LOCALLY-CORRECTED MULTIDIMENSIONAL QUADRATURE RULES FOR SINGULAR FUNCTIONS [*]

JOHN STRAIN [†]

Abstract. Accurate numerical integration of singular functions usually requires either adaptivity or product integration. Both interfere with fast summation techniques and thus hamper large-scale computations.

This paper presents a method for computing highly accurate quadrature formulas for singular functions which combine well with fast summation methods. Given the singularity and the N nodes, we first construct weights which integrate smooth functions with order-k accuracy. Then we locally correct a small number of weights near the singularity, to achieve order-k accuracy on singular functions as well. The method is highly efficient and runs in $O(Nk^{2d}+N\log^2 N)$ time and $O(k^{2d}+N)$ space. We derive precise error bounds and time estimates and confirm them with numerical results which demonstrate the accuracy and efficiency of the method in large-scale computations. As part of our implementation, we also construct a new adaptive multidimensional product Gauss quadrature routine with an effective error estimate, and compare it with a standard package.

The approach generalizes to interpolate and differentiate scattered data and to integrate singular functions over curves and surfaces in several dimensions.

Key words. numerical integration, singular integrals, fast algorithms, quadtrees, singular value decomposition, vortex methods, potential theory, interpolation

AMS subject classifications. 65D32, 65D05, 65D30, 65R20

1. Introduction.

Many numerical problems require the evaluation of integrals

$$\int_B f(x)dx, \tag{1.1}$$

where B is a D-dimensional subset of $\mathbf{R}^d$ and f is an integrable function on B. Many methods have been devised for the numerical calculation of such integrals, each useful for certain values of D and d and certain classes of B and f. In the case $d = D = 1$ an extensive literature is summarized in [6], while in $d > 1$ dimensions much recent work is presented in [7, 15].

This paper focuses on the evaluation of (1.1) in the following common situation.

(a) B is a rectangle $[a, b] := [a_1, b_1] \times \ldots \times [a_d, b_d]$.

(b) We are given values $f(x_j)$ of f at N points x_j not of our choosing.

(c) We are given an integrable but singular function $\sigma : B \to \mathbf{R}^s$, which is C^k away from a lower-dimensional subset S of B, and f has the form

$$f(x) = \varphi(x) \cdot \sigma(x) + \psi(x), \tag{1.2}$$

where $\varphi : B \to \mathbf{R}^s$ and $\psi : B \to \mathbf{R}$ are unknown C^k functions on B.

We construct two rules for numerical integration. In §2, we construct a rule W with weights W_j, $1 \leq j \leq N$, which integrates smooth functions accurately:

$$\sum_{j=1}^{N} W_j g(x_j) = \int_B g(x)dx + E_N, \tag{1.3}$$

[*] This research was supported by a NSF Young Investigator Award, by Air Force Office of Scientific Research Grant FDF49620-93-1-0053, and by the Applied Mathematical Sciences Subprogram of the Office of Energy Research, U.S. Department of Energy under Contract DE-AC03-76SF00098.

[†] Department of Mathematics and Lawrence Berkeley Laboratory, University of California, Berkeley, California 94720 (strain@math.berkeley.edu).

where E_N decreases rapidly as $N\to\infty$ if g is smooth enough and the points x_j happen to be distributed appropriately. For example, $E_N = O(N^{-k/d})$ if g is C^k and the points are uniformly distributed on B, where k is the order of accuracy of the rule. The computation of W requires $O(N(k^{2d} + \log^2 N))$ time and $O(k^{2d} + N)$ space. Precise error bounds are proven in §2.3 and numerical examples are given in §2.4.

In §3, we construct a rule w with weights w_j which integrates singular functions of the form (1.2) accurately. The singular rule w has the "local correction" property that $w_j = W_j$ except for a small number of j's, those for which x_j is near the singular set. This property is important in the application of fast algorithms to the efficient evaluation of families of singular integrals. The computation of those w_j's differing from W_j requires $O(k^{3d})$ time. Error bounds are proved in §3.2 and numerical results are given in §3.3.

These general rules are constructed with certain specific classes of applications in mind, including computational fluid dynamics, potential theory and crystal growth. These applications require the application of integral operators

$$u(x) = \int_B K(x,x')\omega(x')dx' \tag{1.4}$$

where K has known singular behavior on a lower-dimensional set but ω is (at least piecewise) smooth. Typically K is singular at a single point, we know $\omega(s_j)$ at N points s_j, and we would like to approximate M values $u(t_i)$ at points $t_i \in \mathbf{R}^d$. We have no control over the location of the s_j and would like to avoid the artificial viscosity produced by interpolating, so we take the s_j's as given.

A classical approach to this problem is product integration [6]. Here we approximate $u(t_i)$ by a rule of the form

$$u_i = \sum_{j=1}^{N} K_{ij}\omega(s_j) \tag{1.5}$$

with K_{ij} chosen to integrate some class of ω exactly for each i. This is a $M\times N$ matrix multiplication, so it costs $O(MN)$ work, which is very expensive when M and N are large. This has been a stumbling block in computational fluid dynamics [5], potential theory for the Laplace equation [14], and crystal growth [23]. Product integration also tends to require difficult, expensive, and sometimes impossible algebraic manipulations and evaluation of integrals in closed form. A major objective of this paper is to eliminate the calculations required by product integration, and replace them with a single general-purpose method which produces locally corrected quadrature rules of arbitrary order for any given singularity.

More recently, fast summation methods have been developed for several kernels K. These methods evaluate the discrete sum

$$u_i = \sum_{j=1}^{N} K(t_i,s_j)W_j\omega(s_j) \qquad 1 \le i \le M \tag{1.6}$$

to accuracy ϵ, in $O((N+M)\log\epsilon)$ work. See [1, 3, 4, 12, 25, 29] for vortex methods and potential theory and [11, 24] for Gaussian kernels. However, these methods cannot be combined with product integration, where the weights depend on the point of evaluation t_i.

Another class of recently-developed fast methods is aimed more directly at the continuous problem (1.4); see [18, 20, 25] for vortex methods and potential theory and [10, 26] for heat potentials. These methods are related to product integration in some cases, usually have a fixed and not too high order of accuracy, and tend to be slower than fast methods for discrete sums (1.6). Like product integration, they sometimes require difficult and expensive algebraic manipulations and evaluation of integrals which can be carried out only in special cases.

Singular quadrature rules of the type developed in this paper allow the application of fast algorithms for discrete sums (1.6) to the continuous problem (1.4), because w_j are independent of the point of evaluation t_i except for a few points near the singularity. Thus fast methods can be applied to the sum (1.6) with weights W_j, and then u_i can be corrected locally to get an accurate and inexpensive approximation of $u(t_i)$. This observation was apparently first made in [19], where one-dimensional singular endpoint-corrected trapezoidal rules were developed. It has been applied to one-dimensional integral equations in [22].

Our method requires knowledge of the singularity $\sigma(x)$ only in the weak sense that we need modified moments

$$\int_{B\cap C} P_\alpha(x)\sigma(x)dx \tag{1.7}$$

over rectangles C, with P_α a suitable family of multidimensional orthogonal polynomials. Obtaining these moments is itself a highly nontrivial task in this generality, with many possibilities depending on the singularity and on B. We have implemented, as part of our method, a general multidimensional adaptive Gaussian quadrature code, with a novel error estimator, which may be of some independent interest and is therefore described in §4. It is sufficient for vortex methods and for volume potentials in potential theory, and hence for the solution of variable-coefficient elliptic partial differential equations, as in [27]. Numerical results in §4 indicate that it is competitive with standard codes in dimensions $d = 2$ and $d = 3$.

The techniques presented in this paper generalize immediately to solve several other problems of considerable interest. We can approximate and differentiate functions known at arbitrary points, a technique which is useful in many computational problems. We can integrate singular functions over more general domains, such as curves and surfaces. Several such generalizations, along with several refinements of the basic method, are discussed in §5.

2. Smooth rules.

2.1. Overview of the construction.

We construct rules with N given points x_j for integrating smooth functions over a d-dimensional rectangle $B = [a, b] := [a_1, b_1] \times \cdots \times [a_d, b_d]$. The structure of these rules will make a good base for the construction of singular rules with locally corrected weights.

Let $k \geq 1$ be the desired order of accuracy of the rule, assume $N \geq m := m(k, d) := (k + d - 1)(k + d - 2)\cdots(k + 1)k/d!$, and choose an integer L with $p := \lfloor N/2^L \rfloor \geq m$. Using a data structure developed below, we divide B into $M = 2^L$ rectangular subcells B_i with disjoint interiors such that B is their union and each B_i contains either p or $p + 1$ points x_j. Then on each B_i, we construct local weights W_i^j for $x_j \in B_i$ which integrate the m monomials of degree $\leq k - 1$ exactly over B_i. (A monomial of degree k in d dimensions has the form $x_1^{\alpha_1} x_2^{\alpha_2} \cdots x_d^{\alpha_d}$, where each α_l is a nonnegative integer and $|\alpha| := \alpha_1 + \alpha_2 + \ldots + \alpha_d = k$. There are precisely m monomials of degree $\leq k - 1$.) Because of the ill-conditioning of the power basis, we construct these weights by solving the following system of m linear equations in at least p unknowns:

$$\sum_{x_j \in B_i} P_\alpha(x_j) W_j^i = \int_{B_i} P_\alpha(x)dx \qquad |\alpha| \le k-1. \tag{2.1}$$

Here

$$P_\alpha(x) = P_{\alpha_1}(x_1) \cdots P_{\alpha_d}(x_d)$$

is a product of one-dimensional Legendre polynomials, with the lth factor scaled and shifted to live on the interval $[a_l, b_l]$. Since $p \ge m$, this system of m equations in at least p unknowns generically has solutions. We compute the solution W_j^i of least Euclidean norm, using the singular value decomposition [9]. §2.3 discusses what to do when no solution exists. The global weights of the rule W are then defined to be $W_j = W_j^i$ for $x_j \in B_i$.

2.2. Details of the construction. We now construct a data structure with two useful features; first, it partitions B into rectangular cells over which we can easily integrate polynomials and second, there are neither too many nor too few of the points x_j in each cell. Too many points makes the singular value decomposition too expensive and produce a less accurate rule because the cell size increases, while too few points makes (2.1) overdetermined so generically no solution exists. When the number of points p is very close to the minimum required m, so (2.1) is barely solvable, the solution tends to have large 1-norm, making it unsuitable for numerical integration. This is similar to the well-known Runge phenomenon encountered in interpolation (where $p = m$) at equidistant points. We found that p of order $2m$ gave excellent results.

