A Note on a Verified Automatic Integration Algorithm[*]

Naoya Yamanaka
Graduate School of Science and Engineering
, Waseda University
yamanaka@suou.waseda.jp

Masahide Kashiwagi
Faculty of Science and Engineering,
Waseda University

Shin'ichi Oishi
Faculty of Science and Engineering,
Waseda University

Takeshi Ogita
Department of Mathematics
, Tokyo Woman's Christian University

Abstract

A verified integration algorithm is proposed for calculating s-dimensional integrals over a finite domain using numerical computations. To construct an efficient verified numerical integrator, the truncation error and the rounding error need to be considered. It has been known that interval arithmetic is one of the most efficient methods of evaluating the rounding error. However, it is much slower than pure floating-point arithmetic, so that in an inclusion algorithm for integrals, the computational effort by the interval arithmetic tends to become a large part. To overcome this problem, an algorithm for evaluating the rounding error using floating-point computations is proposed. The proposed algorithm is based on calculating a priori error bounds for function evaluations and an accurate sum algorithm. With the use of the proposed algorithm and a inclusion algorithm for evaluating the truncation error, we propose an automatic inclusion algorithm. Numerical examples are presented for illustrating the effectiveness of the proposed algorithm.

Keywords: Numerical integration, rounding error, verification
AMS subject classifications: 65G20, 65G50

[*]Submitted: January 26, 2009; Accepted: March 5, 2010.

1 Introduction

This paper is concerned with a verified numerical computation of the definite integral. The purpose of this paper is to present an efficient automatic inclusion algorithm for a certain class of definite integrals. A type of s-dimensional integrals considered in this paper is as follows:

$$I_s = \int_\Omega f(\mathbf{x})\, d\mathbf{x}, \tag{1}$$

where $f : \mathbb{R}^s \to \mathbb{R}$ and $\Omega = [a_1, b_1] \times [a_2, b_2] \times \cdots \times [a_s, b_s]$. In general, an s-dimensional integral can be approximated by a one-dimensional integration algorithm recursively as

$$I_s \approx \hat{I} = \sum_{i_1} \sum_{i_2} \cdots \sum_{i_s} w_{i_1} w_{i_2} \cdots w_{i_s} f(x_{i_1}, x_{i_2}, \cdots, x_{i_s}), \tag{2}$$

where w_{i_k} and x_{i_k} $(k = 1, \cdots, s)$ denote weights and points of the one-dimensional integration algorithm. Here, we assume in this paper that all weights of the integration algorithm used are positive.

We consider the estimation of an error

$$|I_s - \mathbf{res}|,$$

where **res** denotes an approximate value of Eq. (1) by a certain s-dimensional integration algorithm. For example, $\mathrm{fl}\left(\hat{I}\right)$ epitomizes **res**, where fl (formula) means that the formula inside the parenthesis is calculated by the floating point calculations under the rounding to the nearest mode. Here, we divide the error into two parts

$$|I_s - \mathbf{res}| \le |I_s - \hat{I}| + |\hat{I} - \mathbf{res}|.$$

We introduce notations E_t and E_r by

$$E_t = |I_s - \hat{I}| \tag{3}$$

and

$$E_r = |\hat{I} - \mathbf{res}|,$$

respectively. E_t and E_r are called the truncation error and the rounding error, respectively. Obviously, we have

$$|I_s - \mathbf{res}| \le E_t + E_r.$$

It is known that interval arithmetic is one of the most efficient methods for evaluating E_r. The idea of the method is to calculate upper and lower endpoints for the range of values of every operation. However, since the pure floating-point arithmetic is replaced with interval arithmetic in a verified algorithm, it is much slower than the pure floating-point arithmetic, so that in an inclusion algorithm for integrals, the computational effort by the interval arithmetic tends to become a large part of that of calculating **res**.

