Open Journal of Applied Sciences
Supplement: 2012 world Congress on Engineering and Technology

Interval Integration Revisited

Sérgio Galdino
Polytechnic School of Pernambuco University
Rua Benfica, 455 - Madalena
50.750-470 - Recife - PE - Brazil
Email: galdino@poli.br

***Abstract*—We present an overview of approaches to self-validating one-dimensional integration quadrature formulas and a verified numerical integration algorithm with an adaptive strategy. The new interval integration adaptive algorithm delivers a desired integral enclosure with an error bounded by a specified error bound. The adaptive technique is usually much more efficient than Simpson's rule and narrow interval results can be reached.**

***Index Terms*—Interval computing, self-validating methods, numerical integration & interval arithmetic.**

I. Introduction

Numerical integration (quadrature) formulas are of the form

$$I = \int_a^b f(x)dx = \sum_{i=0}^{n} w_i^{(n)} f(x_i) + E,$$

with $a = x_0 < x_1 < x_2 < \cdots < x_n = b$ and weights $w_i^{(n)}$, it is possible to derive a formula for the local error of the method based on a higher derivative of $f(x)$ at an intermediate point ξ as

$$E = C \cdot f^{(k)}(\xi), \text{where } \xi \in [a,\ b],$$

the factor C depends on the method and usually is a power of the width $b-a$.

Automatic integration programs may print out the 'theoretical' error which has been achieved and which is used to provide shutoff. Exist some pitfalls of practical working of automatic integration programs. The tolerance requested may be impossible to meet. When more and more points are called the roundoff error enter to worsen progressively the result. On theoretical side, the estimate of accuracy validity depends on theoretical information about the function such as theoretically accurate bounds on derivatives, monotonicity, convexity, etc.

With interval analyses, it is possible to determine verified bounding of analytically derived quadrature formulas for integration rules. The evaluation code derivative $f^{(k)}$ may be obtained from automatic differentiation. Different from classical schemes, they do not require an *a-priory* derivation of analytical error bounds, furthermore the rigorous bounds are calculated automatically in parallel to integral computation allowing better error control.

Adaptive quadrature is another area in which interval methods is useful. This is because, due to the form of the error term, meaningful interval enclosures for the actual integral can easily be computed. Replacing heuristic error estimates by these rigorous enclosures results in a quadrature algorithm that produces guaranteed bounds on the actual integral. A common interval algorithm is to integrate, with automatic differentiation software, high-order and variable degree Taylor polynomial approximations to f [1], [2], [3].

The next section briefly introduces interval arithmetics. Section 3 describes the interval integration and presents the approach adopted to the respective interval integrations. Section 4 presents numerical experiments and the related analysis results. Section 5 concludes.

II. Interval Arithmetic

A form of interval arithmetic perhaps first appeared in 1924 and 1931 in [4], [5], then later in [6]. Modern development of interval arithmetic began with R. E. Moore's dissertation [7] as a method for determining absolute errors of an algorithm, considering all data errors and rounding, after R.E. Moore introduced interval analysis [8]. Interval arithmetic is an arithmetic defined on sets of intervals, rather than sets of real numbers. The power of interval arithmetic lies in its implementation on computers. In particular, outwardly rounded computations allows rigorous enclosures for the ranges of operations and functions.

A. Notation

Throughout this paper, all scalar variables is denoted by ordinary lowercase letters (a). Interval variables are enclosed in square brackets [a]. Underscores and overscores denote lower and upper bounds, respectively. Angle brackets $\langle\ ,\ \rangle$ are used for defining intervals by midpoints and radius.

A real interval $[x]$ is a nonempty set of real numbers

$$[x] = [\underline{x}, \overline{x}] = \{\tilde{x} \in \mathsf{R} : \underline{x} \le \tilde{x} \le \overline{x}\} \tag{1}$$

where $\underline{x}$ and $\overline{x}$ are called the *infimum (inf)* and *supremum (sup)*, respectively, and $\tilde{x}$ is a point value belonging to an interval variable $[x]$.

