A photograph of a server room. The room is dark, with the primary light source being the green indicator lights on the server racks. On the left, a rack is filled with many green lights. In the center and right, other racks are visible, some with a few lights on. A white cable is visible in the upper left corner. The text 'Matematikken i computerens verden - computeren i matematikkens tjeneste' is overlaid in red on the right side of the image.

# Matematikken i computerens verden - computeren i matematikkens tjeneste

Det engelske ord *computer* betyder *beregner*, og computeren blev oprindelig udviklet netop til at foretage lange og komplekse beregninger, som den kan gøre hurtigere og mere pålideligt end noget menneske (og uden at brokke sig). Det viste sig imidlertid hurtigt, at computeren også er særdeles velegnet til mange andre opgaver, og i dag bruger vi computere til både underholdning og rekreation (f.eks. musik, film og spil), til lektier (stile, rapporter osv.) og til arbejde (f.eks. administration eller kommunikation). I dette kapitel vil vi se på computerens nutidige anvendelse til dens oprindelige formål: Som en meget avanceret regnemaskine og et uundværligt værktøj til at foretage matematiske beregninger samt store og komplicerede simuleringer med vigtige praktiske anvendelsesmuligheder i det moderne samfund.

### Hvorfor bruger vi computeren til matematik?

Som vi har set i andre kapitler i bogen, er matematikken udviklet igennem tusinder af år – mens computeren i dens nuværende form kun har stået til rådighed i de sidste ca. 50 år. Kan en sådan opkomling virkelig tilføre matematikken noget interessant? Svaret er selvfølgelig ”ja”, og det skyldes primært computerens evne til at foretage omfangsrige og komplekse *beregninger*, som det ellers ikke var muligt at gennemføre på anden vis. Mange af de beregninger, som ingeniører rutinemæssigt foretager, når de eksempelvis designer en bilmotor, en vindmølle eller en bro, kan faktisk kun udføres på en computer. Ingeniøren bruger computeren til at *simulere*, hvorledes motoren virker, så den kan udnytte brændstoffet optimalt, samtidig med at den forurener og støjer mindst muligt, eller til at simulere, hvordan en bro kan blive påvirket under en orkan, så man sikrer den bedst mulige konstruktion.

En anden grund til at bruge computeren til disse matematiske beregninger er, at vi – på trods af matematikkens fremskredne stadie – stadig ikke har udviklet præcise matematiske udtryk for mange af de problemer, vi ønsker at løse som delproblemer i vores simuleringer. Faktisk er der en uendelig mængde af problemer, som slet ikke kan løses med matematikkens formler. Eksempler er integralerne, som ikke kan udtrykkes med formler, samt løsningerne til ligninger af grad højere end 5, der heller ikke kan udtrykkes med formler. I stedet kan vi udvikle såkaldte *numeriske metoder* til computeren, som kan give løsningerne til disse problemer i form af tal og grafik.

Et eksempel på, hvordan vi anvender computeren til begge typer opgaver, er Google's Page Rank, som søgemaskinen bruger til at vælge den rækkefølge, hvormed søgeresultaterne vises. Google benytter en kæmpestor matematisk model af internettet som et netværk, hvor nettet udgøres af forbindelserne (links) mellem hjemmesiderne, og hvor nettets knudepunkter udgøres af samtlige hjemmesider på internettet. Netværket indeholder i dag mere end  $10^9$  knudepunkter (hjemmesider), og det vokser for hver dag. Beregning af Page Rank kræver, at man beregner en såkaldt egenvektor for dette kæmpestore

netværk. Imidlertid kan det bevises, at denne matematiske størrelse, egenvektoren, ikke kan udtrykkes med formler, når nettet har mere end 5 (fem!) knudepunkter. Google er altså nødt til at benytte en – hurtig – numerisk metode til at beregne denne egenvektor.

## Fra abacus til computer

Vi har altid brugt hjælpemidler til at udføre forskellige beregninger. Allerede for 3.000 år siden brugte man kuglerammer (abacus) i Kina og andre kulturkredse, og selv i dag kan man stadig se kuglerammer i brug i Asien. De matematiske operationer, man kan udføre på en abacus, er begrænsede, men det var først i det 17. århundrede, at der skete en yderligere udvikling, da regnestokken blev opfundet. Dette hjælpemiddel blev først afløst af lommeregneren i 1970'erne. Det var med udviklingen af elektro-mekaniske komponenter og behovet fra militæret, at der skete store gennembrud ikke mindst med opfindelsen af transistoren og senere integrerede kredsløb, altså muligheden for at bygge mange transistorer ind i en "chip".

