UPPSALA UNIVERSITET

Föreläsningsanteckningar VT22

Beräkningsvetenskap

Rami Abou Zahra

Contents

1. Interpolation 2
1.1. Interpolation med polynom 2
1.2. Numerisk integration 3
1.3. Numerisk kvadratur 3
1.4. Newton-Coates kvadratur 3
1.5. Sammansatta trapetsregeln 4
2. Föreläsning 6
2.1. Trapetsmetoden i python 6
2.2. Simpsons i python 6
2.3. Observation från pythonexempel 6
2.4. Nogrannhetsordning 7
2.5. Feluppskattning 7
2.6. Richardsson-extrapolation 8
3. Förtydligande/Från boken 9
3.1. Integraler 9
3.2. Riemann integralen 9
3.3. Relationen mellan Simpsons och Trapetsapprox. 9
3.4. Analytiskt bevis för feltermen i trapetsmetoden 10
4. Föreläsning 14
4.1. Numerisk integration med scipy 14
4.2. Vad menas med absolut resp. relativt fel? 14
4.3. Funktionsfelet 14
4.4. Nogrannhet - Sammanfattning 14
4.5. Gammal tentauppgift (2009-12-21) 15
5. Icke-linjära ekvationer 16
5.1. Bisektionsmetoden/Intervallhalvering 16
5.2. Newton-Raphson 18
5.3. Gammal tentatal - 2012-04-11 19
6. Forts. Icke-linjära ekvationer 21
6.1. Konvergenshastighet 23
6.2. Konvergens i bisektionsmetoden 23
6.3. Konvergens i Newton-Raphson 23

1. Interpolation

Givet n punkter $(x_i, y_i)i \in \mathbb{N}$ där alla x_i är olika. Då vill vi anpassa en funktion $p(x)$ till detta data så att den går igenom alla dessa punkter, det vill säga $p(x_i) = y_i \forall i \in \mathbb{N}$. Detta kallas för interpolation.

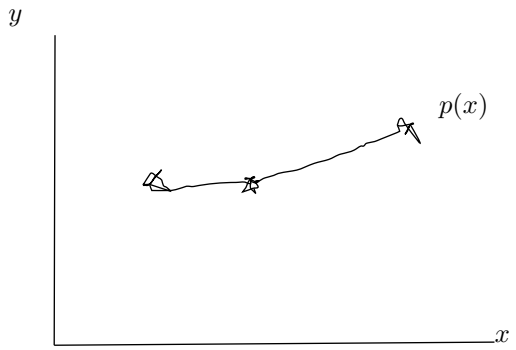


Figure 1. Interpolation idé

Vanliga användningsområden kan vara:

- Att läsa mellan raderna i en tabell, även om vi bara har data för $x = 3$ och $x = 2$ så kan vi finna värden för $x = 2.5$
- Anpassa matematisk modell till data.
- Approximera en "svår" funktion, ex. Weirstrass, med en enklare funktion.

1.1. **Interpolation med polynom.**

- Interpolera alla datapunkter med ett enda polynom:
  - $p_{n-1}(x) = \sum_{i=0}^{n-1} a_i x_i$ där $n-1$ är polynomgraden

    I allmänhet krävs det polynom av grad $n-1$ för att interpolera n punkter.

    Vi sätter in de punkter vi vill lösa i polynomet och löser för koefficienterna. Vi kommer då få ett ekvationssystem.

    Ansats med gradtal mindre än $n-1$ ger generellt ingen interpolationnslösning.

    Vid ansats med gradtal högre än $n-1$ så finns ingen unik lösning.

    Läs gärna vidare om Newton interpolation.
- Styckvis polynominterpolation:
  - Anpassar med polynom av låg grad (exvis grad 1 så att den blir linjär) på varje delintervall (mellan varje punkt).
  - $p_k^j(x)$ för $x \in [x_j, x_{j+1}]$ så att $j = 1, \cdots n-1$ och att punkterna är ordnade så att $x_j < x_j j + 1$. Vanligtvis är $k = 1$ (graden är 1, styckvis linjär) eller $k = 3$ (styckvis kubisk).
  - Vid kubisk interpolation får vi 2 styckna "frihetsgrader", det vill säga x, x^2. Dessa brukar kombineras med en metod som kallas för*spline* för att ge en kontinuerlig första och andra derivata i hela interpolationen så att kurvan blir mer *smooooothh*.
    En annan variant kallas för *pchip* som enbart ger en kontinuerlig första-deriv. Den är monoton mellan datapunkter.

1.2. **Numerisk integration.**
Problemet som vi vil lösa är att vi förståss vill beräkna integralen av en given funktion:

$$I = \int_a^b f(x)dx$$

Speciellt när det inte går att analytiskt integrera funktionen så blir det som intressantast, eller att det är väldigt svårt, eller då f-värden endast finns som mätvärden i vissa punkter.

1.3. **Numerisk kvadratur.**

$$I \approx \sum_{i=1}^{n} w_i f(x_i)$$

Där w_i är "vikter". Då är frågan hur man skall bestämma dessa vikter och vilka punkter x_i som är relevanta att ta med.
Vi gör detta baserat på de interpolerade polynomen och polynom kan vi integrera analytiskt.

1.4. **Newton-Coates kvadratur.**

Vi väljer $x_1, x_2 \cdots x_n$ värden som ekvi-distanta punkter. Sedan approximeras integralen $\int_a^b f(x)dx$ av $\int_a^b p(x)dx$ där $p(x)$ är polynomet av grad $n-1$ som interpolerar alla punkter $(x_i, f(x_i)), i = 1, \cdots, n$

Exempel: När jag bara har två punkter ($n = 2$) kallas det för *trapetsregeln*:

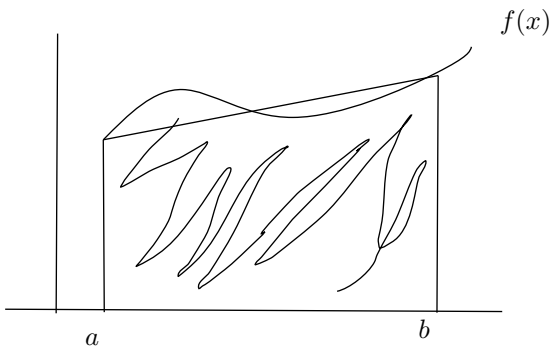


FIGURE 2. Trapetsregeln

En bättre approximering än att använda en första-gradare är simpsons formel ($n = 3$):

Det som kanske verkar rimligt kanske är att ta högre grad, men det som händer är att man får starka oscillationer eftersom ju högre grad detsto fler "kupor". Istället tar vi mindre intervall och kör bitvis linjär/simpsons istället.

