Ickelinjära ekvationer

Beräkningsvetenskap I/KF

Agenda icke-linjära ekvationer

- Varför är det svårt att lösa icke-linjära ekvationer?
- Iterativa metoder, begreppet iteration
- Bisektion (intervallhalvering)
- Newton-Raphsons metod
- Noggrannhet/stoppvillkor
- Konvergenshastighet/exekveringstid

Icke-linjära ekvationer

Ska lösa problemet

$$10\sin(x) + x^2 - 10 = \frac{1}{x}$$

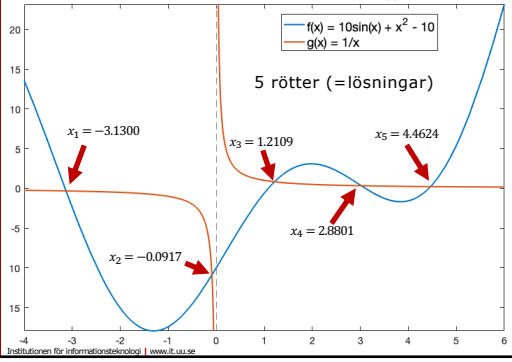
och Matlab tillgängligt.

Vilka steg behövs? (diskutera 2 minuter)

1. Skriv om till f(x) = 0
2. Plotta f(x) (fplot) för att få se hur problemet ”ser ut”, t ex ungefär var nollstället ligger
3. Hitta nollstället (fzero)