To overcome this problem, we propose an algorithm for evaluating the rounding error E_r (**Algorithm 4**). In **Algorithm 4**, we suggest that an approximate value of Eq. (1) be calculated by

$$\mathbf{res} = \mathbf{PrecSum}\left(\sum_{i_1} \sum_{i_2} \cdots \sum_{i_s} \mathrm{fl}\left(w_{i_1} w_{i_2} \cdots w_{i_s} f(x_{i_1}, x_{i_2}, \cdots, x_{i_s})\right), K\right),$$

which K denotes an integer obtained in the proposed algorithm, and **PrecSum** $(\cdot, K)$ denotes the calculations by the K*-fold* accurate sum algorithm **PrecSum** proposed by Rump, Ogita and Oishi [3]. Since **Algorithm 4** is designed to compute all rounding errors that occur throughout the calculation of **res**, the proposed algorithm makes it possible to calculate **res** by the pure floating-point arithmetic.

Namely, **Algorithm 4** is based on two methods: Kashiwagi's method and the error analysis of **PrecSum**. Kashiwagi's method is an algorithm for calculating a priori error bounds of function evaluations using floating-point computations. For a function $g(\mathbf{x}) : \mathbb{R}^s \to \mathbb{R}$, this algorithm calculates for any $\mathbf{x}$ in the domain Ω a global constant ε_f satisfying

$$\max_{\mathbf{x}\in\Omega} |g(\mathbf{x}) - \mathrm{fl}\,(g(\mathbf{x}))| \leq \varepsilon_f.$$

Since numerical integration algorithms generally compute function values at a number of different points as mentioned above, one can expect that the algorithm evaluating the function by pure floating-point operations with a priori error bound ε_f becomes much faster than the algorithm based on the interval arithmetic. Accurate sum algorithm **PrecSum** is used for calculating the main summands of **res**. The error analysis of **PrecSum** is useful for evaluating the rounding error E_r since the result of this algorithm doesn't depend on the condition number, something relevant for these summands.

We assume in this paper that a tolerance for the numerical integration error is given and an inclusion algorithm for evaluating the truncation error E_t (*e.g.* Petras's algorithm [2]) have already been chosen. Then, with the use of **Algorithm 4** and the inclusion algorithm for evaluating E_t, we propose an automatic inclusion algorithm for Eq. (1) (**Algorithm 5**). Here, an automatic inclusion algorithm means an algorithm calculating an interval $\tilde{I}_s$ s.t.

$$I_s = \int_\Omega f(\mathbf{x})\, d\mathbf{x} \in \tilde{I}_s, \quad \left|\frac{I_s - \tilde{I}_s}{I_s}\right| \leq \frac{\mathrm{rad}(\tilde{I}_s)}{|I_s|} \leq \varepsilon_{rel},$$

where ε_{rel} is the given relative tolerance and $\mathrm{rad}(\tilde{I})$ denotes the radius of $\tilde{I}$.

By numerical experiments it is shown that **Algorithm 4** is efficient, i.e., we will present numerical examples which show inclusions of the rounding error E_r by **Algorithm 4** can be obtained with less computational time than that by the interval arithmetic.

In this paper, we assume that in the function description fl (formula) if the formula includes intervals, its value is evaluated as an interval using the interval arithmetic. Furthermore, throughout this paper, we assume floating-point arithmetic adhering to IEEE standard 754, and neither overflow nor underflow occur.

2 An algorithm for evaluating rounding errors and its error analysis

In verified numerical computations, all rounding errors that occur throughout the algorithm must be taken into account. Although the rounding errors can be calculated by interval arithmetic, it is much slower than pure floating-point arithmetic.

To overcome this problem, we propose an algorithm which gives a method of computing the upper bound of E_r. In the algorithm, two algorithms are used: one is an algorithm for calculating a priori error bounds of function evaluations using floating-point computations, which is proposed by Kashiwagi, and another is accurate sum algorithm **PrecSum** proposed by Rump, Ogita and Oishi [3].

In the section 2.2, we present the algorithm in detail and present a theorem on its error analysis.

2.1 Preliminaries

2.1.1 Kashiwagi's method

Now, let us briefly explain Kashiwagi's method. This method is based on the concept of *operator overloading*. Assume that a univariate function $f(x)$ can be calculated by the algorithm recursively using

(1) the four arithmetic operators $\{+,-,\times,/\}$,

(2) elementary functions such as $\sin,\cos,\exp,\log$ and so on,

(3) and composition of functions.