Such a "tree structure" can be constructed by recursive subdivision. Let $B = B_1$ be the level-0 root of the tree. Divide B_1 in half by a plane perpendicular to say the l'th coordinate axis, with the dividing plane located so that each half of B_1 contains either $\lfloor N/2 \rfloor$ or $\lfloor N/2 \rfloor + 1$ points. This gives the level-1 cells B_2 and B_3. Repeat this procedure on B_2 and B_3, with the splitting dimension l chosen independently for each cell, to get B_4 through B_7, each containing $\lfloor N/4 \rfloor$ or $\lfloor N/4 \rfloor + 1$ points x_j. Repeating this procedure L times gives $M = 2^L$ cells B_i on the finest level L, numbered from $i = M$ to $i = 2M - 1$, each containing $p = \lfloor N/M \rfloor$ or $p + 1$ points x_j. The union of all the cells on any given level is B. The tree structure is stored by listing the boundaries of each cell $B_i = [a_i, b_i]$ from $i = 1$ to $i = 2M - 1$, a total of $2d \cdot 2M$ numbers, and indexing the points into a list so that the points $x_j \in B_i$ are given by $j = j(s)$ for $s = b(i), \dots, e(i)$ and three integer functions j, b and e. This can be done in $O(N \log N)$, but the simplest method requires sorting the points in each cell before each subdivision, giving a total cost $O(N \log^2 N)$ for the tree construction when an $O(N \log N)$ sorting method such as heapsort [17] is used. Figure 2.1 shows an example of this construction. We note that hierarchical data structures with similar properties – though not this particular one – have been extensively discussed in [21].

The dimension l across which to split a given cell can be selected in several ways. We can split the longest dimension, so that the length-dependent factor in the error bound of §2.3 is reduced as quickly as possible; choose l with

$$b_l - a_l \ge b_j - a_j \qquad \text{for} \qquad 1 \le j \le d.$$

Alternatively, we can choose l to minimize the second moment of the points in the cell. If function values at the nodes are available when the rule is constructed, other choices intended to minimize integration error for that specific function can be devised.

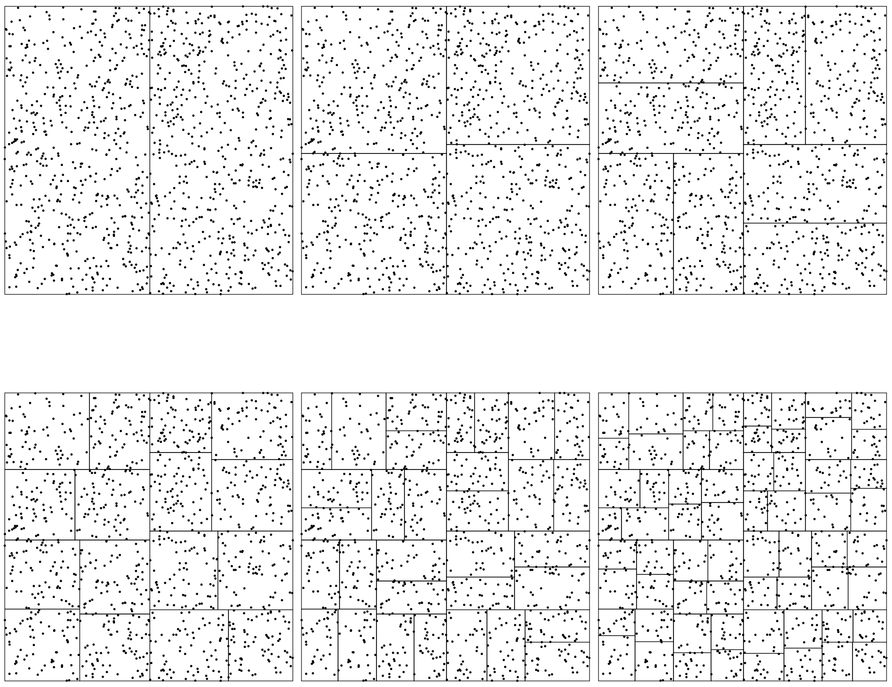

FIG. 2.1. *Levels 1 through 6 in the tree structure with* $N = 1137$ *pseudorandom uniformly distributed points on* $[0,1]^2$.

For future reference, we note that the tree structure permits efficient $O(L)$ lookup of the level-L cell containing any point $x \in B$. We simply begin at the root and discard all children not containing x; the process is then repeated recursively on the remaining child until we reach the lowest level. More generally, we can find all cells intersecting a given rectangle R in time proportional to L and to their number. This will be useful in the construction of singular rules.

2.3. Error bounds. The global weights W_j now integrate all x^α with $|\alpha| \le k-1$ exactly over all level-L cells B_i for $M \le i \le 2M-1$. We now show that this property alone results in order-k accuracy, with a condition number appearing in the error bound.

THEOREM 2.1. *Let* $B = \cup_{i=M}^{2M-1} B_i$ *where* $B_i = [a_i, b_i]$. *Suppose that* W *integrates* x^α *exactly over each* B_i *for* $|\alpha| \le k-1$. *Then for any* C^k *function* g *on* B, *the error*

$$E = \int_B g(x)dx - \sum_{j=1}^{N} W_j g(x_j)$$

satisfies

$$|E| \le \Omega |B| (h/2)^k \sum_{|\alpha|=k} \frac{1}{\alpha!} ||\partial^\alpha g||_{C^0(B)} \tag{2.2}$$

where $h = \max_{i,l} b_{il} - a_{il}$ *is the longest cell edge,*

$$\Omega = 1 + \frac{1}{|B|} \sum_{j=1}^{N} |W_j| \tag{2.3}$$

is the condition number of the rule W, and $|B|$ is the d-dimensional volume of B.

Proof. The error in integrating a smooth function g over B is given by a sum over cells

$$E := \int_B g - \sum_{j=1}^{N} W_j g(x_j) = \sum_{i=M}^{2M-1} E_i, \tag{2.4}$$

where

$$E_i := \int_{B_i} g - \sum_{x_j \in B_i} W_j^i g(x_j). \tag{2.5}$$

Let G be the polynomial of degree $\leq k-1$ which best approximates g on B_i in the maximum norm. Since W is exact for G on B_i, we have

$$E_i = \int_{B_i} (g - G)dx + \sum_{x_j \in B_i} W_j^i (G(x_j) - g(x_j)) \tag{2.6}$$

and thus

$$\begin{aligned} |E_i| &\leq ||g - G||_{L^1(B_i)} + \sum_{x_j \in B_i} |W_j^i|\,|G(x_j) - g(x_j)| \\ &\leq \Omega^i |B_i|\,||g - G||_{C^0(B_i)} \end{aligned}$$

where the local condition number Ω^i is defined by

$$\Omega^i = 1 + \frac{1}{|B_i|} \sum_{x_j \in B_i} |W_j^i|, \tag{2.7}$$

$|B_i|$ is the volume of B_i, and the C^0 norm is defined by

$$||\varphi||_{C^0(B)} := \max_{x \in B} |\varphi(x)| \tag{2.8}$$

for continuous functions φ on a set B.

The error bound on each cell is thus separated into a factor Ω^i independent of the integrand, a factor of $|B_i|$, and a factor which depends only on approximation of the integrand on the cell. The first factor Ω^i cannot be bounded a priori unless all the weights are nonnegative, in which case $\Omega^i = 2$. However, Ω^i can easily be computed a posteriori and thus serves as an extremely useful diagnostic for the quality of the rule.

The volume factor in the error bound depends only on the distribution of points and the tree structure constructed, and will add up to the volume of the domain B.

Finally, we bound the error in approximating g. Assume $g \in C^k(B)$; then by multidimensional Taylor expansion [8], we have

$$g(x + y) = \sum_{|\alpha| \leq k-1} \partial^\alpha g(x) \frac{y^\alpha}{\alpha!} + R_k(x, y) \tag{2.9}$$

where $\alpha! = \alpha_1!\alpha_2! \cdots \alpha_d!$, $\partial^\alpha g = \partial_1^{\alpha_1} \ldots \partial_d^{\alpha_d} g$, $y^\alpha = y_1^{\alpha_1} \ldots y_d^{\alpha_d}$, and the remainder is bounded by

$$\mathbf{R}_k(x, y) \leq \sum_{|\alpha| = k} \frac{(h_i/2)^\alpha}{\alpha!} \max_{0 \leq t \leq 1} |\partial^\alpha g(x + ty)| \tag{2.10}$$

on the cell $B_i = [a_i, b_i]$ with side length $h_i = b_i - a_i$. Since G is the best approximation to g on B_i by a polynomial of degree $\leq k-1$, we have

$$||g - G||_{C^0(B_i)} \leq \sum_{|\alpha|=k} \frac{(h_i/2)^\alpha}{\alpha!} ||\partial^\alpha g||_{C^0(B_i)} \tag{2.11}$$

The global error bound follows immediately:

$$\begin{aligned} |E| &\leq \sum_{i=1}^{M} \Omega^i |B_i| \, ||g-G||_{C^0(B_i)} \\ &\leq \Omega |B| (h/2)^k \sum_{|\alpha|=k} \frac{1}{\alpha!} ||\partial^\alpha g||_{C^0(B)}. \square \end{aligned}$$

$\square$

Note that Ω plays the role of a condition number for W, mediating between the intrinsic difficulty of integrating g (as measured by the derivatives of g) and the accuracy of the final result. There are several ways to reduce each Ω^i and thus obtain a better error bound. Usually taking more points per cell reduces Ω^i, since the additional degrees of freedom are not needed to satisfy (2.1) and can be applied to reducing the 2-norm of W^i_j. However, this increases the cost of computing W considerably and increases the cell size h, so taking larger p is not cost-effective if applied globally.

It can be applied adaptively, however, by going up to a different level of the tree structure when necessary. To implement this, we specify a tolerance Ω_m. When $\Omega^i \geq \Omega_m$, we merge B_i with its sibling in the tree structure, obtaining a cell B_I containing twice as many points x_j. We then recompute all weights W_j for which $x_j \in B_I$, usually obtaining $\Omega^I < \Omega_m$ at the cost of a larger singular value decomposition and a larger cell size h. If Ω^I is still too large, the process may be repeated.

This adaptive technique also permits treatment of the degenerate cases when no solution exists to (2.1) on cell B_i, because the points x_j are not in sufficiently general position. Such a cell can be merged with its sibling, after which a solution is much more likely to exist. The process may be repeated if necessary.

Another approach to reducing the error bound would be to seek the least 1-norm solution of (2.1), which would minimize Ω^i. This 1-norm minimization problem is standard but somewhat more expensive to solve that the 2-norm problem we solve with the singular value decomposition. We found that values of p of order $2m$ usually produce Ω^i within an order of magnitude of the lower bound 2, so we expect little improvement from the 1-norm minimization approach and have not experimented with it.

2.4. Implementation and numerical results. We implemented this method in a portable ANSI Fortran code. The code accepts the order k, the dimension d, the number of levels L, the domain $B = [a, b] \subset \mathbf{R}^d$, and the N user-specified quadrature nodes $x_j \in \mathbf{R}^d$. It returns N weights W_j, the cell structure, the maximum condition number encountered in the singular value decompositions, Ω, the cell size h, and so forth. The numerical results reported here were obtained on a Sun Sparc-2 workstation.

