On Iterated Numerical Integration

Shujun Li, Elise de Doncker, and Karlis Kaugars

Computer Science,
Western Michigan University
{sli, elise, kkaugars}@cs.wmich.edu
http://www.cs.wmich.edu/~parint

Abstract. We revisit the iterated numerical integration method and show that it is extremely efficient in solving certain classes of problems. A multidimensional integral can be approximated by a combination of lower-dimensional or one-dimensional adaptive methods iteratively. When an integrand contains sharp ridges which are not parallel with any axis, iterated methods often outperform adaptive cubature methods in low dimensions. We use examples to support our analysis.

1 Introduction

We will call an integration method *iterated*, if lower-dimensional methods are used for the integration in different coordinate directions [18, 11]. We use an adaptive method from QuadPack [14] to compute the one-dimensional integrals in an iterated method. The work of [5, 6] shows that the iterated method is much more efficient in integrating certain Feynman loop integrals of particle physics. As an example, one of the three-dimensional integrands has a singular behavior located within three narrow adjacent layers of the integration domain. The function has large positive values in the outer two and is negative in the middle layer.

Further study reveals that limitations of the machine precision prevent us from achieving better results for some problems. When a specific integrand parameter becomes small, increasing the function evaluation limit does not lead to a better result. Even when this parameter is not very small, setting the function evaluation limit to a large value results in unnecessary function evaluations and possibly a less accurate result.

Iterated methods deserve a thorough investigation. A prototype implementation is underway for ParInt [16]. After modifying the one-dimensional methods in ParInt to handle discontinuities, the integrals of the Dice functions (see Section 4 below) can be approximated to an accuracy that is limited only by the machine precision.

2 Iterated Method

We will use one-dimensional integration methods iteratively to compute an n-dimensional ($n \geq 2$) integral numerically.

The integral of an n-dimensional scalar function $f(x_1, x_2, ..., x_n)$ over a hyper-rectangular region $\mathcal{D}$ in $\mathcal{R}^n$ is

$$I = \int_{\mathcal{D}} f(x_1, x_2, ..., x_n)\ dx_1 dx_2 ... dx_n, \tag{1}$$

which is the same as

$$I = \int_{x_1^a}^{x_1^b} dx_1 \int_{\mathcal{D}'} f(x_1, x_2, ..., x_n)\ dx_2 ... dx_n. \tag{2}$$

Let

$$F(x_1) = \int_{\mathcal{D}'} f(x_1, x_2, ..., x_n)\ dx_2 ... dx_n, \tag{3}$$

then integral (2) becomes

$$I = \int_{x_1^a}^{x_1^b} F(x_1)\ dx_1, \tag{4}$$

which is a one-dimensional integral. We then start from formula (4), and repeat the process in the remaining coordinate directions.

We found that the iterated method outperforms other methods significantly for a class of problems with steep, narrow ridges for 2D functions, or similar behavior in higher dimensions.

3 Implementation

We can use a one-dimensional integration method to compute an approximation for integral (4). The code in this method will evaluate $F(x_1)$ for a given value of x_1 by calling an integration method. The calls are done recursively in the following pseudo-code. The pseudo-code below uses a C-like syntax.

Iterated Method:

```
n <-- dimension
lower <-- array for lower bounds
upper <-- array for upper bounds
xx <-- temporary array of size n for a point

main() {
  a <-- lower[1]
  b <-- upper[1]
  i <-- 1
  integrate(foo, i, a, b, result, error)
  print(error, result)
}
```

```
foo(n, i, x_i, fcn_value) {
  if i = n then
    integrand(n, x_i, fcn_value)
  else
    xx[i] = x_i
    a <-- lower[i+1]
    b <-- upper[i+1]
    i <-- i + 1
    integrate(foo, i, a, b, result, error)
    fcn_value <-- result;
}

integrand(n, x_i, fcn_value) {
  xx[n] <-- x_i
  fcn_value <-- f(xx)
}
```

The actual code differs from the pseudo-code significantly. It is also more complex. In the pseudo-code *n*, *lower*, *upper* and *xx* are global variables. *foo* is part of the package. The driver function *main* and the *integrand* function are given by the user. The end users do not need to know how the iterated method is implemented. *integrate* is a one-dimensional adaptive method. Any combination of the directions will be implemented in a future release of PARINT [16]. For example, a one-dimensional method in the x direction can call a two-dimensional method in the y and z directions.