The set of all intervals R is denoted by $I(\mathsf{R})$ where

$$I(\mathsf{R}) = \{[\underline{x}, \overline{x}] : \underline{x}, \overline{x} \in \mathsf{R} : \underline{x} \le \overline{x}\} \tag{2}$$

The midpoint of $[x]$ is defined as,

$$mid[x] = \frac{1}{2}(\underline{x} + \overline{x}), \tag{3}$$

the width of $[x]$ is defined as,

$$wid[x] = (\overline{x} - \underline{x}) \tag{4}$$

and the radius of $[x]$ is defined as,

$$rad[x] = \frac{1}{2}(\overline{x} - \underline{x}) = \frac{1}{2}wid[x] \tag{5}$$

The radius may also be used to define an interval $[x] \in I(\mathsf{R})$. Then $\langle m,\ r\rangle$ denotes an interval with midpoint m and radius r. The $[x,\ x] \equiv x$ is called point interval or thin interval. A point or thin interval has zero radius and a thick interval has a radius greater than zero.

III. Interval Integration

From mean value theorem there exists a $\xi \in [x] = [\underline{x},\ \overline{x}]$ such that

$$\int_{\underline{x}}^{\overline{x}} f(x)dx = wid([x])f(\xi).$$

Splitting the interval [x] into n subinterval with break points $\{x_i\}$, $a = x_0 < x_1 < \cdots < x_{n-1}, x_n = b$, then

$$I = \sum_{i=1}^{n} \int_{x_{i-1}}^{x_i} f(x)dx = \sum_{i=1}^{n} wid([x]_i)f(\xi_i),$$

where $[x]_i = [x_{i-1},\ x_i]$ and $\xi_i \in [x]_i$.

Let [f]([x]) be an interval extension of $f(x)$ on $[x]$. Then

$$I_i \equiv w([x]_i)f(\xi_i) \subseteq [R]_i \equiv w([x]_i)[f]([x]_i),$$

and we have an interval inclusion of I,

$$I \subseteq [R] \equiv \sum_{i=1}^{n} [R]_i \tag{6}$$

The width of $[R]$ measures the quality of determined integral given bounds to both rounding errors and truncation errors.

A. Interval Trapezium Rule

If $f(x)$ is two times continuously differentiable, then

$$Ti \equiv \frac{wid([x]_i)}{2}(f(\underline{x}_i) + f(\overline{x}_i)) - \frac{wid([x]_i)^3}{12} f^{(2)}(\xi_i), \tag{7}$$

where $\xi_i \in [x]_i$. If we use interval arithmetic extension $[f]$, $[f]^{(2)}$, then we use these to get

$$I \subseteq [T] \equiv \sum_{i=1}^{n} [T]_i \tag{8}$$

with

$$[T]_i = \frac{wid([x]_i)}{2}([f](\underline{x}_i) + [f](\overline{x}_i)) - \frac{wid([x]_i)^3}{12}[f]^{(2)}([x]_i). \tag{9}$$

Note that the call of [f] with thin interval arguments return intervals that includes rounding errors.

B. Interval Simpson $1/3$

If f(x) is four times continuously differentiable, then

$$\begin{aligned} S_i = &\tfrac{wid([x]_i)}{6}\left(f(\underline{x}_i) + 4f(mid([x]_i) + f(\overline{x}_i)\right) \\ &- \tfrac{wid([x]_i)^5}{2880} f^{(4)}(\xi_i) \end{aligned} \tag{10}$$

where $\xi_i \in [x]_i$. If we use interval arithmetic extension $[f]$, $[f]^{(4)}$, then we use these to get

$$I \subseteq [S] \equiv \sum_{i=1}^{n} [S]_i \tag{11}$$

with

$$\begin{aligned} [S]_i = &\tfrac{wid([x]_i)}{6}\left([f](\underline{x}_i) + 4[f](mid([x]_i) + [f](\overline{x}_i)\right) \\ &- \tfrac{wid([x]_i)^5}{2880}[f]^{(4)}([x]_i) \end{aligned} \tag{12}$$

Note that the call of [f] with thin interval arguments return intervals that includes rounding errors.

