trin op ad linje 1 og lander ved det åbne cirkelpunkt, hvilket er Euler-løsningen. Nu



Figur 4.3: Et RK trin

bestemmer vi y' der, *dvs.*, den approksimerede løsnings hældning ved t_{n+1} (linje 2). En hældning midt imellem de to (linje 3) må være bedre end de to hver alene, og den tager vi nu, med start igen fra løsningen ved t_n (linje 4). Det giver en forbedret løsning for t_{n+1} (i hvert fald på kurven som tegnet!).

Alt dette gøres matematisk ved hjælp af en særlig formalisme karakteristisk for RK-metoden: Vi beregner en række k_i-værdier, som alle er udtryk for ændringer i y. Lad os først formulere Eulermetoden i denne form:

$$\begin{aligned} k_1 &= h\, f(y_n, t_n) \\ y_{n+1} &= y_n + k_1 \,. \end{aligned} \tag{4.8}$$

2.-ordens:

Eulermetoden kan således betegnes som en førsteordens RK-metode. De ovenstående overvejelser om en forbedret gradient kan nu udtrykkes på to forskellige måder, begge med to k-værdier, k_1 og k_2:

$$\begin{aligned} k_1 &= h\, f(y_n, t_n) \\ k_2 &= h\, f(y_n + k_1, t_n + h) \qquad (4.9) \\ y_{n+1} &= y_n + \tfrac{1}{2}\,(k_1 + k_2) \qquad (4.10) \end{aligned}$$

(dette svarer til det ovenstående, middelværdi af to gradienter), eller:

$$\begin{aligned} k_1 &= h\, f(y_n, t_n) \qquad \text{(som før)} \\ k_2 &= h\,(y_n + \tfrac{1}{2}k_1, t_n + \tfrac{1}{2}h) \qquad (4.11) \\ y_{n+1} &= y_n + k_2 \,, \end{aligned}$$

hvilket svarer til en gradient beregnet ved et approksimeret midtpunkt $t + \frac{1}{2}h$. Begge varianter er lige gode, og begge er $O(h^2)$.

For vores eksempel (4.2) bliver (4.11) til

$$\begin{aligned} k_1 &= -h\, y_n \\ k_2 &= -h\,(y_n + k_1) \qquad (4.12) \\ y_{n+1} &= y_n + \tfrac{1}{2}(k_1 + k_2) \,. \end{aligned}$$

Igen er der dog forbedringsmuligheder. Vi brugte en første Eulerapproksimation til at forbedre vores resultat for $y(t + h)$. Vi kunne jo nu bruge dette forbedrede resultat til at finde et endnu bedre, *osv.* Der er mange varianter af sådanne højere-ordens varianter, og vi gengiver her to af dem: en 3.-ordens og en 4.-ordens RK formel:

3.-ordens:

$$\begin{aligned} k_1 &= h\, f(y_n, t_n) \\ k_2 &= h\, f(y_n + \tfrac{1}{2}k_1, t_n + \tfrac{1}{2}h) \\ k_3 &= h\, f(y_n - k_1 + 2k_2, t_n + h) \qquad (4.13) \\ y_{n+1} &= y_n + \tfrac{1}{6}\,(k_1 + 4k_2 + k_3) \,. \end{aligned}$$

4.-ordens:

$$\begin{aligned} k_1 &= h\, f(y_n, t_n) \\ k_2 &= h\, f(y_n + \tfrac{1}{2}k_1, t_n + \tfrac{1}{2}h) \\ k_3 &= h\, f(y_n + \tfrac{1}{2}k_2, t_n + \tfrac{1}{2}h) \\ k_4 &= h\, f(y_n + k_3, t_n + h) \\ y_{n+1} &= y_n + \tfrac{1}{6}\,(k_1 + 2k_2 + 2k_3 + k_4)\ . \end{aligned} \tag{4.14}$$

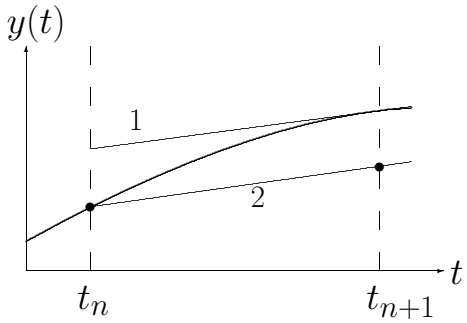
Her er der mere at beregne, men det kan betale sig, fordi man kan tage langt større skridt.

4.4 Baglæns implicit BI

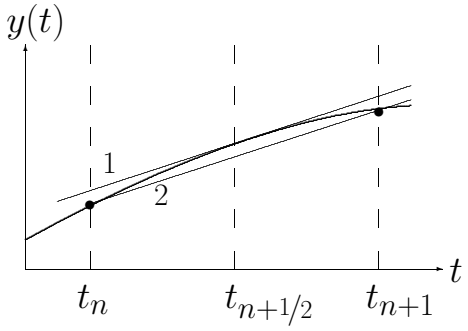
De metoder beskrevet ovenstående er alle **eksplicitte**, *dvs.*, man beregner en ny værdi udelukkende ud fra kendte værdier, med en ligning af formen

$$y_{n+1} = f(y_n, \dots) \tag{4.15}$$

med kun kendte værdier i argumenterne til funktionen på den højre side. Der er imidlertid også **implicitte** metoder, som har visse fordele, hvor den ukendte værdi går ind blandt argumenterne for funktionen. Den mest enkle metode er den der benytter en baglæns difference, og er kaldt **baglæns implicit** eller på engelsk, "backward implicit". Ideen er at benytte sig af den gradient ved den nye (endnu ukendt) værdi y_{n+1}, og bevæge sig frem fra y_n med den gradient. Fig. 4.4 viser dette; linje 1 er gradienten, og linje 2 er parallel med den, fra udgangspunktet. Vi ser, at det giver et resultat der ikke er bedre end det vi får (Fig. 4.2) med Eulermetoden. Faktisk er BI, lige som Eulermetoden, en første-ordens metode, men den kan danne grundlag for bedre, højre-ordens metoder.



Figur 4.4: Et trin med BI



Figur 4.5: Trapezmetoden

Metoden kan udtrykkes matematisk i formen

$$y_{n+1} = y_n + h\, f(y_{n+1}, t_{n+1})\ . \tag{4.16}$$

Afhængigt af hvad funktionen f er, kan det være svært at udtrykke, men ikke normalt. Faktisk benytter man det samme udtryk for approksimationen til derivatet, men man tilordner den til t_{n+1}, hvorfor betegnelsen BI stammer. I tilfældet af vores eksempel (4.2) er der ingen problemer; man skriver

$$\frac{y_{n+1} - y_n}{h} = -y_{n+1} \tag{4.17}$$

hvilket giver

$$y_{n+1} = y_n \frac{1}{1+h}\ . \tag{4.18}$$

4.5 Trapezmetoden

I differentialligningen $y' = f(y,t)$ har vi set, at både Euler- og BI-metoden har den ulempe, at y' bliver approksimeret som et asymetrisk udtryk, (henholdsvis som fremad- og baglæns-approksimation) $(y_{n+1} - y_n)/h$. Fra Fig. 4.5 er det øjensynligt, at denne approksimation passer fint til $t + \frac{1}{2}h$, hvor den udgør en central differens. Hældningen ved midterpunktet kan være ret éns med den af en linje trukket mellem de to punkter, og den tankegang danner også en implicit metode. Man skriver her generelt

$$\frac{y_{n+1} - y_n}{h} = f(y_{n+1/2}, t_{n+1/2})\ . \tag{4.19}$$

Midterværdien for t har man, men ikke den for y, men den kan approksimeres som middelværdien af y_n og den (endnu ukendte) y_{n+1}. Ligningen er derfor

$$\frac{y_{n+1} - y_n}{h} = f(\frac{y_n + y_{n+1}}{2}, t_{n+1/2}) \tag{4.20}$$

og den indeholder den ukendte værdi på højre side. I vores eksempel bliver det til

$$\frac{y_{n+1} - y_n}{h} = -\frac{y_n + y_{n+1}}{2} \tag{4.21}$$

hvilket giver

$$y_{n+1} = y_n \frac{1 - 2h}{1 + 2h} \, . \tag{4.22}$$

Denne metode virker fortræffeligt; den er $O(h^2)$ og er basis for en række beslægtede metoder i mere indviklede sammenhæng, såsom løsning af partielle differentialligninger og systemer af såvel ordinære og partielle differentialligninger.

4.6 Ekstrapolation

BI-metoden i sig selv er ikke ret interessant, da den er unøjagtig; men den egner sig som basis for forbedringer der øger ordenen. Den er, som skrevet, af $O(h)$, og ekstrapolation kan bruges her. Først beregner man en ny y_{n+1} værdi med skridtlængde h; lad os kalde det $y(1)$. Derefter beregner man endnu en ny, $y(2)$, med to skridt, hver af længde $h/2$. Da BI er $O(h)$, ved vi at fejlen i $y(2)$ er kun det halve af den i $y(1)$, og derfor kan vi få et forbedret resultat fra

$$y_{n+1} = 2y(2) - y(1) \, . \tag{4.23}$$

Fejlen er nu $O(h^2)$. Det er dette (og andre metoder, der presser mere ud af BI), der gør BI interessant, blandt nogle andre gode egenskaber metoden har.

4.7 Systemer af differentialligninger

Mange kemiske og fysiske processer udtrykkes som et antal differentialligninger, muligvis koblet sammen, der skal løses samtidigt. Atmosfærisk kemi leverer nogle af de værste eksempler, hvor op til flere hundrede forskellige stoffers kemi er koblet sammen og så danner systemer af flere hundrede differentialligninger. Generelt kan man skrive sådanne systemer i formen

$$\begin{aligned} y_1' &= f_1(y_1, y_2, \cdots, y_N, t) \\ y_2' &= f_2(y_1, y_2, \cdots, y_N, t) \\ &\cdots \\ y_N' &= f_N(y_1, y_2, \cdots, y_N, t) \end{aligned} \tag{4.24}$$

eller i vektornotation

$$\mathbf{y}' = \mathbf{f}(\mathbf{y}, t) \tag{4.25}$$

hvor vektoren $\mathbf{y} = [y_1, y_2, \cdots, y_N]^T$ er de N funktioner af tid t, der skal findes. For at løse sådanne systemer, bruger man de samme metoder allerede beskrevet ovenfor, *dvs.* Euler, RK, BI, trapez *osv.* Lad os bruge et system af to ligninger som eksempel:

$$\left.\begin{array}{rcl} x' &=& -y \\ y' &=& x \end{array}\right\} \tag{4.26}$$

hvor $y' \equiv \mathrm{d}y/\mathrm{d}t$, t værende en tredje, uafhængig løbevariabel. Som altid skal vi have startværdier:

$$\left.\begin{array}{rcl} x(t=0) &=& x_0 \\ y(t=0) &=& y_0 \end{array}\right\} . \tag{4.27}$$

Så kan vi udvikle x- og y-værdier for $t = h, 2h, \cdots$ *osv.* Eulermetoden er simpelthen

$$\left.\begin{array}{rcl} x_{n+1} &=& x_n - h\, y_n \\ y_{n+1} &=& y_n + h\, x_n \end{array}\right\} . \tag{4.28}$$

Runge-Kutta er næsten lige så nem, men nu behøver vi en række $k_1, k_2, \ldots$ for hver af de to funktioner, x og y, og vi skal passe på rækkefølgen, i hvilken vi beregner dem: den er $k_{1x}, k_{1y}; k_{2x}, k_{2y}$ *osv.* For 2.-ordens RK:

$$\left.\begin{array}{rcl} k_{1x} &=& -h\, y_n \\ k_{1y} &=& h\, x_n \end{array}\right\} \qquad \left.\begin{array}{rcl} k_{2x} &=& -h\,(y_n + k_{1y}) \\ k_{2y} &=& h\,(x_n + k_{1x}) \end{array}\right\} . \tag{4.29}$$

Til sidst:

$$\left.\begin{array}{rcl} x_{n+1} &=& x_n + \frac{1}{2}\,(k_{1x} + k_{2x}) \\ y_{n+1} &=& y_n + \frac{1}{2}\,(k_{1y} + k_{2y}) \end{array}\right\} . \tag{4.30}$$

Dette kan let overføres til højereordens RK formler.

Trapezformlen viste sig at være god med en enkelt funktion, så den kan også prøves her:

$$\left.\begin{array}{rcl} x_{n+1} &=& x_n - \frac{h}{2}\,(y_{n+1} + y_n) \\ y_{n+1} &=& y_n + \frac{h}{2}\,(x_{n+1} + x_n) \end{array}\right\} . \tag{4.31}$$

Her opstår der det problem, at vi ikke eksplicit kan løse hver af ligningerne for de nye værdier; de udgør et ligningssystem med de to ubekendte x_{n+1} og y_{n+1}. Lidt omorganisation giver

$$\left[\begin{array}{cc} 1 & \frac{h}{2} \\ -\frac{h}{2} & 1 \end{array}\right] \left[\begin{array}{c} x_{n+1} \\ y_{n+1} \end{array}\right] = \left[\begin{array}{c} x_n - \frac{1}{2}hy_n \\ \frac{1}{2}hx_n + y_n \end{array}\right] .$$

Selvom dette måske ser lidt besværligt ud, kan det betale sig på grund af denne metodes nøjagtighed. Her kan BI med ekstrapolation også anvendes med fordel. Denne metode er meget robust (stabil), når man har at gøre med "stive" systemer, hvor nogle funktioner ændrer sig langt hurtigere end andre. Eulermetoden og også de RK-metoder beskrevet her kan i sådanne situationer svigte og føre til divergens.

4.8 Dynamisk ændring af skridtlængden

Det hænder, at den funktion, vi løser for, har vidt forskellig hældning i t. Det bliver straks logisk at tage små h ved de stejlere dele og større skridt ved de mere flade. Selvom der går lidt regning "til spilde" for at bestemme den passende skridtlængde hver gang, er det alligevel fordelagtigt, fordi man kan spare langt mere end man spilder.

Udgangspunktet her er, at man skal beslutte sig for en acceptabel fejlstørrelse ε. Så gælder det ved en given iteration at bestemme den h, der giver en fejl $\leq \varepsilon$. Det kan lade sig gøre ved brug af ordenen af den metode, man benytter. Som regel er det en højere-orden metode, der bliver anvendt i forbindelse med dette, *f.eks.* 4.-ordens Runge-Kutta, for hvilken vi ved, at fejlen er $O(h^4)$ for et antal skridt til en tid t. Hvis vi først beregner en ny y-værdi, $y(t+h)$, udgående fra $y(t)$, så er der en fejl e_1 vi ikke kender. Lad os kalde y-værdien $y(1)$. Hvis vi nu genberegner en ny $y(2)$ i to trin af $\frac{1}{2}h$ hver, så ved vi, at den nye fejl e_2 er kun $(\frac{1}{2})^4$ eller $\frac{1}{16}$ så stor som e_1. *Dvs.* den første, $y(1)$, værdi kan skrives som

$$y(1) = \hat{y} + e_1 \tag{4.32}$$

hvor $\hat{y}$ er den (ukendte) sande y-værdi ved $t + h$, og ligeledes

$$y(2) = \hat{y} + e_2 \; . \tag{4.33}$$

Hvis vi trækker dem fra hinanden:

$$y(1) - y(2) = e_1 - e_2 \tag{4.34}$$

og, eftersom vi ved, at $e_2 = \frac{1}{16}e_1$,

$$y(1) - y(2) = \tfrac{15}{16}e_1 \approx e_1 \; . \tag{4.35}$$

Differencen mellem de to beregnede y-værdier er altså en rimelig approksimation til selve fejlen e_1. Nu kan vi beregne den passende h^*, fordi vi jo har besluttet hvad ε skal være:

$$h^* = h \left(\frac{\varepsilon}{e_1} \right)^{1/4} \; . \tag{4.36}$$

Hvis $e_1 < \varepsilon$, kan vi bare bruge y_{n+1}, som lige beregnet, og tage den rigtige skridtlængde, h^*, næste gang. Hvis $e_1 > \varepsilon$, gentager vi regnestykket med h^*.

Alt dette virker ret besværligt, men under visse betingelser kan det være til gavn. Man behøver *f.eks.* ikke selv gætte i forvejen, hvad h burde være; programmet finder selv altid den mest passende skridtlængde.

En ulempe ved sådan adaptiv løsning er, at resultatet - altså funktionen y - består af et antal værdier ved tidspunkter, der har nok så tilfældige afstande. Hvis man ønsker lige afstande i t, kan dette være irriterende. Man kan dog hjælpe sig med interpolering på de ønskede (lige) afstande. Metoden viser sig at være meget effektiv.

4.9 Differentialligninger og integration

At løse en differentialligning svarer til at integrere den. Matematisk er det som følgende: Differentialligningen

$$y' = f(y,t) \tag{4.37}$$

bliver integreret i intervallet t_n til t_{n+1} på begge sider:

$$y_{n+1} = y_n + \int_{t_n}^{t_{n+1}} f(y,t)\,\mathrm{d}t\ . \tag{4.38}$$

Funktionen $f(y_n, t_n)$ er kendt, og så kan vi anvende alle de integrationsteknikker i kapitel 3. Vi starter med blokintegration, Fig. 4.6: den nye y-værdi ved t_{n+1} er den gamle, $y(t_n)$ plus integralet over intervallet h . Blokintegrationen betyder, at dette integralstykke er simpelthen $h\,f(t)$, *dvs.*

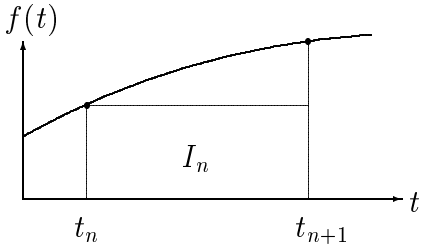
$$y_{n+1} = y_n + h\,f(t_n) \tag{4.39}$$

hvilket er præcis det samme som Eulermetoden beskrevet i afsnit 4.1.

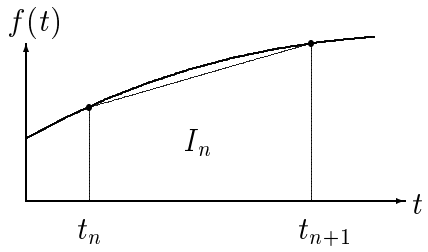
Fra kapitel 3 ved vi, at trapezformlen er bedre end blokintegration. Fig. 4.7 viser dette. Problemet kan være, at f kan være en funktion af både y og t, og at vi jo ikke endnu kender y_{n+1}. Alligevel skriver vi integralet:

$$y_{n+1} = y_n + \tfrac{1}{2}h\,(f(y_n,t_n) + f(y_{n+1},t_{n+1})) \tag{4.40}$$

hvilket svarer til trapez-metoden beskrevet i afsnit 4.5.



Figur 4.6: Blokintegration af en differentialligning

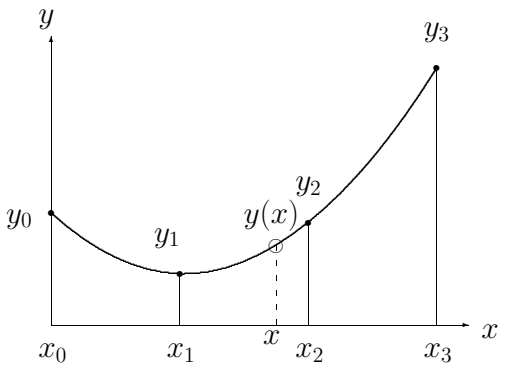


Figur 4.7: Trapezintegration af en differentialligning

Kapitel 5

Interpolation

Man har en funktion defineret ved et antal punkter ved x-værdierne $x_i\,(i = 0 \ldots n)$ (Fig. 5.1), og vi vil få en mellemliggende værdi ved x, (åbent cirkel). Denne proces kaldes interpolation. Afstandene behøver ikke alle at være éns. Interpolation fortages mest ved brug af polynomier tilpasset til et antal støttepunkter,



Figur 5.1: Funktion defineret ved nogle punkter

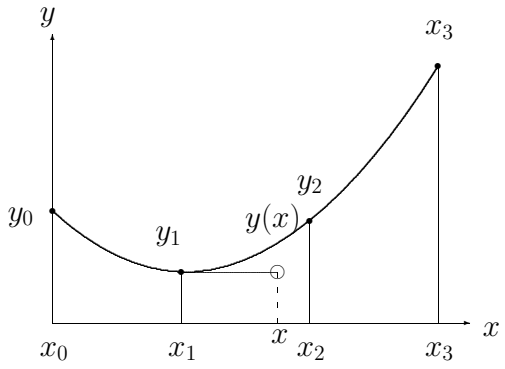
og når der i det følgende er tale om en orden, er det ordenen af polynomiet der er ment. Fejlorden for den beregnede interpolationsværdi er et lidt indviklet emne, men ofte ser man at den er en enhed højrere end ordenen af polynomiet.

5.1 0.-orden "interpolation"

Den simpleste - og groveste - interpolation er den at tage det nærmeste kendte punkt og bruge dens y-værdi. Dette involverer kun et punkt. I Fig. 5.2 ville vi så sige at

$$y(x) = y_1\ . \tag{5.1}$$

Dette svarer til at fortsætte y_1 "vandret" frem til x. Denne "vandrette" funk-



Figur 5.2: 0.-ordens interpolation

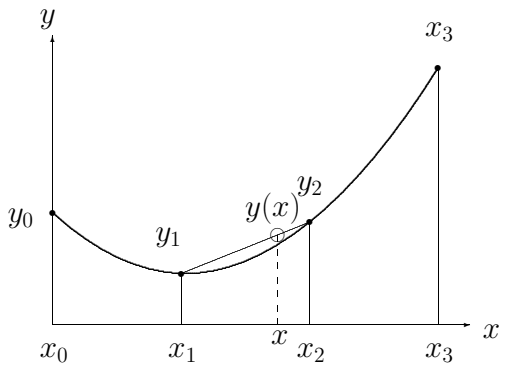
tion er

$$y = y_1 \ . \tag{5.2}$$

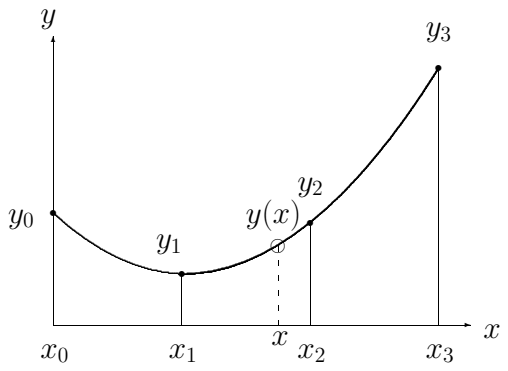
5.2 Lineær (2-pkt) interpolation

Fig. 5.3 viser en 2-punktsinterpolation, som svarer til at "tegne"en ret linje mellem to definerede punkter på kurven, (x_1, y_1) og (x_2, y_2). Simpel geometri fører til

$$\begin{aligned} y &= y_1 + \frac{x - x_1}{x_2 - x_1} \cdot (y_2 - y_1) \\ &= \frac{(x_2 - x)\, y_1 + (x - x_1)\, y_2}{x_2 - x_1} \end{aligned} \tag{5.3}$$



Figur 5.3: 1.ordens (lineær) interpolation



Figur 5.4: 2.ordens (parabolsk) interpolation

men vi kan også gøre det algebraisk: Lad linjen være

$$y = a_0 + a_1 x \tag{5.4}$$

og da vi har to kendte værdier, får vi de to ligninger:

$$\begin{aligned} y_1 &= a_0 + a_1 x_1 \\ y_2 &= a_0 + a_1 x_2 \end{aligned} \tag{5.5}$$

som kan løses for a_0 og a_1:

$$\begin{aligned} a_1 &= \frac{y_2 - y_1}{x_2 - x_1} \\ a_0 &= \frac{x_1 y_2 - x_2 y_1}{x_1 - x_2} \end{aligned} \tag{5.6}$$

og ved at substituere for x får vi

$$\begin{aligned} y &= a_0 + a_1 x \\ &= \frac{x_1 y_2 - x_2 y_1}{x_1 - x_2} + \frac{y_2 - y_1}{x_2 - x_1} x \end{aligned} \tag{5.7}$$

hvilket, efter nogen omordning, giver det samme resultat som i (5.3).