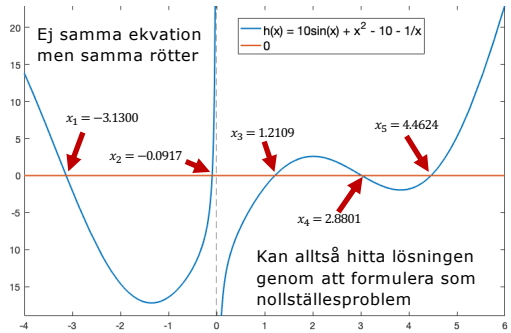
Några icke-linjära exempel

Exempel: $10\sin(x) + x^2 - 10 = \frac{1}{x}$



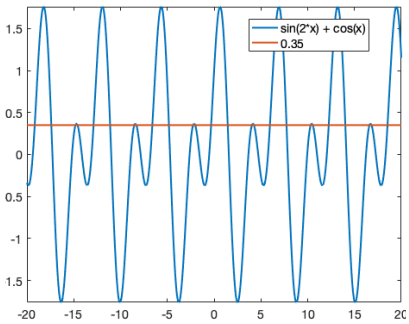
Några icke-linjära exempel

Omformulering: $10\sin(x) + x^2 - 10 - \frac{1}{x} = 0$



Några icke-linjära exempel

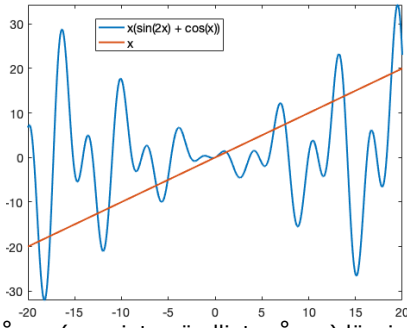
Exempel 2: $\sin(2x) + \cos(x) = 0.35$



Oändligt många lösningar

Några icke-linjära exempel

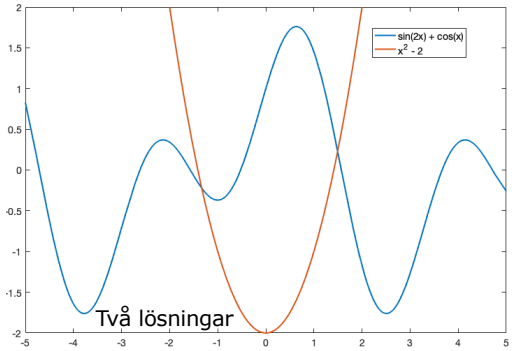
Exempel 3: $x(\sin(2x) + \cos(x)) = x$



Många (men inte oändligt många) lösningar

Några icke-linjära exempel

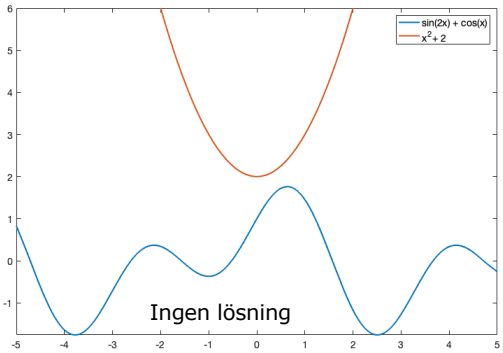
Exempel 4: $\sin(2x) + \cos(x) = x^2 - 2$



Två lösningar

Några icke-linjära exempel

Exempel 5: $\sin(2x) + \cos(x) = x^2 + 2$



Ingen lösning

Slutsatser

- Icke-linjära ekvationer kan ha 1, 2, flera, oändligt många eller ingen lösning alls – det finns ingen regel som säger vad som gäller
- Går (i allmänhet) inte att ”lösa ut” x ur $f(x) = g(x)$ då $f(x)$ och/eller $g(x)$ icke-linjära
  Specialfall t ex 2:a-gradspolynom med pq-formeln
- Hitta rötter till $f(x) = g(x)$ detsamma som att hitta nollställen till $f(x) - g(x)$, dvs rötter till $f(x) - g(x) = 0$

Slutsatser

- För linjära ekvationer kan man ”lösa ut” x (Gauss el. om linj ekv system) – finns ingen motsvarande ”direkt” metod för icke-linjära (i allmänhet)
- Men det finns s k iterativa metoder för att hitta nollställen, dvs för problemet $f(x) - g(x) = 0$

Tumregel:
Icke-linjära problem ⇔ iterativa metoder

Iterativ metod

- I en iterativ metod låter man x_0 vara en startgissning
- Man beräknar sedan en följd av approximationer $x_1, x_2, x_3, \cdots, x_\infty$ till den exakta lösningen x_* (som inte är känd)
- Finns olika algoritmer för att generera denna talföljd
- Om konvergens så är $x_\infty = x_*$
- I praktiken kan man aldrig nå dit utan man stannar iterationerna när viss tolerans uppfylld
- Hur bra kan det bli på en dator?

Från laborationen

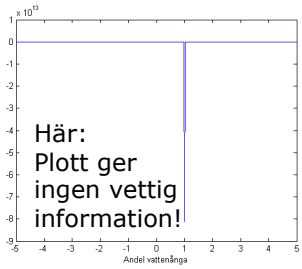
- Standardformulering av ickelinjär ekvation: $f(x) = 0$ => hitta $f(x)$:s nollställe
- Ekvationen representeras i Matlab genom att vi definierar en Matlabfunktion som beskriver $f(x)$
- Matlabs inbyggda kommando för att hitta nollställen heter **`fzero`** (**`roots`** för polynom)
- **`fzero`** hittar *ett* nollställe per anrop
- Kräver att man ger funktionen $f(x)$ och *startgissning* eller *startintervall*

Från laborationen

- Använda **`fzero`** kräver viss kunskap om problemet och var man ska leta

Vattenångauppgiften i labben…

Exempel på arbetsgång:

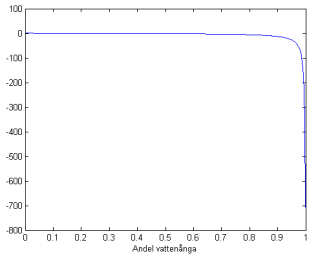


Här:
Plott ger ingen vettig information!