We tested the code by generating $N = 256, 512, \ldots, 16384$ pseudorandom uniformly distributed points in the two-dimensional unit square $[0,1]^2$, computing the weights W with $p = k^2 > m = k(k+1)/2$ for $k = 2, 4, 8, 12$ and 16, and using them to integrate monomials, cosines and Lorentzian functions over $[0,1]^2$. The test integrands are thus the vectors

$$g_1(x) = ((x_1 + x_2)^n : 0 \le n \le 3k)$$

$$g_2(x) = (\cos(n(x_1 - r_1))\cos(n(x_2 - r_2)) : 1 \le n \le 10)$$

$$g_3(x) = \left(\frac{1}{n}(n^{-2} + (x_1 - r_1)^2)^{-1}\frac{1}{n}(n^{-2} + (x_2 - r_2)^2)^{-1} : 1 \le n \le 10\right)$$

with r_i uniformly distributed on $[0,1]$ and k the order of the rule. Note that the family g_1 becomes more difficult as k increases.

Since the N points are randomly generated, we cannot expect a smooth convergence as $N \to \infty$. Hence for each integrand $g_j(x)$, we generated 20 different sets of nodes x_i and computed the minimum, arithmetic mean and maximum of the errors E_j and their base-2 logarithms L_j, and the corresponding standard deviations. The base-2 logarithm makes the order of convergence easier to see: kth order corresponds to L_j decreasing by $k/2$ when N is doubled.

N	h	Ω	T	L_1	L_2	L_3
128	0.3559	2.05	0.10	-10.35	-5.07	-3.03
256	0.2934	2.04	0.15	-11.45	-5.65	-4.13
512	0.1981	2.05	0.32	-13.44	-7.45	-4.40
1024	0.1536	2.04	0.69	-14.07	-8.98	-6.67
2048	0.1095	2.05	1.52	-15.59	-10.20	-7.51
4096	0.0809	2.05	3.28	-17.04	-11.56	-9.13
8192	0.0551	2.05	7.26	-18.94	-12.98	-9.86
16384	0.0428	2.05	15.99	-19.15	-14.91	-11.25

TABLE 2.1

Mesh size h, condition number Ω, CPU time T and average base-2 error logarithms L_j for the second-order smooth rule with N random points.

N	h	Ω	T	L_1	L_2	L_3
128	0.5146	3.22	0.22	-12.67	-4.90	-3.58
256	0.3293	3.51	0.41	-14.54	-7.66	-5.29
512	0.2776	3.60	0.84	-15.78	-9.15	-4.74
1024	0.1764	3.34	1.73	-18.76	-12.09	-7.67
2048	0.1449	3.36	3.57	-19.61	-14.40	-9.93
4096	0.0915	3.42	7.41	-22.30	-17.07	-12.82
8192	0.0752	3.48	15.53	-23.64	-18.85	-14.36
16384	0.0492	3.49	32.52	-26.02	-21.88	-17.27

TABLE 2.2

Mesh size h, condition number Ω, CPU time T and average base-2 error logarithms L_j for the fourth-order smooth rule with N random points.

Tables 2.1 through 2.5 display the averages L_j of the base-2 logarithm of the error E produced when the qth-order smooth rule W is applied to integrate the test functions g_j for $j = 1, 2$ and 3 and k = 2, 4, 8, 12 and 16. Since the number of points doubles in each succeeding row of each table, we expect L_j to decrease by $k/d = k/2$ in each step. This decrease is clearly evident for large N. It tends to occur doubled at alternate lines because only when the number of points increases by $2^d = 4$ does the average spacing h decrease by half.

N	h	Ω	T	L_1	L_2	L_3
128	1.0000	6.20	1.21	-11.95	-5.31	-4.29
256	0.5520	7.35	2.37	-14.60	-8.55	-4.56
512	0.5139	6.01	4.76	-17.92	-12.42	-5.91
1024	0.2893	6.69	9.52	-21.39	-16.62	-9.67
2048	0.2635	6.24	19.16	-24.10	-21.02	-12.25
4096	0.1504	6.15	38.65	-28.57	-25.58	-16.59
8192	0.1350	6.29	77.91	-33.27	-29.16	-17.33
16384	0.0786	6.58	157.08	-36.74	-34.75	-23.32

TABLE 2.3

Mesh size h, condition number Ω, CPU time T and average base-2 error logarithms L_j for the eighth-order smooth rule with N random points.

N	h	Ω	T	L_1	L_2	L_3
256	1.0000	19.48	9.95	-14.34	-9.17	-4.85
512	0.5364	24.37	19.92	-18.62	-15.91	-7.65
1024	0.5073	37.65	39.81	-22.09	-21.00	-9.18
2048	0.2762	27.28	79.75	-26.81	-28.63	-12.96
4096	0.2579	26.22	159.82	-31.52	-33.99	-15.34
8192	0.1431	28.42	320.41	-38.84	-40.07	-21.38
16384	0.1324	26.38	642.06	-43.83	-46.71	-23.67

TABLE 2.4

Mesh size h, condition number Ω, CPU time T and average base-2 error logarithms L_j for the twelfth-order smooth rule with N random points.

The code is extremely efficient. Rules of orders $k = 2$, 4, 8, 12 and 16 with $N = 16384$ nodes require $T = 16$, 33, 157, 642 and 2041 CPU seconds on a Sparc-2 workstation. By comparison, the actual integration of even such simple functions as g_2 and g_3 with the given points and weights takes 3 and 1.5 CPU seconds respectively. Thus an integrand with a substantial degree of complexity will dominate the integration time.

To demonstrate the improvement due to taking p substantially larger than m, we also ran tests with $N = p = m, m+1, \dots, k^2$ for $k = 2$, 4, 8, 12 and 16. Table 2.6 shows some of the results. We see that larger values of p produce dramatic decreases in Ω, especially for higher-order rules.

3. Singular rules.

3.1. Overview of the construction.

We now select and correct certain weights W_j of the smooth rule W, to produce a singular rule w which will integrate singular functions $f(x) = \varphi(x) \cdot \sigma(x) + \psi(x)$ more accurately.

The weights to be corrected are selected by forming a list of cells B_i in the tree structure built for the smooth rule W and correcting all the weights W_j for which x_j lies in some cell on the list. For each cell B_i on the list, we construct w_j for $x_j \in B_i$ by requiring w_j to satisfy the linear system of $(1+s)m$ equations which expresses that $P_\alpha(x)$ and $P_\alpha(x)\sigma(x)$ are integrated exactly for $|\alpha| \le k-1$:

$$\int_{B_i} P_\alpha(x)dx = \sum_{x_j \in B_i} w_j P_\alpha(x_j) \tag{3.1}$$

$$\int_{B_i} P_\alpha(x)\sigma_t(x)dx = \sum_{x_j \in B_i} w_j P_\alpha(x_j)\sigma_t(x_j) \tag{3.2}$$

N	h	Ω	T	L_1	L_2	L_3
256	1.0000	71.16	31.80	-13.39	-9.96	-5.74
512	1.0000	31.93	63.70	-16.26	-16.10	-7.44
1024	0.5210	68.11	127.28	-20.29	-24.86	-11.53
2048	0.5045	53.34	254.55	-26.56	-32.94	-11.66
4096	0.2696	49.57	509.72	-33.16	-42.35	-19.14
8192	0.2576	51.48	1019.63	-39.50	-47.79	-19.69
16384	0.1375	45.07	2041.41	-44.54	-48.48	-29.81

TABLE 2.5

Mesh size h, condition number Ω, CPU time T and average base-2 error logarithms L_j for the sixteenth-order smooth rule with N random points.

$k = 2, N =$	3	4	5	6	7	8
$\Omega =$	34	3.4	2.5	2.2	2.2	2.1
$k = 4, N =$	10	11	12	13	14	16
$\Omega =$	40	10	8.2	5.9	4.6	3.3
$k = 8, N =$	36	40	44	48	56	64
$\Omega =$	6673	85	43	24	13	7.3
$k = 12, N =$	78	84	99	114	129	144
$\Omega =$	4803412	1057	180	46	37	23
$k = 16, N =$	136	144	172	200	228	256
$\Omega =$	313597	16561	923	221	90	66

TABLE 2.6

Average condition number Ω as a function of the number $N = p$ of points per cell for $k = 2$, 4, 8, 12 and 16.

for $|\alpha| \leq k-1$ and $1 \leq t \leq s$. In order for these equations generically to have solutions w, we cannot use the cells B_i on the lowest level L of the tree structure, because each of these contains only $p \geq m$ or $p+1$ points x_j. Instead, we use the cells constructed on a level $L' < L$ of the tree structure, for example with $L' := L - \lceil \log_2(1+s) \rceil$ if p was chosen of order $2m$ to begin with. On level L', we have fewer and larger cells, each containing at least $p' := N/2^{L'} \geq (1+s)m$ points. Thus (3.1) and (3.2) will generically be solvable. In practice, we solve (3.1) and (3.2) by the singular value decomposition, obtaining w as the solution of least 2-norm if it exists. A major new difficulty which requirement (3.2) introduces is the computation of the singular moments

$$\int_{B_i} P_\alpha(x)\sigma(x)dx \tag{3.3}$$

when σ is not smooth. §4 is devoted entirely to this question.

The actual selection of cells to correct can be made in several ways. If σ is singular at a point $x_s \in B$, for example, a natural choice would be simply to correct the cell B_i on level L' which contains x_s. However, x_s might lie at the corner of B_i, so many nearby points would go uncorrected if this selection were made. A variant of single-cell correction is to correct only the neighbor cells of the quadrant of cell B_i where x_s lies. An alternative and natural choice would be to correct all cells intersecting a region of specified size δ around the singular set S; these cells can be found efficiently, as described in §2. However, a fixed size δ requires correction of a number of points proportional to N as $N \to \infty$, which is unacceptably expensive if fast summation methods are employed. Thus this selection scheme is robust but too expensive. Also, it takes no account of local density variations of the points.

We chose to select cells for correction by the following approach. The user specifies a dimensionless correction radius r_c, typically of order unity. We find the cell $B_i =$

$[a, b]$ in which the singularity lies (several cells if σ has a higher-dimensional singular set). We then select for correction all cells intersecting the rectangle $R = [x_s - r_c(b-a)/2, x_s + r_c(b-a)/2]$ of size r_c times B_i and centered at each singular point $x_s \in S$. This scales the size of the corrected area to the local cell size and therefore to the local density of nodes, keeping the number of corrected points per singular point of order unity as $N \to \infty$ with r_c fixed. If $D = \dim S$ then the number of corrected points is $O(N^{D/d})$. We found $r_c = 3$ to give excellent results in practice. The lookup of cells to be corrected costs only $O(L)$ per cell.

Remark: We can construct a locally-corrected product integration rule using the same technique; we simply drop the requirement (3.1) and go up fewer levels in the tree structure. This gives a rule which integrates polynomials times σ accurately, which is enough for many applications. The added generality obtained by requiring (3.1) as well as (3.2) is important when the integrand may be nonsingular (for example when σ happens to vanish at the singularity), and costs little.