5.3 Parabolsk interpolation

I Fig 5.4 vises der brug af tre punkter (ved x_1, x_2 og x_3). Vi vil afgjort få et bedre resultat for $y(x)$, end en lineær interpolation mellem x_1 og x_2, hvis vi tilpasser en parabel til de tre punkter. På den måde inddrager vi funktionens bøjning i en vis grad. Her kan man ikke længere se den tilpassede funktion, da den næsten er identisk med den vi vil interpolere i. Parablen er:

$$y = a_0 + a_1 x + a_2 x^2 \tag{5.8}$$

og ved at sætte alle tre kendte værdier ind får vi

$$\left.\begin{array}{rcl} y_1 &=& a_0 + a_1 x_1 + a_2 x_1^2 \\ y_2 &=& a_0 + a_1 x_2 + a_2 x_2^2 \\ y_3 &=& a_0 + a_1 x_3 + a_2 x_3^2 \end{array}\right\} . \tag{5.9}$$

Vi kan igen løse dette system for koefficienterne, a_0, a_1 og a_2, og udtrykke $y(x)$ ud fra dem. Læseren spares detaljerne, men slutresultatet efter nogen oprydning er

$$y(x) = \frac{(x-x_2)(x-x_3)}{(x_1-x_2)(x_1-x_3)} y_1 + \frac{(x-x_1)(x-x_3)}{(x_2-x_1)(x_2-x_3)} y_2 + \frac{(x-x_1)(x-x_2)}{(x_3-x_1)(x_3-x_2)} y_3 \, . \tag{5.10}$$

5.4 Generel Lagrange-formel

Hvis 1.-orden er bedre end 0.-, og 2.-orden bedre end 1.-, er det logisk at blive ved med at øge ordenen og derved antallet af punkter, der benyttes til at beregne en interpoleret værdi. Antallet af koefficienterne i det resulterende polynom for n punkter:

$$y = a_0 + a_1 x + \cdots + a_{n-1} x^{n-1} \tag{5.11}$$

bliver større og mere og mere besværligt at løse for. Heldigvis er dette gjort og resultatet er, for en n-punktsinterpolation, Lagranges formel for n punkter,

$$\begin{array}{rcl} y(x) &=& \dfrac{(x-x_2)(x-x_3)\ldots(x-x_n)}{(x_1-x_2)(x_1-x_3)\ldots(x_1-x_n)} y_1 \\ &+& \dfrac{(x-x_1)(x-x_3)\ldots(x-x_n)}{(x_2-x_1)(x_2-x_3)\ldots(x_2-x_n)} y_2 \\ &+& \vdots \\ &+& \dfrac{(x-x_1)(x-x_2)\ldots(x-x_{n-1})}{(x_n-x_1)(x_n-x_2)\ldots(x_n-x_{n-1})} y_n \end{array} \tag{5.12}$$

hvori den i.te koefficient (for y_i) i denne sum dannes af alle produkter $(x - x_j)$ for alle $j \neq i$, i tælleren, og alle produkter $(x_i - x_j)$ også for alle $j \neq i$ i nævneren. Bemærk at (5.10) er et specielt tilfælde af (5.12) med $n = 3$.

5.5 Neville-metoden

Lagrange-metoden, vist i sidste afsnit, ser ret besværlig ud. Men ved en række argumenter er det muligt at komme frem til en lettere metode, der svarer fuldstændigt til Lagrange-metoden. Den har yderligere den fordel, at man ender

med et skøn over fejlen, den beregnede $y(x)$-værdi er behæftet med, som totalt mangler i Lagrange.

Princippet er at gå frem i ordenen fra 0.te op til $(n-1)$.te, og forbedre sine nye $y(x)$-skøn som en funktion af de gamle med grundlaget i Lagrange-formlen. Hvis vi ser på Fig. 5.2, er der to mulige 0.-orden interpolationer for $y(x)$: y_1 og y_2. De to danner altså et sæt etpunkts approksimationer, som i Nevillemetoden kaldes P_0, P_1 *osv.*, hvor antallet af indekser indikerer antallet af punkter brugt til den tilsvarende interpolation (her: 1). De to, P_1 og P_2, kan nu føre til en 2-punkts, første-ordens interpolation (som er $y(x)$ vist i Fig. 5.2); i afsnit 5.1 fik vi udtrykket for dette, og vi benytter nu 0.-ordens funktioner $P_1 = y_1$ og $P_2 = y_2$, og får

$$P_{12} = \frac{(x_2 - x)\, P_1 + (x - x_1)\, P_2}{(x_2 - x_1)} \tag{5.13}$$

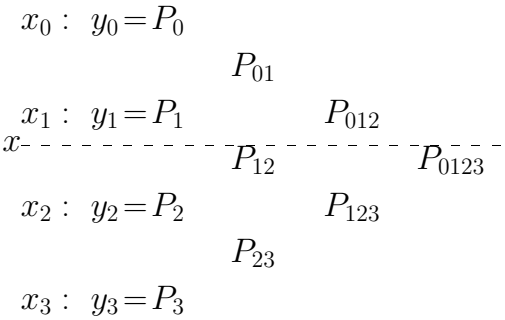
dvs. vi har forbedret vores to groveste formler, P_1 og P_2, til en bedre, P_{12}. Der er en generel rekursiv formel for alle P. Ordet "rekursiv" betyder, at en ny P-polynom udledes af lavere P-polynomer. Formlen er

$$P_{i(i+1)\ldots(i+m)} = \frac{(x - x_{i+m})\, P_{i(i+1)\ldots(i+m-1)} + (x_i - x)\, P_{(i+1)(i+2)\ldots(i+m)}}{x_i - x_{i+m}} \,. \tag{5.14}$$

P_{12} kan udledes fra denne formel ved at indsætte $i = 1$ og $m = 1$, hvilket giver (5.3). Man kan anskueliggøre processen med "Neville-diagrammet", *f.eks.* for fire punkter, se Fig. 5.5.

I praksis vælger man (*dvs.* vælger `POLINT`) det polynomium der er baseret på det ønskede antal punkter, og ligger nærmest til den x-værdi hvor der skal interpoleres. For eksempel, i Fig. 5.5, afhængig af det ønskede n, kunne man bruge P_1 $(n = 1)$, P_{12} $(n = 2)$, P_{012} $(n = 3)$ eller P_{0123} $(n = 4)$.

Faktisk benytter `POLINT` sig af en forbedring af denne metode. Det er nemlig ikke nødvendigt at beregne alle P'er fra bunds. I stedet beregner programmet



Figur 5.5: Neville-diagram

forskellen fra et P-niveau til et andet, og der opstå så to rekursive formler for disse forskelle.

Lad os sige, at vi har approksimationerne P_1 og P_2, og bruger dem til at udregne P_{12}. Vi kan udtrykke en ændringsstørrelse C_{11}, således at

$$P_{12} = P_1 + C_{11} \tag{5.15}$$

og, derfor er den nye approksimation lig med

$$y_{app} = P_1 + C_{11} \ . \tag{5.16}$$

C_{11} fungerer altså som en korrektion på P_1. Ligeledes kan vi udregne en anden, som korrektion på P_2. Systemet er, at (se Fig. 5.5) når vi går nedad i Neville-diagrammet, har vi korrektioner C og når vi går opad, korrektioner D. De er mærket med subscripts, C_{mi} og D_{mi}, som er givet fra udtrykkene

$$C_{mi} = P_{i\ldots(i+m)} - P_{i\ldots(i+m-1)} \tag{5.17}$$

og

$$D_{mi} = P_{i\ldots(i+m)} - P_{i+1\ldots(i+m)} \ . \tag{5.18}$$

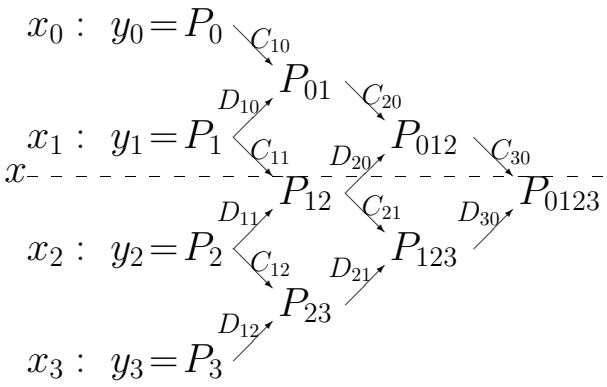
Fig. 5.6 viser billedet. Det første indekstal nummererer rækken, det andet "e-tagen" nedad.

Der er også rekursive udtryk for disse:

$$C_{m+1,i} = \frac{(x_i - x)(C_{m,i+1} - D_{m,i})}{x_i - x_{i+m+1}} \tag{5.19}$$

og

$$D_{m+1,i} = \frac{(x_{i+m+1} - x)(C_{m,i+1} - D_{m,i})}{x_i - x_{i+m+1}} \ . \tag{5.20}$$



Figur 5.6: Neville-diagrammet med C og D mærkede

Interpolationen bliver så til sidst en sum af disse forskelle plus den nærmeste y-værdi der ligger nærmest x. Den procedure er ikke blot nemmere, den giver også til sidst et fejlskøn, som er lig med den sidst beregnede forskel. I vores eksempel ville den bedste fire-punktsapproksimation så blive summen

$$y_{app} = P_2 + C_{11} + D_{20} + C_{30} = P_{0123} \tag{5.21}$$

og den skønnede fejl er lig med C_{31}, det sidst tillagte led.

Hvad er mest effektiv? Man kan tælle antal operationerne. Man regner ofte med, at en addition/subtraktion og en multiplikation/division tager samme computertid, så vi skal bare tælle antallet af dem. Ser man på formlerne og beregner antallet af operationerne, finder man ud af, at de rekursive P'er tager mindst tid, og de andre to ligger ret tæt på hinanden. Man foretrækker alligevel metoden med C og D, selvom den faktisk tager mest tid, fordi man får et fejlskøn ud af den, som de andre to metoder ikke leverer.

5.6 Hermitesk interpolation

Man kan løfte ordenen af en polynomial interpolation uden at bruge flere punkter; metoden er udviklet af den fransk matematiker Hermite. Den baseres på, at man ud over at kende funktionens værdier ved hvert punkt, også kender deres (første) afled. Det giver ekstra information, og så kan man anvende et polynomium af en højere orden med det givne antal punkter. Metoden er beskrevet meget klart af Kopal [6, s.31]. I bogen præsenterer han den følgende tabel:

$$\begin{array}{llll} f(x_0) & f'(x_0) & f''(x_0) & \cdots \\ f(x_1) & f'(x_1) & f''(x_1) & \cdots \\ f(x_2) & f'(x_2) & f''(x_2) & \cdots \\ \vdots & \vdots & \vdots & \\ f(x_n) & f'(x_n) & f''(x_n) & \cdots \end{array}$$

Det vi indtil nu har brugt svarer til den første søjle, hvor der er funktionernes værdier ved $n+1$ positioner $x_0 \cdots x_n$. Man kan også benytte enhver række, som så svarer til brug af Taylors række med deres velkendte koefficienter. Hermites metode benytter de første to søjler, dvs. funktionsværdierne samt deres første afledte. Vi skriver, for at gøre notationen mere kompakt, $y_i = f(x_i)$. Lad os gå frem med udgang i den simpleste etpunktsinterpolation om punktet ved x_1, og tilpasse en ret linje

$$y = a_0 + a_1 x \; ; \tag{5.22}$$

vi kan skrive to ligninger:

$$\begin{aligned} y_1 &= a_0 + a_1 x_1 \\ y_1' &= a_1 \; . \end{aligned} \tag{5.23}$$

Det er nemt at beregne de to koefficienter, og vi har løftet ordenen om en enhed.

Nu bruger vi to punkter, x_1 og x_2, og tilpasser en parabel

$$y = a_0 + a_1 x + a_2 x^2 \tag{5.24}$$

til dem. Vi kan vælge at bruge enten derivatet ved x_1 eller x_2 og vi vælger det første. Så skriver vi tre ligninger

$$\begin{aligned} y_1 &= a_0 + a_1 x_1 + a_2 x_1^2 \\ y_2 &= a_0 + a_1 x_2 + a_2 x_2^2 \\ y_1' &= \quad a_1 + 2a_2 x_1 \ . \end{aligned} \tag{5.25}$$

Igen kan vi beregne de tre koefficienter og ende med en bedre interpolation fra de to punkter. Men vi kan også bruge begge afled og derved benytte en tilpasset kubikligning

$$y = a_0 + a_1 x + a_2 x^2 + a_3 x^3 \ . \tag{5.26}$$

Vi skriver fire ligninger:

$$\begin{aligned} y_1 &= a_0 + a_1 x_1 + a_2 x_1^2 + a_3 x_1^3 \\ y_2 &= a_0 + a_1 x_2 + a_2 x_2^2 + a_3 x_2^3 \\ y_1' &= \quad a_1 + 2a_2 x_1 + 3a_3 x_1^2 \\ y_2' &= \quad a_1 + 2a_2 x_2 + 3a_3 x_2^2 \end{aligned} \tag{5.27}$$

og igen løser for de fire koefficienter (nok ved at bruge et underprogram).

Kopal afprøver den sidstnævnte variant til at interpolere mellem to sinusværdier ved argumentet 0.5, med givne værdier ved 0.4 og 0.6. Når man har sinusværdier, kan man nemt beregne deres afledte, cosinusværdierne. Vi har afprøvet alle fire muligheder nævnt ovenfor, og resultatet ses i tabellen 5.1.

Fordelen af den hermitesk metode er indlysende.

Tabel 5.1: Fejlene i interpolation ved $x = 0.5$ mellem to sinusfunktionens værdier. Den sande værdi er 0.479426

	Interpoleret værdi	Fejl	% fejl
Simpel etpunkt formel	0.389418	-0.090007	-18.7740
Hermitesk etpunktsformel	0.481524	0.002009	0.4378
Parabolsk topunktsformel	0.479277	-0.000148	-0.0309
Kubik topunktsformel	0.479424	-0.000002	-0.0004

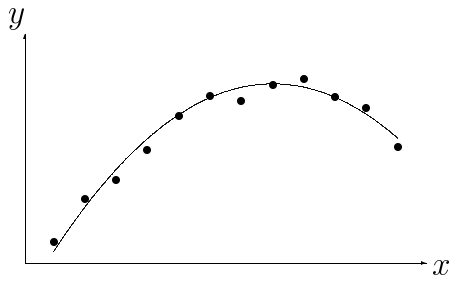
Kapitel 6

Mindste Kvadraters Tilpasning

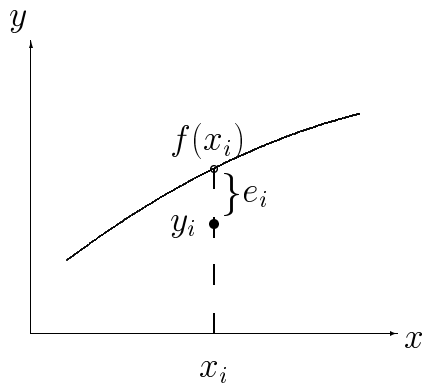
Vi har ofte at gøre med et antal (x, y) punkter, enten målt eller beregnet, som er behæftet med tilfældige fejl i y-retningen; i de fleste tilfælde er der ikke nævneværdige fejl på x. Ofte har vi en idé om, hvilken slags funktion $f(x)$ underligger en sådan punktskare, og opgaven er at bestemme de justerbare parametre i denne funktion, som får funktionen til at passe "bedst" til punktskaren.

En anden situation er, at i stedet for en funktion, der underligger punkterne - altså en matematisk beskrivelse af de fysiske forhold ved de målte eller beregnede punkter - vil vi blot finde en funktion, som kan efterligne sådan en underliggende funktion. Der er visse funktioner, der er god til sådan efterligning, såsom polynomier, summer af eksponentialer eller sommetider sinus- og cosinusbølger (tænk Fourierrækker). Også her bestræber vi os på at finde funktionens parametre for en "bedste" tilpasning.

Eftersom der er fejl på y-værdierne, kan $f(x)$ ikke gå igennem alle (eller måske nogle) af punkterne, og vi er nødt til at definere, hvad vi mener med "bedst". I Fig. 6.1 vises et antal N punkter med fejl på og en funktion, der er blevet tilpasset således at den passer så godt som muligt, i en forstand der skal defineres nedenfor. Fig. 6.2 viser en detalje fra denne tilpasning, hvor vi



Figur 6.1: Et sæt punkter og en tilpasset funktion



Figur 6.2: Detalje fra Fig. 6.1

fokusserer på et enkelt punkt ved x_i, og den forbigående tilpassede funktion $f(x_i)$, hvor nu punktets værdi y_i afviger fra den tilpassede funktion med en fejl på e_i. Vi skal operere med disse fejl for at finde den bedste tilpassede funktion.

Mindste-kvadraters metoden består i, at man definerer "bedste" tilpasning, således at man finder det parametersæt i funktionen $f(x)$, som giver et minimum i summen S af kvadraterne af alle e:

$$S = \sum_{i=1}^{N} e_i^2 = \sum_{i=1}^{N} \left(y_i - f(x_i)\right)^2 \ . \tag{6.1}$$

Ofte kender vi spredningen af fejlene i y, σ_y, og de er muligvis forskellige for de forskellige punkter - *f.eks.* måske var der forskellige måleskalaer i nogle x-områder. Dette generaliserer vi ved at tilegne hvert punkt i sin spredning σ_i; tilpasningen er nu at minimere χ^2,

$$\chi^2 = \sum_{i=1}^{N} \left(\frac{y_i - f(x_i)}{\sigma_i} \right)^2 \ . \tag{6.2}$$

Hvis σ_i ikke er kendt, sætter man dem alle lige med 1. Ved at dividere hver fejl e_i med σ_i taler vi nu om en ny fejl, som er relativ til spredningen σ_i, og det er summen af kvadraterne af disse normaliserede fejl vi minimerer. Som skal beskrives nedenfor, giver den således beregnede χ^2 mulighed for at bedømme, hvor god tilpasningen er; hvis vi ikke kender spredningerne, er dette ikke muligt, men en tilpasning er alligevel mulig.

Lad os sige, at vi tilpasser funktionen $f(x, p_1, p_2, \ldots p_M)$, hvor sættet $p_1, p_2, \ldots, p_M$ er M parametre det beskriver funktionen. For eksempel, hvis funktionen er en ret linje, har vi to parametre, da $f(x) = p_1 + p_2 x$. En tilpasning er således ensbetydende med at finde det bedste sæt parametre. For hver

af dem skal χ^2 have et minimum, hvilket giver betingelsen

$$\frac{\partial \chi^2}{\partial p_i} = 0 \ . \tag{6.3}$$

Vi skal altså differentiere (6.1) eller (6.2) med hensyn til alle parametrene, og sætte alle derivater lige med nul. Det giver et sæt af M ligninger, som kan løses for alle M parametre, med mere eller mindre besvær, afhængig af om ligningerne er lineære eller ikkelineære. Begge situationer bliver beskrevet her.

6.1 Lineære mindste kvadrater

Det drejer sig mest om polynomiumsfunktioner. Ordet "lineær" henviser her ikke til selve funktionen, men det lineære sæt ligninger i de M ubekendte parametre, der skal løses for. Et polynomium med graden $M-1$ er funktionen

$$f(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{M-1} x^{M-1} \ . \tag{6.4}$$

Konstant

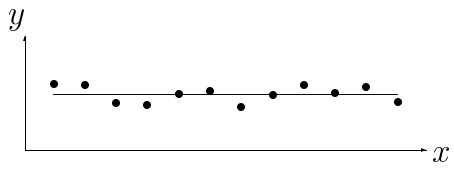
Vi begynder med et simpelt eksempel for at illustrere fremgangsmåden, et "polynom" som er bare en konstant ($M = 1$):

$$f(x) = a_0 \ . \tag{6.5}$$

Den skal tilpasses det sæt punkter vist i Fig. 6.3 (den vandrette linje vist der er den tilpassede funktion).

Vi benytter os af (6.2), det vil sige, vi kender spredningerne for alle N punkter i sættet. Der er kun en parameter, a_0, så vi differentierer med hensyn til den, og får ligningen

$$\frac{\partial \chi^2}{\partial a_0} = 2 \sum_{i=1}^{N} \frac{(y_i - a_0)}{\sigma_i^2}(-1) = 0 \ . \tag{6.6}$$



Figur 6.3: Et sæt punkter tilpasset en konstant

Det giver

$$a_0 = \frac{\sum_{i=1}^{N} y_i/\sigma_i^2}{\sum_{i=1}^{N} 1/\sigma_i^2} \; . \tag{6.7}$$

Hvis alle σ_i er éns, bliver det til

$$a_0 = \frac{1}{N} \sum_{i=1}^{N} y_i \tag{6.8}$$

hvilket vi genkender som middelværdien af alle y_i.

Ret linje (regressionslinje)

Nu sætter vi $M = 2$, *dvs.* vi tilpasser en ret linje til N punkter, så

$$f(x) = a_0 + a_1 x \; . \tag{6.9}$$

Her må vi differentiere (6.2) med hensyn til både a_0 og a_1, og vi får de to ligninger

$$\begin{aligned} \frac{\partial \chi^2}{\partial a_0} &= 2 \sum_{i=1}^{N} \left(\frac{y_i - a_0 - a_1 x_i}{\sigma_i^2} \right) (-1) = 0 \\ \frac{\partial \chi^2}{\partial a_1} &= 2 \sum_{i=1}^{N} \left(\frac{y_i - a_0 - a_1 x_i}{\sigma_i^2} \right) (-x_i) = 0 \end{aligned} \tag{6.10}$$

som fører til det lineære system

$$\begin{aligned} a_0 \sum_{i=1}^{N} \frac{1}{\sigma_i^2} + a_1 \sum_{i=1}^{N} \frac{x_i}{\sigma_i^2} &= \sum_{i=1}^{N} \frac{y_i}{\sigma_i^2} \\ a_0 \sum_{i=1}^{N} \frac{x_i}{\sigma_i^2} + a_1 \sum_{i=1}^{N} \frac{x_i^2}{\sigma_i^2} &= \sum_{i=1}^{N} \frac{x_i y_i}{\sigma_i^2} \; . \end{aligned} \tag{6.11}$$

Vi introducerer her en mere bekvem notation: Lad

$$S_{nm} \equiv \sum_{i=1}^{N} \frac{x_i^n y_i^m}{\sigma_i^2} \; . \tag{6.12}$$

Vi kan så omskrive (6.11) til

$$\begin{aligned} S_{00}\, a_0 + S_{10}\, a_1 &= S_{01} \\ S_{10}\, a_0 + S_{20}\, a_1 &= S_{11} \end{aligned} \tag{6.13}$$

som har løsningerne

$$a_0 = \frac{S_{20}S_{01} - S_{10}S_{11}}{\Delta}$$
$$a_1 = \frac{S_{00}S_{11} - S_{10}S_{01}}{\Delta} \tag{6.14}$$

hvor determinanten er

$$\Delta = S_{00}S_{20} - S_{10}^2 \; . \tag{6.15}$$

En måde at løse det ovenstående system på er at udtrykke (6.13) i matrix-vektor form:

$$\mathbf{Aa} = \mathbf{b} \tag{6.16}$$

hvor

$$\mathbf{A} \equiv \begin{bmatrix} S_{00} & S_{10} \\ S_{10} & S_{20} \end{bmatrix} \; , \tag{6.17}$$

$$\mathbf{a} \equiv \begin{bmatrix} a_0 \\ a_1 \end{bmatrix} \tag{6.18}$$

og

$$\mathbf{b} \equiv \begin{bmatrix} S_{01} \\ S_{11} \end{bmatrix} \; . \tag{6.19}$$

Løsningen er så

$$\mathbf{a} = \mathbf{A}^{-1}\mathbf{b} \tag{6.20}$$

og da det kun drejer sig om et 2×2 system, er det ikke svært at komme frem til at

$$\mathbf{A}^{-1} = \frac{1}{\Delta} \begin{bmatrix} S_{20} & -S_{10} \\ -S_{10} & S_{00} \end{bmatrix} \tag{6.21}$$

hvilket fører til løsningerne (6.14). Vi kommer til at bruge det ovenstående i det næste afsnit.

Usikkerhed på parametrene

Vi benytter det simple eksempel, $M = 2$, for at beskrive hvordan man kommer frem til usikkerhederne på de beregnede parametre. Svaret kommer fra fejlpropagationssætningen. Hvis vi har en funktion f af flere variabler p_i, hvor hver af disse har en standarddeviation σ_i eller en varians σ_i^2, så er variansen for funktionen, σ_f^2, givet ved

$$\sigma_f^2 = \sum_i \sigma_i^2 \left(\frac{\partial f}{\partial p_i} \right)^2 \; . \tag{6.22}$$

Både a_0 og a_1 er funktioner af alle y_i, som her er de variabler, der er usikre, og derfor skriver vi (6.22)

$$\sigma_{a_0}^2 = \sum_{i=1}^{N} \sigma_i^2 \left(\frac{\partial a_0}{\partial y_i} \right)^2 \tag{6.23}$$

og

$$\sigma_{a_1}^2 = \sum_{i=1}^{N} \sigma_i^2 \left(\frac{\partial a_1}{\partial y_i} \right)^2 \ . \tag{6.24}$$

Vi opererer altså med derivaterne $\partial a_0/\partial y_i$ og $\partial a_1/\partial y_i$, for alle $i = 1 \ldots N$. Når vi ser på løsningen for a_0 (6.14) så ser vi, at der er nogle størrelser der er uafhængige af y, og nogle, der er afhængige af y. For at fremhæve de sidste, skriver vi løsningen for a_0 i formen

$$a_0 = \frac{1}{\Delta} \left(S_{20} \sum_{i=1}^{N} \frac{y_i}{\sigma_i^2} - S_{10} \sum_{i=1}^{N} \frac{x_i y_i}{\sigma_i^2} \right) \tag{6.25}$$

(da Δ, S_{20} og S_{10} ikke er afhængige af y). Nu kan vi differentiere for et enkelt i:

$$\frac{\partial a_0}{\partial y_i} = \frac{1}{\Delta} \left(S_{20} \cdot \frac{1}{\sigma_i^2} - S_{10} \cdot \frac{x_i}{\sigma_i^2} \right) \tag{6.26}$$

og straks får vi, ved at anvende (6.22)

$$\sigma_{a_0}^2 = \sum_{i=1}^{N} \frac{1}{\Delta^2} \left\{ S_{20}^2 \cdot \frac{1}{\sigma_i^2} - 2 S_{20} S_{10} \cdot \frac{x_i}{\sigma_i^2} + s_{10}^2 \cdot \frac{x_i^2}{\sigma_i^2} \right\} \ . \tag{6.27}$$

Det fører direkte til

$$\sigma_{a_0}^2 = \frac{1}{\Delta^2} \left\{ S_{20}^2 S_{00} - 2 S_{20} S_{10} S_{10} + S_{10}^2 S_{20} \right\} \tag{6.28}$$

hvilket, efter lidt oprydning (og med blik på (6.15)), giver

$$\sigma_{a_0} = \frac{S_{20}}{\Delta} \ . \tag{6.29}$$

En lignende procedure for a_1 giver

$$\sigma_{a_1} = \frac{S_{00}}{\Delta} \ . \tag{6.30}$$

Der er dog en anden fremgangsmåde, der tager udgangspunkt i løsningen (6.20) af matrix-vektorligningen (6.16). Ser man nøje på løsningen, *dvs.* matricens inverse (6.21), er de to ovenstående resultater for σ_{a_0} og σ_{a_1} at se i diagonalen af den inverterede matrix. Man kan altså få usikkerhederne ved bare at invertere matricen $\mathbf{A}$ i (6.16). For vores ret-linjeeksempel har vi definitionen af $\mathbf{A}$ i ligning (6.17), og matricen her let inverteres til (6.21), fra hvilket vi umiddelbart kan aflæse løsningerne (6.27) og (6.28).