Från laborationen

Vattenångauppgiften i labben…

Hmm, *andel* av en viss substans kan ju bara anta värden mellan 0 och 1.
Ny plott:

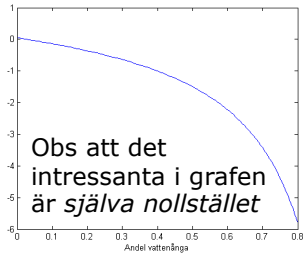


Problem för värdet x=1.
En titt i formeln visar sig att det blir division med noll. Kanske ska utesluta x=1…

Från laborationen

Vattenångauppgiften i labben…

Uteslut 1, intervall t ex mellan 0 och 0.8.
Ny plott igen:



Bättre.
Här kan man se att det finns ett nollställe någonstans nära 0

Obs att det intressanta i grafen är *själva nollstället*

Från laborationen

- Nytt kommando för plottning: **`fplot`**
- Praktisk att använda då det man ska rita är en funktionsuttryck som definieras i en Matlabfunktion
- Ritas på intervallet [a b] med kommandot
  `fplot(@func, [a b])`
- Man slipper använda **`linspace`** för att skapa x-axel och indelning ordnas automatiskt

Ickelinjära i Matlab

- Skriv först på formen $f(x) = 0$ => problemet blir att hitta nollställe
- Använd **`fzero`**

`@func` kallas för ett *funktionshandtag*
`x0` är startgissning

$f(x) = 0$

```
x = fzero(@func, x0)
```

- I matlabfunktionen **`func`** finns $f(x)$ definierad
- Går även att använda intervall som gissning

```
x = fzero(@func, [a b])
```

Ickelinjära i Matlab

- Går att få ut mer information ur **`fzero`**

```
[x, fx, exitflag] = fzero(@func, x0)
```

Ger funktionsvärdet i nollstället. Blir i teorin 0, men här litet.

Felflagga. Visar om allt gått bra eller det blivit något fel

`exitflag` — Integer encoding the exit condition

integer

Integer encoding the exit condition, meaning the reason `fsolve` stopped its iterations.

1	Function converged to a solution x.
-1	Algorithm was terminated by the output function or plot function.
-3	NaN or Inf function value was encountered while searching for an interval containing a sign change.
-4	Complex function value was encountered while searching for an interval containing a sign change.
-5	Algorithm might have converged to a singular point.
-6	`fzero` did not detect a sign change.

Vad gör fzero?

- **`fzero`** bygger på en kombination av två olika grundalgoritmer:
  - *bisektionsmetoden*
  - *Sekantmetoden*, en variant av *Newton-Raphsons metod*

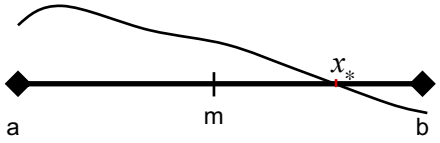
  (dessutom används invers interpolation, som vi hoppar över här)

Iterativ metod - princip

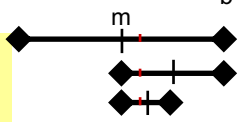
```
x = startgissning
while stoppvillkor ej uppfyllt
   x = ny gissning
end
```

- ”Ny gissning” beräknas utgående från någon formel eller princip
- Stoppvillkor en uppskattning av felet
- Kan lyckas – *konvergens …*
- …eller misslyckas – *divergens*
- Hur snabbt metoden hittar lösningen - *konvergenshastighet*

Bisektionsmetoden, idé



Idé:
- Halvera intervallet
- Välj som nytt intervall den del där teckenbyte finns
- Upprepa tills tillräckligt litet intervall



osv tills intervallet tillräckligt litet

Bisektionsmetoden, algoritm

1. Starta med intervall **`[a, b]`** så att $f(a)$ och $f(b)$ har olika tecken
2. Halvera intervallet, mittpunkten väljs som approximationen x_k
3. Välj som nytt intervall antingen vänster eller höger halva. Välj den halva där teckenbyte finns
4. Sätt **`k=k+1`** och upprepa från punkt 1 med det nya intervallet

I varje iteration ovan vet man att nollstället finns någonstans i intervallet, dvs intervallets längd är osäkerheten (felet)

