TANA23 Föreläsning 3

- Feluppskattning. Konvergenshastighet.
- Existens. Iteration. Konvergens.
- Tillämpning - Kvadratrots implementering.
- Felkällor vid en beräkning. Exempel.

Feluppskattning

Låt $\bar{x}$ vara en approximation av roten x^*. Hur skall felet $|\bar{x} - x^*|$ uppskattas?

Metodoberoende feluppskattning

$$|\bar{x} - x^*| \leq \frac{|f(\bar{x})|}{M}$$

där $|f'(x)| \geq M$ nära roten x^*.

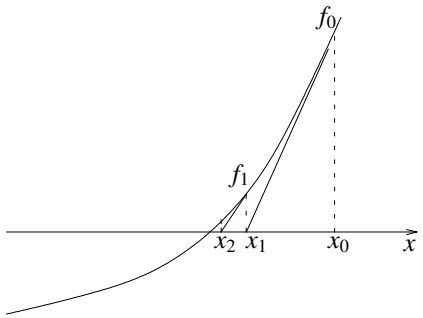
Konstanten M kan vara svår att beräkna. Vanligast är att använda

$$|\bar{x} - x^*| \lesssim \frac{|f(\bar{x})|}{|f'(\bar{x})|}$$

och avrunda för att få lite marginal i uppskattningen.

Exempel Vi har tidigare bestämt en rot $\bar{x} = 0.56714335$ till ekvationen $f(x) = x - \mathrm{e}^{-x}$. Gör en feluppskattning.

Newton-Raphsons metod



Newton-Raphsons metod Givet x_0 beräknar vi en talföljd

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}, \qquad k = 0, 1, 2 \ldots$$

Exempel Lös ekvationen $f(x) = x - e^{-x} = 0$. Här är $f'(x) = 1 + e^{-x}$.

k	x_k	$f(x_k)$
0	0.550000000	$-2.69 \cdot 10^{-2}$
1	0.567089834	$-8.38 \cdot 10^{-5}$
2	0.567143290	$-8.10 \cdot 10^{-10}$

Mycket snabb konvergens.

I varje steg behövs två funktionsanrop (både $f(x)$ och $f'(x)$).

Kräver att vi kan beräkna $f'(x)$.

Konvergenshastighet

Definition Låt $\{x_k\}$ vara den talföljd som beräknas med en iterations metod. I de fall då metoden är konvergent gäller att *konvergensordningen* är det största p, för vilket

$$\frac{|x_{k+1} - x^*|}{|x_k - x^*|^p} \to C \neq 0, \text{ då } k \to \infty.$$

Om $p = 1$ är konvergensen *linjär* och om $p = 2$ *kvadratisk*.

Exempel Om vi har $C = 1$ och $|x_k - x^*| < 3.5 \cdot 10^{-4}$ så fås $|x_{k+1} - x^*| < (3.5 \cdot 10^{-4})^2 < 1.3 \cdot 10^{-7}$. Mycket snabb konvergens!

Konvergenshastighet för Newton-Raphsons metod

Lemma Antag att $f(x)$ har två kontinuerliga derivator och att $f'(a) = 0$. Då gäller att

$$|f(\bar{a}) - f(a)| \leq \frac{M}{2}|\bar{a} - a|^2,$$

där $|f''(\xi)| \leq M$ för något $\xi \in [\bar{a}, a]$.

Detta är en variant av *Medelvärdessatsen*. Mer precist resultat finns i Analysboken.

Lemma Låt $x_{k+1} = \varphi(x_k)$ vara en *fixpunktsiteration* som konvergerar mot x^*. Då gäller att om $\varphi'(x^*) = 0$ så är konvergenshastigheten minst kvadratisk.

Exempel En *enkelrot* har $f(x^*) = 0$ och $f'(x^*) \neq 0$. Vid en *dubbelrot* är både $f(x^*) = 0$ och $f'(x^*) = 0$.

Sats Låt $\varphi(x) = x - f(x)/f'(x)$ vara iterationsfunktionen för Newton-Raphsons metod. För en *enkelrot* x^* gäller att $\varphi'(x^*) = 0$.

Newton-Raphsons metod konvergerar alltså kvadratiskt mot en enkelrot. Vad händer om x^* är en dubbelrot?

Exempel

Lös $f(x) = (x - \mathrm{e}^{-x})^2$ med Newton-Raphson.

k	x_k	$f(x_k)$
0	0.550000000000000	$7.3 \cdot 10^{-4}$
1	0.581851788560507	$5.3 \cdot 10^{-4}$
2	0.555794973335401	$3.2 \cdot 10^{-4}$
3	0.576692919807852	$2.2 \cdot 10^{-4}$
4	0.559647895670487	$1.4 \cdot 10^{-4}$
5	0.57337002513594	$9.5 \cdot 10^{-5}$
20	0.566881467726078	$1.7 \cdot 10^{-7}$

Detta är linjär konvergens!

Lemma Om x^* är en *dubbelrot* så har Newton-Raphsons metod konvergensordning $p = 1$.