Then, consider the data type (J,ε), where J is an interval and ε is a positive number. For this data type, the four arithmetic operations are defined by

(1) (addition)

$$(J_1,\varepsilon_1)+(J_2,\varepsilon_2)=\left(J_1+J_2,\ \mathrm{fl}_+\left(\varepsilon_1+\varepsilon_2+\mathbf{u}\sup|J_1+J_2|\right)\right),$$

(2) (subtraction)

$$(J_1,\varepsilon_1)-(J_2,\varepsilon_2)=\left(J_1-J_2,\ \mathrm{fl}_+\left(\varepsilon_1+\varepsilon_2+\mathbf{u}\sup|J_1-J_2|\right)\right),$$

(3) (multiplication)

$$(J_1,\varepsilon_1)\cdot(J_2,\varepsilon_2)=\left(J_1\cdot J_2,\ \mathrm{fl}_+\left(\varepsilon_1\sup|J_2|+\varepsilon_2\sup|J_1|+\mathbf{u}\sup|J_1\cdot J_2|\right)\right),$$

(4) (division)

$$(J_1,\varepsilon_1)\,/\,(J_2,\varepsilon_2)=\left(J_1/J_2,\ \mathrm{fl}_+\left(\varepsilon_1\sup|1/J_2|+\varepsilon_2\sup\left|J_1/J_2^2\right|+\mathbf{u}\sup|J_1/J_2|\right)\right),$$

where $\mathbf{u}$ denotes the unit roundoff, for example, in the double precision format of the IEEE 754 floating point number standard, $\mathbf{u}$ is 2^{-53}. The notation fl_+ (formula) and fl_- (formula) mean that the formula inside the parenthesis is calculated with floating point arithmetic under the rounding toward $+\infty$ and $-\infty$ modes, respectively. For $J_1=[a,b]$, $J_2=[c,d]$ and $*\in\{+,-,\cdot,/\}$, $J_1{*}J_2$ means the machine interval arithmetic defined by

$$\begin{aligned}
J_1+J_2 &= [\mathrm{fl}_-(a+c)\,,\ \mathrm{fl}_+(a+c)]\,,\\
J_1-J_2 &= [\mathrm{fl}_-(a-d)\,,\ \mathrm{fl}_+(b-c)]\,,\\
J_1\cdot J_2 &= [\min A_-,\ \max A_+]\,,\\
&\quad \begin{cases} A_+=\{\mathrm{fl}_+(ac),\mathrm{fl}_+(ad),\mathrm{fl}_+(bc),\mathrm{fl}_+(bd)\}\\ A_-=\{\mathrm{fl}_-(ac),\mathrm{fl}_-(ad),\mathrm{fl}_-(bc),\mathrm{fl}_-(bd)\}\,,\end{cases}\\
J_1/J_2 &= [\min B_-,\ \max B_+]\,,\\
&\quad \begin{cases} B_+=\{\mathrm{fl}_+(a/c),\mathrm{fl}_+(a/d),\mathrm{fl}_+(b/c),\mathrm{fl}_+(b/d)\}\\ B_-=\{\mathrm{fl}_-(a/c),\mathrm{fl}_-(a/d),\mathrm{fl}_-(b/c),\mathrm{fl}_-(b/d)\}\,.\end{cases}
\end{aligned}$$

Here, in the definition of J_1/J_2, we assume that $0 \notin J_2$. Let $g(x)$ be a continuously differentiable function $g(x)$. Then, $g((J,\varepsilon))$ is defined as

$$g((J,\varepsilon)) = (g(J), \varepsilon \sup |g'(J)| + \mathbf{u} \sup |g(J)|).$$

Here, we assume that we have a software library which evaluate the function g with the following accuracy:

$$|g_{\mathbf{fl}}(x) - g(x)| \le |g(x)|\mathbf{u},$$

where $g_{\mathbf{fl}}(\cdot)$ is an evaluated value of $g(\cdot)$ by this library using the floating point operations. It is known that CRlibm library [5] fulfills this requirement for a number of elementary functions.

