Prosta, efektywna kwadratura adaptacyjna w języku C

A simple and effective adaptive quadrature in C

Paweł Keller

Wrocławska Wyższa Szkoła
Informatyki Stosowanej
ul. Wejherowska 28, 54-239 Wrocław

Treść. Prezentujemy dwa proste, a jednocześnie bardzo skuteczne algorytmy adaptacyjne przybliżania wartości całki oznaczonej funkcji rzeczywistej. Proponowane algorytmy działają w oparciu o zasadę *dziel i zwyciężaj*, i wykorzystują znane kwadratury wysokiego rzędu.

Słowa kluczowe: kwadratura, kwadratura adaptacyjna, całkowanie numeryczne, całka oznaczona.

Abstract. We present two simple and very effective adaptive algorithms for approximating a definite integral of a real function. The proposed methods are based on the *divide and conquer* rule and use well known high order quadrature rules.

Keywords: quadrature, adaptive quadrature, numerical integration, definite integral.

1 Wprowadzenie

Konieczność obliczenia liczbowej wartości całki oznaczonej danej funkcji ciągłej pojawia się w niezliczonej ilości zagadnień naukowych i technicznych. W sytuacji, kiedy całka nieoznaczona pewnej funkcji nie daje się wyrazić za pomocą funkcji elementarnych, zwykle jedynym sposobem na wyznaczenie całki oznaczonej jest przybliżenie jej wartości jedną z metod całkowania numerycznego. Metody takie nazywamy *kwadraturami*. W niniejszej pracy zajmować się będziemy problemem wyznaczenia wartości całki

$$\int_a^b f(x)\,dx\,, \tag{1.1}$$

przy założeniu, że potrafimy obliczyć wartość funkcji podcałkowej f w dowolnym punkcie przedziału $[a,b]$. Zakładamy przy tym, że funkcja podcałkowa nie jest z góry znana. Oznacza to, że nie możemy dopasować metody całkowania do konkretnej funkcji występującej w (1.1). Najlepszym rozwiązaniem w takiej sytuacji wydają się być kwadratury adaptacyjne, które z założenia powinny poprawnie obliczać wartość całki oznaczonej dla możliwie najszerszej klasy funkcji. Ideę działania i konstrukcji kwadratur adaptacyjnych opisujemy dokładniej w rozdziałach 2 i 3.

Kiedy korzystamy z zaawansowanego systemu obliczeń matematycznych, jak *Matlab*, *Maple* czy *Matematica*, możemy skorzystać z wbudowanych procedur całkujących. Jeśli jednak zachodzi potrzeba obliczenia całki typu (1.1) w programie pisanym w jednym z języków niższego poziomu (w C, Javie lub podobnym) musimy albo samodzielnie zaprogramować odpowiedni algorytm albo znaleźć gotową procedurę napisaną w danym języku programowania, wierząc w jej niezawodność.

Klasyką wśród publikacji naukowych dotyczących kwadratur adaptacyjnych jest praca Gandera i Gautchiego [4] z 2000 roku. Do ciekawszych pozycji należy również wcześniejsza propozycja [3], a także całkiem nowa praca Shampine [6] z 2007 roku, na podstawie której powstała nowa wektorowa kwadratura adaptacyjna dostępna od trzech lat w systemie obliczeń matematycznych *MatLab*.

Przeglądając natomiast sieć WWW w poszukiwaniu stron zawierających hasło „*kwadratura adaptacyjna w C*" lub jego angielski odpowiednik, udało się autorom łatwo odnaleźć cztery odnośniki ([8], [9], [10] i [11]) do witryn oferujących biblioteki lub procedury pozwalające obliczać w języku C całki postaci (1.1). Biblioteka [11] („NAG Library") jest płatna i nie będziemy z niej korzystać podczas testów. Procedura oferowana na stronie [8] potrzebowała natomiast aż 11 sekund aby przybliżyć wartość całki funkcji e^x na przedziale [-1,1] z błędem bezwzględnym nieprzekraczającym 10^{-10}, i dlatego również nie będzie uwzględniana podczas testów (proponowana w tej pracy kwadratura oblicza tę całkę, na tym samym komputerze, w czasie krótszym niż 10 mikrosekund, z błędem mniejszym niż 10^{-15}). Metody oferowane na stronach [9] i [10] działają poprawnie, jednak nie pozwalają zadać dokładności obliczanego przybliżenia, ani nie podają oszacowania błędu obliczonej wartości całki, co jest sporym mankamentem w wypadku kwadratur adaptacyjnych. Będą jednak wykorzystane w testach, aby można było lepiej ocenić skuteczność i efektywność proponowanych przez nas algorytmów.

W pracy prezentujemy dwa zbliżone do siebie adaptacyjne algorytmy przybliżania wartości całki (1.1). Naszym celem było zaproponowanie kwadratury prostej w swej konstrukcji (aby można ją było bardzo łatwo zaimplementować w dowolnym języku programowania), a jednocześnie szybkiej i dokładnej. Testy numeryczne przedstawione w rozdziale 4

pokazują, że cel został osiągnięty. Wcześniej, w rozdziałach 2 i 3 opiszemy dokładnie zasadę działania proponowanych kwadratur adaptacyjnych.

W rozdziale 5 podajemy, zapisane w języku C (standard ISO/IEC 9899), kody źródłowe procedur będących implementacją algorytmu opisanego w niniejszej pracy.

2 Wybór kwadratury podstawowej i oszacowanie błędu przybliżenia

Idea działania kwadratury adaptacyjnej jest bardzo prosta. Wartość całki (1.1) nie jest obliczana *od razu* na całym przedziale $[a, b]$, ale przedział ten dzielony jest na szereg podprzedziałów i ostateczny wynik jest obliczany jako suma wartości całek na każdym z nich. Co istotne, podprzedziały te nie są jednakowej długości. W miejscach, w których funkcja podcałkowa zachowuje się regularnie są one dłuższe, a w obszarach silnej zmienności funkcji znacznie krótsze. Ponieważ, jak zaznaczono we wstępie, funkcja f w (1.1) nie jest z góry znana, podział odcinka $[a, b]$ musi być tworzony dynamicznie w trakcie obliczania wartości całki.

