PROCEEDINGS A

rspa.royalsocietypublishing.org

Research

Article submitted to journal

Subject Areas:
Applied Mathematics, Computational Mathematics

Keywords:
quadrature, surface integral, radial basis function, RBFs, RBF-FD

Author for correspondence:
J. A. Reeger
e-mail: jonah.reeger@afit.edu

THE ROYAL SOCIETY PUBLISHING

Numerical quadrature over smooth, closed surfaces

J. A. Reeger[1], B. Fornberg[2] and M. L. Watts[3]

[1]Captain, United States Air Force. Address for correspondence: Air Force Institute of Technology, Department of Mathematics and Statistics, 2950 Hobson Way, Wright-Patterson Air Force Base, OH 45433-7765 USA.

[2]Address for correspondence: University of Colorado, Department of Applied Mathematics, 526 UCB, Boulder, CO 80309 USA.

[3]First Lieutenant, United States Air Force.

The numerical approximation of definite integrals, or quadrature, often involves the construction of an interpolant of the integrand and its subsequent integration. In the case of one dimension it is natural to rely on polynomial interpolants. However, their extension to two or more dimensions can be costly and unstable. An efficient method for computing surface integrals on the sphere is detailed in the literature (Reeger and Fornberg, *Studies in Applied Mathematics*, 2016). The method uses local radial basis function (RBF) interpolation to reduce computational complexity when generating quadrature weights for any given node set. This article generalizes this method to arbitrary smooth closed surfaces.

1 Introduction

Accurate approximation of partial differential equations (PDEs) and integrals on a surface often requires the construction of node sets featuring spatially varying densities in order to capture rapidly changing features (in the integrand or in the surface curvature). Applications where both spherical and non-spherical surfaces arise include geophysics [1] and mathematical biology (e.g. to model bio-molecules [2] or processes on cell membranes). For Radial Basis Function (RBF) based solution methods in these areas, see for ex. [3–8]. This paper considers an RBF-based method for numerical quadrature over smooth, closed surfaces. This method is well suited for obtaining integrated quantities, such as total energy, average temperature, integrating the function

$f(x, y, z) = 1$ over the surface to obtain surface area, etc.

Earlier literature on computing surface integrals focused largely on the sphere and were based on very specific node sets, using tabulated weights for select values of N (the total number of nodes) [9–11]. More recent literature has borrowed concepts from radial-basis-function-generated finite differences (RBF-FD) applied to spatially variable node sets for spherical quadrature [12]. This RBF-FD motivated technique for spherical quadrature is here generalized to approximating integrals over arbitrary smooth, closed surfaces.

Various integration methods on the sphere appear in, e.g., [11,13–16]. Using interpolation with spherical harmonics, Womersley and Sloan [11] found that certain near-uniform node sets performed much worse than others, and designed special maximal determinant (MD) sets to overcome the issue. The reason for the irregularites (large accuracy differences between similar looking node sets) was subsequently found and resolved [16]. Later quadrature work for the sphere [12] has focused on (i) allowing for local node refinement, and (ii) lowering the cost (from $O(N^3)$ to $O(N \log N)$) for calculating quadrature weights for N nodes.

Some of the methods for numerically computing surface integrals on the sphere have been adapted to other surfaces [17,18]. In [17], two methods are presented for numerical integration on a sphere, which are then adapted to integration over other smooth surfaces homeomorphic to the sphere. Similarly, [18] develops a quadrature method that extends to manifolds that are either homogeneous (including $\mathbb{S}^2$) or diffeomorphic to homogeneous spaces.

Additional types of quadrature for surfaces in $\mathbb{R}^3$ can be found in [10,19,20]. For example, [19] discusses numerical quadrature for piecewise smooth surfaces using polynomial interpolants, while [10] couples Thin-Plate Spline interpolation with Green's integral formula [21] for a meshless cubature in $\mathbb{R}^3$. Quadrature involving a Galerkin discretization of a boundary integral equation with a weakly singular kernel is adapted from electromagnetics to a general framework in [20]. In cases when the nodes (where data is available) are not given by an application, but the quadrature method can choose these freely, some additional opportunities arise. Gaussian quadrature for the sphere is described in [22] and for level set surfaces in (hyper–) rectangles in [23]. This latter reference also cites and summarizes several methods based on approximations with smoothed delta functions (generally of low orders of accuracy) .