Denne udvikling – miniaturisering – gjorde det muligt at bygge regnemaskiner i en størrelse, der er egnet til hverdagsbrug: lommeregneren. De første "lommeregner" kom på markedet i slutningen af 1960'erne, men resultaterne blev stadigvæk vist på en papirstrimmel. Først udviklingen af LED og senere LCD displayet gjorde det muligt at tale om en rigtig lommeregner, der faktisk kan ligge i lommen. Mens de første lommeregner på markedet var ret begrænsede i antallet af de indbyggede funktioner, er chippen i de moderne lommeregner så kraftig, at den kan udføre små programmer til at fortolke analytiske udtryk og vise dem grafisk.

## Opbygningen af en computer

Næsten alle computere er bygget efter det samme princip. Beregninger foregår i computerens CPU (Central Processing Unit), mens data og programkoderne bliver lagret i RAM (Random Access Memory). Tallene i en computer bliver repræsenteret af kombinationer af bits og bytes. En bit er 0 eller 1, og 8 bits udgør en byte. Ved hjælp af dette binære system kan vi gemme tal i computeren – men faktisk kun de hele tal! For at kunne repræsentere de reelle tal på computeren har man fundet en repræsentation, der gemmer tallene i to dele: en komma-del (kaldet mantisse) og en eksponent. Disse computer-tal kaldes *flydende tal*, og vi kender dem fra vores lommeregner, hvor  $1.2345E3$  betyder  $1,2345 \times 10^3 = 1,23 \times 1000 = 1234,5$ . I dette eksempel er 1.2345 mantissen, og 3 er 10-tals eksponenten. Princippet i computeren er det samme – bortset fra at den bruger 2-tal systemet. For at kunne regne med flydende tal skal computeren anvende specielle regler, der er forskellige fra de regler, vi bruger til at regne med hele tal. I de første pc'er, der kom på markedet i 1980'erne, var disse regneregler programmeret i software; mens alle moderne CPU'er har specielle hardwareenheder, der kan regne med flydende tal: de såkaldte Floating Point Units (FPU).

Computeren ved nu, hvordan den skal håndtere tal (også kaldt data), men nu skal disse

data også flyttes fra hukommelse (RAM) til processor (CPU) – og resultaterne skal gemmes igen i RAM. I en typisk pc vil hastigheden af beregningerne i CPU'en være mange gange hurtigere end den hastighed, hvormed data kan flyttes fra hukommelsen til CPU'en. Det betyder, at CPU'en skal vente det meste af tiden for at kunne udføre de næste beregninger. For at forkorte disse ventetider har alle moderne processorer indbygget et mellemlager (den såkaldte *cache*), der er ikke så stort, men hvor data kan tilgås meget hurtigere end fra det store datalager (RAM).

Cachens funktion kan sammenlignes med følgende situation: Ved middagsbordet beder far om en øl, og sønnen Børge bliver sendt ned i kælderen for at hente en flaske. Det tager Børge ca. 3 minutter at hente én flaske øl. Nu er far altid meget tørstig, så Børge har udviklet en 'smart' strategi: I stedet for at hente kun én flaske i kælderen, henter han så mange flasker, som han kan bære, f.eks. 4, og gemmer de 3 ekstra flasker i køkkenet. Næste gang far spørger efter en øl, tager det kun mindre end et minut at hente en ny flaske (vejen frem og tilbage i køkkenet). Kun når mellemlageret er tømt, skal Børge gå en tur i kælderen igen – eller når far pludselig beslutter sig for, at han skal have vin i stedet for øl.

På den samme måde fungerer cachen i en computer: Der bliver altid læst mere data ind i cachen, end der skal bruges, i forventningen om, at de tal, der blev hentet fra hukommelsen, skal bruges i de næste beregninger. Gode algoritmer og programmer vil derfor forsøge at udnytte dette for at kunne opnå en god performance, dvs. en hurtig afvikling af beregningerne.

## Algoritmer til computerberegninger

Men hvordan laver computeren (eller lommeregneren) en matematisk beregning, f.eks. kvadratroden af et tal? Her hjælper matematikken ikke rigtig – kvadratroden af 2 er jo simpelthen  $\sqrt{2}$ . Men vi har brug for at beregne dette tal med så mange cifre, som vi nu har på vores computer eller lommeregner. I gamle dage brugte man tabeller – i dag beregner vi faktisk kvadratroden hver gang vi har brug for den. Metoden til beregning af kvadratroden kaldes Herons metode (efter Heron fra Alexandria, ca. 10-70 e.Kr.). Vi starter med et gæt på kvadratroden og forbedrer gradvist vores gæt, indtil det er nøjagtigt nok. Lad os sige, at vi vil beregne  $\sqrt{2}$ . Vi starter med at gætte på tallet 1,5 ( $\sqrt{2}$  må jo ligge mellem 1 og 2). Vi kan nu beregne et bedre gæt som gennemsnittet af vores gæt og 2 divideret med gættet, altså  $\frac{1}{2}(1,5 + 2/1,5) = 1,4167$ . Næste gæt beregnes med samme teknik, og vi får  $\frac{1}{2}(1,4167 + 2/1,4167) = 1,4142$ . Nu har vi allerede – efter to forbedringer – fem korrekte cifre. Vi kunne fortsætte, men stopper her.