Det viser sig, at denne procedure er frugtbar ikke kun ved tilpasning med højere polynomier, men også i ikkelinære tilfælde (se afsnit 6.4).

Generelt polynomium

Vil vi tilpasse højere-ordens polynomier, bruger vi samme procedure som for eksemplerne fra de ovenstående afsnit. Matricerne bliver større. For et polynomium som i (6.4), hvor vi skal estimere M parametre, skal vi løse samme ligning (6.16) for $a_0 \ldots a_M$. Vi omskriver polynomiumsudtrykket i den (lidt) simplere form,

$$f(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_p x^p \tag{6.31}$$

hvor $p = M - 1$, eller graden af polynomiet. Så er matricen $\mathbf{A}$ givet ved

$$\mathbf{A} = \begin{bmatrix} S_{00} & S_{10} & S_{20} & \cdots & S_{p0} \\ S_{10} & S_{20} & S_{30} & \cdots & \\ S_{20} & S_{30} & S_{40} & \cdots & \\ \vdots & & & & \vdots \\ S_{p0} & \cdots & & & S_{2p0} \end{bmatrix} \tag{6.32}$$

og

$$\mathbf{b} = [S_{01}\ S_{11}\ \ldots\ S_{p1})^T \tag{6.33}$$

med S'erne som definerede i (6.12). For en parabel har vi således

$$\begin{bmatrix} S_{00} & S_{10} & S_{20} \\ S_{10} & S_{20} & S_{30} \\ S_{20} & S_{30} & S_{40} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} S_{01} \\ S_{11} \\ S_{21} \end{bmatrix} \tag{6.34}$$

eller, eksplicit

$$\begin{bmatrix} \sum_{i=1}^{N} \frac{1}{\sigma_i^2} & \sum_{i=1}^{N} \frac{x_i}{\sigma_i^2} & \sum_{i=1}^{N} \frac{x_i^2}{\sigma_i^2} \\ \sum_{i=1}^{N} \frac{x_i}{\sigma_i^2} & \sum_{i=1}^{N} \frac{x_i^2}{\sigma_i^2} & \sum_{i=1}^{N} \frac{x_i^3}{\sigma_i^2} \\ \sum_{i=1}^{N} \frac{x_i^2}{\sigma_i^2} & \sum_{i=1}^{N} \frac{x_i^3}{\sigma_i^2} & \sum_{i=1}^{N} \frac{x_i^4}{\sigma_i^2} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{N} \frac{y_i}{\sigma_i^2} \\ \sum_{i=1}^{N} \frac{x_i\, y_i}{\sigma_i^2} \\ \sum_{i=1}^{N} \frac{x_i^2\, y_i}{\sigma_i^2} \end{bmatrix} . \tag{6.35}$$

Det lineære system $\mathbf{Aa} = \mathbf{b}$ skal så "bare" løses for alle parametrene. Man vil nok ikke normalt gør det ved at invertere matricen $\mathbf{A}$; men hvis man gør, så har man samtidigt usikkerhederne på parametrene i diagonalen, som beskrevet i sidste afsnit. Desuden, som beskrives i kapitel 7, er der ret effektive algoritmer for matrixinvertering.

For en ret linje reduceres systemet til (6.17) med løsningen (6.21).

6.2 Ukendt spredning

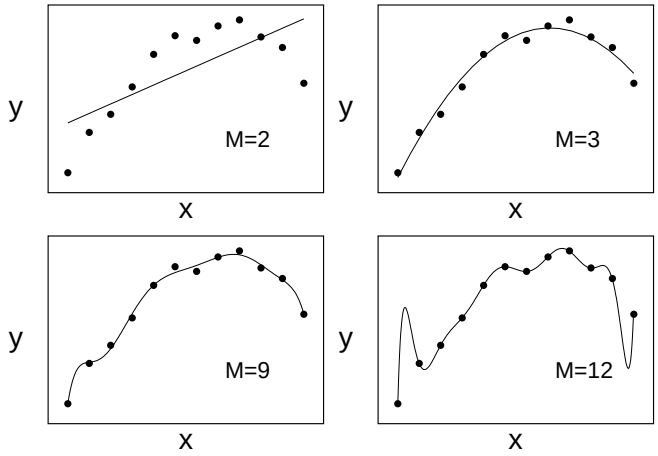
I det foregående blev det forudsat, at vi kender alle σ_i; det blev nævnt, at vi arbitrært sætter dem alle lig med 1, hvis vi ikke kender dem. Vi vil i så fald alligevel kunne beregne sættet **a** for et minimum i χ^2, men selve dette minimum er så skaleret med en ukendt faktor. I så fald kan vi ikke (næste afsnit) bedømme, hvor godt en tilpasning passer, bortset fra en visuel vurdering, som faktisk ikke er så ringe. Tilpasninger for forskellige polynomiumsgrader i Fig. 6.4 viser tydeligt forskel mellem det upassende tilfælde $M = 2$ og $M = 3$, og også at *f.eks.* $M \geq 9$ er for meget.

6.3 Tilpasningens duelighed - goodness of fit

Hvis vi har valgt en passende funktion at tilpasse vores punktskare til, så burde den minimale $\chi^2/(N - M)$ afspejle varianserne σ_i^2. I andre ord, den tilpassede kurve burde ligge inden for usikkerhedsgrænserne af alle punkterne. Det forekommer, at man vælger en upassende funktion, og det kan man afsløre ved at se på det minimerede $\chi^2/(N - M)$. En upassende kurve, som vi skal se, kan være for dårlig, eller også for god. Bedømmelsestallet er $\chi^2/(N - M)$, hvor $N - M$ er antallet af frihedsgrader. Ofte er $N >> M$, og vi kan bruge χ^2/N. Der er de tre tilfælde:

$$\begin{aligned} \chi^2/(N-M) &>> 1 \quad \text{(dårlig tilpasning)} \\ \chi^2/(N-M) &\approx 1 \quad \text{(god tilpasning)} \; . \\ \chi^2/(N-M) &<< 1 \quad \text{(for god tilpasning)} \end{aligned} \tag{6.36}$$

Fig. 6.4 viser eksempler på disse. Det første tilfælde holder, hvis den valgte funktion er upassende, og ligger uden for nogle eller mange punkternes usik-



Figur 6.4: Nogle fits til et sæt af 12 punkter

kerhedsgrænser, *f.eks.* for $M = 2$ i Fig. 6.4, hvor en ret linje blev "tilpasset" en punktskare som tydeligvis ikke ligger på en ret linje. Det sidste tilfælde kan eksempelvis ske, hvis man vælger en polynomfunktion med en så høj grad, at funktionen går præcis igennem alle punkterne. Man ser spor af det ved $M = 9$, og mest tydeligt for $M = N = 12$, hvor tilpasningen følger alle punkterne slavisk. Det er de mellemliggende polynomiumsgrader, der virker bedst, her $M = 3$. Tabel 6.1 viser værdierne af $\chi^2/(N - M)$ for alle M-værdierne.

Tabel 6.1: $\chi^2/(N - M)$ for tilpasningen af funktionen i Fig. 6.4 mod M

M	$\chi^2/(N - M)$
2	10.8
3	0.56
4	0.38
5	0.42
6	0.50
7	0.51
8	0.45
9	0.37
10	0.10
11	0.02
12	0

Vi ser, at der ikke er den store forskel i området $3 \leq M \leq 8$ hvor dog tilfældet $M = 3$ har den værdi, der ligger tættest på 1; men for $M > 8$ daler værdierne ned mod nul, hvor kurverne altså passer for godt.

6.4 Ikke-lineære mindste kvadrater

Hvis man til sit punktsæt vil tilpasse en funktion hvor differentiering af udtrykket for χ^2 ikke er lineær i parametrene, bliver det sværere. Et simpelt eksempel kunne være

$$y = a_1 \exp(a_2 x) \tag{6.37}$$

hvor vi har et problem med a_2: stiller vi den sædvanlige ligning op for χ^2:

$$\chi^2 = \sum_{i=1}^{N} \frac{1}{\sigma_i^2} \{y_i - a_1 \exp(a_2 x_i)\}^2 \tag{6.38}$$

og minimerer χ^2 ved at søge efter $\partial\chi^2/\partial a_1 = \partial\chi^2/\partial a_2 = 0$, bliver det hurtigt klart, at vi ikke uden problemer kan løse ligningssystemet, der kommer ud af

det:

$$\begin{aligned}\frac{\partial\chi^2}{\partial a_1} &= 0 = -2\sum_{i=1}^{N}\frac{1}{\sigma_i^2}\left\{y_i - a_1\exp(a_2x_i)\right\}\exp(a_2x_i)\\ \frac{\partial\chi^2}{\partial a_2} &= 0 = -2\sum_{i=1}^{N}\frac{1}{\sigma_i^2}\left\{y_i - a_1\exp(a_2x_i)\right\}a_1x_i\exp(a_2x_i)\end{aligned} \tag{6.39}$$

og der skal gribes til særlige midler, beskrevet *f.eks.* i Press & al. [4], men ikke her. Der beskrives imidlertid en (forholdsvis) nem metode til at komme uden om problemet: lineariseringsmetoden. Hvis vi allerede har et sæt af parameterværdier $(a_1, a_2, \ldots, a_M)$, som er nogenlunde nær det korrekte sæt, der skal findes, kan vi bruge Taylorekspansionen til at korrigere vores sæt. Lad sættet, vi har, være $\mathbf{a}$ og lad den række $\partial\chi^2/\partial a_i$ funktioner, (som alle burde være lig nul, men som ikke er det) være vektoren $\mathbf{F}$. En korrektion $\delta\mathbf{a}$ af $\mathbf{a}$ skulle altså give

$$\mathbf{F}\left(\mathbf{a}+\delta\mathbf{a}\right) = 0 \tag{6.40}$$

og efter Taylor kan vi skrive

$$\mathbf{F}(\mathbf{a}+\delta\mathbf{a}) = \mathbf{F}(\mathbf{a}) + \mathbf{F}'(\mathbf{a})\cdot\delta\mathbf{a} + \ldots \tag{6.41}$$

(vi ser bort fra led højere end $\mathbf{F}'$). Følgeligt kan vi skrive

$$\mathbf{F}(\mathbf{a}) + \mathbf{F}'(\mathbf{a})\cdot\delta\mathbf{a} = 0 \tag{6.42}$$

og det giver et sæt ligninger lineært i sættet $\delta\mathbf{a}$, korrektionssættet. Det er nødvendigt at evaluere både $\mathbf{F}$ og derivaterne $\mathbf{F}'$, men det kan altid lade sig gøre. Når man har løst for $\delta\mathbf{a}$, finder man så ud af, at $\mathbf{F}\left(\mathbf{a}+\delta\mathbf{a}\right)$ alligevel ikke er lig med nul, fordi man jo har forkortet Taylorrækken; som regel vil man dog være kommet nærmere. Man lægger altså $\delta\mathbf{a}$ til $\mathbf{a}$ og bruger summen som det nye sæt $\mathbf{a}$. Normalt skal man gentage hele processen nogle gange, før man når et acceptabelt sæt $\mathbf{a}$. Det, der er acceptabelt, skal defineres. Det kan *f.eks.* være at en eller anden norm af $\mathbf{F}$ er mindre end et tal, man selv sætter.

Lad os tage den ovennævnte funktion, $y = a_1\exp(a_2x)$ som eksempel. Her,

$$\frac{\partial\chi^2}{\partial a_1} = F_1 \qquad \text{og} \qquad \frac{\partial\chi^2}{\partial a_2} = F_2 \tag{6.43}$$

dvs.

$$\begin{aligned}F_1\left(a_1,a_2\right) &= -2\sum_{i=1}^{N}\frac{1}{\sigma_i^2}\left\{y_i - a_1\exp(a_2x_i)\right)\exp(a_2x_i)\\ F_2\left(a_1,a_2\right) &= -2\sum_{i=1}^{N}\frac{1}{\sigma_i^2}\left\{y_i - a_1\exp(a_2x_i)\right\}a_1x_i\exp(a_2x_i)\end{aligned} \tag{6.44}$$

og for forbedringerne δa_1 og δa_2,

$$\begin{aligned} F_1(a_1+\delta a_1, a_2+\delta a_2) &= F_1(a_1,a_2) + \delta a_1 \frac{\partial F_1(a_1,a_2)}{\partial a_1} + \delta a_2 \frac{\partial F_1(a_1,a_2)}{\partial a_2} = 0 \\ F_2(a_1+\delta a_1, a_2+\delta a_2) &= F_2(a_1,a_2) + \delta a_1 \frac{\partial F_2(a_1,a_2)}{\partial a_1} + \delta a_2 \frac{\partial F_2(a_1,a_2)}{\partial a_2} = 0 \end{aligned} \tag{6.45}$$

eller

$$\begin{bmatrix} \dfrac{\partial F_1}{\partial a_1} & \dfrac{\partial F_1}{\partial a_2} \\ \dfrac{\partial F_2}{\partial a_1} & \dfrac{\partial F_2}{\partial a_2} \end{bmatrix} \begin{bmatrix} \delta a_1 \\ \\ \delta a_2 \end{bmatrix} = \begin{bmatrix} -F_1 \\ \\ -F_2 \end{bmatrix} \tag{6.46}$$

(hvor F_1 og F_2 betyder $F_1(a_1,a_2)$ *osv.*; *dvs.* med de nuværende, ukorrigerede parametre a_1 og a_2). Derivaterne i matricen er faktisk (se definition af F_1 og F_2) dobbelderivater af χ^2, så ligningen kan også skrives som

$$\begin{bmatrix} \dfrac{\partial^2\chi^2}{\partial a_1^2} & \dfrac{\partial^2\chi^2}{\partial a_1 \partial a_2} \\ \dfrac{\partial^2\chi^2}{\partial a_2 \partial a_1} & \dfrac{\partial^2\chi^2}{\partial a_2^2} \end{bmatrix} \begin{bmatrix} \delta a_1 \\ \\ \delta a_2 \end{bmatrix} = \begin{bmatrix} -\dfrac{\partial\chi^2}{\partial a_1} \\ -\dfrac{\partial\chi^2}{\partial a_2} \end{bmatrix} \ . \tag{6.47}$$

Så har vi at $\mathbf{J}\cdot\delta\mathbf{a} = -\mathbf{F}$ ($\mathbf{J}$ værende Jacobian).

Dobbelderivaterne (der er tre forskellige) er

$$\begin{aligned} \frac{\partial^2\chi^2}{\partial a_1^2} &= 2\sum_{i=1}^{N} \frac{\exp(2a_2x_i)}{\sigma_i^2} \\ \frac{\partial^2\chi^2}{\partial a_1 \partial a_2} &= -2\sum_{i=1}^{N} \frac{x_i y_i \exp(a_2x_i) - 2a_1x_i\exp(2a_2x_i)}{\sigma_i^2} \qquad (6.48) \\ \frac{\partial^2\chi^2}{\partial a_2^2} &= -2\sum_{i=1}^{N} \frac{a_1x_i^2 y_i \exp(a_2x_i) - 2a_1^2x_i^2\exp(2a_2x_i)}{\sigma_i^2} \ . \qquad (6.49) \end{aligned}$$

En sidste ting er, at vi som altid jo er interesserede i de beregnede parametres usikkerheder eller deres varianser. De er de samme som varianserne af korrektionerne $\delta\mathbf{a}$. Som før, for generel mindste kvadrater, er de indeholdt i den inverse matrix $\mathbf{J}^{-1}$, i diagonalen.

Den her beskrevne proces kan generaliseres til en tilpasning til en generel funktion $f(x,\mathbf{a})$, hvor vektoren $\mathbf{a}$ består af et antal M parametre. Vi løser

altid ligningen $\mathbf{J} \cdot \delta\mathbf{a} = -\mathbf{F}$, og matricen $\mathbf{J}$ er

$$\mathbf{J} = \begin{bmatrix} \dfrac{\partial^2\chi^2}{\partial a_1^2} & \dfrac{\partial^2\chi^2}{\partial a_1 \partial a_2} & \cdots & \dfrac{\partial^2\chi^2}{\partial a_1 \partial a_M} \\ \dfrac{\partial^2\chi^2}{\partial a_2 \partial a_1} & \dfrac{\partial^2\chi^2}{\partial a_2^2} & \cdots & \dfrac{\partial^2\chi^2}{\partial a_2 \partial a_M} \\ \vdots & & & \vdots \\ \dfrac{\partial^2\chi^2}{\partial a_M \partial a_1} & \dfrac{\partial^2\chi^2}{\partial a_M \partial a_2} & \cdots & \dfrac{\partial^2\chi^2}{\partial a_M^2} \end{bmatrix} \quad \text{og} \quad \mathbf{F} = \begin{bmatrix} \dfrac{\partial\chi^2}{\partial a_1} \\ \dfrac{\partial\chi^2}{\partial a_2} \\ \vdots \\ \dfrac{\partial\chi^2}{\partial a_M} \end{bmatrix} \tag{6.50}$$

og, som ovenstående, inverset af $\mathbf{J}$ har usikkerhederne af de beregnede parametre i diagonalen.

Kapitel 7

Lineær Algebra og Matricer

7.1 Elementære definitioner

Et vektorrum (reelt eller komplekst) $\mathbf{V}$ over et givent vektorfelt $\mathbf{F}$ består af et sæt af vektorer

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \quad , \qquad \mathbf{x^T} = \begin{bmatrix} x_1, x_2, \ldots, x_n \end{bmatrix} \ .$$

Følgende operationer er relevante: Sum:

$$\mathbf{x} + \mathbf{y} = \begin{bmatrix} x_1 + y_1 \\ x_2 + y_2 \\ \cdots \\ x_n + y_n \end{bmatrix} \ ,$$

det ydre (direkte) produkt:

$$\mathbf{x} \otimes \mathbf{y^T} = \begin{bmatrix} x_1\mathbf{y^T} \\ x_2\mathbf{y^T} \\ \cdots \\ x_n\mathbf{y^T} \end{bmatrix} \quad \text{dvs. matrix} \quad \begin{bmatrix} x_1y_1 & x_1y_2 & x_1y_3 & \ldots & x_1y_n \\ x_2y_1 & x_2y_2 & x_2y_3 & \ldots & x_2y_n \\ \cdots & & & & \\ x_ny_1 & x_ny_2 & x_ny_3 & \ldots & x_ny_n \end{bmatrix} \ .$$

En matrix $\mathbf{A}$ og dens transponerede $\mathbf{A^T}$ defineres som

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \ldots & a_{mn} \end{bmatrix} , \mathbf{A^T} = \begin{bmatrix} a_{11} & a_{21} & \cdots & a_{m1} \\ a_{12} & a_{22} & \cdots & a_{m2} \\ \vdots & \vdots & \ddots & \vdots \\ a_{1n} & a_{2n} & \ldots & a_{mn} \end{bmatrix} \ .$$

Det gælder, at matricen $\mathbf{A}$ er orthonormal såfremt

$$\mathbf{A^T A} = \mathbf{I}$$

hvor **I** er enhedsmatricen. Før vi går videre med operationer af matricer præsenteres nogle hyppige former for "sparse" matricer; *dvs.* matricer med mange nuller systematisk fordelt over matricen:

$$\begin{bmatrix} a_{11} & 0 & \cdots & 0 \\ 0 & a_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & a_{nn} \end{bmatrix} \quad \text{(diagonal)},$$

$$\begin{bmatrix} a_{11} & a_{12} & 0 & \cdots & 0 \\ a_{21} & a_{22} & a_{23} & \cdots & 0 \\ \ddots & \ddots & \ddots & & \vdots \\ & & \ddots & \ddots & \ddots \\ 0 & 0 & a_{m-1,n-2} & a_{m-1,n-1} & a_{m-1,n} \\ 0 & 0 & \cdots & a_{m,n-1} & a_{mn} \end{bmatrix} \quad \text{(tridiagonal)},$$

$$\begin{bmatrix} a_{11} & 0 & \cdots & 0 \\ a_{21} & a_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \ldots & a_{mn} \end{bmatrix} \quad \text{(nedre trekant)},$$

$$\begin{bmatrix} a_{11} & a_{12} & & \cdots & a_{1n} \\ 0 & a_{22} & a_{23} & \cdots & a_{2n} \\ \vdots & & \ddots & \ddots & \vdots \\ & \cdots & & a_{m-1,n-1} & a_{m-1,n} \\ 0 & \cdots & & 0 & a_{mn} \end{bmatrix} \quad \text{(øvre trekant)}.$$

7.2 Vektor- og matrixnormer

Normen af en vektor eller matrix skrives som $\|\mathbf{A}\|$. For normen af en vektor eller matrix gælder følgende:

1. normen skal være større end nul hvis vektorens eller matricens elementer ikke alle er lige nul eller
   $\|\mathbf{A}\| > 0$; $\|\mathbf{A}\| = 0$ **iff** $\mathbf{A}$=0 (hvor **iff** betyder "if and only if");

2. normen ganges med skalaren k's magnitude hvis alle elementer i vektoren eller matricen ganges med k eller
   $\|k\mathbf{A}\| = |k|\,\|\mathbf{A}\|$;

3. normen af summen af to vektorer eller matricer overgår ikke summen af deres respektive normer eller
   $\|\mathbf{A} + \mathbf{B}\| \leq \|\mathbf{A}\| + \|\mathbf{B}\|$;

4. normen af produktet af to vektorer eller matricer overgår ikke produktet af deres respektive normer eller
$\|\mathbf{AB}\| \leq \|\mathbf{A}\|\|\mathbf{B}\|$.

Der er en række normer for vektorer og matricer. For vektorer er den generelle form

$$\|\mathbf{x}\|_p = \left(\sum_{i=1}^{n} |x_i|^p \right)^{1/p} . \tag{7.1}$$

Den euklidiske norm $\|\mathbf{x}\|_2$ (eller 2-norm) er givet ved $p = 2$, og infinitynormen $\|\mathbf{x}\|_\infty$ er defineret for $p \to \infty$, som $\max_{1 \leq i \leq n} |x_i|$, hvor n er vektorens længde.

Normen for en matrix $\mathbf{A}$ kan ligeledes defineres på forskellige måder, på lignende måder som for vektorer. For eksempel for $p = 1$,

$$\|\mathbf{A}\|_1 = \max_{1 \leq j \leq n} \sum_{i=1}^{n} |a_{ij}| \tag{7.2}$$

som er den største søjlesum; mens infinitynormen er her

$$\|\mathbf{A}\|_\infty = \max_{1 \leq i \leq n} \sum_{j=1}^{n} |a_{ij}| \tag{7.3}$$

eller den største rækkesum.

7.3 Lineær transformation

En lineær transformation kan anskues som følger. Antag, at vi har en vektor $\mathbf{x}$, der ved hjælp af en operator $\mathbf{A}$ transformeres ind i en ny vektor $\mathbf{y}$ (indenfor det samme vektorrum). Såfremt $\mathbf{A}$ repræsenterer transformationsmatricen kan denne proces beskrives ved en matrix-vektor multiplikation af typen

$$\mathbf{y} = \mathbf{Ax} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix} , \quad y_i = \sum_{j=1}^{m} a_{ij} x_j . \tag{7.4}$$

7.4 Matrixmultiplikation.

Produktet $\mathbf{C}$ af $m \times n$ matricen $\mathbf{A}$ og $n \times p$ matricen $\mathbf{B}$ er givet ved

$$\mathbf{C} = \mathbf{AB} = \begin{bmatrix} C_{11} & C_{12} & \cdots & C_{1p} \\ C_{21} & C_{22} & \cdots & C_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ C_{m1} & C_{m2} & \ldots & C_{mp} \end{bmatrix} , \qquad C_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj} , \tag{7.5}$$

hvor dimensionen af $\mathbf{C}$ matricen er $m \times p$. Det bemærkes, at elementet C_{ij} er produktet af i'te række i $\mathbf{A}$ og j'te søjle i $\mathbf{B}$. Matrixmultiplikation er distributiv

$$\mathbf{A}\,(\mathbf{B}+\mathbf{C}) = \mathbf{AB}+\mathbf{AC} \qquad \text{og} \qquad (\mathbf{B}+\mathbf{C})\,\mathbf{D} = \mathbf{BD}+\mathbf{CD}$$

men ikke nødvendigvis kommutativ, *dvs.*

$$\mathbf{AB} \neq \mathbf{BA}$$

vil ofte være tilfældet.

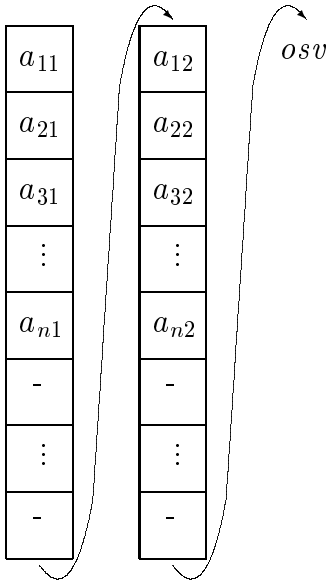
Da matrixmultiplikationen indgår som et centralt element i numerisk beregning er det nyttigt at forstå computerens behandling af denne proces. Lad for bekvemmeligheds skyld matricerne $\mathbf{A}$, $\mathbf{B}$ og $\mathbf{C}$ være $n \times n$ matricer. Matrixmultiplikationen defineret ovenfor kan i Fortran-90 udføres under anvendelse af operationen `MATMUL` (efter passende erklæring af matricerne). Man vil umiddelbart selv programmere computeren til en operation af typen

```
c = 0
do i = 1,n
  do j = 1,n
    do k = 1,n
      c(i,j) = c(i,j) + a(i,k)*b(k,j)
    enddo
  enddo
enddo
```

Bemærk at `c` er nulstillet før start.

