Gregory's quadrature method

Gregory's method is among the very first quadrature formulas ever described in the literature, dating back to James Gregory (1638-1675). It seems to have been highly regarded for centuries, but it is less often seen nowadays. Reference [1] offers an operator algebra type derivation, but it also makes the puzzling comment "Gregory's formula, as a rule, is not used for numerical quadrature but sometimes for summation". Later text book authors, in the habit of nowadays mostly copying from each other, usually ignore Gregory's method altogether. Nevertheless, it is one of the real gems from long ago.

1. Introduction: Concept behind Gregory's formula, and summary of the rest of these notes

The key idea is that virtually all the errors in the classical trapezoidal rule comes from what happens at the ends of the interval, and that these errors can be greatly reduced by a sequence of 'end corrections'.

For simplicity of notation, we start by considering the approximation of a convergent integral $\int_0^\infty f(x)\,dx$ based on the function values only at the integer locations $x_\nu = \nu$, $\nu = 0, 1, 2, \ldots$ A really crude approximation would be

$$\int_0^\infty f(x)\,dx \approx \sum_{\nu=0}^{\infty} f(\nu) = f(0) + f(1) + f(2) + \ldots \qquad (1)$$

In the next several sections of these notes, we will

- Develop a sequence of 'end corrections' at $x = 0$ for increasing the accuracy of (1),
- Generalize the node locations from $x_\nu = \nu$ to $x_\nu = \nu h$, where h denotes an arbitrary spacing between the nodes,
- Change the integration interval from $[0, \infty]$ to an arbitrary finite interval $[a, b]$. In that step, the assumption that the integral converges over the infinite interval can be dropped.
- Discuss Gregory's method and, in particular, compare it with Simpson's method.

The last three sections of these notes focus on its numerical implementation:

- Provide a Matlab code for generating the Gregory weights,
- Give a numerical example,
- Tabulate the exact (rational) quadrature weights.

2. Sequence of end corrections

We know from the trapezoidal rule that the coefficient for $f(0)$ in (1) ought to be $\frac{1}{2}$ rather than 1. If we used a node spacing of h (instead of just $h = 1$), this simple change in the first coefficient will reduce the overall error from $O(h)$ to $O(h^2)$. It is then natural to further ask whether additional end corrections can reduce the errors, in some step-by-step manner, to any desired order. This is not at all implausible, given that the trapezoidal rule is spectrally accurate over $[-\infty, \infty]$ for analytic functions that decay sufficiently fast in both directions. All algebraic error terms must therefore come from what happens at the $x = 0$ boundary. These error terms ought to be possible to identify and remove. Euler-MacLaurin's formula does this, but requires us to know high derivatives at the end point. The Gregory approach is conceptually similar, but requires no such extra information.

With the notation

$$\Delta f(x_\nu) = f(x_{\nu+1}) - f(x_\nu)$$

we obtain

$$\begin{array}{ll} \Delta^0 f(0) & = f(0) \\ \Delta^1 f(0) & = f(1) - f(0) \\ \Delta^2 f(0) & = f(2) - 2f(1) + f(0) \\ \Delta^3 f(0) & = f(3) - 3f(2) + 3f(1) - f(0) \\ \vdots & \quad \vdots \end{array} \quad ,$$

where we recognize the coefficients as coming from the successive lines of Pascal's triangle. We next try to 'correct' (1) by adding yet unknown amounts of the terms above

$$\int_0^\infty f(x)\,dx = \sum_{\nu=0}^{\infty} f(\nu) + [b_0\Delta^0 + b_1\Delta^1 + b_2\Delta^2 + b_3\Delta^3 + \ldots] f(0) \tag{2}$$

A major advantage in the formulation (2) (as opposed to trying to find the Gregory weights directly) is that the sequence of increasing order Gregory methods will just correspond to including additional terms in the expansion (2), with a single set of coefficients b_k. In contrast, all the quadrature weights change whenever the order of the approach is changed. Hence, the expansion (2) serves as a very convenient intermediate step towards calculating the actual weights.

At this point, we recall that an arbitrary periodic function over $[0, 2\pi]$ can be written as a combination of e^{ikx}, $k \in Z$ (a Fourier series) and that functions that decay fast enough over $[-\infty, \infty]$ similarly can be written as combinations of $e^{i\omega x}$, $\omega \in R$ (real) (a Fourier transform). One might therefore guess that functions $f(x)$ that decay for increasing x similarly can be written as combinations of e^{-zx} where $\operatorname{Re} z > 0$. Substituting $f(x) = e^{-zx}$ into (2) gives, after a few quick simplifications

$$\frac{1}{z} = \frac{1}{1-e^{-z}} + [b_0 - b_1(1-e^{-z}) + b_2(1-e^{-z})^2 - b_3(1-e^{-z})^3 + - \ldots].$$

