Forelesning III

IMAT2150: Høsten 2023

Kent Holing, NTNU (29. august 2023)

LÆRINGSMÅL for forrige forelesning

- Tap av signifikante sifre (S: 0.4)
- Halveringsmetoden (S: 1.1)
- Fikspunktiterasjon (FPI) (S: 1.2)

Tilbakeblikk på FORRIGE forelesning

Gitt ligningen $f(x) = 0$ med en rot r, dvs. $f(r) = 0$.

Omskriv ligningen med å velge $g(x)$ slik at $f(x) = x - g(x)$.

Da $f(r) = 0$, er $g(r) = r$.

Gitt ligningen $f(x) = 0$ med en rot r, dvs. $f(r) = 0$.

Omskriv ligningen med å velge $g(x)$ slik at $f(x) = x - g(x)$.

Da $f(r) = 0$, er $g(r) = r$.

M.a.o. er r en rot til f(x) = 0 og et fikspunkt for g(x).

Gitt ligningen $f(x) = 0$ med en rot r, dvs. $f(r) = 0$.

Omskriv ligningen med å velge $g(x)$ slik at $f(x) = x - g(x)$.

Da $f(r) = 0$, er $g(r) = r$.

M.a.o. er r en rot til f(x) = 0 og et fikspunkt for g(x).

Vi skulle bestemme en effektiv metode for å beregne kvadratrøtter.

La $s = \sqrt{a}$ for $a > 0$. Vi argumenterte som følger:

Gitt ligningen $f(x) = 0$ med en rot r, dvs. $f(r) = 0$.

Omskriv ligningen med å velge $g(x)$ slik at $f(x) = x - g(x)$.

Da $f(r) = 0$, er $g(r) = r$.

M.a.o. er r en rot til f(x) = 0 og et fikspunkt for g(x).

Vi skulle bestemme en effektiv metode for å beregne kvadratrøtter.

La $s = \sqrt{a}$ for $a > 0$. Vi argumenterte som følger:

Hvis a_n er en tilnærmelse for s, så ligger s mellom a_n og a/a_n.

Gitt ligningen $f(x) = 0$ med en rot r, dvs. $f(r) = 0$.

Omskriv ligningen med å velge $g(x)$ slik at $f(x) = x - g(x)$.

Da $f(r) = 0$, er $g(r) = r$.

M.a.o. er r en rot til f(x) = 0 og et fikspunkt for g(x).

Vi skulle bestemme en effektiv metode for å beregne kvadratrøtter.

La $s = \sqrt{a}$ for $a > 0$. Vi argumenterte som følger:

Hvis a_n er en tilnærmelse for s, så ligger s mellom a_n og a/a_n.

Dette følger av identiteten $a_n \times a/a_n = a$ da vi ser at faktorene ikke begge kan være større eller mindre enn $\sqrt{a}$.[1] Og, da vil middelverdien av disse tilnærmelsene forhåpentligvis være en enda bedre tilnærmelse til s.

[1] Vi ser at hvis a_n er eksakt lik $\sqrt{a}$, så er a/a_n også det, og omvendt. Og, hvis så, er også a_{n+1} eksakt lik $\sqrt{a}$.

Gitt ligningen $f(x) = 0$ med en rot r, dvs. $f(r) = 0$.

Omskriv ligningen med å velge $g(x)$ slik at $f(x) = x - g(x)$.

Da $f(r) = 0$, er $g(r) = r$.

M.a.o. er r en rot til f(x) = 0 og et fikspunkt for g(x).

Vi skulle bestemme en effektiv metode for å beregne kvadratrøtter.

La $s = \sqrt{a}$ for $a > 0$. Vi argumenterte som følger:

Hvis a_n er en tilnærmelse for s, så ligger s mellom a_n og a/a_n.

Dette følger av identiteten $a_n \times a/a_n = a$ da vi ser at faktorene ikke begge kan være større eller mindre enn $\sqrt{a}$.[1] Og, da vil middelverdien av disse tilnærmelsene forhåpentligvis være en enda bedre tilnærmelse til s.

Ligningen $f(x) = x^2 - a = 0$ for $a > 0$ er ekvivalent med ligningen $x = (x + a/x)/2$, så $a_{n+1} = (a_n + a/a_n)/2$, er en FPI-metode som kan tilnærme kvadratrøtter svært godt.

[1] Vi ser at hvis a_n er eksakt lik $\sqrt{a}$, så er a/a_n også det, og omvendt. Og, hvis så, er også a_{n+1} eksakt lik $\sqrt{a}$.

Gitt ligningen $f(x) = 0$ med en rot r, dvs. $f(r) = 0$.

Omskriv ligningen med å velge $g(x)$ slik at $f(x) = x - g(x)$.

Da $f(r) = 0$, er $g(r) = r$.

M.a.o. er r en rot til f(x) = 0 og et fikspunkt for g(x).

Vi skulle bestemme en effektiv metode for å beregne kvadratrøtter.

La $s = \sqrt{a}$ for $a > 0$. Vi argumenterte som følger:

Hvis a_n er en tilnærmelse for s, så ligger s mellom a_n og a/a_n.

Dette følger av identiteten $a_n \times a/a_n = a$ da vi ser at faktorene ikke begge kan være større eller mindre enn $\sqrt{a}$.[1] Og, da vil middelverdien av disse tilnærmelsene forhåpentligvis være en enda bedre tilnærmelse til s.

Ligningen $f(x) = x^2 - a = 0$ for $a > 0$ er ekvivalent med ligningen $x = (x + a/x)/2$, så $a_{n+1} = (a_n + a/a_n)/2$, er en FPI-metode som kan tilnærme kvadratrøtter svært godt.

Vi kommer tilbake til denne metoden når vi behandler Newtons iterasjonsmetode.

[1] Vi ser at hvis a_n er eksakt lik $\sqrt{a}$, så er a/a_n også det, og omvendt. Og, hvis så, er også a_{n+1} eksakt lik $\sqrt{a}$.

LÆRINGSMÅL for dagens forelesning

- Begrenset nøyaktighet (S: 1.3)
- Newtons metode (S: 1.4)
- Gausseliminasjon (innledning) (S: 2.1)
- LU faktorisering del 1 (S: 2.2)

Begrenset nøyaktighet [S:1.3]

Foroverfeil

Gitt en ligning $f(x) = 0$ med rot r og iterasjonsverdier $x_0, x_1, \cdots$ som tilnærmelser til r.

Da kalles $e_i = |x_i - r|$ foroverfeilen, som forteller hvor stor feil det er i den utregnede verdien x_i som tilnærmelse til roten r.

Foroverfeil

Gitt en ligning $f(x) = 0$ med rot r og iterasjonsverdier $x_0, x_1, \cdots$ som tilnærmelser til r.

Da kalles $e_i = |x_i - r|$ foroverfeilen, som forteller hvor stor feil det er i den utregnede verdien x_i som tilnærmelse til roten r.

Bakoverfeil

Dette er $|f(x_i) - f(r)| = |f(x_i)|$, som forteller hvor langt fra 0 $f(x_i)$ er.

INPUT (bakoverfeil) | **LØSNING** | **OUTPUT (foroverfeil)**

Foroverfeil

Gitt en ligning $f(x) = 0$ med rot r og iterasjonsverdier $x_0, x_1, \cdots$ som tilnærmelser til r.

Da kalles $e_i = |x_i - r|$ foroverfeilen, som forteller hvor stor feil det er i den utregnede verdien x_i som tilnærmelse til roten r.

Bakoverfeil

Dette er $|f(x_i) - f(r)| = |f(x_i)|$, som forteller hvor langt fra 0 $f(x_i)$ er.

INPUT (bakoverfeil) | **LØSNING** | **OUTPUT (foroverfeil)**

Fra SAUER:

> Backward error is a measure of error associated with an approximate solution to a problem. Whereas the forward error is the distance between the approximate and true solutions, the backward error is how much the data must be perturbed to produce the approximate solution.

[1] Ligningen har trippelroten $r = 2/3$.

Vi skal ved hjelp av halveringsmetoden løse ligningen

$$x^3 - 2x^2 + 4/3\,x - 8/27 = 0$$

med minst 6 korrekte signifikante sifre.

[1] Ligningen har trippelroten $r = 2/3$.

Vi skal ved hjelp av halveringsmetoden løse ligningen

$$x^3 - 2x^2 + 4/3\,x - 8/27 = 0$$

med minst 6 korrekte signifikante sifre.

Ligningen har en rot i $(0, 1)$,[1] og det skal være nok med 20 intervallhalveringer for å oppnå den ønskede nøyaktighet.

[1] Ligningen har trippelroten $r = 2/3$.

Vi skal ved hjelp av halveringsmetoden løse ligningen

$$x^3 - 2x^2 + 4/3\,x - 8/27 = 0$$

med minst 6 korrekte signifikante sifre.