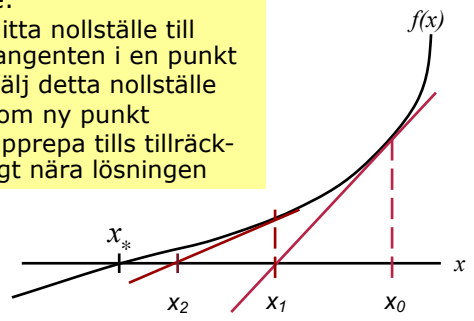
Bisektionsmetoden, kod

Indata: Startintervall **`[a, b]`** så att $f(a)$ och $f(b)$ har olika tecken, samt funktionen $f(x)$

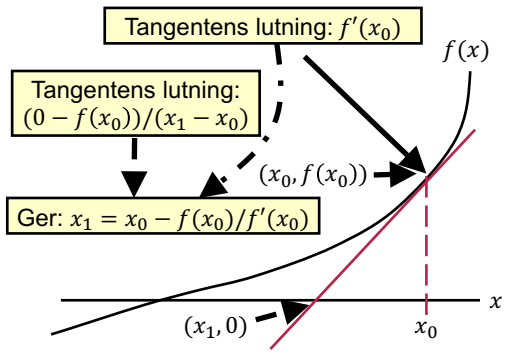
```
xl = a; xu = b
while stoppvillkor ej uppfyllt
  xm = xl + (xu-xl)/2 %mittpunkt
  if sign(f(xl)) == sign(f(xm)) then
     xl = xm %Välj höger intervall
  else
     xu = xm %Välj vänster intervall
  end
  x = xm  % som beräknat nollställe
          % väljs mittpunkten
end
```

Newton-Raphsons metod, idé

Idé:
- Hitta nollställe till tangenten i en punkt
- Välj detta nollställe som ny punkt
- Upprepa tills tillräckligt nära lösningen



Newton-Raphson – från idé till formel



Newton-Raphson som pseudokod

```
x = startgissning
while stoppvillkor ej uppfyllt
    x = x - f(x)/fprime(x)
end
```

Tänker oss att `fprime` innehåller $f'(x)$, kan även beräknas numeriskt

Newton-Raphson, ytterligare en härledning

- Taylorutveckla funktionen kring x_k och ta med termer upp till 1:a derivata
- Hitta nollstället till detta uttryck

$$f(x_k+h)=f(x_k)+hf'(x_k)+\cancel{\frac{h^2}{2}f''(x_k)+\cdots}=0$$

$$\Downarrow$$

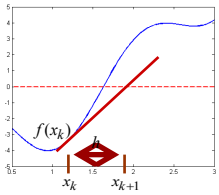
$$f(x_k)+hf'(x_k)=0$$

$$\Downarrow$$

$$f(x_k)+(x_{k+1}-x_k)f'(x_k)=0$$

$$\Downarrow$$

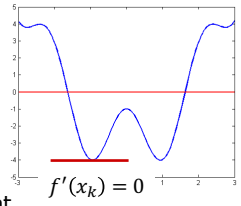
$$x_{k+1}=x_k-\frac{f(x_k)}{f'(x_k)}$$



Newton-Raphsons brister

Newton-Raphsons metod är snabb men ganska mycket kan gå fel…

- Hamnar nära eller i extrempunkt $\Rightarrow f'(x_k) = 0$ eller ≈ 0
- Svårt att förutse vilket nollställe som hittas, något x_k hamnar på en plats så att tangenten ”skjuter” iväg till ett annat nollställe
- Kan även hamna i cykliska förlopp – studsar hela tiden mellan samma sekvens av värden



Robustare: Bisektion + N-R

- Man kan visa att N-R konvergerar om startgissningen är ”tillräckligt nära” lösningen
- Hur kommer man ”tillräckligt nära”? Koppla in Bisektionsmetoden, t ex så här:

1. Bestäm ett intervall där nollstället finns
2. Kör N-R
3. **Om** konvergens så klart **annars** (t ex **N-R** hamnat utanför intervallet) koppla in Bisektion några steg för att minska intervallet
4. Gå till punkt 1 med nya intervallet

- Kombinerar N-R:s snabbhet och Bisektions säkerhet. `fzero` innehåller en mer sofistikerad algoritm men enligt samma princip.

