CUMULATIVE HIGH-ORDER NEWTON-COTES ALGORITHMS[*]

GARRETT J. YOUNG[†] AND HEATHER E. YOUNG[‡]

Abstract. The authors derive a modified cumulative Simpson 3/8ths method, generalize that method for higher-order closed Newton-Cotes formulae, and perform a check of their accuracy using various non-linear functions. The authors then identify the ideal implementation of these methods and provide an example using Python. The authors finally derive a version of the method that accounts for time step variance, therefore addressing a common issue with high-order integration using Newton-Cotes methods.

Key words. numerical integration, Newton-Cotes, Simpson method, cumulative numerical integration, signal processing

AMS subject classifications. 65Y04, 65R04

1. Introduction.

One difficulty when performing a numerical integral arises when the exact function is unknown. For example, when cumulatively integrating sampled data from a signal, classically the trapezoidal method is used [1] because more sophisticated methods require knowledge of the function or that the signal is post-processed. However, the trapezoidal method has a less favorable error term than many higher order approximations.

To address these problems, we have derived a method for generating strictly cumulative versions of closed Newton-Cotes formulae, expanding upon the work of L.V. Blake [2].

It should be noted that normally the Simpson 3/8ths rule, the third order closed Newton-Cotes rule, is written as [3]

$$\int_{x_1}^{x_4} f(x)\,dx = \frac{3}{8}h\left(f_1 + 3f_2 + 3f_3 + f_4\right) - \frac{3}{80}h^5 f^{(4)}(\xi) \tag{1.1}$$

and Boole's rule, the fourth order closed Newton-Cotes rule, is written as [4]

$$\int_{x_1}^{x_5} f(x)\,dx = \frac{2}{45}h(7f_1 + 32f_2 + 12f_3 + 32f_4 + 7f_5) - \frac{8}{945}h^7 f^{(6)}(\xi) \tag{1.2}$$

2. Derivation for 3/8ths Simpson Rule.

Beginning with the derivation of the cumulative version of the Simpson 3/8ths rule, we can consider the graph of a random function, $y(x)$ (see Figure 2.1) which we wish to integrate.

We split the subinterval further into four sub-subintervals as required by the method and then combine these sub-subintervals into two subintervals, A_1 and A_2. These are defined as from $-2h$ to 0 and from 0 to h, respectively. Although only the A_2 subinterval is added each step (therefore making the method cumulative), the polynomial fit benefits from all the points given.

$$y(x) = ax^3 + bx^2 + cx + d \tag{2.1}$$

[*]Submitted to editors July 1, 2018.

Funding: This work was done while the authors were employees at Wet Design.

[†]Wet Design, 10847 Sherman Way, Sun Valley, CA 91352 (Garrett.Young@wetdesign.com).

[‡]Wet Design, 10847 Sherman Way, Sun Valley, CA 91352 (Heather.Young@wetdesign.com).

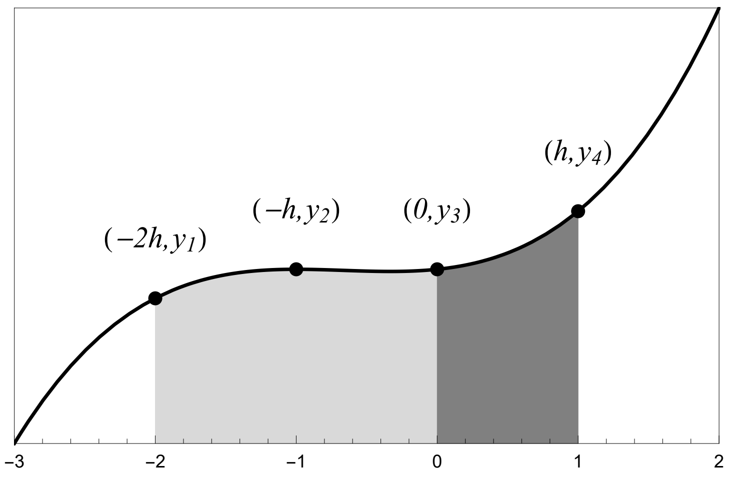

FIGURE 2.1. *A function $y(x)$ to be integrated. Notice in particular the coordinates for each point. The interval from x-value $-2h$ to 0 is interval A_1, and from x-value 0 to h is A_2. (The horizontal distance between each point may be defined as h.)*

