ADAPTIVE QUADRATURE METHOD TO APPROXIMATE DOUBLE INTEGRALS OVER NON-RECTANGULAR REGIONS

Pham Thi Thu Hang, Dinh Van Tiep*
University of Technology - TNU

ABSTRACT

Recently, in an early publication [1], the author presented an algorithm to approximate double integral over a rectangle basing on the adaptive quadrature method. This method has the upper hand comparing with many other approaches due to its low cost and high efficiency. In that paper, even the algorithm was constructed strictly for approximating double integrals over rectangles only, it can be also extended to treat the case of non-rectangular regions. Nevertheless, for non-rectangular regions, it is not somewhat practical to program such an algorithm because of the consumption a large amount of RAM to store data of values for functions taking on the boundary of the region. The storage is performed at many steps in each repeating loop. This shortcoming often makes the program implement awkwardly. This paper aims to revise the aforementioned algorithm to be more efficient.

Key words: *numerical integration, approximate double integral, adaptive method, adaptive quadrature, non-rectangle region.*

INTRODUCTION*

Similar to an algorithm for the adaptive quadrature of double integrals over a rectangular region, presented in [1], one of double integral over general regions, or more exactly, over non-rectangular regions, could be constructed. Here, we confine the consideration to such regions bounded by graphs of functions,

$\Omega = \{(x,y)|a \le x \le b, c(x) \le y \le d(x)\}$,

where c, d are functions defined on $[a,b]$. Without losing of generality, we can assume that $c(x) \le d(x), \forall x \in [a,b]$, keeping in mind that the algorithm we are going to discuss here still works very well if the role of x and y are switched. We are going to approximate the double integral

$$I = \iint_{\Omega} f(x,y)\, dA$$

of a continuous function f defining on Ω. To apply the adaptive quadrature method which had been well developed for one variable, we rewrite the double integral in the form of an iterated integral

$$I = \int_a^b \int_{c(x)}^{d(x)} f(x,y) dy dx. \qquad (1)$$

First, fixing x in $[a,b]$, we use the method on the interval $[c(x), d(x)]$ of the vertical axis, and then on the interval $[a,b]$ of the horizontal axis. For a given tolerance $\varepsilon > 0$, and a given level of subdivision N, after a finite number of repetitions of this procedure, we can decide whether the obtained approximation meets the requirement or not. (Failure to meet the requirement is signified by exceeding the given level of the needed level of subdivision.)

CONSTRUCTION OF THE METHOD

In this section, we are going to formulate the theoretical basis for the algorithm. Firstly, set

$$2p = h = \frac{b-a}{2}, \text{and functions k, q with}$$

$$2q(x) = k(x) = \frac{d(x)-c(x)}{2}, \forall x \in [a,b].$$

Apply Simpson's and Composite Simpson's Rule with number of nodes $n = 4$, successively, we get

$$I = \frac{h}{9}\sum_{l=0}^{2} n_l \left[\sum_{j=0}^{2} n_j k(x_j) f(x_j, y_{lj})\right] - \widehat{E_1}$$
$$=: \widehat{S_1} - \widehat{E_1},$$

* *Tel: 0888 272060, Email: tiepdinhvan@gmail.com*

$$I = \frac{p}{9}\sum_{l=0}^{4} m_l \left[\sum_{j=0}^{4} m_j q(x_j) f(x_j, y_{lj})\right] - \widehat{E_2} =: \widehat{S_2} - \widehat{E_2},$$