A common theme throughout the development of each of these methods is that they are fixed to particular node sets featuring near-uniform density in order to achieve stability. Some of these can achieve high orders of accuracy, even spectral, on the near-uniform data sets, but at the expense of a lengthy, often intractable, process for constructing the node sets themselves. On the other hand, the method introduced in this paper can be applied to node sets that are highly nonuniform. For smooth integrands, this new method achieves a convergence rate of $O(N^{-3.5})$, which corresponds to $O(h^7)$ on nearly uniformly spaced node sets where all points are roughly $h = O(1/\sqrt{N})$ apart. This can be contrasted with the one-dimensional case of nodes spaced h apart, where the classical trapezoidal and Simpson's rules only achieve $O(h^2)$ and $O(h^4)$ convergence rates, respectively. While no results are displayed in this paper for highly nonuniform node sets, the method in [12] (which this method extends from) performs comparably well on even pseudorandom node sets and also on node sets that are strongly concentrated around critical features of the integrand. The steps in the method described here which differ from the one in [12] are impacted little, if at all, by the configuration of nodes.

This article begins with a summary of the RBF-FD motivated method for quadrature over a general smooth, closed surface. This is followed by a discussion of the computational efficiency of the method and demonstrations of the accuracy on several combinations of surfaces and test integrands. Implementations of this method are available at Matlab Central's File Exchange [24] or on Github for Python or Julia implementations (see, [25] or [26], respectively). All of the results presented herein were generated using the Matlab® [27] implementation.

2 A Description of the Method

We wish to approximate the surface integral of the scalar function $f(\mathbf{x})$, $\mathbf{x} \in \mathbb{R}^3$, over a smooth, closed surface $S \subset \mathbb{R}^3$ via

$$\mathcal{I}_S(f) := \iint_S f(\mathbf{x}) dS \approx \sum_{i=1}^{N} w_i f(\mathbf{x}_i)$$

where the node set $\mathcal{S}_N = \{\mathbf{x}_i\}_{i=1}^N \subset S$ is a set of (scattered) data sites located on the surface. Let the surface S be given in either of several forms: (i) as a level surface $h(\mathbf{x}) = 0$, (ii) as an explicit parameterization $\mathbf{x}(u, v)$ for parameters u and v, or (iii) only requiring the surface to be smooth, and include the node set $\mathcal{S}_N$. In each case, the method introduced in this paper requires that the set $\mathcal{S}_N$ and its corresponding triangulation $T = \{t_k\}_{k=1}^K$ are given (or have been calculated, for example with a Delaunay triangulation routine). The set T itself approximates the surface very crudely, but can be related to a set $\mathcal{T} = \{\tau_k\}_{k=1}^K$ of *curved* triangles on the surface such that $\mathcal{T}$ fully covers the surface and no two elements of $\mathcal{T}$ intersect except at, possibly, their edges. The surface integral is then the sum over the curved triangles

$$\mathcal{I}_S(f) = \sum_{k=1}^{K} \iint_{\tau_k} f(\mathbf{x}) dS \tag{2.1}$$

This method can be summarized in five steps similar to those presented in [12]:

(i) For each of the curved triangles in $\mathcal{T}$, find a projection point.
(ii) From the projection point, project a neighborhood of the three vertices of the curved triangle (points in $\mathcal{S}_N$) on S into the plane containing the corresponding flat triangle in T. This neighborhood will include $n - 3$ neighboring nodes.
(iii) Find quadrature weights over the local projected node set for numerical evaluation of the definite integral over the projected central flat planar triangle.
(iv) Convert quadrature weights in each plane to corresponding weights for the surface.
(v) Combine the weights for the individual curved triangles to obtain the full weight set for the surface.

(a) Step 1: Find a Projection Point

In the method presented in [12], for each spherical triangle in $\mathcal{T}$ a neighborhood of the three vertices of the triangle on the surface of the sphere is projected radially from the sphere center onto the plane tangent to the surface of the sphere at the midpoint of the *spherical* triangle. This projection is known as the gnomonic projection and it is a well known map projection technique, used already by the Greek Thales of Miletus in the seventh and sixth centuries B.C. [28]. It projects any geodesic (great-circle arc) as a straight line into a plane tangent to the sphere. In the case of the sphere each neighborhood can be projected from a common projection point–the center of the sphere.