Med et *flop* (floating point operation) defineret som en addition (subtraktion) eller en multiplikation (division) som vi har i den midterste linje, vil en matrixmultiplikation kræve $2n^3$ flop's. Det bemærkes, at antal flop per sekund hyppigt anvendes som et mål for computerens hastighed (ca. 4 Gflop for `uran2` svarende til $2.5 \cdot 10^{-10}$ s/flop) . Hukommelseskravet vil for normale reelle tal (32 bits eller 4 bytes af 8 bits hver) være $4n^2$ bytes (hvilket for en 1000 x 1000 matrix svarer til 4 Mb). Da computeren har en endelig hukommelse - og denne ofte er fordelt mellem RAM (Random Access Memory) med "pages" (*f.eks.* 512 kb) og virtuel hukommelse (nogle Gb på harddisken) - er det af stor betydning at ovenstående operation udnytter computerens lagring af matricer på optimal vis for at undgå for meget "paging" og disktransporter. Sidstnævnte operation er op til to størrelsesordner langsommere end flytning fra RAM hukommelse til CPU'en. For at få et indtryk af processeringstid ved matrixmultiplikationer kan nævnes, at maskinen `uran2` kan klare en 100 x 100 matrixmultiplikation i 1.6 ms, men tager 1.6 s for en 1000 x 1000 (skrevet i 2010).

Idet processeringstiden for store matricer, selv på kraftige computere, er anseelig er det umagen værd at optimere processer som matrixmultiplikation så meget som mulig - eksempelvis ved minimering af unødig datatransport. Antag at de tre matricer $\mathbf{A}$, $\mathbf{B}$ og $\mathbf{C}$ har den fysiske dimension $m \times m$ (dimensionen



Figur 7.1: Fortrans lagringsorden af matricer i hukommelsen

defineret ved erklæring af de variable), mens den logiske dimension er $n \times n$ (den aktuelt anvendte dimension af arrayene) med $n < m$. Computeren lagrer de tre matricer "søjlevist" efter hinanden *dvs.*, som illustreret i Fig. 7.1.

Udføres nu en matrixmultiplikation efter skemaet ovenfor vil beregning af eksempelvis C_{11} involvere multiplikation af elementnumrene 1 (i $\mathbf{A}$) og n^2+1 (i $\mathbf{B}$), $n+1$ og n^2+2 etc. og dermed et stort antal spring i hukommelsen. Denne ulempe kan reduceres væsentlig ved i stedet at anvende proceduren

```
c = 0
do j = 1,n
  do k = 1,n
    do i = 1,n
      c(i,j) = c(i,j) + a(i,k)*b(k,j)
    enddo
  enddo
enddo
```

hvor de tilsvarende operationer involverer elementerne 1 og n^2+1, 2 og n^2+2 *osv.* Man har således omformet den inderste løkkes operationer (som jo går hurtigst) fra række- til søjleoperation, hvilket passer bedre med den aktuelle lagring af matricer. Som en generel regel skal man altid tilstræbe mindst mulig grad af "springen rundt" i hukommelsen, hvilket for matrixoperationer opnås ved at tage hensyn til at første index løber hurtigst i Fortran. En god f90/95-compiler vil sørge for, at intrinsic function MATMUL udføres på denne måde. Det anføres, som et andet argument, at programmer med minimal "paging" i

almindelighed også vektoriserer og paralleliserer godt - hvilket er af betydning for anvendelse af vektorprocessorer og klynger af computere.

7.5 Determinanter

Den måde en matrices determinant er defineret på er den mest tidskrævende måde at evaluere den. Der er dog nogle egenskaber af determinanter, der gør arbejdet mindre tidskrævende. Der er en rekursiv måde at udtrykke bestemmelsen af en determinant på. Lad os sige, vi har en $n \times n$ matrix $\mathbf{A}$. Dens determinant $|\mathbf{A}|$ er så summen, hen over en række (eller søjle) af rækkens (eller søjlens) elementer ganget med kofaktorens determinant, med alternerende foretegn (sign= $(-1)^{(i+j)}$). Kofaktoren for det (i,j)te element er den mindre matrix dannet af $\mathbf{A}$ med rækken i og søjlen j udeladt. For eksempel, hvis

$$\mathbf{A} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 5 & 7 & 8 \end{bmatrix}$$

så er kofaktoren for det første rækkeelement a_{11} undermatricen $\begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix}$, *osv.* Determinanten af $\mathbf{A}$ er således

$$\begin{vmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 5 & 7 & 8 \end{vmatrix} = 1 \begin{vmatrix} 5 & 6 \\ 7 & 8 \end{vmatrix} - 2 \begin{vmatrix} 4 & 6 \\ 5 & 8 \end{vmatrix} + 3 \begin{vmatrix} 4 & 5 \\ 5 & 7 \end{vmatrix}$$

hvilket ekspanderer til

$$\begin{vmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 5 & 7 & 8 \end{vmatrix} = 1(5 \times 8 - 7 \times 6) - 2(4 \times 8 - 5 \times 6) + 3(4 \times 7 - 5 \times 5) = 3 \; . \quad (7.6)$$

Udtrykket er rekursiv fordi den benytter ordet "determinant" inden for definitionen af en determinant. Hvis matricen havde været større end bare 3×3, ville der have været flere trin, med flere og flere mindre determinanter at evaluere, ned til et antal 2×2, som her.

Der er en række egenskaber der kan benyttes til at gøre den process nemmere, der har relevans til senere afsnit:

1. hvis to rækker af en matrix byttes, ændres foretegnet, men ikke den absolutte værdi, af determinanten.

2. Hvis man lægger en række til en anden, ændres ikke determinantens værdi (eller foretegn).

3. Har man trianguleret (eller diagonaliseret) en matrix ved eliminering af det nedre og/eller de øvre trekant (som i Gauss-eliminering eller LUD, se senere afsnit), så er determinanten blot produktet af de diagonale elementer.

Som skal ses, indgår posterne (1) og (2) i metoderne til løsningen af lineære ligningsystemer, og er meget effektive. Når man benytter (3) så skal man passe på, hvor mange gange man har pivoteret (byttet rækker), ifølge post (1).

Der er mere effektive metoder til at beregne en determinant. De baserer sig på punkt (3) i det ovenstående, som også indebærer de andre to. Triangulering opnår man med de metoder, man løser ligningssystemer med, som beskrevet i afsnit 7.7, *dvs.* efter Gauss-eliminering eller løsning med LUD. Disse er mindre computertidskrævende end den metode beskrevet ovenpå.

7.6 Matrixinvertering

Matrixinvertering kan også være meget tidskrævende, hvis man ikke benytter effektive tricks. Den måde hvorpå invertering er tit beskrevet, er nok den mindst effektive. Vi vil invertere matricen $\mathbf{A}$. Først må vi beregne dens determinant $|\mathbf{A}|$ (det kan i sig selv være en tidskrævende proces). Så transponerer man matricen, og hvert element af den inverse matrix $\mathbf{V}$, V_{ij}, er givet ved kofaktoren af det tilsvarende element i $\mathbf{A}^T$ (dvs, determinanten af den mindre matrix dannet af $\mathbf{A}^T$ med den $i$te række og den $j$te søjle udeladt) divideret med determinanten $|\mathbf{A}|$, samt at der skal skiftes foretegn ved de lige steder, lige som når man beregner en determinant. Her er et simpelt eksempel. Vi har

$$\mathbf{A} \equiv \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 5 & 7 & 8 \end{bmatrix} .$$

Determinanten er lig med 3 som det sås i (7.6). Inversen dannes herefter fra den transponerede matrix:

$$\mathbf{A}^T = \begin{bmatrix} 1 & 4 & 5 \\ 2 & 5 & 7 \\ 3 & 6 & 8 \end{bmatrix}$$

og så har vi at elementerne i den inverse $\mathbf{V}$ er dannet af kofaktorerne hen af en række eller søjle:

$$V_{11} = \frac{1}{3}\begin{vmatrix} 5 & 7 \\ 6 & 8 \end{vmatrix} ,$$

$$V_{12} = -\frac{1}{3}\begin{vmatrix} 2 & 7 \\ 3 & 8 \end{vmatrix} ,$$

$$V_{13} = \frac{1}{3}\begin{vmatrix} 2 & 5 \\ 3 & 6 \end{vmatrix} ,$$

osv., og resultatet er til sidst

$$\mathbf{V} = -\frac{1}{3}\begin{bmatrix} -2 & 5 & -3 \\ -2 & -7 & 6 \\ 3 & 3 & -3 \end{bmatrix} .$$

Denne metode er som sagt meget tidskrævende (også for en computer), men egner sig fint for matricer op til 3×3. For større matricer anvender man mere effektive metoder, se nedenfor under afsnit 7.7.

7.7 Løsning af lineære ligningssystemer

Inden for kemi og fysik støder man ofte på lineære ligningssystemer af typen

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \cdots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + \cdots + a_{2n}x_n &= b_2 \\ a_{31}x_1 + a_{32}x_2 + a_{33}x_3 + \cdots + a_{3n}x_n &= b_3 \\ \cdots & \qquad (7.7) \\ a_{m1}x_1 + a_{m2}x_2 + a_{m3}x_3 + \cdots + a_{mn}x_n &= b_n \end{aligned}$$

(7.8)

med n ubekendte $x_1, x_2, \cdots, x_n$ relateret via m ligninger. I tilfældet $n = m$ er der lige så mange ligninger som ubekendte og systemet kan løses, såfremt alle ligninger er lineært uafhængige (det ikke-singulære tilfælde). Ligningssystemet er singulært, når to eller flere ligninger er lineært afhængige (ofte betegnet rækkedegeneration).

Ovenstående ligningssystem kan bekvemt skrives på matrixformen

$$\mathbf{A} \cdot \mathbf{x} = \mathbf{b} \,,$$

hvor $\mathbf{A}$ er den såkaldte koefficientmatrix, mens $\mathbf{b}$ og $\mathbf{x}$ er søjlevektorer med sidstnævnte indeholdende de ubekendte

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \ldots & a_{mn} \end{bmatrix} \,, \qquad \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \cdots \\ x_n \end{bmatrix} \,, \qquad \mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ \cdots \\ b_n \end{bmatrix} \,.$$

Når man udfører disse operationer, betyder ordet "række" ikke blot matricens række, men også det tilsvarende element i vektoren $\mathbf{b}$ på højre side af systemet (7.7). Som det ses senere, benytter man en *udvidet matrix*, som indeholder både matricens koefficienter, samt en ekstra søjle med den højre side i, og så behandler man alt på en gang.

Der findes en række mere eller mindre effektive metoder til løsning af lineære ligningssystemer. I dette og efterfølgende afsnit beskrives nogle af de simpleste løsningsmetoder. Indledningsvis beskrives den såkaldte *Gauss eliminationsmetode*, der ikke kan betegnes som den mest robuste eller hurtigste metode, men fordi den konceptionelt er simpel at forstå og er nyttig i praksis, eksempelvis til matrix invertering.

Trekantsmatricer

Mange metoder for løsningen af lineære systemer går ud på at konvertere en given matrix til en øvre og/eller nedre trekantmatrix.

En øvre trekantsmatrix har elementer forskellige fra nul på diagonalen og over diagonalen. Alt under diagonalen er lig nul. Hvis det ligningssystem, der skal løses har sådan en matrix, er det let at løse:

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \cdots + a_{1n}x_n &= b_1 \\ a_{22}x_2 + a_{23}x_3 + \cdots + a_{2n}x_n &= b_2 \\ a_{33}x_3 + \cdots + a_{3n}x_n &= b_3 \\ \cdots &= \cdots \\ a_{nn}x_n &= b_n \end{aligned} \tag{7.9}$$

Såfremt alle $a_{ii} \neq 0$ kan x_n bestemmes fra sidste ligning, x_{n-1} ved indsættelse af denne løsning i ligningen ovenfor og så fremdeles. Denne metode, der benævnes *back substitution*, kan rekursivt udtrykkes som

$$x_i = \frac{1}{a_{ii}} \left(b_i - \sum_{j=i+1}^{n} a_{ij}x_j \right) . \tag{7.10}$$

En nedre trekantsmatrix har elementer forskellige fra nul på diagonalen og under diagonalen. Alt over diagonalen er lig nul. Et ligningessystem med sådan en matrix, er ligeledes let at løse, med *forward substitution*:

$$\begin{aligned} a_{11}x_1 &= b_1 \\ a_{21}x_1 + a_{22}x_2 &= b_2 \\ a_{31}x_1 + a_{32}x_2 + a_{33}x_3 &= b_3 \\ \cdots & \\ a_{n1}x_1 + a_{n2}x_2 + a_{n3}x_3 + \cdots + a_{nn}x_n &= b_n \end{aligned} \tag{7.11}$$

Her starter vi med første ligning, som leverer x_1 direkte, og så fremdeles. Den generelle rekursive formel her er

$$x_i = \frac{1}{a_{ii}} \left(b_i - \sum_{j=1}^{i-1} a_{ij}x_j \right) . \tag{7.12}$$

Gauss og Gauss-Jordan elimination

Ovenstående ligningssystem (7.7) potentielt kan løses i tilfældet, hvor $m = n$ og koefficientmatricen er en øvre (eller nedre) trekantsmatrix, eller kan bringes på *fx.* øvre trekantsform. Dette gøres ved såkaldt *Gauss elimination*. Denne metode er baseret på at benytte sig af en systematisk eliminering af elementer, for at nå frem til en øvre trekantsmatrix.

Der er nogle nyttige regler for hvad man må med matricer, der gør løsninger mulige. Man må

- bytte rækker
- gange en række med en konstant.
- lægge rækker sammen, dvs, erstatte en række med en (vægtet) sum af selve rækken og en anden.

Det sidste kan eliminere elementer vi gerne vil nulstille. For eksempel, i systemet (7.7) vil vi gerne have alt det der ligger direkte under det øverst til venstre element stillet til nul. Generelt er operationen den, når vi vil eliminere $a_{i,1}$, at vi erstatter linje i med summen af linje 1 ganget med a_{i1} og linje i ganget med $-a_{11}$, osv. Et eksempel vil illustrere dette. Lad os sige, vi har med dette system at gøre:

$$\begin{bmatrix} 1 & 2 & 3 \\ 2 & -2 & -1 \\ 3 & -1 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 11 \\ 2 \\ 12 \end{bmatrix} . \tag{7.13}$$

Det er en fordel, hvis systemet er, så vidt muligt, diagonal dominant, og det er ikke tilfældet her. Derfor bytter vi om på række 1 og 3. Dette kaldes **pivotering**. Inden vi udfører det, må vi beskrive en anden mekanisme. Når vi bytter rækker eller lægger rækker sammen, må vi også gør det samme med den højre side, vektoren $\mathbf{b}$. For at gøre det nemmere, udvider vi koefficientmatricen til en *udvidet matrix* (eng.: *augmented matrix*), som har denne vektor lagt til som en ektra søjle. Matricen bliver så repræsenteret i sin udvidet (og pivoteret) form som

$$\left[\begin{array}{ccc|c} 3 & -1 & 2 & 12 \\ 2 & -2 & -1 & 2 \\ 1 & 2 & 3 & 11 \end{array}\right] . \tag{7.14}$$

Nu kan vi begynde at eliminere elementer. Lad R_i stå for række i. Vi begynder med at erstatte R_2 med summen $-2 \times R_1 + 3 \times R_2$, og R_3 med $-R_1 + 3 \times R_3$, hvilket giver

$$\left[\begin{array}{ccc|c} 3 & -1 & 2 & 12 \\ 0 & -4 & -7 & -18 \\ 0 & 7 & 7 & 21 \end{array}\right] . \tag{7.15}$$

Vi mangler at eliminere et element mere, ved at erstatte R_3 med $-7 \times R_2 + (-4) \times R_3$, og vi ender med

$$\left[\begin{array}{ccc|c} 3 & -1 & 2 & 12 \\ 0 & -4 & -7 & -18 \\ 0 & 0 & 21 & 42 \end{array}\right] . \tag{7.16}$$

Nu har vi et øvre trekantssystem, og kan tilbagesubstituere, fra bunden. Vi kan direkte finde at $x_3 = 2$, og gå op ad systemet. Vi får løsningen til systemet (7.14) (7.13), $(x_1, x_2, x_3) = (3, 1, 2)$.

Vi kan også, efter at have beregnet (7.16), fortsætte og eliminere elementerne over diagonalen, i stedet for at tilbagesubstituere. Det forgår på analog vis, og vi ender med

$$\left[\begin{array}{ccc|c} 1764 & 0 & 0 & 5292 \\ 0 & -84 & 0 & -84 \\ 0 & 0 & 21 & 42 \end{array}\right] . \tag{7.17}$$

Dette er metoden Gauss-Jordan.

Læg mærke til, at koefficienterne bliver større, men nu kan vi bare aflæse løsningen. Der er en alternativ procedure, der forhindrer, at koefficenterne bliver større og større. I stedet for at gange med matricens koefficienter, dividere vi med dem. Det første elimineringstrin bliver således at erstatte R_2 med $-\frac{1}{3}R_1+\frac{1}{2}R_2$, hvilket også eliminerer et element, og analog med de andre. Dette ender med

$$\left[\begin{array}{ccc|c} 3 & -1 & 2 & 12 \\ 0 & \frac{2}{3} & \frac{7}{6} & 3 \\ 0 & 0 & \frac{7}{12} & \frac{7}{6} \end{array}\right] \tag{7.18}$$

hvilket naturligvis giver samme resultat. Hvis vi går videre med Gauss-Jordan bliver det diagonaliserede system til

$$\left[\begin{array}{ccc|c} \frac{7}{5} & 0 & 0 & \frac{21}{5} \\ 0 & -\frac{4}{7} & 0 & -\frac{4}{7} \\ 0 & 0 & \frac{7}{12} & \frac{7}{6} \end{array}\right] . \tag{7.19}$$

Vi kan skalere hele systemet ved at dividere med diagonalelementerne, hvilket er en måde at præsentere løsningen på. Det giver

$$\left[\begin{array}{ccc|c} 1 & 0 & 0 & 3 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 2 \end{array}\right] \tag{7.20}$$

hvor løsningen står nu direkte på højre side. Dette egner sig til at invertere matricer, se nedenfor.

LU dekomponering (LUD). Crout's metode

En effektiv og fleksibel metode for at løse lineære systemer er LUD eller LU-dekomponering. Vi forudsætter, at matricen $\mathbf{A}$ i systemet

$$\mathbf{Ax} = \mathbf{b} \tag{7.21}$$

kan omregnes til et produkt af en nedre og en øvre trekantsmatrix, $\mathbf{LU} = \mathbf{A}$. Så har vi, efter multiplikation af (7.21) med $\mathbf{L}^{-1}$

$$\mathbf{Ux} = \mathbf{L}^{-1}\mathbf{b} = \mathbf{y} , \tag{7.22}$$

$$\mathbf{Ly} = \mathbf{b}\ . \tag{7.23}$$

Fra sidste ligning bestemmes $\mathbf{y}$ ved "forward substitution" som er en let process da matricen $\mathbf{L}$ jo er en nedre trekant. Herefter anvendes $\mathbf{y}$ i (7.22) til bestemmelse af $\mathbf{x}$ ved "back-substitution", da $\mathbf{U}$ er ligeledes en trekantsmatrix.

Hovedproblemet består nu i at finde $\mathbf{L}$ og $\mathbf{U}$. Dem finder man ud fra at deres produkt $\mathbf{LU}$ skal være lig med $\mathbf{A}$. Til at gøre det muligt, skal der være en betingelse; der skal være et-taller på diagonalerne i enten $\mathbf{L}$ eller $\mathbf{U}$. Har vi det, kan vi gradvis, med rekursive udtryk, beregne alle andre elementer i de to matricer. Vi benytter igen ovenstående eksempel (7.13), efter rækkebytning. Når man LU-dekomponerer, kommer den hørje side $\mathbf{b}$ ikke i billedet, indtil man benytter $\mathbf{L}$ og $\mathbf{U}$. Matricen der skal LUD'es er altså produktet

$$\begin{bmatrix} 1 & 0 & 0 \\ \ell_{21} & 1 & 0 \\ \ell_{31} & \ell_{32} & 1 \end{bmatrix} \times \begin{bmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{bmatrix} = \begin{bmatrix} 3 & -1 & 2 \\ 2 & -2 & -1 \\ 1 & 2 & 3 \end{bmatrix}\ . \tag{7.24}$$

Processen er nu nem at forstå. Vi kan uden videre sætte den øverste række i $\mathbf{U}$, som er $(3, -1, 2)$ ud fra at $1 \times u_{11} = 3$ *osv.* Nu at vi kender de tre u-værdier kan vi sætte den første søjle i $\mathbf{L}$: ved at gange række i i $\mathbf{L}$ med første søjle i $\mathbf{U}$, for $i = 2, 3$. Derefter benytter vi disse nu kendte værdier. For eksempel får vi u_{22} ved at bruge $\ell_{21}u_{12} + u_{22} = -2$, og så fremdeles. Det endelige resultat bliver

$$\mathbf{L} = \begin{bmatrix} 1 & 0 & 0 \\ \frac{2}{3} & 1 & 0 \\ \frac{1}{3} & -\frac{7}{4} & 1 \end{bmatrix} \tag{7.25}$$

og

$$\mathbf{U} = \begin{bmatrix} 3 & -1 & 2 \\ 0 & -\frac{4}{3} & -\frac{7}{3} \\ 0 & 0 & -\frac{7}{4} \end{bmatrix}\ . \tag{7.26}$$

I praksis, eftersom vi ved at diagonalen af $\mathbf{L}$ er 1'er, kan vi slå de to sammen i en enkelt matrix:

$$\begin{bmatrix} 3 & -1 & 2 \\ \frac{2}{3} & -\frac{4}{3} & -\frac{7}{3} \\ \frac{1}{3} & -\frac{7}{4} & -\frac{7}{4} \end{bmatrix} \tag{7.27}$$

med diagonalen underforstået. Det er således, computerrutiner gemmer begge matricer væk i én.

Ovenstående eksempel illustrerer proceduren. Den formelle beskrivelse er følgende. Man udfører, skiftevis, for alle $j = 1, \ldots, n$, de to operationer

$$u_{ij} = a_{ij} - \sum_{k=1}^{i-1} \ell_{ik}u_{kj} \qquad i = 1, 2, \ldots, j \tag{7.28}$$

og

$$l_{ij} = \frac{1}{u_{jj}} \left(a_{ij} - \sum_{k=1}^{j-1} \ell_{ik} u_{kj} \right) \qquad i = j+1, j+2, \ldots, n \tag{7.29}$$

hvor a_{ij} er det tilsvarende element i matricen $\mathbf{A}$.

Nu kommer den store fordel ved LUD. Den bliver aktuel nu. Lad os sige, i vores eksempel, at den højre side $\mathbf{b}$ er (passende til det pivoterede system (7.14)), vektoren $[12, 2, 11]^T$. Notér ligning (7.23). Da vi jo kender $\mathbf{L}$, kan vi beregne vektoren $\mathbf{y}$. Den bliver til $[12, -6, -3.5]^T$. Har vi den, kan vi gå videre til løsningen af den eftersøgte vektor $\mathbf{x}$, givet ud fra ligning (7.22), igen en let proces. Begge disse operationer er meget lette, da de bare involverer brug af trekantsmatricer. Hvis vi nu får en ny $\mathbf{b}$, kan vi hurtigt udregne den tilsvarende nye $\mathbf{x}$, uden igen at skulle udføre en LUD. Dette forekommer ret tit i fysik-kemiske beregninger.

Matrixinvertering

I afsnit 7.6 blev en systematisk metode for matrixinvertering beskrevet og det blev noteret, at denne er meget ueffektiv for matricer større end 3×3. For større matricer anvender man mere effektive metoder.

Bl.a. LUD kan bekvemt anvendes til bestemmelse af den inverse $\mathbf{A}^{-1}$ til en matrix $\mathbf{A}$. Inversion af $n \times n$ matricen $\mathbf{A}$ kan foretages ved løsning af de n ligninger $\mathbf{A}x_i = e_i$ indeholdt i

$$\mathbf{A}\mathbf{A}^{-1} = \mathbf{I} \qquad \rightarrow \qquad \mathbf{A} \begin{bmatrix} x_{11} & x_{12} & \ldots & x_{1n} \\ x_{21} & x_{22} & \ldots & x_{2n} \\ \cdot & \cdot & \ldots & \cdot \\ x_{n1} & x_{n2} & \ldots & x_{nn} \end{bmatrix} = \begin{bmatrix} 1 & 0 & \ldots & 0 \\ 0 & 1 & \ldots & 0 \\ \cdot & \cdot & \ldots & \cdot \\ 0 & 0 & \cdot & 1 \end{bmatrix} \tag{7.30}$$

hvor $\mathbf{A}$ bliver løst for et antal højre sider, således at den udvidede matrix er

$$\left[\begin{array}{cccc|cccc} a_{11} & a_{12} & \ldots & a_{1n} & 1 & 0 & \ldots & 0 \\ a_{21} & \ldots & & & 0 & 1 & 0 & \ldots \\ \vdots & & & & \vdots & & & \\ a_{n1} & \ldots & & a_{nn} & 0 & \ldots & & 1 \end{array}\right] . \tag{7.31}$$

Når $\mathbf{A}$ er reduceret til en diagonal (dvs, den er løst for alle højre sider, som hver indeholder et enkelt 1-tal), står $\mathbf{A}^{-1}$ i den udvidede del. Det kan lade sig gøre med stor effektivitet med LUD. Her LU-dekomponerer man først matricen $\mathbf{A}$, og løser den derefter for n højre sider som alle består af en vektor af nuller, med et enkelt 1-tal i det i'te sted, hvilket giver den i'te søjle i inverset, for i løbende fra 1 til n.

