Lecture `quad3`: Numerical Quadrature: Composite Rules

November 28, 2022

Summary: Derivation and implementation of composite quadrature rules. Optimization of function evaluations. Declaration and verification of error bound.

References: Material here taken mostly from C. F. Van Loan's *Introduction to Scientific Computing*. See also T. Sauer's *Numerical Analysis*, Section 5.2, 1st ed. 253–263, 2nd ed. 254–265.

1 Comments

Last time we wrote down (without proof) an error estimate corresponding to the m–point closed Newton–Cotes quadrature rule,

$$\left| \int_a^b f(x)dx - Q_{NC(m)} \right| \le |c_m| M_{d+1} \left(\frac{b-a}{m-1} \right)^{d+2}, \qquad d = \begin{cases} m-1 & \text{if } m \text{ even} \\ m & \text{if } m \text{ odd.} \end{cases} \tag{1}$$

Again, c_m is a small constant, and M_{d+1} is a uniform bound on the $(d+1)$st derivative $f^{(d+1)}(x)$ of $f(x)$. See page 142 of Van Loan for the values of c_2 through c_{11}; here we note

$$c_2 = -\frac{1}{12}, \quad c_3 = -\frac{1}{90}, \quad c_4 = -\frac{3}{80}, \quad c_5 = -\frac{8}{945}, \quad c_6 = -\frac{275}{12096}. \tag{2}$$

Therefore, the formula is valid if the function $f(x)$ is $d+1$ times continuously differentiable on the interval, and we have found the bound $|f^{(d+1)}(x)| \le M_{d+1}$, which must hold independent of $x \in [a, b]$. The estimate then tells us that $Q_{NC(m)}$ is a good approximation to the integral, provided $b-a$ is small enough. This lecture considers what to do when $b-a$ is not small enough.

Here is another way you might see the error formula for Simpson's rule $Q_{NC(3)}$ written

$$\int_{x_0}^{x_2} f(x)dx = \frac{h}{3}(y_0 + 4y_1 + y_2) - \frac{h^5}{90} f^{(iv)}(c), \tag{3}$$

where $x_2 - x_1 = h = x_1 - x_0$ and c is between x_0 and x_2. Let's compare this to our formula above. First, write the last formula as

$$\int_{x_0}^{x_2} f(x)dx - Q_{NC(3)} = -\frac{h^5}{90} f^{(iv)}(c) \implies \left| \int_{x_0}^{x_2} f(x)dx - Q_{NC(3)} \right| \le |c_3| M_4 h^5, \tag{4}$$

with c_3 and M_4 as above. Now use $h = (b-a)/2 = (b-a)/(3-1)$ to complete the comparison.

2 Derivation of composite rules

If we have a partition of the interval $[a, b]$,

$$a = z_1 < z_2 < \cdots z_{n+1} = b, \tag{5}$$

then we can use the additivity property of the integral to write

$$\int_a^b f(x)dx = \sum_{i=1}^{n} \int_{z_i}^{z_{i+1}} f(x)dx. \tag{6}$$

That is to say, we can split an integral over a long interval into a collection of integrals, each over a shorter interval. The idea is to exploit this property and apply $Q_{NC(m)}$ to each of the individual integrals in the sum. Furthermore, we assume that each subinterval $[z_i, z_{i+1}]$ is of a sufficiently small length. This assumption ensures that the Newton–Cotes rule gives an accurate approximation for each piece of the integral.

The choice (5) of partition is quite arbitrary, and *adaptive integration* relies on this freedom. We'll get to that subject in a later lecture. Today, we'll assume the partition is equally spaced, that is

$$\Delta = (b-a)/n, \quad z_i = a + (i-1)\Delta, \quad i = 1, \cdots, n+1. \tag{7}$$

In Matlab the composite rule might then be expressed as (Van Loan, page 145)

```
Q = 0;
Delta = (b-a)/n;
for i = 1:n
    Q = Q + ClosedQNC(@f,a+(i-1)*Delta,a+i*Delta,m);
end
```

where `f` is a handle for a function $f(x)$ and `ClosedQNC` is our m–point closed Newton–Cotes rule (described last time, this is Van Loan's `QNC`). We could of course do the same thing with `OpenQNC`. Van Loan refers to the closed composite rule as $Q_{NC(m)}^{(n)}$, which signifies applying the standard m–point closed rule $Q_{NC(m)}$ to n subintervals.