Den præcise beskrivelse af, hvorledes vi forbedrer vores gæt, er et eksempel på en algoritme, dvs. en opskrift for, hvad vi skal gøre (sammenlign med bageopskrifter). En god algoritme beregner en nøjagtig løsning hurtigt – altså med få trin.

Når vi skal have computeren til at beregne kvadratroden af et vilkårligt tal, må vi omsætte algoritmen til et computerprogram. Algoritmen ser således ud, når vi skal beregne kvadratroden af tallet a:

$x = \text{startgæt}$   
 gentag indtil  $x$  er nøjagtig nok  

$$x = (x + a/x) / 2$$
  
 slut

## OPGAVE

Prøv at bruge denne algoritme til at beregne kvadratroden af 3, 9 og 0,3.  
 Hvilken strategi vil du bruge til at stoppe processen (stopkriterium)? Kan du selv finde en god algoritme til at lave et godt startgæt for et vilkårligt positivt tal  $a$ ?

Selvom vi har mange cifre i vores lommeregner og endnu flere cifre i en stor computer, så må vi fortsat leve med et fast og begrænset antal cifre. Og ikke alle tal kan skrives med et begrænset antal cifre: Tallet  $1/3$  er ikke det samme som  $0,3333333333$  (selvom de to tal er ganske tæt på hinanden). Computeren kan altså ikke altid regne helt korrekt med sine flydende tal. Men skulle det dog betyde noget, at vi får disse ganske små fejl i vores beregninger, fordi vi ikke har uendelig mange cifre i vores computer? Desværre er svaret "ja". Når vi laver millionvis af beregninger, kan disse fejl godt hobe sig op og få betydning for resultaterne.

Men selv med ganske få beregninger kan vi få en stor fejl, og det er nemt at give et eksempel. Det er velkendt, at de to løsninger til 2. grads ligningen  $A X^2 + B X + C = 0$  (hvis de eksisterer) er givet ved

$$X = (-B \pm (B^2 - 4AC)^{1/2}) / (2A).$$

Lad os prøve at bruge en lommeregner (TI-89) til at beregne de to løsninger, når  $A = 1$ ,  $B = 87654321$  og  $C = -3$ . Lommeregneren beregner svarene

$$X_1 = -8.7654321E7 \text{ og } X_2 = 0.$$

De korrekte løsninger er imidlertid ca.  $-8.7654321E7$  og  $3.42254E-8$ . Den første løsning er altså beregnet rigtigt, men den anden er beregnet ganske forkert – formlen er matematisk korrekt, men ikke velegnet til beregner med flydende tal på en computer. Denne algoritme er meget bedre:

Hvis  $B > 0$

$$X_1 = (-B - (B^2 - 4AC)^{1/2}) / (2A)$$

ellers

$$X_1 = (-B + (B^2 - 4AC)^{1/2}) / (2A)$$

$$X_2 = C / (A X_1)$$

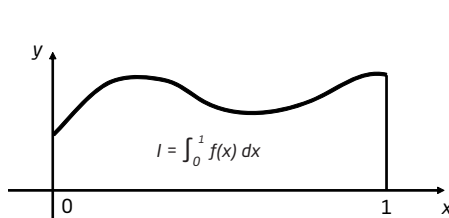
## OPGAVE

Prøv at bruge denne algoritme på problemet med  $A = 1$ ,  $B = 87654321$  og  $C = -3$ . Får du mere nøjagtige resultater?

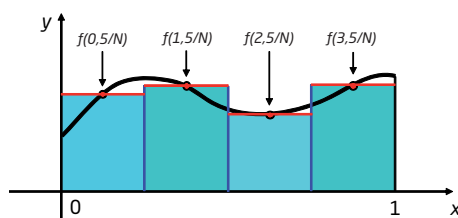
## Andre beregninger

Nu kan vi beregne kvadratrødder og løse 2. grads-ligninger. Det er et skridt på vejen ... men det rækker ikke så langt, hvis vi skal designe en motor eller en bro! I dette kapitel kommer vi naturligvis ikke frem til at beskrive matematikken og de numeriske metoder, der bruges til sådanne store og komplicerede designopgaver. Men vi vil fortælle om et par matematiske beregninger, som forekommer ganske ofte som dele af større beregningsopgaver.