W tej pracy rozważać będziemy jedynie schemat doboru długości podprzedziałów oparty na zasadzie połowienia. Ogólnie, sposób działania kwadratury adaptacyjnej można opisać następująco:

Algorytm 1 *Schemat adaptacyjnego algorytmu przybliżania wartości całki $\int_a^b f(x)dx$ z błędem bezwzględnym nie przekraczającym δ.*

Krok 1. Wyznacz *pewnym sposobem* przybliżenie I całki funkcji f na przedziale $[a, b]$ oraz oszacowanie ε błędu bezwzględnego tego przybliżenia.

Krok 2. Jeśli $\varepsilon < \delta$, zaakceptuj I jako dobre przybliżenie całki i zakończ.

Krok 3. Wyznacz środek przedziału: $m = (a + b)/2$.

Krok 4. Oblicz rekurencyjnie przybliżenie I_1 całki $\int_a^m f(x)dx$ z błędem bezwzględnym nie przekraczającym $c\,\delta$, dla pewnej ustalonej stałej $\frac{1}{2} \le c \le 1$.

Krok 5. Oblicz rekurencyjnie przybliżenie I_2 całki $\int_m^b f(x)dx$ z błędem bezwzględnym nieprzekraczającym $c\,\delta$.

Krok 6. Przyjmij $I = I_1 + I_2$ i zakończ.

Łatwo zauważyć, że sumując oszacowania błędów bezwzględnych obliczonych przybliżeń całek na wyznaczonych w algorytmie podprzedziałach otrzymamy oszacowanie błędu całki na całym przedziale $[a, b]$.

Pozostają jednak wciąż dwie niewyjaśnione kwestie. Jak w kroku 1 wyznaczyć przybliżenie I oraz jak wyznaczyć jego wiarygodne oszacowanie ε. Zanim to opiszemy zajmiemy się jeszcze problemem żądanej tolerancji błędu w krokach 4 i 5.

Jest rzeczą intuicyjną, że skoro przedział całkowania jest dzielony na dwie równe części, to chcemy aby błędy przybliżeń całki na każdej połowie przedziału nie przekraczały $\frac{1}{2}\delta$. Wtedy błąd na całym przedziale będzie na pewno nie większy niż δ. W [7, rozdz. 5.2] uzasadniono jednak, że takie podejście jest nazbyt ostrożne i niepotrzebnie wydłuża obliczenia. Zwykle w krokach 4 i 5 Algorytmu 1 można przyjąć $c = 1$, jednak dla „trudnych" funkcji zbyt często zdarza się wtedy, że otrzymany błąd przybliżenia całki na całym przedziale jest nazbyt duży. Na podstawie wielu przeprowadzonych doświadczeń, w zaproponowanych w tej pracy metodach adaptacyjnych przyjmujemy $c = 0.8125$.

Powróćmy teraz do zagadnienia wyznaczenia przybliżenia całki w kroku 1 Algorytmu 1. Użytą w tym celu kwadraturę nazywać będziemy *kwadraturą podstawową*. W naszej pracy rozważać będziemy jedynie kwadratury liniowe postaci

$$Q(f, a, b) = \sum_{k=0}^{n} w_k f(x_k) \approx \int_a^b f(x)\,dx\,, \qquad (2.1)$$

gdzie ustalone wartości $a \le x_0 < x_1 < \cdots < x_n \le b$ nazywamy *węzłami* kwadratury, a wielkości $w_0, w_1, \ldots, w_n$ *wagami* lub *współczynnikami* kwadratury. Spośród kwadratur postaci (2.1) nie da się wybrać jednej, która gwarantowałaby najmniejszy błąd przybliżenia dla wszystkich całkowalnych funkcji f. Dlatego często stosuje się inną miarę jakości danej kwadratury.

Definicja 1 Mówimy, że kwadratura (2.1) jest *rzędu* r, jeśli jest ona dokładna dla każdego wielomianu stopnia niższego niż r, ale już nie dla każdego wielomianu stopnia r.

Ponieważ każdą funkcję ciągłą można dowolnie dobrze przybliżyć na odcinku domkniętym za pomocą wielomianu, można przypuszczać, że im wyższy rząd kwadratury, tym powinna ona średnio lepiej przybliżać wartość całki. W naszej metodzie wykorzystamy kwadratury możliwie najwyższych rzędów.

Omówimy teraz kwestię oszacowania ε błędu przybliżenia całki w kroku 1 Algorytmu 1, czyli oszacowania wielkości

$$\left| \int_a^b f(x)\,dx - Q(f, a, b) \right|.$$

Najczęstszym rozwiązaniem tego zagadnienia jest zastosowanie pary kwadratur

$$\begin{aligned} Q_1(f,a,b) &= \sum_{k=0}^{n} w_k f(x_k), \\ Q_2(f,a,b) &= \sum_{k=0}^{s} v_k f(y_k), \end{aligned} \tag{2.2}$$

gdzie kwadratura Q_2 powinna być teoretycznie znacznie dokładniejsza niż kwadratura Q_1. Za przybliżenie wartości całki przyjmuje się wartość kwadratury Q_2, natomiast błąd bezwzględny przybliżenia szacuje się w oparciu o wielkość modułu różnicy wartości obu kwadratur. Często po prostu przyjmuje się

$$\varepsilon = |Q_1(f,a,b) - Q_2(f,a,b)|, \tag{2.3}$$

choć nie jest to regułą.