Remark: $\sigma(x_s)$ may be infinite or undefined, so we don't want to evaluate f at x_s. If x_s is one of the quadrature points x_j, then we eliminate it from the list of points to be corrected, set $w_j = 0$, and proceed.

3.2. Error bounds.

The error bounds for singular rules can be derived by polynomial approximation, as in the smooth case. No matter how the list of corrected cells is made up, there will be two types of cells to consider; corrected and uncorrected. On the corrected cells, both φ and ψ can be approximated and the remainder estimated as for smooth rules. On the uncorrected cells, the derivatives of the singularity come into play; the key assumption in the error bound is that we correct all cells sufficiently close to the singularity.

For notational convenience, let's renumber the M cells used in the singular rule, so that the first n are corrected and the last $M - n$ are not: thus $B = \cup_{i=1}^{M} B_i$ where each cell B_i contains at least $(1+s)m$ points for $1 \le i \le n$ and at least m points for $n+1 \le i \le M$. Let the sides of B_i be h_{il} for $l = 1, 2, \ldots, d$ and let $h = \max_{i,l} h_{il}$ be the maximum cell edge. Assume that we have weights w_j such that

$$\int_{B_i} x^\alpha dx = \sum_{x_j \in B_i} w_j x_j^\alpha$$

and

$$\int_{B_i} \sigma(x) x^\alpha dx = \sum_{x_j \in B_i} w_j \sigma(x_j) x_j^\alpha$$

for $|\alpha| \le k-1$ and $1 \le i \le n$, while

$$\int_{B_i} x^\alpha dx = \sum_{x_j \in B_i} w_j x_j^\alpha$$

for $|\alpha| \le k-1$ and $n+1 \le i \le M$.

Assume also that the union $\cup_{i=1}^{n} B_i$ of the corrected cells contains the set R_δ of all points within distance δ of the singular set S. For example, we assume $\cup_{i=1}^{n} B_i$ contains the ball $\{y \in \mathbf{R}^d : ||x_s - y|| \le \delta\}$ around each singular point x_s. Finally, assume that σ is C^k outside the singular set S and that its derivatives satisfy a growth condition

$$|\partial^\alpha \sigma(x)| \le C\delta^{-d-|\alpha|} \tag{3.4}$$

for $|\alpha| = 0$ and $|\alpha| = k$, $\delta > 0$ and $x \notin R_\delta$. Here C is a constant and $\delta > 0$ is arbitrary. This assumption is very benign since it does not even guarantee that σ is in $L^1(B)$. It is satisfied by the singularities occurring in potential theory as well as by the Biot-Savart kernel $\sigma = x/||x||^d$.

Starting from these assumptions, we derive a bound for the error

$$E = \int_B \varphi(x) \cdot \sigma(x) + \psi(x) dx - \sum_{j=1}^{N} w_j(\varphi(x_j) \cdot \sigma(x_j) + \psi(x_j)).$$

where φ and ψ are C^k. The nonsingular term in ψ can be bounded exactly as for the smooth rule in §2, giving

$$|\int_B \psi(x)dx - \sum_{j=1}^{N} w_j \psi(x_j)| \le \Omega |B| (h/2)^k \sum_{|\alpha|=k} \frac{1}{\alpha!} ||\partial^\alpha \psi||_{C^0(B)}.$$

For the singular term, we have to consider corrected and uncorrected cells separately. On corrected cells B_i ($1 \le i \le n$), we have a best approximation Φ to φ by a polynomial of degree $k-1$ and the resulting bound

$$\begin{aligned} E_i &= |\int_{B_i} \varphi(x) \cdot \sigma(x) dx - \sum_{x_j \in B_i} w_j \varphi(x_j) \cdot \sigma(x_j)| \\ &\le \Omega^i_\sigma |B_i| \, ||\varphi - \Phi||_{C^0(B_i)} \end{aligned}$$

where the singular condition number Ω^i_σ is defined by

$$\Omega^i_\sigma = 1 + \frac{1}{|B_i|} \sum_{x_j \in B_i} |w_j \sigma(x_j)|.$$

As in §2, the best approximation error $||\varphi - \Phi||_{C^0(B_i)}$ can be bounded by Taylor expansion to get

$$E_i \le \Omega^i_\sigma |B_i| (h/2)^k \sum_{|\alpha|=k} \frac{1}{\alpha!} ||\partial^\alpha \varphi||_{C^0(B_i)}.$$

Note that a priori Ω^i_σ can be infinite, if one of the quadrature nodes happens to coincide with a singular point $x_s \in S$. Thus Ω^i_σ must be computed a posteriori and used as a measure of the quality of the rule. The methods for reducing Ω^i discussed in §2.3 apply to Ω^i_σ as well. In our examples, however, we rarely encountered large values of Ω^i_σ.

Now consider the error due to integrating $\varphi \cdot \sigma$ over an uncorrected cell B_i where $w = W$. From §2.3, we know that the error on cell B_i is bounded by

$$E_i \le \Omega^i |B_i| (h/2)^k \sum_{|\alpha|=k} \frac{1}{\alpha!} ||\partial^\alpha (\varphi \cdot \sigma)||_{C^0(B_i)}.$$

We simplify this bound by separating derivatives of φ and σ, using the standard inequality for Hölder norms proved in [13]:

$$||(\varphi \cdot \sigma)||_{C^k(B_i)} \le C \left(||\sigma||_{C^0(B_i)} ||\varphi||_{C^k(B_i)} + ||\sigma||_{C^k(B_i)} ||\varphi||_{C^0(B_i)} \right).$$

Here the C^k norm is defined by

$$||\varphi||_{C^k(B)} = ||\varphi||_{C^0(B)} + \sum_{|\alpha|=k} ||\partial^\alpha \varphi||_{C^0(B)}$$

for $k > 0$, so

$$E_i \le C\Omega_i |B_i| h^k ||\varphi \cdot \sigma||_{C^k(B_i)}.$$

This separates the bound for E_i into two pieces E_i^1 and E_i^2:

$$\begin{aligned} E_i^1 &= C\Omega_i |B_i| h^k ||\sigma||_{C^0(B_i)} ||\varphi||_{C^k(B_i)} \\ &\le C\Omega_i |B_i| \delta^{-d} h^k ||\varphi||_{C^k(B_i)} \end{aligned}$$

and

$$\begin{aligned} E_i^2 &= C\Omega_i |B_i| h^k ||\sigma||_{C^k(B_i)} ||\varphi||_{C^0(B_i)} \\ &\le C\Omega_i |B_i| \delta^{-d} \left(\frac{h}{\delta}\right)^k ||\varphi||_{C^0(B_i)} \end{aligned}$$

where we have used assumption (3.4).

We now pause momentarily to discuss our strategy for selecting corrected cells B_i. Clearly the choice δ =constant, correcting all cells within a fixed distance from S, produces the simplest error bound. Indeed, if δ is fixed, then the global error E satisfies

$$E \le C\Omega |B| ||\varphi||_{C^k(B)} h^k$$

just as for the smooth rule, with a constant which depends on δ. Unfortunately, in practice we cannot afford to compute the $O(N)$ correction weights within fixed distance δ from S as $N \to \infty$. Thus we give up the simplicity of this error bound.

Instead, we take $\delta = r_c h$ where r_c is fixed, in order to correct fewer points as $N \to \infty$. This complicates both pieces of the error bound in two different ways. First, the factor δ^{-d} seems to cancel the volume factor $|B_i| = O(h^d)$, so naively summing over all $O(N)$ uncorrected cells produces a factor of N in both E_i^1 and E_i^2. Second, the factor δ^{-k} in E_i^2 eliminates the usual $O(h^k)$ error altogether.

We handle the second difficulty by seeking an error bound of a different form from the usual $O(h^k)$. We choose δ so that $(h/\delta)^k \le \epsilon$ where ϵ is a user-specified parameter, usually smaller than h^k over the range of affordable h. Then we seek an error bound of the form $E \le O(\epsilon) + O(h^k)$ where the constant in $O(\epsilon)$ is allowed to depend on derivatives of σ but not on those of φ. The constant in $O(h^k)$, on the other hand, may depend on derivatives of φ as usual, but not on those of σ. Similar error bounds often occur in the design of fast algorithms [4, 25] and are quite useful in practical computations.

Thus we choose $(h/\delta)^k \le \epsilon$ to get

$$E_i \le C\Omega^i |B_i| \delta^{-d} \left(h^k ||\varphi||_{C^k(B_i)} + \epsilon ||\varphi||_{C^0(B_i)} \right)$$

and it remains to deal with the first difficulty, of summing over all $O(N)$ uncorrected cells B_i.

Let $\Omega_m = \max \Omega^i$, and divide the uncorrected cells B_i into $P = O(1/\delta) = O(1/h)$ shells

$$S_p = \{B_i : p\delta \leq d(B_i, S) \leq (p+1)\delta\}$$

where the distance from B_i to S is defined by

$$d(B_i, S) := \min\{||x - x_s|| : x \in B_i, x_s \in S\}.$$

For $B_i \in S_p$, we have the stronger bound

$$E_i \leq C\Omega_m |B_i| (p\delta)^{-d} \left(h^k ||\varphi||_{C^k(B_i)} + \epsilon p^{-k} ||\varphi||_{C^0(B_i)} \right).$$

Thus

$$\begin{aligned} \sum_{i=n+1}^{M} E_i &= \sum_{p=1}^{P} \sum_{B_i \in S_p} E_i \\ &\leq C\Omega_m \left(\sum_{p=1}^{P} p^{-d} \sum_{B_i \in S_p} |B_i| \right) \delta^{-d} h^k ||\varphi||_{C^k(B)} \\ &+ C\Omega_m \left(\sum_{p=1}^{P} p^{-d-k} \sum_{B_i \in S_p} |B_i| \right) \delta^{-d} \epsilon ||\varphi||_{C^0(B)}. \end{aligned} \tag{3.5}$$

The volume of the shell S_p is bounded by $Cp^{d-1}\delta^d$, and since the cell edges are all bounded by $h = O(\delta)$, the sum over i satisfies

$$\sum_{B_i \in S_p} |B_i| \leq C p^{d-1} \delta^d$$

for some constant C. This cancels the factor of δ^{-d}. The first sum over p in (3.5) then diverges logarithmically, giving a factor of $\log P = O(|\log h|)$, and the second is bounded by $\sum_{p=1}^{\infty} p^{-1-k} < \infty$ if $k \geq 1$. Thus

$$\sum_{i=n+1}^{M} E_i \leq C\Omega_m \left(|\log h| h^k ||\varphi||_{C^k(B)} + \epsilon ||\varphi||_{C^0(B)} \right)$$

We see that we suffer for the singularity by a factor $|\log h|$ and a term $\epsilon ||\varphi||_{C^0(B)}$. We conclude that the total error due to uncorrected cells is bounded by

$$E \leq C\Omega_m \left(|\log h| h^k ||\varphi||_{C^k(B)} + \epsilon ||\varphi||_{C^0(B)} \right)$$

whenever $(h/\delta)^k \leq \epsilon$. Our numerical experiments tend to confirm the accuracy of this bound.