Let us assume that a program language supporting *operator overloading*, such as C++ or MATLAB, is used. Then, for a given function g, the following bottom-up algorithm using the object (J,ε) calculates a global constant ε satisfying

$$\max_{a\le x\le b} |g_{\mathbf{fl}}(x) - g(x)| \le \varepsilon :$$

Algorithm 1 *Computation of an a priori error bound on the rounding errors ε_y when evaluating $g(\xi) : \mathbb{R} \to \mathbb{R}$ for any ξ $(a \le \xi \le b,\ \xi \in \mathbb{F})$ by floating-point operations, and an interval J_y satisfying*

$$\inf J_y \le g([a,b]) \le \sup J_y.$$

```
Step 1  Set an interval $J = [a,b]$.
Step 2  Put $x = (J, \max(|a|,|b|)\,\boldsymbol{u})$.
Step 3  Calculate $(J_y, \varepsilon_y) = g(x)$ using operator overloading.
Step 4  Output $\varepsilon_y$ and $J_y$.
```

Algorithm 1 is easily applicable to multi-dimensional problems. For a given function $g(\mathbf{x})$, the following bottom-up algorithm using the object $(\mathbf{J},\varepsilon)$ calculates a global constant ε for any $\mathbf{x}$ in the domain Ω satisfying

$$\max_{\mathbf{x}\in\Omega} |g_{\mathbf{fl}}(\mathbf{x}) - g(\mathbf{x})| \le \varepsilon.$$

Algorithm 2 *Computation of an a priori error bound on the rounding errors ε_y when evaluating $g(\mathbf{t}) : \mathbb{R}^s \to \mathbb{R}$ for any $\mathbf{t}$ $(\mathbf{t} \in [a_1,b_1],[a_2,b_2],\cdots,[a_s,b_s],\ \mathbf{t} \in \mathbb{F})$ by floating-point operations, and an interval J_y satisfying*

$$\inf J_y \le g(\mathbf{t}) \le \sup J_y.$$

```
Step 1  Set an area $\mathbf{J} = ([a_1,b_1],[a_2,b_2],\cdots,[a_s,b_s])$.
Step 2  Put $\mathbf{x} = (\mathbf{J}, \max(\max_k |a_k|, \max_k |b_k|)\,\boldsymbol{u})$.
Step 3  Calculate $(J_y, \varepsilon_y) = g(\mathbf{x})$ using operator overloading.
Step 4  Output $\varepsilon_y$ and $J_y$.
```

2.1.2 Accurate Sum Algorithm PrecSum

Here we consider the summands of a vector p_i $(i = 1, 2, \cdots, n)$. An accurate summation algorithm **PrecSum**, developed by Rump, Ogita and Oishi [3], is useful to evaluate rounding errors in the summands. For the algorithm details, see [3]. The error bound for the result by **PrecSum** is given as follows [3]:

Theorem 1 (Rump et al. [3]) *Let p be a vector of n floating-point numbers, let $1 \leq K \in \mathbb{N}$, define $M = \lceil \log_2(n+2) \rceil$ and assume $\boldsymbol{u} \leq 1/1024$ and $2^{2M}\boldsymbol{u} \leq 1$. Assume $K \leq \left(4\sqrt{\boldsymbol{u}}\right)^{-1}$. Let* **res** *be the result of* **PrecSum** *applied to p. Abbreviate $s := \sum_{i=1}^{n} p_i$ and $S := \sum_{i=1}^{n} |p_i|$. Then $s \neq 0$ and*

$$|\mathbf{res} - s| \leq \boldsymbol{u}^K S + 2\boldsymbol{u}|s|. \tag{4}$$

2.2 Main Algorithm

We now consider the problem of constructing an algorithm for evaluating the rounding error E_r using Kashiwagi's method and accurate sum algorithm **PrecSum**. For this purpose, we introduce an algorithm calculating an approximate value of Eq. (1). Here, abbreviate

$$\begin{aligned} w_{i_1,\cdots,i_s} &:= w_{i_1} w_{i_2} \cdots w_{i_s}, \\ f_{i_1,\cdots,i_s} &:= f\left(x_{i_1}, x_{i_2}, \cdots, x_{i_s}\right), \end{aligned}$$

respectively.