Iterated integration methods were implemented in FORTRAN for the computations of [5, 6]. Other implementations include D01DAF in NAG (a FORTRAN subroutine for two-dimensional iterated numerical integration) [18]. According to its documentation, D01DAF is not well-suited for non-smooth integrands. Two-dimensional iterated numerical integration is also explained in [11] and [14].

For a given total error tolerance, selecting the error tolerances of the inner integrals is non-trivial. Currently we use the same relative error tolerance for all levels. The contribution of the inner and outer integration errors and a heuristic estimation of the total error are outlined in [11], where it is suggested that, for a two-dimensional iterated method, the inner integral be computed about a factor of ten more accurately than the outer. If the total absolute error tolerance is ε_a, then $\varepsilon_a^O = 0.9\varepsilon_a$ and $\varepsilon_{ai}^I = \frac{\varepsilon_a}{10(x_2^b - x_2^a)}$. The total estimate error is given by $error_O + (x_2^b - x_2^a)\max_{x_i} error_I(x_i)$, where $error_O$ is the estimated error of the outer integral, and $error_I(x_i)$ is that of the inner for a given value x_i of x..

Fritsch, Kahaner and Lyness [7] study the error tolerance assignment in a two-dimensional iterated method. The total absolute error tolerance, $\varepsilon_a^T = \varepsilon_a^O + \varepsilon_a^I$, where ε_a^O and ε_a^I are the absolute error tolerances for the outer and the inner integrations, respectively. For m-panel ($(m+1)$-point) closed Newton-Cotes rules, an optimal ratio is $\frac{\varepsilon_a^I}{\varepsilon_a^O} = \frac{m^O+2}{m^I+2}$. The authors discuss the assignment of ε_a^I for the inner integrations. One way is to use a constant error tolerance ε_{ai}^I for all points x_i in the x direction. Another way is to have $\varepsilon_{ai}^I|W_i| = constant$, where W_i is the weight assigned at the first appearance of $F(x_i)$. The former is intended for situations where the function has no peaks or untoward behavior, while the latter assignment will apply if the function is well behaved over most of its domain but has peaks or oscillations in a small portion of the domain.

4 Performance Analysis

Let us address the performance of the iterated method using two sample integrals from [17], which we refer to as DICE1 and DICE2 below.

$$\text{DICE1} \int_0^1 dx \int_0^1 dy \frac{2\varepsilon y}{(x+y-1)^2+\varepsilon^2}$$

$$\text{DICE2} \int_{-1}^1 dx \int_{-1}^1 dy \frac{\varepsilon y^2 \theta(1-x^2-y^2)}{(x^2+y^2-b^2)^2+\varepsilon^2}$$

Here $\theta(t) = 1$ for $t \geq 0$, and 0 otherwise. The DICE1 integrand has a ridge of height $\frac{2y}{\varepsilon}$ along the diagonal $y = 1 - x$. The DICE2 function has a ridge along the circle of radius b centered at the origin, and a discontinuity at the unit circle.

Two-dimensional adaptive cubature methods are not very effective for computing these integrals, which mimic the behavior of certain integration problems arising in high-energy physics computations. Results are given in [4] for $b = 0.8$, $\varepsilon = 10^{-1}, 10^{-2}, \ldots, 10^{-6}$, a relative error tolerance of 10^{-5} and a function evaluation limit of 250 million.

In order to show the behavior of the DICE2 integrand aggregated in the y direction, we graph the inner integral $F(x)$ in Figure 1 *(Left)* and *(Right)* for $\varepsilon = 10^{-1}$ and 10^{-4}, respectively. It emerges that these are rather smooth functions of x. Consequently, the integral in x can be carried out easily and only a moderate number of subdivisions in the x direction is carried out for the iterated integration. This is also true in the y-direction, if the x direction is aggregated first.