Tridiagonale systemer

Der opstår ikke så sjældent ligningssystemer, hvis koeffcientmatricer er tridiagonale, dvs, hver linje har kun tre elementer liggende omkring diagonalen (undtagen den første og sidste linje, som kun indeholder to). Det ville være spild af computertid at løse sådanne systemer med de normale metoder beskrevet ovenpå, da der er en del nuller man ellers kan se bort fra. Der er mere effektive metoder til at løse disse. Lad os sige, vi har systemet

$$\begin{bmatrix} a_{11} & a_{12} & 0 & 0 & \dots & \\ a_{21} & a_{22} & a_{23} & 0 & \dots & \\ 0 & a_{32} & a_{33} & a_{34} & 0 & \dots \\ \vdots & \ddots & \ddots & \ddots & & \\ 0 & \dots & & a_{n-1,n-2} & a_{n-1,n-1} & a_{n-1,n} \\ 0 & & & \dots & a_{n,n-1} & a_{n,n} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_{n-1} \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_{n-1} \\ b_n \end{bmatrix} . \quad (7.32)$$

Det er ofte sådan, at vi kender enten a_{11} eller a_{nn}. Hvis det sidste er tilfældet, går vi baglæns op ad matricen og reducerer den til en didiagonal. Gående baglæns fra bunden og ved at substituere i hver linje i for x_{i+1}, producerer vi et nyt sæt, hvor hver linje nu kun har to elementer, indtil vi når op til den første, som reduceres til et enkelt element. Det giver direkte løsningen for x_1, og så går vi frem i nedad retning og løser for alle. Læs nærmere om det i Press & al. [4].

Iterativ løsning (Jacobi, Gauss-Seidel)

Iterationsmetoden, allerede beskrevet i afsnit 2.7, side 12, for enkelte ligninger, kan også bruges til løsning af ligningssystemer, enten lineære eller ikke-lineære. Den metode, der umiddelbart resulterer fra "g-metoden" i afsnit 2.7 ved at udvide den, kaldes Jacobi-metoden (ikke at forveksle med Jacobis metode til at beregne egenværdier). Metoden illustreres bedst med et eksempel. Lad os sige, vi har et sæt af tre ligninger at finde løsninger for:

$$\begin{aligned} 8x_1 + x_2 - x_3 &= 8 \\ 2x_1 + x_2 + 9x_3 &= 12 \\ x_1 - 7x_2 + 2x_3 &= -4 \end{aligned} \quad (7.33)$$

(løsningen er, at alle tre x_i er lige med 1, men vi lader som om vi ikke kender den). Man starter med et gætsæt. Proceduren er nu, at man bytter ligningerne således, at diagonalen er så vidt muligt dominerende. Det kan vi gøre ved at bytte ligning 3 med ligning 2. Hvis vi udtrykker det resulterende system generelt som det lineære system

$$\mathbf{Ax} = \mathbf{b} \quad (7.34)$$

så er den iterative proces den, at vi anvender ligningen

$$x_i^{(n+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j=1, j\neq i}^{n} a_{ij} x_j^{(n)} \right) \quad (7.35)$$

for successive i. I ovenstående eksempel er systemet, efter bytning,

$$\begin{aligned} x_1 &= \frac{1}{8}\big(8 - x_2 + x_3\big) \\ x_2 &= -\frac{1}{7}\big(-4 - x_1 - 2x_3\big) \\ x_3 &= \frac{1}{9}\big(12 - 2x_1 - x_2\big) \end{aligned} \tag{7.36}$$

og vi starter med et gætsæt, for eksempel, $\Big[x_1^{(0)}, x_2^{(0)}, x_3^{(0)}\Big]^T = [0,0,0]^T$. Det giver regningerne

$$x_1^{(1)} = \tfrac{8}{8} - \big(\tfrac{1}{8}\cdot 0 - \tfrac{1}{8}\cdot 0\big) = 1.000$$

$$x_2^{(1)} = \tfrac{-4}{-7} - \big(\tfrac{1}{-7}\cdot 0 + \tfrac{2}{-7}\cdot 0\big) = 0.571 \tag{7.37}$$

$$x_3^{(1)} = \tfrac{12}{9} - \big(\tfrac{2}{9}\cdot 0 + \tfrac{1}{9}\cdot 0\big) = 1.333\ . \tag{7.38}$$

Den proces gentager vi nu. Her er en tabel over de værdier man får:

trin:	0	1	2	3	...	7
x_1	0	1.000	1.095	0.995		1.001
x_2	0	0.571	1.095	1.026	...	1.001
x_3	0	1.333	1.048	0.969		1.001

(vi har udeladt nogle trin). Dette er Jacobi-metoden. Det ses, at værdierne konvergerer på løsningen, men ret langsomt. Det er muligt at fremskynde konvergensen en smule, ved at benytte, indenfor hvert trin, de allerede nye beregnede; i det ovenstående benyttes der startgættet hele vejen igennem den første iteration, *dvs.*, i anden beregning bruges der $x_1 = 0$, selvom den er lige beregnet til at være lig med 1. Bruger man disse nye værdier omgående, hedder metoden Gauss-Seidel og for denne metode, ser de første tre ud som følgende:

$$x_1^{(1)} = \tfrac{8}{8} - \big(\tfrac{1}{8}\cdot 0 - \tfrac{1}{8}\cdot 0\big) = 1.000$$

$$x_2^{(1)} = \tfrac{-4}{-7} - \big(\tfrac{1}{-7}\cdot 1 + \tfrac{2}{-7}\cdot 0\big) = 0.714 \tag{7.39}$$

$$x_3^{(1)} = \tfrac{12}{9} - \big(\tfrac{2}{9}\cdot 1 + \tfrac{1}{9}\cdot 0.714\big) = 1.032\ . \tag{7.40}$$

Tabellen bliver nu til

trin:	0	1	2	3	4
x_1	0	1.000	1.041	0.997	1.001
x_2	0	0.741	1.014	0.996	1.000
x_3	0	1.032	0.990	1.002	1.000

og det er tydeligt at løsningen konvergerer hurtigere.

Denne metode virker dog ikke altid. For eksempel kan den ikke anvendes til løsningen af systemet (7.13); den beregnede løsningsvektor divergerer mere og mere.

7.8 Egenværdier og egenvektorer

En matrices egenværdier og tilhørende egenvektorer er defineret ved egenværdiligningen

$$\mathbf{A}\mathbf{x} = \lambda\mathbf{x} \,, \tag{7.41}$$

hvor λ er en egenværdi og $\mathbf{x}$ den tilhørende egenvektor. λ er en egenværdi, hvis og kun hvis

$$|\mathbf{A} - \lambda\mathbf{I}| = 0 \,, \tag{7.42}$$

hvor $|\ldots|$ betegner determinanten. Det fremgår herved, at bestemmelsen af $\mathbf{A}$'s egenværdier svarer til bestemmelse af rødderne til det karakteristiske polynomium

$$\lambda^n + a_{n-1}\lambda^{n-1} + \cdots + a_1\lambda + a_0 = 0 \,, \tag{7.43}$$

hvor koefficienterne afhænger af $n \times n$ matricen $\mathbf{A}$.

Man kunne altså "bare" finde rødderne af (7.43). Problemet er blot, at det er meget vanskeligt og tidskrævende, at finde et polynomiums rødder. Faktisk er det sådan, at man gør det omvendt: eftersom der er mere effektive metoder til at finde egenværdier på, benytter man sig af dem til at finde rødder af polynomier, ved nogle kunstgreb, se kapitel 2, afsnit 2.8.

Der er ikke nogle direkte metoder for at finde egenværdier eller egenvektorer; de er alle i højre eller mindre grad iterative. Her nævnes der kun nogle af disse, uden at gå i detaljer. Press & al. [4] beskriver således Jacobitransformationen, som er anvendelig for reelle symmetriske matricer (og har reelle egenværdier), og **QR**-faktorisering for alle andre matricer. Press & al. [4] anbefaler også at man bruger færdige rutiner, og vi følger deres råd her, undtagen de iterative metoder der beskrives i næste afsnit.

"Power"-metode til bestemmelse af højeste egenværdi

Man starter med et gæt på egenvektoren; det kunne passende være vektoren $[1, 1, \ldots, 1]^T$. Gang matricen med den, hvilket giver en produktvektor. Prøveværdien af egenværdien er så det element i produktvektoren, der har den største magnitude. Divider produktvektoren med dette tal, som giver den forbedrede egenvektor. Gentag processen, indtil egenværdien ikke længere ændrer sig (betydeligt). Her er et eksempel. Vi vil have den højeste egenværdi for matricen $\begin{bmatrix} 4 & 1 & 0 \\ 0 & 2 & 1 \\ 0 & 0 & 1 \end{bmatrix}$. Første prøve er altså

$$\begin{bmatrix} 4 & 1 & 0 \\ 0 & 2 & 1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 5 \\ 3 \\ 1 \end{bmatrix} = 5 \begin{bmatrix} 1 \\ 0.6 \\ 0.2 \end{bmatrix} \tag{7.44}$$

hvor vores første estimat på egenværdien er 5. Vi gentager:

$$\begin{bmatrix} 4 & 1 & 0 \\ 0 & 2 & 1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0.6 \\ 0.2 \end{bmatrix} = \begin{bmatrix} 4.6 \\ 1.0 \\ 0.2 \end{bmatrix} = 4.6 \times \begin{bmatrix} 1 \\ 0.217 \\ 0.0435 \end{bmatrix} . \tag{7.45}$$

Efter et antal af disse iterationer får vi en endelig egenværdi lig med 4.0 og en egenvektor lig med $[1, 0, 0]^T$.

Hvis vi heller vil have den mindste egenværdi, kan vi få den ved først at invertere matricen, og så iterere på samme vis (og tilbageinvertere den nu største).

Iterativ metode til bestemmelse af laveste egenværdi

Idet matrixdiagonalisering normalt er forbundet med et operationstal (antal flops) af størrelsesordenen n^3, vil direkte bestemmelse af alle egenværdier og egenvektorer ofte være særdeles tids- og hukommelseskrævende for matricer af dimension $n > 500$. I specielle tilfælde, hvor matricen $\mathbf{A}$ er "sparse" (*dvs.* tyndt besat, indeholder mange nuller), og man kun er interesseret i en eller få af de laveste egenværdier, kan man med fordel anvende iterative metoder.

Antag, at vi har en tyndt besat symmetrisk matrix $\mathbf{A}$ af dimension n for hvilken vi simpelt kan udføre den lineære transformation jvf. (7.4), side 61.

$$\mathbf{y} = \overbrace{\mathbf{A}\mathbf{x}} \tag{7.46}$$

hvor den vandrette "Tuborg" her og i det følgende for overskuelighedens skyld indikerer en lineær transformation. Bestemmelse af $\mathbf{A}$'s i'te egenværdi, bestemt via

$$\mathbf{A}\mathbf{x}_i = \lambda_i \mathbf{x}_i \tag{7.47}$$

på baggrund af et begyndelsesgæt $\mathbf{x}^{(0)}$ på egenvektoren kan foretages som følger: (7.47), med $\mathbf{x}^{(0)}$ sat ind på $\mathbf{x}_i$'s plads, multipliceres med $\left(\mathbf{x}^{(0)}\right)^T$

$$\left(\mathbf{x}^{(0)}\right)^T \overbrace{\mathbf{A}\mathbf{x}^{(0)}} = \lambda^{(0)} \left(\mathbf{x}^{(0)}\right)^T \mathbf{x}^{(0)} \tag{7.48}$$

og, da $\left(\mathbf{x}^{(0)}\right)^T \mathbf{x}^{(0)}$ er en skalar,

$$\lambda^{(0)} = \frac{\left(\mathbf{x}^{(0)}\right)^T \overbrace{\mathbf{A}\mathbf{x}^{(0)}}}{\left(\mathbf{x}^{(0)}\right)^T \mathbf{x}^{(0)}} . \tag{7.49}$$

Dernæst antager vi, at $\mathbf{A}$ kan omskrives til

$$\mathbf{A} = \mathbf{A}^{(0)} + \mathbf{A}^{(1)} , \tag{7.50}$$

hvor $\mathbf{A}^{(0)}$ er en dominerende, let inverterbar (normalt diagonal) del og $\mathbf{A}^{(1)}$ en korrektionsdel. Tilsvarende anvendes

$$\lambda = \lambda^{(0)} + \lambda^{(1)}\,; \qquad \mathbf{x} = \mathbf{x}^{(0)} + \mathbf{x}^{(1)}\,, \tag{7.51}$$

hvor venstresiderne stadig repræsenterer eksakte værdier. Ved indsættelse af disse omskrivninger i (7.47) opnås

$$\overbrace{\left[\mathbf{A}^{(0)} + \mathbf{A}^{(1)}\right]\left[\mathbf{x}^{(0)} + \mathbf{x}^{(1)}\right]} = \left[\lambda^{(0)} + \lambda^{(1)}\right]\left[\mathbf{x}^{(0)} + \mathbf{x}^{(1)}\right]\,, \tag{7.52}$$

der ved negligering af led kvadratiske i korrektionsleddene let omskrives til

$$\mathbf{x}^{(1)} = -\left[\mathbf{A}^{(0)} - \lambda^{(0)}\mathbf{I}\right]^{-1}\overbrace{\left[\mathbf{A} - \lambda^{(0)}\mathbf{I} - \lambda^{(1)}\mathbf{I}\right]\mathbf{x}^{(0)}}\,. \tag{7.53}$$

Det bemærkes, at indholdet i $[\ldots]$ i ovenstående ligning svarer til matricen $\mathbf{A}$ med λ'erne fratrukket diagonalen (*dvs.* svarende til, at disse blev multipliceret med enhedsmatricen $\mathbf{I}$). Kræver vi nu orthogonaliteten

$$\left(\mathbf{x}^{(0)}\right)^T\mathbf{x}^{(1)} = 0 \tag{7.54}$$

kan $\lambda^{(1)}$ opnås fra (7.52) ved multiplikation med $\left(\mathbf{x}^{(0)}\right)^T$ fra venstre på begge sider

$$\begin{aligned} 0 = &-\left(\mathbf{x}^{(0)}\right)^T\left[\mathbf{A}^{(0)} - \lambda^{(0)}\mathbf{I}\right]^{-1}\overbrace{\left[\mathbf{A} - \lambda^{(0)}\mathbf{I}\right]\mathbf{x}^{(0)}} \\ &+ \left(\mathbf{x}^{(0)}\right)^T\left[\mathbf{A}^{(0)} - \lambda^{(0)}\mathbf{I}\right]^{-1}\lambda^{(1)}\mathbf{x}^{(0)} \end{aligned}$$

som giver

$$\begin{aligned} \lambda^{(1)} = &\left\{\left(\mathbf{x}^{(0)}\right)^T\left[\mathbf{A}^{(0)} - \lambda^{(0)}\mathbf{I}\right]^{-1}\mathbf{x}^{(0)}\right\}^{-1} \\ &\cdot\left(\mathbf{x}^{(0)}\right)^T\left[\mathbf{A}^{(0)} - \lambda^{(0)}\mathbf{I}\right]^{-1}\overbrace{\left[\mathbf{A} - \lambda^{(0)}\mathbf{I}\right]\mathbf{x}^{(0)}}\,. \end{aligned} \tag{7.55}$$

Som ses i eksemplet nedenfor, kan dette udtryk sommetider forenkles, hvis matricen $\mathbf{A}$ ikke blot er diagonaldominant, men hvor alle diagonalelementer har samme værdi.

Fra ovenstående udledning ses let, at korrektionsleddene for såvel egenværdi $\lambda^{(1)}$ og egenvektor $\mathbf{x}^{(1)}$ let bestemmes via ligningerne (7.55) og (7.53) under anvendelse af den lineære transformation (indikeret med vandret Tuborg og det indre (skalare) vektorprodukt (7.46)). Metoden er iterativ, idet man ved bestemmelse af korrektionsleddet $\mathbf{x}^{(1)}$ til det oprindelige bud på egenvektoren $\mathbf{x}^{(0)}$ opnår et forbedret bud

$$\mathbf{x}_{forbedret} = \mathbf{x}^{(0)} + \mathbf{x}^{(1)}\,, \tag{7.56}$$

der benyttes som nyt $\mathbf{x}^{(0)}$ til bestemmelse af en ny værdi for $\lambda^{(0)}$ ((7.49)) og derefter gentagelse af hele proceduren ovenfor. Såfremt residualet

$$R = \|\left[\mathbf{A} - \lambda^{(0)}\mathbf{I}\right]\mathbf{x}^{(0)}\| < \delta \tag{7.57}$$

er under en given (lille) værdi δ afgøres, at $\lambda^{(0)}$, $\mathbf{x}^{(0)}$ danner det endelige egenværdi-, egenvektorpar (*dvs.* konvergens er opnået). Værdien af δ må vælges passende.

Bemærkning om den lineære transformation

Her har vi tit at gøre med meget store matricer, og det kunne virke som om man skal udføre store matrixmultiplikationer for den lineære transformation. Det er dog ikke tilfældet. Matricerne er tyndt besat, med et indholdsmønster der gør, at multiplikationen kan forenkles, ud fra vores viden om, hvor elementerne i matricen ligger. Et eksempel er Hückelmatricen, givet ved

$$\mathbf{A} = \begin{bmatrix} \alpha & \beta & & \dots & & & \beta \\ \beta & \alpha & \beta & & & & \\ \ddots & \ddots & \ddots & & & & \\ & & & & & & \\ & & & & \ddots & \ddots & \ddots \\ & & & & \beta & \alpha & \beta \\ \beta & & & \dots & & \beta & \alpha \end{bmatrix} . \tag{7.58}$$

Når vi multiplicerer denne matrix med vektoren $\mathbf{x}$ for at opnå produktvektoren $\mathbf{y}$, er opskriften følgende:

$$y_1 = \alpha x_1 + \beta x_2 + \beta x_n \ , \tag{7.59}$$

derefter for alle $i = 2 \dots n-1$

$$y_i = \beta x_{i-1} + \alpha x_i + \beta x_{i+1} \ ; \tag{7.60}$$

og til sidst,

$$y_n = \beta x_1 + \beta x_{n-1} + \alpha x_n \ . \tag{7.61}$$

For en $n \times n$ matrix $\mathbf{A}$, betyder det således $3n$ multiplikationer samt n additioner, hvilket er langt mindre end de ca. n^2 operationer nødvendig for en generel, fuldt besat, matrix når n er stor.

Eksempel: Hückelmatricen

Hvis vi vil finde den laveste egenværdi af en Hückelmatrix som i (7.58), kan vi forenkle (7.55) til

$$\lambda^{(1)} = \frac{\left(\mathbf{x}^{(0)}\right)^T \overbrace{\left[\mathbf{A} - \lambda^{(0)}\mathbf{I}\right] \mathbf{x}^{(0)}}}{\left(\mathbf{x}^{(0)}\right)^T \mathbf{x}^{(0)}} \tag{7.62}$$

fordi her er alle diagonalelementer éns. Opskriften for dette problem er så følgende:

(a) sæt startværdier for $\mathbf{x}^{(0)}$; det kunne passende være vektoren $[0, 0, \frac{1}{\sqrt{n}}, \cdots, \frac{1}{\sqrt{n}}]^T$;

(b) udfør den lineære transformation (7.46) og beregn $\lambda^{(0)}$ ud fra (7.49);

(c) chek for konvergens ved at anvende (7.57); er $R < \delta$, exit fra iterationen;

(d) bestem $\lambda^{(1)}$, lign. (7.62);

(e) bestem $\mathbf{x}^{(1)}$, lign. (7.53);

(f) find $x_{forbedret}$ og benyt den som nyt $\mathbf{x}^{(0)}$;

(g) gå til (a).

En passende værdi for δ skal naturligvis sættes inden processen startes.

Kapitel 8

Funktionsevaluering

Computersprog som Fortran, Pascal, C++ *osv.* har indbyggede funktioner som `SQRT`, `SIN`, `EXP`, `LOG`, *osv.*, som skal evalueres på stedet af computeren. Og tit har man selv brug for en ny funktion der ikke er med, og så spekulerer man på, hvordan den kan evalueres. Tit griber man så til en download fra professionelle pakker, men det er alligevel interessant at vide, hvordan man gør. Et eksempel er fejlfunktionen `erf` og dens komplement, `erfc`, der bruges som eksempel i kurset `DatK`, hvor den evalueres ved integration.

Der er blandt andre følgende metoder der kan bruges:

- Numerisk integration
- Newtons metode
- approksimerende polynomier
- rationelle funktioner
- asymptotiske rækker.

I mange lærebøger finder man opskrifter for visse tricks, der siges at forbedre effektiviteten af en given metode. Mange af disse stammer fra en tid, hvor folk udførte disse beregninger med hånden, og de beskrevne tricks er i dag ofte ikke meget mere end kuriosummer, da en computer uden problemer bare kan øge antallet af iterationer eller af led i en række. Nogle af disse tricks nævnes alligevel nedenfor, fordi de synes interessant og desuden kan give en idé om strategier i numerisk regning.

For utallige eksempler af approksimationer til mange funktioner, henvises læseren til Abramowitz & Stegun [3].

8.1 Generelt

Præcisionskrav

Nogle aspekter i præcision er:

- Beregnede værdier bør være lige så præcise som de andre regnestykker computeren udfører. Det stiller ikke så skarpe krav til enkelt præcision, men ret skarpe til dobbelt præcision (dvs normalt ca. 16 eller flere decimaler).

- Nogle funktioner har kendte områder, og regninger må ikke resultere i værdier der træder ud af disse. For eksempel må sin x aldrig træde udenfor ± 1.000.

Der er to slags definition eller krav på præcision: absolut eller relativ. Hvis vi kalder den beregnede værdi v_b og den sande værdi $\hat{v}$, så er den absolutte fejl defineret ved

$$e_a = v_b - \hat{v} \tag{8.1}$$

og den relative fejl ved

$$e_r = \frac{v_b - \hat{v}}{\hat{v}} \; . \tag{8.2}$$

Den første kan være mere relevant for en funktion med et beskedent område, som *fx.* sinusfunktionen, mens en funktion som exponentialfunktionen skal have en vis relativ præcision.

Hvor kommer formlerne fra?

Nogle formler stammer simpelthen fra de kendte ekspansioner (de såkaldte McLaurinrækker). Et eksempel er eksponentialfunktionen, som også kendes som

$$\mathrm{e}^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \cdots \tag{8.3}$$

som for visse argumenter (mindre end 1 i størrelse) kan anvendes til at beregne funktionens værdi. Man kan finde sådanne rækker for mange andre funktioner så som logaritmer, de trigonometriske funktioner samt andre. I andre tilfælde har man tilpasset et polynomium til en funktion; et eksempel her er fejlfunktionen erf og dens komplement erfc, som også kan udtrykkes som McLaurinrækker, men de egner sig ikke så godt til evaluering. Polynomier er ofte beregnet empirisk som det bedste bud, dvs et polynomium, der passer ret godt over et bestemt område af funktionens argument.

Forhold mellem funktioner

Mange funktioner kan udtrykkes som funktioner af andre funktioner. I princippet behøver man ikke, for eksempel, at approksimere cosinusfunktionen, hvis man har en approksimation til sinusfunktionen, fordi de har en kendt relation

til hinanden. Det kan dog hænde, at man alligevel hellere vil bruge en særskildt approksimation, selv om man i princippet ikke behøver. Et eksempel er fejlfunktionen erf og dens komplement erfc. Det gælder jo at

$$\operatorname{erfc}(x) = 1 - \operatorname{erf}(x) \tag{8.4}$$

og så kunne man nøjes med kun den ene. Men for store x er der langt mere præcise formler for erfc end resultatet af at beregne erf og benytte ligning (8.4). I bogen af Press & al. [4] anbefales der at bruge den inkomplette gammafunktion til at beregne fejlfunktionen, mens der er et antal polynomiumsudtryk i Abramowitz & Stegun [3] for denne funktion.

Segmentering

I nogle tilfælde er det bedst at bruge forskellige formler for forskellige områder af argumenter til en funktion. Således bliver værdierne mere og mere unøjagtige for funktionen erfc hvis man bruger en approksimation for erf samt (8.4), jo større argumentet x bliver. For store x er en særskildt approksimation for erfc bedre. Dette kaldes (eng.) "segmentation". Det ser man også i de hyperbolske funktioner. For eksempel kan man bruge approksimationer for eksponentialfunktionen til at beregne $\sinh(x)$, men hvis $|x| \ll 1$, er det bedre at bruge en polynomiumsapproksimation i stedet, fordi man ellers trækker to temlig ens værdier (ca. 1) fra hinanden, hvilket er unøjagtigt.

Områdereduktion

Hvis man ser på funktionen $\log_{10}$ som et eksempel, er det klart at man ikke behøver at skrive en approksimation til tal i alle mulige størrelser, fordi man kender funktionen for potenser af 10. Man omskriver derfor argumentet x i formen

$$x = y \times 10^n \tag{8.5}$$

hvor n er det heltal, for hvilket $1 < y \leq 10$. Så er resultatet

$$\log_{10} x = n + \log_{10} y \; . \tag{8.6}$$

Dette er (eng.) "range reduction". Det samme gør man med mange funktioner med argumenter der potentielt har et uendeligt område. I tilfældet af funktionen ln, kan man bruge reduktionen

$$x = y \times 2^n \; , \tag{8.7}$$

reducere argumentet til $0.75 \leq y < 1.5$ og den endelige formel bliver derved

$$\ln x = n \ln 2 + \ln y \; . \tag{8.8}$$

Værdien $\ln 2$ kan beregnes i forvejen og lagres til brug.