Algorithm 3 *Computation of an approximation value of Eq. (1).*

Input	f, Ω
Output	**res**

Step 1 *Calculate ε_f and J_f, which are the upper bound of error occurred when the calculating $f_{i_1,\cdots,i_s}$ and the inclusion interval of $f_{i_1,\cdots,i_s}$ respectively, using* **Algorithm 2**. *We note that ε_f and J_f satisfy*

$$\max_{\Omega} |fl(f_{i_1,\cdots,i_s}) - f_{i_1,\cdots,i_s}| \leq \varepsilon_f, \tag{5}$$

$$\inf J_f \leq fl(f_{i_1,\cdots,i_s}) \leq \sup J_f. \tag{6}$$

Step 2 *Choose a minimum integer K satisfying*

$$\boldsymbol{u}^K \sup |J_f| \leq \varepsilon_f. \tag{7}$$

Step 3 *Calculate*

$$\mathbf{res} = \mathbf{PrecSum}\left(\sum_{i_1}\sum_{i_2}\cdots\sum_{i_s} p_{i_1,\cdots,i_s}, K\right),$$

with floating point arithmetic, where $p_{i_1,\cdots,i_s}$ denotes

$$p_{i_1,\cdots,i_s} := fl(w_{i_1,\cdots,i_s} f_{i_1,\cdots,i_s}).$$

Then, we consider the rounding errors occurred during the calculation of **Algorithm 3**. More concretely, we will consider the problem of evaluation of the upper bound of E_r;

$$E_r = \left| \sum_{i_1} \sum_{i_2} \cdots \sum_{i_s} w_{i_1,\cdots,i_s} f_{i_1,\cdots,i_s} - \mathbf{res} \right|.$$

To solve this problem, we present the following theorem:

Theorem 2 *Let a positive number $d = \prod_{k=1}^{s} (b_k - a_k)$. Let a constant ε_f, an interval J_f and a positive integer K be defined by Eq. (5), Eq. (6) and Eq. (7) respectively. If a constant $c_0 (\geq 1)$ satisfies*

$$\varepsilon_w := |\mathit{fl}(w_{i_1,\cdots,i_s}) - w_{i_1,\cdots,i_s}| \leq c_0 \boldsymbol{u} |w_{i_1,\cdots,i_s}|, \tag{8}$$

and a positive number c_1 and c_2 are selected as

$$\begin{aligned} c_1 &= (1+\boldsymbol{u})^3 \left(\varepsilon_f + \boldsymbol{u}\left(\varepsilon_f c_0 + \sup|J_f| \left(c_0 + (1 + c_0 \boldsymbol{u})^2\right)\right)\right), && (9) \\ c_2 &= (1 + c_0 \boldsymbol{u}) \left(1 + 2\boldsymbol{u}^{K+1} (1 + 3\boldsymbol{u})\right) \varepsilon_f, && (10) \end{aligned}$$

then

$$E_r \leq (c_1 + c_2)\, d + 2\boldsymbol{u} (1 + 3\boldsymbol{u}) |\mathbf{res}|.$$

holds.

Proof. Clearly it follows from the definition of **res** that

$$E_r \leq E_1 + E_2,$$

where E_1 and E_2 denote

$$\begin{aligned} E_1 &\leq \left| \sum_{i_1} \sum_{i_2} \cdots \sum_{i_s} w_{i_1,\cdots,i_s} f_{i_1,\cdots,i_s} - \sum_{i_1} \sum_{i_2} \cdots \sum_{i_s} p_{i_1,\cdots,i_s} \right|, && (11) \\ E_2 &\leq \left| \sum_{i_1} \sum_{i_2} \cdots \sum_{i_s} p_{i_1,\cdots,i_s} - \mathbf{res} \right|, \end{aligned}$$

respectively.