W praktyce ważne jest też, aby $\{x_0, x_1, \ldots, x_n\} \subset \{y_0, y_1, \ldots, y_s\}$. Wtedy żadna z obliczonych wartości $f(x_k)$ $(k = 0, 1, \ldots, n)$ się nie marnuje.

Najbardziej popularnymi w profesjonalnych algorytmach adaptacyjnych obliczania całek parami kwadratur postaci (2.2) są tak zwane pary Gaussa-Kronroda. Kwadratura Q_1 jest kwadraturą Gaussa-Legendre'a, czyli kwadraturą mającą możliwie najwyższy rząd przy ustalonej liczbie węzłów. Kwadratura Q_2 jest tak zwanym rozszerzeniem Kronroda, czyli kwadraturą o najwyższym rzędzie spośród kwadratur opartych na $2n + 3$ węzłach zawierających wszystkie węzły kwadratury Q_1 (więcej informacji na temat kwadratur Gaussa-Kronroda można znaleźć w [2, rozdz. 2.7.1.1].

W proponowanej w naszej pracy metodzie zastosujemy inne podejście. Skorzystamy mianowicie z pojedynczej kwadratury podstawowej. Dzięki temu mamy możliwość zastosowania w schemacie adaptacyjnym praktycznie dowolnej kwadratury postaci (2.1). Do oszacowania błędu i przybliżenia wartości całki wykorzystamy natomiast kwadratury utworzone następująco:

$$\begin{aligned} Q_1(f,a,b) &= Q(f,a,b), \\ Q_2(f,a,b) &= Q(f,a,m) + Q(f,m,b), \end{aligned} \tag{2.4}$$

gdzie $m = (a+b)/2$, a Q jest wybraną kwadratura podstawową. Jeśli kwadratura Q w (2.1) ma wysoki rząd, to (porównaj [7, rozdz. 5.1]) jej błąd jest proporcjonalny do $(b-a)^p$, gdzie $p \gg 1$ (przy pewnych założeniach dotyczących funkcji f). Zdefiniowana w (2.4) kwadratura Q_2 powinna być zatem znacznie dokładniejsza niż kwadratura Q_1. Oszacowanie błędu będziemy obliczać zgodnie z (2.3).

Zauważmy, że przy takim podejściu w każdym rekurencyjnym kroku Algorytmu 1 oszczędzamy czas potrzebny na wyznaczenie wartości kwadratury Q_1, ponieważ odpowiednie wartości były już wyznaczone wcześniej dla każdej z obu połówek przedziału.

W naszej pracy proponujemy dwie metody adaptacyjne obliczania całki (1.1). Pierwsza z nich korzysta z kwadratury podstawowej Gaussa-Legendre'a. Przypomnijmy, że jest to kwadratura postaci (2.1) mająca możliwie najwyższy rząd (równy $2n + 2$) przy zadanej liczbie węzłów. Dodatkowe informacje na temat kwadratur Gaussa-Legendre'a można znaleźć np. w [1, rozdz. 5.3.2], [2, rozdz. 2.7] lub [5, rozdz. 7.3]. W drugiej, bliźniaczej metodzie kwadraturą podstawową jest kwadratura Lobatto. Jest to kwadratura postaci (2.1) mająca możliwie najwyższy rząd przy ustalonej liczbie węzłów i dodatkowych warunkach $x_0 = a$ i $x_n = b$ (więcej informacji na temat kwadratur Lobatto można znaleźć w [1, rozdz. 5.3.3] lub [2, rozdz. 2.7.1])

Kwadratury postaci (2.1) dla których $x_0 > a$ i $x_n < b$ nazywane są często kwadraturami *otwartymi*. Jeśli żaden z dwóch powyższych warunków nie jest spełniony, mówimy o kwadraturze *zamkniętej*. Ogólnie uważa się, że kwadratury otwarte mają lepsze własności aproksymacyjne, jednak nie powinny być one używane w algorytmach adaptacyjnych do przybliżania całek funkcji nieciągłych lub mających nieciągłą pierwszą pochodną. Rozważmy na przykład funkcję $f(x) = |x - d|$, gdzie d jest takie, że $d < x_0$, $d < y_0$ oraz $d > a$. W takiej sytuacji obie kwadratury Q_1 i Q_2 w (2.2) będą miały identyczną wartość, równą całce funkcji $g(x) = x - d$ (zakładamy, że kwadratury te są wysokiego rzędu, więc są dokładne dla funkcji liniowej). Zatem obliczone oszacowanie błędu wyniesie 0 i algorytm adaptacyjny się zakończy. W rzeczywistości natomiast błąd będzie równy $(d-a)^2$.

Mankamentu tego nie mają kwadratury zamknięte. Tych jednak nie można zastosować do przybliżania całek funkcji mających osobliwości na końcach przedziału lub w dowolnym punkcie podziału. Problem pojawia się również wtedy, gdy w takim punkcie wartość funkcji podcałkowej jest symbolem nieoznaczonym typu 0/0 (jak na przykład dla funkcji $x^{-1}\sin x$ w punkcie 0). Z tego też powodu w naszej pracy przedstawiamy dwie propozycje kwadratur adaptacyjnych.

Na zakończenie tego rozdziału pozostało jeszcze do ustalenia, ile węzłów mają mieć zastosowane w proponowanych algorytmach kwadratury podstawowe. Jeśli liczba węzłów będzie zbyt mała, kwadratura podstawowa nie będzie dobrze przybliżać całki i cały algorytm może działać za wolno. Podobnie, w wypadku za dużej liczby węzłów, łączna liczba wywołań funkcji podcałkowej może być zbyt duża, co też obniży efektywność algorytmu. Innymi słowy, kwadratura podstawowa nie powinna być *za dokładna*, bo wtedy część obliczeń będzie zmarnowana (niemal taki sam rezultat można było osiągnąć w danej

arytmetyce używając mniejszej liczby węzłów). Rozsądna liczba węzłów, w zależności od funkcji podcałkowej, to od 8 do 40. Jako, że nie wiemy jakich funkcji całki ma nasz algorytm obliczać, w obu zaproponowanych przez nas kwadraturach adaptacyjnych korzystamy z rozwiązania pośredniego, 18. punktowych kwadratur podstawowych.