C. Adaptive quadrature

The drawback with any algorithm based on the composite quadrature rules is that it makes no attempt to respond to the local form of the integrand. The objective of an adaptive scheme is to use an unequal mesh spacing and to determine the size of each subinterval so as to satisfy the overall accuracy requirement with the minimum number of subintervals (and consequently the minimum number of evaluations of the integrand).

Total approximate integral = Sum of approximate integrals over subintervals.

Total estimated error = Sum of estimated errors over subintervals:

- Locally Adaptive Quadrature:
  Error for each subinterval $\leq$ global error target $\times \dfrac{\text{length of subinterval}}{b-a}$
  Note that locally adaptive quadrature is a natural example of recursion.
- Globally Adaptive Quadrature:
  Error for all subintervals $\leq$ global error target

Adaptive algorithms are just as efficient and effective as traditional algorithms for "well-behaved" integrands, but can be also effective for "badly-behaved" integrands for which traditional algorithms fail.

D. Adaptive Simpson's method

Adaptive Simpson's method, also called adaptive Kuncir's Simpson rule, was proposed by G.F. Kuncir in 1962 [9]. Adaptive Simpson's method uses an estimate of the error we get from calculating a definite integral using Simpson's rule. If the error exceeds a user-specified tolerance, the algorithm calls for subdividing the interval of integration in two and applying adaptive Simpson's method to each subinterval in a recursive manner. A criterion for determining when to stop subdividing an interval, suggested by J.N. Lyness [10], is

$$|S(a,c) + S(c,b) - S(a,b)|/15 < \tau \tag{13}$$

where $[a,\ b]$ is an interval with midpoint c, $S(a,c)$, $S(c,b)$, and $S(a,b)$ are the estimates given by Simpson's rule on the corresponding intervals and τ is the desired tolerance for the interval.

E. Adaptive Interval Simpson's method

If we are using recursion, the implementation of an interval algorithm is simple. The following Matlab with the Intlab Toolbox [12] function is based on `adaptsimp.m` [13].

```
function int = Iadaptsmp(f,f4,X,tol,lev,fa,fm,fb)
global n
global D
%IADAPTSMP Interval Adaptive Simpson 1/3 rule
echo on
if nargin == 4
lev = 1;
a=inf(X);
b=sup(X);
D=diam(X);
fa = feval(f,a);
fm = feval(f,(a+b)/2);
fb = feval(f,b);
n=3;
end
% recursive calls start here
% checking for too many levels of recursion
if lev > 1000
disp('1000 levels of recursion reached.')
X=infsup(a,b);
wx=diam(X);
int = (b-a)*(fa+4*fm+fb)/6 -wx^5/2880*f4(X);
else
% Divide the interval in half and apply interval
% Simpson 1/3 rule on each half. As an verified
% error estimate: rad(int) > 2*tol*h/D;
a=inf(X);
b=sup(X);
h = b - a;
wx=h/2;
flm = feval(f,a+h/4);
frm = feval(f,b-h/4);
n=n+2;
X=infsup(a,a+h/2);
simpl = h*(fa + 4*flm + fm)/12 - wx^5/2880*f4(X);
X=infsup(a+h/2,b);
simpr = h*(fm + 4*frm + fb)/12 - wx^5/2880*f4(X);
int = simpl + simpr;
X=infsup(a,b);
% err =  rad(int) > 2*tol*h/D
% tolerance not satisfied, recursively refine
if rad(int) > 2*tol*h/D;
m = (a + b)/2;
XL=infsup(a,m);
XR=infsup(m,b);
int = Iadaptsmp(f,f4,XL,tol,lev+1,fa,flm,fm) ...
+ Iadaptsmp(f,f4,XR,tol,lev+1,fm,frm,fb);
end
end
```