Ligningen har en rot i $(0, 1)$,[1] og det skal være nok med 20 intervallhalveringer for å oppnå den ønskede nøyaktighet.

i	a_i	$f(a_i)$	c_i	$f(c_i)$	b_i	$f(b_i)$
0	0.0000000	−	0.5000000	−	1.0000000	+
1	0.5000000	−	0.7500000	+	1.0000000	+
2	0.5000000	−	0.6250000	−	0.7500000	+
3	0.6250000	−	0.6875000	+	0.7500000	+
4	0.6250000	−	0.6562500	−	0.6875000	+
5	0.6562500	−	0.6718750	+	0.6875000	+
6	0.6562500	−	0.6640625	−	0.6718750	+
7	0.6640625	−	0.6679688	+	0.6718750	+
8	0.6640625	−	0.6660156	−	0.6679688	+
9	0.6660156	−	0.6669922	+	0.6679688	+
10	0.6660156	−	0.6665039	−	0.6669922	+
11	0.6665039	−	0.6667480	+	0.6669922	+
12	0.6665039	−	0.6666260	−	0.6667480	+
13	0.6666260	−	0.6666870	+	0.6667480	+
14	0.6666260	−	0.6666565	−	0.666687	+
15	0.6666565	−	0.6666718	+	0.6666870	+
16	0.6666565	−	0.6666641	0	0.6666718	+

[1] Ligningen har trippelroten $r = 2/3$.

```
1  from scipy import optimize
2  def f(x):
3  #naive polynomial evaluations
4      return x**3-2*x**2+4/3*x - 8/27
5  def g(x):
6  #horner's method for polynomial evaluations
7      return ((x-2)*x+4/3)*x-8/27
8
9  rootf=optimize.bisect(f,0,1)
10 rootg=optimize.bisect(g,0,1)
```

```python
from scipy import optimize
def f(x):
#naive polynomial evaluations
    return x**3-2*x**2+4/3*x - 8/27
def g(x):
#horner's method for polynomial evaluations
    return ((x-2)*x+4/3)*x-8/27

rootf=optimize.bisect(f,0,1)
rootg=optimize.bisect(g,0,1)
```

```
root found using naive  polynomial evaluations: 0.6666641235351562
f(root)  0.0
root found using horner polynomial evaluations: 0.6666698455810547
g(root)  0.0

f(0.666662);  -1.1102230246251565e-16
g(0.666662);  -5.551115123125783e-17
f(0.666663);  0.0
g(0.666663);  -5.551115123125783e-17
f(0.666664);  0.0
g(0.666664);  -1.1102230246251565e-16
f(0.666665);  0.0
f(0.666665);  0.0
```

Vi har også at g(0.666665) = 0.0.

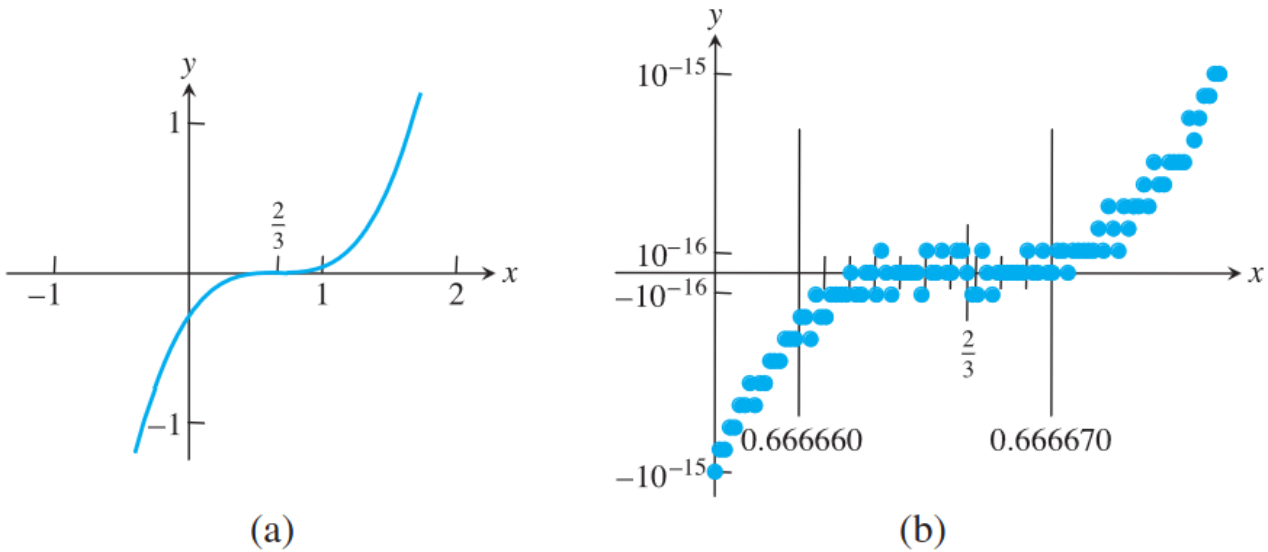


Figure 1.7 The shape of a function near a multiple root. (a) Plot of $f(x) = x^3 - 2x^2 + 4/3x - 8/27$. (b) Magnification of (a), near the root $r = 2/3$. There are many floating point numbers within 10^{-5} of 2/3 that are roots as far as the computer is concerned. We know from calculus that 2/3 is the only root.

Hva kan vi lese ut av eksemplet over?

Hva kan vi lese ut av eksemplet over?

Problemet er ikke halveringsmetoden (eller enn annen tilnærmingsmetode som er avhengig av å beregne verdier av $f(x)$) da det er begrenset hvor nøyaktig $f(x)$ kan beregnes i nærheten av roten r. (Jfr. figuren på forrige lysark.)

Denne begrensningen gjelder alle metoder som kun har kjennskap til f(x) under løsningen.

Hva kan vi lese ut av eksemplet over?

Problemet er ikke halveringsmetoden (eller enn annen tilnærmingsmetode som er avhengig av å beregne verdier av $f(x)$) da det er begrenset hvor nøyaktig $f(x)$ kan beregnes i nærheten av roten r. (Jfr. figuren på forrige lysark.)

Denne begrensningen gjelder alle metoder som kun har kjennskap til f(x) under løsningen.

Her er bakoverfeilen i nærheten av $\epsilon_M = 2^{-52} \approx 2.2\,10^{-16}$ (for IEEE754 DP flytall) mens foroverfeilen er i størrelsesorden 10^{-5}.

Og, da bakoverfeilen ikke kan reduseres ytterligere, kan heller ikke foroverfeilen det.

Merk at vi her må ha at $f'(r) \neq 0$! (Vi altså at r er en enkel rot.)

Sensitivity Formula for Roots

Assume that r is a root of $f(x)$ and $r + \Delta r$ is a root of $f(x) + \epsilon g(x)$. Then

$$\Delta r \approx -\frac{\epsilon g(r)}{f'(r)}$$

if $\epsilon \ll f'(r)$.

Feilforstørringsfaktor.

Feilforstørringsfaktor=relativ foroverfeil/relativ bakoverfeil

$$\textbf{Feilforstørringsfaktor} = \left|\frac{\Delta r/r}{\epsilon}\right| = \frac{|g(r)|}{|r\,f'(r)|}.$$

Wilkinson-polynomet

Wilkinson-polynomet:
$W(x) = (x-20)(x-19)(x-18)(x-17)\cdots(x-2)(x-1)$ har røtter lik $1, 2, 3, \ldots, 19, 20$. Om vi ekspanderer polynomet får vi

$$\begin{aligned} W(x) = x^{20} &- 210\,x^{19} + 20615\,x^{18} - 1256850\,x^{17} + 53327946\,x^{16} \\ &- 1672280820\,x^{15} + 40171771630\,x^{14} - 756111184500\,x^{13} \\ &+ 11310276995381\,x^{12} - 135585182899530\,x^{11} + 1307535010540395\,x^{10} \\ &- 10142299865511450\,x^{9} + 63030812099294896\,x^{8} \\ &- 311333643161390640\,x^{7} + 1206647803780373360\,x^{6} \\ &- 3599979517947607200\,x^{5} + 8037811822645051776\,x^{4} \\ &- 12870931245150988800\,x^{3} + 13803759753640704000\,x^{2} \\ &- 8752948036761600000\,x + 2432902008176640000 \end{aligned} \quad (1)$$

Vi har for eksempel $W(20) = 0$, men hvilken verdi får vi i dobbel presisjon på en datamaskin?

$$W(20) \approx -2{,}7 \cdot 10^{10}$$

Hvorfor?

NB! Brukes Python til å beregne W(20), så er det er viktig at koeffisientene til polynomet er definert som floats. (Hvis ikke får vi "integer overflow".)

Feilen i Wilkinsonpolynomet

Feilen i koeffisientene til Wilkinson-polynomet når de representeres som fyttall med dobbel presisjon er

$$\begin{aligned} c_3 - fl(c_3) &= -512, \\ c_4 - fl(c_4) &= 384, \\ c_5 - fl(c_5) &= -160, \\ c_6 - fl(c_6) &= 112, \text{ og} \\ c_7 - fl(c_7) &= 16. \end{aligned}$$

I ekspandert form blir $W(x)$ lik

$$W(x) - 16x^7 - 112x^6 + 160x^5 - 384x^4 + 512x^3$$

på grunn av avrundinger av koeffisientene.