Jak łatwo sprawdzić, jeśli

$$Q(f,0,1) \;=\; \sum_{k=0}^{n} w_k f(x_k)\,,$$

to analogiczna kwadratura dla dowolnego przedziału $[a,b]$ wyraża się wzorem

$$Q(f,a,b) \;=\; (b-a)\sum_{k=0}^{n} w_k f((b-a)x_k + a)\,. \quad (2.5)$$

W programie realizującym algorytm adaptacyjny obliczania całki (1.1) należy zatem pamiętać zestaw węzłów i współczynników kwadratury podstawowej dla jednego wybranego przedziału. Ze względu na prostą postać wzoru (2.5) jest to zwykle przedział $[0,1]$.

3 Oszacowania błędów zaokrągleń, wcześniejsze zakończenie obliczeń

Schemat adaptacyjny opisany w poprzednim rozdziale nie nadaje się jeszcze do zastosowań praktycznych. Wystarczy, że użytkownik zada tolerancję $\delta = 0$ i algorytm może nigdy się nie zakończyć. Wprowadzenie dolnego ograniczenia na wartość tolerancji błędu bezwzględnego niewiele pomoże, ponieważ zależy to w istotny sposób od postaci funkcji podcałkowej f. Jeśli na przykład $f(x) = 10^{10}e^x$, to najmniejszy błąd bezwzględny jakiego możemy się spodziewać stosując arytmetykę podwójnej precyzji wynosi ok. 10^{-6}. Wynika z tego, że dolne ograniczenie na wartość parametru tolerancji błędu powinno być w razie potrzeby modyfikowane podczas obliczeń.

W związku z powyższym, w proponowanych algorytmach, w trakcie wyznaczania wartości kwadratury podstawowej (2.1) obliczamy dodatkowo wielkość

$$S = \frac{1}{n+1}\sum_{k=0}^{n} |f(x_k)|$$

i modyfikujemy wartość tolerancji następująco:

$$\delta \;:=\; \max\{\delta,\, \epsilon_{mach} S\}\,,$$

gdzie ϵ_{mach} to tak zwany *epsilon maszynowy*, czyli najmniejsza wartość, dla której liczby $z_1 = 1$ i $z_2 = 1 + \epsilon_{mach}$ są różne w danej arytmetyce zmiennoprzecinkowej. Dodatkowo, wyznaczamy oszacowanie bezwzględnych błędów zaokrągleń obliczania kwadratury podstawowej:

$$\sigma := (b-a)\,S\,.$$

Oszacowania błędów zaokrągleń są sumowane i na zakończenie algorytmu dodawane do końcowego oszacowania błędu bezwzględnego obliczonego przybliżenia całki.

Eksperymenty pokazały, że w wypadku kwadratury podstawowej Gaussa-Legendre'a oraz stromo nachylonych i trudnych do całkowania funkcji obliczone oszacowanie błędu może być czasami zbyt małe. Dlatego, ostatecznie, końcowe oszacowanie błędu bezwzględnego wyznaczamy następująco:

$$\bar{\varepsilon} \;=\; \varepsilon(1 + CL) + D\,\frac{\epsilon_{mach}|q_2 - q_1|}{2(b-a)} + \bar{\sigma}\,,$$

gdzie ε jest obliczane jak w (2.3), L jest liczbą rekurencyjnych, zagnieżdżonych podziałów wyjściowego przedziału, $q_1 = Q(f,a,m)$, $q_2 = Q(f,m,b)$ (porównaj (2.4)), $\bar{\sigma}$ jest sumą oszacowań błędów zaokrągleń powstałych podczas obliczania wartości q_1 i q_2, a C i D są pewnymi stałymi. W wypadku kwadratur podstawowych Gaussa-Legendre'a i arytmetyki podwójnej precyzji przyjmujemy $C = \frac{3}{2\cdot 40}$ i $D = 1$ (dla arytmetyki pojedynczej precyzji przyjmiemy $C = \frac{3}{2\cdot 20}$). Stałe te wyznaczone zostały na podstawie wielu doświadczeń. W wypadku kwadratur podstawowych Lobatto kładziemy $C = 0$ i $D = 0$. Decyzję o kontynuowaniu algorytmu adaptacyjnego (krok 2 Algorytmu 1) podejmujemy jednak tylko na podstawie wartości ε z (2.3).

Aby uniknąć zbyt długich obliczeń w wypadku ekstremalnie skomplikowanych funkcji, w praktycznej realizacji Algorytmu 1, w kroku 2 przerywamy obliczenia także jeśli: liczba rekurencyjnych podziałów początkowego przedziału jest większa lub równa 40 (20 dla pojedynczej precyzji), długość powstałego po podziale podprzedziału jest mniejsza niż $250\epsilon_{mach}$ ($250 \approx 15\,n$), liczba wywołań funkcji podcałkowej przekroczy $2{\cdot}10^7$.

W rozdziale 5 prezentujemy procedury napisane w języku C, będące implementacją opisanej wyżej metody adaptacyjnego przybliżania całki (1.1), z wykorzystaniem kwadratury Gaussa-Legendre'a jako kwadratury podstawowej. W przedstawionych procedurach umożliwiamy dodatkowo zadanie minimalnej liczby rekurencyjnych podziałów przedziału początkowego. Zwiększenie tego parametru może być przydatne w wypadku funkcji, które zmieniają się mocno na bardzo wąskim odcinku przedziału początkowego (przykładem takim może być funkcja $f(x) =$

$e^{-(10000x)^2}$, która poza odcinkiem $[-0.0006, 0.0006]$ przyjmuje wartości praktycznie równe 0).