Sample Matlab M-flle with the Intlab Toolboxthat calls the above function is followed by running results:

```
format long
% intvalinit('DisplayMidRad')
global n
f = @(x)  23/25*cosh(x)-cos(x)
f4 = @(x)  23/25*cosh(x)-cos(x)
X=infsup(-1,1)
err=1.0E-12;
IADAPTSMP(f,f4,X,err)
n
err1=rad(ans)

>> ADAPTXI
intval ans =
<   0.47942822668878,  0.00000000000047>
n =
   241
err1 =
    4.607425552194400e-013
>>
```

IV. Numerical experiments

In the following, we will use the "battery" test of functions which are a subset of the set used by Gonnet [11]. The battery of test functions are listed in Table I. The result has been calculated using 20 digits of precision with the Mathcad computer software.

The main mechanism for get a narrow interval results is to increase the number of subdivisions. Next we consider the progression of the discretization error as we increase n. Table II shows that in the early stages increasing n improves the discretization.

We observe however that if the number of subdivisions were extremely large this might lead to significant round-of (more terms in the sum, each with a round-off error to contribute). Verified trapezoidal rule with 10^5 subdivisions is wider than 10^4 subdivisions and verified Simpson 1/3 rule with 10^4 subdivisions is wider than 10^3. A narrow interval implies high precision; we can specify plausible values to within a tiny range. Verified step rule interval results are narrowing (from 10 to 10^6 subdivisions), but intervals are wider, compared with verified trapezoidal and Simpson 1/3 rules, implies poor precision.

In Table III we present the results of numerical experiments with interval adaptive Simpson 1/3 rule. The battery of test functions are listed in Table I. For these test functions we will consider the usual metrics of efficiency,i.e. number of function evaluations required for a given accuracy. We have tested the code on four radius interval tolerances $\tau = 10^{-3}, 10^{-6}, 10^{-9}, 10^{-12}$.

Interval adaptive Simpson 1/3 algorithm was more efficient than interval Simpson 1/3, the only exception is with f_1 for $\tau = 10^{-3}$. Should be stressed that interval tolerance $\tau = 10^{-12}$ is not reached, by the interval Simpson 1/3 algorithm, for f_9, f_{10}, f_{11} test functions, with the used precision.

V. Conclusion

The design of interval iterative formulas for self-validating one-dimensional integration quadrature formulas is very important and is also an interesting task in interval computing. In this paper, we have proposed a new (supposed) one-dimensional interval integration adaptive algorithm, to get rigorous bounds of a desired integral. Numerical tests demonstrate that interval adaptive technique is usually more efficient than interval Simpson's rule with narrow interval results. Finding self-validating one-dimensional definite integration by derivative-free interval methods should be considered in future studies.

TABLE I
THE FUNCTIONS USED FOR THE "BATTERY" TEST.

$$f_1 = \int_0^1 e^x dx = 1.7182818284590452354$$

$$f_2 = \int_{-1}^1 \left(\frac{23}{25} cosh(x) - cos(x)\right) dx = 0.47942822668880166736$$

$$f_3 = \int_{-1}^1 \left(x^4 + x^2 + 0.9\right)^{-1} dx = 1.5822329637296729331$$

$$f_4 = \int_0^1 \left(1 + x^4\right)^{-1} dx = 0.86697298733991103757$$

$$f_5 = \int_0^1 2\left(2 + sin(10\pi x)\right)^{-1} dx = 1.154700538379251529$$

$$f_6 = \int_0^1 (1+x)^{-1} dx = 0.69314718055994530942$$

$$f_7 = \int_0^1 (1+e^x)^{-1} dx = 0.37988549304172247537$$

$$f_8 = \int_{0.1}^1 sin(100\pi x)/(\pi x)\, dx = 0.00909863753916684291 56$$

$$f_9 = \int_0^{10} \sqrt{50}\, e^{-50\pi x^2} dx = 0.5$$

$$f_{10} = \int_0^{10} 25\, e^{-25x} dx = 1.0$$

$$f_{11} = \int_0^{10} 50\left(\pi\left(2500x^2+1\right)\right)^{-1} dx = 0.49936338107645674464$$

$$f_{12} = \int_{-1}^1 \left(1.005 + x^2\right)^{-1} dx = 1.5643964440690497731$$

$$f_{13} = \int_0^1 \left(1 + (230x - 30)^2\right)^{-1} dx = 0.013492485649467772692$$