Det er katastrofalt.

Merk at bidraget fra de andre polynomkoeffisientene i pertubasjonen $\epsilon g(x)$ av $W(x)$ er neglisjerbart.[1]

[1] Disse koeffisientene er nøyaktig (eller eksakt) representert som flyttall i IEEE754 DP. I Python gir int(cj) - int(float(cj)) differansene ovenfor. Ellers kan https://www.binaryconvert.com brukes (referert til tidligere).

Eksempel:

Feilforsterknings/forstørringsfaktor for W(20)

Feilforstørringsfaktor=relativ foroverfeil/relativ bakoverfeil

$$\textbf{Feilforstørringsfaktor} = \left|\frac{\Delta r/r}{\epsilon}\right| = \frac{|g(r)|}{|r\,f'(r)|}.$$

Eksempel (Wilkinson $r = 20$). $W'(20) = 19! \approx 1.2165 \cdot 10^{17}$
$\epsilon g(20) \approx -2{,}7 \cdot 10^{10}$.

$$\Delta r \approx -\frac{-2{,}7\cdot 10^{10}}{1.2165\cdot 10^{17}} = 2.2355 \cdot 10^{-07}$$

Feilforstørringsfaktor: $\left|\frac{\Delta r/r}{\epsilon}\right| = 5.0338 \cdot 10^{7}$.

Fra lysark 13 har vi $W(20) \approx -2.7 \times 10^{10}$.
Husk at her er $\Delta r = -\epsilon g(20)/W'(20)$.
$FE = |\Delta r|$, $RFE = FE/r = FE/20$, $BE = |\epsilon g(20)|$
$RBE = BE/|g(20)| = \epsilon = \epsilon_M = 2.2204 \times 10^{-16}$
Feilforsterkningsfaktoren:
$EMF = RFE/RBE = 2.2355 \times 10^{-7}/(20 \times 2.2204 \times 10^{-16}) = 5.0338 \times 10^{7}$ som ovenfor.

Formelen ovenfor er viktig, men vi utelater beviset.

Kondisjon

Kondisjonsnummeret = største feilforstørringsfaktor

Vel-kondisjonert : Kondisjonsnummer nær 1.

Dårlig kondisjonert: Stort kondisjonsnummer.

I Wilkinsonpolynomet for $r = 20$ så vi at forstørrelsesfaktoren EMF var $5.0338 \times 10^7 \gg 1$.

WP-roten 20 er altså dårlig kondisjonert.

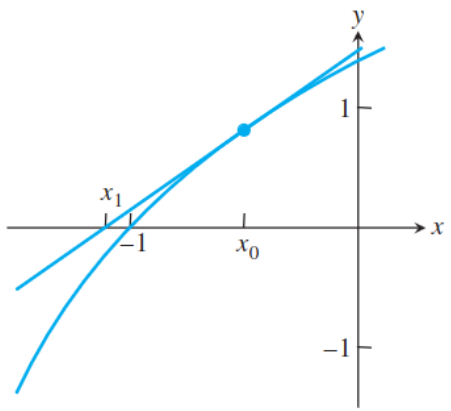
(Python ga $W(20) \approx -2.7 \times 10^{-10}$.)

EMF sier noe om hvor mange signifikante sifre som mistes i prosessen fra INPUT til OUTPUT for et problem.

NEWTONS METODE [S:1.4]

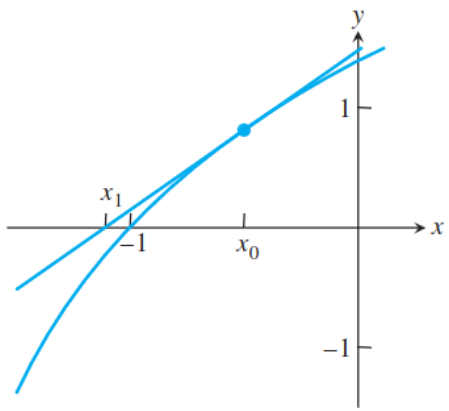
Denne metoden skal være kjent fra "calculus" tidligere.

Metoden kan enkelt illustreres geometrisk.



Denne metoden skal være kjent fra "calculus" tidligere.

Metoden kan enkelt illustreres geometrisk.



Da vinkelkoeffisienten til tangenten i et punkt med abscisse x_0 på grafen til $f(x) = 0$ er $f'(x_0)$, får en lett følgende algebraisk form for Newtons metode:

$$x_{i+1} = x_i - f(x_i)/f'(x_i)$$

for en startverdi x_0 og $i = 0, 1, \cdots$.

Vi observerer at Newtons metode er en fikspunktiterasjon, og da vet vi allerede mye om egenskapene til metoden.

Fikspunktiterasjonen er definert ved med $g(x) = x - f(x)/f'(x)$.

Newtons metode er effektiv når den først konvergerer, noe den ikke alltid gjør (i likhet med andre FPI metoder).

Vi observerer at Newtons metode er en fikspunktiterasjon, og da vet vi allerede mye om egenskapene til metoden.

Fikspunktiterasjonen er definert ved med $g(x) = x - f(x)/f'(x)$.

Newtons metode er effektiv når den først konvergerer, noe den ikke alltid gjør (i likhet med andre FPI metoder).

Det som gjør at metoden kan være så effektiv er at $g'(r) = 0$ for en rot r av ligningen $f(x) = 0$. (Vises nedenfor.)

Vi observerer at Newtons metode er en fikspunktiterasjon, og da vet vi allerede mye om egenskapene til metoden.

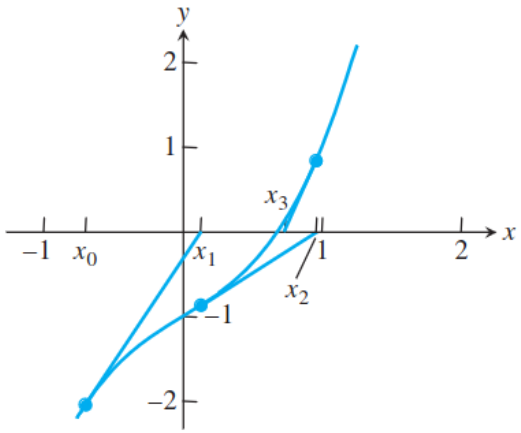
Fikspunktiterasjonen er definert ved med $g(x) = x - f(x)/f'(x)$.

Newtons metode er effektiv når den først konvergerer, noe den ikke alltid gjør (i likhet med andre FPI metoder).

Det som gjør at metoden kan være så effektiv er at $g'(r) = 0$ for en rot r av ligningen $f(x) = 0$. (Vises nedenfor.)

Effektiviteten til metoden illustreres tydelig med eksemplet $x^3 + x - 1 = 0$:

i	x_i	$e_i = \lvert x_i - r \rvert$	e_i / e_{i-1}^2
0	−0.70000000	1.38232780	
1	0.12712551	0.55520230	0.2906
2	0.95767812	0.27535032	0.8933
3	0.73482779	0.05249999	0.6924
4	0.68459177	0.00226397	0.8214
5	0.68233217	0.00000437	0.8527
6	0.68232780	0.00000000	0.8541
7	0.68232780	0.00000000	



Vi utelater beviset av følgende viktige teorem.

TEOREM (teorem 1.11, s. 53 i SAUER)

La $f(x)$ være to ganger kontinuerlig deriverbar og $f(r) = 0$. Hvis $f'(r) \neq 0$, så er Newtons metode lokalt kvadratisk konvergent, dvs. feilen $e_i = |x_i - r|$ i iterasjon i oppfyller $e_{i+1}/e_i^2 \to M < \infty$ der $M = |f''(r)/(2f'(r))|$.

M for eksemplet ovenfor er 0.85 ($f(x) = x^3 + x - 1$, $r \approx 0.6823$), som er i overenstemmelse med tabellen.

Vi utelater beviset av følgende viktige teorem.

TEOREM (teorem 1.11, s. 53 i SAUER)

La $f(x)$ være to ganger kontinuerlig deriverbar og $f(r) = 0$. Hvis $f'(r) \neq 0$, så er Newtons metode lokalt kvadratisk konvergent, dvs. feilen $e_i = |x_i - r|$ i iterasjon i oppfyller $e_{i+1}/e_i^2 \to M < \infty$ der $M = |f''(r)/(2f'(r))|$.

M for eksemplet ovenfor er 0.85 ($f(x) = x^3 + x - 1$, $r \approx 0.6823$), som er i overenstemmelse med tabellen.

Beviset for teoremet bygger på at $g'(x) = f(x)\, f''(x)/f'(x)^2$ ($g(x) = x - f(x)/f'(x)$) (vis det!) og derfor at $g'(r) = 0$.

Vi utelater beviset av følgende viktige teorem.