For an arbitrary smooth, closed surface we can no longer use a common projection point for all of the curved triangles. Section (b) presents a formula for the projection that indicates the possibility of a singularity or numerical cancellation if the projection point falls in or near the same plane as the corresponding flat triangle being projected. Further, projection points for adjacent curved triangles must be selected so that the portion of the surface projected onto a flat triangle is accounted for in the integral over 1) only the flat triangle corresponding to the current curved triangle (i.e. t_k when considering τ_k) and 2) at least one flat triangle in (2.1). For these reasons a different approach is taken.

i Defining the "Cutting" Plane

Locating the projection point for each curved triangle in $\mathcal{T}$ begins by defining a unique "cutting" plane for each edge in the triangulation T so that both of the two triangles containing

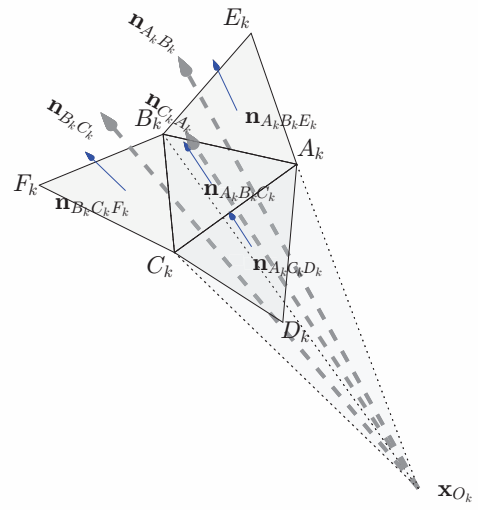

Figure 1: An illustration of a central flat triangle and its three immediate neighbors, all with the normal vectors shown as solid arrows. The three dashed vectors are displayed as originating from the projection point associated with the central flat triangle, and each goes through one side of it. These dashed vectors are defined by $\mathbf{n}_{A_kB_k} = \frac{1}{2}(\mathbf{n}_{A_kB_kC_k} + \text{sign}(\mathbf{n}_{A_kB_kC_k}^T\mathbf{n}_{A_kB_kE_k})\mathbf{n}_{A_kB_kE_k})$, $\mathbf{n}_{B_kC_k} = \frac{1}{2}(\mathbf{n}_{A_kB_kC_k} + \text{sign}(\mathbf{n}_{A_kB_kC_k}^T\mathbf{n}_{B_kC_kF_k})\mathbf{n}_{B_kC_kF_k})$ and $\mathbf{n}_{C_kA_k} = \frac{1}{2}(\mathbf{n}_{A_kB_kC_k} + \text{sign}(\mathbf{n}_{A_kB_kC_k}^T\mathbf{n}_{A_kC_kD_k})\mathbf{n}_{A_kC_kD_k})$.

a given edge will define the same plane. Throughout the remainder of this paper, the subscript k will be used to indicate that the steps are carried out for each triangle separately. When further subscripting is necessary–for instance, to indicate entries of a vector, matrix, or a set–the necessary indexing will follow after a comma.

Consider the flat triangle $t_k = t_{A_kB_kC_k}$ in figure 1. The cutting plane along the edge A_kB_k is defined to contain the edge and to be parallel to the average of the normals $\mathbf{n}_{A_kB_kC_k}$ and $\mathbf{n}_{A_kB_kE_k}$ of $t_{A_kB_kC_k}$ and $t_{A_kB_kE_k}$, respectively, pointing in the same general direction (that is, the angle between them is less than $\frac{\pi}{2}$). This average vector is given by

$$\mathbf{n}_{A_kB_k} = \frac{1}{2}\left(\mathbf{n}_{A_kB_kC_k} + \text{sign}\left(\mathbf{n}_{A_kB_kC_k}^T\mathbf{n}_{A_kB_kE_k}\right)\mathbf{n}_{A_kB_kE_k}\right).$$

The vectors $\mathbf{n}_{B_kC_k}$ and $\mathbf{n}_{C_kA_k}$ can be defined similarly as in figure 1.