Thus we have proved the following theorem:

THEOREM 3.1. *Fix $\epsilon > 0$ and correct the $O(N^{D/d})$ cells intersecting R_δ where $D = \dim S$ and*

$$\delta = r_c h = \epsilon^{-1/k} h.$$

Then the error in integrating $\varphi \cdot \sigma + \psi$ over B with the locally corrected rule w is bounded by

$$|E| \leq C(\Omega + \Omega_\sigma) \left(h^k ||\psi||_{C^k(B)} + |\log h| h^k ||\varphi||_{C^k(B)} + \epsilon ||\varphi||_{C^0(B)} \right).$$

In particular, we need only correct a fixed number of points as $N \to \infty$ if σ has only point singularities.

The absence of a volume factor $|B|$ in this bound is dismaying at first sight but actually natural, because under the weak assumption (3.4) on σ, the integral itself need not scale with $|B|$. If $\sigma(x) = ||x||^{-d}$, for example, then scaling the variables shows that the integral

$$\int_{\delta \leq ||x|| \leq R} \sigma(x) dx = \int_{\epsilon\delta \leq ||x|| \leq \epsilon R} \sigma(x) dx$$

for any ϵ. Under stronger growth conditions on σ, for example those satisfied by the Biot-Savart kernel, the error estimate would scale in the same way as the integral.

3.3. Implementation and numerical results . We have implemented these techniques in a portable ANSI Fortran program which constructs the singular weights w_j from the data structure and weights W constructed in §2. The singularity is evaluated by a user-supplied subroutine, and is thus quite general. The dimension and order are also arbitrary user-specified parameters. A routine for evaluating the singular volume moments by the technique of §4 is supplied, but the code is highly modular and the user can freely import routines for evaluating the singular moments if they are available e.g. in closed form. The polynomials $P_\alpha(x)$ can also be replaced by other basis functions if desired. The code contains several other refinements discussed in Section 5.

We have tested the code on several singularities in $d = 2$ and $d = 3$ dimensions. Here we report on the results obtained with $d = 2$ and the Biot-Savart kernel

$$\sigma(x) = \frac{x}{||x||^d}.$$

We ran two sequences of tests. First, we carried out a convergence study with a regular grid. We placed $N = 256, 1024, \ldots, 65536$ points in a square grid in $B = [0,1]^2$. For each $k = 2$, 4 and 6, we constructed the smooth rule with these N points and $p = k^2$ points per cell. We then generated 20 random points x_s in B and computed the kth-order correction weights for each singularity $\sigma(x - x_s)$, correcting cells containing $p' = 2k^2$ points and within a correction radius $r_c = 3$ times the cell containing x_s. Tables 3.1 through 3.3 report the averages $L_<$ and $L_>$ of the base-2 logarithms of the errors in using these weights to integrate the singular monomials $(x_1 + x_2)^\alpha \sigma(x - x_r)$ with $0 \leq \alpha \leq k - 1$ for $L_<$ and $k \leq \alpha \leq 3k - 1$ for $L_>$. Note that the error for $\alpha \leq k - 1$ is not zero for two reasons; we compute the singular moments approximately and we only correct nearby cells. We compute the singular moments with the code described in §4, using increasing accuracy as the number of points increased: $\epsilon_a = \epsilon_r = 10^{-3}, 10^{-5}, \ldots, 10^{-11}$ for $N = 256, \ldots, 66536$. The tables also report the average CPU time per correction T, condition numbers Ω and Ω_σ, the maximum cell edge length h, and the number C of corrected points.

The following observations can be made from these results. The convergence rate is somewhat irregular, but roughly accords with theoretical expectations. The use of base-2 logarithms means that $L_<$ and $L_>$ should decrease by k each time N is quadrupled, for the kth-order method. The number of corrected points does not increase with

N. However, the correction is rather expensive due to the general-purpose nature of the code and the necessity of obtaining singular moments by numerical integration. The increase in accuracy of the numerical integration accounts for the increase of T with N. We believe a more efficient and specialized implementation for a specific singularity such as the Biot-Savart kernel could achieve faster run times by orders of magnitude. Finally, we observe that the condition numbers Ω and Ω_σ are bounded by 2 and 3.8 respectively.

N	h	C	T	Ω	Ω_σ	$L_<$	$L_>$
256	0.2500	99	0.62	2.00	3.62	-12.21	-12.57
1024	0.1250	134	1.63	2.00	3.66	-13.15	-13.66
4096	0.0625	134	3.65	2.00	3.69	-15.30	-15.57
16384	0.0312	139	8.18	2.00	3.71	-18.43	-17.87
65536	0.0156	139	18.16	2.00	3.71	-20.73	-19.89

TABLE 3.1

Results of integrating monomials times the Biot-Savart kernel, with second-order singular rules with N regular grid points.

N	h	C	T	Ω	Ω_σ	$L_<$	$L_>$
256	0.5000	256	2.44	2.00	3.64	-13.57	-14.64
1024	0.2500	396	5.09	2.01	3.68	-17.14	-19.54
4096	0.1250	537	9.72	2.00	3.70	-18.40	-20.71
16384	0.0625	537	17.56	2.00	3.71	-20.62	-23.29
65536	0.0312	556	36.72	2.00	3.71	-23.27	-25.75

TABLE 3.2

Results of integrating monomials times the Biot-Savart kernel, with fourth-order singular rules with N regular grid points.

N	h	C	T	Ω	Ω_σ	$L_<$	$L_>$
256	0.5000	256	7.45	2.00	3.69	-13.58	-14.67
1024	0.2500	640	19.06	2.01	3.69	-18.48	-20.18
4096	0.1250	883	31.14	2.01	3.74	-20.40	-24.74
16384	0.0625	1075	49.23	2.00	3.71	-21.53	-26.84
65536	0.0312	1113	81.26	2.00	3.71	-26.02	-31.08

TABLE 3.3

Results of integrating monomials times the Biot-Savart kernel, with sixth-order singular rules with N regular grid points.

Our second sequence of tests used $N = 128, 256, \ldots, 16384$ pseudorandom uniformly distributed points on $B = [0,1]^2$. We repeated the previous tests with these points replacing the grid points, and the results are reported in Tables 3.4 through 3.6. We observe a reasonable convergence rate at first, with $L_<$ eventually levelling off to about 10^{-3}, 10^{-5} and 10^{-7} for the 2nd, 4th and 6th order rules respectively. This is the $O(\epsilon)$ error due to integrating the singularity over the uncorrected cells by the smooth rule W. It appears in $L_<$ and not in $L_>$ because $L_>$ involves higher-order monomials with larger C^k norms, so the $O(h^k)$ term dominates the $O(\epsilon)$ term for the values of N used in these experiments.

4. Singular moments.

N	h	C	T	Ω	Ω_σ	$L_<$	$L_>$
128	0.3410	101	0.68	2.25	4.17	-11.10	-9.24
256	0.2952	145	1.15	2.24	4.32	-10.26	-10.06
512	0.1958	179	1.76	2.25	4.47	-9.26	-10.87
1024	0.1505	188	2.69	2.17	4.22	-10.08	-11.86
2048	0.1081	217	4.10	2.13	4.16	-10.37	-12.43
4096	0.0793	219	5.93	2.08	3.94	-10.94	-12.99
8192	0.0560	264	8.93	2.09	4.23	-11.64	-13.59
16384	0.0426	273	13.17	2.06	3.97	-12.34	-14.83

TABLE 3.4
Results of integrating monomials times the Biot-Savart kernel, with second-order singular rules with N uniformly distributed random points.

N	h	C	T	Ω	Ω_σ	$L_<$	$L_>$
128	0.5136	128	1.65	2.15	3.96	-13.41	-10.84
256	0.3269	256	3.13	2.12	4.14	-17.19	-12.43
512	0.2799	407	5.11	2.41	4.61	-17.11	-14.23
1024	0.1748	535	7.34	2.50	4.65	-16.00	-17.05
2048	0.1451	696	10.93	2.56	4.80	-15.76	-18.68
4096	0.0883	752	14.53	2.78	5.11	-17.20	-20.34
8192	0.0755	1011	21.59	3.10	6.23	-16.34	-20.16
16384	0.0498	957	29.01	3.25	6.50	-17.91	-22.72

TABLE 3.5
Results of integrating monomials times the Biot-Savart kernel, with fourth-order singular rules with N uniformly distributed random points.

4.1. Overview. We now describe the evaluation of the *sm* singular moments

$$\int_{B_i} P_\alpha(x)\sigma_t(x)dx, \qquad |\alpha| \le k-1, 1 \le t \le s. \tag{4.1}$$

We treat (4.1) as a special case of a general problem: Given $f : B \rightarrow \mathbf{R}^n$, smooth away from a lower-dimensional singular set S, evaluate the n-vector of integrals

$$F = \int_B f(x)dx \tag{4.2}$$

We compute (4.2) by a multidimensional adaptive product Gaussian quadrature method, with an error estimate based on Chebyshev differentiation. This is a nonstandard approach to (4.2) in several ways, so we describe it in detail and present numerical results showing that it is more efficient than at least one standard multidimensional adaptive quadrature package.

Our algorithm is organized along the following standard lines. We proceed step by step to refine an approximation $\hat{F}$ to F. At each step, we have a subdivision of B into rectangular cells B_i, an error estimate E_i on each B_i, and an approximation $\hat{F}$ to F formed by integrating over each B_i with product q-point Gauss-Legendre quadrature [6]. We store this information in a heap [28], a data structure which allows us to select the cell B_i with the largest error estimate at each step. We refine $\hat{F}$ by choosing a cell B_i with maximum error estimate, choosing one of the coordinate axes, bisecting B_i along that coordinate axis, and computing the new integrals and error estimates. We then insert the new information into the heap and the next step can proceed. We stop refining when one of the following three situations occurs: we run out of memory, we encounter roundoff error limitations, or we have a total error estimate E satisfying

$$E \le \epsilon_a + \epsilon_r ||\hat{F}||_\infty$$

N	h	C	T	Ω	Ω_σ	$L_<$	$L_>$
128	0.5654	128	4.50	2.15	4.10	-14.65	-10.93
256	0.5212	256	8.51	2.20	4.57	-17.80	-13.57
512	0.3097	512	16.17	2.23	4.52	-19.84	-16.28
1024	0.2656	788	25.21	2.53	5.00	-21.02	-18.70
2048	0.1626	1085	36.05	3.01	5.58	-21.13	-22.77
4096	0.1363	1351	47.72	3.37	6.04	-20.91	-24.65
8192	0.0851	1843	68.95	4.30	8.22	-21.93	-27.15
16384	0.0718	1848	81.79	5.18	10.10	-21.90	-28.67

TABLE 3.6

Results of integrating monomials times the Biot-Savart kernel, with sixth-order singular rules with N uniformly distributed random points.

where ϵ_a and ϵ_r are user-specified absolute and relative error tolerances.