Integraler er et godt eksempel. Vi kan bruge et *integral*, når vi skal beregne, hvor meget cigaretrøg vi indånder til en fest, eller hvor mange UV-stråler vi modtager, når vi soler os, eller hvor meget energi vi bruger på at køre fra punkt A til punkt B. Vi kender formelen for integralet af funktionen  $f(x)$ . Lad os antage, at vi vil beregne integralet over intervallet fra 0 til 1 (se figur 1):



Figur 1. Integralet  $I$  er arealet under grafen for  $f(x)$ .



Figur 2. Vi tilnærmer integralet  $I$  med arealet af kasserne.

Ved hjælp af vores formelsamling kan vi finde stamfunktionen til  $f(x)$  og bruge den til at beregne integralet. Men hvad nu hvis  $f(x)$  ikke har en stamfunktion (dette er faktisk tilfældet for en lang række funktioner), eller hvad hvis vi ikke kender formelen for  $f(x)$ , men i stedet har målt en række værdier for denne funktion? Husk, at hvis funktionen  $f(x)$  er positiv for alle  $x$  i integrationsintervallet, så er integralet  $I$  lig med arealet mellem  $x$ -aksen og grafen for  $f(x)$ . Det bruger vi som udgangspunkt! Vi vælger et heltal  $N$  og deler intervallet fra 0 til 1 op i  $N$  del-intervaller

$$[0, 1/N], [1/N, 2/N], [2/N, 3/N], \dots, [(N-1)/N, 1].$$

I hvert af disse små intervaller gør vi noget brutalt: Vi erstatter funktionen  $f(x)$  med den konstante funktion, som har værdien af  $f(x)$  i intervallets midtpunkt. I første interval bruger vi altså den konstante funktion med værdien  $f(0,5/N)$ , i næste interval den konstante funktion med værdien  $f(1,5/N)$  osv. (se figur 2). Vi har nu erstattet problemet at beregne integralet  $I$  med beregning af integralet af en funktion, som er stykkevis konstant i de  $N$  delintervaller. Det nye integral er supernemt at beregne: Vi skal blot beregne summen af arealerne (altså af kasserne i figur 2) i alle delintervallerne.

I matematisk notation skriver vi altså:

$$I_N = (f(0,5/N) + f(1,5/N) + \dots + f((N-0,5)/N)) / N$$

Tallet  $I_N$  kan bruges som en tilnærmelse til det integral  $I$ , vi egentlig ønsker at beregne, og formelen er ikke begrænset til positive funktioner. Det kan vises, at hvis  $N$  er et meget stort tal, så er  $I_N$  en god tilnærmelse til  $I$  (vores intuition siger os det samme).

## OPGAVE



Prøv at bruge formlen til at beregne integralet af  $\sin(x)$  i intervallet fra 0 til 1 for forskellige værdier af  $N$  (prøv f.eks.  $N = 1, 2, 4, 8$  og  $16$ ), og sammenlign med det korrekte resultat  $I = 1 - \cos(1) = 0,45969769413186$ . Du skulle gerne se, at fejlen  $I - I_N$  bliver mindre, når  $N$  bliver større.

## Flere integrationsformler

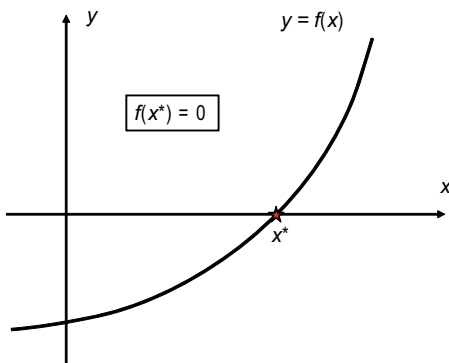
Formlen, vi lige har set ovenfor, er nem at forstå. Der findes mere nøjagtige formler, men de er sværere at udlede (det springer vi over her). Disse to formler giver mere nøjagtige resultater, når man bruger samme værdi af  $N$  til beregning af integralet fra 0 til 1:

$$J_N = ( \frac{1}{2} f(0) + f(1/N) + f(2/N) + \dots + f((N-1)/N) + \frac{1}{2} f(1) ) / N$$
$$K_N = ( f(0) + 4 f(1/N) + 2 f(2/N) + 4 f(3/N) + 2 f(4/N) + \dots + 4 f((N-1)/N) + f(1) ) / (3N)$$

NB: Den sidste formel kan kun bruges, når  $N$  er et lige tal. Vi beregner igen tilnærmelser til  $I$  fra før med forskellige værdier af  $N$ , og i tabellen herunder viser vi fejlene  $|I - J_N|$  og  $|I - K_N|$ . Bemærk, at den sidste formel (som hedder Simpsons formel) giver meget mere nøjagtige resultater for det samme beregningsarbejde!