Sekantmetoden

Om derivatan $f'(x)$ är svår att beräkna kan vi approximera

$$f'(x_k) \approx \frac{f(x_k) - f(x_{k-1})}{x_k - x_{k-1}}.$$

Definition Givet x_0 och x_1 beräknar *Sekantmetoden* en talföljd

$$x_{k+1} = x_k - f(x_k) \left(\frac{f(x_k) - f(x_{k-1})}{x_k - x_{k-1}} \right)^{-1}.$$

Frågor Konvergensordning? Stabil?

Exempel Lös ekvationen $f(x) = x - \mathrm{e}^{-x} = 0$ med Sekantmetoden.

k	x_k	$f(x_k)$
0	0.700000000000000	$2.03 \cdot 10^{-1}$
1	0.600000000000000	$5.12 \cdot 10^{-2}$
2	0.566373515585849	$-1.21 \cdot 10^{-3}$
3	0.567147844602993	$7.14 \cdot 10^{-6}$
4	0.567143291044208	$9.94 \cdot 10^{-10}$
5	0.567143290409783	$-8.9 \cdot 10^{-16}$
6	0.567143290409784	$-1.11 \cdot 10^{-16}$

Snabb konvergens! Kan visa att $p = (1 + \sqrt{5})/2 \approx 1.618$ för Sekantmetoden. Konvergerar mot en enkelrot om start approximationen är bra nog.

Exempel Funktionen

$$f(x) = \cos(x^2) - \frac{1}{2}\mathrm{e}^x,$$

har en positiv rot $x^* \approx 0.5$. Bestäm en approximation av roten med åtminstone 5 korrekta decimaler.

Tillämpning - Kvadratrotsberäkning på dator

Vi vill implementera $\sqrt{a}$ så effektivt som möjligt på dator en dator med IEEE dubbel precisions aritmetik. Hur skall vi göra?

- Lös ekvationen $f(x) = x^2 - a$ med Newton-Raphsons metod.
- Välj start approximation x_0. Utnyttja att a är ett *normaliserat flyttal* så $1 \leq a < 4$ och $1 \leq x < 2$.
- Välj antal iterationer. Vill ha `for`-loop och inte `while`-loop.
- Kontrollera hur bra noggranheten blev. Vill helst ha relativt fel $\leq \mu$.

Start approximation $x_0 = 1.5$ ger ett fel $|x_0 - \sqrt{a}| \leq \frac{1}{2}$.

Alternativt gör en *tabell*

a	$\sqrt{a}$
1.0	1.000000000000000
1.5	1.224744871391589
2.0	1.414213562373095
2.5	1.581138830084190
3.0	1.732050807568877
3.5	1.870828693386971

Nu blir startfelet högst $|x_0 - \sqrt{a}| < 0.12$. Större tabell ger färre iterationer.

Enklare i $C++$ än i Python. Tillgång till bitoperationer.

Sats För varje $0 < x_0 < \infty$ genererar iterationen

$$x_{k+1} = \frac{1}{2}(x_k + \frac{a}{x_k})$$

en konvergent följd och

$$\lim_{k\to\infty} x_k = \sqrt{a}.$$

Sats Konvergensen är *kvadratisk* och

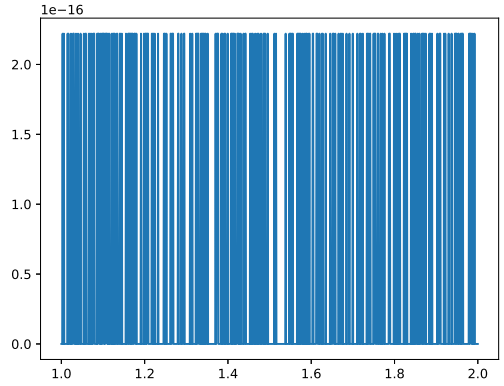
$$|x_{k+1} - \sqrt{a}| \leq \frac{1}{2}(x_k - \sqrt{a})^2.$$

Python Implementation

```
def kvadratrot(a):
    table=[1.000000000000000,
           1.224744871391589,
           1.414213562373095,
           1.581138830084190,
           1.732050807568877,
           1.870828693386971,
           2.000000000000000]
    x=table[int(round(2*(a-1)))]
    for k in range(1,5):
       x=(x+a/x)/2
    return x
```

Totalt 4 divisioner och 4 additioner för att beräkna $\sqrt{a}$.

Hur blir felet? Finns ändligt många flyttal $1 \leq a < 4$. Testa alla fall!



Skillnaden mellan `sqrt` från `numpy` och `kvadratrot` funktionen för 1000 jämnt utspridda tal på intervallet $[1,2]$. Likhet i precis 754 fall och skillnad 2μ i övriga fall.

Ekvationslösning i Python

Paketet `scipy` innehåller ekvationslösaren `fsolve`. Vi får tillgång till den genom att skriva

```
from scipy.optimize import *
```

Exempel Lös ekvationen $f(x) = x - \mathrm{e}^{-x} = 0$ med Python. Undersök även hur snabbt metoden konvergerar.