ii Locating a Projection Point

Once the cutting planes have been defined, for each flat triangle in T the point, $\mathbf{x}_{O_k}$, of intersection of the three cutting planes is found. The three cutting planes for each of the flat triangles will, in general, intersect at a single point in three dimensional space (with a special case of a point at infinity when $\mathbf{n}_{A_kB_k}$, $\mathbf{n}_{B_kC_k}$, and $\mathbf{n}_{C_kA_k}$ are all three parallel and the projection into the plane becomes the orthogonal projection). This point can be written as, for example,

$$\mathbf{x}_{O_k} = \mathbf{x}_{A_k} + \frac{\mathbf{n}_{O_kB_kC_k}\cdot(\mathbf{x}_{B_k} - \mathbf{x}_{A_k})}{\mathbf{n}_{O_kB_kC_k}\cdot\mathbf{v}_{O_kA_k}}\mathbf{v}_{O_kA_k}.$$

where $\mathbf{n}_{O_kA_kB_k} = \mathbf{n}_{A_kB_k} \times (\mathbf{x}_{B_k} - \mathbf{x}_{A_k})$, $\mathbf{n}_{O_kC_kA_k} = \mathbf{n}_{C_kA_k} \times (\mathbf{x}_{A_k} - \mathbf{x}_{C_k})$, and

$$\mathbf{v}_{O_kA_k} = \mathbf{n}_{O_kA_kB_k} \times \mathbf{n}_{O_kC_kA_k},$$

with $\times$ the vector cross product. An illustration of this point of intersection can be found in figure 1.

(b) Step 2: Project Locally to the Plane Containing the Flat Triangle

Once the point $\mathbf{x}_{O_k}$ is available, points $\mathbf{x}$ in a neighborhood of the flat triangle $t_{A_kB_kC_k}$ (and on the surface S) must be projected into the plane containing $t_{A_kB_kC_k}$ for the interpolation procedure described in section (c). The projection occurs by determining the intersection of this plane and the line through $\mathbf{x}_{O_k}$ and in the direction of $(\mathbf{x}-\mathbf{x}_{O_k})$. Denote the region of S that projects onto $t_{A_kB_kC_k}$ (including its boundary) to be $\tau_{A_kB_kC_k}=\tau_k$. The projection neighborhood containing 12 neighboring vertices is illustrated in the left frame of figure 2.

To reduce the dimension of the interpolation problem a 2-dimensional coordinate system in the plane containing $t_{A_kB_kC_k}$ is defined by first translating the original coordinate system so that $\mathbf{x}_{O_k}$ is at the origin. This translated coordinate system is then rotated so that $\mathbf{n}_{ABC}$ aligns with the vertical axis. This rotation can be realized by multiplication by the matrix (as long as $\sqrt{n_{x_k}^2+n_{y_k}^2}\neq 0$ in which case $R_k=I$)

$$R_k=\begin{bmatrix} \frac{n_{x_k}n_{z_k}}{\sqrt{n_{x_k}^2+n_{y_k}^2}\sqrt{n_{x_k}^2+n_{y_k}^2+n_{z_k}^2}} & \frac{n_{y_k}n_{z_k}}{\sqrt{n_{x_k}^2+n_{y_k}^2}\sqrt{n_{x_k}^2+n_{y_k}^2+n_{z_k}^2}} & \frac{-\sqrt{n_{x_k}^2+n_{y_k}^2}}{\sqrt{n_{x_k}^2+n_{y_k}^2+n_{z_k}^2}} \\ \frac{-n_{y_k}}{\sqrt{n_{x_k}^2+n_{y_k}^2}} & \frac{n_{x_k}}{\sqrt{n_{x_k}^2+n_{y_k}^2}} & 0 \\ \frac{n_{x_k}}{\sqrt{n_{x_k}^2+n_{y_k}^2+n_{z_k}^2}} & \frac{n_{y_k}}{\sqrt{n_{x_k}^2+n_{y_k}^2+n_{z_k}^2}} & \frac{n_{z_k}}{\sqrt{n_{x_k}^2+n_{y_k}^2+n_{z_k}^2}} \end{bmatrix},$$

where n_{x_k}, n_{y_k} and n_{z_k} are the three components of $\mathbf{n}_{A_kB_kC_k}$.

