Numerical integration

Basic idea:

$$\int_a^b f(x)\,\mathrm{d}x \quad = \quad \underbrace{(b-a)\sum_{k=1}^{m} w_k f(x_k)}_{\textbf{rule}} \quad + \quad \underbrace{\mathrm{const}(b-a)^{r+1} f^{(r)}([a\,..\,b])}_{\textbf{error}}, \tag{1}$$

where the **nodes** x_k are (usually) in the interval $[a\,..\,b]$, the **weights** w_k are scalars, as is the **error constant** const, and the strange notation $f^{(r)}([a\,..\,b])$ stands for some $f^{(r)}(\eta)$ with η some (unknown) point in $[a\,..\,b]$. At times, it is convenient to think of $f^{(r)}(a\,..\,b)$ equivalently as the *interval*

$$f^{(r)}(a\,..\,b) := [\inf_{a\le\eta\le b} f^{(r)}(\eta)\,..\,\sup_{a\le\eta\le b} f^{(r)}(\eta)]$$

in which case (1) tells us that the unknown number $\int_a^b f(x)\,\mathrm{d}x$ is certain to lie in the *interval* **rule** $+ \mathrm{const}(b-a)^{r+1} f^{(r)}(a\,..\,b)$.

To help you remember special cases, pay attention to the *units* used. If, e.g., $f(x)$ measures the rate at which you consume potatos, then $\int_a^b f(x)\,\mathrm{d}x$ gives your total potato consumption during the time period from a to b. If you measure potatos by the sack, and time in years, then $f(x)$ would be in units of sacks/year and the integral would be in units of sacks. But then, both summands on the right must be in sacks, as is obviously the case for the rule (since the w_k are scalars, $(b-a)$ is in years and $f(x_k)$ is in sacks/year. For the error, we notice that $f^{(r)}$ is in (sacks/year)/year^r, while const is a scalar, so the factor $(b-a)^{r+1}$ is just right to make the error in units of sacks. (This is an example of **dimensional analysis**, as a quick check whether details of a formula are correct.)

The r in the error term tells you the (exact) **order** of the rule, in that it tells you that the rule is **exact** (i.e., the error is zero) whenever f is a polynomial of degree $< r$, while the rule is guaranteed to be wrong (i.e., the error is nonzero) for the specific rth degree polynomial $f(x) = x^r$.

Any choice of nodes $a \le x_1 < \cdots < x_m \le b$ can be used to construct a rule of order $\ge m$ simply by integrating the polynomial p_{m-1} that interpolates the integrand f at those points: Follow Lagrange in writing p_{m-1} in the suggestive form

$$p_{m-1}(x) = \sum_{k=1}^{m} f(x_k)\ell_k(x), \qquad \ell_k(x) := \prod_{j\ne k} \frac{x - x_j}{x_k - x_j} \tag{2}$$

Indeed, each ℓ_k is the product of $m-1$ linear terms, hence is a polynomial of degree $< m$, therefore, so is the entire right side; also, each ℓ_k vanishes at every x_i *except* x_k, hence, at x_i, the right side has the value $f(x_i)\ell_i(x_i)$ and that equals $f(x_i)$ since, for $x = x_i$, all the factors in $\ell_i(x_i)$ equal 1. So, altogether, the right side equals $f(x_i)$ at x_i, $i = 1{:}m$.

Since the interpolating polynomial is unique, this must be yet another way of writing it. – For our present purposes, the **Lagrange form** (2) of the interpolating polynomial is ideal since it implies at once that

$$\int_a^b p_{m-1}(x)\,\mathrm{d}x = (b-a)\sum_{k=1}^m f(x_k)w_k, \qquad w_k := \int_a^b \ell_k(x)\,\mathrm{d}x/(b-a).$$

Now, the weights w_k are usually *not* actually calculated this way. One way to compute them is to observe that exactness for $f(x) = x^{j-1}$, $j = 1{:}m$ gives the system of linear equations

