TMV035 Analysis and Linear Algebra B[1]

Integralen

1.1 Riemannsumma

I Adams kap 5 definieras (konstrueras) integralen $\int_a^b f(x)\,dx$ med hjälp av Riemannsumman

$$\sum_{i=1}^{n} f(c_i)h_i$$

där vi har gjort en partition av intervallet

$$a = x_0 < x_1 < x_2 < \cdots < x_{i-1} < x_i < \cdots < x_n = b$$

med steg

$$h_i = \Delta x_i = x_i - x_{i-1}$$

och där c_i är en godtycklig punkt i intervallet $[x_{i-1}, x_i]$. Integralen är det unika gränsvärdet

$$\int_a^b f(x)\,dx = \lim_{\substack{n\to\infty \\ \max h_j \to 0}} \sum_{i=1}^{n} f(c_i)h_i$$

Att det är unikt innebär att man får samma gränsvärde oberoende av hur man väljer partitionerna och hur man väljer c_i.

1.2 Rektangelregeln

Riemannsumman är alltså en numerisk approximation av integralen. Att beräkna integralen numeriskt kallas *numerisk kvadratur* (numerical quadrature) och en algoritm för numerisk kvadratur brukar kallas *kvadraturregel* (quadrature rule). Namnet kvadratur syftar på areaberäkning, dvs att finna en kvadrat som har samma area som en given yta i planet.

Riemannsumman kallas även *rektangelregeln* därför att varje term i summan är arean av en rektangel med basen h_i och höjden $f(c_i)$, räknad med tecken.

Om vi väljer c_i som den vänstra ändpunkten av det aktuella intervallet, dvs $c_i = x_{i-1}$, och tar konstant steg $h_i = h$, får vi

$$\int_a^b f(x)\,dx \approx \sum_{i=1}^{n} f(x_{i-1})h$$

Denna summa kan beräknas med en loop:

$$\text{initiera: } \begin{cases} x_0 = a \\ Q = 0 \end{cases}$$

$$\text{uppdatera: } \begin{cases} \text{while } x_i < b \\ x_i = x_{i-1} + h \\ Q = Q + hf(x_{i-1}). \end{cases}$$

Obs att detta är detsamma som Eulers metod som vi använde för att beräkna den primitiva funktionen i programmet `myprim.m`. Detta är ingen tillfällighet eftersom enligt Integralkalkylens fundamentalsats vet vi att den primitiva funktionen ges av integralen:

$$F(x) = \int_a^x f(y)\,dy, \quad \text{så att } \int_a^b f(y)\,dy = F(b).$$

[1] 2006-12-12 /stig

Andra möjliga val av c_i är högra ändpunkten, $c_i = x_i$, eller mittpunkten, $c_i = (x_{i-1} + x_i)/2$. Den senare metoden kallas *mittpunktsregeln*. Vi har redan sett att mittpunktsregeln ger snabbare konvergens.

Skriv ett program `minintegral.m` med anropet `Q=minintegral(f,I,h,k)` som beräknar integralen approximativt med rektangelregeln. Anänd ditt gamla program `myprim.m` som utgångspunkt. Invariablerna `f,I,h` har samma mening som där. Variabeln `k` skall användas för att välja metod enligt:

$$k = \begin{cases} 1, & \text{rektangelregeln vänster} \\ 2, & \text{rektangelregeln höger} \\ 3, & \text{mittpunktsregeln} \end{cases}$$

Testa programmet på några integraler som du även kan beräkna analytiskt.

1.3 Trapetsregeln

Adams kap 6.6. Om vi ersätter rektangelarean $f(c_i)h$ med en trapetsarea, dvs om vi approximerar integralen med

$$\int_{x_{i-1}}^{x_i} f(x)\,dx \approx h\frac{f(x_{i-1}) + f(x_i)}{2}$$

så får vi ett alternativ till mittpunktsregeln. Summering av bidragen ger

$$\int_a^b f(x)\,dx \approx \sum_{i=1}^{n} h\frac{f(x_{i-1}) + f(x_i)}{2}.$$

Denna algoritm kallas *trapetsregeln* (trapezoidal rule).

Bygg in denna metod i ditt program som alternativ $k = 4$.

1.4 Effektivare program

Att använda en loop är mycket ineffektivt i MATLAB. Skriv om ditt program så att det först genererar en vektor av alla funktionsvärdena $f(x_i)$ och sedan summerar dessa (på korrekt sätt) med hjälp av MATLAB-funktionen `sum`.

Till exempel, på kommandoraden kan man skriva

```
>> x=linspace(0, 1-0.001, 1000);
>> y=sin(x);
>> q=sum(y)*0.001;
```

vilket ger rektangelregeln för $\int_0^1 \sin(x)\,dx$ med $h = 0.001$. Men du skall naturligtvis skriva något dylikt i filen `minintegral.m`.

Detta sätt att organisera en beräkning kallas att vektorisera den, dvs man genererar först en eller flera vektorer och utför sedan den önskade beräkningen på dem. De algebraiska "prick" operationerna

```
.*  ./  .^
```

är exempel på vektoriserade operationer. Här använder vi funktionen `sum` som snabbt summerar en vektor. Likaså kan man tillämpa andra funktioner på vektorer, till exempel, `y=sin(x)` ger en vektor av y-värden om x är en vektor.

1.5 Matlabs kvadraturfunktioner

MATLAB har några egna funktioner för numerisk kvadratur, till exempel, `quad` och `quadl`. Läs om dem i dokumentationen och prova dem på samma integraler som ovan.