Områdereduktion bliver brugt til mange funktioners approksimation, inklusiv kvadratrod. Her er det oplagt at bruge reduktionen

$$x = y \times n^2 \tag{8.9}$$

hvilket gør at y er reduceret til $0.25 \leq y < 1$. Formlen er så

$$\sqrt{x} = n\sqrt{y}\ . \tag{8.10}$$

For de trigonometriske funktioner, som er periodiske med 2π kunne man, for store argumenter, nøjes med at trække så mange gange 2π fra, indtil argumentet ligger i området $0 < x < 2\pi$, men også det ville betyde for stort et område. Her kan man bruge det endnu mindre område $0 < x < \pi/2$, samt de kendte simple relationer mellem de fire kvadranter.

Chebyshev-polynomier

Der er en familie af polynomier, som er yderst nyttige i sammenhæng med, blandt andet, funktionsapproksimationer. En gruppe af disse er alle betegnet som T_n, med $n = 0, 1, \ldots$. De første 11 kommer her:

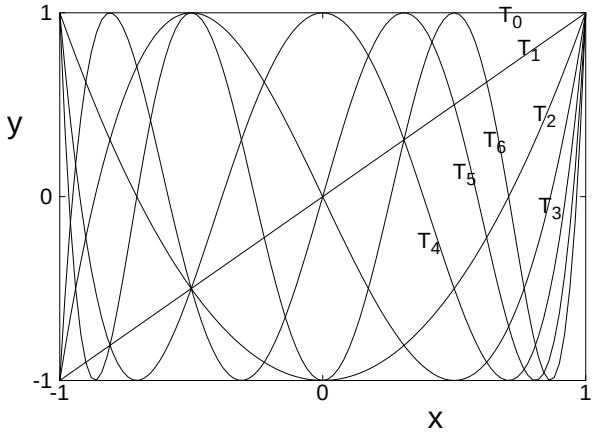
$$\begin{aligned}
T_0 &= 1 \\
T_1 &= x \\
T_2 &= 2x^2 - 1 \\
T_3 &= 4x^3 - 3x \\
T_4 &= 8x^4 - 8x^2 + 1 \\
T_5 &= 16x^5 - 20x^3 + 5x \\
T_6 &= 32x^6 - 48x^4 + 18x^2 - 1 \\
T_7 &= 64x^7 - 112x^5 + 56x^3 - 7x \\
T_8 &= 128x^8 - 256x^6 + 160x^4 - 32x^2 + 1 \\
T_9 &= 256x^9 - 576x^7 + 432x^5 - 120x^3 + 9x \\
T_{10} &= 512x^{10} - 1280x^8 + 1120x^6 - 400x^4 + 50x^2 - 1\ .
\end{aligned} \tag{8.11}$$

Man vil bemærke et antal regulariteter i disse formler. Der er en mere generel måde at skrive dem på:

$$T_n(x) = \cos\left(n \arccos x\right) \tag{8.12}$$

hvilket producerer alle. Dog er de ovenstående i de fleste tilfælde mere bekvemme at arbejde med.

En vigtig egenskab af alle disse polynomier er at, indenfor området $-1 \leq x \leq 1$, holder de sig indenfor det samme område $[-1, +1]$. Fig. 8.1 viser nogle af funktionerne. Man kan også vende formlerne om, og det kan være meget



Figur 8.1: De første 7 Chebyshevfunktioner i $[-1, +1]$

nyttigt, som vil ses senere:

$$\begin{aligned}
1 &= T_0 \\
x &= T_1 \\
x^2 &= \tfrac{1}{2}(T_0 + T_2) \\
x^3 &= \tfrac{1}{4}(3T_1 + T_3) \\
x^4 &= \tfrac{1}{8}(3T_0 + 4T_2 + T_4) \\
x^5 &= \tfrac{1}{16}(10T_1 + 5T_3 + T_5) \\
x^6 &= \tfrac{1}{32}(10T_0 + 15T_2 + 6T_4 + T_6) \\
x^7 &= \tfrac{1}{64}(35T_1 + 21T_5 + 7T_5 + T_7) \\
x^8 &= \tfrac{1}{128}(35T_0 + 56T_2 + 28T_4 + 8T_6 + T_8) \\
x^9 &= \tfrac{1}{256}(126T_1 + 84T_3 + 35T_5 + 9T_7 + T_9) \\
x^{10} &= \tfrac{1}{512}(126T_0 + 210T_2 + 120T_4 + 45T_6 + 10T_8 + T_{10}) \ .
\end{aligned} \tag{8.13}$$

Begge disse tabeller bruges til at komme frem til effektive polynomiumsrækker i afsnit 8.4.

8.2 Numerisk integration

Selvom den metode ikke er prominent i lærebøger om emnet, kan den bruges i tilfælde hvor man ikke behøver høj effektitivet men hurtig programmering. Man har for eksempel erfaring i at bruge integration, og har brug for fejlfunktionen, som let kan evalueres ved integration, og det kan være lettere end at bruge en af de flere metoder beskrevet i Abramowitz & Steguns bog [3], baserede på polynomier. Men generelt er integration ikke den foretrukne metode. Se kapitel 3 for denne metode.

8.3 Newtons metode

Newtons metode kan anvendes i de tilfælde, hvor den funktion der skal evalueres let kan vendes om. Kvadratroden er sådan én funktion. Vil vi have $\sqrt{a}$, kan vi evaluere det som rod af ligningen

$$f(x) = x^2 - a \tag{8.14}$$

hvilket er nemt med Newtons metode. Man benytter selvfølgeligt områdereduktion inden man begynder med løsningen. Her skal man beslutte en startværdi. Den kan meget passende være selve argumentet a, især med områdereduktion (men se afsnit 8.4). Opskriften er er altså følgende. Find n i ligningen (8.9), erstat (8.14) med

$$f(y) = y^2 - a/n^2 \tag{8.15}$$

og løs for y. Newtons generelle formel (2.4) på side 9 bliver her til

$$y_{n+1} = \frac{1}{2}\left(y_n + \frac{a}{n^2 y_n}\right) . \tag{8.16}$$

Det kan også betale sig at bruge denne metode til at finde en kubikrod, i stedet for at bruge logaritmer (som computere generelt gør for sådanne rødder). Vil vi have kubikroden af a, kan vi skrive

$$f(x) = x^3 - a \tag{8.17}$$

og løse det. Fike [7] anbefaler dog at dividere med x, hvilket giver

$$f(x) = x^2 - a/x . \tag{8.18}$$

Newton-formlen bliver så

$$x_{n+1} = x_n - \frac{x_n^2 - a/x_n}{2x_n + a/x_n^2} . \tag{8.19}$$

Denne form siges at give en hurtigere konvergens. Naturligvis er det også her oplagt at bruge områdereduktion.

8.4 Polynomier

Polynomier bruges i høj grad til at approximere funktioner med. Vi kender alle polynomiumsrækken for $\exp(x)$, McLaurinrækken

$$\mathrm{e}^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} \cdots \tag{8.20}$$

og der findes et hav af sådanne rækker i bogen Abramowitz & Stegun [3]. Tit er der varianter, der egner sig bedst for visse argumentområder. For eksempel er der polynomiumsrækker for $\ln(x)$ (flere varianter), $\ln(1+x)$ og $\ln(\frac{x+1}{x-1})$ *mm.* (se Abramowitz & Stegun [3]).

Effektiv evaluering af et polynomium

Et polynomium udtrykkes i formen

$$p(x) = a_0 + a_1x + a_2x^2 + a_3x^3 + \cdots + a_nx^n \tag{8.21}$$

og det leder tanken umiddelbart hen til en programstump som denne

```
p = a(0)
xpow = x
do i = 1, n
  p = p + a(i)*xpow
  xpow = xpow * x
enddo
```

(forudsat at `a(0:n)` er erklæret som række og at dens værdier er kendte). Denne algoritme bruger $2n$ multiplikationer samt n additioner, altså har vi i alt $3n$ flops. Vi undgår her at bruge de mere dyre potenser `x**i`, men alligevel kan man forbedre algoritmen, baseret på omskrivningen af (8.22) i formen

$$p(x) = (\cdots(a_nx + a_{n-1})x + a_{n-2})x + \cdots + a_1)x + a_0 \ , \tag{8.22}$$

for eksempel for et fjerdegrads polynomium

$$p(x) = (((a_4x + a_3)x + a_2)x + a_1)x + a_0 \ . \tag{8.23}$$

Det kan man programmere baglæns på følgende måde

```
p = 0
do i = n, 1, -1
  p = (p+a(i)) * x
enddo
```

hvilket kun bruger $2n$ flops. Hvis n er stor, sparer det nogen computertid. Dette er måske en af de ovennævnte kuriosummer, da besparelsen på det niveau er i dag nok uden megen betydning. Det der muligvis tæller mere er, at algoritmen ikke umiddelbart er lige så nem at gennemskue end den første, og det betyder muligvis mere i overvejelserne end en besparelse på ca. 30% cpu-tid.

Brug af Chebyshevpolynomier

Der er to anvendelser af Chebyshevpolynomier (se afsnittet 8.1) i forbindelse med polynomiumsapproksimationer. Den ene betyder at vi kan forkorte polynomiumsrækker og derfor gør dem (lidt) hurtigere at evaluere, og den anden leverer en transformerede række som, med samme antal led, forøger regnepræcisionen.

Det første af de to har det engelske navn “economised power series”. Metoden illustreres her ved hjælp af eksemplet (8.20) for eksponentialfunktionen.

Lad os sige, at vi vil approksimere den med syv led i rækken. Vi skriver nu fakulteterne ud, og bruger som start approksimationen

$$\mathrm{e}^x \approx 1 + x + \frac{x^2}{2} + \frac{x^3}{6} + \frac{x^4}{24} + \frac{x^5}{120} + \frac{x^6}{720} \; . \tag{8.24}$$

Notér tabellen af Chebyshevpolynomierne i (8.11). Lad os trække fra summen i (8.24) en mængde lig med $\frac{1}{720}\frac{T_6}{32}$. Vi ved, at T_6, lige som alle T_n, inden for området $[-1,+1]$ holder sig inden for området $[-1,+1]$, og at derfor vores subtraktion af denne term ikke betyder mere end $\frac{1}{720\times 32}$ eller 0.000043. Er vi tilfreds med en præcision på 4 cifre, gør denne subtraktion ikke forskel. Men se nu hvad formlen bliver til, når vi ændrer den til

$$\begin{aligned}\mathrm{e}^x \approx 1 + x + \frac{x^2}{2} + \frac{x^3}{6} + \frac{x^4}{24} + \frac{x^5}{120} + \frac{x^6}{720} \\ - \frac{1}{720 \times 32}\left(32x^6 - 48x^4 + 18x^2 - 1\right) \; ,\end{aligned} \tag{8.25}$$

hvilket eliminerer termen i x^6 og så bliver til

$$\mathrm{e}^x \approx 1.000043 + x + 0.499219x^2 + \frac{1}{6}x^3 + 0.043750x^4 + \frac{1}{120}x^5 \; . \tag{8.26}$$

Vi har altså forkortet rækken om et led, og bibeholdt den ønskede præcision. Vi kan blive ved og forkorte mere. Naturligvis er det et groft eksempel, og normalt ville man angribe polynomiumsrækken et langt højere sted. Også det kan opfattes som kuriosum, da vi næppe vinder ret meget cpu-tid med denne fidus.

Den anden anvendelse af Chebyshevpolynomier er et erstatte et polynonium med et andet, baseret på Chebyshevrækker. Det giver en ny række med samme antal led, men fejlen kan minimeres i vis grad. Igen bruger vi eksponentialfunktionen som eksempel, denne gang kun som i (8.20) op til led med x^4. Alle potenser bliver erstattet med deres inverse Chebyshevudtryk set i tabellen (8.13). Approximationen (8.20) bliver så til

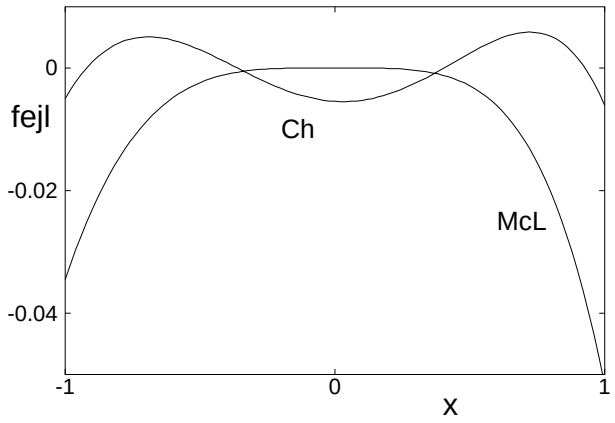
$$\begin{aligned}\mathrm{e}^x \approx T_0 + T_1 + \frac{1}{4}(T_0 + T_2) + \frac{1}{24}(3T_1 + T_3) + \frac{1}{192}(3T_0 + 4T_2 + \cdots) \\ + \frac{1}{1920}(10T_1 + 5T_3 + \cdots) + \frac{1}{23040}(10T_0 + 15T_2 + \cdots) + \cdots\end{aligned} \tag{8.27}$$

hvor vi for alle terme har udeladt T_n for $n > 3$. Ved lidt besvær giver dette så den nye række

$$\mathrm{e}^x \approx 1.2661T_0 + 1.1303T_1 + 0.2715T_2 + 0.0443T_3 \tag{8.28}$$

og, hvis vi ønsker at have det med potenser af argumentet x i stedet, kan vi ekspandere alle T_n i følge tabellen (8.11), hvilket giver til sidst den nye formel

$$\mathrm{e}^x \approx 0.9946 + 0.9973x + 0.5430x^2 + 0.1773x^3 \; . \tag{8.29}$$



Figur 8.2: Fejlene af McLaurin (McL) og Chebyshevpolynomier (Ch) for $\exp(x)$

Fig. 8.2 viser forskellen mellem de to formler, McLaurin (8.20) gående til x^4 og (8.29). Den sidste har en fejl ved $x = 0$, men fejlene over området $[-1, +1]$ er generelt mindre end dem fra McLaurinformlen, og det har vi opnået med et færre terme i approksimationen.

Minimax polynomier

Når man vil tilpasse et polynomium $P_n(x)$ til en funktion $f(x)$, uden at benytte en McLaurinrække, er der altid et optimalt polynomium, i den forstand, at det har en minimum største fejl $|P_n(x) - f(x)|$ (absolut) eller $|(P_n(x) - f(x))/f(x)|$ (relativ) i det område, funktionen skal tilpasses. Alle polynomier vil have en fejlfunktion der varierer i området, og inden for det er der en maksimum magnitude i fejlen. Den skal naturligvis gerne minimeres, og et såkaldt minimaxpolynomium er det, der gør det bedst. Fike [7] giver en detaljeret beskrivelse af, hvordan man kommer frem til sådan en approksimation, hvor man iterativt finder både polynomiets koefficienter og punkter i området, hvor der er de største udsving i fejlene. Læseren er henvist til Fike [7] for detaljer. Her nøjes vi med et yderst simpelt eksempel. Hvis vores funktion er $\sqrt{x}$ inden for området $[\frac{1}{16}, 1]$, og vi gerne vil tilpasse et førstegrads polynomium $a_0 + a_1 x$, så er det let at finde frem til, at minimax-funktionen er $\frac{2}{9} + \frac{8}{9}x$, med 3 maksimale relative fejl, ved $x = \frac{1}{16}, 0.25$ og 1. Fejlene er alle éns i størrelse, henholdsvis $\frac{1}{9}, -\frac{1}{9}, \frac{1}{9}$, *ca.* 11%. Det virker ikke særlig tilfredsstillende som approksimation; men det giver en startværdi for Newtonløsningen beskrevet i afsnittet 8.3, som kræver færre iterationer til konvergens for de fleste argumenter i området.

Det er ofte sådan, som noteret i Press & al. [4], at et tilpasset Chebyshevpolynomium er meget tæt på at være minimax, og de er meget nemmere at beregne. Se eksemplet i det foregående afsnit.

8.5 Asymptotiske rækker

Der er funktioner for hvilke det ikke er muligt at bruge områdereduktion, hvor man altså er nødt til at evaluere en funktion for store argumenter. I disse tilfælde kan en polynomiumsrække være umulig at bruge. En sådan funktion er, for eksempel, erfc, eller også erf. Her benytter man tit rækker af inverse potenser, kaldt asymptotiske rækker eller ekspansioner. Der er dog også asymptotiske rækker for funktioner som *kan* områdereduceres eller for hvilke der er andre approksimationer.

Den generelle formel for en asymptotisk række er

$$f(x) \approx b_0 + b_1 x^{-1} + b_2 x^{-2} + \cdots \ . \tag{8.30}$$

For erfc er der rækken,

$$\mathrm{erfc}(x) = \frac{1}{\sqrt{\pi} x \exp(x^2)} \left(1 - \frac{1}{2} x^{-2} + \frac{1 \cdot 3}{2^2} x^{-4} + \frac{1 \cdot 3 \cdot 5}{2^3} x^{-6} \cdots \right) \ . \tag{8.31}$$

For gammafunktionen, defineret ved

$$\Gamma(x) = \int_0^\infty t^{x-1} \exp(-t) \mathrm{d}t \ , \tag{8.32}$$

er der (Abramowitz & Stegun [3]) et par polynomiumsrækker med forskellig præcision, samt en asymptotisk række for store argumenter.

8.6 Rationelle funktioner og kædebrøker

Det engelske udtryk "rational functions" er her direkte oversat, og ligeledes er "continued fractions" oversat til kædebrøker. De udgør endnu flere metoder til at approksimere funktioner på, og er fattet sammen her fordi de er relateret til hinanden.

En rationel funktion er en brøk, hvor der er polynomier både som tæller og nævner, altså generelt

$$f(x) = \frac{a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n}{1 + b_1 x + b_2 x^2 + \cdots + b_m x^m} \tag{8.33}$$

(der er ingen b_0 fordi det kan bare faktoriseres ud). Sådan en brøk kan, ved (møjsom) division konverteres til en kædebrøk, som enten er afsluttet som i

$$f(x) = b_0 + \cfrac{a_1}{b_1 + \cfrac{a_2}{b_2 + \cfrac{a_3}{b_3 + \cfrac{a_4}{b_4 + \ddots + \cfrac{a_n}{b_n}}}}} \tag{8.34}$$

eller som bare fortsætter, hvor man så selv bestemmer, hvor den skal hugges af.

Mange lærebøger bruger en anden måde at præsentere disse (for bogtrykkere) vanskelige formler på. Ovenstående ligning (8.34) ofte trykkes således i formen

$$f(x) = b_0 + \frac{a_1}{b_1+}\ \frac{a_2}{b_2+}\ \frac{a_3}{b_3+}\ \frac{a_4}{b_4+} \cdots \frac{a_n}{b_n} \tag{8.35}$$

(der er flere varianter, i lignende stil).

Et udtryk som (8.34) kan kodes i Fortran på følgende måde, gående baglæns nedefra:

```
sum = a(n) / b(n)
do i = n-1, 1, -1
  sum = a(i) / (b(i) + sum)
enddo
func = b(0) + sum
```

Her blev den terminerende del startværdien af variablen `sum`. Rationelle brøkker kan let evalueres på samme vis som polynomier, som beskrevet ovenstående (8.22).

Som nævnt kan man tit konvertere en rationel funktion til en kædebrøk ved algebraisk division. Eksempler kan findes i Fike [7]. Det kan føre til lidt mere enkelte udtryk, men må siges i dag ikke at være helt så interessant som det var inden computerens fremmarch.

Kapitel 9

Fouriertransformation

9.1 Indledning

Fouriertransformationen repræsenterer en særdeles vigtig metode inden for moderne databehandling og anvendes som et uundværligt analytisk/numerisk redskab i en lang række matematiske, fysiske og kemiske discipliner.

Først beskrives der noget som ikke rigtigt er en Fouriertransformation, men er relateret til den og belyser, hvad der foregår når man benytter den.

Lad os bruge en cosinus- og sinustransformation på nogle forskellige trigonometriske funktioner af tiden t, $h(t)$. Vi transformerer funktionen ved at integrere den, ganget med cosinus- eller sinusfunktionen. Således har vi de to formler

$$H_c(f) = \frac{1}{T}\int_0^T h(t)\cos(2\pi f t)\,\mathrm{d}t \tag{9.1}$$

og

$$H_s(f) = \frac{1}{T}\int_0^T h(t)\sin(2\pi f t)\,\mathrm{d}t\ . \tag{9.2}$$

Her er f frekvensen, normalt i Hz eller s^{-1} (men funktionen behøver ikke at være en funktion af tid). Ofte forkorter man udtrykkene ved at bruge vinkelfrekvensen $\omega = 2\pi f$, som giver de mere kompakte formler

$$H_c(\omega) = \frac{1}{T}\int_0^T h(t)\cos\omega t\,\mathrm{d}t \tag{9.3}$$

og

$$H_s(\omega) = \frac{1}{T}\int_0^T h(t)\sin\omega t\,\mathrm{d}t\ . \tag{9.4}$$

Disse transformationer afviger fra den egentlige Fouriertransformation idet de er specifikke for cosinus og sinus, at der integreres over et begrænset tidsområde, samt at vi dividerer med tidsområdet T. Det gør det muligt for os at

transformere rene trigonometriske funktioner over et begrænset område, som ellers ikke (strengt sagt) kan fouriertransformeres.

Sæt nu $h(t) = \cos \omega t$ og $T = 2\pi/\omega$, *dvs.*, vi integrerer over en hel periode af funktionen $h(t)$. Så er det, ved opslag i tabeller af integraler, let at finde ud af, at $H_c(\omega) = \frac{1}{2}$. Hvis vi derimod benytter (9.4), får vi resultatet $H_s(\omega) = 0$.

Hvis vi gentager processen for $h(t) = \sin \omega t$, får vi det modsatte: nu er $H_c(\omega) = 0$ og $H_s(\omega) = \frac{1}{2}$ for alle værdier af ω (undtaget 0).

Nu lader vi funktionen være en trigonometrisk funktion med en anden frekvens, mens vi stadig ganger den med enten $\cos \omega t$ eller $\sin \omega t$: lad $h(t) = \cos a\omega t$, med $a \neq 1$. Vi integrerer nu over et variabelt område $[0, T]$. Processen kan gennemføres ved hjælp af identiteten

$$\cos x \cos y = \tfrac{1}{2} \left(\cos(x - y) + \cos(x + y)\right) \tag{9.5}$$

samt tabelopslag, og vi får endeligt at

$$H_c(\omega) = \frac{(a+1)\sin([a-1]\omega T) + (a-1)\sin([a+1]\omega T)}{2(a^2-1)\omega T} \,. \tag{9.6}$$

Inspektion af resultatet viser at det er en aftagende funktion af T, som går mod nul for store T. Samme procedure for funktionen med vores sinustransformation (9.4) giver et lignende resultat, igen lig nul for stor T.

Her har vi lært noget. Hvis vi sætter $h(t) = \cos a\omega t$ og lader a variere fra nul opad, og sætter T stor og til et multipel af 2π, så får vi at $H_c(\omega)$ er tæt på nul, undtaget når $a = 1$. Transformationen (9.3) reagerer altså (for stor T) kun på en cosinusbølge med samme frekvens ω, og ikke på andre frekvenser. Desuden reagerer den heller ikke på en sinusbølge, lige meget hvilken frekvens den har. Analog med det, reagerer (9.4) kun på en sinusbølge med samme frekvens ω, og ikke på andre frekvenser eller en cosinusbølge.

Målesignaler består ofte, eller kan repræsenteres som, et antal cosinus- og sinusbølger med forskellige frekvenser. Vi vil gerne finde amplituderne af disse komponenter. Vi kan nu let se, at de to transformationer kan pille disse komponenter ud af sådan en blanding af bølger. Operationerne (9.3) og (9.4) er lineære, *dvs.* transformationen af $bh(t)$ giver b gange den for $h(t)$. Hvis signalet altså består af en blanding af cosinusbølger med forskellige frekvenser, og vi transformerer med variabel ω og (9.3), hver gang vi rammer en ω-værdi der matcher en af bølgene, er resultatet denne bølges (halverede) amplitude. Det kalder vi for et spektrum.

Vi mangler kun en ting mere. Hvis vi nu lader vores funktion være en cosinusbølge med et faseskift: $h(t) = \cos(\omega t + \theta)$, hvad er så resultatet? Ved at anvende (9.3) får vi nu at $H_c(\omega) = \frac{1}{2}\cos\theta$, og at $H_s(\omega) = \frac{1}{2}\sin\theta$ for alle ω. Nu reagerer begge transformationer, og resultaterne afspejler faseskiftet. Det vil sige, at hvis vi anvender begge transformationer, får vi to tal, der giver os information om den bølges amplitude samt dens faseforhold. Vi kan også udtrykke det som, at vi har opdelt bølgen i dens cosinus- og sinuskomponenter. Det er altså en god ide, at anvende begge transformationer samtidigt.

Vi er nu klar til den egentlige Fouriertransformation.