$N$	2	4	8	16
$ I - J_N $	0,00962	0,002397	0,00059872	0,000149651
$ I - K_N $	0,00016	0,000010	0,00000062	0,000000039

Rødder er et andet eksempel på beregninger, som forekommer ganske ofte som dele af større beregningsopgaver. En rod er løsningen til ligningen  $f(x) = 0$  (se figur 3).



Figur 3. Roden  $x^*$  er løsningen til ligningen  $f(x) = 0$ .

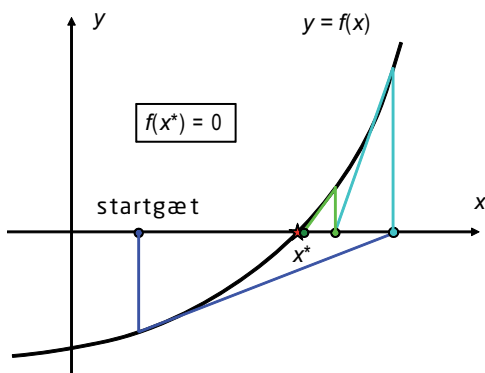
Hvis vi kaster en bold og vil ramme noget (eller nogen) i afstanden  $A$  fra os, så skal vi løse denne nulpunktligning med  $f(x)$  givet ved:

$$f(x) = V^2 \sin(x) \cos(x) - \frac{1}{2} g A ,$$

hvor  $V$  er hastigheden, vi kaster med,  $x$  er kastevinklen, og  $g = 9,8 \text{ m/s}^2$  er tyngdeaccelerationen (vi har ignoreret luftmodstand osv.). Hvis vi løser ligningen, kan vi altså beregne den vinkel, vi skal kaste, hvis vi ønsker at ramme. Godt at vide – men hvordan løser vi ligningen?

Præcis som ved beregningen af kvadratrødder skal vi bruge en algoritme, som, ud fra et startgæt, beregner bedre og bedre gæt på løsningen (beregning af kvadratroden er faktisk et specialtilfælde med  $f(x) = x^2 - a$ ). Vi vil beskrive en algoritme, som tilskrives Newton, og som forudsætter, at vi kan beregne både funktionen  $f(x)$  og dens første afledede  $f'(x)$ . Algoritmen tager denne form:

$x = \text{startgæt}$   
 gentag indtil  $x$  er nøjagtig nok  
 $h = -f(x) / f'(x)$   
 $x = x + h$   
 slut



Figur 4. Newtons metode. Efter tre forbedringer af vores startgæt er vi kommet tæt på roden  $x^*$ .

Det er nemmest at forklare denne algoritme ud fra figur 4. Når vi står i gættet  $x$ , så tager vi et skridt  $h$  hen til det næste gæt. Dette skridt får vi ved at gå ud langs tangenten i  $f(x)$ , indtil vi rammer  $x$ -aksen. Hvis vi startede med et gæt, som er tæt nok på roden, så kan vi bevise, at vi vil beregne bedre og bedre gæt!

## OPGAVE

Kan du finde frem til udtrykket  $h = -f(x) / f'(x)$  for skridtet ud fra figuren? Prøv at bruge Newtons algoritme på kasteprøbet med  $A = 35 \text{ m}$  og  $V = 22 \text{ m/s}$ . Vi oplyser, at den første afledede, er givet ved

$$f'(x) = V^2 (\cos^2(x) - \sin^2(x))$$

og det er vigtigt, at vinklerne udregnes i radianer i formlerne for  $f(x)$  og  $f'(x)$ . Brug startgættet  $x = 0$ . Hvor mange gange skal du forbedre gættet, før  $x$  er nøjagtig nok?

## Eksempel på computerkode

Der findes mange programmeringssprog til matematiske beregninger på computere. Fortran, C, C++, Java og Matlab er blot nogle få af de vigtigste. Her viser vi, hvordan Matlab-koden kan se ud for beregning af integral og rod:

```
function I = integral(f,a,b,N)
I = 0;
for i=1:N
    x = (i-0.5)/N;
    I = I + f(x);
end

function x = rod(f,fafl,x0)
x = x0; % Brugerens startgæt
h = 1;
while abs(h) > 1e-6 % Stop når skridtet er lille
    h = - feval(f,x)/feval(fafl,x);
    x = x + h;
end
```

## Hvad gør vi, når vi har data med målefejl?

Matematikken beskæftiger sig med eksakte og præcise formler og udtryk. Interessant nok giver matematikken os også et værktøj til at håndtere de fejl og den støj, som vi i praksis altid må leve med, når vi laver målinger. Matematikken fortæller os præcist, hvad vi skal gøre for at undertrykke fejlene og filtrere støjen, og hvor meget vores beregnede resultater bliver påvirket af fejl og støj.