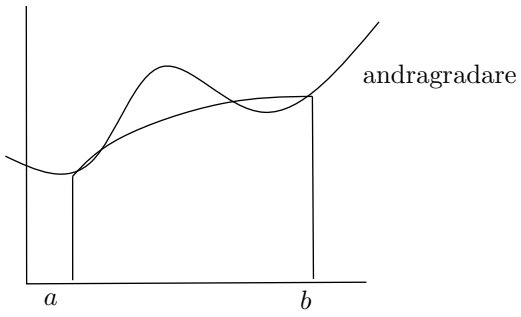


FIGURE 3. Simpsons

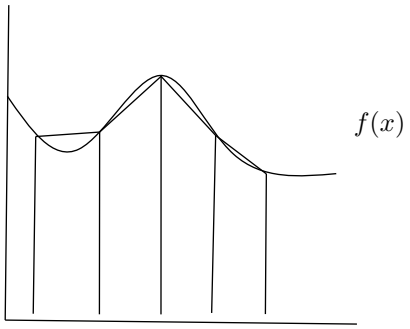


FIGURE 4. Exempel: 4 intervall

1.5. **Sammansatta trapetsregeln.**

Generella trapetsformeln:

$$x_i = a + ih, h = \frac{b-a}{n}$$

$$\int_a^b f(x)dx \approx T(h) = h\left(\sum_{i=0}^{n} f(x_i) - \frac{f(a)+f(b)}{2}\right)$$

Simpsons formel:

$$x_i = a + ih, h = \frac{b-a}{a}$$

$$\int_a^b f(x)dx \approx S(h) = \frac{h}{3}\left(f(x_0) + 4f(x_1) + 2f(x_2) + 4f(x_3) + \cdots + 2f(x_{n-2}) + 4f(x_{n-1}) + f(x_n)\right)$$

n måste vara jämn.

2. Föreläsning

Idag skall vi:

- Implementera trapetsmetoden i python
- Implementera simpsons metod i python
- Hur beror felet på h ($E(h) = I - T(h)$)?
- Är det skillnad p trapets eller simpsons?
- Nogrannhetsordning
- Resultat från anal.
- Konvergensstudie
- Feluppskattning i praktiken.
- Richardsson-extrapolation (ett sätt att förbättra nogrannheten på numeriska kvadraturen)

2.1. **Trapetsmetoden i python.**

```
import numpy as np

#function = funktionen vi vill integrera
#a = undre integral gräns
#b = övre
#n = antal delintervall

def trapets(func, a, b, n):

  h = (b-a)/n
  x = np.linspace(a, b, n+1) #Gå från a->b med n+1 punkter
  fx = func(x)
  T = h*(np.sum(fx)-(fx[0]+fx[-1])/2)

  return T
```

2.2. **Simpsons i python.**

```
import numpy as np

#function = funktionen vi vill integrera
#a = undre integral gräns
#b = övre
#n = antal delintervall

def simpsons(func, a, b, n):
  h = (b-a)/n
  x = np.linspace(a,b,n+1)
  fx = func(x)
  S = (h/3)*(fx[0]+4*np.sum(fx[1:-1:2])+2*np.sum(fx[2:-2:2])+fx[-1])

  return S
```

2.3. **Observation från pythonexempel.**

Vi noterade att i början var trapetsmetoden bättre (mindre fel) än simpsons, men att med finare delintervall så blev simpsons betydligt bättre. Senare tester verifierade följande:

- Trapetsmetoden:
  - Minskning av h med faktor 2 $\Rightarrow$ Minskning av felet med faktor 4
- Simpsons:
  - Minskning av h med faktor 2 $\Rightarrow$ Minskning av fel med faktor 16

2.4. **Nogrannhetsordning.**

Antag att felet beter sig som Ch^p där C är en konstant, h är steglängden, p är metodens nogrannhetsordning.
Låt $E(h)$ beteckna felet med steglängd h. Då har vi:

$$E(h) = Ch^p$$
$$E(2h) = C2^p h^p$$
$$\frac{E(2h)}{E(h)} = \frac{2^p Ch^p}{Ch^p} = 2^p$$
$$p = \log_2 \frac{E(2h)}{E(h)}$$

Trapetsmetoden ($\frac{E_T(2h)}{E_T(h)} = 4) \Rightarrow p_T = \log_2(4) = 2$, det vill säga nogrannhetsordning 2. Vad vi har gjort nu är uppskattat det med ändligt antal punkter, något som egentligen kräver oändligt antal, och sen har vi dragit en slutsats på detta. Vi säger att *diskretiseringsfelet* (eller *trunkeringsfelet*) är av storleken $O(h^2)$

Simpsons ($\frac{E_S(2h)}{E_S(h)} = 16) \Rightarrow p_S = \log_2(16) = 4$, det vill säga nogrannhetsordning 4 och diskretiseringsfelet $= O(h^4)$

Felet kan härledas analytiskt:

$$\text{Låt } I = \int_a^b f(x)dx$$

För trapetsmetoden: $E_T(h) = I - T(h) = -\frac{b-a}{12}h^2 f''(\xi_h)$ för något $\xi \in [a, b]$ eller $E_T(h) = Ch^2 + O(h^4)$

För simpsons: $E_S(h) = I - S(h) = -\frac{b-a}{180}h^4 f^{(4)}(\xi_h)$ för något $\xi_h \in [a, b]$

Exempelvis såg vi att $\int_0^1 x^3 dx$ blev exakt med simpsons, detta beror på att fjärdederivatan av $x^3 = 0$.

2.5. **Feluppskattning.**

Trapetsregeln ges en härledning av feluppskattningen på följande:

$$I_1 = T(h) + Ch^2 + O(h^4)$$
$$I_2 = T(2h) + C(2h)^2 + O(h^4)$$
$$I_1 - I_2 = 0 = T(h) - T(2h) + (1 - 2^2)Ch^2 + O(h^4)$$
$$Ch^2 = \frac{T(h) - T(2h)}{3} + O(h^4)$$

Uppskattning av dominerande termen i trunkeringsfelet för $T(h)$. "Tredjedelsregeln". Notera att vi tittar bara i ett ändligt antal punkter, alltså kan vi konstruera en funktion där felet ser ut att vara noll, men i verkliga fallet så kanske felet är jättestort.

