Hybrid Newton-Cotes Integrals

By

Namir C. Shammas

Introduction

Newton-Cotes integration methods are numerical methods for integration. These methods calculate the estimate of an integral using either an array of function values or by calculating the function values at different points within the integration interval. This article focuses on the latter type.

Newton-Cotes integrals fall into two categories—closed and open. Closed integrals calculate the integrals for the interval (α, β) using values at the interval's ends and values in between. This class of integrals assume that the integrated function, f(x), can be calculated at any point in the interval (α, β). Open integrals calculate the integrals for an interval (α, β) using values strictly inside that interval. This class of integrals assume that the integrated function, f(x), cannot be calculated at either or both interval ends. Open Newton-Cotes integrals are less accurate than their closed counterpart when using the same integration step sizes. However, they can estimate the integrals of a function f(x) where f(α) and/or f(β) cannot be calculated!

Popular closed Newton-Cotes integrals include the Trapezoidal rule, Simpson's rule, Simpson's 3/8 rule, and Boole's rule. Popular (but less known) open Newton-Cotes integrals include the Trapezoid method and Milne's rules.

This paper introduces you to a new category of numerical integration methods that are a mix between the open and closed Newton-Cotes integrals. The new *hybrid* algorithms calculate the value of the integrated function at one end of the interval (α, β). This approach works when either f(α) or f(β) cannot be calculated.

I used MATLAB's symbolic computation features to assist me in quickly and accurately derive the equations used in numerical integration. Such a wonderful tool allowed me to rederive equations and perform corrections, quickly and painlessly, when needed. The MATLAB symbolic computation empowered me to pursue the new algorithms in greater variety. See Appendix A for sample MATLAB scripts that I used.

✍

In this paper, I present pseudo-code for various integration algorithms. I focus on applying these algorithms to each (i.e. inside) integration step size h, and not across the integration interval (α, β) where you make a single function evaluation per integration step. I assume that α – β is an integer multiple of the integration step size h. As such, the pseudo-code is ready for implementing the chained versions of the various algorithms.

Translating the pseudo-code into working subroutine functions (in your preferred programming language) will cause the executing code to perform more evaluations of the integrated functions, at the benefit of more accurate answers than traditional integration schemes that make a single function evaluations per integration step—which you can also implement. In the day and age of fast CPU, the additional CPU effort should not be a burden. Of course you can always adapt the algorithms to make single function evaluation per integration step.

The Three-Point Hybrid Newton-Cotes Integral

This section presents two three-points hybrid Newton-Cotes integration methods. Let me first reintroduce Milne's method which is the comparable Open Newton-Cotes integration method:

```
Given f(x), the integration interval (A, B) and the increment h.
X = A
Sum = 0
Do
  X1 = X + 0.25 * h
  X2 = X + 0.5 * h
  X3 = X + 0.75 * h
  Sum = Sum + 2 * f(X1) - f(X2) + 2 * f(X3)
  X = X + h
Loop Until X > B
Integral = Sum * h / 3
```

The above method samples at (h/4, y1), (h/2, y2), and (3h/4, y3) within a given integration step h. Throughout this study, I use the approach where I apply the algorithm for each integration step h.

The left-anchor three-points hybrid Newton-Cotes integration method uses the following equation:

```
Integral = (Y1 + 3*Y3)*h/4
```

For an integral that samples at (0, y1), (h/3, y2), and (2h/3, y3) within a given integration step h. This new algorithm calculates function values starting at α and moving upward close to, but never reaching, β, where f(β) is undefined. The pseudo-code for this algorithm is:

```
Given f(x), the integration interval (A, B) and the increment h.
X = A
Sum = 0
Do
  X1 = X + 0.25 * h
  X2 = X + 0.5 * h
  X3 = X + 0.75 * h
  Sum = Sum + f(X1) + 3 * f(X3)
  X = X + h
Loop Until X > B
Integral = Sum * h / 4
```

Here is another version of the left-anchor (let’s call it *extended left-anchor*) method that samples at (0, y1), (9h/20, y2), and (9h/10, y3). The equation is:

```
Integral = (38 * Y1 + 140 * Y2 + 65 * Y3)*h/243
```