After changing variables $w = 1 - e^{-z}$, i.e. $z = -\log(1-w)$, we get

$$\frac{1}{\log(1-w)} + \frac{1}{w} = -b_0 + b_1 w - b_2 w^2 + b_3 w^3 - + .., \tag{3}$$

and can therefore pick up the unknown coefficients by Taylor expanding the function $\frac{1}{\log(1-w)}+\frac{1}{w}$ around $w=0$. Multiplying up the denominators, utilizing the elementary Taylor expansion for function $\log(1-w)$ and then equating coefficients lead to a simple recursion for calculating any number of b-coefficients. Expressed in matrix form, the coefficients are obtained by back substitution in the triangular Toeplitz system

$$\begin{bmatrix} 1 & & & & \\ -\frac{1}{2} & 1 & & & \\ \frac{1}{3} & -\frac{1}{2} & 1 & & \\ -\frac{1}{4} & \frac{1}{3} & -\frac{1}{2} & 1 & \\ \vdots & \vdots & & \ddots & \ddots \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ \vdots \end{bmatrix} = \begin{bmatrix} -\frac{1}{2} \\ \frac{1}{3} \\ -\frac{1}{4} \\ \frac{1}{5} \\ \vdots \end{bmatrix} \quad , \tag{4}$$

from which follows

$$b_0=-\tfrac{1}{2}, b_1=\tfrac{1}{12}, b_2=-\tfrac{1}{24}, b_3=\tfrac{19}{720}, b_4=-\tfrac{3}{160}, b_5=\tfrac{863}{60480}, b_6=-\tfrac{2751}{24192}, \ldots$$

Analysis (which we do not include here) will show that, for each extra coefficient that is included, the overall accuracy will increase with one power of h (in the case of using a spacing of h rather than just the special case of $h = 1$).

3. Change of grid spacing from $h = 1$ to a general h

We first rearrange (2) in terms of modified weights near the left end

$$\int_0^\infty f(x)\,dx \approx \sum_{\nu=0}^{\infty} c_\nu f(\nu)$$

where a number of the leading c_ν no longer are equal to one. After this, one merely inserts an h in front of the sum. For nodes $x_\nu = \nu h$, the quadrature formula thus becomes

$$\int_0^\infty f(x)\,dx \approx h \sum_{\nu=0}^{\infty} c_\nu f(x_\nu) \ . \tag{5}$$

If we include k correction terms, the coefficients $c_\nu,\ \nu=0,1,\ldots,k-1$ have been modified while the remaining ones $c_\nu,\ \nu=k,k+1,\ldots$ remain at the value one.

4. Change to a finite integration interval

Also this change is trivial. All that is required is to apply the corrections described by (5) for the left end of the interval also (but backwards) at the right end. When all is said and done, the first and the last c_ν coefficient will be the same, then the second and the next-to-last the same, etc.

5. Brief discussion

It is well known that the trapezoidal rule usually is pretty bad, with an error of size $O(h^2)$. However, as noted above, the key exceptions arise if the problem either is periodic or on an infinite interval, when the accuracy for sufficiently smooth functions becomes spectral. The two enhanced approaches for bounded intervals that modern text books tend to focus on are

i. Newton-Cotes formulas (including Simpson's rule), and
ii. Gaussian quadrature (GQ).

If GQ can be used, that is often a good option. However, assuming as we do here that the function to be integrated is only available at equispaced node locations, GQ is is not directly applicable.

Simpson's rule / Newton Cotes formulas have several shortcomings:

- The accuracy of Simpson is only $O(h^4)$. Higher orders are sufficiently awkward to reach that a they are seldom used.

- With the Simpson quadrature weights $\frac{h}{3}[1\,4\,2\,4\,2\,\ldots\,2\,4\,2\,4\,1]$, it is necessary to have an odd number of nodes. Higher order versions come with still more awkward node number restrictions. It is not trivial to just jack up the order step-by-step.

- The oscillating Simpson / Newton-Cotes weights across the whole interval is a very clumsy approach to handle errors that have their cause in what happens only at the two boundaries. One can obtain Simpson's rule on a grid spaced h by Richardson extrapolating two trapezoidal approximations, based on grids h and $2h$. In the important cases for which trapezoidal rule is spectrally accurate, 'contaminating' the grid h-based approximation with one using $2h$ grossly damages the accuracy. The Gregory approach involves no such accuracy destruction for smooth functions.

- As we will see in the test case in Section 7 below, it is trivial to run Gregory for a succession of increasing orders (without any restrictions on the number of nodes that are used).

6. Matlab code for generating the Gregory weights

```
function w = Gregory(n_nodes,h,order)

%   This function calculates the Gregory quadrature weights for equispaced integration
%   n_nodes      Total number of nodes
%   h            Step size
%   order        Order of accuracy desired, 2,3,4,... (with 2 giving the trapezoidal
%                rule). The value must satisfy  2 <= order <= n_nodes
%   w            Output: The weights to be used for the successive function values
%
%   If  f  is a row vector containing the function values, the integral is
%   approximated by the statement   integral = f*w'

% Create the sequence of Gregory coefficients
r = 1./(1:order); gc = toeplitz(r(1:order-1),[r(1),zeros(1,order-2)])\r(2:order)';

% Create the weights vector  w  and then update it at the two ends of the interval
```

```
w                          = ones(1,n_nodes)*h;
w_updates                  = sum(h*repmat(gc,1,order-1).*pascal(order-1,1));
w(1:order-1)               = w(1:order-1)-w_updates(1:order-1);
w(n_nodes-order+2:n_nodes) = w(n_nodes-order+2:n_nodes)-fliplr(w_updates(1:order-1));
```