TEOREM (teorem 1.11, s. 53 i SAUER)

La $f(x)$ være to ganger kontinuerlig deriverbar og $f(r) = 0$. Hvis $f'(r) \neq 0$, så er Newtons metode lokalt kvadratisk konvergent, dvs. feilen $e_i = |x_i - r|$ i iterasjon i oppfyller $e_{i+1}/e_i^2 \to M < \infty$ der $M = |f''(r)/(2f'(r))|$.

M for eksemplet ovenfor er 0.85 ($f(x) = x^3 + x - 1$, $r \approx 0.6823$), som er i overenstemmelse med tabellen.

Beviset for teoremet bygger på at $g'(x) = f(x)\, f''(x)/f'(x)^2$ ($g(x) = x - f(x)/f'(x)$) (vis det!) og derfor at $g'(r) = 0$.

Da vet vi fra tidligere at Newtons metode er lokalt konvergent. (Metoden er jo en FPI metode.) Men metoden kan som sagt i tilfeller som oppfyller teoremet konvergere kvadratisk.

Sammenlign $e_{i+1}/e_i^2 \approx M < \infty$ med $e_{i+1} \approx S\, e_i$ for en lineært konvergent metode. For sistnevnte er betingelsen at $S < 1$ kritisk for konvergens mens størrelsen på M ikke er det så lenge $M\, \epsilon_0 < 1$. (Hvorfor?)

Kvadratrotutdragninger med Newtons metode

Tidligere viste vi en effektiv metode til å beregne kvadratroten til 2 på.

Kvadratrotutdragninger med Newtons metode

Tidligere viste vi en effektiv metode til å beregne kvadratroten til 2 på.

Newtons metode for å bestemme roten $r = \sqrt{a}$ for ligningen $f(x) = x^2 - a$ for $a > 0$ gir iterasjonene

$$x_{i+1} = x_i - (x_i^2 - a)/(2x_i) = (x_i + a/x_i)/2.$$

Så, vi ser at Newtons metode er den samme som den tidligere metoden vi ga.

Konvergensen blir kvadratisk da ligningen har to enkle røtter $\pm\sqrt{a}$.

M i teoremet for $e_{i+1} \approx M\, e_i^2$ er med $M = |f''(r)/(2f'(r)) = 1/(2\sqrt{a})$ da $r = \sqrt{a}$.[1]

[1] For $a = 2$, er $M = 1/2\sqrt{2} \approx 0.35$.

Feil/problemsituasjoner med Newtons metode

I tilfeller der Newtons metode feiler eller ikke oppnår kvadratisk konvergens, så skyldes det ofte at ikke alle antagelsene i teorem 1.11 (nevnt ovenfor) er oppfylt.

Eksempler på dette kan være $f'(x_j) = 0$ for en iterasjon j (som gir divisjon med 0) eller at $f'(r) = 0$ (som signaliserer at ligningen $f(x) = 0$ har en multippel rot).

Feil/problemsituasjoner med Newtons metode

I tilfeller der Newtons metode feiler eller ikke oppnår kvadratisk konvergens, så skyldes det ofte at ikke alle antagelsene i teorem 1.11 (nevnt ovenfor) er oppfylt.

Eksempler på dette kan være $f'(x_j) = 0$ for en iterasjon j (som gir divisjon med 0) eller at $f'(r) = 0$ (som signaliserer at ligningen $f(x) = 0$ har en multippel rot).

Siden konvergensen er lokal kan det også hende at metoden feiler eller må bruke veldig mange iterasjoner før konvergens etableres. Dette kan skje hvis startverdien for metoden er for dårlig.

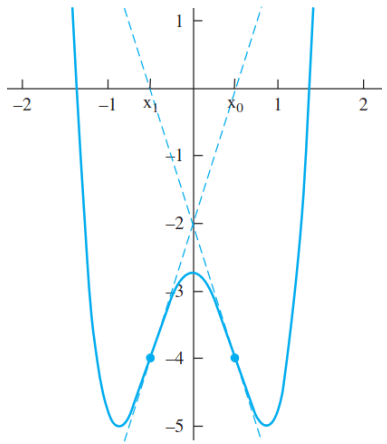
Feil/problemsituasjoner med Newtons metode

I tilfeller der Newtons metode feiler eller ikke oppnår kvadratisk konvergens, så skyldes det ofte at ikke alle antagelsene i teorem 1.11 (nevnt ovenfor) er oppfylt.

Eksempler på dette kan være $f'(x_j) = 0$ for en iterasjon j (som gir divisjon med 0) eller at $f'(r) = 0$ (som signaliserer at ligningen $f(x) = 0$ har en multippel rot).

Siden konvergensen er lokal kan det også hende at metoden feiler eller må bruke veldig mange iterasjoner før konvergens etableres. Dette kan skje hvis startverdien for metoden er for dårlig.

Newton iterasjoner kan også alternere mellom to verdier uten å konvergere, som for $f(x) = 4x^4 - 6x^2 - 11/4 = 0$ med $x_0 = 1/2$ der vi alternerer mellom verdiene $\pm 1/2$:[1]



[1] SAUER, eksempel 1.15, s. 58.

Lineær konvergens med Newtons metode

Når ligningen $f(x) = 0$ har en multippel rot r (dvs. $f(r) = 0$ og $f'(r) = 0$) konvergerer Newtons metode lineært. (Vi viser ikke de to neste teoremene.)

TEOREM

Anta at $f(x)$ er $m+1$ ganger kontinuerlig deriverbar med r en multippel rot av multiplisitet $m > 1$.[1]

Da er Newtons metode lokalt lineært konvergent der $e_{i+1}/e_i \to S = (m-1)/m$ når $i \to \infty$.

[1] Dvs. at $f(r) = f'(r) = f''(r) \cdots f^{(m-1)}(r) = 0$, men $f^{(m)}(r) \neq 0$ der $f^{(j)}(x)$ den j-te deriverte av $f(x)$.

Lineær konvergens med Newtons metode

Når ligningen $f(x) = 0$ har en multippel rot r (dvs. $f(r) = 0$ og $f'(r) = 0$) konvergerer Newtons metode lineært. (Vi viser ikke de to neste teoremene.)

TEOREM

Anta at $f(x)$ er $m+1$ ganger kontinuerlig deriverbar med r en multippel rot av multiplisitet $m > 1$.[1]

Da er Newtons metode lokalt lineært konvergent der $e_{i+1}/e_i \to S = (m-1)/m$ når $i \to \infty$.

Modifisert Newtons metode (for tilfeller med multiple røtter)

TEOREM

La f(x) være som i teoremet ovenfor.

Da konvergerer $x_{i+1} = x_i - m\, f(x_i)/f'(x_i)$, $i = 0, 1 \cdots$ lokalt og kvadratisk mot roten r.

[1] Dvs. at $f(r) = f'(r) = f''(r) \cdots f^{(m-1)}(r) = 0$, men $f^{(m)}(r) \neq 0$ der $f^{(j)}(x)$ den j-te deriverte av $f(x)$.

IMAT2150: Newtons metode- Eksempel med trippel rot

Vi skal se på Newtons metode for $f(x) = \sin x + x^2 \cos x - x^2 - x = 0$ med trippelroten $r = 0$.[1]

[1] Dette er eksempel 1.14 i SAUER, 56.

IMAT2150: Newtons metode- Eksempel med trippel rot

Vi skal se på Newtons metode for $f(x) = \sin x + x^2 \cos x - x^2 - x = 0$ med trippelroten $r = 0$.[1]

For å oppnå konvergens til r korrekt inntil 6 desimaler, må vi bruke omtrent 35 iterasjoner når $x_0 = 1$.[2]

[1] Dette er eksempel 1.14 i SAUER, 56.

[2] Her er $S = (m-1)/m$ for $m = 3$, dvs. $S = 2/3$. Ønsket nøaktighet oppnås når antall iterasjoner n oppfyller $(2/3)^n < 10^{-6}$, dvs. $n > 37.8$.

IMAT2150: Newtons metode- Eksempel med trippel rot

Vi skal se på Newtons metode for $f(x) = \sin x + x^2 \cos x - x^2 - x = 0$ med trippelroten $r = 0$.[1]

For å oppnå konvergens til r korrekt inntil 6 desimaler, må vi bruke omtrent 35 iterasjoner når $x_0 = 1$.[2]

Siden vi har en trippelrot, bruker vi modifisert Newtons metode for $m = 3$ og $x_0 = 1$, som gir

i	x_i
0	1.00000000000000
1	0.16477071958224
2	0.01620733771144
3	0.00024654143774
4	0.00000006072272
5	−0.00000000633250

med svært stor økning i konvergensraten, som er kvadratisk.

[1] Dette er eksempel 1.14 i SAUER, 56.

[2] Her er $S = (m-1)/m$ for $m = 3$, dvs. $S = 2/3$. Ønsket nøaktighet oppnås når antall iterasjoner n oppfyller $(2/3)^n < 10^{-6}$, dvs. $n > 37.8$.