We therefore know from this that

$$\begin{aligned} A_1 &= \int_{-2h}^{0} y(x)\,dx = \left[\frac{ax^4}{4} + \frac{bx^3}{3} + \frac{cx^2}{2} + dx\right]_{-2h}^{0} \\ &= -4ah^4 + \frac{8bh^3}{3} - 2ch^2 + 2dh \end{aligned} \tag{2.2}$$

$$\begin{aligned} A_2 &= \int_{0}^{h} y(x)\,dx = \left[\frac{ax^4}{4} + \frac{bx^3}{3} + \frac{cx^2}{2} + dx\right]_{0}^{h} \\ &= \frac{ah^4}{4} + \frac{bh^3}{3} + \frac{ch^2}{2} + dh \end{aligned} \tag{2.3}$$

We also notice from Figure 2.1 that we can define the equations y_1, y_2, y_3, y_4 as follows by substituting the assigned x values into 2.1:

$$y_1 = -8ah^3 + 4bh^2 - 2ch + d \tag{2.4}$$

$$y_2 = -ah^3 + bh^2 - ch + d \tag{2.5}$$

$$y_3 = d \tag{2.6}$$

$$y_4 = ah^3 + bh^2 + ch + d \tag{2.7}$$

We may then solve the system defined by equations 2.4 through 2.7 for the coefficients a, b, c, d and find that

$$a = -\frac{y_1 - 3y_2 + 3y_3 - y_4}{6h^3} \tag{2.8}$$

$$b = -\frac{-y_2 + 2y_3 - y_4}{2h^2} \tag{2.9}$$

$$c = -\frac{-y_1 + 6y_2 - 3y_3 - 2y_4}{6h} \tag{2.10}$$

$$d = y_3 \tag{2.11}$$

And then substitute these coefficients into 2.2 and 2.3 to find the new equations for A_1 and A_2:

$$A_1 = \frac{1}{3}h\left(y_1 + 4y_2 + y_3\right) \tag{2.12}$$

Note that this is the exact formula for Simpson's 1/3 rule; further investigation revealed that A_1 equals the previous order's formula when $y(x)$ is of an even degree.

$$A_2 = \frac{1}{24}h\left(y_1 - 5y_2 + 19y_3 + 9y_4\right) \tag{2.13}$$

The formula A_2 is used to calculate the cumulative integral; A_1 is simply provided for reference.

The same method applied to Boole's rule yields:

$$A_1 = \frac{3}{80}h(9y_1 + 34y_2 + 24y_33 + 14y_4 - y_5) \tag{2.14}$$

$$A_2 = \frac{1}{720}h(-19y_1 + 106y_2 - 264y_3 + 646_4 + 251y_5) \tag{2.15}$$

3. Generalization. We may now generalize the method described in Section 2 as follows. First, consider a function that may be approximated by an nth order polynomial

$$y(x) = a_n x^n + a_{n-1}x^{n-1} + ... + a_1 x + a_0 \tag{3.1}$$

Next, define a subinterval from $-(n-1)h$ to 0 called A_1, and another subinterval from 0 to h called A_2 as follows:

(3.2)
$$\begin{aligned} A_1 &= \int_{-(n-1)h}^{0} y(x)\,dx \\ &= -\left(\frac{a_n[-(n-1)h]^{n+1}}{n+1} + \frac{a_{n-1}[-(n-1)h]^n}{n} + ... + \frac{a_1[-(n-1)h]^2}{2} - a_0(n-1)h\right) \end{aligned}$$

$$A_2 = \int_0^h y(x)\,dx = \frac{a_n h^{n+1}}{n+1} + \frac{a_{n-1}h^n}{n} + ... + \frac{a_1 h^2}{2} + a_0 h \tag{3.3}$$