9.2 Kontinuert Fouriertransformation

Vi holder os her til notationen i Press & al. [4]. Der er forskellige varianter for hvordan Fouriertransformationen bliver præsenteret. Én af definitionerne er

$$H(f) = \int_{-\infty}^{\infty} h(t)\, \mathrm{e}^{i2\pi ft} \mathrm{d}t \tag{9.7}$$

og for inverset

$$h(t) = \int_{-\infty}^{\infty} H(f)\, \mathrm{e}^{-i2\pi ft} \mathrm{d}f \tag{9.8}$$

hvori $i = \sqrt{-1}$. Press & al. [4] har fortegnene i eksponentialdelene i denne orden, men de fleste lærebøger definerer transformene med fortegnene byttet om. Der er desuden mange varianter på faktorer, integralerne kan ganges med, men alle varianter kan bruges i praksis. Den første af de to fører os fra tidsdomænet til frekvensdomænet, og den anden er dens inverse der fører os tilbage. Vi skriver funktioner af tid med små bogstaver, som $h(t)$, og deres transform $H(f)$ med store. Her har vi nu integrationsgrænser der går fra $-\infty$ til $+\infty$, og det giver nogle begrænsninger eller betingelser.

Hvis man husker identiteten

$$\cos x + i \sin x = \mathrm{e}^{ix} \tag{9.9}$$

så kan man se, at integrationen (9.7) faktisk står for både en cosinus- og en sinustransformation, og at vi derved får begge transformationer samtidigt, gemt i den reelle og den imaginære del af resultatet. Vi har altså her at gøre med komplekse tal; det gælder i princippet også for $h(t)$, som dog oftest er reel. Men som vi ser senere, skal det i praksis repræsenteres som komplekstal, oftest med et nul i den imaginære del.

Vi bruger igen Press & al's [4] notation, som repræsenterer et transformationspar i form af

$$h(t) \Leftrightarrow H(f) \tag{9.10}$$

som forenkler mange af de følgende udtryk.

Betingelser

Der er nogle betingelser for at kunne udføre disse operationer. Den mest vigtige, som er tilstrækkelig men ikke helt nødvendig, er at selve funktionen $h(t)$ opfylder at

$$\int_{-\infty}^{\infty} |h(t)| \mathrm{d}t < \infty \tag{9.11}$$

hvilket tilsyneladende udelukker de trigonometriske funktioner vi brugte i det ovenstående, samt nogle andre. Men betingelsen er ikke helt nødvendig; den

garanterer at transformationen kan lade sig gøre, men der er funktioner som ikke opfylder betingelsen for hvilke der alligevel en transformation er mulig. Én af dem er sinc-funktionen $\sin(\omega t)/\omega t$. Læseren henvises til Brigham [8] for de teoretiske detaljer. Men i praksis har vi at gøre med signaler (funktioner af tid eller afstand) der *er* givet indenfor et begrænset område, og her integrerer man så med de limits der er givet. Dette giver også et bindeled til den diskrete transformation, der følger nedenfor. Men selv med de uendelige grænser kan man komme frem til en transformation. Således siger man at hvis $h(t) = \cos(2\pi f_0 t)$, så er $H(f)$ to linjer ved $f = \mp f_0$, og af uendelig magnitude. Det svarer til to "deltafunktioner" $\frac{a}{2}\delta(f \mp f_0)$. På denne måde har vi så lov til at tale om transformationerne af disse funktioner, selvom de ikke opfylder (9.11).

Nogle relationer

Press & al. [4] opstiller en tabel som kort og tydeligt fortæller nogle interessante forhold, og vi følger stilen:

Hvis ...	så...
$h(t)$ er reel	$H(-f) = H^*(f)$
$h(t)$ er imaginær	$H(-f) = -H^*(f)$
$h(t)$ er lig	$H(-f) = H(f)$ (*dvs.* $H(f)$ er lig)
$h(t)$ er ulig	$H(-f) = -H(f)$ (*dvs.* $H(f)$ er ulig)
$h(t)$ er reel og lig	$H(f)$ er reel og lig
$h(t)$ er reel og ulig	$H(f)$ er imaginær og ulig
$h(t)$ er imaginær og lig	$H(f)$ er imaginær og lig
$h(t)$ er imaginær og ulig	$H(f)$ er reel og ulig

Notationen $H^*(f)$ betegner det komplekse komplement.

Følgende relationer er også interessante:

$ah(t)$	$\Leftrightarrow$	$aH(f)$	linearitet
$h(t) + g(t)$	$\Leftrightarrow$	$H(f) + G(f)$	additivitet
$h(at)$	$\Leftrightarrow$	$\frac{1}{\lvert a\rvert}H(\frac{f}{a})$	tidsskalering
$\frac{1}{\lvert b\rvert}h(\frac{t}{b})$	$\Leftrightarrow$	$H(bf)$	frekvensskalering
$h(t - t_0)$	$\Leftrightarrow$	$H(f)\mathrm{e}^{i2\pi f t_0}$	tidsforskydning
$h(t)\mathrm{e}^{-i2\pi f_0 t}$	$\Leftrightarrow$	$H(f - f_0)$	frekvensforskydning
$\frac{\mathrm{d}^n}{\mathrm{d}t^n}(h(t))$	$\Leftrightarrow$	$(-i2\pi f)^n H(f)$	differentiering

Foldning

To funktioner, $h(t)$ og $g(t)$, foldes efter definitionen

$$h(t) * g(t) \equiv \int_{-\infty}^{\infty} g(\tau)h(t-\tau)\mathrm{d}\tau \ . \tag{9.12}$$

Denne operation, for et område af t, er tidskrævende, men kan lempes på basis af transformationsparret

$$h(t) * g(t) \Leftrightarrow G(f)\,H(f) = H(f)\,G(f) \tag{9.13}$$

dvs. kan beregnes i frekvensdomænet ved simpel multiplikation af de to funktioners transformer. Det er mindre regnetungt takket være den hurtige FFT, som beskreves nedenfor.

Korrelation

Korrelation mellem to funktioner er relateret til foldning, og også her er der en nyttig transformsammenhæng. Korrelation er defineret ved

$$\mathrm{corr}(g,h) \equiv \int_{-\infty}^{\infty} g(t+\tau)h(\tau)\mathrm{d}\tau \tag{9.14}$$

og her er transformationsparret

$$\mathrm{corr}(g,h) \Leftrightarrow G(f)\,H^*(f) \ . \tag{9.15}$$

Hvis de to funktioner er forskellige, er der tale om krydskorrelation, for et område t, og når de to funktioner er den samme funktion, taler vi om en autokorrelation. Det kan opfattes som en funktion af t, som giver et udsagn om, hvor hvidt funktionen ligner sig selv på en afstand t. Hvis, for eksempel, vi har en cosinusbølge og $t = 2\pi t_0$ (*dvs.* en afstand på en hel bølgelængde), så ligner funktionen fulstændig sig selv der (det giver jo det samme som funktionen i anden $\cos^2 \omega t_0$); hvilket giver en "autokorrelation" på 1. Derimod er det modsat for $t = \pi t_0$, hvor autokorrelationen er lig -1, og for $t = \frac{1}{2}\pi t_0$ er den lig nul.

Autokorrelationskoefficienten er defineret for en kontinuært funktion som

$$r(\tau) = \lim_{T\to\infty} \frac{1}{T} \int_0^T h(t)h(t+\tau)\mathrm{d}t \tag{9.16}$$

hvor vi nu bruger τ som symbolet for afstanden. Denne formel ændres for en diskret sekvens af længde n (se næste afsnit) til

$$r(\tau) = \frac{1}{n-\tau} \sum_{i=1}^{n-\tau} h_i h_{i+\tau} \ . \tag{9.17}$$

Grunden til at lade i løbe kun til $n-\tau$ er selvfølgeligt, at der ellers ikke er en værdi ved indekset $i+\tau$. Denne formel, som for store n er regnetung, kan ifølge sammenhængen mellem autokorrelation og power spektrum beregnes effektiv ved hjælp af den diskrete Fouriertransform.

9.3 Diskrete Fouriertransformation DFT

Vi har ofte at gøre med såkaldte tidssekvenser eller generelt sekvenser af måletal målt i samme afstand i den uafhængige variabel, som tit er tid. Her er der altså tale om sekvenser af tal. De repræsenterer punkter eller "samples" h_k på en underliggende kontinuærte funktion $h(t)$. Man siger at sekvensen er "samplet" i tidsdomænet med en vis samplingafstand δt eller med en samplingsfekvens $f_s = \delta t^{-1}$. Vi vil nu gerne beregne en Fouriertransform af sådan en sekvens, som gerne skulle ligne Fouriertransformationen af funktionen. Vi skal altså lave den kontinuærte definition om til en diskret implementering af integralerne (9.7) og (9.8). Det er klart, at vi ikke integrerer fra $-\infty$ til $+\infty$, og vores sekvens er heller ikke uendeligt lang. En skjult antagelse her er, at sekvensen $h_k, k = 0 \ldots N-1$ gentager sig selv uendeligt.

Inden vi går i gang, må vi etablere nogle vigtige sammenhæng mellem størrelser i tidsdomænet og frekvensdomænet. I tidsdomænet er der N punkter $h_k, k = 0, \ldots N-1$, og sekvensen svarer til et område af længden T. Nummereringen går fra $0 \ldots N-1$, for at lette udtrykket for transformationen. Det vil sige, at intervallerne δt er lige med T/N (vi holder os her til ideen af en tidssekvens, selv om der også kan opstå sekvenser hen af en anden dimension, som for eksempel afstand; så er der andre symboler, men alt bliver behandlet tilsvarende som her). Når man transformerer sådan en sekvens, får man en ny sekvens af punkter i spektrummet, hvor den ny underliggende uafhængige variabel er frekvens f (eller generelt den inverse af dimensionen af den originale sekvens). De to uafhængige variabler er ikke med i sekvenserne; de er noget, vi selv må være opmærksom på. Det vil sige, vi skal kende T og δt. En ting, der komplicerer sagen er, at spektrummet er en funktion af en frekvensvariabel, som går fra en negativ værdi, over nul, til en positiv værdi, og også det må vi holde i øjnene. Den mindste frekvens, der kan være i en tidssekvens, er den der svarer til netop en hel bølgelængde over hele sekvensens længde. Den mindste frekvens er altså T^{-1}. Alle andre frekvenser er multipler af denne værdi, som vi kalder δf. Spektrummet består af N punkter $H_k, k = -\frac{N}{2}, \ldots \frac{N}{2} - 1$, gældene for frekvenser f_k, og de løber fra $-\frac{N}{2}\delta f$ igennem nul og op til $(\frac{N}{2} - 1)\delta f$.

Her er en opsummering af det ovenstående:

- Tidssekvensen er $h_k, k = 0, \ldots N-1$
- Sekvensen svarer til tiderne $t_k = k\delta t, k = 0, \ldots N-1$.
- Den totale tid er $T = N\delta t$.
- Samplingintervallerne $\delta t = \frac{T}{N}$.
- Spektrumsekvensen er $H_k, k = -\frac{N}{2}, \ldots \frac{N}{2} - 1$.
- Spekrummet svarer til frequenserne $f_k, -\frac{N}{2}\delta f, \ldots (\frac{N}{2} - 1)\delta f$
- Frekvensintervallerne $\delta f = \frac{1}{T}$.

- Samplingfrekvens $f_s = \frac{1}{\delta t}$.

De kontinuerte integraler (9.7) og (9.8) kan nu udtrykkes i diskret form som summer. Vi mindes om, at sekvensen, der transformeres, i sig selv ikke bærer information om samplingintervallene; disse tages derfor som værende lige med 1. Den maksimale tid T er således lig med N. Det medfører at $\delta f = N^{-1}$. Man skal selv skalere frekvenserne i spektrummet. Formlerne er

$$H_j = \sum_{k=0}^{N-1} h_k \mathrm{e}^{i2\pi f_k t_j} \tag{9.18}$$

og, da $f_k = k\delta f = k/N$ og $t_j = j\delta t = j$, bliver det til

$$H_j = \sum_{k=0}^{N-1} h_k \mathrm{e}^{i2\pi jk/N} \ . \tag{9.19}$$

Bemærk at vi bør gange summen med δt, men det er jo lig med 1. Den inverse transform er

$$h_j = \sum_{k=0}^{N-1} H_k \mathrm{e}^{-i2\pi t_k f_j} \tag{9.20}$$

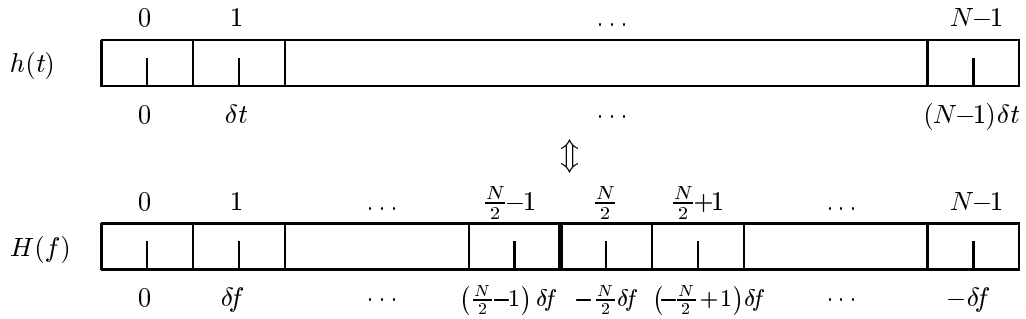
og det bliver til den tilsvarende

$$h_j = \sum_{k=0}^{N-1} H_k \mathrm{e}^{-i2\pi jk/N} \ . \tag{9.21}$$

Hver af de to formler skal udføres for alle mulige (*dvs.* N) f_k eller t_k, og for hver af dem skal der udregnes N eksponentialer. Det giver i alt N^2 regnestykker, hvoraf eksponentialerne tager den meste regnetid. De to former kaldes DFT efter den diskrete Fouriertransformation.

Der er et lille forbedringstrick: Det vi gør her er at erstatte et integral af en sum, der viser sig at være baseret på blokintegration. Det er den mindst eksakte metode at integrere på. Fra kapitel 3 ved vi imidlertid, at trapezformlen er meget bedre, og at den er næsten det samme som blokformlen, blot at den første og sidste værdi i sekvensen vægtes med en faktor $\frac{1}{2}$. Det kan vi sikre simpelthen ved at halvere de to værdier i sekvensen, og så får vi et bedre resultat. I praksis skal man selv sørge for dette, fordi de professionelle computerrutiner for en DFT (eller, som følger, FFT) ikke fortager sådanne halveringer selv.

Der er en yderligere mindre komplikation her. Når der bliver udført transformationen (9.19) for alle frekvenser, har vi en spektrumssekvens. Den er imidlertid i en mærkelig rækkefølge, se Fig. 9.1. De første $N/2$ punkter i $H(f)$-sekvensen står, som måske forventet, for frekvenserne $0, \delta f, 2\delta f, \ldots, (\frac{N}{2} - 1)\delta f$. Men der var jo også punkterne med negative frekvenser, og de kommer så i



Figur 9.1: Nummerering og orden ved Fouriertransformation. Hver kasse er delt i den reele og imaginære del. I frekvensdomænet kommer de positive frekvenser først, efterfulgt i anden halvdel af de negative.

den anden halvdel, gående i rækkefølgen $-\frac{N}{2}\delta f, (-\frac{N}{2}+1)\delta f, \ldots, -\delta f$. Hvis man vil have et spektrum ud af denne sekvens, må man bytte om på de to halvdele, så man får punkter svarende til frekvensrækken $-\frac{N}{2}\delta f, (-\frac{N}{2}+1)\delta f, \ldots, -\delta f, 0, \delta f, 2\delta f, \ldots (\frac{N}{2}-1)\delta f$.

Vi har at gøre med komplekse tal i begge domæner. Og til sidst er der et skaleringsaspekt. Man skulle kunne transformere fra tidsdomænet, sekvensen h, til frekvensdomænet H og tilbage igen. Når man gør dette, finder man ud af, at de oprindelige værdier er nu ganget med N. Det skal man være bevidst på hvis selve skaleringen er vigtig. Normalt betyder det ikke noget.

Fast Fouriertransform FFT

En Fouriertransformation udført efter ligningerne (9.19) eller (9.21) kræver, for en sekvens af N punkter, N^2 multiplikationer og, værre endnu, lige så mange beregninger af eksponentialer. Det kan blive rigtigt regnetungt, når N bliver stor. Der er heldigvis metoder til drastisk at forkorte processen. Den ene blev opdaget af Runge allerede i 1903 [9]. Han konstaterede, at der ikke er N^2 forskellige eksponentialværdier at beregne, men kun N, fordi mange af dem kan bringes ned på de første N. Hvis argumentet i e^{ix} overgår $i2\pi$, så kan man bare trække $i2\pi$ fra argumentet, og så havner man på et argument man allerede har regnet med. Det letter regningen betydeligt.

En anden måde at lette processen beskrives i Press et al. [4] og er baseret på en opdagelse af Danielson og Lanczos i 1942 [10]. Kort sagt går det ud på, at man kan splitte en Fouriertransformation af N punkter op i to, hver af $N/2$ punkter, som består af rækken af dem med lig nummerering $H^{(l)}$, og rækken af dem med ulig nummerering, $H^{(u)}$. Formlen er

$$H(j) = H^{(l)}(j) + \mathrm{e}^{i2\pi j/N} H^{(u)}(j) \; . \tag{9.22}$$

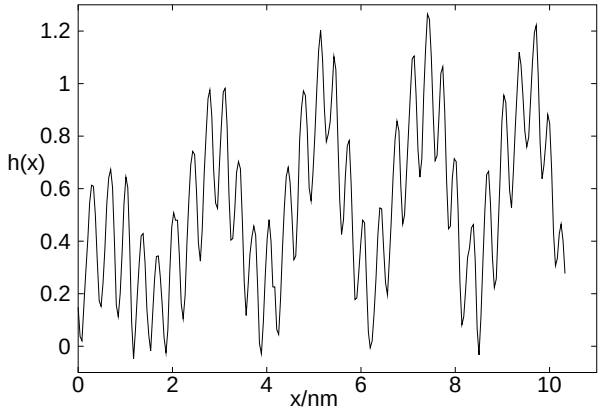
Hvis man gør det bare én gang, har man allerede sparet meget regning, fordi man nu kun har med $N^2/4$ operationer at gøre. Men det standser ikke her.

Sætningen gælder også de halve dele, som også kan splittes op i yderligere halvdele, *osv.* Det kræver en del administration, at holde orden i eksponentialerne og hvad der skal ganges med og lægges sammen med hvad, men det ender med, at det kræver blot $N \log_2 N$ operationer. Hvis for eksempel $N = 1000$, taler vi her om 10^6 operationer for DFT, og kun ca. 10^4 for den forbedrede algoritme, som har navnet **Fast Fourier Transform** eller **FFT**. Den interesserede læser henvises til Press & al. [4] eller Brigham [8] for en nærmere bekrivelse af algoritmen. Enderesultatet er selvfølgeligt præcist den samme sekvens af tal, som ved DFT.

For at FFT'en skulle være mest effektiv, bør antallet af punkterne i sekvensen være en potens af 2, *dvs.* $N = 2^m$.

9.4 Et eksempel

For at belyse det ovenstående, kommer der her en gennemgang af et konkret eksempel. Der foreligger en datafil bestående af (x, h) koordinater, hvor x er afstanden fra startpunktet i nm af en nål, der føres hen over overfladen af en Ni-krystal, hvor der blev lagt Au på. Nålen går op og ned så den skøjter hen over atomerne, og h er højden af nålespidsen. Fig. 9.2 viser signalet, som består af 256 målepunkter, der strækker sig over en total afstand $X = 10.33$ nm. Dette er et signal fra et såkaldt *scanning tunnelling microscope* eller STM. Der er dog en vis usikkerhed i målingerne, og Fouriertransformationen kan nu benyttes til at finde de gennemsnitlige afstande mellem Ni-atomer og de påliggende Au-atomer. Figuren viser ret tydeligt, at der er to forskellige afstande mellem toppene. Vi kan se, at der er korte og længere bølger. Vi kan allerede nu få et skøn over afstandene ved at tælle antallet af bølger. Af de korte bølger er der 31, og af de lange er der ca. 5. Det giver afstandene (bølgelængderne) på henholdsvis 0.32 og 2 nm. Vi kommer til at forfine disse værdier. Problemet her



Figur 9.2: STM-signal, nålens højde h som funktion af afstand x

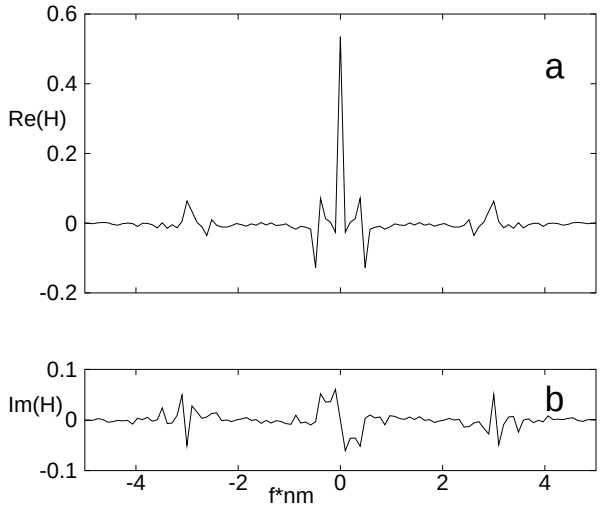
er, at afstandene fra top til top varierer, så vores skøn er ikke særlig nøjagtigt.

Vi har altså følgende oplysninger om signalet, som vi skal bruge:

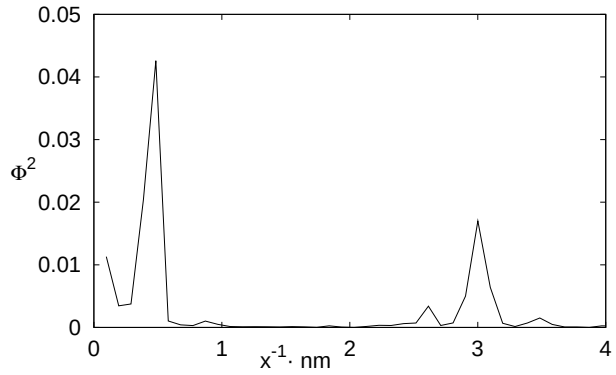
- $X = 10.33$ nm (den totale afstand)
- Det giver en minimum frekvens $\delta f = 1/X = 1/10.33$ nm^{-1}.
- Der er 256 punkter, og så er $\delta x = X/256 = 10.33/256$.
- Samplingfrekvens er $f_s = 1/\delta x = 256/10.33$ nm^{-1}.
- Den maksimale frekvens er $F_{max} = 1/(2\delta x) = 128/10.33$. Den svarer til bølgelængden af $2\delta x$.

Den gennemsnitlige afstand mellem atomerne vil give udslag i spektrummet, som gerne skulle have ret skarpe toppe ved de frekvenser (her i nm^{-1}) som er de inverse afstande. Vi tager derfor en Fouriertransformation af signalet, som først skal gemmes væk i et array af typen `complex`. Fig. 9.3 viser både den reelle og imaginære del af det resulterende spektrum. Den mindste frekvens er 10.33^{-1} og den største frekvens, med de 256 punkter vi har, er derfor 127.5/10.33 eller ca. 12.3. Der er dog ikke meget at se udenfor området $[-5, 5]$, og det er blevet udeladt.

En bemærkning om frekvensaksen er på plads her. Den er blevet *beregnet*, ud fra at $\delta f = 1/10.33$ og at N = 256. Det der bliver Fouriertransformeret er kun sekvensen af h-værdierne, og der ligger ingen information i spektrummet



Figur 9.3: Fourier-spektrummet H af signalet i Fig. 9.2. (a): den reelle komponente og (b), den imaginære.



Figur 9.4: Powerspektrummet beregnet fra spektrummet vist i Fig. 9.3

om frekvenserne. Spektrumssekvensen H er bare nummereret $0 \ldots 255$, og vi må selv oversætte disse til frekvenser (se også Fig. 9.1).

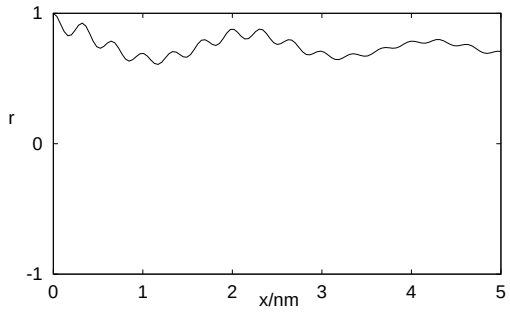
Vi bemærker, i den reelle del af spektrummet, en høj top ved $f = 0$. Den afspejler middelværdien af signalet h, som også er klart fra ligning (9.19). Vi skal fjerne denne værdi senere. Derudover ser vi to flere toppe, men de er ikke særlig tydelige; de ligger henholdsvis ved ca. 0.5 og 3, som svarer til afstandene ca. 2 og 0.33 nm.

Nu konverterer vi det komplekse spektrum til et powerspektrum, ved at bruge, for hvert element, formlen

$$\Phi^2(j) = |H(j)|^2 \tag{9.23}$$

hvilket giver en sekvens af power-værdier, vist i Fig. 9.4, hvor vi nu kun viser den halvdel af spektrummet, der svarer til positive frekvenser. Det nul'te element, som stadig indeholder kvadradet af middelværdien, er ikke vist. Nu ser vi ret tydelige toppe, hvis position vi kan måle. De ligger meget tæt på henholdsvis 0.5 og 3 nm^{-1}, som svarer til vores ovenstående skøn på afstandene 2.0 og 0.33 nm.