### Filtrering

Du har lige optaget din guitarsolo – men desværre er der kommet noget støj med ind i mikrofonen. Kan vi fjerne denne støj med matematiske metoder? Sommetider! Musik gemmes på computeren som en lang række tal. Lad os sige, at vores lydsignal ideelt (dvs. uden støj) skulle være

122 129 137 144 151 158 164 171 177 182 187 192 196 200 203 ...

men pga. støjen har vi optaget dette signal:

134 117 121 156 157 138 180 171 179 186 197 206 192 184 213 ...

Kan vi beregne det støjfri signal? Nej – ikke fuldstændig – men vi kan slippe af med noget af støjen ved at *filtrere* det støjfyldte signal.

Her er algoritmen: Lav et nyt signal, hvor hvert tal beregnes som gennemsnittet af de 7 tal, der ligger omkring det gamle tal (tallet selv, de tre til venstre og de tre til højre). Altså, vi tager det fjerde tal, 156, og beregner gennemsnittet af de 7 tal, der ligger symmetrisk omkring 156, dvs. tallene

134 117 121 156 157 138 180 .

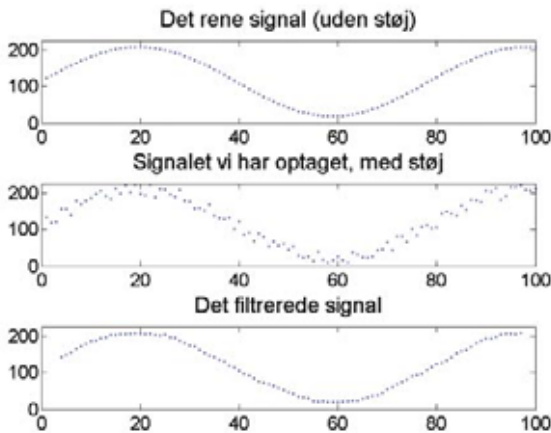
Gennemsnittet er  $(134+117+121+156+157+138+180)/7 = 143$  (afrundet til heltal), som er første tal i det filtrerede signal. Nu gentager vi processen med det næste tal, 157, og de 7 tal, der ligger omkring dette:

117 121 156 157 138 180 171 .

Gennemsnittet af disse tal er 149 (igen afrundet til heltal). Og sådan fortsætter vi, indtil vi er færdige, med alle tallene. Hvis  $x_1, x_2, x_3, \dots$  er tallene i det støjfyldte signal, og  $z_1, z_2, z_3, \dots$  er tallene i det filtrerede signal, så kan vores algoritme beskrives således:

$$z_i = (x_{i-3} + x_{i-2} + x_{i-1} + x_i + x_{i+1} + x_{i+2} + x_{i+3})/7, \quad i = 4, 5, 6, \dots$$

Bemærk, at vores filtrerede signal er en smule kortere end det oprindelige signal – vi mangler de første tre og sidste tre elementer. Figuren herunder viser, at vi faktisk får filtreret en god del af støjen!



Figur 5. Filtrering af et signal med støj.

### Unøjagtigheder

Unøjagtigheder i vores målinger har indflydelse på de resultater, vi beregner med vores matematiske algoritmer. Et interessant spørgsmål er, hvor meget disse fejl kan påvirke vores resultater. Igen giver matematikken os et værktøj til at lave denne analyse.

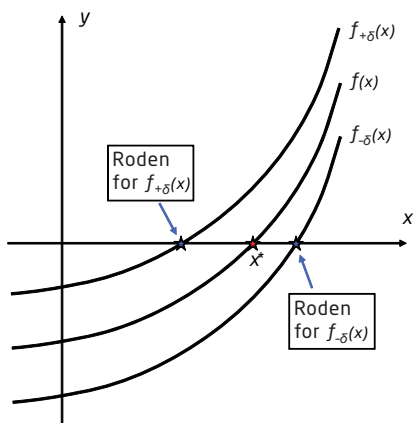
Som eksempel vender vi tilbage til kasteproblemet fra tidligere, og lad os sige, at vi ikke kender afstanden  $A$  præcist – hvad vi véd er, at afstanden er målt med en nøjagtighed på  $\delta = 10$  cm. Det skriver vi på formen  $A = 33,0 \text{ m} \pm 0,1 \text{ m}$ . Hvor stor indflydelse har usikkerheden  $\delta$  på den kastevinkel  $x$ , vi beregner?