IMAT2150: Newtons metode-
Vårt tidligere eksempel med trippel rot

Vi løste tidligere ved hjelp av halveringsmetoden ligningen $f(x) = x^3 - 2x^2 + 4/3\,x - 8/27 = 0$ som ga problemer med nøyaktigheten.[1]

Ligningen foran har har de samme type problemer som denne ligningen hadde, og nå med Newtons metode.[2]

Bruker vi modifisert Newtons metode for $m = 3$ med startverdi $x_0 = 1$ gir dette trippelroten $2/3$ eksakt etter en iterasjon.

[1] Ligningen har trippelroten $2/3$.

[2] Begge ligningene har en trippel rot, som -påpekt tidligere- er roten til problemene, og ikke metodevalget.

IMAT2150: Newtons metode-
Vårt tidligere eksempel med trippel rot

Vi løste tidligere ved hjelp av halveringsmetoden ligningen
$f(x) = x^3 - 2x^2 + 4/3\,x - 8/27 = 0$ som ga problemer med nøyaktigheten.[1]

Ligningen foran har har de samme type problemer som denne ligningen hadde, og nå med Newtons metode.[2]

Bruker vi modifisert Newtons metode for $m = 3$ med startverdi $x_0 = 1$ gir dette trippelroten $2/3$ eksakt etter en iterasjon.

Dette gjelder her for alle startverdier x_0![3]

[1] Ligningen har trippelroten $2/3$.

[2] Begge ligningene har en trippel rot, som -påpekt tidligere- er roten til problemene, og ikke metodevalget.

[3] For spesialtilfellet $f(x) = (x - r)^m$ (si m heltalltig) er $m \times f(x)/f'(x) = x - r$, så $x - (x - r) = r$ uavhengig av x.

IMAT2150: Newtons metode-
Vårt tidligere eksempel med trippel rot

Vi løste tidligere ved hjelp av halveringsmetoden ligningen
$f(x) = x^3 - 2x^2 + 4/3\,x - 8/27 = 0$ som ga problemer med nøyaktigheten.[1]

Ligningen foran har har de samme type problemer som denne ligningen hadde, og nå med Newtons metode.[2]

Bruker vi modifisert Newtons metode for $m = 3$ med startverdi $x_0 = 1$ gir dette trippelroten $2/3$ eksakt etter en iterasjon.

Dette gjelder her for alle startverdier x_0![3]

MERK:

Å bestemme multiplisiteten til en ukjent rot er ikke pensum!

[1] Ligningen har trippelroten $2/3$.

[2] Begge ligningene har en trippel rot, som -påpekt tidligere- er roten til problemene, og ikke metodevalget.

[3] For spesialtilfellet $f(x) = (x - r)^m$ (si m heltalltig) er $m \times f(x)/f'(x) = x - r$, så $x - (x - r) = r$ uavhengig av x.

GAUSSELIMINASJON [S:2.1]

- Hensikt: Omforme til system som er enklere å løse, for eksempel:

$$\begin{aligned} x + 2y - 3z &= 0 \\ y + z &= 3 \\ 2z &= 4 \end{aligned} \qquad \begin{aligned} x &= 4 \\ y &= 1 \\ z &= 2 \end{aligned}$$

- Hvordan:
  1. Bytte plass for to likninger
  2. Legge til et multiplum av en likning til en annen.
  3. Multiplisere en likning med en konstant ikke lik null.

$$\begin{bmatrix} 1 & 2 & 1 \\ 1 & 3 & 2 \\ -2 & 2 & 1 \end{bmatrix} \begin{matrix} \\ (-1) \\ (2) \end{matrix} \sim \begin{bmatrix} 1 & 2 & 1 \\ 0 & 1 & 1 \\ 0 & 6 & 3 \end{bmatrix} \begin{matrix} \\ \\ (-6) \end{matrix}$$

$$\begin{bmatrix} 1 & 2 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & -3 \end{bmatrix}$$

LS fra forrige slide med høyreside $(1, -1, 4)^T$ nedenfor løses med såkalt "tilbakesetting" (back substitution) med Python der høyresiden først har blitt transformert til $bt = (1, -2, 18)^T$. (Vis det!).

LS fra forrige slide med høyreside $(1, -1, 4)^T$ nedenfor løses med såkalt "tilbakesetting" (back substitution) med Python der høyresiden først har blitt transformert til $bt = (1, -2, 18)^T$.
(Vis det!).

```
A=array([[1,2,1,1],[1,3,2,-1],[-2,2,1,4]])
print()
print("naiv gauss eliminasjon for");print(A)
print()
print("med øvre triangulært resultat");print(gausselimination(A))
print()
print("og løsning av LS ved tilbakeinnsetting")
print(backsub(gausselimination(A)))
print()
```

```
naiv gauss eliminasjon for
[[ 1  2  1  1]
 [ 1  3  2 -1]
 [-2  2  1  4]]

med øvre triangulært resultat
[[ 1  2  1  1]
 [ 0  1  1 -2]
 [ 0  0 -3 18]]

og løsning av LS ved tilbakeinnsetting
[-1.0, 4.0, -6.0]
```

LS fra forrige slide med høyreside $(1, -1, 4)^T$ nedenfor løses med såkalt "tilbakesetting" (back substitution) med Python der høyresiden først har blitt transformert til $bt = (1, -2, 18)^T$. (Vis det!).

```
A=array([[1,2,1,1],[1,3,2,-1],[-2,2,1,4]])
print()
print("naiv gauss eliminasjon for");print(A)
print()
print("med øvre triangulært resultat");print(gausselimination(A))
print()
print("og løsning av LS ved tilbakeinnsetting")
print(backsub(gausselimination(A)))
print()
```

```
naiv gauss eliminasjon for
[[ 1  2  1  1]
[ 1  3  2 -1]
[-2  2  1  4]]

med øvre triangulært resultat
[[ 1  2  1  1]
[ 0  1  1 -2]
[ 0  0 -3 18]]

og løsning av LS ved tilbakeinnsetting
[-1.0, 4.0, -6.0]
```

LS-et $Ux = bt$ (med koeffisientmatrise nå på såkalt "øvre-triangulær form" U) løses så baklengs.[1]

[1] Ligning 3 gir $-3z = 18$, dvs, $z = -6$, og da gir ligning 2 at $y + z = -2$, dvs. $y = 4$, som til slutt gir at $x + 2y + z = 1$, dvs. $x = -1$ fra ligning 1, i samsvar med den gitte løsningen.

Gitt ligningssystemet $Ax = b$ for en $n \times n$ koeffisientmatrise.

[1] Det blir klart senere hvorfor dette kalles naiv Gausseliminasjon.

Gitt ligningssystemet $Ax = b$ for en $n \times n$ koeffisientmatrise.

Vi starter med å eliminere alle koeffisienter i kolonne 1 under matrise-diagonalen ved hjelp av rad 1 (ved bruk av det såkalte pivotelementet a_{11}).

Dette gjentas tilsvarende for kolonne 2, 3 og til og med kolonne $n - 1$.

[1] Det blir klart senere hvorfor dette kalles naiv Gausseliminasjon.

Gitt ligningssystemet $Ax = b$ for en $n \times n$ koeffisientmatrise.

Vi starter med å eliminere alle koeffisienter i kolonne 1 under matrise-diagonalen ved hjelp av rad 1 (ved bruk av det såkalte pivotelementet a_{11}).

Dette gjentas tilsvarende for kolonne 2, 3 og til og med kolonne $n - 1$.

Hvis alle pivotelementer vi møter er forskjellig fra 0, så virker eliminasjonen; hvis ikke feiler den (divisjon med 0).

[1] Det blir klart senere hvorfor dette kalles naiv Gausseliminasjon.

Gitt ligningssystemet $Ax = b$ for en $n \times n$ koeffisientmatrise.

Vi starter med å eliminere alle koeffisienter i kolonne 1 under matrise-diagonalen ved hjelp av rad 1 (ved bruk av det såkalte pivotelementet a_{11}).

Dette gjentas tilsvarende for kolonne 2, 3 og til og med kolonne $n - 1$.

Hvis alle pivotelementer vi møter er forskjellig fra 0, så virker eliminasjonen; hvis ikke feiler den (divisjon med 0).

Hvis algoritmen er en suksess, så er LS-et (som i eksemplet) transformert til et LS med koeffisentmatrise som er en øvre triangulær matrise og med en tilsvarende transformert høyreside.

[1] Det blir klart senere hvorfor dette kalles naiv Gausseliminasjon.

Gitt ligningssystemet $Ax = b$ for en $n \times n$ koeffisientmatrise.

Vi starter med å eliminere alle koeffisienter i kolonne 1 under matrise-diagonalen ved hjelp av rad 1 (ved bruk av det såkalte pivotelementet a_{11}).

Dette gjentas tilsvarende for kolonne 2, 3 og til og med kolonne $n - 1$.

Hvis alle pivotelementer vi møter er forskjellig fra 0, så virker eliminasjonen; hvis ikke feiler den (divisjon med 0).