En sidste ting vi kan gøre, er at beregne autokorrelationsfunktionen. Signalet består jo af periodiske bølger, og de skulle afspejles i $r(\tau)$ som toppe som også er periodiske, med samme perioder. Man ved, at $r(0) = 1$, og vi kan derfor skalere hele r-sekvensen ved at dividere med $r(0)$. Vi igen beregner x-aksen, ud fra at $\delta x = 10.33/255$ og ellers nummereringen af sekvensen. Godt nok ser vi (Fig. 9.5) periodiciteten, men kurven ligger oppe ved 1, hvilket skyldes middelværdien, som ikke blev fjernet. Der bør jo være nogle negative r-værdier. Så renser vi powerspekrummet for middelværdien, ved at sætte $\Phi(0) = 0$. Fig. 9.6 viser resultatet. Nu bevæger r sig mellem -1 og +1, som det skal, og vi kan benytte figuren til igen at måle afstandene. Vi tæller 15 korte bølger over afstanden 5 nm, hvilket giver 0.33 nm; og af de lange bølger er der

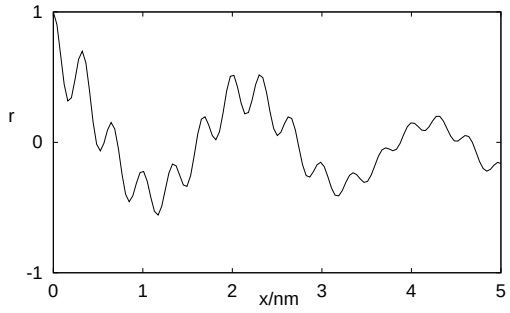


Figur 9.5: Autokorrelationsfunktion beregnet fra powerspektrummet vist i Fig. 9.4, uden nulstilling ved $f = 0$.

2 op til ca. $x = 4.2$, så de har længden 2.1 nm (der er dog lidt mere usikkerhed på den værdi).

9.5 Sampling og aliasing

Når man tager samples af et kontinuert signal, skal man være opmærksom på samplingfrekvensen f_s eller samplingintervallerne $\delta t = f_s^{-1}$. Det forklares lettest på følgende måde. Hvis signalet indeholder komponenter op til en vis højeste frekvens f_{max}, så skal sekvensen af samples kunne nogenlunde følge med denne komponente. Hvis man ikke sampler tilstrækkelig tit, så opstår der den mærkelige effekt kaldt **aliasing**. Fig. 9.7 viser den situation. Den trukne linje er en sinusbølge, og punkterne er samples af kurven, med intervallerne lidt større end bølgelængden. Det iøjnefaldene er, at punkterne danner en sinusbølge med en frekvens, der ikke har med kurvens frekvens at gøre. De repræsenterer slet



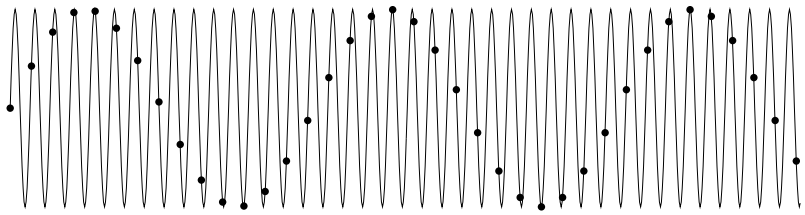
Figur 9.6: Autokorrelationsfunktion beregnet fra powerspektrummet vist i Fig. 9.4, efter nulstilling ved $f = 0$.

ikke kurven. Fig. 9.8 er derimod et eksempel på tilstrækkelig sampling, hvor der er flere samples for hver bølge. Hvis man trækker en kurve igennem disse punkter, repræsenterer den resulterende kurve den oprindelige rimelig godt.

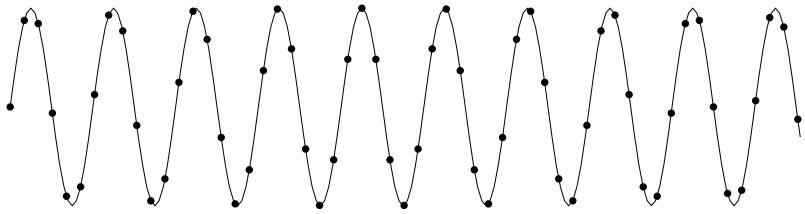
Betingelsen for tilstrækkelig sampling er, at der er mindst to samples per bølge for den højeste frekvens indeholdt i det samplede signal. Det vil sige,

$$f_s \geq 2f_{max} \ . \tag{9.24}$$

Dette kaldes **Nyquist-criteriet**. Der er to praktiske måder at opfylde criteriet. Den ene er først at finde ud af, hvad f_{max} er, og så at sætte f_s passende. Den anden, som tit opstår når man har en maksimum mulig samplingfrekvens eller vil begrænse antallet af samples er, at pille ved f_{max} med hjælp af filtrering af det målte signal. Det burde man i al fald gøre, for at sikre sig, at der ikke er nogle tilfældige højfrekvente komponenter som ville gøre samplingfrekvensen for lav. Filtrering er et omfattende tema, delvis behandlet i kap. 10.



Figur 9.7: Aliasing af en sinusbølge under utilstrækkelig sampling



Figur 9.8: Sinusbølge med tilstrækkelig sampling

Kapitel 10

Digital Signalbehandling

Under digitale signaler forstår vi sekvenser af punkter, der repræsenterer en underliggende kontinuerte funktion eller signal. Punkterne kan komme fra "sampling" af et fysisk signal eller fra en beregning. Da værdierne består af en serie af diskrete punkter, taler man om en diskret sekvens eller, hvis afstandene er i tid, en tisdserie (eng.: time series). Lige som der er fysiske metoder for at behandle elektriske signaler (filtrering, glatning, differentiering *mm.*), findes der digitale metoder for de samme operationer. Emnet falder under digital signalbehandling (eng.: digital signal processing). Der er mange lærebøger med det udtryk som titel, og der beskrives her nogle af de basale koncepter.

10.1 Filtrering

Ordet filtrering betyder i praksis oftest glatning af et diskret signal. Her beskrives der nogle af de gængse filtre. De er: middelværdi-, mindste-kvadraters- og rekursive filtre. Alle disse filtre falder teknisk set i de to store kategorier "finite impulse response" (FIR) eller "infinite impulse response" (IIR) filtre.

Lad os sige, vi har en sekvens af N værdier $s_i, i = 1 \ldots N$, og vil filtrere den til en ny sekvens $u_i, i = 1 \ldots N$, kaldt responsen. Generelt kan man udtrykke ethvert digitalt filter i formen

$$u_i = \sum_{k=i-m}^{i+m} a_k u_k + \sum_{k=i-m}^{i+m} b_k s_k \tag{10.1}$$

hvor m er et halv-vindue, dvs, vi filtrer med et "vindue" af bredden $w = 2m+1$. Hvis nogle af koefficienterne a_i er forskellige fra nul, har vi at gøre med et IIR filter, fordi man, for hver i, bruger u_k værdier, som er allerede nyberegnede filterresponser. Det fører med sig "IIR" fænomenet, som vil sige, at en given s_i værdi vil producere en respons ved alle filterpunkter u_i fremover. Hvis man for eksempel forestiller sig en sekvens af punkter, $1, 0, 0, \ldots 0$, og et filter som er ren IIR, *fx.*

$$u_i = 0.5u_{i-1} \quad (u_1 = s_1) \tag{10.2}$$

så vil man se at responsen er sekvensen $1, 0.5, 0.25, ...\left(\frac{1}{2}\right)^{N-1}$, og det er tydeligt, at det ene 1-tal kan "mærkes" i hele respons-sekvensen. Hvis derimod alle a_i er lige nul, så har vi et FIR filter. Den samme sekvens med filtret

$$u_i = s_{i-1} + s_i + s_{i+1} \quad (u_1 = s_1) \tag{10.3}$$

så er responsen $1, 1, 0, \ldots, 0$, *dvs.* at 1-tallet har ført til en respons der kun varer et enkelt interval.

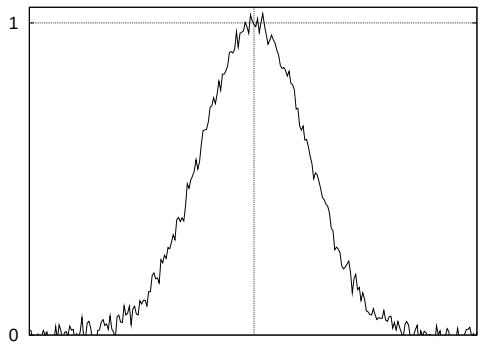
I det følgende beskrives der de to slags filtre, under tre grupper. Som eksempel tjener en sekvens af 256 punkter, set i Fig.10.1, en gaussisk curve belagt med gaussisk støj (det betyder, at selve støjet er normalfordelt).

Middelværdifilter

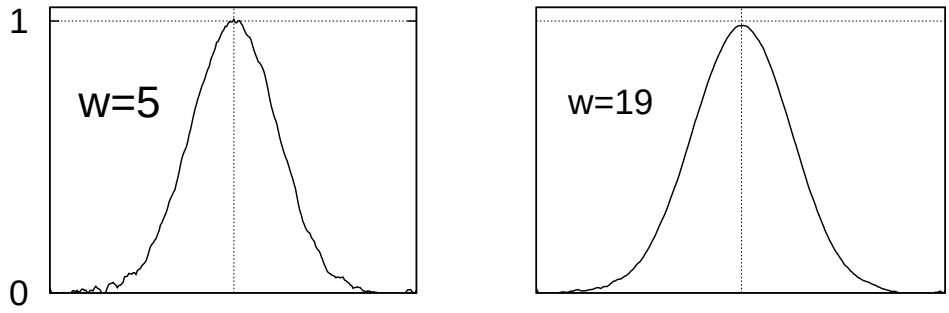
Et middelværdifilter går over sekvensen og for hvert punkt er responsen middelværdien af sekvensen taget over et vindue (bredden $w = 2m + 1$). Formlen er altså

$$u_i = \frac{1}{2m+1} \sum_{k=i-m}^{i+m} s_k \ . \tag{10.4}$$

Dette filter er en FIR type. I praksis er der et mindre problem ved enderne, hvor der ikke er punkter nok til at danne summen. Her begynder man med $m = 0$ for u_1, $m = 1$ for u_2 (3 punkter), og så fremdeles, indtil man har de ønskede w punkter. Når man nærmer sig enden af sekvensen, lader man vinduet atter skrumpe ind i modsat forstand. Fig. 10.2 viser resultatet af at filtrere kurven vist i Fig. 10.1, glattet med henholdsvis $w = 5$ og $w = 19$ (w er ulig). Dette filter er simpelt at programmere, men ikke særlig god. Den har den ulempe at presse signaltoppe ned, som kan ses i Figuren for $w = 19$.



Figur 10.1: Signal med støj på



Figur 10.2: Signal i Fig. 10.1 glattet med middleværdifilter

Mindste-kvadraters filtre

Årsagen til at middelværdifiltret trykker toppe ned er klart, at den ikke kan følge med kurven. Det kan derimod det filter, der er baseret på at man, for hvert vindue, tilpasser et mindre-kvadraters polynom, og lader den nye, respons-, værdi blive værdien ved midten af vinduet på polynomiet. Man tilpasser som regel parabler eller kubikfunktionen, fordi jo højere man går op i ordenen af polynomiet, jo mere ville den følge med de svingninger i s, som man jo gerne vil udglatte. Dette filter er også af typen FIR. Man behøver ikke beregne koefficienterne for hvert vindue, hvilket ville gøre filtret meget ueffektivt. Det undgår man idet man kan forudberegne et antal koefficienter b_k for et givet polynom. Et relativt simple eksempel viser, hvordan de beregnes.

Lad os sige, vi har 5 punkter i et vindue, og vi centrerer, for nemheds skyld, vinduet, så punkterne befinder sig ved $x_i, i = 1, \ldots 5$, og de er $-2, -1, 0, 1, 2$. Vi har ved disse positioner ukendte (altså, generelle) værdier $s_i, i = 1, \ldots, 5$. Vi vil tilpasse en parabel til de fem punkter, *dvs.* funktionen

$$u = a_0 + a_1 x + a_2 x^2 \ . \tag{10.5}$$

Mindste kvadrater er beskrevet i Kap. 6, og her får vi så de tre udtryk for koefficienterne

$$\begin{aligned} -2\sum_{i=1}^{5}\left(y_i - a_0 - a_1 x_i - a_2 x_i^2\right) &= 0 \\ -2\sum_{i=1}^{5}\left(y_i - a_0 - a_1 x_i - a_2 x_i^2\right)x_i &= 0 \\ -2\sum_{i=1}^{5}\left(y_i - a_0 - a_1 x_i - a_2 x_i^2\right)x_i^2 &= 0 \end{aligned} \tag{10.6}$$

som vi kan omskrive, ved at pille terme fra hinanden og omrokere lidt, til

$$\begin{aligned} a_0\sum_i 1 \; + a_1\sum_i x_i + a_2\sum_i x_i^2 &= \sum_i y_i \\ a_0\sum_i x_i + a_1\sum_i x_i^2 + a_2\sum_i x_i^3 &= \sum_i x_i y_i \\ a_0\sum_i x_i^2 + a_1\sum_i x_i^3 + a_2\sum_i x_i^4 &= \sum_i x_i^2 y_i \end{aligned} \tag{10.7}$$

(summerne er skrevet uden grænserne for overskuelighed). Vi kender alle x_i, og kan sætte dem ind i de venstre sider. Gør vi dette, får vi

$$\begin{aligned} 5a_0 \qquad\quad +10a_2 &= \sum_i y_i \\ 10a_1 \qquad\quad &= \sum_i x_i y_i \\ 10a_0 \qquad\quad +34a_2 &= \sum_i x_i^2 y_i \; . \end{aligned} \tag{10.8}$$

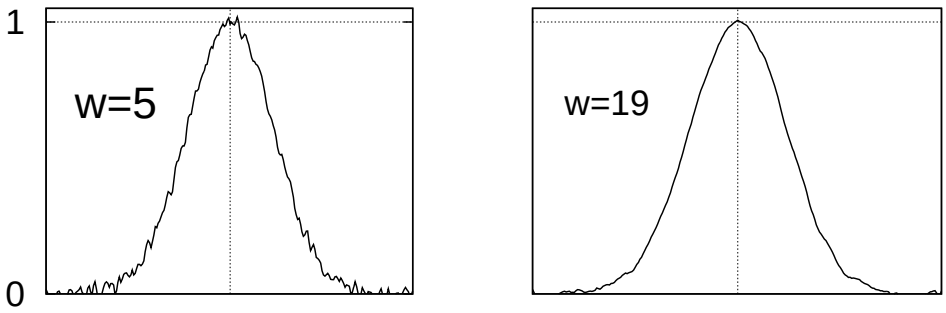
Vi er kun interesseret i den ene koefficient a_0, fordi vi vil kun have den ene værdi på parablen ved $x = 0$. Det er let at løse systemet for at få resultatet

$$a_0 = \frac{1}{35}\Big(-3y_1 + 12y_2 + 17y_3 + 12y_4 - 3y_5\Big) \; . \tag{10.9}$$

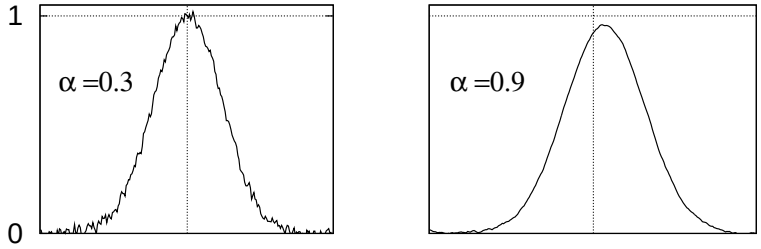
For at bruge dette som filter, skal vi bare gå over ethvert vindue og anvende disse koefficienter i filterformlen

$$u_i = \frac{1}{35}\Big(-3s_{i-2} + 12s_{i-1} + 17s_i + 12s_{i+1} - 3s_{i+2}\Big) \; . \tag{10.10}$$

Det tunge arbejde her var at beregne koefficiencerte, og hvis vi øger w, bliver det værre endnu. Alt det er blevet gjort og præsenteres i nogle store tabeller i den kemiske klassiker Savitsky og Golay [11]. Sådanne filtre er kaldt efter de to



Figur 10.3: Signal i Fig. 10.1 glattet med mindste-kvadraters-filter



Figur 10.4: Rekursiv IIR filter, med α som vist

forfattere, altså Savitsky-Golay filtre. Tabellerne indeholder også koefficienter for differentiering af sekvenser, se nedenfor.

Fig. 10.3 viser resultatet for en tilpasset parabel, med to vinduesbredder. Filtret virker ikke så meget bedre i glatningsforstand, men den presser ikke toppen ned, hvilket er en fordel.

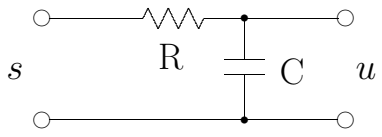
Rekursive (IIR) filtre

Som nævnt ovenpå, er et rekursiv eller IIR filter et, som har nogle af a-koefficienterne forskellige fra nul i den generel filterligning (10.1). Et eksempel er

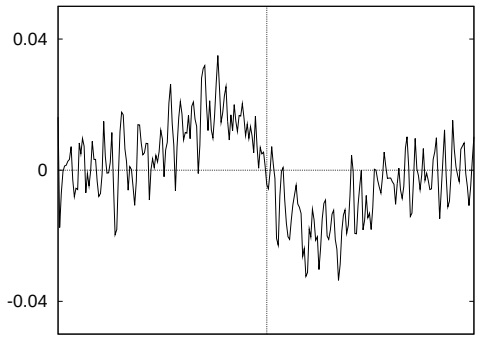
$$u_i = \alpha u_{i-1} + (1-\alpha)s_i \; . \tag{10.11}$$

Jo større α er, desto mere genbruger filteralgoritmen allerede beregende u-værdier.

Fig. 10.4 viser resultatet. Ved $\alpha = 0.3$ er der ikke meget glatning, men det er der for $\alpha = 0.9$. Til gengæld kan det tydeligt ses, at toppen er blevet flyttet til højre og formindsket. Filteret opfører sig som et analogfilter. Faktisk er dette filter en diskret approksimation til et såkaldt lowpass filter man kan realisere i elektronik, som vist i Fig. 10.5. Dette filter har en tidskonstant lige med RC, hvilket betyder at det har en skæringsfrekvens ω_c lige med $(RC)^{-1}$. Det er ved denne frekvens at signalets Fouriertransform begynder at dale nedad.



Figur 10.5: Analog low-pass filter



Figur 10.6: Fem-punkts mindste kvadraters differentierede

Frekvensen er relativ til samplingfrekvensen, her antaget som lige med 1. Vi vender tilbage til filtrenes frekvensspektra nedenfor.

Differentiering

Man kan også differentiere et diskret signal. En måde at gøre det på er at anvende et tilpassede polynom beskrevet i Sect. 10.1, for eksempel parablen (10.5). For et centreret sæt punkter behøver vi altså koefficienten a_1, og fra ligningssættet (10.8) får vi, for en fempunktsparabel,

$$u'_i = \frac{1}{10}\big(-2\,y_1 - y_2 + y_4 - 2\,y_5\big) \tag{10.12}$$

(læg mærke til at det midterste punkt indgår ikke i ligningen). For en simpel trepunktsapproksimation får vi

$$u'_i = \frac{1}{2}\big(-\,y_1 + y_3\big) \tag{10.13}$$

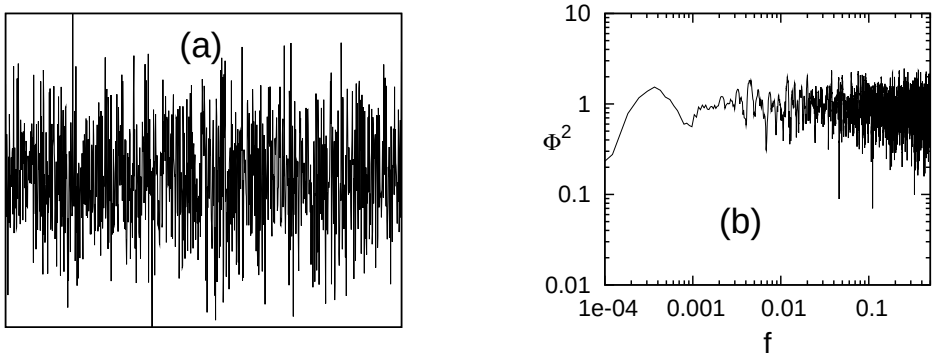
som kan opfattes som hældningen af en ret linje trukket mellem punkterne 1 og 3. Der har været instrumenter, som benytter en endnu mere simpel algoritme:

$$u'_i = -y_1 + y_2 \tag{10.14}$$

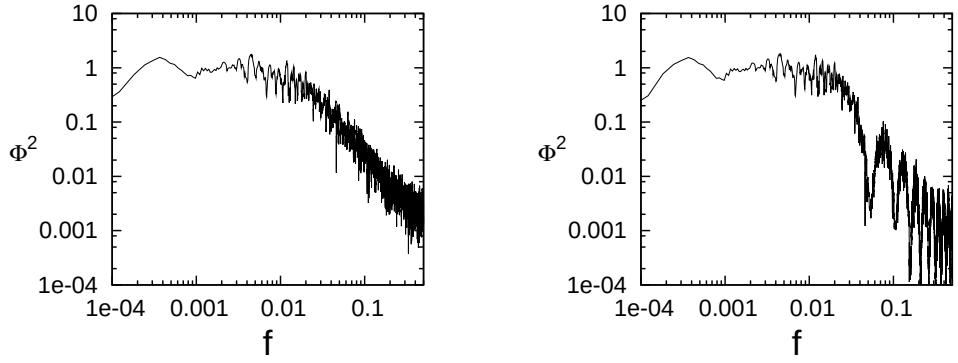
som jo også er en hældning. Den formel har den ulempe at den faktisk hører til punkterne midtvejs imellem eksisterende punkter, så sekvensen u' bliver forskudt om et halvt interval.

I Fig. 10.6 ses den differentierede kurve fra Fig. 10.1, hvor fempunktsparablen blev anvendt.

Artiklen af Savitsky og Golay indeholder også tabeller over koefficienter for multipunkt-differentiering.



Figur 10.7: Tilfældig støj (a) og dens powerspektrum (b)



Figur 10.8: Powerspektrum af (a) IIR-fitrerede tilfældig støj, $\alpha = 0.9$, og (b) FIR-filtrerede, 19-pkt parabel

10.2 Frekvensspektra eller filterrespons

Man er tit interesseret i at styre frekvens- eller powerspektrummet af et signal der skal filtreres, eller selve filtrets respons i frekvens-forstand. Man tegner filtrets respons som funktion af frekvens, og designer sig et filter der har den ønskede respons. Hvad responsen er, kan man finde ud af ud fra filtrets formel. Fig. 10.7(a) viser en sekvens af tal, bestående af tilfældig (gaussisk) fordelt støj. Det er ufiltreret, og derfor kaldes "hvid støj", og skulle have et jævnt powerspektrum. Fig. 10.7(b) viser så selve powerspektrummet af dette signal. Man plejer at præsentere det i en log-log-skala. Spektrummet er nogenlunde jævn, som det skal være. *Dvs*, at signalet indeholder nogenlunde ens intensiteter for alle mulige frekvenser. Når vi så filtrerer signalet, skulle vi kunne se effekten i spektrummet.

I Fig. 10.8(a) ser vi effekten af det simple IIR-filter (10.11) med α sat til 0.9. Man kan tydeligt se at spektrummet begynder at dale ved frekvensen ca. 0.02,

som svarer til en bølgelængde af ca. 50 sampling intervaller. Frekvensen hvor det begynder at dale er "knækfrekvensen". I Fig. 10.8(b) ser vi så resultatet af at filtrere signalet med et FIR-filter, hvor en parabel blev tilpasset 19 punkter ad gangen. Knækfrekvensen er nogenlunde den samme som for IIR-filtrets.

Man kan, ved hjælp af z-transformationsteori, som går ud over denne ramme, bestemme responsen for ethvert digitalt filter. For de to ovennævnte, det simple IIR (10.11) er resultatet at knækfrekvensen er ca. lige med $(1-\alpha)/2\pi\alpha$. I eksemplet var $\alpha = 0.9$, og det passer ret godt med knækfrekvensen 0.02, vi observerede fra Fig. 10.8(a). For en parabel over N punkter er knækfrekvensen ca. $0.5/N$.

Litteratur

[1] Løfstedt, Datamaskinens tal, Technical Report DAIMI FN-45, Daimi, Aarhus Universitet, 1990, kan skaffes fra DAIMI.

[2] O. Østerby, The error of the Crank-Nicolson method for linear parabolic equations with a derivative boundary condition, Report PB-534, DAIMI, Aarhus University, 1998, kan skaffes fra DAIMI.

[3] Abramowitz, Stegun (editors), Handbook of Mathematical Functions, Dover, New York, 1965, reprinted 1972; haves i KI bibliotek.

[4] Press, Teukolsky, Vetterling, Flannery, Numerical Recipes in Fortran, Cambridge University Press, Cambridge, 2 edition, 1986, haves i KI bibliotek.

[5] O. Østerby, Romberg integration, Report, DAIMI, Aarhus University, Nov. 17, 2005, kan skaffes fra DAIMI.

[6] Z. Kopal, Numerical Analysis, Chapman & Hall, London, 1955, haves i Statsbiblioteket.

[7] Fike, Computer Evaluations of Mathematical Functions, Prentice-Hall, NJ, 1968, haves i Statsbibliotek.

[8] Brigham, The Fast Fourier Transform, Prentice-Hall, New Jersey, USA, 1974, haves i KI bibliotek.

[9] Runge, Z. Math. Phys. 48 (1903) 433.

[10] Danielson, Lanczos, J. Franklin Inst. 233 (1942) 435.

[11] A. Savitzky, M. J. E. Golay, Anal. Chem. 36 (1964) 1627.