Figure 2 (*Left*) and (*Right*) displays visualizations of the function values evaluated by the adaptive cubature and by the iterated method, respectively, for DICE1. We use the (parallel) cubature methods in PARINT, based on the integration rules of Genz and Malik [9, 1].

For the adaptive cubature, subdivision is performed in a 2D surface. For the iterated method, subdivision is mainly done along the y axis. This behavior is also confirmed by the views of $F(x)$ for DICE2 in Figure 1 *(Left)* and *(Right).* In spite of the ridges on the two-dimensional domains, the functions aggregated to one dimension depict a relatively smooth behavior. Thus the outer integral is easy to calculate.

Let us furthermore examine the function evaluation count. As a rough estimate, the number of function evaluations of the two-dimensional adaptive cubature method is about the square of that of the iterated method for the problems under consideration.

The adaptive cubature method chooses a direction to bisect a region so that the subsequent computation becomes easier on the subregions. If the ridges are

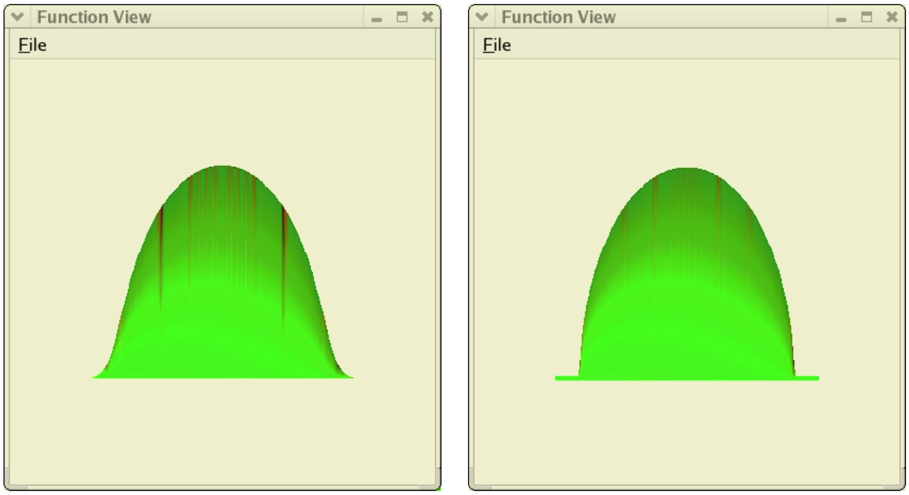

Fig. 1. $F(x)$ of the DICE2 function, where *Left:* $\varepsilon = 0.1$, and *Right:* $\varepsilon = 0.0001$

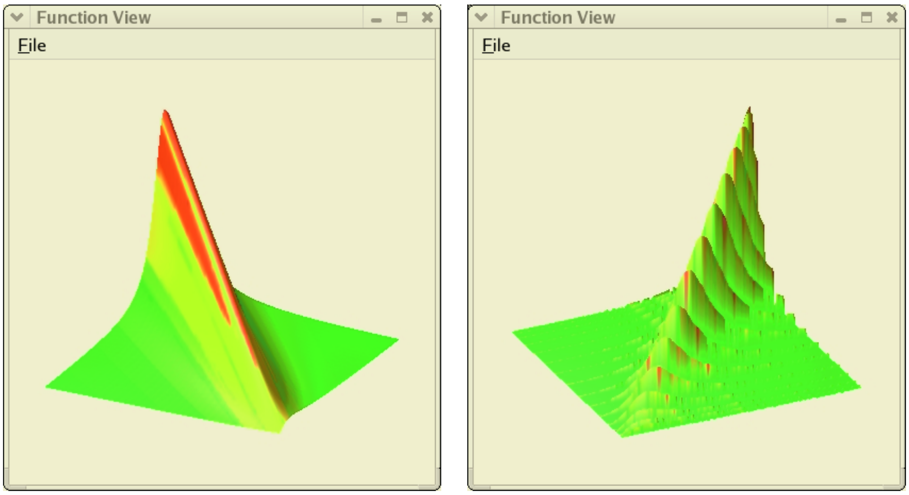

Fig. 2. *Left:* Adaptive cubature method for DICE1, where $b = 0.8$ and $\varepsilon = 0.1$; *Right:* Iterated method for DICE1. This image has been rotated for a better view