2.6. **Richardsson-extrapolation.**

"Kostar" det något att beräkna detta? Nä, vi har ekvidistanta punkter! Att beräkna ena funktionsvärdet ger den andra.

$$\text{Vi stoppar in } Ch^2 = \frac{T(h) - T(2h)}{3} + O(h^4) \text{ i } I_1 :$$

$$I = T(h) + \frac{T(h) - T(2h)}{3} + O(h)$$

Övning: visa att richardsson-extrapolation tillämpat på trapetsmetoden ger simpsons

$$I_1 = Q(h) + Ch^p + O(h^{p+1})$$

$$I_2 = Q(2h) + C(2h)^p + O(h^{p+1})$$

$$Ch^p = \frac{Q(h) - Q(2h)}{2^p - 1} + O(h^{p+1})$$

3. Förtydligande/Från boken

3.1. **Integraler.**

Ibland går det inte att lösa integraler analytiskt och då kan vi använda numeriska metoder för att lösa dem. Vi kan göra detta givet hela funktionen, eller så kan vi göra detta givet punkter och där vi senare använder olika metoder för att approximera grafen så att vi kan integrera den. Det finns 2 typer av beräkningsalgoritmer:

- Öppna
  - En väldigt kraftfull typ av öppen integrering är så kallad *Gaussisk kvadratur*. Här uppskattas integralen genom att evaluera den i icke-ekvidistanta punkter och genom att ansätta en "vikt" vid varje punkt som talar om hur mycket den punkten bidrar.
- Stängda
  - Evaluerar function i ändpunkterna av det givna intervallet. Den mest grundläggande algoritmen för detta kallas för *Romberg* integrering vilket är baserad på trapetsidén som man senare har förfinat till Simpsons formula. Romberg integrering är alltså i princip Simpsons, men med lite mer logisk koppling mellan trapetsidén och Simpsons.

3.2. **Riemann integralen.**

Vi vet från envariabelanalysen att integralens definition ser ut på följande sätt:

$$I = \int_a^b f(x)dx = \lim_{\Delta x \to 0} \sum_{i=0}^{n} f(x_i)\Delta x$$

$$\Delta x = \frac{b-a}{n}$$

Detta bildar våra rektanglar som vi sedan summerar deras area över ett intervall. Ett lite bättre sätt att approximera integralen hade varit att om vi istället för att betrakta rektanglar, att vi betraktar trapetser, det vill säga man "ansluter" $f(x_i)$ och $f(x_{i+1})$. Arean av en trapets ges av $A = \frac{a+b}{2} \cdot h$. Om vi låter höjden vara Δx och a resp. b ges av funktionsvärderna får vi istället $\frac{f(x_i)+f(x_{i+1})}{2} \cdot \Delta x = \Delta A$, varpå trapetsformeln kommer ifrån.

Det visar sig att felet som uppstår i trapetsens uppskattning är proportionell mot $\Delta x^3 f''(\xi)$, där ξ är någon okänd punkt i intervallet. Denna metod för att räkna fram felet används sällan, ty man behöver finna andra-derivatan och inte nog med det så måste man även hitta när uttrycket antas sitt största värde (man utgår alltid från det värsta fallet, det vill säga att man har det högsta felet på intervallet).

3.3. **Relationen mellan Simpsons och Trapetsapprox.**

Eftersom både trapets och Simpsons evaluerar funktionen på precis samma mängd av punkter, bör det finnas något form av samband. De använder även samma geometriska härledning för att komma fram till formlerna, en där man använder parabler, och den andra använder vi trapetser. Både geometriskt och algebraiskt måste det finnas någon koppling!

Simpsons behöver som absolut minst 3 datapunkter som input. Låt oss kalla dessa f_1, f_2, f_3. Vad händer om vi stoppar in samma datapunkter i trapetsen istället, kan vi härleda något algebraiskt uttryck/samband mellan de två?

$$S_k = T_k + \frac{T_k - T_{k-1}}{3}$$

3.4. **Analytiskt bevis för feltermen i trapetsmetoden.**

Antag att vi har en funktion $f(t)$, vars integral $I = \int_a^b f(t)dt$ vi vill approximera.

Vi påstår att med hjälp av trapetsmetoden kommer feltermen vara $\frac{M(b-a)^2}{4n}$ där M är derivatans övre begränsning och n antal steglängder.

Vi påminner oss om att steglängden $\neq$ antal steglängder och att steglängden $h = \frac{b-a}{n}$.

Feltermen uttrycks av $|I - T(h)|$

Vi börjar!

Lemma 3.1: Indeldning i delintervall

Låt varje steglängd betecknas med intervallet $[x_i, x_{i+1}]$, där mittpunkten ges av $c_i = \frac{x_{i+1} + x_i}{2}$. Detta betyder att $h = x_{i+1} - x_i$. Då uttrycks feltermen för varje delintervall av:

$$\left|I - \frac{h}{2}(f(x_{i+1}) + f(x_i)\right| = \left|-(I - \frac{h}{2}(f(x_{i+1}) + f(x_i))\right| = \left|\frac{h}{2}(f(x_{i+1}) + f(x_i) - I\right|$$
$$\left|\frac{h}{2}(f(x_{i+1}) + f(x_i) - \int_{x_i}^{x_{i+1}} f(t)dt\right|$$

Och för alla delintervall är det summan av alla delintervall:

$$\sum_{i=0}^{n} \left|\frac{h}{2}(f(x_{i+1}) + f(x_i) - \int_{x_i}^{x_{i+1}} f(t)dt\right|$$

Lemma 3.2: Felterm uttryckt i integral

Vi påminner oss om partial integrering, $\int uv' = uv - \int u'v$

Vi vill försöka flytta in feltermen så att den ser ut som att den är partialt integrerad:

$$\left.\begin{array}{l}(t - c_i) = u(t)\\ f'(t) = v'(t)\end{array}\right\}$$

$$\frac{h}{2}(f(x_{i+1}) + f(x_i) = \frac{x_{i+1} - x_i}{2}f(x_{i+1}) + \frac{x_{i+1} - x_i}{2}f(x_i)$$
$$(x_{i+1} - c_i)f(x_{i+1}) + (c_i - x_i)f(x_i) = (x_{i+1} - c_i)f(x_{i+1}) + (-(c_i - x_i))f(x_i)$$
$$(x_{i+1} - c_i)f(x_{i+1}) - (c_i - x_i)f(x_i)$$