Our method employs the following nonstandard features. First, the use of product Gauss rules rather than nonproduct rules. Since we are interested primarily in $d = 2$- or $d = 3$-dimensional problems, the number q^d of points required by a product Gauss rule of order $2q$ is quite competitive with standard fully symmetric rules. Another advantage of Gauss rules is the arbitrary order of accuracy available: Using e.g. routine GRULE of [6], Gauss points and weights of order $2q$ are readily available for any q. Second, the error estimate we give below requires little additional work and identifies the direction contributing most to the error, the obvious candidate for bisection. The usual technique for selecting a direction to bisect is based on fourth differences and is somewhat unjustified.

4.2. Error estimation. We begin by bounding the maximum (over $1 \le i \le n$) error in product q-point Gauss-Legendre quadrature of $f_i(x)$ over a cell $B = [a, b]$; this will suggest a direction along which to subdivide. Although our estimate is really a bound and not an estimate, it turns out to be sufficiently sharp in practice. The error estimate in one dimension for a single function f reads [6]

$$\begin{aligned} E_q^1(f) &:= \int_a^b f(x)dx - \sum_{i=1}^q w_i f(x_i) \\ &= C_q(b-a)^{2q+1} f^{(2q)}(\xi) \end{aligned}$$

where $\xi \in (a, b)$, w_i and x_i are the weights and nodes for q-point Gauss-Legendre quadrature on $[a, b]$, and the error constant is given by

$$C_q := \frac{(q!)^4}{(2q+1)((2q)!)^3}.$$

In $d > 1$ dimensions, adding and subtracting gives

$$\begin{aligned} E_q^d(f) &:= \int_{a_1}^{b_1} \dots \int_{a_d}^{b_d} f(x_1, \dots, x_d) dx_1 \dots dx_d - \sum_{i_1=1}^q w_{i_1}^1 \dots \sum_{i_d=1}^q w_{i_d}^d f(x_{i_1}^1, \dots, x_{i_d}^d) \\ &= \int_{a_1}^{b_1} E_q^{d-1}[f(x_1, .)] dx_1 + \sum_{i_2=1}^q w_{i_2}^2 \dots \sum_{i_d=1}^q w_{i_d}^d E_q^1[f(., x_{i_2}^2, \dots, x_{i_d}^d)] \end{aligned}$$

Here w_i^j is the ith weight and x_i^j the ith node for Gauss-Legendre quadrature on $[a_j, b_j]$. Thus, by induction on d and the positivity of the weights w_i^j,

$$|E_q^d[f]| \leq C_q|B| \sum_{l=1}^{d} (b_l - a_l)^{2q} ||\partial_l^{2q} f||_{C^0(B)} =: C_q|B| \sum_{l=1}^{d} E_{lq}^d. \tag{4.3}$$

where $|B| = (b_1 - a_1) \cdots (b_d - a_d)$ is the volume of B.

This error bound displays the contribution E_{lq}^d of each dimension to the total error bound; thus we can choose the dimension l where E_{lq}^d is maximum over l as the dimension across which to split a given cell B. This bound is highly practical because only pure derivatives $\partial_l^{2q} f$ are involved; these require only values of f along a single line and are thus much less expensive to compute than mixed derivatives.

In order to approximate this bound, we will need estimates of the quantities E_{lq}^d. We approximate the C^0 norm by a maximum over r randomly chosen points $p^{(1)}, \ldots, p^{(r)}$ distributed in a Latin square [17] in B, and calculate the approximation

$$D_{lq}^d := (b_l - a_l)^{2q} \max_{1 \leq j \leq r} \max_{a_l \leq x_l \leq b_l} |\partial_l^{2q} f(p_1^{(j)}, \ldots, x_l, \ldots, p_d^{(j)})|$$

by Chebyshev differentiation. Fix j and l and let

$$g(s) = f(p_1^{(j)}, \ldots, c_l + hs_l, \ldots, p_d^{(j)}),$$

where $c_l = (a_l + b_l)/2$ and $h_l = (b_l - a_l)/2$. Then

$$\partial_s^{2q} g(s) = h^{2q} \partial_l^{2q} f(p_1^{(j)}, \ldots, c + hs, \ldots, p_d^{(j)}),$$

so

$$E_{lq}^d = 2^{2q} \max_{1 \leq j \leq r} ||\partial_s^{2q} g||_{C^0}$$

We approximate the $2q$th derivative of g by Chebyshev differentiation. Approximate g by a t-term Chebyshev series

$$g(s) \approx \frac{1}{2} g_1 + \sum_{k=2}^{t} g_k T_{k-1}(s),$$

where the coefficients g_k are computed by pth-order Chebyshev quadrature with $p \geq t + 2$;

$$\begin{aligned} g_k &= \frac{2}{p} \sum_{l=1}^{p} g(t_l) T_{k-1}(t_l) \\ &= \frac{2}{p} \sum_{l=1}^{p} g(\cos(\frac{\pi(l - 1/2)}{p})) \cos(\frac{\pi(k - 1/2)(l - 1/2)}{p}). \end{aligned} \tag{4.4}$$

The jth derivative of g is approximated by

$$g^{(j)}(s) \approx \frac{1}{2} g_1^{(j)} + \sum_{k=2}^{t-j} g_k^{(j)} T_{k-1}(s), \tag{4.5}$$

where the coefficients $g_k^{(j)}$ are determined by backward recurrence

$$\begin{aligned} g_k^{(0)} &= g_k & 1 \le k \le t, \\ g_{k-1}^{(j)} &= g_{k+1}^{(j)} + 2(k-1)g_k^{(j-1)} & t-j \ge k \ge 2q+1-j, \\ g_{t-j+1}^{(j)} &= g_{t-j+2}^{(j)} = 0. \end{aligned} \tag{4.6}$$

Note that the last two coefficients, $g_{t-j-1}^{(j)}$ and $g_{t-j}^{(j)}$, can be explicitly evaluated in terms of g_{t-1} and g_t alone. Similar though more complicated expressions exist for the lower coefficients, but it is easier to evaluate them by recurrence (4.6) even if we only want the top two.

Finally, the fact that $|T_k(s)| \le 1$ for $|s| \le 1$ allows us to bound $g^{(2q)}$:

$$||g^{(2q)}||_{C^0} \le \frac{1}{2}|g_1^{(2q)}| + \sum_{k=2}^{t-2q} |g_k^{(2q)}|. \tag{4.7}$$

Note that we need only compute the coefficients g_k with $2q+1 \le k \le t$; lower-order polynomials drop out after taking $2q$ derivatives.

For efficiency of implementation, however, we do not employ recurrence (4.6) and formulas (4.4) directly. Instead, we observe that in the final estimate (4.7) each $g_k^{(2q)}$ is a linear functional of the p-vector f with components $f_l = f(p_1^{(j)}, \dots, c+hs_l, \dots, p_d^{(j)} : 1 \le j \le p)$. Thus there is a $(t-2q) \times p$ matrix e_{kl} such that

$$g_k^{(2q)} = \sum_{l=1}^{p} e_{kl} f_l \qquad 1 \le k \le t-2q.$$

We simply precompute this matrix, which depends only on p, t and q, and store it. Then each error estimate E_{lq}^d requires only p function evaluations, $(t-2q)p$ multiplications and additions. At minimum, $p = t = 2q+2$, so each error estimate costs $2(2q+2)$ multiplications and $2q+2$ function evaluations. Thus the total error estimate on B_i requires $rd(2q+2)$ function evaluations. Since the integral requires q^d function evaluations, the error estimate is not expensive if $2rd \le q^{d-1}$. It also has the advantage that the points of evaluation for the integral and the error estimate are completely different (and random for the error), reducing the chance of missing cells with large errors.

4.3. Refinements.

The quadrature scheme outlined above is robust and flexible. We found, however, that its efficiency and accuracy can be improved by several refinements discussed below.

4.3.1. Getting started.

In the scheme above, we start with a single cell B and subdivide as necessary. But when f is known to be singular at some known point x_s, we know that many subdivisions will be necessary. Any integrals and errors computed for a cell which is later refined represent wasted effort. This waste can be reduced by beginning with several cells instead of one, in essence taking advantage of prior knowledge of the singularity location to carry out the first few refinements beforehand. A reasonable way to do this is to divide B into 2^d subcells with one corner of each being x_s, then construct a quadtree with several levels by recursively bisecting each cell touching x_s. Such a subdivision of B can be extremely helpful in reducing the time required to integrate f.

4.3.2. Double-loop integration. A related feature of our method is the independence of the error estimator from the integration rule. An extreme way to use this independence is to compute only error estimates as we subdivide, computing the integrals only when the final cell structure has been completed. This saves all the wasted effort of integrating over cells later to be refined, and this can be very substantial when n is very large. Unfortunately, the use of both absolute and relative error criteria

$$E \leq \epsilon_a + \epsilon_r ||F||$$

makes this impractical since F is involved in the stopping condition. We could use the initial value of F computed over the input cells, but this is likely to be unnecessarily expensive since the value of F is likely to increase substantially as the singularity is resolved. The way out of this dilemma is a double-loop procedure in which we start out with a stopping criterion

$$E \leq \epsilon_a + \epsilon_r G$$

with G set to, say, $100||F||$. When this test is passed, we integrate over the resulting cell structure and set G to the $||F||$ thus obtained. Then we repeat the inner loop with the new stopping criterion. In this way, we can save a large number of unnecessary integrations over cells.

Another situation where the double loop approach is useful is when roundoff error may be important. We maintain an error estimate for each cell separately, as well as a global estimate formed by summing them up. Thus each subdivision requires subtraction of the old error estimate for the subdivided cell and addition of the two new estimates. When the initial error estimate is orders of magnitude larger than the final result, serious roundoff problems occur. A double loop is therefore employed; after termination of the inner loop over cells, we re-sum the integral and error estimates. If the stopping criterion is violated after resumming, we restart immediately from where we left off.

4.3.3. Cautious error estimation. A refinement which is important for accuracy and occurs in most effective quadrature routines is the idea of cautious two-level error estimation (see e.g. [7]). Here we use, in addition to the error estimate E_i computed for the current cell, information about the parent cell. The errors and integrals computed for the parent cell are used separately.

Caution means that we do not believe an error estimate which is much smaller than the parental estimate; thus we replace the new error estimate E_i by $\max(E_i, \epsilon_c E_{old})$ where ϵ_c is a user-specified degree of caution related to the order of accuracy of the rule. Typically $\epsilon_c = 10^{-2}$ is a reasonable choice. The idea of nonzero ϵ_c is to prevent old information from being ignored in later decisions.

The use of two-level error estimates, on the other hand, means that we consider also the change in the integrals produced by the subdivision. Thus we replace E_i by $\max(E_i, \epsilon_d |\Delta F|)$ where ΔF is the maximum change in any integral due to the subdivision. Note that two-level error estimators are incompatible with the double loop procedure proposed above, and the two are therefore offered as mutually exclusive options in our implementation.

