Programmierpraktikum

Claudius Gros, SS2012

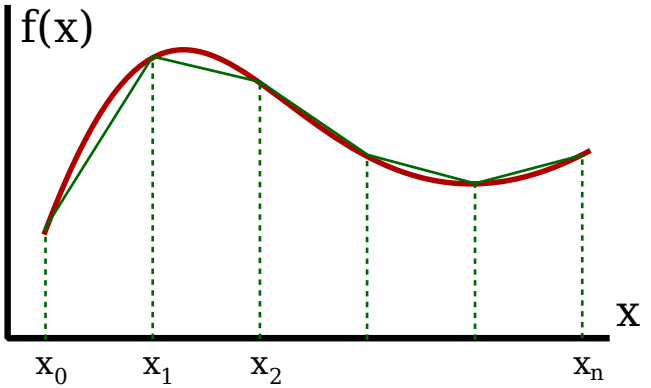
Institut für theoretische Physik
Goethe-University Frankfurt a.M.

Numerical Evaluation of Integrals

numerical integration

- well behaved integrand

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



- trapezoidal rule

$$T_n \equiv \sum_{k=1}^{n} \frac{f(x_{k-1}) + f(x_k)}{2}\,\Delta x$$

- $x_0 = a,\ x_n = b$
  $\Delta x = x_n - x_{n-1}$ constant
- convergence with the number n of trapezoids

$$lim_{n\to\infty}\, T_n = I$$

trapezoidal rule at work

$$y_m = \int_0^1 \frac{x^m}{x+a}\, dx$$

- $a = 10, m = 19, n = 2^p$

n	p	T(p)	(4T(p)-T(p-1)/3
1	0	0.0454545454545455	
2	1	0.0227273635533981	0.0151516362530156
4	2	0.0114620139299330	0.0077068973887780
8	3	0.0066417103644638	0.0050349425093074
16	4	0.0051140844181988	0.0046048757694438
32	5	0.0047045044781142	0.0045679778314194
64	6	0.0046002267297100	0.0045654674802419
128	7	0.0045740370560200	0.0045653071647900
256	8	0.0045674820820493	0.0045652970907257
512	9	0.0045658428656951	0.0045652964602438
1024	10	0.0045654330320428	0.0045652964208253
2048	11	0.0045653305717818	0.0045652964183615
4096	12	0.0045653049566010	0.0045652964182075
8192	13	0.0045652985527986	0.0045652964181978
16384	14	0.0045652969518476	0.0045652964181972
		^	^
		9	16

- inverse recursion ($a = 10, m = 19$): 0.0045652964181972

- convergence acceleration with

$$T_p^* \;=\; \frac{1}{3}\,\Big(4\,T(p) - T(p-1)\Big)$$

for $n = 2^p$. Why?

convergence scaling

- **scaling of accuracy as a function of effort**
  accuracy of trapezoidal approximation improves quadratically with the number of trapezoids

$$I = \int_a^b f(x)\,dx \approx T_n + O(n^{-2})$$

$$T_n = \frac{\Delta x}{2} \sum_{k=1}^{n} \Big(f(x_{k-1}) + f(x_k) \Big), \qquad \Delta x = \frac{b-a}{n}$$

- can be derived via recursive partial integration

$$\int_{x_{k-1}}^{x_k} 1 \cdot f(x)\,dx = (x+c_1) f(x) \Big|_{x_{k-1}}^{x_k} - \int_{x_{k-1}}^{x_k} dx\,(x+c_1)\,\frac{d}{dx} f(x)$$

valid for well behaved (analytic) integrands

leading to the Euler–Maclaurin formula

convergence acceleration

- exploit scaling for performance improvements

$$T_n = I + \frac{c}{n^2} + O(n^{-4}), \qquad T_{2n} = I + \frac{c}{4n^2} + O(n^{-4})$$

$$\frac{4}{3} T_{2n} - \frac{1}{3} T_n = \left(\frac{4}{3} - \frac{1}{3}\right) I + \left(\frac{4}{3}\,\frac{1}{4} - \frac{1}{3}\right) \frac{c}{n^2} + O(n^{-4})$$

- qualitative convergence improvement

$$\frac{4}{3} T_{2n} - \frac{1}{3} T_n = I + O(n^{-4})$$

  at essentially constant numerical effort

- generalization to Romberg's method

- how can we speed an algorithm scaling like n^{-z} , for any $z \leq 1$?

accelerated integration

- simple to use, never hurts
- quadratic convergence accleration

```
 1. /** Integrating
 2.   * y_m = \int_0^1 x^m/(x+a)dx
 3.   * for various m and n of the trapezoidal integration method
 4.   */
 5. public class InteTrapezoidal {
 6.
 7. public static void main(String[] args) {
 8.   int    m = 19;
 9.   double a = 10.0;
10. // *** ************************** ***
11. // *** loop of numbers of trapzoids ***
12. // *** ************************** ***
13.   double Tn = 0.0;
14.   double lastTn = 0.0;
15.   int     n = 1;
```

```
16.  System.out.printf("%8s %5s %22s %22s\n",
17.                    "n","p","T(p)","(4T(p)-T(p-1)/3");
18.   for (int jj=0; jj<15; jj++)
19.     {
20.     Tn =  sumTrapezoids(n, m, a);
21.     if (jj==0)
22.       System.out.printf("%8d %5d %22.16f\n",
23.                         n, jj, Tn);
24.     else
25.       System.out.printf("%8d %5d %22.16f %22.16f\n",
26.                         n, jj, Tn,(4*Tn-lastTn)/3.0);
27.     n = n*2;
28.     lastTn = Tn;
29.     }
30.   System.out.println(" ");
31. }  // end of InteTrapezoidal.main()
32.
33. /** Straight summing trapezoids
34.   */
35. public static double sumTrapezoids(int n, int m, double a) {
36.   double result = 0.0;
37.   double x0, x1;
38.   double DeltaX = 1.0/n;
```

```
39.   for (int i=0; i<n; i++)
40.     {
41.     x0  = i*DeltaX;
42.     x1 = (i+1)*DeltaX;
43.     result +=  0.5*( Math.pow(x0,1.0*m)/(x0+a)
44.                    + Math.pow(x1,1.0*m)/(x1+a) );
45.     }
46.   return result*DeltaX;
47.   }    //  end of summationUP
48.
49. }    //  end of class InteTrapezoidal
```