Feltermen ges alltså av $((x_{i+1} - c_i)f(x_{i+1}) - (c_i - x_i)f(x_i)) - \int_{x_i}^{x_{i+1}} f(t)dt$

$$\Leftrightarrow \int_{x_i}^{x_{i+1}} (t - c_i)f'(t)dt$$

Bevis 3.1: Felterm med första derivata

Lemma 3.1 och 3.2 ger att vi kan använda faktumet att $f'(t)$ hade en övre begränsning på M:

$$\int_{x_i}^{x_{i+1}} (t-c_i)f'(t)dt \leq \left| M \int_{x_i}^{x_{i+1}} (t-c_i)dt \right| = M \int_{x_i}^{x_{i+1}} |t-c_i|\, dt$$

$$\text{Låt } u = t - c_i, \int_{x_i}^{x_{i+1}} |t-c_i| = \int_{x_i}^{x_{i+1}} |u|$$

$$\left.\begin{aligned} f &= |u| \\ f' &= \frac{|u|}{u} g' = 1 \end{aligned}\right\}$$

$$\int_{x_i}^{x_{i+1}} |u| = ||u|\, u|_{x_i}^{x_{i+1}} - \int_{x_i}^{x_{i+1}} \frac{|u|}{u} \cdot u$$

$$\int_{x_i}^{x_{i+1}} |u| = |u\,|u||_{x_i}^{x_{i+1}} - \int_{x_i}^{x_{i+1}} |u|$$

$$2\int_{x_i}^{x_{i+1}} |u| = |u\,|u||_{x_i}^{x_{i+1}}$$

$$\Leftrightarrow \int |u| = \frac{u\,|u|}{2}$$

$$\text{Bort med } u\text{-sub: } \int_{x_i}^{x_{i+1}} |t-c_i| = \frac{(x_{i+1}-c_i)\,|x_{i+1}|}{2} - \frac{(x_i-c_i)\,|x_i-c_i|}{2}$$

$$\Leftrightarrow \frac{(x_{i+1}-c_i)\,|x_{i+1}|}{2} - \frac{(-(c_i-x_i))\,|-(c_i-x_i)|}{2} = \frac{(x_{i+1}-c_i)\,|x_{i+1}|}{2} + \frac{(c_i-x_i)\,|c_i-x_i|}{2}$$

$$\left.\begin{aligned} x_{i+1}-c_i \\ c_i - x_i \end{aligned}\right\} = \frac{x_{i+1}-x_i}{2}$$

$$\frac{2\cdot\left(\frac{x_{i+1}-x_i}{2}\right)\left|\frac{x_{i+1}-x_i}{2}\right|}{2} = \left(\frac{x_{i+1}-x_i}{2}\right)\left|\frac{x_{i+1}-x_i}{2}\right| = \frac{1}{4}(x_{i+1}-x_i)^2$$

$$\text{Nu kan vi skriva tillbaks } M \text{ framför: } \frac{M}{4}(x_{i+1}-x_i)^2$$

$$\text{Notera att } (x_{i+1}-x_i) \text{ är ju steglängden, dvs } h:$$

$$\frac{Mh^2}{4} \geq \frac{f'(t)h^2}{4}$$

□

Vi vill nu visa att om andraderivatan är så kan vi använda begränsningen för att få en ännu finare felterm.

Lemma 3.3: Partialintegrering av feltermintegral för delintervall

Vi vill partialintegrera, men vi kommer inte göra det enkelt för oss utan vi ansätter $(t - c_i)$ som den "redan deriverade" termen:

$$\int_{x_i}^{x_{i+1}} (t - c_i) f'(t) dt \Leftrightarrow \left| \frac{(t - c_i)^2}{2} f'(t) \right|_{x_i}^{x_{i+1}} - \int_{x_i}^{x_{i+1}} \frac{(t - c_i)^2}{2} f''(t) dt$$

Vi påminner oss om:

$$x_{i+1} - c_i = c_i - x_i = \frac{x_{i+1} - x_i}{2} = \frac{h}{2}$$

Insättning av detta i första partialintegrerade termen ger:

$$\frac{\left(\frac{x_{i+1} - x_i}{2}\right)^2}{2} f'(x_{i+1}) - \frac{(x_i - c_i)^2}{2} f'(x_i) = \frac{\left(\frac{x_{i+1} - x_i}{2}\right)^2}{2} f'(x_{i+1}) - \frac{(-(c_i - x_i))^2}{2} f'(x_i)$$

$$\Leftrightarrow \frac{\left(\frac{x_{i+1} - x_i}{2}\right)^2}{2} f'(x_{i+1}) - \frac{\left(\frac{x_{i+1} - x_i}{2}\right)^2}{2} f'(x_i)$$

$$\Leftrightarrow \frac{\left(\frac{x_{i+1} - x_i}{2}\right)^2}{2} (f'(x_{i+1}) - f'(x_i)) = \frac{\left(\frac{h}{2}\right)^2}{2}$$

$$= \frac{h^2}{8} (f'(x_{i+1}) - f'(x_i)) = \frac{1}{2} \left(\frac{h}{2}\right)^2 (f'(x_{i+1}) - f'(x_i))$$

Men detta är bara följande integral:

$$\frac{1}{2} \int_{x_i}^{x_{i+1}} f''(t) dt = \int_{x_i}^{x_{i+1}} \frac{1}{2} \left(\frac{h^2}{2}\right)^2 f''(t) dt$$

Då har vi alltså:

$$\int_{x_i}^{x_{i+1}} \frac{1}{2} \left(\frac{h^2}{2}\right)^2 f''(t) dt - \int_{x_i}^{x_{i+1}} \frac{(t - c_i)^2}{2} f''(t) dt$$

$$= \int_{x_i}^{x_{i+1}} \frac{1}{2} \left(\frac{h}{2}\right)^2 f''(t) - \frac{(t - c_i)^2}{2} f''(t) dt$$

$$= \frac{1}{2} \int_{x_i}^{x_{i+1}} f''(t) \left(\left(\frac{h}{2}\right)^2 - (t - c_i)^2 \right)$$

Bevis 3.2: Felterm med andraderivata

Från Lemma 3.3 påminner oss om att vi hade en begränsning M på $f''(t)$:

$$\frac{1}{2}M\int_{x_i}^{x_{i+1}}\left(\left(\frac{h}{2}\right)^2-(t-c_i)^2\right)dt$$

Vi vill lösa för integralen:

$$\begin{aligned}
\int_{x_i}^{x_{i+1}}\left(\left(\frac{h}{2}\right)^2-(t-c_i)^2\right)dt &= \int_{x_i}^{x_{i+1}}\left(\frac{h}{2}\right)^2dt-\int_{x_i}^{x_{i+1}}(t-c_i)^2dt \\
&= \left(\frac{h}{2}\right)^2[x_{i+1}-x_i]-\left|\frac{(t-c_i)^3}{3}\right|_{x_i}^{x_{i+1}} \\
&= \left(\frac{h}{2}\right)^2[x_{i+1}-x_i]-\left(\frac{(x_{i+1}-c_i)^3}{3}-\frac{(x_i-c_i)^3}{3}\right) \\
&= \left(\frac{h}{2}\right)^2[x_{i+1}-x_i]-\left(\frac{\left(\frac{h}{2}\right)^3}{3}-\frac{(-(c_i-x_i))^3}{3}\right) \\
&= \left(\frac{h}{2}\right)^2[x_{i+1}-x_i]-\left(\frac{\left(\frac{h}{2}\right)^3}{3}-\frac{-\left(\frac{h}{2}\right)^3}{3}\right) \\
&= \left(\frac{h}{2}\right)^2[x_{i+1}-x_i]-\left(\frac{h^3}{24}-\frac{-h^3}{24}\right) \\
&= \left(\frac{h}{2}\right)^2[x_{i+1}-x_i]-2\left(\frac{h^3}{24}\right) \\
&= h\left(\frac{h}{2}\right)^2-\left(\frac{h^3}{12}\right)=\frac{h^3}{4}-\frac{h^3}{12}=\frac{h^3}{6}
\end{aligned}$$