The pseudo-code for this algorithm is:

```
Given f(x), the integration interval (A, B) and the increment h.
X = A
Sum = 0
Do
  X1 = X
  X2 = X + 9 * h / 20
  X3 = X + 9 * h / 10
  Sum = Sum + 38 * f(X1) + 140 * f(X2) + 65 * f(X3)
  X = X + h
Loop Until X > B
Integral = Sum * h / 243
```

The right-anchor three-points hybrid Newton-Cotes integration method uses the following equation:

```
Integral = (3*y1 + y3)*h/4
```

For an integral that samples at (h/3, y1), (2h/3, y2), and (h, y3) within a given integration step h. This version calculates values starting just above the lower integration interval, α, and includes the value at the upper interval end, β. This method works for integrals where f(α) is undefined. The pseudo-code for this algorithm is:

```
Given f(x), the integration interval (A, B) and the increment h.
X = A
Sum = 0
Do
  X1 = X + h/3
  X2 = X + 2 * h/3
  X3 = X + h
  Sum = Sum + 3 * f(X1) + f(X3)
  X = X + h
Loop Until X > B
Integral = Sum * h / 4
```

An extended right-anchor method samples the points (h/10, y1), (11h/20, y2), and (h, y3). The equation used is:

```
Integral = (65 * Y1 + 140 * Y2 + 38 * Y3)*h/243
```

The pseudo-code for this algorithm is:

```
Given f(x), the integration interval (A, B) and the increment h.
X = A
Sum = 0
Do
  X1 = X
  X2 = X + 9 * h / 20
  X3 = X + 9 * h / 10
  Sum = Sum + 65 * f(X1) + 140 * f(X2) + 38 * f(X3)
  X = X + h
Loop Until X > B
Integral = Sum * h / 243
```

So how do the left-anchor and right-anchor methods perform compared to each other and compared to Milne’s method? Table 1 shows a summary of results for a sample of test functions. The table entries are the differences in absolute percent errors between any two algorithms. The errors are calculated for different integral intervals and also for different integration steps. The first seven functions were defined for all the points at the end of the integration interval. The function ln(x) (the function before last) is not defined at the start of the integration interval, x=0. The function ln(10-x) (the last function) is not defined at the end of the integration interval, x=10.

f(x)	Diff ABS % Err of Milne's method - Left-Anchor	Diff ABS % Err of Milne's method - Right-Anchor	Diff ABS % Err of Left-Anchor - Right-Anchor
1/x	-0.00015	4.43756E-07	0.000147913
ln(x)/x	-0.00022	-0.000427003	-0.00020681
1/(1+x^4)	4.55E-05	-4.69589E-05	-9.2498E-05
1/(1+exp(x))	-0.11928	0.009789866	0.129068906
x * sin(30 * x) * cos(x)	-9.8E-06	-9.40563E-06	4.32939E-07
1/x^4	0.068319	-0.00501006	-0.07332909
x/ln(x)	-2.7E-05	-9.1289E-06	1.78373E-05
ln(x)		-0.014986359	
ln(10-x)	-0.008715121		

Table 1. Comparing three-point integral methods.

Looking at the values in Table 1, I can say that the three methods give practically (or statistically speaking, if you prefer) comparable results. In the case of the last two functions, the absolute differences in percent error are a bit higher, because we are dealing with functions with one end of the integration interval is undefined.

What about the extended left-anchor and extended right-anchor compared with the Milne method? Table 2 shows the same type of results as in Table 1.

f(x)	Diff ABS % Err of Milne's method - Left-Anchor	Diff ABS % Err of Milne's method - Right-Anchor	Diff ABS % Err of Left-Anchor - Right-Anchor
1/x	-1.47825E-05	1.67735E-07	1.49503E-05
ln(x)/x	-1.3008E-05	-2.30907E-05	-1.0083E-05
1/(1+x^4)	3.49789E-06	-4.90769E-06	-8.4056E-06
1/(1+exp(x))	9.08736E-07	-4.22528E-07	-1.3313E-06
x * sin(30 * x) * cos(x)	-8.7485E-07	-8.36932E-07	3.79225E-08
1/x^4	0.0404723	0.004779989	-0.03569235
x/ln(x)	-2.5354E-06	-5.24364E-07	2.01099E-06
ln(x)		0.017635792	
ln(10-x)	0.010255871		

Table 2. Comparing three-point extended integral methods.