Hvis algoritmen er en suksess, så er LS-et (som i eksemplet) transformert til et LS med koeffisentmatrise som er en øvre triangulær matrise og med en tilsvarende transformert høyreside.

Og, da kan LS-et løses lett (effektivt) baklengs.

[1] Det blir klart senere hvorfor dette kalles naiv Gausseliminasjon.

Dette LS-et har samme løsning som det opprinnelige LS-et (hvis det da har en løsning).

Hvordan unngå å møte pivotelementer som er 0? (Noe som kan skje her.)

Dette LS-et har samme løsning som det opprinnelige LS-et (hvis det da har en løsning).

Hvordan unngå å møte pivotelementer som er 0? (Noe som kan skje her.)

For en ikke-singulær matrise A kan det vises at null-pivotlementer kan unngås. Dette oppnås med såkalt pivotering. (Dette forklares senere.)

Dette LS-et har samme løsning som det opprinnelige LS-et (hvis det da har en løsning).

Hvordan unngå å møte pivotelementer som er 0? (Noe som kan skje her.)

For en ikke-singulær matrise A kan det vises at null-pivotlementer kan unngås. Dette oppnås med såkalt pivotering. (Dette forklares senere.)

Vi skal senere se at pivotering også bøter på et annet problem som ellers kan gi løsninger som er helt feil pga. såkalt oversvømming ("swamping").

Dette LS-et har samme løsning som det opprinnelige LS-et (hvis det da har en løsning).

Hvordan unngå å møte pivotelementer som er 0? (Noe som kan skje her.)

For en ikke-singulær matrise A kan det vises at null-pivotlementer kan unngås. Dette oppnås med såkalt pivotering. (Dette forklares senere.)

Vi skal senere se at pivotering også bøter på et annet problem som ellers kan gi løsninger som er helt feil pga. såkalt oversvømming ("swamping").

Disse to muligheter for problemer med algoritmen rettferdigjør å kalle den naiv: Uten modifikasjoner kan altså algoritmen feile eller gi helt feilaktige løsninger.

Det på de to forrige slidene kan illustreres som følger:

Det på de to forrige slidene kan illustreres som følger:

$$\left[\begin{array}{cccc|c} a_{11} & a_{12} & \dots & a_{1n} & b_1 \\ a_{21} & a_{22} & \dots & a_{2n} & b_2 \\ \vdots & \vdots & \dots & \vdots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} & b_n \end{array}\right]$$

Det på de to forrige slidene kan illustreres som følger:

$$\left[\begin{array}{cccc|c} a_{11} & a_{12} & \dots & a_{1n} & b_1 \\ a_{21} & a_{22} & \dots & a_{2n} & b_2 \\ \vdots & \vdots & \dots & \vdots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} & b_n \end{array}\right]$$

$$\begin{array}{cccc|c} a_{11} & a_{12} & \dots & a_{1n} & b_1 \\ 0 & a_{22} - \frac{a_{21}}{a_{11}}a_{12} & \dots & a_{2n} - \frac{a_{21}}{a_{11}}a_{1n} & b_2 - \frac{a_{21}}{a_{11}}b_1 \end{array}$$

Det på de to forrige slidene kan illustreres som følger:

$$\left[\begin{array}{cccc|c} a_{11} & a_{12} & \dots & a_{1n} & b_1 \\ a_{21} & a_{22} & \dots & a_{2n} & b_2 \\ \vdots & \vdots & \dots & \vdots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} & b_n \end{array}\right]$$

$$\begin{array}{cccc|c} a_{11} & a_{12} & \dots & a_{1n} & b_1 \\ 0 & a_{22} - \frac{a_{21}}{a_{11}}a_{12} & \dots & a_{2n} - \frac{a_{21}}{a_{11}}a_{1n} & b_2 - \frac{a_{21}}{a_{11}}b_1 \end{array}$$

$$\left[\begin{array}{cccc|c} a_{11} & a_{12} & \dots & a_{1n} & b_1 \\ 0 & a_{22} & \dots & a_{2n} & b_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & a_{nn} & b_n \end{array}\right]$$

Merk at (variablene) $a_{2,2}$ etc. redefineres fortløpende.

Det på de to forrige slidene kan illustreres som følger:

$$\left[\begin{array}{cccc|c} a_{11} & a_{12} & \dots & a_{1n} & b_1 \\ a_{21} & a_{22} & \dots & a_{2n} & b_2 \\ \vdots & \vdots & \dots & \vdots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} & b_n \end{array}\right]$$

$$\begin{array}{cccc|c} a_{11} & a_{12} & \dots & a_{1n} & b_1 \\ 0 & a_{22} - \frac{a_{21}}{a_{11}}a_{12} & \dots & a_{2n} - \frac{a_{21}}{a_{11}}a_{1n} & b_2 - \frac{a_{21}}{a_{11}}b_1 \end{array}$$

$$\left[\begin{array}{cccc|c} a_{11} & a_{12} & \dots & a_{1n} & b_1 \\ 0 & a_{22} & \dots & a_{2n} & b_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & a_{nn} & b_n \end{array}\right]$$

Merk at (variablene) $a_{2,2}$ etc. redefineres fortløpende.
Hva skjer hvis alle elementer i 1. kolonne i utgangspunktet er 0?

Setning

For en $n \times n$ matrise A er antall aritmetiske regneoperasjoner $(+, -, x, /)$ for Gausseliminasjonen er lik $2n^3/3 + n^2/2 - 7n/6 = \mathcal{O}(n^3)$.[1]

[1] GE krever altså såkalt $\mathcal{O}(n^3)$ regneoperasjoner, som betyr at antallet operasjoner er høyst et multiplium av n^3 for alle tilstrekkelige store verdier av n.

Setning

For en $n \times n$ matrise A er antall aritmetiske regneoperasjoner $(+, -, x, /)$ for Gausseliminasjonen er lik $2n^3/3 + n^2/2 - 7n/6 = \mathcal{O}(n^3)$.[1]

Vi utelater beviset, men beviset er i detalj gitt i SAUER 2.1.2 (s. 74–76).

[1] GE krever altså såkalt $\mathcal{O}(n^3)$ regneoperasjoner, som betyr at antallet operasjoner er høyst et multiplium av n^3 for alle tilstrekkelige store verdier av n.

Setning

For en $n \times n$ matrise A er antall aritmetiske regneoperasjoner $(+, -, x, /)$ for Gausseliminasjonen er lik $2n^3/3 + n^2/2 - 7n/6 = \mathcal{O}(n^3)$.[1]

Vi utelater beviset, men beviset er i detalj gitt i SAUER 2.1.2 (s. 74–76).

Og, i stedet for å bevise algoritmen for GE i det generelle tilfellet skal vi illustrere den ved hjelp av flere eksempler.

[1] GE krever altså såkalt $\mathcal{O}(n^3)$ regneoperasjoner, som betyr at antallet operasjoner er høyst et multiplium av n^3 for alle tilstrekkelige store verdier av n.

Setning

For en $n \times n$ matrise A er antall aritmetiske regneoperasjoner $(+, -, x, /)$ for Gausseliminasjonen er lik $2n^3/3 + n^2/2 - 7n/6 = \mathcal{O}(n^3)$.[1]

Vi utelater beviset, men beviset er i detalj gitt i SAUER 2.1.2 (s. 74–76).

Og, i stedet for å bevise algoritmen for GE i det generelle tilfellet skal vi illustrere den ved hjelp av flere eksempler.

Men, først litt om regneeffektiviteten av GE!

[1] GE krever altså såkalt $\mathcal{O}(n^3)$ regneoperasjoner, som betyr at antallet operasjoner er høyst et multiplium av n^3 for alle tilstrekkelige store verdier av n.

[1] Antall flops er antall flyttalsoperasjoner.

I tabellen nedefor er antall flops[1] for GE angitt.

n	Elimination	Back Substitution	Total Flops	$2n^3/3$	Percent Due to Elimination
10	705	100	805	667	87.58%
100	671550	10000	681550	666667	98.53%
1000	6.67×10^8	1×10^6	6.68×10^8	6.67×10^8	99.85%

[1] Antall flops er antall flyttalsoperasjoner.

I tabellen nedefor er antall flops[1] for GE angitt.

n	Elimination	Back Substitution	Total Flops	$2n^3/3$	Percent Due to Elimination
10	705	100	805	667	87.58%
100	671550	10000	681550	666667	98.53%
1000	6.67×10^8	1×10^6	6.68×10^8	6.67×10^8	99.85%

Merk at for $n = 1000$ har LS-et 1 million ukjente, som i realistiske anvendelser, ikke er av de største LS-ene som opptrer på regelmessig basis.

[1] Antall flops er antall flyttalsoperasjoner.

I tabellen nedefor er antall flops[1] for GE angitt.

n	Elimination	Back Substitution	Total Flops	$2n^3/3$	Percent Due to Elimination
10	705	100	805	667	87.58%
100	671550	10000	681550	666667	98.53%
1000	6.67×10^8	1×10^6	6.68×10^8	6.67×10^8	99.85%

Merk at for $n = 1000$ har LS-et 1 million ukjente, som i realistiske anvendelser, ikke er av de største LS-ene som opptrer på regelmessig basis.