We then define $y_1, ..., y_n$:

$$y_1 = a_n(-(n-1)h)^n + a_{n-1}(-(n-1)h)^{n-1} + ... + a_1(-(n-1)h) + a_0 \tag{3.4}$$

$$\vdots$$

$$y_{n-1} = a_0 \tag{3.5}$$

$$y_n = a_n h^n + a_{n-1}h^{n-1} + ... + a_1 h + a_0 \tag{3.6}$$

By assigning the appropriate value to n, we may now find a nth order approximating cumulative formula by solving the system of equations defined by 3.4 through 3.6 for the coefficients and then substituting these coefficients into equations 3.2 and 3.3.

4. Implementation. In order to implement a third order cumulative Newton-Cotes formula in Python we define a function that takes in a list of points `new_val` with length $n+1$, the current time step `step`, the size of the time step `t_step`, the previous value of the integral `integral_val`, and the initial value of the integral `init_val`:

```
1  def integrate(step, t_step, new_val, integral_val, init_val):
2      if step == 1:
3          return t_step/2*(new_val[-2] + new_val[-1])+init_val
4      elif step == 2:
5          array_values = new_val[-3]+4*new_val[-2]+new_val[-1]
6          return t_step/3 * (array_values) + init_val
7      elif step == 3:
8          values1 = new_val[-4] + 3*new_val[-3]
9          values2 = 3*new_val[-2] + new_val[-1]
10         return 3*t_step/8 * (values1 + values2) + init_val
11     else:
12         values1 = new_val[-4] - 5*new_val[-3]
13         values2 = 19*new_val[-2] + 9*new_val[-1]
14         return t_step/24 * (values1 + values2) + integral_val
```

The function operates in two phases. The first phase, in lines 1 through 10, operates as the initialization of the integral. The full cumulative method is not used, but rather the standard composite rule. Note that in this phase, as well, `integral_val` is not used, but `init_val`. This first phase continues until the higher-order method desired has been reached.

The second phase, shown here in lines 11 through 14, executes the cumulative method. Here `integral_val` is used to carry forward the previous sum, and it stays at the highest order method. This code was tested on a variety of signals and systems of differential equations and achieved improved accuracy in all cases tested.

5. Ideal Integration Method. We now can use a cumulative version of any order closed Newton-Cotes method; this then raises the question of determining the best combination. A balance must be achieved here between Runge's phenomenon, which implies interpolating polynomials of too high an order will increase error, [5] and the knowledge that to some point higher order interpolating polynomials bring higher accuracy.

Another consideration is the initial error introduced by the use of the trapezoidal method for the first step. Several methods were considered for minimizing the error. The selected method used the highest order interpolation scheme possible at each step, updating the cumulative integral value until the fifth order is applied.

A simple analysis of this was done using a selection of nonlinear functions and various combinations of cumulative Newton-Cotes methods, and the results can be found in Table 5.1.

Function	x^4	$\ln(1+x)$	$\sqrt{x}$	$\sin^2 x$	$e^{-x} - e^{-x}(1+x)$
Trapezoidal	-1.042×10^{5}	1.976×10^{-2}	7.152×10^{-2}	-3.464×10^{-6}	-8.330×10^{-4}
Trapezoidal & Simpson 1/3	-4.167×10^{1}	3.824×10^{-3}	4.414×10^{-2}	-9.527×10^{-3}	6.452×10^{-5}
Trapezoidal & Simpson 1/3 & Simpson 3/8	-1.547×10^{-2}	6.224×10^{-3}	5.266×10^{-3}	-1.475×10^{-2}	-7.150×10^{-6}
Trapezoidal & Simpson 1/3 & Simpson 3/8 & Boole	2.127×10^{-5}	-3.782×10^{-7}	2.234×10^{-3}	-1.328×10^{-7}	4.942×10^{-7}
Trapezoidal & Simpson 1/3 & Simpson 3/8 & 5th order	2.624×10^{-6}	1.764×10^{-7}	2.496×10^{-3}	4.334×10^{-8}	-8.938×10^{-8}

TABLE 5.1

Shows average error of each combination of Newton-Cotes methods on each nonlinear function over a sample interval of 0 to 5000 with $\Delta x = 0.1$.