The values, for the first seven test functions, in Table 2 are smaller than their counterparts in Table 1. This difference is due to the fact that the extended left-anchor and extended right-anchor methods sample more of the integration step h. As such these extended algorithms are better than the regular left and right anchor methods that I presented earlier.

The Four-Point Hybrid Newton-Cotes Integral

This section presents two four-points hybrid Newton-Cotes integration methods. Let me first introduce an *unnamed* method (according to Wikipedia) which is the comparable Open Newton-Cotes integration method:

```
Given f(x), the integration interval (A, B) and the increment h.
X = A
Sum = 0
Do
  X1 = X + 0.2 * h
  X2 = X + 0.4 * h
  X3 = X + 0.6 * h
  X4 = X + 0.8 * h
  Sum = Sum + 11 * f(X1) + f(X2) + f(X3) + 11 * f(X4)
  X = X + h
Loop Until X > B
Integral = Sum * h / 24
```

The above method samples at (2h/10, y1), (4h/10,y2), (6h/10, y3), and (8h/10, y4) within a given step.

The left-anchor four-points hybrid Newton-Cotes integration method uses the following equation:

```
Integral = (17 * y1 + 60 * y2 + 45 * y3 + 40 * y4)*h/162
```

For an integral that samples at (0, y1), (3h/10, y2), (6h/10, y3), and (9h/10, y4) within a given step. This version calculates function values starting at α and moving upward close to, but never reaching, β. The pseudo-code for this algorithm is:

```
Given f(x), the integration interval (A, B) and the increment h.
X = A
Sum = 0
Do
  X1 = X
  X2 = X + 3 * h / 10
  X3 = X + 6 * h / 10
  X4 = X + 9 * h / 10

  Sum = Sum + 17 * f(X1) + 60 * f(X2) + 45 * f(X3) +
              40 * f(X4)
  X = X + h
Loop Until X > B
Integral = Sum * h / 162
```

The right-anchor four-points hybrid Newton-Cotes integration method uses the following equation:

```
Integral = (40 * y1 + 45 * y2 + 60 * y3 + 17 * y4)*h/162
```

For an integral that samples at (h/10, y1), (4h/10, y2), (7h/10, y3), and (h, y4) within a given step. This version calculates values starting just higher than the lower integration interval, α, and includes the value at the upper interval end, β. The pseudo-code for this algorithm is:

```
Given f(x), the integration interval (A, B) and the increment h.
X = A
Sum = 0
Do
  X1 = X + h / 10
  X2 = X + 4 * h / 10
  X3 = X + 7 * h / 10
  X4 = X + h
  Sum = Sum + 40 * f(X1) + 45 * f(X2) + 60 * f(X3) +
              17 * f(X4)
  X = X + h
Loop Until X > B
Integral = Sum * h / 162
```

Again, we ask how do the four-point left-anchor and right-anchor methods perform compared to each other and compared to the *unnamed* method? Table 3 shows a summary of results for a sample of test functions. The table entries are the differences in absolute percent errors between any two algorithms. The errors are calculated for different integral intervals and also for different integration steps.

f(x)	Diff ABS % Err of Milne's method - Left-Anchor	Diff ABS % Err of Milne's method - Right-Anchor	Diff ABS % Err of Left-Anchor - Right-Anchor
1/x	-3.73308E-07	-3.15975E-07	5.7333E-08
ln(x)/x	6.70469E-06	6.83118E-06	1.26483E-07
1/(1+x^4)	-3.4935E-07	-3.9745E-07	-4.81005E-08
1/(1+exp(x))	1.29374E-07	1.29645E-07	2.71092E-10
x * sin(30 * x) * cos(x)	6.09862E-09	6.09568E-09	-2.93771E-12
1/x^4	0.00373444	0.00073932	-0.0029951
x/ln(x)	6.38385E-08	7.47017E-08	1.08631E-08
ln(x)		0.013072584	
ln(10-x)	0.007602195		

Table 3. Comparing four-point integral methods.