3 Function evaluations

The composite algorithm just considered has one flaw, when based on `ClosedQNC`. Namely, at all interior points z_k of the partition, the function $f(x)$ determined by `fname` is evaluated twice. This might prove expensive, if the function is difficult to evaluate. For example, see Fig. 1 which depicts an interval $[a, b]$ divided into 10 subintervals, each of width Δ. According to the figure, we plan to use (the closed rule) $Q_{NC(3)}$ on each subinterval. Recall, that $Q_{NC(3)}$ has weights `w = [1,4,1]/6`. As it stands the partition is comprised of points $z_1, z_2, \cdots, z_{11}$. Introducing the notation $z_{3/2}$ for, say, the midpoint between z_1 and z_2, the composite rule as written above is

$$\begin{aligned} Q_{NC(3)}^{(10)} &= \tfrac{1}{6}\Delta\big[f(z_1) + 4f(z_{3/2}) + f(z_2)\big] \\ &+ \tfrac{1}{6}\Delta\big[f(z_2) + 4f(z_{5/2}) + f(z_3)\big] \\ &+ \tfrac{1}{6}\Delta\big[f(z_3) + 4f(z_{7/2}) + f(z_4)\big] \\ &+ \cdots \\ &+ \tfrac{1}{6}\Delta\big[f(z_9) + 4f(z_{19/2}) + f(z_{10})\big] \\ &+ \tfrac{1}{6}\Delta\big[f(z_{10}) + 4f(z_{21/2}) + f(z_{11})\big], \end{aligned} \tag{8}$$

where it's obvious that, for example, $f(z_2)$ is computed twice. Clearly, each of the evaluations $f(z_2), f(z_3), \cdots, f(z_{10})$ are twice computed, and that's the wasteful aspect of the algorithm. Note that such double evaluations will not occur if the above algorithm is based on `OpenQNC`, as may be understood from Fig. 2. So one easy way to sidestep this issue of double function evaluations is to switch to the open rules, again by simply replacing `ClosedQNC` with `OpenQNC` in the code fragment above.

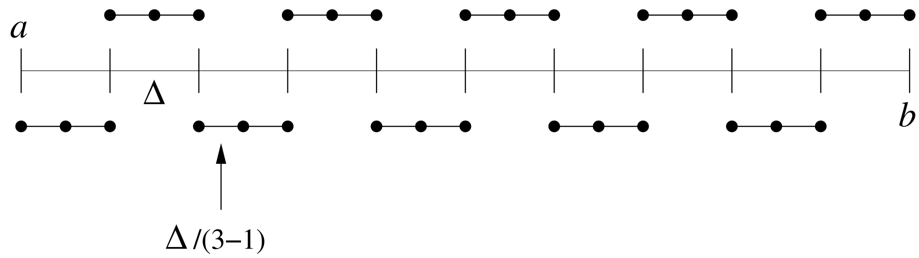

Figure 1: Composite rule based on 3–point closed Newton–Cotes rule with 10 subintervals.

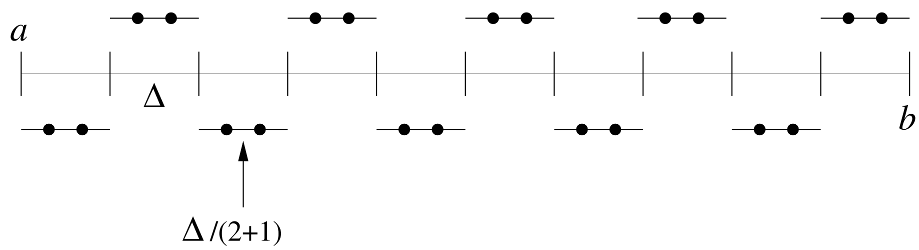

Figure 2: Composite rule based on 2–point open Newton–Cotes rule with 10 subintervals.

To avoid overly many function evaluations while still using a closed rule, we may combine like terms, rewriting the last equation (8) as

$$\begin{aligned} Q^{(10)}_{NC(3)} =& \tfrac{1}{6}\Delta\big[f(z_1)+4f(z_{3/2})+2f(z_2)+4f(z_{5/2})+2f(z_3) \\ &+4f(z_{7/2})+2f(z_4)+\cdots+2f(z_{10})+4f(z_{21/2})+f(z_{11})\big]. \end{aligned} \tag{9}$$