parallel to one of the axes, the advantage of the iterated method disappears. We removed y in the numerator of the DICE1 integrand and rotated it with respect to the point $(0.5, 0.5)$. We performed numerical integration for a variety of angles between the ridge and the x axis, with $\varepsilon = 0.01$ and a relative error tolerance of 10^{-10}. Figure 3 illustrates the relationship between the ridge angle and the number of function evaluations performed. When the ridge is parallel to

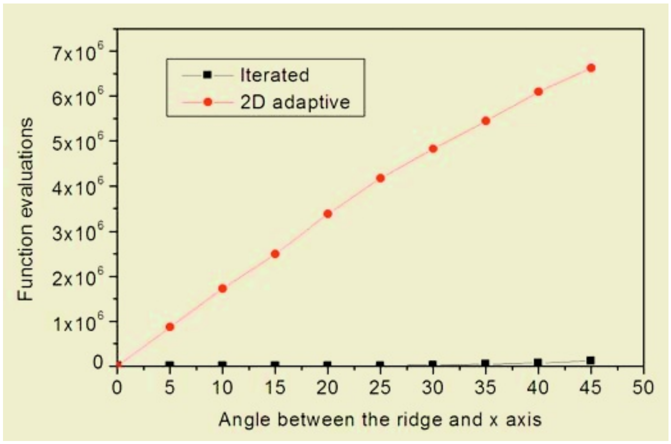

Fig. 3. The relationship between number of function evaluations and ridge orientation

the x axis, the number of function evaluations is 5355 for the adaptive cubature method, which is close to that of the iterated method (6,975). When the ridge is along the diagonal, 6,621,069 evaluations are needed, which is almost the square of the former.

To generate the function visualizations we used AdaptView [12], which utilizes adaptive numerical integration to visualize the integrand function.

Iterated methods are able to produce end-results with a very small estimated error for the problems discussed above. In order to avoid early termination,

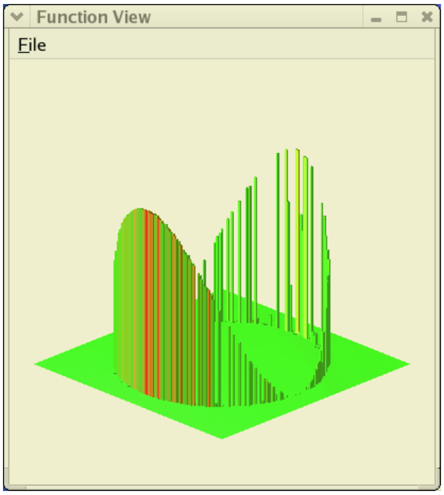

Fig. 4. Visualization of the DICE2 integrand, where $b = 0.8$, $\varepsilon = 10^{-6}$ and the absolute error tolerance is 10^{-6}

especially in cases where the ridges are extremely narrow, the error tolerance should be set small enough, so that the singularities are "discovered" before the computation is trapped in unstable areas or the error estimate has met the requirement. Figure 4 illustrates this effect. For a fixed value of x, the (inner) one-dimensional integration is done in the y variable. When a peak is not sampled by the integration rule, some regions may have an under-estimated error, and will not be evaluated again.

5 Conclusions and Future Work

We demonstrated that the iterated method is extremely fast for an important class of problems with *ridged* integrand behavior. If the integrand function has only a limited number of singular points (i.e., a *localized* singular behavior), or if the singular ridge is parallel to one of the axes, adaptive method are usually effective. We plan to investigate loop integrals of fairly low dimensions (less than 6, for example) [15, 6, 5, 2, 8, 3] using iterated methods. Note that some methods involving multi-dimensional integrals reduce to the computation of many two- and three-dimensional integrals as in [3].