Table 5.1 indicates that a combination of the trapezoidal, Simpson 1/3rd, Simpson 3/8th, Boole, and 5th order methods produces the best results. Interestingly, this seems directly contrary to Runge's phenomenon; the authors believe this could be due to the fact that the functions are used in a piecewise composite manner.

6. Time Step Variance. Up until now, all algorithms discussed rely upon the distance between each point being the same; that is, there being no time step variance. However, time step variance is sometimes necessary with unevenly sampled data, or using time step variance may increase accuracy of the output while still retaining a reasonable speed for calculation when dealing with signals with a high rate of change.

The authors have developed a simple method that allows changing the time step using a cumulative method similar to that previously discussed. First, we consider the graph of a random function $y(x)$ (see Figure 6.1) which may be approximated by a third-degree polynomial, as in equation 2.1.

On this graph, we still have points y_1, y_2, y_3, y_4, but the interval is now from $-2k$ to h, where $k = h/2$. Since h and k have this ratio, we can find the equations $y_1, ..., y_4$ and A_1, A_2 in a similar way as before, but we can transition between time step sizes by combining the two k subintervals into one subinterval of size h.

This means we only take three points as input (though y_2 still affects the results) and must use the Simpson 1/3rd method for two steps (or until enough points with the h interval between are obtained for the higher order method) before transitioning back to the 3/8ths method. To get a different ratio between k and h, one must attempt to balance the subintervals such that the subintervals can combine into the number of intervals necessary for an existing lower order method. Larger ratios can therefore be achieved by using a different sort of stepping, like jumping from sixth order approximations to third order approximations.

We can then define A_1 and A_2 in terms of k and h as in equations 2.2 and 2.3 (here the interval for 2.2 would be $-2k$ to 0). We also define our equations $y_1, ...y_4$ as in equations 2.4 through 2.7 except in equation 2.4, one would substitute $-2k$ for x in 2.4, and $-k$ for x in 2.5. We then solve our resulting system of equations for

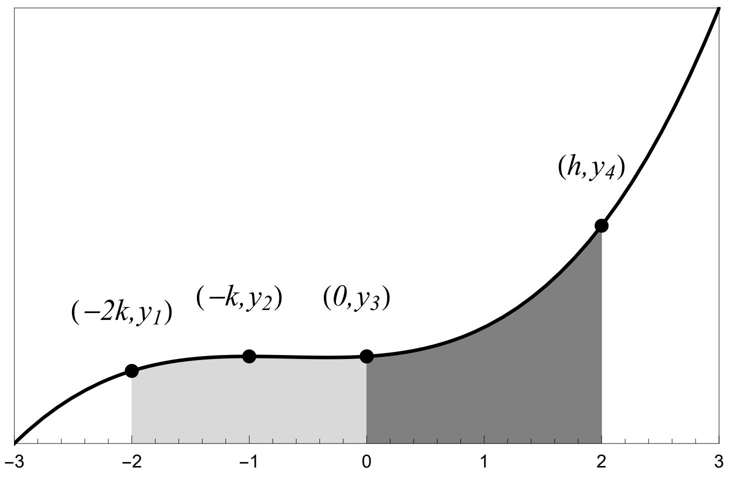

FIGURE 6.1. *A function $y(x)$ to be integrated. Notice in particular the coordinates for each point. The interval from x-value $-2h$ to 0 is interval A_1, and from x-value 0 to h is A_2. (The horizontal distance between the first two points is k and the distance between the first three points is h - that is, $k = h/2$.*