Nu när integralen är löst kan vi sätta fram koefficienterna:

$$\frac{1}{2}M\cdot\frac{h^3}{6}=\frac{Mh^3}{12}\geq\frac{h^3}{12}f''(t)$$

□

4. Föreläsning

- Adaptiva metoder
- Numerisk integration i scipy
- Relativt och absolut fel
- Funktionsfel

Vi vill göra en figur där yaxeln motsvarar felet och x_axeln motsvarar h.

4.1. **Numerisk integration med scipy.**

Givet en integral $I = \int_a^b f(x)dx$ så kan vi skriva följande kod för att evaluera integralen:

```
from scipy.integrate import quad

(q, aerr) = quad(func, a, b) # Feluppskattning |q-I| ungefär lika med aerr
#Vi kan begära viss noggrannhet:

(q, aerr) = quad(func, a, b, epsabs = atol, epsrel = rtol) #scipy försöker uppfylla kraven

|q-I| <= max(atol, rtol*|I|)
```

4.2. **Vad menas med absolut resp. relativt fel?**

Om ens chef säger "ge mig det här med 5 siffrors nogrannhet", men vad menar grabben egentligen? Säg att jag skall approximera 1 med 3 siffrors nogrannhet och min approximation är 0.9999999 (inga siffror är korrekt!):

4.3. **Funktionsfelet.**

$I = \int_a^b f(x)dx$. Vi approxierar denna med numerisk kvadratur där vi antar att vi har exakta värden på funktionen. Men vad händer om vi inte har exakta funktionsvärden, det vill säga att vi har ett fel? Det kanske kommer från mätvärden där det är fel i dem, eller något som vi alltid har, avrundningsfel. Antag att vi vet att $|f(x) - \tilde{f}(x)| \leq \varepsilon$, för sammansatta trapets/simpson gäller $\sum_{i=0}^{n} |w_i| = b - a \Rightarrow$ $\varepsilon \sum_{i=0}^{n} |w_i| = \varepsilon(b - a)$

- Diskretiseringsfelet är vanligtvis större än avrundningsfelet.
- Om funktionsvärden kommer från mätningar kan de ha stor effekt på nogrannheten.

4.4. **Nogrannhet - Sammanfattning.**

Två typer av fel

- Diskretiseringsfel (trunkeringsfel)
- Fel i funktionsvärden

$$I = \int_a^b f(x)dx \approx Q(h) \approx \tilde{Q}(h)$$

Totalt fel: $I - \tilde{Q}(h) = I - Q(h) + Q(h) - \tilde{Q}(h)$ där första 2 termerna är diskret. fel och sista 2 är funktionsfel

Vi kan använda Ye Olde' triangelolikheten:

$$|I - \tilde{Q}(h)| \leq |I - Q(h)| + |Q(h) - \tilde{Q}(h)|$$

4.5. **Gammal tentauppgift (2009-12-21).**

Om en funktion $f(x)$ vet man dels att den för vissa x-värden antar värden enligt tabellen nedan (två korrekta decimaler):

x	0.1	0.2	0.3	0.4	0.5
$f(x)$	1.89	2.07	2.89	2.18	1.74

och dels att absolutbeloppet för de 5 första derivatorna av $f(x)$ är begränsade och mindre än 19 i intervallet. Givet denna information, approximera integralen $I = \int_{0.1}^{0.5} f(x)dx$ så exakt som möjligt samt ge en strikt feluppskattning.

Det är trapets och simpsons som vi har gått igenom, och simpsons är den mest nogranna. Vad är det som gör att jag kan använda simpsons? Jo, jag har ekvidistanta punkter (alla x-värden skiljer sig med lika mycket) och jämt antal intervall:

$$S(h) = \frac{h}{3}[f(x_0) + 4f(x_1) + 2f(x_2) + 4f(x_3) + f(x_4)]$$

Men! Funktionsvärderna är inte exakta, så jag har alltså $\tilde{S}(h)$:

$$\tilde{S}(h) = \frac{0.1}{3}(1.89 + 4 \cdot 2.07 + 2 \cdot 2.89 + 4 \cdot 2.18 + 1.74) \approx 0.88033$$

Vi har 2 fel som vi skall svara på, diskretiseringsfelet och funktionsfelet. Låt oss börja med trunkeringsfelet:

$$I = \int_a^b f(x)dx = S(h) - \frac{(b-a)}{180}h^4 f^{(4)}(\xi) \text{ ej } \tilde{S}(h)$$

$$|I - S(h)| = |\frac{(b-a)}{180}h^4 f^{(4)}(\xi)| < \frac{0.5 - 0.1}{180} \cdot 0.1^4 \cdot 19 \approx 4.22 \cdot 10^{-6} = \text{ trunkeringsfelet}$$

Funktionsfelet:

$$|f(x) - \tilde{f}(x)| < 0.5 \cdot 10^-2$$

$$|S(h) - \tilde{S}(h)| \leq (b-a)\varepsilon = (0.5 - 0.1) \cdot 0.5 \cdot 10^-2$$

Totala felet:

$$|I - \tilde{S}(h)| = |I - S(h) + S(h) - \tilde{S}(h)| \leq |I - S(h)| + |S(h) - \tilde{S}(h)| \leq 4.22 \cdot 10^-6 + 0.0002 \approx 0.0002$$

$$I \approx 0.880 \pm 0.002$$

För att minska felet, vad ska man göra då? Man kan använda flera värden eller testa trapets, men notera att funktionsfelet är 100 gånger större än trunkeringsfelet, så vi behöver mer exakta värden!

Exempelvis: Hitta x då $e^x = 10\cos(x)$. Dessa typer av ekvationer finns det ingen explicit form där man kan lösa ut x. Något vi kommer göra under föreläsningen är att skriva på formen $f(x) = 0$ (detta går alltid genom att flytta över allt till en sida).

Vi använder så kallade *iterativa* metoder/algoritmer för att lösa dessa typer av icke-linjära ekvationer.