Stoppvillkor

Hittills

```
while stoppvillkor ej uppfyllt
    …
end
```

Vilket stoppvillkor?

Stoppvillkor

- Newton-Raphson: Man brukar använda
  $$|x_{k+1} - x_k| < tol$$
  eller
  $$\frac{|x_{k+1} - x_k|}{|x_{k+1}|} < tol$$
  där *tol* ges av användaren
- Om skillnaden mellan två steg liten bör man vara nära roten
- Används även som uppskattning av felet

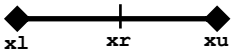
Stoppvillkor

- Bisektion (se algoritm/kod): Stanna då intervallets längd < tolerans, dvs

```
while abs(xu - xl)> tol
```

där `tol` ges av användaren

- Som lösning x_{k+1} används mittpunkten `xr`



- Obs |`xu` – `xl`| blir detsamma som $|x_{k+1} - x_k|$

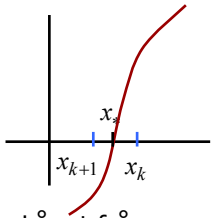
Stoppvillkor

- Vad händer om metoderna inte konvergerar? Det kanske inte finns något nollställe.
- Man hamnar i en oändlig loop eftersom stoppvillkoret aldrig uppfyllt
- Man brukar därför lägga till maximalt antal iterationer som stoppvillkor

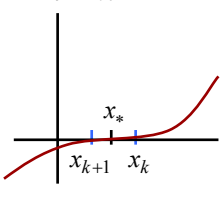
```
maxiter = 100; % t ex 100
iter = 0;
while (…)&&(iter < maxiter)
  iter = iter + 1;
        ⋮
```


Stoppvillkor

- Kan man inte använda $f(x_{k+1}) < tol$?



- Långt från x_*
- stort avstånd mellan x_{k+1} och x_k
- $f(x_{k+1}) \neq 0$



- Lika långt från x_*
- Lika stort avstånd mellan x_{k+1} och x_k
- $f(x_{k+1}) \approx 0$

Fungerar inte bra vid ”diffus” skärning med x-axeln

Konvergens

- Viktigt att metoder konvergerar snabbt, mäts med konvergenshastighet:
  $$\lim_{k\to\infty} \frac{|x_{k+1} - x_*|}{|x_k - x_*|^r} = C \quad , C \text{ konstant}$$
  - r = 1, C<1 => linjär konvergens
  - r = 2 => kvadratisk konvergens
  - r = 3 => kubisk konvergens
- Gäller ”i limes”, men kan ses när man närmar sig lösningen

Konvergens

- Tolkning:
  fel i iteration $k+1$ = C·(fel i iteration k)^r
- Har man felet i iteration k vet man alltså ungefär vad felet i nästa iteration blir
- Hur vet man felet, x_* är ju inte känd? Man använder feluppskattningen
  $|x_{k+1} - x_k|$ eller $\frac{|x_{k+1}-x_k|}{|x_{k+1}|}$

Konvergens

- Test

```
>> [x,fx,fel] = NewtonRaphson(@func1, 3);
>> [x fel fel.^2]
ans =
   3.0000   1.0000e+000   1.0000e+000
   2.3320   6.6802e-001   4.4625e-001
   1.9228   4.0920e-001   1.6744e-001
   1.7604   1.6237e-001   2.6363e-002
   1.7346   2.5795e-002   6.6536e-004
   1.7340   6.2460e-004   3.9012e-007
   1.7340   3.6208e-007   1.3110e-013
```

- Fel vid iteration $k+1$ är ungefär felet vid iteration k i kvadrat => kvadratisk konvergens hos Newton-Raphson

Konvergens

- Vilken konvergenskvot har Bisektion?
- Felet halveras varje steg, dvs

$$\frac{|\text{fel i steg } k+1|}{|\text{fel i steg } k|^1} = \frac{1}{2}$$

- Linjär konvergens med $C = \frac{1}{2}$
- Detta gör att Newton-Raphson konvergerar MYCKET snabbare (Bisektion å andra sidan säkrare)