It follows clearly from Eq. (11),

$$E_1 \leq \left| \sum_{i_1} \sum_{i_2} \cdots \sum_{i_s} (w_{i_1,\cdots,i_s} f_{i_1,\cdots,i_s} - p_{i_1,\cdots,i_s}) \right|.$$

Now, from the error analysis for interval multiplication shown in subsection 2.1.1, it is seen that

$$\begin{aligned} |w_{i_1,\cdots,i_s} f_{i_1,\cdots,i_s} - p_{i_1,\cdots,i_s}| &\leq \mathrm{fl}_{+}\left(\varepsilon_f |J_w| + \varepsilon_w |J_f| + \mathbf{u} |J_w \cdot J_f|\right) \\ &\leq (1 + \mathbf{u})^3 \left(\varepsilon_f |J_w| + \varepsilon_w |J_f| + \mathbf{u} |J_w \cdot J_f|\right), \end{aligned}$$

where J_w denotes an interval satisfying

$$\inf J_w \leq \mathrm{fl}\left(w_{i_1,\cdots,i_s}\right) \leq \sup J_w. \tag{12}$$

Since ε_w, $|J_w|$ and $|J_w \cdot J_f|$ satisfy from Eq. (8) and Eq. (12),

$$\begin{aligned}\varepsilon_w &\leq c_0\mathbf{u}\,|w_{i_1,\cdots,i_s}|\,,\\ |J_w| &\leq (1+c_0\mathbf{u})\,|w_{i_1,\cdots,i_s}|\,,\\ |J_w\cdot J_f| &\leq (1+c_0\mathbf{u})^2\sup|J_f|\,|w_{i_1,\cdots,i_s}|\,,\end{aligned}$$

we have

$$|w_{i_1,\cdots,i_s} f_{i_1,\cdots,i_s} - p_{i_1,\cdots,i_s}| \leq c_1\,|w_{i_1,\cdots,i_s}|\,.$$

Here, assuming all of weights of an integration algorithm are positive, clearly it is seen that

$$\sum_{i_1}\sum_{i_2}\cdots\sum_{i_s}|w_{i_1,\cdots,i_s}| = \sum_{i_1} w_{i_1}\sum_{i_2} w_{i_2}\cdots\sum_{i_s} w_{i_s} \leq d,$$

then we have

$$E_1 \leq c_1 d.$$

Secondly, we consider the upper bound of E_2. From the error analysis of **PrecSum**, we have

$$E_2 \leq \mathbf{u}^K\sum_{i_1}\sum_{i_2}\cdots\sum_{i_s}|p_{i_1,\cdots,i_s}| + 2\mathbf{u}\left|\sum_{i_1}\sum_{i_2}\cdots\sum_{i_s}p_{i_1,\cdots,i_s}\right|.$$

Here since the upper bound of $|s|$ in Theorem 1 satisfies from Eq. (4),

$$|s| \leq (1+3\mathbf{u})\left(\mathbf{u}^K S + |\mathbf{res}|\right),$$

and the upper bound of S satisfies,

$$\begin{aligned}S = \mathbf{u}^K\sum_{i_1}\sum_{i_2}\cdots\sum_{i_s}|p_{i_1,\cdots,i_s}| &\leq \mathbf{u}^K\sup|J_f|\sum_{i_1}\sum_{i_2}\cdots\sum_{i_s}\sup|J_w|\\ &\leq \varepsilon_f\,(1+c_0\mathbf{u})\,d,\end{aligned}$$

then we have

$$E_2 \leq c_2 d + 2\mathbf{u}\,(1+3\mathbf{u})\,|\mathbf{res}|\,.$$

Combining these results proves the required inequality. □

Based on **Theorem 2**, we propose a concrete algorithm for evaluating E_r.

Algorithm 4 (Rounding Error Evaluation Algorithm) *Computation of an approximation value* **res** *of Eq. (1) and an error bound e on rounding errors incurred when calculating* **res** *satisfying $e = e_1 + e_2$.*

Input	f, Ω
Output	**res**, e_1, e_2

Step 1 *Calculate ε_f and J_f, which are the upper bound on the error incurred when calculating $f_{i_1,\cdots,i_s}$ and the inclusion interval of $f_{i_1,\cdots,i_s}$ respectively, using* **Algorithm 2**.