4.3.4. Shared singularities. In the special situation we consider here, we are integrating a long vector of $n = sm$ functions simultaneously, where each function has the same singularity structure. The repeated evaluations of all the functions involved in the error estimates is wasteful, so we have implemented a restart facility. We first integrate the singularity $\sigma(x)$ alone, then use the cell structure constructed as a starting point for the integration of the polynomials $P_\alpha(x)\sigma(x)$ as well. Numerical experiments with $k = 2, 4, 6$ and 8 and $q = 2, 3, 4, 6, 8$ and 10 and σ the Biot-Savart kernel (so $d = s = 2$) shows that this can save a factor of five to ten in CPU time. However, they also show that further improvements in the efficiency of obtaining the initial cell structure cannot improve the speed of the code much; indeed, even if the initial cell structure were known a priori, we would only save about one-third of the CPU time. Further speedups can come only from reducing the number of points employed or evaluating the functions faster. Improvement in either area is certainly possible.

4.4. Numerical results. We implemented the multidimensional adaptive product Gaussian scheme above in a portable ANSI Fortran code, with the dimension d as a parameter. Although our aim was primarily robustness and reliability, the resulting code is surprisingly efficient.

We tested the code on three problems of various degrees and types of difficulty, following the probabilistic technique of [16]. In each case, we integrated a family of integrands with randomly placed or randomly oriented singularities and measured the average error and success rate. We used three families of integrands. First, a smooth but oscillatory family of cosines:

$$f_1(x) = (\cos(j(x_1 - x_{r1}))\cos(j(x_2 - x_{r2}))\cdots\cos(j(x_d - x_{rd})) : j = 1, 2, \ldots, 10)$$

Second, skewed exponentials of increasing steepness with discontinuities at angles to the coordinate axes:

$$f_2(x) = (\exp(-j||Ax - x_r||_1) : j = 1, 2, \ldots, 10)$$

where A is a random matrix with entries chosen from a uniform distribution on [0,1] and $||x||_1 = \sum_{i=1}^d |x_i|$ is the Manhattan norm. Finally, $m = 36$ Legendre polynomials on $[0, 1]^2$ times the 2-dimensional Biot-Savart kernel as in moment calculations:

$$f_3(x) = P_\alpha(x)\sigma(x - x_r)$$

with $|\alpha| \leq 7$ and $\sigma(x) = x/||x||^d$. Here x_r is chosen from a uniform distribution on $[0, 1]^d$. In all cases the domain of integration was $[0, 1]^d$ and the dimension was $d = 2$. We ran 100 samples of each family. The results are shown in Tables 4.1 through 4.3 below. For these tables, we used $\epsilon_c = \epsilon_d = 10^{-2}$, $r = 2$, $t = p = 2q + 2$ and $\epsilon := \epsilon_a = \epsilon_r = 10^{-1}, 10^{-2}, \ldots, 10^{-7}$. We report the number of function evaluations N_F, the CPU time T and the error E produced by our code. We found $q = 10$, $q = 3$ and $q = 4$ to be the most efficient rule sizes for f_1, f_2 and f_3 respectively. Figure 4.1 shows the tree-structured subdivisions constructed with $\epsilon_a = 10^{-3}, 10^{-5}$ and 10^{-7} for f_2 and f_3. It is clear that the code is refining in the right places.

For comparison, Tables 4.4 through 4.6 show the corresponding results for the multidimensional adaptive fully symmetric quadrature routine DCUHRE presented in [2]. The following conclusions can be drawn from this comparison.

First, in the integration of the Biot-Savart kernel times polynomials, DCUHRE achieved most efficient results with the 13th order rule, because the kernel is smooth away from the singularity. It required 48 CPU sec. with $\epsilon = 10^{-7}$. The errors were very reliably less than the estimate, and in fact very close to the estimate. Gaussian quadrature, on the other hand, was most efficient with a 4-point 8th-order rule when $\epsilon = 10^{-7}$. It required 11 CPU sec. with $\epsilon = 10^{-7}$, about four times faster than DCUHRE. The errors from our Chebyshev error estimator were less reliable in the sense that they were sometimes much less than the estimate and sometimes slightly more.

On cosines, high-order rules were the most effective. For example, 20th order Gaussian quadrature required 0.04 CPU sec. to achieve precision 10^{-7}. DCUHRE required 0.17 CPU sec. with the 13th order rule.

For skew exponentials, which are C^0 but are not C^1 along the randomly oriented hyperplanes determined by A and x_r, the 9th order rule of DCUHRE was more efficient than 13th or 7th. This is a little surprising, because the 7th order rule is recommended by its authors for problems —like this one— requiring great adaptivity. The 9th order rule required 72 CPU sec. with $\epsilon = 10^{-7}$ and achieved error 10^{-7} reliably. Gaussian quadrature, on the other hand, got best results with a 6th-order rule, requiring 73 CPU sec. with $\epsilon = 10^{-7}$.

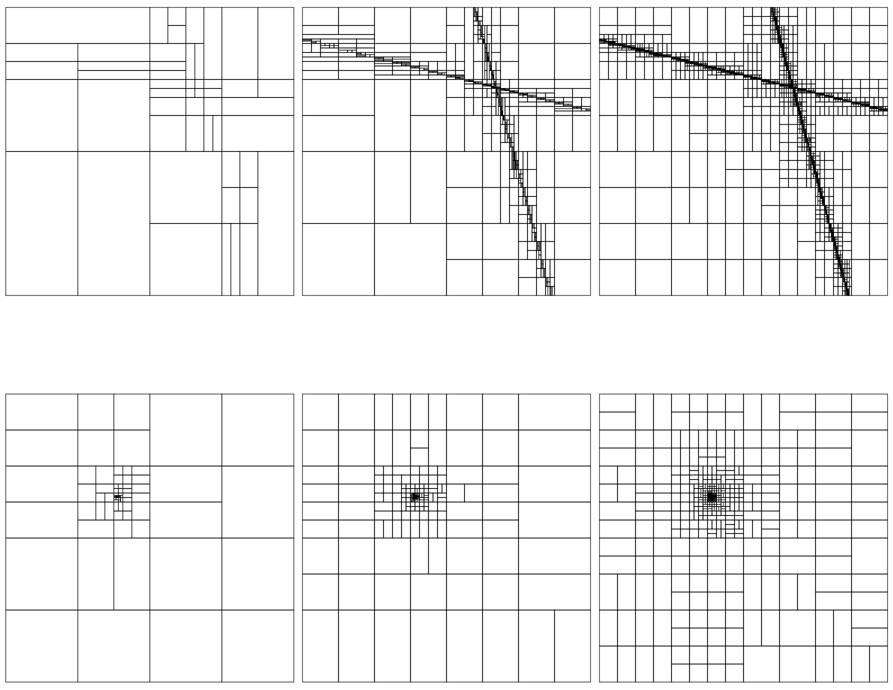

FIG. 4.1. *Tree structure for adaptive Gaussian quadrature.*

$\epsilon_a = \epsilon_r$	N_F	T	E
0.10E+00	188	0.03	0.63E-11
0.10E-01	188	0.03	0.63E-11
0.10E-02	188	0.04	0.63E-11
0.10E-03	188	0.03	0.63E-11
0.10E-04	188	0.04	0.63E-11
0.10E-05	188	0.03	0.63E-11
0.10E-06	188	0.03	0.63E-11

TABLE 4.1
Twentieth-order Gaussian quadrature on cosines.

$\epsilon_a = \epsilon_r$	N_F	T	E
0.10E+00	221	0.03	0.65E+00
0.10E-01	1115	0.13	0.97E-02
0.10E-02	4788	0.55	0.65E-03
0.10E-03	17736	2.01	0.29E-04
0.10E-04	59655	6.71	0.24E-05
0.10E-05	201884	22.62	0.69E-06
0.10E-06	651924	72.97	0.86E-07

TABLE 4.2
Sixth-order Gaussian quadrature on skewed exponentials.

$\epsilon_a = \epsilon_r$	N_F	T	E
0.10E+00	321	0.28	0.12E+01
0.10E-01	771	0.67	0.28E-01
0.10E-02	1536	1.33	0.11E-02
0.10E-03	2793	2.43	0.27E-03
0.10E-04	4649	4.04	0.66E-05
0.10E-05	7638	6.65	0.21E-05
0.10E-06	12651	11.01	0.33E-06

TABLE 4.3
Eighth-order Gaussian quadrature on the Biot-Savart kernel times polynomials.

$\epsilon_a = \epsilon_r$	N_F	T	E
0.10E+00	195	0.04	0.11E-03
0.10E-01	195	0.04	0.11E-03
0.10E-02	195	0.04	0.11E-03
0.10E-03	286	0.05	0.43E-04
0.10E-04	442	0.08	0.50E-05
0.10E-05	793	0.14	0.65E-06
0.10E-06	975	0.17	0.45E-07

TABLE 4.4
DCUHRE on cosines.

$\epsilon_a = \epsilon_r$	N_F	T	E
0.10E+00	178	0.02	0.77E-01
0.10E-01	1069	0.12	0.90E-02
0.10E-02	4158	0.44	0.99E-03
0.10E-03	15886	1.65	0.10E-03
0.10E-04	58733	6.10	0.10E-04
0.10E-05	202989	21.12	0.10E-05
0.10E-06	685872	71.72	0.10E-06

TABLE 4.5
DCUHRE on skewed exponentials.

$\epsilon_a = \epsilon_r$	N_F	T	E
0.10E+00	2613	2.05	0.89E-01
0.10E-01	5473	4.30	0.93E-02
0.10E-02	10270	8.06	0.93E-03
0.10E-03	17147	13.50	0.97E-04
0.10E-04	26702	21.03	0.97E-05
0.10E-05	40742	32.12	0.98E-06
0.10E-06	61308	48.37	0.98E-07

TABLE 4.6
DCUHRE on the Biot-Savart kernel times polynomials.

5. Refinements and Generalizations. The above methods for constructing smooth and singular quadrature rules can be refined and generalized in several ways.

The smooth rule can be made adaptive to reduce Ω, and the order can be locally varied to match the smoothness of the integrand. Chebyshev polynomials can replace Legendre polynomials, allowing the use of non-equidistant FFT techniques to speed up the least 2-norm calculations. For that matter, any other set of basis functions can replace Legendre polynomials, yielding rules which are exact for that class of basis functions.

Both singular and smooth rules can be derived for approximating linear functionals other than integration over B. An important example, interpolation, is discussed in detail below. This leads to a different approach to evaluating integrals of singular functions; transfer the integrand values to nice points by interpolation, then use nice rules on the nice points. This eliminates the necessity of computing singular moments for every corrected point.

Both rules can also be used to integrate over more general domains than rectangles, as discussed below. A particularly exciting prospect is the construction of rules for integrating singular functions over curves and surfaces, for the boundary integral solution of partial differential equations. This is of course another special case of the approximation of other linear functionals mentioned in the previous paragraph.