Let $\mathcal{N}_k^n=\{\mathbf{x}_{k,j}\}_{j=1}^n$ be the set containing the n nearest points in $\mathcal{S}_N$ to the flat triangle $t_{A_kB_kC_k}$. After projecting, shifting and rotating the point in $\mathcal{N}_k^n$, their representations in the two-dimensional coordinate system may be centered at a point far from the origin in the two-dimensional space. This can create numerical issues when inverting the RBF interpolation matrix because of the size of the polynomial terms. To remedy this situation one final translation is performed and the coordinates are define as

$$\boldsymbol{\chi}_k=\begin{bmatrix}1&0&0\\0&1&0\end{bmatrix}R_k\frac{1}{\mathbf{n}_{A_kB_kC_k}\cdot(\mathbf{x}-\mathbf{x}_{O_k})}\left(\mathbf{n}_{A_kB_kC_k}\times((\mathbf{x}-\mathbf{x}_{O_k})\times(\mathbf{x}_{M_k}-\mathbf{x}_{O_k}))\right) \quad (2.2)$$

to place the projection of the average of the triangle vertices, $\mathbf{x}_{M_k}=1/3(\mathbf{x}_{A_k}+\mathbf{x}_{B_k}+\mathbf{x}_{C_k})$, at the origin. The two-dimensional coordinates of the points in a neighborhood of the central curved triangle in the left frame of figure 2 are shown in the right frame of the same figure.

(c) Step 3: Find quadrature weights over the local projected node set

For the simplicity of discussion consider approximately evaluating the double integral of a function $g(\boldsymbol{\chi}_k)$ over a planar triangle $t_{A_kB_kC_k}$. Denote this integral by

$$I_{t_{A_kB_kC_k}}(g):=\iint_{t_{A_kB_kC_k}} g(\boldsymbol{\chi}_k)dA \quad (2.3)$$

It can be evaluated approximately by integrating the RBF interpolant of $g(\boldsymbol{\chi}_k)$ with basis functions $\phi\left(\|\boldsymbol{\chi}_k-\boldsymbol{\chi}_{k,j}\|\right)$ centered at the projections of the n points in $\mathcal{N}_k^n$, which are all in a neighborhood of $t_{A_kB_kC_k}$. RBF interpolation has been used successfully in the approximation of differential operators over subsets of scattered data through the concept of RBF-FD [3,6,29,30].

Following common RBF/RBF-FD procedures, let $\{\pi_l(\boldsymbol{\chi}_k)\}_{l=1}^M$, with $M=\frac{(m+1)(m+2)}{2}$ be the set of all of the bivariate polynomial terms up to degree m. The interpolant is constructed as

$$s(\boldsymbol{\chi}_k):=\sum_{j=1}^n c_{k,j}^{\text{RBF}}\phi\left(\|\boldsymbol{\chi}_k-\boldsymbol{\chi}_{k,j}\|\right)+\sum_{l=1}^M c_{k,l}^p\pi_l(\boldsymbol{\chi}_k)$$

where $c_{k,1}^{RBF},\ldots,c_{k,n}^{RBF},c_{k,1}^p,\ldots,c_{k,M}^p\in\mathbb{R}$ are chosen to satisfy the interpolation conditions $s(\boldsymbol{\chi}_{k,j})=g(\boldsymbol{\chi}_{k,j})$, $j=1,2,\ldots,n$, along with constraints $\sum_{j=1}^n c_{k,j}^{RBF}\pi_l(\boldsymbol{\chi}_{k,j})=0$, for $l=1,2,\ldots,M$. By integrating the interpolant the approximation of the integral of g is reduced