Step 2 *Choose a minimum integer K satisfying*

$$\mathbf{u}^K\sup|J_f| \leq \varepsilon_f.$$

Step 3 *Calculate c_1 and c_2 defined by Eq. (9) and Eq. (10) respectively.*

Step 4 *Calculate*

$$e_1 = (c_1 + c_2)d.$$

Step 5 *Calculate*

$$\mathbf{res} = \mathbf{PrecSum}\left(\sum_{i_1}\sum_{i_2}\cdots\sum_{i_s} p_{i_1,\cdots,i_s}, K\right),$$

by the floating point arithmetic.

Step 6 *Calculate*

$$e_2 = 2\boldsymbol{u}(1 + 3\boldsymbol{u})\,|\mathbf{res}|\,.$$

2.3 Fast Verified Automatic Integration Algorithm

Summarizing the above discussions, we propose the following fast verified numerical inclusion algorithm for the definite integral (1) :

Algorithm 5 (Fast Verified Automatic Integration Algorithm) *Consider the definite integral (1). Assume that a relative tolerance for the numerical integration error ε_{rel} is given. If $|I|$ is greater than a certain upper bound on the rounding errors, then this verified automatic integration algorithm outputs an approximate value I_c satisfying*

$$\left|\frac{I - I_c}{I}\right| \le \varepsilon_{rel}.$$

Otherwise, it outputs I_c satisfying

$$|I - I_c| \le C.$$

Here, C is a certain constant proportional to a certain bound on the rounding errors. To describe this function, we use MATLAB-like programming constructs.

```
function $[I_c, e, s] = \mathbf{AutoIntegralRel}(f, \Omega, \varepsilon_{rel})$
$E_r = \mathbf{RoundErrorStep4}(f, \Omega)$;
$\varepsilon = E_r$;
while 1,
  $I_c = \mathbf{AutoIntegral}(f, \Omega, \varepsilon)$;
  $e = fl_+(\varepsilon + E_r + \mathbf{RoundErrorStep6}(f, \Omega))$;
  If $|I_c| > e$,
    $\varepsilon = \varepsilon_{rel}(|I_c| - e)$;
    If $\varepsilon > E_r$,
      $I_c = \mathbf{AutoIntegral}(f, \Omega, \varepsilon)$;
      $s = 1$;
    else,
      $I_c = \mathbf{AutoIntegral}(f, \Omega, E_r)$;
      $s = 0$;
    end;
```

```
            $e = fl_+(\varepsilon + E_r + \textbf{RoundErrorStep6}(f, \Omega))$;
            *break*;
        *else*,
            $\varepsilon = \varepsilon \cdot \varepsilon_{rel}$
            *If* $E_r > \varepsilon$,
                $I_c = \textbf{AutoIntegral}(f, \Omega, E_r)$;
                $e = fl_+(\varepsilon + E_r + \textbf{RoundErrorStep6}(f, \Omega))$;
                $s = 0$;
                *break*;
            *end*;
        *end*;
    *end*;
```

Remarks:

(R1) **AutoIntegral** *function*

$$I = \textbf{AutoIntegral}\,(f, \Omega, \varepsilon)$$

is a function calculating an approximation value I of Eq. (1) over Ω by floating point arithmetic using the number of points n satisfying rigorously $|E_t\,(n)| \leq \varepsilon$, where E_t is defined by Eq. (3). For example, Petras's algorithm based on Gauss-Legendre quadrature [2] is useful algorithm as **AutoIntegral**. *See details in [2].*