where $n_0 = n_2 = 1, n_1 = 4$, and $m_0 = m_4 = 1, m_2 = 2, m_3 = m_4 = 4$. The error estimates are $\widehat{E_1} = \frac{b-a}{90}\left[h^4 k(\bar{\eta}) f(\bar{\eta}, \bar{\xi}) + k^5(\bar{\mu})\frac{\partial^4 f}{\partial y^4}(\bar{\mu}, \gamma_{\bar{\mu}})\right]$, and $\widehat{E_2} = \frac{1}{16}\frac{b-a}{90}\left[h^4 k(\hat{\eta})\frac{\partial^4 f}{\partial x^4}(\hat{\eta}, \hat{\xi}) + k^4(\hat{\mu})\frac{\partial^4 f}{\partial y^4}(\hat{\mu}, \gamma_{\hat{\mu}})\right]$, for some $(\bar{\eta}, \bar{\xi}), \left(\bar{\mu}, \gamma_{\bar{\mu}}\right)$, and $(\hat{\eta}, \hat{\xi}), \left(\hat{\mu}, \gamma_{\hat{\mu}}\right) \in \Omega$. By assuming that $(\bar{\eta}, \bar{\xi}) \approx (\hat{\eta}, \hat{\xi}), \left(\bar{\mu}, \gamma_{\bar{\mu}}\right) \approx \left(\hat{\mu}, \gamma_{\hat{\mu}}\right)$, we get $\widehat{E_2} \approx \frac{1}{16}\widehat{E_1}$, and $|\widehat{S_2} - I| \approx \frac{1}{15}|\widehat{S_2} - \widehat{S_1}|$. Thus, $\widehat{S_2}$ estimates I about 15 times better than it agrees with $\widehat{S_1}$. So, if $\left|\widehat{S_1} - \widehat{S_2}\right| < 15\varepsilon$, then that $\left|I - \widehat{S_2}\right| = \left|\iint_\Omega f(x,y)dxdy - \widehat{S_2}\right| < \varepsilon$ might be accepted confidently. We can trust that $\widehat{S_2}$ is good enough to approximate I within the given tolerance ε. Conversely, if $\left|\widehat{S_1} - \widehat{S_2}\right| \geq 15\varepsilon$, we subdivide Ω into four smaller pieces by mesh points

$$a = x_0 < x_1 = \frac{a+b}{2} < x_2 = b,$$

$$c(x) = y_0(x) < y_1(x) = \frac{c(x)+d(x)}{2} < y_2(x) = d(x).$$

Now, on each piece, we required that the difference between the corresponding $\widehat{S_1}$, and $\widehat{S_2}$ does not exceed a quarter of 15ε. If this condition is fulfilled, we use this value of $\widehat{S_2}$ as an approximation of the integral over the considered piece. On the other hand, if this condition fails to meet, we keep going on subdividing the considered piece into four smaller pieces and carry on performing the above procedure until reaching the target. The required tolerance is now reduced by a factor of four from the previous level. For our algorithm, we repeat the procedure a finite number of times which guarantees that the level of subdivision does not exceed N. If this demand is undertaken and the procedure completes, it is successful. By summing up such approximations all over considered pieces on which the procedure succeeds, we obtain a significant approximation for I. In the case where the level of subdivision exceeds N, we conclude that the whole procedure fails to meet the demand. To expect the method can be reapplied successfully on Ω, we need to reduce the tolerance ε and/or increase N.

ALGORITHM

The above algorithm could be expressed with the use of a pseudo-code, as follows

INPUT region Ω with $a, b, c(x), d(x)$, tolerance $\epsilon > 0$, limit N to number of levels.

OUTPUT approximation AP of I, or message that N is exceeded (the procedure fails).

Step 1 (*Initiate the procedure*)

$AP := 0; i := 0, L_i := 1; \epsilon_i := 15\epsilon$; *(Here, L_i indicates the recent level of the subdivision.)*

$n_0 := 1; n_1 := 4; n_2 := 1$; *(The coefficients in Simpson's Rule.)*

$m_0 := 1; m_1 := 4; m_2 := 2; m_3 := 4; m_4 := 1$. *(The coefficients in Composite Simpson's Rule.)*

$r_i := 1; s_i := 2^N + 1$; (The starting and ending indices of mesh points traced on $[a, b]$.)

$R_i := 0$; *(Initial region Ω is numbered by 0.)*

For j from 1 to $2^N + 1$ do

$$X_j := a + (j-1)\frac{b-a}{2^N};$$

$$F_i(j) := c(X_j); G_i(j) := d(X_j);$$

(See the NOTE below for F_i, G_i.)

End do.

Step 2 While $i > 0$ do Steps 3-5.