TABLE II
ESTIMATES OF $\int_{-1}^{1} cosh(x)\frac{23}{25} - cos(x)dx = \left[sinh(x)\frac{23}{25} - sin(x)\right]_{-1}^{1}$.

Analytical Answer Bounds $= [0.47942822668880,\ 0.47942822668881]$

Step Rule with Verification

Subdivisions	Verified Result
10	$\langle\ 0.492244876649698,\ 0.191866375632378\ \rangle$
100	$\langle\ 0.479556403654468,\ 0.019186637563241\ \rangle$
1000	$\langle\ 0.479429508459513,\ 0.001918663756343\ \rangle$
10000	$\langle\ 0.479428239506509,\ 1.918663758142536e-004\ \rangle$
100000	$\langle\ 0.479428226816973,\ 1.918663937550136e-005\ \rangle$
1000000	$\langle\ 0.479428226690078,\ 1.918681864720995e-006\ \rangle$

Trapezoidal Rule with Verification

Subdivisions	Verified Result
10	$\langle\ 0.479421870930105,\ 6.395545854416262e-004\ \rangle$
100	$\langle\ 0.479428226049601,\ 16.395545891768606e-007\ \rangle$
1000	$\langle\ 0.479428226688739,\ 6.395906027023557e-010\ \rangle$
10000	$\langle\ 0.479428226688802,\ 1.002808946992673e-012\ \rangle$
100000	$\langle\ 0.479428226687608,\ 3.629152534045943e-012\ \rangle$
1000000	$\langle\ 0.479428226671977,\ 3.620936883663717e-011\ \rangle$

Simpson $1/3$ Rule with Verification

Subdivisions	Verified Result
10	$\langle\ 0.479428217025575,\ 2.131848625963606e-007\ \rangle$
100	$\langle\ 0.479428226688792,\ 2.139066701545289e-012\ \rangle$
1000	$\langle\ 0.479428226688796,\ 7.244205235679146e-014\ \rangle$
10000	$\langle\ 0.479428226688723,\ 7.243095012654521e-013\ \rangle$
100000	$\langle\ 0.479428226688001,\ 7.241873767327434e-012\ \rangle$
1000000	$\langle\ 0.479428226680795,\ 7.241879318442557e-011\ \rangle$

TABLE III
RESULTS OF BATTERY TEST: ADAPTIVE AND SIMPSON 1/3 RULES WITH VERIFICATION

$f(x)$	**Algorithms: Adaptive Simpson** $1/3$ **(Simpson** $1/3$**)**			
	$\tau = 10^{-3}$	$\tau = 10^{-6}$	$\tau = 10^{-9}$	$\tau = 10^{-12}$
f_1	5 (3)	9 (12)	33 (39)	129 (150)
f_2	5 (6)	17 (21)	65 (78)	241 (306)
f_3	25 (30)	97 (108)	361 (423)	1441 (1701)
f_4	9 (12)	33 (45)	109 (174)	429 (693)
f_5	105 (135)	393 (522)	1489 (2064)	5921 (8718)
f_6	5 (6)	13 (18)	49 (63)	189 (252)
f_7	5 (6)	17 (18)	65 (69)	257 (273)
f_8	445 (516)	1797 (2070)	7077 (8241)	28125 (32922)
f_9	49 (684)	133 (2640)	449 (10488)	1725 (–)
f_{10}	49 (531)	125 (2106)	433 (8376)	1681 (–)
f_{11}	89 (2916)	305 (10833)	1193 (42870)	4765 (–)
f_{12}	17 (24)	57 (84)	241 (330)	945 (1323)
f_{13}	57 (519)	153 (1872)	541 (7323)	2161 (29193)

