UPPSALA UNIVERSITET

Föreläsningsanteckningar VT22

Beräkningsvetenskap

Rami Abou Zahra

Contents

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.

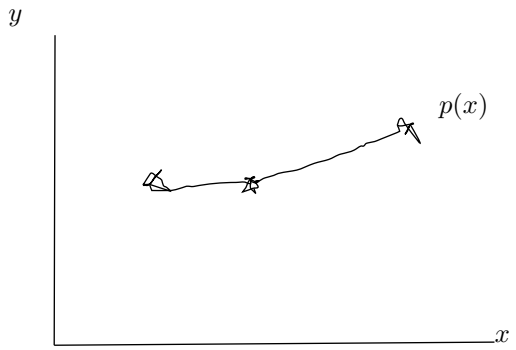

Figure 1. Interpolation idé

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*:

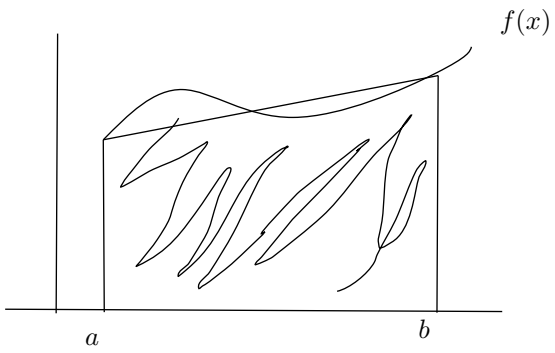

FIGURE 2. Trapetsregeln

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.

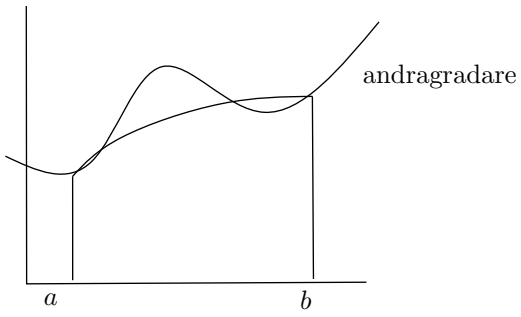

FIGURE 3. Simpsons

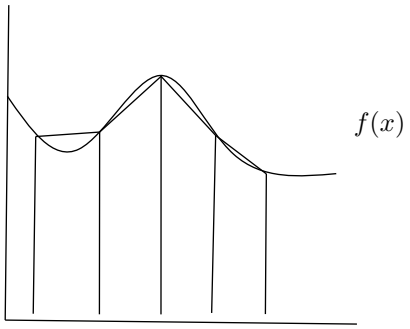

FIGURE 4. Exempel: 4 intervall

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

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

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

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