Step 3

$h_i := 0.5(X_{s_i} - X_{r_i}); t_i := 0.5(s_i - r_i);$

IF $(L_i \geq N)$ or $(ht_i$ is odd) **THEN**

OUTPUT ("LEVEL IS EXCEEDED.");

STOP.

ELSE

For $j = 1$ to 3 do

$$x_j := X_{r_i} + (j-1)h_i;$$

$k_{ij} \coloneqq 0.5[G_i(r_i + (j-1)t_i) - F_i(r_i + (j-1)t_i)];$

For $l = 1$ to 3 do

$y_{lj} \coloneqq F_i(r_i + (j-1)t_i) + (l-1)k_{ij}$

End do;

End do; *(Set up data of mesh points for Simpson's Rule.)*

For $j = 1$ to 5 do

$z_j \coloneqq X_{r_i} + 0.5(j-1)h_i;$

$p_{ij} \coloneqq 0.25[G_i(r_i + 0.5(j-1)t_i) - F_i(r_i + 0.5(j-1)t_i)];$

For $l = 1$ to 5 do

$q_{lj} \coloneqq F_i(r_i + 0.5(j-1)t_i) + (l-1)p_{ij};$

End do;

End do; *(Set up data of mesh points for Composite Simpson's Rule.)*

$S_1 \coloneqq 0; S_2 \coloneqq 0;$ *(Initiate the values for Simpson's and Composite Simpson's Rule, respectively.)*

For $l = 1$ to 3 do

For $j = 1$ to 3 do

$$S_1 \coloneqq S_1 + \frac{h_i k_{ij}}{9} n_l n_j f(x_l, y_{lj});$$

End do; *(Simpson's Rule.)*

End do;

For $l = 1$ to 5 do

For $j = 1$ to 5 do

$$S_2 \coloneqq S_2 + \frac{h_i p_{ij}}{18} m_l m_j f(z_j, q_{lj});$$

End do; *(Composite Simpson's Rule.)*

End do;

$u_1 \coloneqq r_i; u_2 \coloneqq s_i; u_3 \coloneqq t_i; u_4 \coloneqq F_i;$

$u_5 \coloneqq G_i; u_6 \coloneqq \epsilon_i; u_7 \coloneqq L_i.$ *(Save data.)*

$level_i \coloneqq u_7;$ *(The variable level stores the sequence of operating levels in whole procedure.)*

If $size(level) > 1$ then

$count \coloneqq 0; j \coloneqq 1;$ *(Initiate counting pieces of the same level with the current level.)*

while $j \le size(level) - 1$ do

if $level_j = level_{size(level)}$ then

$count \coloneqq count + 1;$

end if; $j \coloneqq j + 1;$

end do; $R_i \coloneqq 4 - \mathbf{mod}(count, 4);$

End if.

Step 4 $i := i - 1$. *(Delete the level.)*

Step 5 **If** $|S_2 - S_1| < u_6$ **then**

$AP \coloneqq AP + S_2; status_i \coloneqq$ "PASS".

(Variable status is used to store the sequence of results revealed when performing the procedure on each piece of the region.)

Else *(Add one level.)* $status_i \coloneqq$ "FAIL";

$i \coloneqq i + 1;$ *(Data for region R_1, cf. Figure 1.)*

$r_i \coloneqq u_1; s_i \coloneqq u_1 + u_3; F_i \coloneqq u_4;$

$G_i \coloneqq 0.5(u_4 + u_5); \epsilon_i \coloneqq \frac{u_6}{4}; L_i \coloneqq u_7 + 1.$

$i \coloneqq i + 1;$ *(Data for region R_2.)*

$r_i \coloneqq u_1; s_i \coloneqq u_1 + u_3; F_i \coloneqq G_{i-1}; G_i \coloneqq u_5;$

$\epsilon_i \coloneqq \epsilon_{i-1}; L_i \coloneqq L_{i-1};$

$i \coloneqq i + 1;$ *(Data for region R_3.)*

$r_i \coloneqq u_1 + u_3; s_i \coloneqq u_2; F_i \coloneqq u_4;$

$G_i \coloneqq 0.5(u_4 + u_5); \epsilon_i \coloneqq \epsilon_{i-1}; L_i \coloneqq L_{i-1};$

$i \coloneqq i + 1;$ *(Data for region R_4.)*

$r_i \coloneqq u_1 + u_3; s_i \coloneqq u_2; F_i \coloneqq G_{i-1}; G_i \coloneqq u_5;$

$\epsilon_i \coloneqq \epsilon_{i-1}; L_i \coloneqq L_{i-1}.$

End if.