This composite rule is more easily expressed in terms of a finer partition. The partition z_k corresponds to n subintervals, where $n = 10$ in our example above. Then, also as above, $\Delta = (b-a)/n$. Now we introduce $h = \Delta/(m-1)$, where m specifies which closed rule $Q_{NC(m)}$ where using on each interval ($m = 3$ for the example above). We introduce a finer partition $a = x_1 < x_2 < \cdots < x_{n(m-1)+1}$, such that

$$h = \Delta/(m-1) = (b-a)/(n(m-1)), \quad x_j = a + (j-1)h, \quad \text{for } j = 1, 2\cdots, n(m-1)+1. \tag{10}$$

The x_k partition contains all the original z_k points, as well was the half–integer points such as $z_{3/2} = x_2$. The idea now is to precompute all function evaluations on the x_k partition. In Matlab this might read

```
x = transpose(linspace(a,b,n*(m-1)+1)); % set up finer partition, here as column vector.
fx = f(x);
```

By construction, here the function is only evaluated once at each point, but we need extra memory to store all the results! Then, in mixed Matlab/standard notation, we may express the composite quadrature rule from our example as

$$Q^{(10)}_{NC(3)} = \Delta\big(\mathtt{w}*\mathtt{fx(1:3)} + \mathtt{w}*\mathtt{fx(3:5)} + \cdots + \mathtt{w}*\mathtt{fx(17:19)} + \mathtt{w}*\mathtt{fx(19:21)}\big), \tag{11}$$

again where the weights here are `w = [1,4,1]/6` (row vector). Notice that in this expression function evaluations are used twice, say for $f(x_{19})$, but not computed twice. The general algorithm is (compare with Van Loan, page 146) is the following.

```
   function Q = CompClosedQNC(f,a,b,m,n)
%  function Q = CompClosedQNC(f,a,b,m,n)
%  Integrates a function of the form f(x), passed as a handle,
%  from a to b. f must be defined on [a,b] and it must return a
%  column vector. m is an integer that satisfies 2 <= m <= 11.
%  Q is the composite m-point closed Newton-Cotes approximation
%  (based on equal length subintervals) of the integral of f
%  from a to b.
   Delta = (b-a)/n;
   w = NewtonCotesClosedWeights(m);
   % Finer partition than one determined by uniform Delta spacing.
   x = transpose(linspace(a,b,n*(m-1)+1));
   fx = f(x);
   Q = 0;
   first = 1;
   last = m;
   for i = 1:n
       % Add the integral over the i-th subinterval.
       Q = Q + w*fx(first:last); % w is a row vector.
       first = last;
       last = last+m-1;
   end
   Q = Delta*Q;
```

4 Error formula

On page 147 Van Loan quotes the error formula for the composite closed Newton-Cotes rule. Namely,

$$\left| \int_a^b f(x)dx - Q^{(n)}_{NC(m)} \right| \leq \left[|c_m| M_{d+1} \left(\frac{b-a}{m-1} \right)^{d+2} \right] \frac{1}{n^{d+1}}, \tag{12}$$

where d is as in (1). This formula is simply the error formula (1) for a single interval, divided by the factor n^{d+1}, that is the number of subintervals used in the composite rule raised to some power. The formula indicates that we may reduce the error by increasing the number n of subintervals.

As an example, consider, the composite Simpson rule $Q^{(n)}_{NC(3)}$ as before. Then we have $c_3 = -1/90$ (again using Van Loan's list on page 142), so that

$$\left| \int_a^b f(x)dx - Q^{(n)}_{NC(3)} \right| \leq \frac{1}{90} M_4 \left(\frac{b-a}{2} \right)^5 \frac{1}{n^4}. \tag{13}$$

Notice that the $90 \cdot 2^5 = 2880$ factor in the denominator is what we found during the last lecture while deriving the single–interval error formula. To express this error formula in a slightly different way, in Eq. (13) we make a substitution with $h = (b-a)/(n(m-1)) = (b-a)/(2n)$ (for Simpson's rule $m = 3$), in order to find

$$\left| \int_a^b f(x)dx - Q^{(n)}_{NC(3)} \right| \leq \frac{1}{90} M_4 \left(\frac{b-a}{2} \right) \left(\frac{b-a}{2n} \right)^4 = \frac{(b-a)}{180} M_4 h^4. \tag{14}$$

Let us now further specialize to the interval $[a,b] = [0,1]$, and choose $f(x) = \cos(x^2)$. For this function, we compute

$$f^{(4)}(x) = -12\cos(x^2) + 16x^4\cos(x^2) + 48x^2\sin(x^2), \tag{15}$$

so that

$$|f^{(4)}(x)| \leq M_4 = 12 + 16 + 48 = 76 \tag{16}$$

is an easy–to–get bound on the fourth derivative, but Fig. 3 shows that $M_4 \simeq 43$ is a tighter bound. In anycase, with the poorer bound the error formula becomes

$$\left| \int_0^1 \cos(x^2)dx - Q_{NC(3)}^{(n)} \right| \leq \frac{76}{2880}\frac{1}{n^4} \simeq 2.64 \times 10^{-2}\frac{1}{n^4}. \tag{17}$$