Sats 5.1: Iterativ metod

- Behöver en startgissning x_0
- Bilda en följd med bättre och bättre approximationer till den exakta lösningen x_*

Metoden konvergerar om $\lim_{k\to\infty} x_k = x_*$

I praktiken når man inte till x_* utan stannar när ett stoppvillkor uppfylls.

Implementationen av en sådan metod kallas för *algoritm*.

```
x = [START GISSNING]
while [STOPPVILLKOR EJ UPPFYLLT]:
  x = [NY APPROX]
end
```

Nya approximationen räknas från någon formel eller princip. Det kan hända att den lyckas, eller misslyckas. Lyckas den så har vi konvergens, misslyckas den så har vi divergens.

Hur snabbt metoden når konvergens/hittar lösningen är också av relevans. Kallas för *konvergenshastigheten*.

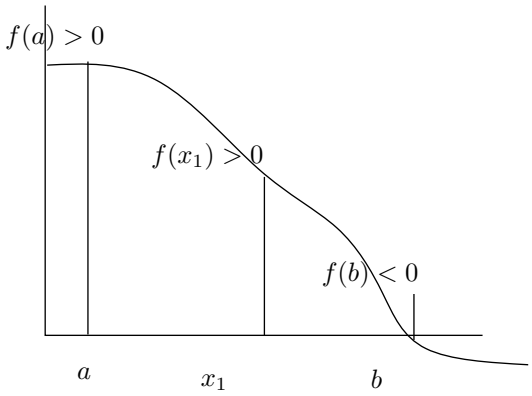
Vi kommer ta upp 2 olika metoder, bisektionsmetoden och Newton.

5.1. **Bisektionsmetoden/Intervallhalvering.**

Denna går ut på att skriva om på formen $f(x) = 0$ (detta gäller allmänt ty det är så de är implementerade, även i SciPy). Fortsätter vi på exemplet $e^x = 10\cos(x)$ får vi $f(x) = e^x - 10\cos(x)$

Idén är att börja med ett startintervall $I = [a, b]$ där vi antar att den är kontinuerlig på det intervallet och att $f(x)$ har teckenväxling på detta intervall. Satsen om mellanliggande värden säger då attden måste korsa x-axeln någonstans, så vi vill helt enkelt förfina intervallet I.

Det gör vi genom att dela intervallet i två delintervall, väl sedan den delen som har teckenväxling. Upprepa tills intervallet är tillräckligt litet. Hur stort fel man har i sin lösning är begränsat till halva intervall-längden. Vi ritar för en bättre känsla:



Vi skriver upp en pseudo-kod för detta:

```
#f,a,b givet
#Vill börja med att kontrollera input

Kontrollera att sign(f(a)) är skilt från sign(f(b))

x= (a+b)/2
while [!Stoppvillkor]:
  if sign(f(a)) = sign(f(x)):
    a = x #Om vänstra delen ej innehåller teckenväxling så förkortar vi bort VL
  else:
    b = x #Annars, förkorta HL
  x = (a+b)/2
end
```

5.1.1. *Konvergenshastighet.*

Konvergerar alltid. Men, det kan finnas flera rötter så det säger inte mycket om *vilken* rot. Hastigheten ges av:

$$|x_* - x_1| \leq \frac{b-a}{2}$$

I varje iteration halverar vi avståndet (delar på 2) alltså kommer felet halveras.

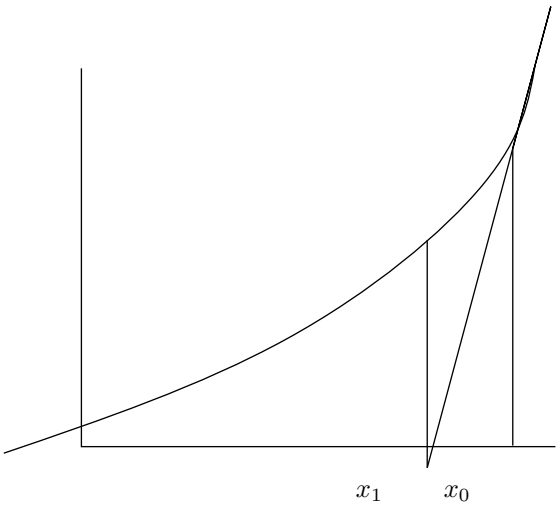
$$|x_* - x_k| \leq \frac{b-a}{2^k}$$

Nu kan vi skriva in lite saker om stoppvillkor:

Given tolerans $|x_* - x_k| \leq Tol$. Vi vet att $|x_* - x_k| \leq \frac{b-a}{2^k}$. Då vet vi att vi ska stanna när $\frac{b-a}{2^k} \leq Tol$. Då kan vi räkna ut k och därmed räkna ut antal iterationer vi behöver köra i while-loopen.

5.2. **Newton-Raphson.**

Vi skriver återigen på formen $f(x) = 0$. Vi har någon startgissning x_0 (ej intervall, utan värde). Idén är att vi drar en tangent till funktionen och nästa gissning blir roten/nollställe till tangenten. Illustrerat:



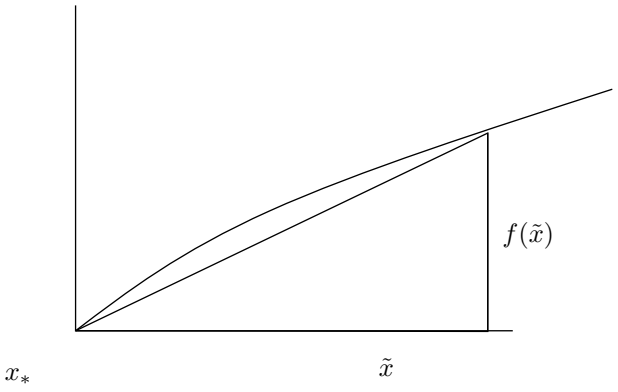
Tangenten ges av $l(x) = f'(x_0)(x - x_0) + f(x_0)$. Vi löser för när $l(x_1) = 0 \Rightarrow f'(x_0)(x_1 - x_0) + f(x_0) = 0 \Rightarrow x_1 = x_0 - \dfrac{f(x_0)}{f'(x_0)}$

Låt oss skriva en pseudo-kod:

```
x = startgissning
while (!stoppvillkor):
  dx = -f(x)/f'(x)
  x = x+dx
end
```

5.2.1. *Stoppvillkor.*

Säg att vi har en approximativ lösning $\tilde{x}$ och x_* som är exakt. Då är vi intresserade av $|x_* - \tilde{x}|$. Vi utnyttjar att om vi zoomar in så blir det en triangel:



Vi får då:

$$f'(\tilde{x}) \approx \frac{f(\tilde{x})}{\tilde{x} - x_*}$$
$$\Rightarrow |x_* - \tilde{x}| \approx \left| \frac{f(\tilde{x})}{f'(\tilde{x})} \right| = |\Delta x|$$

Alltså stanna då $|x_{k+1} - x_k| < Tol$

Då kan man fråga, "men farbror Melker, kan man inte använda $|f(x_k)| < Tol$?" Nej, tänk om vi har asymptoter.