4 Testy numeryczne

W tym rozdziale przedstawimy wyniki eksperymentów numerycznych otrzymane podczas obliczania całek

$$\int_{-1}^{1} f_i(x)\,dx$$

dla zestawu następujących funkcji testowych:

$f_1(x) = x\sin(3x)$,

$f_2(x) = (x-\frac{1}{2})^2\sin(13x) + 20e^{-(10x)^2}$,

$f_3(x) = (1.000001 + x)^{-1}$,

$f_4(x) = \sqrt{2+\cos(100x)}$,

$f_5(x) = (1+x)\sin(\frac{1}{1+x})$,

$f_6(x) = 1000\,(1+x)\sin(\frac{1}{1+x})$,

$f_7(x) = e^{\sqrt{|5x|^3}}$,

$f_8(x) = \log(1+x)\sqrt{\frac{2+x}{1-x}}$,

$f_9(x) = \log(\cos(30x)^2)$,

$f_{10}(x) = |\cos(20.001\,\pi x)|$.

Wszystkie całki obliczano na tym samym przedziale, aby zmniejszyć ilość informacji zawartych w tabelach z wynikami. Obliczenia wykonano na komputerze z procesorem Intel Core2 pracującym z częstotliwością 3.16GHz. Testy przeprowadzono w arytmetyce podwójnej precyzji.

Zaproponowane w tej pracy kwadratury adaptacyjne będziemy w skrócie nazywać G18, jeśli kwadraturą podstawową kwadratura Gaussa-Legendre'a, oraz L18, gdy kwadraturą podstawową jest kwadratura Lobatto. Kwadraturę z pracy [3], dostępną na stronie internetowej [10], nazwiemy RMS, a kwadraturę z biblioteki [9] – ALGLIB.

Przypomnijmy, że w wypadku algorytmów G18 i L18 można zadać dopuszczalną tolerancję δ błędu bezwzględnego obliczonego przybliżenia. Algorytmy te podają również oszacowanie tego błędu. Jeśli $\bar{\varepsilon}$ jest obliczonym oszacowaniem błędu bezwzględnego, a γ jego rzeczywistą wartością, to idealna sytuacja ma miejsce, gdy $\gamma \le \bar{\varepsilon} \le \delta$. Ponieważ jednak nie zawsze w danej arytmetyce żądaną dokładność da się uzyskać, uznamy, że algorytm działa poprawnie, jeśli $\gamma \le \bar{\varepsilon}$. Algorytmy RMS i ALGLIB liczą całkę najdokładniej jak potrafią i nie oferują oszacowań błędu obliczonego przybliżenia.

W Tabeli 1 porównujemy dokładność (błąd bezwzględny) i czas działania algorytmów G18, L18, RMS i ALGLIB dla testowego zestawu funkcji podcałkowych $f_1, f_2, \ldots, f_9$. Dla algorytmów G18 i L18 podajemy dodatkowo obliczone przez nie oszacowanie błędu. W Tabeli 2 porównujemy kwadraturę adaptacyjną G18 z kwadraturą `quadgk` systemu obliczeń matematycznych *MatLab*. Podane w tej tabeli czasy należy traktować orientacyjnie, gdyż trudno obiektywnie porównać czas działania fragmentu programu w języku C z czasem działania procedury wbudowanej w pewien system obliczeń matematycznych.

Nasze algorytmy adaptacyjne w łatwy sposób mogą podawać, w ilu łącznie punktach przedziału należało obliczyć wartość funkcji podcałkowej (porównaj kody źródłowe procedur w rozdziale 5). Dla przykładu, dla zadanej wartości tolerancji $\delta = 10^{-14}$, wartość funkcji f_1 wystarczyło obliczyć 54 razy, natomiast wartość funkcji f_5 już 1710342 razy.

Przetestujemy jeszcze możliwość zastosowania naszych algorytmów do obliczania całek na płaszczyźnie. Zauważmy, że

$$\int_a^b \int_c^d f(x,y)\,dx\,dy \;=\; \int_a^b g(y)\,dy\,,$$

gdzie

$$g(y) = \int_c^d f(x,y)\,dx\,.$$

Zastosowaliśmy powyższy schemat i prezentowane w pracy kwadratury adaptacyjne, prosząc o przybliżenia całki

$$\int_{-10}^{10}\int_{-10}^{10} \frac{\cos(x^2+y^2+1)}{x^2+y^2+1}\,dx\,dy$$

z błędem nieprzekraczającym 10^{-12}. Oba prezentowane algorytmy adaptacyjne spisały się dobrze, podając w czasie 67 milisekund wynik z błędem mniejszym niż $3{\cdot}10^{-15}$.

4.1 Podsumowanie i wnioski

Jak wynika z zamieszczonych wyników eksperymentów, obie zaproponowane metody, choć bardzo proste, z powodzeniem konkurują ze znacznie bardziej zaawansowanymi algorytmami. Na podstawie przeprowadzonych testów nie można stwierdzić, aby którakolwiek z porównywanych metod była najlepsza. Jeśli przy obliczaniu wartości funkcji podcałkowej nie występuje dzielenie przez 0, najlepiej spisuje się algorytm L18. Metoda RMS również działa bardzo dobrze, lecz momentami kapryśnie. Dzieje się tak za sprawą wbudowanego w nią mechanizmu ekstrapolacji, który często daje bardzo dobre rezultaty (funkcja f_8), ale czasami całkiem zawodzi

(funkcja f_9). Algorytm G18 spisywał się poprawnie dla wszystkich przykładowych całek i działa bardzo podobnie jak oparty na parze kwadratur Gaussa-Kronroda algorytm `quadgk` z systemu *MatLab*. Choć zdaniem autorów ten ostatni zbyt często podaje trochę niepoprawne oszacowanie błędu.