(R2) **RoundErrorStep4** *and* **RoundErrorStep6**

$$\begin{aligned} e_1 &= \textbf{RoundErrorStep4}(f, \Omega) \\ e_2 &= \textbf{RoundErrorStep6}(f, \Omega) \end{aligned}$$

are functions calculating e_1 and e_2 in **Algorithm 4**, *respectively.*

(R3) If the function **AutoIntegralRel** *returns $s = 1$, then*

$$\left|\frac{I - I_c}{I}\right| \leq \varepsilon_{rel}$$

holds. If $s = 0$ is returned, the following holds:

$$|I - I_c| \leq e.$$

3 Numerical results

In this section, we present the numerical experiments. These experiments have been done by a computer having an Intel Core 2 Extreme 3.0 GHz CPU with 8G Byte Memory. We use the C++ language (GCC 4.1.2 with CRlibm 1.0 beta [5]) under the Fedora Core 8 Linux operating system. We use only 1-core of the CPU. Gauss-Legendre quadrature is employed as one-dimensional quadrature.

We compare the computational costs of the following two algorithms:

(A) proposed algorithm (**Algorithm 4**),

(B) an algorithm calculating the upper bound on

$$\hat{I} - \mathrm{fl}\left(\hat{I}\right)$$

by interval arithmetic.

Both algorithms (A) and (B) give the upper bound of the rounding errors of the approximation value **res** of Eq. (1). The differences between Algorithms (A) and (B) are as follows: **res** of Algorithm (B) is

$$\mathbf{res} = \mathrm{fl}\left(\hat{I}\right).$$

On the other hand, **res** of Algorithm (A) is

$$\mathbf{res} = \mathbf{PrecSum}\left(\sum_{i_1}\sum_{i_2}\cdots\sum_{i_s}\mathrm{fl}\left(w_{i_1,\cdots,i_s} f_{i_1,\cdots,i_s}\right), K\right).$$

Furthermore, Algorithm (B) uses interval arithmetic to evaluate the rounding errors. Algorithm (A) is based on Kashiwagi's method and the error analysis of **PrecSum**.

Examples

$$\begin{aligned} I_1 &= \int_{-1}^{1} \exp\left(\frac{\pi}{2}\exp(x)\right) dx, \\ I_2 &= \int_{-1}^{1}\int_{-1}^{1} \exp\left(\frac{\pi}{2}\exp(xy)\right) dxdy \end{aligned}$$

Figure 1 shows a comparison of the execution time of (A) and (B) for I_1 and I_2, respectively. From these figures, it is seen that the computational time for (A) is 3-5 times faster than that for (B), provided that the same relative tolerance is used.

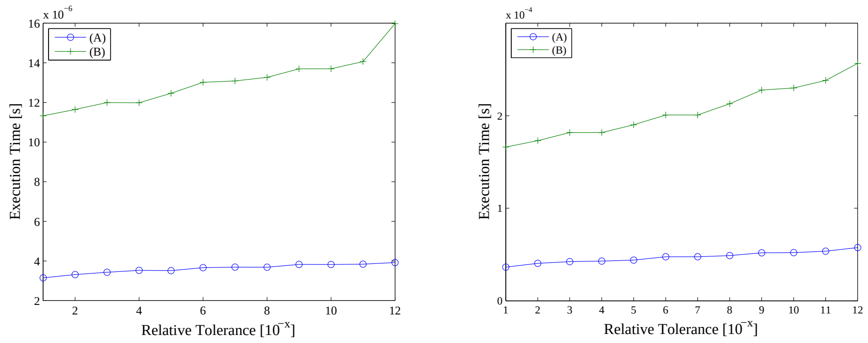

Figure 1: The execution time of I_1 (left) and that of I_2 (right).

References

[1] P. J. Davis and P. Rabinowitz, *Methods of Numerical Integration,* Academic Press, New York, 1975.

[2] K. Petras, Self-validating integration and approximation of piecewise analytic functions, *J. Comput. Appl. Math.*, vol. 145, pp. 345–359, 2002.

[3] S.M. Rump, T. Ogita, S. Oishi, Fast high precision summation, (submitted).

[4] T. Ogita, S. M. Rump, S. Oishi, Accurate sum and dot product, *SIAM J. Sci. Comput.*, vol. 26, pp. 1955–1988, 2005.

[5] Correctly Rounded Mathematical Library, `http://lipforge.ens-lyon.fr/www/crlibm/`