- Vertikal asymptot ger att
  - $|x_* - x_{k+1}|$ är stort
  - $|f(x_{k+1})|$ är stort
- Horisontell asymptot ger att
  - $|x_* - x_{k+1}|$ är stort
  - $|f(x_{k+1})|$ är litet

När man skriver while loopar skall man vara aktsam över att man kanske inte konvergerar. Då får man en oändlig loop eftersom stoppvillkoret aldrig uppfylls. Då är det smart att lägga in ett max antal iterationer:

```
maxiter = 100
niter = 0
while felet > tol && niter < maxiter
  niter = niter+1
end
```

5.3. **Gammal tentatal - 2012-04-11.**

Ekvationen $\cos(x) + 5 = e^x$. Den har en lösning mellan $[1, 2]$

- Formulera intervallhalveringsmetoden för att hitta roten och gör 3 iterationer
- Hur många iterationer skulle krävas för att få ett absolut fel i det beräknade värdet mindre än 10^{-8}?

Vi börjar med första. Vi börjar givetvis med att skriva om på $f(x) = 0$:

$$f(x) = \cos(x) + 5 - e^x = 0$$

Vi kan nu kontrollera startinervallet $[1, 2]$:

$$f(1) = \cos(1) + 5 - e > 0$$
$$f(2) = \cos(2) + 5 - e^2 < 0$$

Sartintervallet är ok, ty jag har olika tecken. Mittpunkten är 1.5:

$$f(\frac{1+2}{2}) = f(1.5) \approx 0.6 > 0$$

Vi vill ha delen med teckenväxling, så vi väljer "högra" delen. Mittpunkten är 1.75:

$$f(1.75) \approx -0.93 < 0$$

Vi väljer "vänstra" delen:

$$f(1.625) \approx -0.13 < 0$$

Mittpunkten mellan 1.5 och 1.625 är 1.5625. Vi har nu kört 3 iterationer som frågan frågar efter, så vi uppskattar felet:

$$|x - x_*| \leq \frac{b-a}{2} = \frac{1.625 - 1.5}{2} = 0.0625$$

Svaret blir $x = 1.5625 \pm 0.0625$

Vi kör andra frågan:

Låt ε_k vara feluppskattning efter k iterationer. Då har vi i detta fall:

- $\varepsilon_0 = \frac{b-a}{2} = \frac{2-1}{2} = 0.5$
- $\varepsilon_1 = \varepsilon_0 \cdot 0.5 = 0.25$
- $\varepsilon_2 = \varepsilon_1 \cdot 0.5 = \varepsilon_0 \cdot 0.5^2$
- $\varepsilon_k = 0.5^k \varepsilon_0 = 0.5^{k+1}$

Vi vill hitta $\varepsilon_k \leq 10^{-8}$:

$$0.5^{k+1} < 10^{-8}$$
$$(k+1)\log(0.5) < \log(10^{-8})$$
$$k > \frac{\log(10^{-8})}{\log(0.5)} - 1 \approx 25.6$$

Eftersom vi ska ha strikt norgannhet så rundar vi uppåt, så det skulle alltså krävas 26 iterationer.

6. Forts. Icke-linjära ekvationer

Vi börjar med exemplet $e^x = 10\cos(x) \Leftrightarrow e^x - 10\cos(x)$. Vi ska göra koden:

```
import numpy as np
import matplotlib.pyplot as plt

def bisection(func, a, b, tol):
    # sign_fa = np.sign(func(a))
    # sign_fb = np.sign(func(b))

    x = (a+b)/2
    itercounter = 0
    sign = np.sign(func(a)) == np.sign(func(b))

    assert(not sign)

    while((b-a)/2) > tol:

        itercounter += 1
        localSign = np.sign(func(x))

        if sign == localSign:
            a = x
        else:
            b = x

      x = (a+b)/2
    return x, itercounter
```

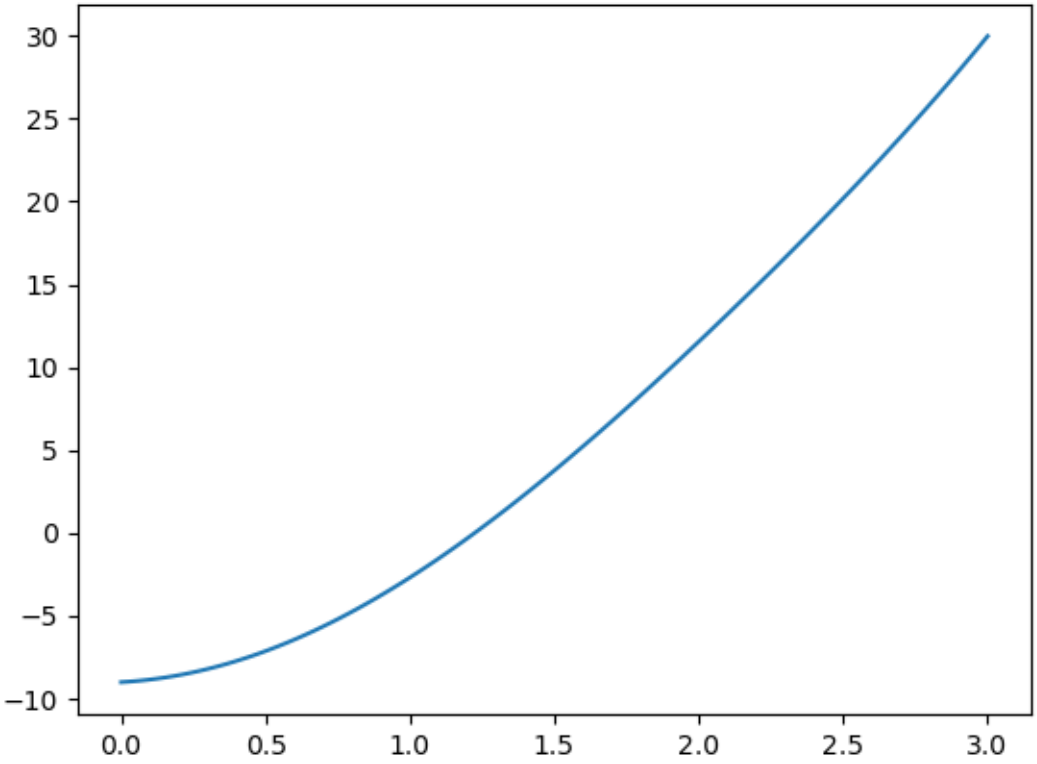
Hitta en positiv lösning ($x > 0$) till ekvationen med 8 korrekta decimaler.