Jak przewidziano w rozdziale 2, wszystkie metody adaptacyjne wykorzystujące otwartą kwadraturę podstawową mniej lub bardziej zawiodły w wypadku funkcji f_{10}, która ma nieciągłą pierwszą pochodną. W tym wypadku jedynie algorytm L18 podał poprawne wyniki.

5 Kwadratura adaptacyjna – kod źródłowy

Procedura 1 *18. punktowa kwadratura podstawowa Gaussa-Legendre'a. Argumenty:* `f` *– funkcja podcałkowa;* `a, b` *– końce przedziału;* `eps` *– epsilon maszynowy;* `*tol, *fe` *– (uaktualniane) tolerancja błędu kwadratury adaptacyjnej i oszacowanie błędów zaokrągleń;* `*n` *– (uaktualniana) liczba wywołań funkcji podcałkowej. Wartością procedury jest przybliżona wartość całki funkcji f na przedziale* $[a, b]$.

```
double BaseQuad(double f(double), double a, double b, double eps,
                double *tol, double *fe, long *n)
{
  #define N 18
  static double x[N] = { // węzły kwadratury ...
     4.217415789534526634992e-03, 2.208802521430112240940e-02, 5.369876675122213039697e-02,
     9.814752051373844215879e-02, 1.541564784698233960626e-01, 2.201145844630262326961e-01,
     2.941244192685786769820e-01, 3.740568871542472452055e-01, 4.576124934791323493789e-01,
     5.423875065208676506211e-01, 6.259431128457527547945e-01, 7.058755807314213230180e-01,
     7.798854155369737673039e-01, 8.458435215301766039374e-01, 9.018527948626215578412e-01,
     9.463012332487778696030e-01, 9.779119747856988775906e-01, 9.957825842104654733650e-01 };
  static double w[N] = { // współczynniki kwadratury ...
     1.080800676324165515667e-02, 2.485727444748489822667e-02, 3.821286512744452826456e-02,
     5.047102205314358278141e-02, 6.127760335573923009226e-02, 7.032145733532532560237e-02,
     7.734233756313262246271e-02, 8.213824187291636149303e-02, 8.457119148157179592033e-02,
     8.457119148157179592033e-02, 8.213824187291636149303e-02, 7.734233756313262246271e-02,
     7.032145733532532560237e-02, 6.127760335573923009226e-02, 5.047102205314358278141e-02,
     3.821286512744452826456e-02, 2.485727444748489822667e-02, 1.080800676324165515667e-02 };
  double b_a = (double)(b-a);
  double mfx = 0.0;          // średnia wartość eps*|f(x)|
  double s = 0.0;            // obliczana całka
  double fx;                 // f(x)
  for (int i=0; i<N; i++) {
    fx = f((b_a)*x[i]+a);
    mfx += fabs(fx);
    s += fx*w[i];
  }
  *n += N;
  mfx = eps*mfx/(double)N;   // szacowanie wpływu błędów zaokrągleń...
  if ( mfx > *tol ) { *tol = mfx; }
  *fe += b_a*mfx;
  return b_a*s;
}
```

Procedura 2 *Rekurencyjna kwadratura adaptacyjna. Argumenty:* `f` *– funkcja podcałkowa;* `a, b` *– końce przedziału;* `w0` *– poprzednia wartość całki;* `tol` *– tolerancja błędu kwadratury adaptacyjnej;* `lev` *– poziom zagłębienia rekurencyjnego;* `eps` *– epsilon maszynowy;* `*err` *– oszacowany błąd bezwzględny obliczonego przybliżenia;* `*n` *– liczba wywołań funkcji podcałkowej. Wartością procedury jest przybliżona wartość całki funkcji f na przedziale* $[a, b]$.

```
double adaptiveR(double f(double), double a, double b, double w0, double tol,
                 int lev, double eps, double *err, long *n)
{
  #define TolM 0.8125        // modyfikator tolerancji błędu
  #define ErM1 0.0375        // parametr szacowania błędu (0.0 dla kw. Lobatto)
  #define ErM2 1.0000        // parametr szacowania błędu (0.0 dla kw. Lobatto)
  #define MaxN 20000000      // maks. liczba wywołań funkcji podcałkowej
  #define MivL 250.0         // MivL*eps = min. długość podprzedziału
  #define MaxL 40            // maks. liczba zagłębień rekurencyjnych
  #define MinL 1             // min. liczba zagłębień rekurencyjnych
  double e;                  // oszacowanie błędu przybliżenia
  double fe = 0.0;           // błąd zaokrągleń
  double m = (a+b)*0.5;
  double w1 = BaseQuad(f,a,m,eps,&tol,&fe,n);
  double w2 = BaseQuad(f,m,b,eps,&tol,&fe,n);
  e = fabs(w1+w2-w0);
  if ( lev >= MinL && ( e < tol || (b-a)<MivL*eps || lev >= MaxL || *n > MaxN ) ) {
    *err += e*(1.0+ErM1*(double)lev) + ErM2*eps*fabs((w2-w1)/(m-a)) + fe;
    return w1+w2;
  }
  else {
```

```
    tol = TolM*tol;
    return adaptiveR(f,a,m,w1,tol,lev+1,eps,err,n) +
           adaptiveR(f,m,b,w2,tol,lev+1,eps,err,n);
  }
}
```

Procedura 3 *Obliczanie wartości epsilon maszynowego (precyzji danej arytmetyki).*

```
double eps_mach()
{
  double x = 0.125;
  double y = 1.0 + x;
  while ( y != 1.0 ) {
    x /= 2.0;
    y = 1.0 + x;
  }
  return x*2.0;
}
```