a, b, c, d and get

$$-\frac{3y_1 - 8y_2 + 6y_3 - y_4}{3h^3} \tag{6.1}$$

$$-\frac{-y_1 + 2y_3 - y_4}{2h^2} \tag{6.2}$$

$$-\frac{-3y_1 + 16y_2 - 12y_3 - y_4}{6h} \tag{6.3}$$

$$d = y_3 \tag{6.4}$$

We then substitute equations 6.1 through 6.4 into our previous equations for A_1 and A_2 to produce our final equations for A_1 and A_2:

$$A_1 = \frac{h}{6}(y1_+ 4y_2 + y_3) \tag{6.5}$$

$$A_2 = \frac{h}{6}(y_1 - 4y_2 + 7y_3 + 2y_4) \tag{6.6}$$

As before, we simply add A_2 each time step. This method works to make the time step larger. (Although the trapezoidal mention is an option to transition to larger step sizes, the error term may be unacceptable at the larger intervals.)

To make time steps smaller, we switch to the trapezoidal method with the new step size and transition to the more accurate methods as soon as possible. Here, there is no need for k, the old step size, to be related to h, the new step size.

7. Discussion. This method produces a truly cumulative method for integration. The applications for this are broad. It is very helpful in simulations where one is doing the calculations in real time, such as if one was simulating an analog computer.

For example, some of the problems described in [6] and [7] could be solved using this method. The method can do this when dealing only with real time data, which is very useful when one does not have knowledge of the exact function or the derivative of the function, as in, for example, signal processing.

Another advantage of the proposed cumulative method is that it removes the need for the total number of data points to be divisible by the order of the approximating function plus one. This is because in each individual addition the new step is added as part of the required number of subintervals, but overall, there is no need for a specific number of steps.

The authors developed these methods when simulating the motion of an ion in a crossed electric and magnetic field. The motion of the ion and the spatial fields may be described by a system of coupled second order nonlinear differential equations, and our integration method was able to correctly calculate the motion of that ion.

Further research in this area lies in creating a cumulative method that allows for time step variance that is neater and more concise than what has been described above. One method for doing this could be developing a formula that has the correct weighting coefficients so that no switching between different order formulas is required.

Acknowledgements. The authors would like to thank Wet Design for providing a fun and creative environment where research can occur.

REFERENCES

[1] Juana Arias-Trujillo, Rafael Blázquez, and Susana López-Querol, *Frequency-Domain Assessment of Integration Schemes for Earthquake Engineering Problems*, Mathematical Problems in Engineering, vol. 2015, 2015, Article ID 259530.

[2] L.V. Blake, *A Modified Simpson's Rule and Fortran Subroutine for Cumulative Numerical Integration of a Function Defined by Data Points*, Report 2231, Naval Research Laboratory, Radar Geophysics Branch, Radar Division, Washington, D.C., 1971.

[3] C. W. Ueberhuber, Numerical Computation 2: Methods, Software, and Analysis, Berlin: Springer-Verlag, Berlin, 1997, p. 100

[4] Abramowitz, M. and Stegun, I. A. (Eds.). "Integration." 25.4 in Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables, 9th printing. Dover, New York, 1972, p. 886.

[5] Carl Runge, *Über empirische Funktionen und die Interpolation zwischen quidistanten Ordinaten*, Zeitschrift für Mathematik und Physik, 46, 1901, pp. 224243

[6] John W. Wilson and George G. Steinmetz, Analysis of Numerical Integration Techniques for Real-time Digital Flight Simulation, NASA Technical Note, 1968, TN D-4900.

[7] Robert J Butera and Maeve L McCarthy, Analysis of real-time numerical integration methods applied to dynamic clamp experiments, J. Neural Eng. 1 (2004) 187194.