Vi skriver

```
from scipy.optimize import *
from numpy import *
def f(x):
    y=x-exp(-x)
    print( x , y )
    return y
 x0=fsolve( f , 0.5 )
 print( x0 , f(x0) )
```

Paketet `numpy` behövs för att `exp` skall vara definierat.

Resultat En rot $x^* = 0.56714329$ hittas av `fsolve` efter 6 funktionsevalueringar.

k	x_k	$f(x_k)$
1	0.5	$-1.07 \cdot 10^{0}$
2	0.50000001	$-1.07 \cdot 10^{-1}$
3	0.566311	$-1.30 \cdot 10^{-3}$
4	0.56713307	$-1.60 \cdot 10^{-5}$
5	0.56714329	$-2.41 \cdot 10^{-9}$
6	0.56714329	$-4.44 \cdot 10^{-15}$

Detta är inte linjär konvergens utan betydligt snabbare! Hur ser snabbare algoritmer ut?

Python lagrar fler decimaler internt än vad som skrivs ut!

Sammanfattning

- Ekvationslösning är en viktig del av flera tillämpningar.
- Kan vi beräkna $f(x)$ och $f'(x)$ är nästan alltid Newton-Raphsons metod bäst.
- Sekant metoden nästan lika snabb och kräver inte $f'(x)$.
- Pythons `fsolve` kan lösa ekvationer av typen $f(x) = 0$. Finns flera alternativ i `Scipy`.
- Fixpunktsiteration är enkelt att använda men långsamt. Används främst då det är svårt att använda andra metoder.

Felkällor vid en beräkning

Exempel Genom att lösa en ekvation har vi fått fram vikten på en cylinder till $m = 92.167339917\,kg$ med ett fel högst $|\Delta m| \leq 0.5 \cdot 10^{-9}$. Vi nöjer oss med att svara $\bar{m} = 92.2$ och gör då ett avrundningsfel $|R_B| \leq 0.5 \cdot 10^{-1}$.

Fråga Vilka fler möjliga felkällor finns?

Vi brukar alltid avrunda slutresultatet till ett lämpligt antal signifikanta siffror.

Definition Då ett tal x avrundas till ett närmevärde $\bar{x}$ görs ett *Avrundningsfel* som betecknas med $|R_B|$.

Om $\bar{x}$ är avrundat på ett korrekt sätt till k decimaler så blir $|R_B| \leq 0.5 \cdot 10^{-k}$.

Definition Då en eller flera parametrar endast är kända med en viss noggranhet fås ett *dataosäkerhetsfel* som betecknas med $|R_X|$.

Exempel Vi vill beräkna en cylinders volym genom att använda

$$V = \pi r^2 h,$$

där r är radien och h är höjden. Vi har felgränser $|\Delta r|$ och $|\Delta h$. Det därav orsakade felet uppskattas med felfortplantningsformeln,

$$|R_X| = |\delta V| \leq |\pi 2\bar{r}\bar{h}||\Delta r| + |\pi \bar{r}^2||\Delta h|.$$

Kommentar Ofta skiljer man på fallet där de använda parametrarna är funktionsvärden från en tabell och betecknar då felet med $|R_{XF}|$.

Definition En numerisk metod innebär att vi inte löser det egentliga problemet exakt utan endast får en approximation. Detta fel kallas *trunkeringsfelet* och betecknas med $|R_T|$.

Exempel Vi vill beräkna ett gränsvärde

$$\lim_{x\to 0} F(x) = \lim_{x\to 0} \frac{1-\cos(x)}{\sin(x)}.$$

På datorn kan vi inte beräkna uttrycket för $x = 0$. Istället väljer vi ett litet x-värde, exempelvis $x = 10^{-5}$ och approximerar

$$\lim_{x\to 0} F(x) \approx F(10^{-5}), \text{ och får } |R_T| = |F(10^{-5}) - F(0)|.$$

Exempel Vi vill beräkna exponentialfunktionen genom att utnyttja serien

$$e^x = 1 + x + \frac{x^2}{2} + \frac{x^3}{6} + \cdots = \sum_{k=0}^{\infty} \frac{x^k}{k!}.$$

Approximerar vi e^x med en partialsumma fås

$$e^x \approx \sum_{k=0}^{n} \frac{x^k}{k!} \quad \text{och} \quad R_T = \sum_{k=n+1}^{\infty} \frac{x^k}{k!}.$$

Här styr antalet n noggranheten och $R_T \to 0$ as $n \to \infty$.

Sammanfattning

- Avrundningsfelet R_B kan väljas fritt och styrs av den noggranhet som efterfrågas.
- Dataosäkerhetsfelet R_X (eller R_{XF}), som kan uppskattas med felfortplantningsformeln, är en del av problemet och kan normalt inte påverkas särskilt mycket.
- Trunkeringsfelet R_T beror på vilken metod vi väljer för att lösa problemet. Många metoder innehåller parametrar som gör det möjligt att få godtyckligt litet trunkeringsfel genom att arbeta hårdare.
- Totalfelet i en beräkning är $R_{TOT} \leq R_B + R_T + R_X$. Helst skall R_B vara det största felet som görs.
- Det går inte alltid att uppfylla noggranhetskrav.