Using the formula, we find

For $n = 1$, error $\simeq$ `2.6389e-02`,

For $n = 2$, error $\simeq$ `1.6493e-03`,

For $n = 3$, error $\simeq$ `3.2579e-04`,

For $n = 4$, error $\simeq$ `1.0308e-04`,

For $n = 5$, error $\simeq$ `4.2222e-05`,

$\vdots$

For $n = 256$, error $\simeq$ `6.1441e-12`.

Notice that we expect at least four digits of accuracy for $n = 5$. But these bounds prove very conservative. In Matlab we find

```
>> abs(CompClosedQNC(@cosx2,0,1,3,1) - CompClosedQNC(@cosx2,0,1,3,256))
ans =
   1.8656e-03
>> abs(CompClosedQNC(@cosx2,0,1,3,2) - CompClosedQNC(@cosx2,0,1,3,256))
ans =
   2.2972e-05
>> abs(CompClosedQNC(@cosx2,0,1,3,3) - CompClosedQNC(@cosx2,0,1,3,256))
ans =
   1.3129e-06
>> abs(CompClosedQNC(@cosx2,0,1,3,4) - CompClosedQNC(@cosx2,0,1,3,256))
ans =
   7.8693e-08
>> abs(CompClosedQNC(@cosx2,0,1,3,5) - CompClosedQNC(@cosx2,0,1,3,256))
ans =
   2.9962e-08
>>
```

Here we are using the $n = 256$ approximation as the “exact” answer, which is OK since these errors are much larger than its reported `6.1467e-12` accuracy. To 12 digits that value is $Q_{NC(3)}^{(256)} = 0.904524237900$.

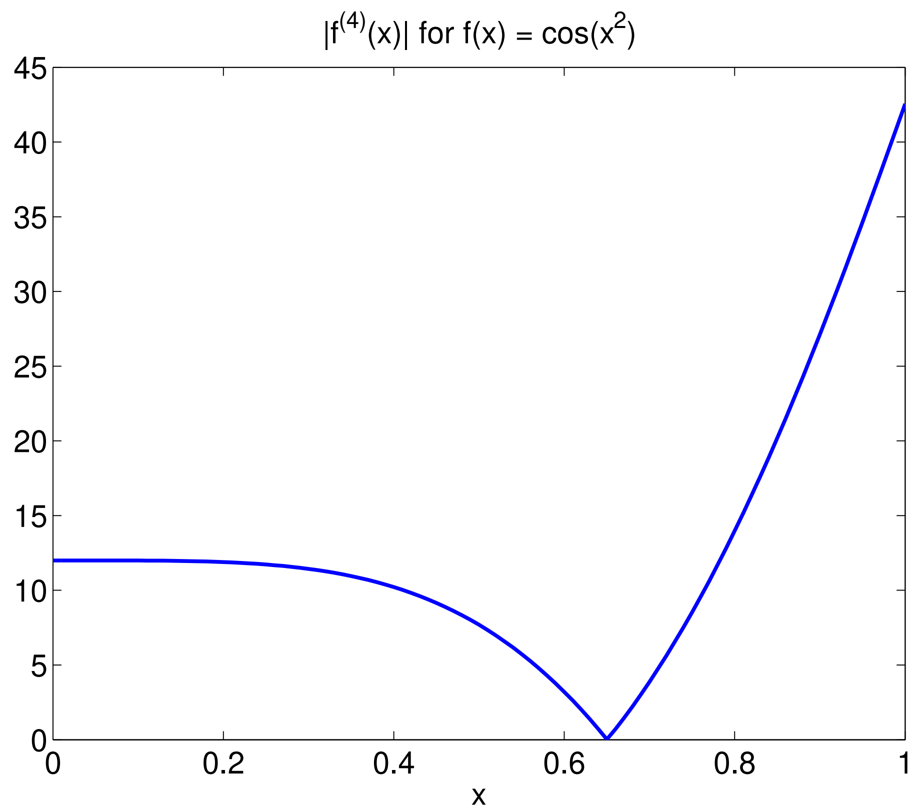

Figure 3: Plot to get better M_4 by inspection.