Procedura 4 *Główna procedura całkująca. To jej należy używać w programach korzystających z proponowanej metody. Argumenty:* `f` *– funkcja podcałkowa;* `a,b` *– końce przedziału;* `tol` *– tolerancja błędu kwadratury adaptacyjnej;* `*err` *– oszacowany błąd bezwzględny obliczonego przybliżenia;* `*n` *– liczba wywołań funkcji podcałkowej. Wartością procedury jest przybliżona wartość całki funkcji f na przedziale $[a,b]$.*

```
double aGL(double f(double), double a, double b, double tol,
           double *err, long *n)
{
  double fe = 0.0;  // zmienna nieużywana; potrzebna do wywołania procedury BaseQuad
  *err = eps_mach();
  *n = 0;
  return adaptiveR(f,a,b,BaseQuad(f,a,b,*err,&tol,&fe,n),tol,1,*err,err,n);
}
```

6 Tabele

		G18		L18		RMS		ALGLIB	
f	tol.	błąd (oszac.)	czas	błąd (oszac.)	czas	błąd	czas	błąd	czas
f_1	10^{-10}	$1{\cdot}10^{-16}$ ($5{\cdot}10^{-16}$)	18	$1{\cdot}10^{-16}$ ($4{\cdot}10^{-16}$)	18				
	10^{-14}	$1{\cdot}10^{-16}$ ($5{\cdot}10^{-16}$)	18	$1{\cdot}10^{-16}$ ($4{\cdot}10^{-16}$)	18	$1{\cdot}10^{-16}$	24	$1{\cdot}10^{-16}$	35
f_2	10^{-10}	$2{\cdot}10^{-16}$ ($6{\cdot}10^{-14}$)	117	$2{\cdot}10^{-16}$ ($7{\cdot}10^{-13}$)	116				
	10^{-14}	$2{\cdot}10^{-16}$ ($8{\cdot}10^{-15}$)	171	$2{\cdot}10^{-16}$ ($2{\cdot}10^{-15}$)	170	$2{\cdot}10^{-16}$	929	$7{\cdot}10^{-16}$	257
f_3	10^{-10}	$9{\cdot}10^{-11}$ ($1{\cdot}10^{-10}$)	17	$9{\cdot}10^{-11}$ ($1{\cdot}10^{-10}$)	17				
	10^{-14}	$9{\cdot}10^{-11}$ ($1{\cdot}10^{-10}$)	17	$9{\cdot}10^{-11}$ ($1{\cdot}10^{-10}$)	17	$8{\cdot}10^{-11}$	286	$8{\cdot}10^{-11}$	165
f_4	10^{-10}	$3{\cdot}10^{-16}$ ($1{\cdot}10^{-10}$)	657	$1{\cdot}10^{-16}$ ($1{\cdot}10^{-15}$)	696				
	10^{-14}	$3{\cdot}10^{-16}$ ($8{\cdot}10^{-15}$)	696	$1{\cdot}10^{-16}$ ($1{\cdot}10^{-15}$)	696	$6{\cdot}10^{-16}$	788	$1{\cdot}10^{-16}$	1430
f_5	10^{-4}	$2{\cdot}10^{-6}$ ($3{\cdot}10^{-5}$)	72	– (–)	–				
	10^{-6}	$6{\cdot}10^{-8}$ ($3{\cdot}10^{-7}$)	180	– (–)	–				
	10^{-8}	$2{\cdot}10^{-9}$ ($8{\cdot}10^{-9}$)	597	– (–)	–	$7{\cdot}10^{-9}$	806		
	10^{-10}	$2{\cdot}10^{-11}$ ($2{\cdot}10^{-10}$)	3950	– (–)	–				
	10^{-12}	$2{\cdot}10^{-13}$ ($4{\cdot}10^{-12}$)	25600	– (–)	–			$5{\cdot}10^{-13}$	149000
	10^{-14}	$3{\cdot}10^{-15}$ ($8{\cdot}10^{-14}$)	162000	– (–)	–				
	0	$2{\cdot}10^{-16}$ ($8{\cdot}10^{-15}$)	754000	– (–)	–				
f_6	10^{-5}	$2{\cdot}10^{-6}$ ($8{\cdot}10^{-6}$)	589	– (–)	–	$7{\cdot}10^{-6}$	810		
	10^{-10}	$7{\cdot}10^{-11}$ ($6{\cdot}10^{-10}$)	67200	– (–)	–			$5{\cdot}10^{-10}$	145000
	10^{-14}	$1{\cdot}10^{-13}$ ($8{\cdot}10^{-12}$)	75800	– (–)	–				
f_7	10^{-10}	$7{\cdot}10^{-14}$ ($4{\cdot}10^{-11}$)	256	$2{\cdot}10^{-12}$ ($1{\cdot}10^{-11}$)	286	$6{\cdot}10^{-12}$	742	$2{\cdot}10^{-11}$	241
	10^{-14}	$4{\cdot}10^{-12}$ ($2{\cdot}10^{-11}$)	319	$4{\cdot}10^{-12}$ ($5{\cdot}10^{-12}$)	347				
f_8	10^{-10}	$8{\cdot}10^{-8}$ ($8{\cdot}10^{-8}$)	818	– (–)	–			–	–
	10^{-14}	$8{\cdot}10^{-8}$ ($8{\cdot}10^{-8}$)	882	– (–)	–	$9{\cdot}10^{-13}$	180		
f_9	10^{-10}	$5{\cdot}10^{-12}$ ($8{\cdot}10^{-11}$)	13700	$1{\cdot}10^{-12}$ ($1{\cdot}10^{-10}$)	13700	$4{\cdot}10^{-3}$	1160		
	10^{-14}	$5{\cdot}10^{-12}$ ($3{\cdot}10^{-11}$)	14600	$5{\cdot}10^{-13}$ ($9{\cdot}10^{-12}$)	14700			$5{\cdot}10^{-13}$	15700
f_{10}	10^{-10}	$4{\cdot}10^{-7}$ ($6{\cdot}10^{-11}$)	2630	$3{\cdot}10^{-12}$ ($3{\cdot}10^{-11}$)	3300	$1{\cdot}10^{-4}$	540	$4{\cdot}10^{-7}$	3490
	10^{-14}	$4{\cdot}10^{-7}$ ($1{\cdot}10^{-14}$)	4110	$8{\cdot}10^{-16}$ ($2{\cdot}10^{-15}$)	5170				

Tabela 1: Porównanie dokładności i efektywności (czas w mikrosekundach) adaptacyjnych algorytmów G18, L18, RMS i ALGLIB na przykładach całek funkcji f na odcinku $[-1,1]$. Znak „–" oznacza, że algorytm nie potrafił obliczyć danej całki.