TABLE IV
RESULTS OF BATTERY TEST: ADAPTIVE SIMPSON 1/3 RULE WITH VERIFICATION

Integral f	**Adaptive Simpson 1/3** $(\tau = 10^{-12})$	
	Verified Result	Error
f_1	$< 1.71828182845904,\ 0.00000000000029 >$	0.29×10^{-12}
f_2	$< 0.47942822668878,\ 0.00000000000047 >$	0.47×10^{-12}
f_3	$< 1.58223296372967,\ 0.00000000000048 >$	0.48×10^{-12}
f_4	$< 0.86697298733961,\ 0.00000000000072 >$	0.72×10^{-12}
f_5	$< 1.15470053839294,\ 0.00000000000056 >$	0.56×10^{-12}
f_6	$< 0.69314718055994,\ 0.00000000000056 >$	0.56×10^{-12}
f_7	$< 0.37988549304172,\ 0.00000000000019 >$	0.19×10^{-12}
f_8	$< 0.00909863753917,\ 0.00000000000044 >$	0.44×10^{-12}
f_9	$< 0.50000000000000,\ 0.00000000000004 >$	0.04×10^{-12}
f_{10}	$< 0.99999999999999,\ 0.00000000000008 >$	0.08×10^{-12}
f_{11}	$< 0.49936338107645,\ 0.00000000000062 >$	0.62×10^{-12}
f_{13}	$< 1.56439644405124,\ 0.00000000000057 >$	0.57×10^{-12}
f_{13}	$< 0.01349248564896,\ 0.00000000000054 >$	0.54×10^{-12}

REFERENCES

[1] G. F. Corliss and L. B. Rall: Adaptive, self-validating numerical quadrature. SIAM J. Sci. Statist. Comput. 8(5):831–47,(1985)

[2] R. Kelch. Ein adaptives Verfahren zur numerischen Quadratur mit automatischer Ergebnisverifikation. PhD thesis, Universität Karlsruhe, (1989)

[3] U. Storck. Numerical integration in two dimensions with automatic result verification. In E. Adams and U. Kulisch, editors, Scientific Computing with Automatic Result Verification,Academic Press, New York, etc., 187–224, (1993) .

[4] J. C. Burkill: Functions of intervals. Proceedings of the London Mathematical Society, 22:375-446, (1924)

[5] R. C. Young: The algebra of many-valued quantities. Math. Ann. 104:260-290, (1931)

[6] T . Sunaga: Theory of an Interval Algebra an d its Application to Numerical Analysis. Gaukutsu Bunken Fukeyu-kai, Tokyo, (1958)

[7] R. E. Moore: Interval Arithmetic and Automatic Error Analysis in Digital Computing. PhD thesis, Stanford University, October, (1962)

[8] R.E. Moore: *Interval Analysis*. Prentice Hall, Englewood Clifs, NJ, USA, (1966)

[9] G.F. Kuncir: Algorithm 103: Simpson's rule integrator. Communications of the ACM 5(6): 347, (1962)

[10] J.N. Lyness: Notes on the adaptive Simpson quadrature routine. Journal of the ACM 16 (3): 483–495, (1969)

[11] P. Gonnet: Adaptive quadrature re-revisited. ETH Zürich Thesis Nr. 18347 (2009). http://dx.doi.org/10.3929/ethz-a-005861903

[12] S.M. Rump: INTLAB - INTerval LABoratory. Tibor Csendes, Developments in Reliable Computing, Kluwer Academic Publishers, Dordrecht, 77–104, (1999), http://www.ti3.tu-harburg.de/rump/

[13] Douglas N. Arnold: A Concise Introduction to Numerical Analysis. Institute for Mathematics and its Applications, University of Minnesota, Minneapolis, MN 55455 (2001). http://www.ima.umn.edu/~arnold/