Undertegnede jobbet tdiligere med anvendelser der LS-ene rutinemessig var så store, om ikke større. (Men, vi brukte ikke GE!)

[1] Antall flops er antall flyttalsoperasjoner.

I tabellen nedefor er antall flops[1] for GE angitt.

n	Elimination	Back Substitution	Total Flops	$2n^3/3$	Percent Due to Elimination
10	705	100	805	667	87.58%
100	671550	10000	681550	666667	98.53%
1000	6.67×10^8	1×10^6	6.68×10^8	6.67×10^8	99.85%

Merk at for $n = 1000$ har LS-et 1 million ukjente, som i realistiske anvendelser, ikke er av de største LS-ene som opptrer på regelmessig basis.

Undertegnede jobbet tdiligere med anvendelser der LS-ene rutinemessig var så store, om ikke større. (Men, vi brukte ikke GE!)

Og, disse LS-ene måtte løses veldig mange ganger med forskjellige koeffisientmatriser og høyresider.

[1] Antall flops er antall flyttalsoperasjoner.

I tabellen nedefor er antall flops[1] for GE angitt.

n	Elimination	Back Substitution	Total Flops	$2n^3/3$	Percent Due to Elimination
10	705	100	805	667	87.58%
100	671550	10000	681550	666667	98.53%
1000	6.67×10^8	1×10^6	6.68×10^8	6.67×10^8	99.85%

Merk at for $n = 1000$ har LS-et 1 million ukjente, som i realistiske anvendelser, ikke er av de største LS-ene som opptrer på regelmessig basis.

Undertegnede jobbet tdiligere med anvendelser der LS-ene rutinemessig var så store, om ikke større. (Men, vi brukte ikke GE!)

Og, disse LS-ene måtte løses veldig mange ganger med forskjellige koeffisientmatriser og høyresider.

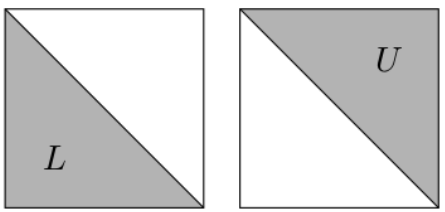
Slike store LS-er oppstår ved for eksempel løsning av kompliserte ikke-lineære partielle differentialligninger for modelliering av tre-fase flyt av olje og gass i reservoarer.

[1] Antall flops er antall flyttalsoperasjoner.

Gausseliminasjon ved hjelp av matriseoperasjoner

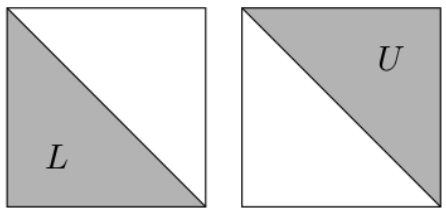
Definisjon:

En $m \times n$ matrise L er **nedre triangulær** hvis og bare hvis dens elementer $l_{ij} = 0$ for $i < j$, og en $m \times n$ matrise U er **øvre triangulær** hvis og bare hvis dens elementer $u_{ij} = 0$ for $i > j$.



Definisjon:

En $m \times n$ matrise L er **nedre triangulær** hvis og bare hvis dens elementer $l_{ij} = 0$ for $i < j$, og en $m \times n$ matrise U er **øvre triangulær** hvis og bare hvis dens elementer $u_{ij} = 0$ for $i > j$.

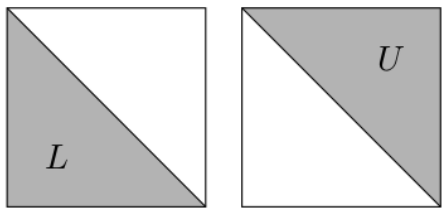


Spesielt for LU faktorisering av en kvadratisk $(n \times n)$ matrise $A = L\,U$ skal vi som illustrert ovenfor bruke.[1]

[1] Betingelsene nedenfor sikrer at algoritmen vår gir en entydig LU faktorisering. Det er $n^2 + n$ elementer å bestemme for koeffisientene til L og U mens betingelsen $A = LU$ bare gir n^2 ligninger. Så, for å sikre entydighet, må vi ha tilleggsbetingelser.

Definisjon:

En $m \times n$ matrise L er **nedre triangulær** hvis og bare hvis dens elementer $l_{ij} = 0$ for $i < j$, og en $m \times n$ matrise U er **øvre triangulær** hvis og bare hvis dens elementer $u_{ij} = 0$ for $i > j$.



Spesielt for LU faktorisering av en kvadratisk ($n \times n$) matrise $A = L\,U$ skal vi som illustrert ovenfor bruke.[1]

L: En nedre triangulær matrise med bare 1-ere på diagonalen og bare 0-ere over diagonalen

U: En øvre triangulær matrise med bare 0-ere under diagonalen

[1] Betingelsene nedenfor sikrer at algoritmen vår gir en entydig LU faktorisering. Det er $n^2 + n$ elementer å bestemme for koeffisientene til L og U mens betingelsen $A = LU$ bare gir n^2 ligninger. Så, for å sikre entydighet, må vi ha tilleggsbetingelser.

LU FAKTORISERING [S:2.2]

LU faktoriseringen av A som $A = LU$ gjør det mulig å effektivt løse ligningssystemet $Ax = b$ ($L(Ux) = b$, $y = Ux$, $Ly = b$):

LU faktoriseringen av A som $A = LU$ gjør det mulig å effektivt løse ligningssystemet $Ax = b$ ($L(Ux) = b$, $y = Ux$, $Ly = b$):

> Now consider a system $A\mathbf{x} = \mathbf{b}$ where A can be factored as $A = LU$ where L is lower triangular and U is upper triangular. Then the system $A\mathbf{x} = \mathbf{b}$ can be solved in two stages as follows:
>
> 1. *First solve* $L\mathbf{y} = \mathbf{b}$ *for* $\mathbf{y}$ *by forward substitution.*
> 2. *Then solve* $U\mathbf{x} = \mathbf{y}$ *for* $\mathbf{x}$ *by back substitution.*

LU faktoriseringen av A som $A = LU$ gjør det mulig å effektivt løse ligningssystemet $Ax = b$ ($L(Ux) = b$, $y = Ux$, $Ly = b$):

> Now consider a system $A\mathbf{x} = \mathbf{b}$ where A can be factored as $A = LU$ where L is lower triangular and U is upper triangular. Then the system $A\mathbf{x} = \mathbf{b}$ can be solved in two stages as follows:
>
> 1. *First solve* $L\mathbf{y} = \mathbf{b}$ *for* $\mathbf{y}$ *by forward substitution.*
> 2. *Then solve* $U\mathbf{x} = \mathbf{y}$ *for* $\mathbf{x}$ *by back substitution.*

LU faktoriseringen gjør det også mulig å effektivt løse mange LS med samme koeffisient-matrise, men med forskjellige høyresider:

LU faktoriseringen av A som $A = LU$ gjør det mulig å effektivt løse ligningssystemet $Ax = b$ ($L(Ux) = b$, $y = Ux$, $Ly = b$):

> Now consider a system $A\mathbf{x} = \mathbf{b}$ where A can be factored as $A = LU$ where L is lower triangular and U is upper triangular. Then the system $A\mathbf{x} = \mathbf{b}$ can be solved in two stages as follows:
>
> 1. *First solve* $L\mathbf{y} = \mathbf{b}$ *for* $\mathbf{y}$ *by forward substitution.*
> 2. *Then solve* $U\mathbf{x} = \mathbf{y}$ *for* $\mathbf{x}$ *by back substitution.*

LU faktoriseringen gjør det også mulig å effektivt løse mange LS med samme koeffisient-matrise, men med forskjellige høyresider:

> LU-factorization is particularly important if, as often happens in business and industry, a series of equations $A\mathbf{x} = B_1$, $A\mathbf{x} = B_2$, $\ldots$, $A\mathbf{x} = B_k$, must be solved, each with the same coefficient matrix A. It is very efficient to solve the first system by gaussian elimination, simultaneously creating an LU-factorization of A, and then using the factorization to solve the remaining systems by forward and back substitution.

EKSEMPEL:[1]

Assume that it takes one second to factorize the 300×300 matrix A into $A = LU$. How many problems $Ax = b_1, \ldots, Ax = b_k$ can be solved in the next second?

The two back substitutions for each b_i require a total of $2n^2$ operations. Therefore, the approximate number of b_i that can be handled per second is

$$\frac{\frac{2n^3}{3}}{2n^2} = \frac{n}{3} = 100.$$

◀

[1] Example 2.8 SAUER, s. 84.

Definisjon

Let $L_{ij}(-c)$ denote the lower triangular matrix whose only nonzero entries are 1's on the main diagonal and $-c$ in the $(i,\ j)$ position. Then $A \longrightarrow L_{ij}(-c)A$ represents the row operation "subtracting c times row j from row i."