7. Numerical example of using the Gregory method

The following main script tests the function Gregory by approximating $\int_{-1}^{1} e^x dx = e^1 - e^{-1}$ for a variety of choices of the variables n_nodes and order:

```
% Test the Gregory code for evaluating the integral of exp(x) over [-1,1]

exact = exp(1)-exp(-1);                % Exact value for integral
for n_nodes = 11:10:31                 % Loop over the number of nodes n_nodes
    h = 2/(n_nodes-1);
    x = linspace(-1,1,n_nodes); f = exp(x);
    for order = 2:8                    % Loop over increasing orders
        w = Gregory(n_nodes,h,order);  % Get the Gregory weights
        int = f*w';                    % Evaluate the integral
        [n_nodes,order]                % Print out [n_nodes,order]
        error = int-exact              % Print out the error
    end
end
```

If we order the results produced by this test program in the form of a table, we get

	n_nodes = 11	21	31
order = 2	0.0078	0.0020	8.7045e-004
3	9.7460e-004	1.2510e-004	3.7405e-005
4	8.0001e-005	5.5814e-006	1.1425e-006
5	1.5622e-005	5.2890e-007	7.1695e-008
6	1.3010e-006	2.6528e-008	2.5251e-009
7	3.4126e-007	3.0112e-009	1.8488e-010
8	2.3506e-008	1.5047e-010	6.7168e-012

As a practical method for error estimation, one can just see how much the approximated integral values vary when the order is successively increased.

8. Tables of weights

Since the weights are the same at the two interval ends (although in reversed order) and they just scale directly with the step size h, it suffices here to list the beginning weights (at the left interval edge) in the case of $h = 1$. The Matlab code

```
c = zeros(10,10);
for order = 2:10
    cf = Gregory(20,1,order); c(order,:) = cf(1:10);
end
c(2:10,:)
```

produces (for orders 2-10) the output

```
0.5000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000
0.4167    1.0833    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000
0.3750    1.1667    0.9583    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000
0.3486    1.2458    0.8792    1.0264    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000
0.3299    1.3208    0.7667    1.1014    0.9813    1.0000    1.0000    1.0000    1.0000    1.0000
0.3156    1.3922    0.6240    1.2441    0.9099    1.0143    1.0000    1.0000    1.0000    1.0000
0.3042    1.4604    0.4535    1.4714    0.7394    1.0825    0.9886    1.0000    1.0000    1.0000
0.2949    1.5259    0.2570    1.7989    0.4119    1.2790    0.9231    1.0094    1.0000    1.0000
0.2870    1.5890    0.0360    2.2409   -0.1406    1.7209    0.7021    1.0725    0.9921    1.0000
```

It is also straightforward to implement the coefficient algorithm in Mathematica rather than in Matlab, instead producing the weights in exact rational form. These exact values can be convenient to use if one just wants to implement Gregory's method for some fixed order of accuracy. For the nontrivial weights in the output above, this gives

$$\begin{array}{cccccccccc}
\frac{1}{2} \\
\frac{5}{12} & \frac{13}{12} \\
\frac{3}{8} & \frac{7}{6} & \frac{23}{24} \\
\frac{251}{720} & \frac{299}{240} & \frac{211}{240} & \frac{739}{720} \\
\frac{95}{288} & \frac{317}{240} & \frac{23}{30} & \frac{793}{720} & \frac{157}{160} \\
\frac{19087}{60480} & \frac{84199}{60480} & \frac{18869}{30240} & \frac{37621}{30240} & \frac{55031}{60480} & \frac{61343}{60480} \\
\frac{5257}{17280} & \frac{22081}{15120} & \frac{54851}{120960} & \frac{103}{70} & \frac{89437}{120960} & \frac{16367}{15120} & \frac{23917}{24192} \\
\frac{1070017}{3628800} & \frac{5537111}{3628800} & \frac{103613}{403200} & \frac{261115}{145152} & \frac{298951}{725760} & \frac{515677}{403200} & \frac{3349879}{3628800} & \frac{3662753}{3628800} \\
\frac{25713}{89600} & \frac{1153247}{725760} & \frac{130583}{3628800} & \frac{903527}{403200} & -\frac{797}{5670} & \frac{6244961}{3628800} & \frac{56621}{80640} & \frac{3891877}{3628800} & \frac{1028617}{1036800} \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots
\end{array}$$

The last few rows give an indication about why extremely high orders of Gregory's procedure might not be advisable. Just like for Newton-Cotes methods of very high orders, the weights start to grow and oscillate, causing a danger of accuracy losses due to numerical cancellations.

Reference:

[1] Fröberg, C-E., Introduction to Numerical Analysis, Addison-Wesley, Second Edition, 1969.