The comparison of accuracy and efficiency (time in microseconds) of the adaptive algorithms G18, L18, RMS and ALGLIB in the case of the integrals of the function f over the interval $[-1,1]$. The "–" sign means that the algorithm failed to compute the given integral.

		G18		MATLAB	
f	tol.	błąd (oszac.)	czas	błąd (oszac.)	czas
f_1	10^{-10}	$1{\cdot}10^{-16}$ ($5{\cdot}10^{-16}$)	18	$1{\cdot}10^{-16}$ ($3{\cdot}10^{-17}$)	568
	10^{-14}	$1{\cdot}10^{-16}$ ($5{\cdot}10^{-16}$)	18	$1{\cdot}10^{-16}$ ($3{\cdot}10^{-17}$)	568
f_3	10^{-10}	$9{\cdot}10^{-11}$ ($1{\cdot}10^{-10}$)	17	$1{\cdot}10^{-10}$ ($8{\cdot}10^{-11}$)	1590
	10^{-14}	$9{\cdot}10^{-11}$ ($1{\cdot}10^{-10}$)	17	$8{\cdot}10^{-11}$ ($2{\cdot}10^{-11}$)	16900
f_5	10^{-4}	$2{\cdot}10^{-6}$ ($3{\cdot}10^{-5}$)	72	$2{\cdot}10^{-4}$ ($8{\cdot}10^{-5}$)	863
	10^{-6}	$6{\cdot}10^{-8}$ ($3{\cdot}10^{-7}$)	180	$5{\cdot}10^{-8}$ ($4{\cdot}10^{-7}$)	1650
	10^{-8}	$2{\cdot}10^{-9}$ ($8{\cdot}10^{-9}$)	597	$1{\cdot}10^{-10}$ ($9{\cdot}10^{-9}$)	3750
	10^{-10}	$2{\cdot}10^{-11}$ ($2{\cdot}10^{-10}$)	3950	$9{\cdot}10^{-13}$ ($9{\cdot}10^{-11}$)	16300
	10^{-12}	$2{\cdot}10^{-13}$ ($4{\cdot}10^{-12}$)	25600	$4{\cdot}10^{-15}$ ($9{\cdot}10^{-13}$)	280000
	10^{-14}	$3{\cdot}10^{-15}$ ($8{\cdot}10^{-14}$)	162000	$2{\cdot}10^{-15}$ ($4{\cdot}10^{-13}$)	716000
	0	$2{\cdot}10^{-16}$ ($8{\cdot}10^{-15}$)	754000	$2{\cdot}10^{-15}$ ($4{\cdot}10^{-13}$)	716000
f_8	10^{-10}	$8{\cdot}10^{-8}$ ($8{\cdot}10^{-8}$)	818	$1{\cdot}10^{-12}$ ($5{\cdot}10^{-11}$)	1630
	10^{-14}	$8{\cdot}10^{-8}$ ($8{\cdot}10^{-8}$)	882	$1{\cdot}10^{-8}$ ($1{\cdot}10^{-8}$)	13500
f_{10}	10^{-10}	$4{\cdot}10^{-7}$ ($6{\cdot}10^{-11}$)	2630	$4{\cdot}10^{-7}$ ($6{\cdot}10^{-11}$)	3450
	10^{-14}	$4{\cdot}10^{-7}$ ($1{\cdot}10^{-14}$)	4110	$4{\cdot}10^{-7}$ ($2{\cdot}10^{-14}$)	4607

Tabela 2: Porównanie dokładności i efektywności (czas w mikrosekundach) algorytmów G18 i `quadgk` z systemu *MatLab* na przykładach całek funkcji f na odcinku $[-1, 1]$.

The comparison of accuracy and efficiency (time in microseconds) of the algorithm G18 and the *MatLab* `quadgk` algorithm in the case of the integrals of the function f over the interval $[-1, 1]$.

Literatura (References)

[1] G. Dahlquist i Å. Björck, *Numerical Methods in Scientific Computing, Volume 1*, Society for Industrial Mathematics, 2008.

[2] P. J. Davis i P. Rabinowitz, *Methods of Numerical Integration (2nd edition)*, Academic Press, New York, 1984.

[3] P. Favati, G. Lotti i F. Romani, *Algorithm 691: Improving QUADPACK automatic integration routines*, ACM Trans. Math. Soft **17** (1991), 218–232.

[4] W. Gander i W. Gautschi, *Adaptive Quadrature – Revisited*, BIT **40** (2000), 84–101.

[5] D. Kincaid i W. Cheney, *Analiza numeryczna*, WNT 2006.

[6] L. F. Shampine, *Vectorized Adaptive Quadrature in Matlab*, J. Comput. Appl. Math. **211** (2008), 131–140.

[7] L. F. Shampine, R. C. Allen, Jr. i S. Pruess, *Fundamentals of Numerical Computing*, Wiley, New York, 1997.

[8] `http://alpha.uwb.edu.pl/amicke/globalq.shtml`

[9] `http://www.alglib.net/`

[10] `http://www.codecogs.com/code/maths/calculus/quadrature/adaptive.php`

[11] `http://www.nag.com/`