Definisjon

Let $L_{ij}(-c)$ denote the lower triangular matrix whose only nonzero entries are 1's on the main diagonal and $-c$ in the $(i,\ j)$ position. Then $A \longrightarrow L_{ij}(-c)A$ represents the row operation "subtracting c times row j from row i."

The following matrix product equation holds.

$$\begin{bmatrix} 1 & & \\ c_1 & 1 & \\ & & 1 \end{bmatrix}\begin{bmatrix} 1 & & \\ & 1 & \\ c_2 & & 1 \end{bmatrix}\begin{bmatrix} 1 & & \\ & 1 & \\ & c_3 & 1 \end{bmatrix} = \begin{bmatrix} 1 & & \\ c_1 & 1 & \\ c_2 & c_3 & 1 \end{bmatrix}.$$

Definisjon

Let $L_{ij}(-c)$ denote the lower triangular matrix whose only nonzero entries are 1's on the main diagonal and $-c$ in the $(i,\ j)$ position. Then $A \longrightarrow L_{ij}(-c)A$ represents the row operation "subtracting c times row j from row i."

The following matrix product equation holds.

$$\begin{bmatrix} 1 & & \\ c_1 & 1 & \\ & & 1 \end{bmatrix}\begin{bmatrix} 1 & & \\ & 1 & \\ c_2 & & 1 \end{bmatrix}\begin{bmatrix} 1 & & \\ & 1 & \\ & c_3 & 1 \end{bmatrix} = \begin{bmatrix} 1 & & \\ c_1 & 1 & \\ c_2 & c_3 & 1 \end{bmatrix}.$$

Sammenhengen over kan skrives som $L_{21}(c_1)\, L_{31}(c_2)\, L_{32}(c_3)$ og (som vi skal se) brukes i LU faktoriseringen.

Definisjon

Let $L_{ij}(-c)$ denote the lower triangular matrix whose only nonzero entries are 1's on the main diagonal and $-c$ in the $(i,\ j)$ position. Then $A \longrightarrow L_{ij}(-c)A$ represents the row operation "subtracting c times row j from row i."

The following matrix product equation holds.

$$\begin{bmatrix} 1 & & \\ c_1 & 1 & \\ & & 1 \end{bmatrix}\begin{bmatrix} 1 & & \\ & 1 & \\ c_2 & & 1 \end{bmatrix}\begin{bmatrix} 1 & & \\ & 1 & \\ & c_3 & 1 \end{bmatrix} = \begin{bmatrix} 1 & & \\ c_1 & 1 & \\ c_2 & c_3 & 1 \end{bmatrix}.$$

Sammenhengen over kan skrives som $L_{21}(c_1)\, L_{31}(c_2)\, L_{32}(c_3)$ og (som vi skal se) brukes i LU faktoriseringen.

Gausseliminasjonen er som sagt NAIV når det ikke forekommer rad (eller kolonne) ombyttinger underveis. Vi kaller pivoteringen som delvis ("partial pivoting") hvis vi kun gjør radbyttinger.[1] Vi kommer tilbake til det viktige temaet pivotering.

[1] Gjøres både rad- og kolonnebyttinger sier vi at pivoteringen er fullstendig. Fullstendig pivotering øker regnetiden drastisk for store ligningssystemer og brukes sjelden.

To nyttige egenskaper til $L_{ij}(-c)$

To nyttige egenskaper til $L_{ij}(-c)$

Minner om at radbyttet $R_i - c\,R_j$ er ekvivalent med matrisemultiplikasjonen $L_{ij}(-c)\,A$.

For example, multiplication by $L_{21}(-c)$ yields

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \longrightarrow \begin{bmatrix} 1 & 0 & 0 \\ -c & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

$$= \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} - ca_{11} & a_{22} - ca_{12} & a_{23} - ca_{13} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

To nyttige egenskaper til $L_{ij}(-c)$

Minner om at radbyttet $R_i - c\,R_j$ er ekvivalent med matrisemultiplikasjonen $L_{ij}(-c)\,A$.

For example, multiplication by $L_{21}(-c)$ yields

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \longrightarrow \begin{bmatrix} 1 & 0 & 0 \\ -c & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

$$= \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} - ca_{11} & a_{22} - ca_{12} & a_{23} - ca_{13} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

$$L_{ij}(-c)^{-1} = L_{ij}(c).$$

For example,

$$\begin{bmatrix} 1 & 0 & 0 \\ -c & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ c & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

IMAT2150: LU faktorisering med NAIV Gausseliminasjon- Eksempel 2.5 (SAUER s. 80) 1(2)

Bestem LU faktoriseringen av $\mathbf{A} = \begin{pmatrix} 1 & 2 & -1 \\ 2 & 1 & -2 \\ -3 & 1 & 1 \end{pmatrix}$.

$$\left[\begin{array}{ccc} 1 & 2 & -1 \\ 2 & 1 & -2 \\ -3 & 1 & 1 \end{array}\right] \qquad R_2 - 2R_1$$

$$\Rightarrow \left[\begin{array}{ccc} 1 & 2 & -1 \\ 0 & -3 & 0 \\ -3 & 1 & 1 \end{array}\right] \qquad R_3 + 3R_1$$

$$\Rightarrow \left[\begin{array}{ccc} 1 & 2 & -1 \\ 0 & -3 & 0 \\ 0 & 7 & -2 \end{array}\right] \qquad R_3 + \tfrac{7}{3}R_2$$

$$\Rightarrow \left[\begin{array}{ccc} 1 & 2 & -1 \\ 0 & -3 & 0 \\ 0 & 0 & -2 \end{array}\right] \qquad = \mathbf{U}$$

IMAT2150: LU faktorisering med NAIV Gausseliminasjon- Eksempel 2.5 (SAUER) 2(2)

Minner igjen om at $R_i - c\,R_j$ er ekvivalent med $L_{ij}(-c)\,A$!

IMAT2150: LU faktorisering med NAIV Gausseliminasjon- Eksempel 2.5 (SAUER) 2(2)

Minner igjen om at $R_i - c\,R_j$ er ekvivalent med $L_{ij}(-c)\,A$!

Vi har brukt følgende radoperasjoner $R_2 - 2R_1$, $R_3 + 3R_1$ og $R_3 + \frac{7}{3}R_2$, som da er ekvivalent med $\tilde{L}A = U$ for

$$\tilde{L} = L_{32}(7/3)\,L_{31}(3)\,L_{21}(-2)\,.$$

IMAT2150: LU faktorisering med NAIV Gausseliminasjon- Eksempel 2.5 (SAUER) 2(2)

Minner igjen om at $R_i - c\,R_j$ er ekvivalent med $L_{ij}(-c)\,A$!

Vi har brukt følgende radoperasjoner $R_2 - 2R_1$, $R_3 + 3R_1$ og $R_3 + \frac{7}{3}R_2$, som da er ekvivalent med $\tilde{L}A = U$ for

$$\tilde{L} = L_{32}(7/3)\,L_{31}(3)\,L_{21}(-2)\,.$$

Da er $A = LU$ med $L = (\tilde{L})^{-1} = L_{21}(2)\,L_{31}(-3)\,L_{32}(-7/3)$.[1]

Så $\mathbf{L} = \begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -3 & -\frac{7}{3} & 1 \end{pmatrix}$ og $\mathbf{U} = \begin{pmatrix} 1 & 2 & -1 \\ 0 & -3 & 0 \\ 0 & 0 & -2 \end{pmatrix}$.

[1] Merk at for kvadratiske inverterbare matriser A og B har vi at $(A\,B)^{-1} = B^{-1}\,A^{-1}$ og $L_{ij}^{-1}(-c) = L_{ij}(c)$.

IMAT2150: LU faktorisering med NAIV Gausseliminasjon- Eksempel 2.5 (SAUER) 2(2)

Minner igjen om at $R_i - c\,R_j$ er ekvivalent med $L_{ij}(-c)\,A$!

Vi har brukt følgende radoperasjoner $R_2 - 2R_1$, $R_3 + 3R_1$ og $R_3 + \frac{7}{3}R_2$, som da er ekvivalent med $\tilde{L}A = U$ for

$$\tilde{L} = L_{32}(7/3)\,L_{31}(3)\,L_{21}(-2)\,.$$

Da er $A = LU$ med $L = (\tilde{L})^{-1} = L_{21}(2)\,L_{31}(-3)\,L_{32}(-7/3)$.[1]

Så $\mathbf{L} = \begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -3 & -\frac{7}{3} & 1 \end{pmatrix}$ og $\mathbf{U} = \begin{pmatrix} 1 & 2 & -1 \\ 0 & -3 & 0 \\ 0 & 0 & -2 \end{pmatrix}$.

Kontroller selv at $A = LU$![2]

[1] Merk at for kvadratiske inverterbare matriser A og B har vi at $(A\,B)^{-1} = B^{-1}\,A^{-1}$ og $L_{ij}^{-1}(-c) = L_{ij}(c)$.

[2] For slike oppgaver er dette er alltid å anbefale.