Vi bliver nødt til at se på, hvordan funktionen  $f(x)$  ændres, når vi ændrer  $A$ . Hvis vi erstatter  $A$  med  $A \pm \delta$ , får vi to nye funktioner

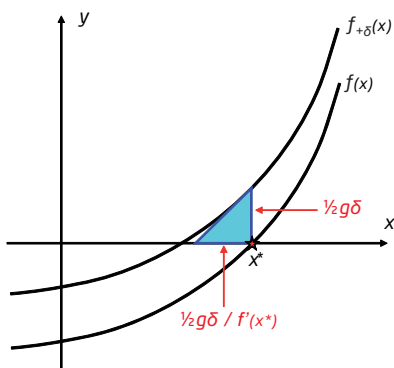
$$\begin{aligned} f_{+\delta}(x) &= f(x) + \frac{1}{2} g \delta, \\ f_{-\delta}(x) &= f(x) - \frac{1}{2} g \delta. \end{aligned}$$

Figur 6 viser graferne for  $f(x)$ ,  $f_{+\delta}(x)$  og  $f_{-\delta}(x)$  omkring roden  $x^*$ . Vi kan se, at når  $f(x)$  ændres til  $f_{+\delta}(x)$ , altså flyttes op, så flytter roden sig til venstre. Tilsvarende: Når  $f(x)$  ændres til  $f_{-\delta}(x)$ , flytter roden sig til højre.

Hvor meget flytter roden sig? Vi kan få et godt gæt ved at se på den trekant, der udgøres af  $x$ -aksen, en lodret linje i roden  $x^*$  og tangenten i  $f_{+\delta}(x^*)$ , se figur 7. Hældningen af denne tangent er  $f'_{+\delta}(x^*) = f'(x^*)$ , og tangenten skærer  $x$ -aksen i  $x = x^* - \frac{1}{2}g\delta/f'(x^*)$ . Roden har altså ca. flyttet sig stykket  $\frac{1}{2}g\delta/f'(x^*)$  til venstre. Helt tilsvarende ser vi, at roden for  $f_{-\delta}(x)$  ca. flytter sig  $\frac{1}{2}g\delta/f'(x^*)$  til højre. Usikkerheden på roden er altså ca.  $\pm \frac{1}{2}g\delta/f'(x^*)$ , og vi kan skrive usikkerheden på roden som  $x^* \pm \frac{1}{2}g\delta/f'(x^*)$ . Når vi indsætter tallene, får vi  $f'(x^*) = V^2(\cos^2(x) - \sin^2(x)) = 341,5$  og  $\frac{1}{2}g\delta/f'(x^*) = 0,000143$  radianer  $= 0,082^\circ$ .



Figur 6. Roden flytter sig, når vi ændrer  $f(x)$ .



Figur 7. Hvor meget har roden flyttet sig?

Vi kan naturligvis vende argumentet og sige, at hvis vi ændrer kastevinklen  $\pm\Delta x$ , så rammer vi inden for afstanden  $A \pm \Delta A$ .

## OPGAVE

Kan du vise, at sammenhængen mellem  $\Delta x$  og  $\Delta A$  er

$$\Delta A = 2\Delta x/g?$$

Hvor meget kan vi ca. risikere at ramme ved siden af, hvis  $\Delta x = 5^\circ = 0,0873$  radianer?

Kan du gennemføre en lignende analyse, hvis vi kender  $A$  præcist, men har usikkerheden  $V \pm \Delta V$  på kastehastigheden? Hjælp: Du skal finde ud af, hvordan  $f(x)$  ændrer sig, når du ændrer  $V$  til  $V \pm \Delta V$ .

## Store beregninger kræver store computere

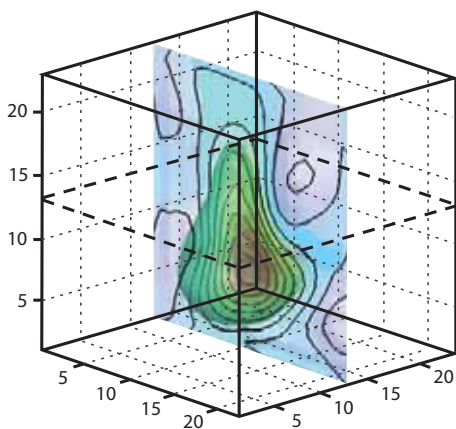
Hvis man skal løse store opgaver som en simulering af tusindvis af molekyler eller en beregning af luftstrømmen omkring en vindmøllevinge, løber man hurtigt ind i problemer. Computerens hukommelse er ikke stor nok, og selv med meget RAM tager det alt for lang tid at udføre beregningerne. Man kan ikke nøjes med bare én computer eller CPU til sådanne store opgaver. Enten kræves flere CPU'er i én stor computer (en såkaldt 'Symmetric Multi Processor' eller SMP computer) eller en klynge af små standard-computere med en eller flere CPU'er, der er forbundet med et hurtigt netværk.