END IF.

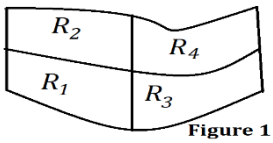

Figure 1

Step 6

OUTPUT AP, and $R_i, level_i, status_i$.

STOP.

NOTE: F_i, G_i are used in place of the functions c, d in order to reduce essentially the memory for storing data of values for functions at the corresponding step i because the data of F_i, G_i are finite vectors instead of continuum ones for continuous functions. This technique improves dramatically the efficiency of the method comparing to one mentioned in [1].

EXAMPLES

Basing on the pseudo-code, one can easily edit a code with a help of a plenty of powerful software. The examples presented in this paper are ones experimented with code designed for implementing on Matlab R2017a and Maple 2016.

Example 1 Approximate the double integral

$$I = \iint\limits_{\Omega=\{(x,y)|-1\le y\le 3, 1\le x\le 3\}} \frac{2x}{x^2+y+1}dA,$$

with the tolerance $\epsilon = 0.0004$, and the number of limit level $N = 5$.

The procedure applied for this example in fact is one presented in [1], where functions $c(x) \equiv -1, d(x) \equiv 3$ are constants. The algorithm in this paper is more general than one presented in [1]. The obtained approximation is $AP = 5.522168791$. The exact value of the integral is $I = 13\ln 13 - 18\ln 3 - 5\ln 5 \approx 5.52213$.

The procedure is described by the sequences $R_i, level_i$ and $status_i$ released from the algorithm. They are shown in Table 1.

Table 1.

$level_i$	R_i	$status_i$	$level_i$	R_i	$status_i$
1	0	FAIL	3	4	FAIL
2	4	PASS	3	3	PASS
2	3	PASS	3	2	PASS
2	2	PASS	3	1	PASS
2	1	FAIL			

We are going to interpret data in Table 1. Initiated with the level $level_i = 1$, and the original region Ω indicated by $R_i = 0$, the procedure fails to meet the requirement. This conclusion is identified by the value $status_i = FAIL$. Then, next step, Ω is partitioned into 4 smaller rectangles, this implementation is indicated by the increased value $level_i$ up to 2. The procedure is reapplied on one of such rectangles, which is numbered by $R_i = 4$, the results is represented by the value $status_i = PASS$. Next, the procedure is reapplied again on another subrectangle numbered by $R_i = 3$ of the same level $level_i = 2$. This produces an acceptable result indicated by the value $status_i = PASS$. And so on, the last performance is undertaken at $level_i = 3$, on subrectangle numbered 1 of the subrectangle 1 obtained at $level_i = 2$. The ending $level_i = 3$ does not exceed the limit level N tells us that the procedure completes successfully. The process and the graph of the function f are illustrated in Figure 2.

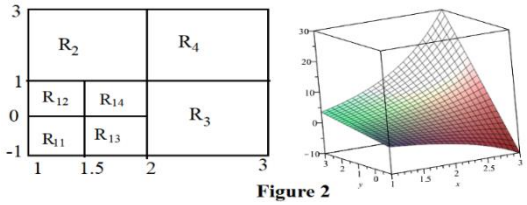

Figure 2

Example 2 Approximate the integral

$$I = \iint\limits_{\Omega=\{(x,y)|x^2\le y\le x, 0\le x\le 1\}} (x^2+2xy)dA,$$

with $\epsilon = 10^{-5}, N = 5$.

The obtained result $AP = 0.1333283695$ is significant to estimate I to within 10^{-5}. Indeed, it is very simple to see that $I = \frac{2}{15} = 0.1(3)$. Other information about the performance of the procedure applied on Ω is contained in sequences $R_i, level_i$, and $status_i$ which are released along with AP. Table 2 tells us the complete process. The graph of the function f is shown in Figure 3.