The results of Table 3 pretty much agree with those in Tables 1 and 2. The three methods give practically comparable results.

Enhancing Open Newton Cotes Methods

This *bonus* section presents enhanced open Newton-Cotes method that I used earlier in this paper to compare with the new hybrid Newton-Cotes methods.

Enhancing the Milne Method

I used Milne's method earlier in this article and compared it with various hybrid Newton-Cotes methods. Milne's method samples at (h/4, y1), (h/2, y2) and (3h/4, y3) in the integration step h. In this bonus section I present a *Modified Milne* method that samples at (h/10, y1), (h/2, y2), (9h/10, y3) in the integration step h. The modified method covers a wider interval in the integration step h, than the original method. The equation for the Modified Milne method is:

```
Integral = (25*y1 + 46*y2 + 25*y3)*h/96
```

The pseudo-code for the Modified Milne method is:

```
Given f(x), the integration interval (A, B) and the increment h.
X = A
Sum = 0
Do
  X1 = X + h / 10
  X2 = X + 4 * h / 10
  X3 = X + 7 * h / 10
  Sum = Sum + 25 * f(X1) + 456 * f(X2) + 25 * f(X3) +
  X = X + h
Loop Until X > B
Integral = Sum * h / 96
```

I tested the Modified Milne method with the regular one using the seven test functions listed in the tables of this paper. In 7 out of 9 cases, the Modified Milne method performed better than the original method. So, if you are going to stick with the Milne method, I comfortably recommend the Modified Milne method that uses three points.

Before we wrap things up in this section, let’s take the Milne method to an *extreme*. This new scheme samples at (h/100, y1), (h/2, y2), (99h/100, y3) in the integration step h. The equation for the Modified Milne method is:

```
Integral = (1250*y1 + 4703*y2 + 1250*y3)*h/7203
```

The pseudo-code for the Extreme Modified Milne method is:

```
Given f(x), the integration interval (A, B) and the increment h.
X = A
Sum = 0
Do
  X1 = X + h / 100
  X2 = X + h / 2
  X3 = X + 99 * h / 100
  Sum = Sum + 1250 * f(X1) + 4703 * f(X2) + 1250 * f(X3)
  X = X + h
Loop Until X > B
Integral = Sum * h / 4703
```

I tested the Extreme Milne method with the regular one using the seven test functions listed in the tables of this paper. In 6 out of 9 cases, the Extreme Milne method performed better than the original method. Thus, I also recommend the Extreme Milne method that uses three points.

What about the difference between the Modified Milne method and the Extreme Milne method? The test involving the seven functions shows, surprisingly, mixed results with no clear winner between the two methods. It seems both methods cover a wide enough part of the integration step h.

Enhancing the Four-Point Unnamed Method

In this section I introduced an enhanced version of the four-point *unnamed* (open Newton-Cotes) integration method. I will call this one the *four-point Shammas method* since it is based on an unnamed method. The basic scheme for the

Shammas method samples function values at (h/10, y1), (34h/100, y2), (67h/100, y3), and (9h/10, y4). The equation for this method is:

```
Integral = [(4033 * Y1) / 17496 + (32200 * Y2) / 115911 +
            (575 * Y3) / 2187 + (97 * Y4) / 424]*h
```

The pseudo-code for this algorithm is:

```
Given f(x), the integration interval (A, B) and the increment h.
X = A
Sum = 0
Do
  X1 = X + h / 10
  X2 = X + 37 * h / 100
  X3 = X + 64 * h / 100
  X4 = X + 9 * h / 10
  Y1 = f(X1)
  Y2 = f(X2)
  Y3 = f(X3)
  Y4 = f(X4)
  Sum = Sum + (4033 * Y1) / 17496 + (32200 * Y2) / 115911 +
              (575 * Y3) / 2187 + (97 * Y4) / 424
  X = X + h
Loop Until X > B
Integral = Sum * h
```

As expected, the Shammas method performs better than (in 6 out of 7 test function cases) the *unnamed method* because it samples from a wider interval in the integration step h.