Hardwareudviklingen og priserne har gjort klynger til de dominerede maskiner på ranglisten over verdens hurtigste computere, TOP500 listen ([www.top500.org](http://www.top500.org)). For 15-20 år siden var denne liste domineret af specielt udviklede maskiner. Fælles for alle store computersystemer er, at man ved hjælp af parallelle programmer deler problemet op i

mindre problemer, der kan løses på de enkelte processorer. Da de små problemer typisk ikke er uafhængige, skal man også bruge en del ressourcer til kommunikation mellem de enkelte beregninger. Denne kommunikation sker enten gennem udveksling af data i hukommelsen (SMP maskiner) eller via netværkskommunikation (klynge). Afhængig af problemstillingen vil der altid være problemer, der bedst kan løses på enten den ene eller den anden type af parallelcomputer.

Kunsten at skrive programmer til disse parallelcomputere består i at designe nogle koder, der udnytter den underliggende hardware bedst muligt, eksempelvis ved at minimere kommunikationen over netværket i en klynge.

### Eksempel med storskalaberegning



Figur 8. Tomografianalyse af et lille stykke metal.

På Risø DTU analyserer man, hvorledes metaller påvirkes, når man f.eks. trykker på dem eller trækker i dem. Metallet bestråles med røntgenstråler, og ved hjælp af målinger af disse kan man ”kigge ind” i metallet uden at skære det i stykker. Dette kaldes computertomografi, og det kræver store beregninger ofte med millioner af data. Figur 8 viser resultatet af en beregning på et lille stykke metal ud fra røntgen-data.

### OPGAVE: Tobaksrøg

Susanne holder fest for sine venner, og der bliver røget til festen. Lillebror Knud (som er en nörd) har lånt en røgmåler og har målt, hvor meget nikotin der er i luften på Susannes værelse under festen:

Tid	7 <sup>00</sup>	7 <sup>15</sup>	7 <sup>30</sup>	7 <sup>45</sup>	8 <sup>00</sup>	8 <sup>15</sup>	8 <sup>30</sup>	8 <sup>45</sup>	9 <sup>00</sup>	9 <sup>15</sup>	9 <sup>30</sup>	9 <sup>45</sup>	10 <sup>00</sup>	10 <sup>15</sup>	10 <sup>30</sup>	10 <sup>45</sup>	11 <sup>00</sup>	11 <sup>15</sup>	11 <sup>30</sup>	11 <sup>45</sup>	12 <sup>00</sup>
Nikotin	0	3	2	5	7	4	12	10	15	21	18	15	14	16	15	19	25	24	21	20	16

Nikotinkoncentration på Susannes værelse (målt i  $\mu\text{g}/\text{m}^3$ ) hvert kvarter.

Da man indånder ca.  $1 \text{ m}^3$  luft i timen, er den totale mængde af røgpartikler, som man indånder under Susannes fest, givet ved integralet

$$I_{\text{fest}} = \int_7^{12} f(x) dx$$

hvor  $x$  er tiden (målt i timer fra klokken 7 til kl. 12), og  $f(x)$  er røgen i lokalet. Kan du hjælpe lille Knud med at beregne dette integral? Hvor mange % røg indånder man mindre, hvis man går hjem klokken 11 i stedet for klokken 12?

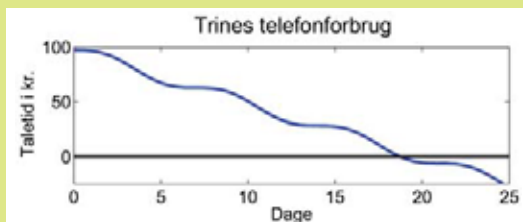
## OPGAVE: Taletid



Trines telefonforbrug er større sidst på ugen og i weekenden end på hverdage. Vi vil antage, at hun starter på dag 0 med en taletid på  $T = 97,40$  kr., og at hendes resterende taletid kan beskrives ved denne funktion

$$f(x) = T - a(bx - \sin(ax)), \quad a = 2\pi/7, \quad b = 5,50$$

hvor  $x$  er tiden målt i dage. Kan du bruge Newtons metode til at løse  $f(x) = 0$ , dvs. beregne den dag, hvor Trine løber tør for taletid?



Figur 9. Roden i ligningen  $f(x) = 0$  er antal dage, inden Trine løber tør for taletid.

## Artiklens forfattere



Lektor Bernd Dammann



Professor Per Christian Hansen