$$\sum_{k=1}^m x_k^{j-1} w_k = (b^j - a^j)/j, \qquad j = 1{:}m,$$

with the transpose of the Vandermonde matrix as its coefficient matrix, hence its unique solution can be obtained with `matlab`'s backslash operator. However, except for some very special cases, listed below, one usually looks up the weights and nodes of specific rules in books or, these days in matlab, gets them from some m-file (e.g., `NCWeights` in the book). Such listings give rules for a special interval $[a\,..\,b]$ only, typically $[0\,..\,1]$ for the Newton-Cotes rules, and $[-1\,..\,1]$ for the Gauss-Legendre rules, and you must use a linear change of variables to get the information transferred to the interval $[a\,..\,b]$ of interest. E.g., if the rule is stated in terms of $[-1\,..\,1]$, and you are interested in $\int_a^b f(x)\,\mathrm{d}x$, the linear change $x(t) = a + ((b-a)/2)(t+1)$ (NOTE that this $x(t)$ is the linear interpolant in Newton form for the data $(-1, a)$, $(1, b)$) has $\mathrm{d}x/\,\mathrm{d}t = (b-a)/2$, and gives

$$\int_a^b f(x)\,\mathrm{d}x = \frac{b-a}{2}\int_{-1}^1 f(x(t))\,\mathrm{d}t,$$

and the integral on the right is right for the rule given for the interval $[-1\,..\,1]$.

Any rule we are going to consider has at least order 1. Since $\int_a^b 1\,\mathrm{d}x = (b-a)$, this implies that

$$\sum_{k=1}^m w_k = 1$$

for any rule of interest. In particular, for $m = 1$, the only choice of w is `w = 1`.

Here are three specific rules used all the time:

- **Midpoint rule**

$$\int_a^b f(x)\,\mathrm{d}x = (b-a)f((a+b)/2) + (b-a)^3/24\, f''(a\,..\,b), \qquad \text{i.e., } \mathtt{w} = [1];$$

- **Trapezoid rule**

$$\int_a^b f(x)\,\mathrm{d}x = (b-a)(f(a)+f(b))/2 - (b-a)^3/12\, f''(a\,..\,b), \qquad \text{i.e., } \mathtt{w} = [1,1]/2;$$

- **Simpson's rule**

$$\int_a^b f(x)\,\mathrm{d}x = (b-a)(f(a)+4f((a+b)/2)+f(b))/6 - (b-a)^5/(90*32)\; f^{(4)}(a\,..\,b),$$

i.e. `w` $= [1,4,1]/6;$

Each of these is an example of a **Newton-Cotes** rule. These rules come in two flavors. The **closed** $NC(m)$ rule uses `x = linspace(a,b,m)`; the **open** $NC(m)$ rule uses the 'interior' points `x = xx(2:end-1)` of the sequence `xx = linspace(a,b,m+2)`. Both the open and the closed Newton-Cotes rules have the misfortune that, for higher m, some weights are negative. For that reason, in a situation requiring a higher m, one either uses composite rules or else, if the nodes can be chosen at will, one uses Gauss rules.

In any case, the Midpoint rule is the open $NC(1)$, while the Trapezoid Rule is the closed $NC(2)$ and Simpson's Rule is the closed $NC(3)$. The NC-rules are entirely determined by their nodes since they are obtained by integrating the corresponding interpolating polynomial, as described earlier.

In addition to these rules, there are also the **Gauss(-Legendre)** rules. The m-point Gauss rule has the highest-possible order for an m-point rule, namely $r = 2m$. (An m-point rule cannot do better than that for the following simple reason: If $x_1, \ldots, x_m$ are the nodes for the rule, then the $2m$-degree polynomial $p(x) := (x-x_1)^2 \cdots (x-x_m)^2$ is positive except at the nodes, hence $\int_a^b p(x)\,\mathrm{d}x$ is bound to be positive (assuming that $a < b$), yet our rule will provide the value 0, which therefore must be wrong.) Also, the $GL(m)$ rule has positive weights for every m.

For various reasons, one usually does not use any of these rules with very large m. Rather, if a low-m rule does not suffice, one cuts the interval of integration into small (enough) pieces and uses one of these simple rules on each interval, thus getting the corresponding **composite** rule.

Here is again an opportunity to avoid a loop (an opportunity not taken by our textbook). For simplicity, I discuss this question in the special context of the composite $NC(3)$-rule.

Suppose we cut $[a\,..\,b]$ into n equal pieces, each of length

$$h := (b-a)/n,$$

and want to use the closed $NC(3)$ on each of them. This makes our approximation to the integral the double sum:

$$\begin{aligned}
&\quad h\cdot(w_1 f(a) + w_2 f(a+h/2) + w_3 f(a+h)) \\
&+ h\cdot(w_1 f(a+h) + w_2 f(a+3h/2) + w_3 f(a+2h)) \\
&+ h\cdot(w_1 f(a+2h) + w_2 f(a+5h/2) + w_3 f(a+3h)) \\
&\quad \cdots \\
&+ h\cdot(w_1 f(b-h) + w_2 f(b-h/2) + w_3 f(b))
\end{aligned}$$