- Använd bisection
- Använd Newton-Raphson

Man börjar givetvis alltid med att skriva om på formen $f(x) = e^x - 10\cos(x)$. Vi behöver också ett startintervall/startgissning, men hur ska man hitta det? Ett sätt är att fundera över hur funktionen ser ut. Ett annat sätt är att plotta funktionen (beror lite på det specifika fallet man undersöker).
Vi ser att $f(0) = 1 - 10 = 9 < 0$ och $f(\frac{\pi}{2}) = e^{\pi/2} - 0 > 0$.
. Vi testar att plotta vår funktion:

```
func = lambda x: np.exp(x)-10*np.cos(x)
```

Här ser vi en rot mellan 1 och 1.5:


30
25
20
15
10
5
0
−5
−10
0.0
0.5
1.0
1.5
2.0
2.5
3.0

Kör vi koden får vi att den konvergerar mot ≈ 1.2 med 26 iterationer. Implementerar vi Newtons metod och använder startvärde 1.25 får vi konvergens med inget mindre än 4 iterationer! Givetvis måste vi dock räkna derivator.

Slutsats: Newton är mycket snabbare.
Frågan är varför? Kan det bero på likformig kontinuitet eller strikt monoton?

6.1. **Konvergenshastighet.**

Låt x_* vara vår exakta lösning och vi antar att vi har en iterativ metod som generar en talföljd $x_1, x_2, \cdots$. Antag även att vi har konvergens, dvs $\lim_{k\to\infty} x_k = x_*$.

Sats 6.1: Konvergensordning

En metod har *konvergensordning* r om $\lim_{j\to\infty} \dfrac{|x_* - x_{k+1}|}{|x_* - x_k|^r} = C$ där C (konstant) kallas för *asymptotisk felkonstant*

Notera att ju större r är, desto snabbare konvergerar den. Om:

- $r = 1, C < 1$ har vi *linjär konvergens*
- $r = 2$ har vi *kvadratisk konvergens*
- $r > 1$ har vi *superlinjär konvergens* (Busigt att använda $r > 1$)

6.2. **Konvergens i bisektionsmetoden.**

Felet halveras i varje steg. Vi kommer ihåg att vi har $|x_* - x_k| \leq \dfrac{B-A}{2^k}$ där $[A, B]$ är startintervallet. Det nya felet kommer då vara $\dfrac{1}{2}$ gamla felet, därför har vi $r = 1$ och $C = \dfrac{1}{2}$ det vill säga linjär konvergens.

6.3. **Konvergens i Newton-Raphson.**

Påminner oss om metoden: $x_{k+1} = x_k - \dfrac{f(x_k)}{f'(x_k)}$. Vi Taylorutvecklar kring x_k med steget $(x_* - x_k)$. sedan har vi $\underbrace{f(x_k + (x_* - x_k))}_{f(x_*) = 0} = f(x_k) + (x_* - x_k)f'(x_k) + (x_* - x_k)^2 f''(\xi) \cdot \dfrac{1}{2}$

Omskrivning ger att $x_k = x_{k+1} + \dfrac{f(x_k)}{f'(x_k)}$, vi stoppar in x_k i Taylorutvecklingen:

$$f(x_k) + (x_* - x_{k+1} - \frac{f(x_k)}{f'(x_k)})f'(x_k) + (x_* - x_k)^2 \frac{f''(\xi)}{2} = 0$$

$$\Leftrightarrow 0 = (x_* - x_{k+1})f'(x_k) + (x_* - x_k)^2 \frac{f''(\xi)}{2}$$

$$(x_* - x_{k+1}) = -(x_* - x_k)^2 \frac{f''(\xi)}{2f'(x_k)}$$

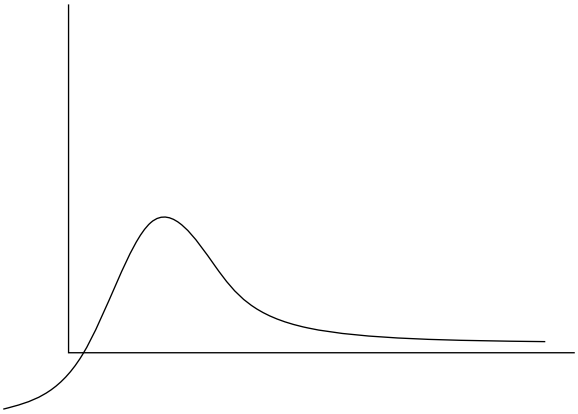
$$\frac{x_* - x_{k+1}}{(x_* - x_k)^2} = -\frac{f''(\xi)}{2f'(x_k)} \text{ här antas konvergens och } f'(x_k) \neq 0$$

$$\Leftrightarrow \lim_{x_k \to x_*} \frac{|x_* - x_{k+1}|}{|x_* - x_k|^2} = \frac{|f''(x_*)|}{2\,|f'(x_*)|}$$

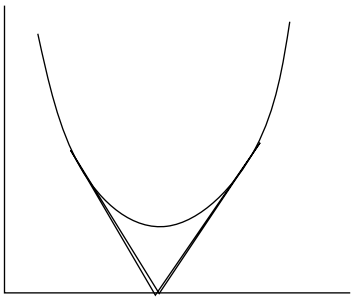
Notera att vi har kvadratisk konvergens. Det nya felet ges av $C \cdot (Fel)^2$

6.3.1. *Komplikationer med Newton-Raphson.*

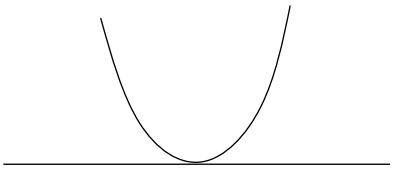
Det första som kan hända är divergens:



Alternativt:



Något annat som kan hända är om vi har multipla rötter:



Exempelvis $f(x) = x^2$, $f'(x) = 2x$, $f''(x) = 2$. Från analys:

$$x_* - x_{k+1} = -(x_* - x_k)^2 \frac{f''(\xi)}{2f'(x_k)}$$

$$\text{Låt } x_* = 0 \Rightarrow -x_{k+1} = -x_k^2 \frac{2}{2 \cdot 2x_k} = -\frac{1}{2}x_k$$

Detta blir dock linjär konvergens och vi har förlorat det som var så schysst med Newton-Raphson. Generellt för dubbelrötter så är $C = \lim_{k\to\infty} \dfrac{|x_* - x_{k+1}|}{|x_* - x_k|} = \dfrac{1}{2}$. Här är givetvis $r = 1$