Table 2.

$level_i$	R_i	$status_i$	$level_i$	R_i	$status_i$
1	0	FAIL	3	2	PASS
2	4	FAIL	3	1	FAIL
3	4	PASS	4	4	PASS
3	3	PASS	4	3	PASS
3	2	PASS	4	2	PASS
3	1	PASS	4	1	PASS
2	3	FAIL	2	2	FAIL
3	4	FAIL	3	4	PASS
4	4	PASS	3	3	PASS
4	3	PASS	3	2	PASS
4	2	PASS	3	1	PASS
4	1	PASS	2	1	FAIL
3	3	FAIL	3	4	PASS
4	4	PASS	3	3	PASS
4	3	PASS	3	2	PASS
4	2	PASS	3	1	PASS
4	1	PASS			

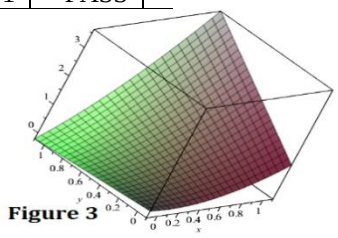

Figure 3

SUMMARY

Storing only data for a vector rather than data of values for a function reduces dramatically the memory of the computer in implementing an algorithm. This technique is pretty much suitable for the algorithm developed. A further extension for this algorithm on higher dimensions with the use of the technique is hopefully expected to be a useful and efficient algorithm in a future paper.

REFERENCES

1. Dinh Van Tiep (2017), "An algorithm to approximate double integrals by using adaptive quadrature method", *Journal of Sciences and Technology*, TNU, 176(16), pp 121-126.

2. Richard L. Burden, J. Douglas Faires (2010), *Numerical Analysis*, 9th Edition, Brooks/Cole.

3. William M. McKeeman (1962), "Algorithm 145: Adaptive numerical integration by Simpson's rule". Commun. ACM 5(12), pp 604.

4. John R. Rice (1975), "A Metalgorithm for Adaptive Quadrature". *Journal of the* ACM, 22(1), pp 61-8.

TÓM TẮT

PHƯƠNG PHÁP CẦU PHƯƠNG THÍCH ỨNG XẤP XỈ TÍCH PHÂN KÉP TRÊN MIỀN HÌNH CHỮ NHẬT CONG

Phạm Thị Thu Hằng, Đinh Văn Tiệp*

Trường Đại học Kỹ thuật Công nghiệp – ĐH Thái Nguyên

Mới đây, trong bài báo [1], tác giả đã giới thiệu một thuật toán xấp xỉ tích phân kép trên miền hình chữ nhật dựa trên phương pháp cầu phương thích ứng. Phương pháp này tỏ ra vượt trội hơn các phương pháp cầu phương khác bởi chi phí thấp song hiệu quả cao. Trong bài báo [1], mặc dù thuật toán chỉ được xây dựng cụ thể và chặt chẽ cho tích phân kép trên miền hình chữ nhật, nhưng nó có thể được mở rộng sang trường hợp của miền hình chữ nhật cong. Tuy nhiên, với miền hình chữ nhật cong, một chương trình được lập trình dựa trên thuật toán đó sẽ khá rườm rà bởi việc sử dụng một dung lượng lớn RAM máy tính để lưu trữ dữ liệu về giá trị hàm số trên biên của miền. Việc lưu trữ này được thực hiện trong nhiều bước tại mỗi vòng lặp. Điều này khiến chương trình thực thi chậm chạp, độ chính xác thấp. Bài báo này sẽ đưa ra giải pháp khắc phục một cách hiệu quả nhược điểm này.

Từ khóa: *tích phân số, xấp xỉ tích phân kép, phương pháp thích ứng, cầu phương thích ứng, miền hình chữ nhật cong.*

Ngày nhận bài: 27/4/2018; Ngày phản biện: 23/5/2018; Ngày duyệt đăng: 31/5/2018

* *Tel: 0888 272060, Email: tiepdinhvan@gmail.com*