If we can arrange these various function values into the matrix

$$\texttt{F} := \begin{bmatrix} f(a) & f(a+h/2) & f(a+h) \\ f(a+h) & f(a+3h/2) & f(a+2h) \\ f(a+2h) & f(a+5h/2) & f(a+3h) \\ & \cdots & \\ f(b-h) & f(b-h/2) & f(b) \end{bmatrix},$$

then the calculation of our sum is simple: Applying the matrix `F` to the column vector `w` of weights, gives us the appropriately weighted sum of the columns of `F`, summing the resulting vector `F*w` does the rest. In `matlab`, this would be

```
h*sum(F*w)
```

or, better yet (as it avoids almost all multiplications)

```
h*(sum(F)*w)
```

in which we make use of the fact that `matlab`'s `sum` command works columnwise when applied to a matrix (rather than a vector). **WARNING**: If `F` happens to have just one row, (i.e., if $n = 1$), then `sum(F)` will be just a number, namely the sum of the entries in the sole row of `F`, and this has rather shocking consequences in the present context. So, if you want to make sure, use instead `sum(F,1)`, i.e., specify explicitly in which dimension you want to do the summing. This is something to pay attention to in the meaning of `sum` (and `prod`, `max`, `min`) when writing general purpose m-files like the one we are working on!

This leaves the question of how to organize the function values into the matrix `F`. Here is one way, using `matlab`'s `reshape(A,M,N)` command. This command generates the matrix of size `[M,N]` that has, in its first column, the first `M` entries of `A`, in its second column the next `M` entries of `A`, etc. What if `A` is not a vector but a matrix? Then `matlab` first makes a column-vector from it, taking its first column, then its next column, etc., i.e., treat it as you had written `A(:)` instead of `A`. E.g., the matrix `[1 2 3;4 5 6]` taken as a column-vector would be `[1;4;2;5;3;6]`, hence `reshape([1 2 3;4 5 6],3,2)` would give the matrix `[1 5;4 3;2 6]`. (It is worthwhile to play around with simple examples directly in `matlab` to get a good feel for this important `matlab` tool.)

Here is how `reshape` might be used in our problem. We need the value of f at all the points `x = linspace(a,b,1+n*2)` since we are using the closed $NC(3)$ for our simple rule. If we were using, more generally, the closed $NC(m)$, we would want `x = linspace(a,b,1+n*(m-1))` . So, let `y = f(x)`. Then, for our special case $m - 1 = 2$, `reshape(y(2:length(y)), 2, n)` gives the rearrangement of the sequence `y(2), y(3), ..., y(1+2*n)` into the matrix with 2 rows and `n` columns in which the first 2 entries make up the first column, the next 2 entries make up the second column, etc. . Hence, if we also transpose this matrix, i.e., compute

```
reshape(y(2:length(y)), 2 , n)';
```


then `ans` will look like this:

```
 y(2)      y(3)
 y(4)      y(5)
 y(6)      y(7)
...
y(2*n-2)  y(2*n-1)
y(2*n)    y(2*n+1)
```

But this says that the final construct

```
F = [ [y(1); ans(1:n-1,2)] ans ];
```

gives the desired matrix since its first column has in it the entries `y(1), y(3), y(5), ..., y(2*n-1)` , hence the whole assembled `F` is the matrix

```
y(1)     y(2)      y(3)
y(3)     y(4)      y(5)
y(5)     y(6)      y(7)
...
y(2*n-1) y(2*n)   y(2*n+1)
```

i.e., exactly what we wanted.

This is what is done in the following version of the `function CompQNC` on pages 146-7 of the book:

```
  function numI = CompQNC(fname,a,b,m,n)
% numI = CompQNC(fname,a,b,m,n)
%
% Integrates a function of the form f(x) named by the string fname from a to b.
% f must be defined on [a..b] and it must return a column vector if x is a
% column vector. m is an integer that satisfies 2 <= m <= 11.
% numI is the composite m-point Newton-Cotes approximation of the integral of f
% from a to b with n equal length subintervals.
%
   h = (b-a)/n;
   w = NCWeights(m);
   x = linspace(a,b,1+n*(m-1))'; % transposition needed in case fname
                                 % expects x to be a column vector.
   y = feval(fname,x);
   if n==1, numI = h*(y.'*w); return, end
   reshape(y(2:length(y)),m-1,n).';
   numI = h*(sum([[y(1);ans(1:n-1,m-1)] ans])*w);
```