It is feasible to compute many two- and three-dimensional integrals numerically to produce a reasonably small error, because a thousand of function evaluations in one direction can often achieve decent accuracy. Even if the total evaluation count is a billion for, say, a three-dimensional problem, that may not be a real obstacle for even a desktop computer today. We do need to make use of reasonable summation methods [10].

Iterated integration yields a good candidate for parallel/distributed integration methods in view of the large granularity of the inner integral evaluations. We performed preliminary tests on distributed computations with with a Web service based integration system, PI service [13].

Acknowledgment

This work is supported in part by Western Michigan University, and by the National Science Foundation under grants ACI-0000442, ACI-0203776 and EIA-0130857.

References

1. Berntsen, J., Espelid, T. O., and Genz, A. Algorithm 698: DCUHRE-an adaptive multidimensional integration routine for a vector of integrals. *ACM Trans. Math. Softw. 17* (1991), 452–456. Available from http://www.sci.wsu.edu/math/faculty/genz/homepage.
2. Binoth, T., and Heinrich, G. An automized algorithm to compute infrared divergent multi-loop integrals. hep-ph/0004013 v2.
3. Binoth, T., Heinrich, G., and Kauer, N. A numerical evaluation of the scalar hexagon integral in the physical region. hep-ph/0210023.

4. de Doncker, E., Kaugars, K., Cucos, L., and Zanny, R. Current status of the ParInt package for parallel multivariate integration. In *Proc. of Computational Particle Physics Symposium (CPP 2001)* (2001), pp. 110–119.
5. de Doncker, E., Shimizu, Y., Fujimoto, J., and Yuasa, F. Computation of loop integrals using extrapolation. *Computer Physics Communications 159* (2004), 145–156.
6. de Doncker, E., Shimizu, Y., Fujimoto, J., Yuasa, F., Cucos, L., and Van Voorst, J. Loop integration results using numerical extrapolation for a non-scalar integral. *Nuclear Instruments and Methods in Physics Research Section A 539* (2004), 269–273. hep-ph/0405098.
7. Fritsch, F. N., Kahaner, D. K., and Lyness, J. N. Double integration using one-dimensional adaptive quadrature routines: A software interface problem. *ACM Trans. Math. Softw. 7*, 1 (1981), 46–75.
8. Fujimoto, J., Shimizu, Y., Kato, K., and Oyanagi, Y. Numerical approach to one-loop integrals. *Progress of Theoretical Physics 87*, 5 (1992), 1233–1247.
9. Genz, A. MVNDST Mult. Normal Dist. software, 1998. Available from web page at http: //www.sci.wsu.edu/math/faculty/genz/homepage.
10. Kahan, W. Further remarks on reducing truncation errors. *Comm. ACM 8* (1965), 40.
11. Kahaner, D., Moler, C., and Nash, S. *Numerical Methods and Software.* Prentice Hall, 1989.
12. Li, S., Kaugars, K., and de Doncker, E. Grid-based distributed function visualization.
13. Li, S., Kaugars, K., and de Doncker, E. Massive scale distributed integration using Web service. In *The Hawaii International Conference on Computer Sciences* (2003). CDROM Proceedings.
14. Piessens, R., de Doncker, E., Überhuber, C. W., and Kahaner, D. K. *QUADPACK, A Subroutine Package for Automatic Integration.* Springer Series in Computational Mathematics. Springer-Verlag, 1983.
15. Son, D. H. *Feynman Loop Integrals and their automatic Computer-aided Evaluation.* PhD dissertation, Johannes Gutenberg-Universität Mainz, June 2003.
16. ParInt Group. http://www.cs.wmich.edu/parint, ParInt web site.
17. Tobimatsu, K., and Kawabata, S. Multi-dimensional integration routine DICE. Tech. Rep. 85, Kogakuin University, 1998.
18. Überhuber, C. *Numerical Computation 2 - Methods, Software, and Analysis.* Springer-Verlag, 1997.