We could equally well construct W_i^j to integrate exactly the k^d monomials $x_1^{\alpha_1} \cdots x_d^{\alpha_d}$ with *product* degree $\max \alpha_l \leq k-1$, rather than integrating the $m(k,d) = O(k^d/d!)$ monomials with standard degree $\alpha_1 + \ldots + \alpha_d \leq k-1$. This choice is a nonstandard one (see [6]), and would have several advantages and disadvantages. The first, and most important, is the improved accuracy of such a rule (see [6]). Rules of product order k have order k in the standard sense, as well, but they tend to have considerably smaller errors than most rules of standard order k. They use more points than the minimum necessary to achieve standard order k by a factor of $d!$, but this is not overwhelmingly expensive in small dimensions like $d = 2$ or $d = 3$. Another reason is that we use product Gaussian quadrature rules to evaluate the moments (see §4), so product order is more convenient. And finally, it is easier to construct a general multidimensional routine in which the dimension d is an input parameter when rules of product order k are constructed, because it is easier to map a rectangle than a simplex onto an interval. Such a rule is more efficient than standard rules in some ways, because we are evaluating all necessary Legendre polynomials $P_{\alpha_l}(x_l)$ with $0 \leq \alpha_l \leq k-1$, so we might as well multiply them together to get the remaining terms. Our experimental implementation, however, revealed that product rules produce slightly larger errors at greater expense, due to increased cell sizes. Hence our final code used rules which integrate exactly monomials of standard degree $\leq k-1$ exclusively.

Another refinement is as follows. The error analysis suggests that it might be computationally useful to have two different orders of accuracy, for the smooth rule and the singular rule. For example, we might construct a 16th-order smooth rule but correct it locally only to 4th order. We have implemented this feature in our current code but our experience is not yet sufficient to indicate its usefulness.

5.1. Scattered data interpolation.

A common problem of computational physics is to construct a globally defined "nice" function which takes given values $u(x_j)$ at given points x_j. The techniques developed above generalize immediately to solve this problem.

The function we construct is a polynomial $p(x)$ on each cell B_i of the tree structure we constructed for the smooth rule. A polynomial p of degree $\leq k-1$ can be represented as a Legendre series

$$p(x) = \sum_{|\alpha| \leq k-1} \hat{p}(\alpha) P_\alpha(x)$$

where P_α is a shifted and scaled Legendre polynomial on $B_i = [a, b]$ and $\hat{p}(\alpha)$ are the Legendre coefficients of p. Each $\hat{p}(\alpha)$ is a linear functional of p, hence can be approximated by

$$\hat{p}(\alpha) = \sum_{x_j \in B_i} w_j(\alpha) p(x_j)$$

where w_j are exact for $p = P_\alpha$, $|\alpha| \leq k - 1$. Thus $w(\alpha) = (w_j(\alpha) : x_j \in B_i)$ can be found as e.g. the least 2-norm solution of

$$\delta_{\alpha\beta} = \sum_{x_j \in B_i} w_j(\alpha) P_\beta(x_j)$$

for $|\alpha| \leq k-1$ and $|\beta| \leq k-1$. The m by p or $p+1$ matrix $(P_\beta(x_j))$ which appears need be subjected to the singular value decomposition only once, and then each α requires only two matrix-vector multiplies and a scaling by the singular values. Thus given the m by p or $p + 1$ matrix $(w_j(\alpha))$, the Legendre coefficients of a nice polynomial interpolating values p_j at points x_j can be computed by matrix multiplication:

$$\hat{p}(\alpha) = \sum_{x_j \in B_i} w_j(\alpha) p_j.$$

Then the Legendre series provides an interpolant to the scattered data p_j.

This local interpolant on B_i is not of course continuous between cells. However, it is likely to be reasonably smooth since w_j solves a least 2-norm problem. It will have order of accuracy $O(h^k)$ where B_i has sides of length $\leq h$ and p_j are values of a C^k function on B. An expansion in other basis functions on each B_i can be constructed in the same way, as can the derivative of scattered data values.

5.2. General B. The techniques developed in §2 and §3 extend to integrate over curves and surfaces in $\mathbf{R}^2$ and $\mathbf{R}^3$. Suppose we want to calculate

$$\int_\Gamma f(x)dx$$

where f is singular at some point x_s which may be in or near the curve or surface Γ. We enclose Γ in a box B and construct the usual tree structure containing the N given points x_j, which may be either in or outside Γ. Now we construct, e.g. for the smooth rule, weights W_j^i satisfying

$$\sum_{x_j \in B_i} W_j^i P_\alpha(x_j) = \int_{\Gamma \cap B_i} P_\alpha(x)dx \tag{5.1}$$

on each cell B_i. The global weights defined to be $W_j = W_j^i$ if $x_j \in B_i$ can be computed by the singular value decomposition if enough points are in B_i and will integrate smooth functions accurately over Γ. The singular rule is produced from the smooth rule in the usual way.

There are two new complications in this approach when Γ is not a rectangle. First, we need the moments

$$\int_{\Gamma \cap B_i} P_\alpha(x)dx$$

of polynomials over $\Gamma \cap B_i$. If Γ is a piecewise linear manifold these moments are exactly computable. In general, however, and certainly when a singular rule is desired, some form of adaptive numerical integration over Γ will be needed. For general Γ this is a difficult problem; we expect approximation by piecewise polynomial Γ and numerical integration as in the finite element method will work, but other techniques may be faster. Note that the Gaussian integration code we have developed in §4 can easily be extended to integrate over polyhedra rather than rectangles, because Gaussian rules can readily be mapped to polyhedra with 2^d vertices in d dimensions. Polyhedra can be subdivided into polyhedra with 2^d vertices, with only the boundary cells being non-rectangular.

Second, the equations (5.1) are more likely to be rank-deficient, in which case no solution W_j^i will exist. If Γ is a plane, for example, then polynomials in variables perpendicular to the plane are superfluous and we cannot integrate them exactly with any W. The singular value decomposition provides a natural treatment of this difficulty; simply ignore all equations which cannot be satisfied. They will not affect the accuracy of the rule W, because W only integrates over Γ in any case.

The accuracy of the rule requires more machinery to analyze. The additional ingredient is extension theorems; we need to extend functions on Γ to smooth functions on B_i without increasing the size of derivatives. That this can be done is proved in e.g. [8]. It follows that a rule constructed in this way will enjoy the same convergence properties as in the case when Γ is a rectangle.

6. Acknowledgements . The author would like to thank Prof. V. Rokhlin for helpful conversations. This research was supported by a NSF Young Investigator Award, by Air Force Office of Scientific Research Grant 92-0165 and by the Applied Mathematical Sciences Subprogram of the Office of Energy Research, U.S. Department of Energy under Contract DE-AC03-76SF00098.

REFERENCES

[1] C. R. Anderson, *A method of local corrections for computing the velocity field due to a collection of vortex blobs*, J. Comput. Phys., 62 (1986), pp. 111–127.

[2] J. Berntsen, T. O. Espelid, and A. Genz, *An adaptive algorithm for the approximate calculation of multiple integrals*, ACM Trans. Math. Softw., 17 (1991), pp. 437–451.

[3] A. Brandt and A. A. Lubrecht, *Multilevel matrix multiplication and fast solution of integral equations*, J. Comput. Phys., 90 (1990), p. 348.

[4] J. Carrier, L. Greengard, and V. Rokhlin, *A fast adaptive multipole method for particle simulations*, SIAM J. Sci. Stat. Comput., 9 (1988), pp. 669–686.

[5] A. J. Chorin, *Numerical study of slightly viscous flow*, J. Fluid Mech., 57 (1973), pp. 785–796.

[6] P. J. Davis and P. Rabinowitz, *Methods of Numerical Integration*, Computer science and applied mathematics, Academic Press, second ed., 1984.

[7] T. O. Espelid and A. Genz, eds., *Numerical integration : recent developments, software, and applications*, Kluwer Academic, Dordrecht; Boston, 1992.

[8] D. Gilbarg and N. S. Trudinger, *Elliptic partial differential equations of second order*, Springer-Verlag, 1983.

[9] G. H. Golub and C. F. van Loan, *Matrix Computations*, Johns Hopkins University Press, Baltimore, second ed., 1989.

[10] L. Greengard and J. Strain, *A fast algorithm for the evaluation of heat potentials*, Comm. Pure Appl. Math., XLIII (1990), pp. 949–963.

[11] ———, *The fast Gauss transform*, SIAM J. Sci. Stat. Comput, 12 (1991), pp. 79–94.
[12] W. Hackbusch and Z. P. Nowak, *On the fast matrix multiplication in the boundary element method by panel clustering*, Numer. Math., 54 (1989), p. 463.
[13] L. Hörmander, *The boundary problems of physical geodesy*, Arch. Rational Mech. Analysis, 62 (1976), pp. 1–52.
[14] M. A. Jaswon and G. T. Symm, *Integral equation methods in potential theory and elastostatics*, Academic Press, 1977.
[15] P. Keast and G. Fairweather, eds., *Numerical integration : recent developments, software, and applications*, Kluwer Academic, Dordrecht; Boston, 1987.
[16] J. N. Lyness and J. J. Kaganove, *A technique for comparing automatic quadrature routines*, Comput. J., 20 (1977), pp. 170–177.
[17] W. H. Press, W. T. Vetterling, B. P. Flannery, and S. A. Teukolsky, *Numerical recipes in* FORTRAN*: the art of scientific computing*, Cambridge University Press, second ed., 1992.
[18] V. Rokhlin, *Rapid solution of integral equations of classical potential theory*, J. Comput. Phys., 60 (1985), pp. 187–207.
[19] ———, *End-point corrected trapezoidal quadrature rules for singular functions*, Research Report YALEU/DCS/RR-441, Yale University Department of Computer Science, November 1985.
[20] G. Russo and J. Strain, *Fast triangulated vortex methods for the 2-D Euler equations*, J. Comput. Phys., 111 (1994), pp. 291–323.
[21] H. Samet, *The design and analysis of spatial data structures*, Addison-Wesley, Reading, Massachusetts, 1990.
[22] H. P. Starr, *Rapid solution of one-dimensional integral and differential equations*, PhD thesis, Yale University Department of Computer Science, 1993.
[23] J. Strain, *A boundary integral approach to unstable solidification*, J. Comput. Phys., 85 (1989), pp. 342–389.
[24] ———, *The fast Gauss transform with variable scales*, SIAM J. Sci. Stat. Comput, 12 (1991), pp. 1131–1139.
[25] ———, *Fast potential theory II: Layer potentials and discrete sums*, J. Comput. Phys., 99 (1992), pp. 251–270.
[26] ———, *Fast adaptive methods for the free-space heat equation*, SIAM J. Sci. Comput., 15 (1994), pp. 185–206.
[27] ———, *Efficient spectrally-accurate solution of variable-coefficient elliptic problems*, Proc. Amer. Math. Soc., 122 (1995), pp. 843–850.
[28] R. E. Tarjan, *Data structures and network algorithms*, CBMS-NSF regional conference series in applied mathematics, no. 44, SIAM, Philadelphia, 1983.
[29] L. van Dommelen and E. A. Rundensteiner, *Fast adaptive summation of point forces in the two-dimensional Poisson equation*, J. Comput. Phys., 83 (1989), pp. 126–147.