Conclusion

This short paper introduces you to a whole new category of numerical integration comparable to the open Newton-Cotes integration methods. The paper offers you six hybrid Newton-Cotes integration methods that you can add to your repertoire of open integration methods that deal with integrals where either the lower or the upper integral values cannot be calculated for the integrated function.

To choose between open Newton-Cotes methods and their hybrid counterpart, I suggest the following *calibration*. Test your target functions with sample parameters and known integrals using both types of integration methods. After comparing the results, select the type of Newton-Cotes method that generated less error in your *calibration* test. Apply that method for problems that you are working with.

You can use the same approach that I presented to derive hybrid Newton-Cotes integration methods that use five or more points in the integration schemes. You can also create your own versions of the three-point and four-point methods by tweaking the sampling points within the integration step h. The best scheme is to have equidistant sampling points that either start at (x=0) or end at (x=h) the integration interval.

Deriving hybrid (and even open) Newton-Cotes methods that cover as wide as possible integration steps yield more accurate methods. This enhancement is true, because the hybrid and open Newton-Cotes methods approach the same interval of values incorporated in closed Newton-Cotes methods. Stated as a general limit we can say:

$$\lim_{\gamma\to 1} Open\ NC\ Integral(\alpha,\beta,\gamma h) = Closed\ NC\ Integral(\alpha,\beta,h)$$

$$\lim_{\gamma\to 1} Hybrid\ NC\ Integral(\alpha,\beta,\gamma h) = Closed\ NC\ Integral(\alpha,\beta,h)$$

The first two parameters, $\alpha\ and\ \beta$, define the integration interval. Parameter h is the integration step size. Parameter γ is the fraction of the step size h covered by the interpolation polynomials used to calculate the equation for numerical integration.

The paper also introduced you to two variants of the Milne methods and a variant of the *unnamed method*. These variants generally perform better than the original versions since their interpolative polynomials cover a wider range of the integration steps.

Appendix A

This appendix shows sample MATLAB scripts that I used to generate the formulas for numerical integration. The first script is for the left-anchored hybrid Newton-Cotes method:

```
syms a b c d h y1 y2 y3 y4
% (0,y1), (3h/`0,y2), (6h/10, y3), (9/10h, y4)
eqn1 = d == y1
eqn2 = a*((3*h/10)^3) + b*((3*h/10)^2) + c*(3*h/10) + d == y2
eqn3 = a*((6*h/10)^3) + b*((6*h/10)^2) + c*(6*h/10) + d  == y3
eqn4 = a*((9*h/10)^3) + b*((9*h/10)^2) + c*(9*h/10)+ d  == y4
[A,B] = equationsToMatrix([eqn1, eqn2, eqn3, eqn4], [a, b, c,
d])
X=linsolve(A,B)
```

```
fprintf('Leaving out h factor out we get\n')
Area = 1/4*X(1)*h^3 + 1/3*X(2)*h^2 +1/2*X(3)*h + X(4)

% Output is
%
% Leaving out h factor out we get
%
% Area =
%
% (17*y1)/162 + (10*y2)/27 + (5*y3)/18 + (20*y4)/81
```

The next MATLAB script is for the right-anchored hybrid Newton-Cotes method:

```
syms a b c d h y1 y2 y3 y4
% (h/10,y1), (4h/10, y2), (7/10h, y3), (h, y4)
eqn1 = a*((h/10)^3) + b*((h/10)^2) + c*(h/10) + d == y1
eqn2 = a*((4*h/10)^3) + b*((4*h/10)^2) + c*(4*h/10) + d  == y2
eqn3 = a*((7*h/10)^3) + b*((7*h/10)^2) + c*(7*h/10)+ d  == y3
eqn4 = a*h^3 + b*h^2 + c*h + d  == y4
[A,B] = equationsToMatrix([eqn1, eqn2, eqn3, eqn4], [a, b, c,
d])
X=linsolve(A,B)
fprintf('Leaving out h factor out we get\n')
Area = 1/4*X(1)*h^3 + 1/3*X(2)*h^2 +1/2*X(3)*h + X(4)

% output is
%
% Leaving out h factor out we get
%
% Area =
%
% (20*y1)/81 + (5*y2)/18 + (10*y3)/27 + (17*y4)/162
```