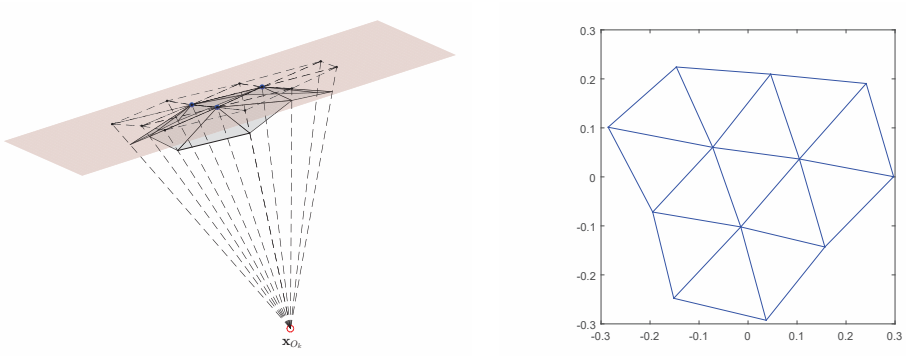

Figure 2: An illustration of the projection of a triangle and its neighbors. Left: Here $\tau_{A_kB_kC_k}$ is the central triangle and $n = 12$ of its nearest neighbors (including the 3 vertices) are projected from $\mathbf{x}_{O_k}$ into the plane containing the vertices of $\tau_{A_kB_kC_k}$. The dashed line segments originating at $\mathbf{x}_{O_k}$ illustrate the projection of each of the nearest neighbors into the plane. Right: The 2D coordinates of the nearest neighbors from the left frame, which are computed from (2.2).

to $I_{t_{A_kB_kC_k}}(g) \approx \sum_{j=1}^{n} w_{k,j}^{RBF} g(\boldsymbol{\chi}_{k,j})$. A simple derivation can be carried out to show that the weights can be found by solving the linear system $\tilde{A}_k W_k = \tilde{I}_k$ with

$$\tilde{A}_k = \begin{bmatrix} A_k^T & P_k \\ P_k^T & 0 \end{bmatrix}, \tilde{I}_k = \begin{bmatrix} I_k^{RBF} \\ I_k^p \end{bmatrix}, \text{ and } W_k = \begin{bmatrix} w_{k,1}^{RBF} \\ \vdots \\ w_{k,n}^{RBF} \\ w_{k,1}^{p} \\ \vdots \\ w_{k,M}^{p} \end{bmatrix},$$

where $A_{k,ij} = \phi\left(\left\|\boldsymbol{\chi}_{k,i} - \boldsymbol{\chi}_{k,j}\right\|\right)$, $P_{k,il} = \pi_l(\boldsymbol{\chi}_{k,i})$, $I_{k,j}^{RBF} = I_{t_{A_kB_kC_k}}\left(\phi\left(\left\|\boldsymbol{\chi}_k - \boldsymbol{\chi}_{k,j}\right\|\right)\right)$, and $I_{k,l}^{p} = I_{t_{A_kB_kC_k}}(\pi_l(\boldsymbol{\chi}_k))$, for $i, j = 1, 2, \ldots, n$ and $l = 1, 2, \ldots, M$ [3, Section 5.1.4]. When a node set comes from an application such as solving a system of PDEs, chances for singularities to occur are for all practical purposes negligible (cf. [31], Theorem 8.21 for precise non-singularity conditions). To guarantee nonsingularity, a node set of high quality does not seem to be needed. However, some care would need to be taken if the nodes form patterns of extreme regularity, such as for satellite obtained data that might be very densely sampled along a just few near-straight paths.

The integrals $I_{k,l}^{p} = I_{t_{A_kB_kC_k}}(\pi_l(\boldsymbol{\chi}_k))$, $l = 1, 2, \ldots, M$, can be evaluated exactly via, for instance, Green's theorem or through the conversion of the integral to barycentric coordinates. Exact evalutations of $I_{k,j}^{RBF} = I_{t_{A_kB_kC_k}}\left(\phi\left(\left\|\boldsymbol{\chi}_k - \boldsymbol{\chi}_{k,j}\right\|\right)\right)$, $j = 1, 2, \ldots, n$, are described in great detail in [12], where the integration over an arbitrary planar triangle is replaced by a combination of integrals over six right triangles (all available analytically). The results presented at the end of this article use the basis function $\phi(r) = r^7$, with $r = \left\|\boldsymbol{\chi}_k - \boldsymbol{\chi}_{k,j}\right\|_2$, where the integral over a right triangle, t, with $\boldsymbol{\chi}_{k,j}$ a vertex located at one of the acute angles has closed form

$$\iint_t r^7 dA = \frac{\alpha\left(105\alpha^8 \sinh^{-1}\left(\frac{\beta}{\alpha}\right) + \beta\sqrt{\alpha^2+\beta^2}\left(279\alpha^6 + 326\alpha^4\beta^2 + 200\alpha^2\beta^4 + 48\beta^6\right)\right)}{3456}.$$

In the preceding expression α is the distance (the base) between $\boldsymbol{\chi}_{k,j}$ and the vertex at the right angle and β is the length (the height) of the opposite side. Such expressions have also been found for many of the most popular choices of RBFs (e.g., $\phi(r) = r^{2k+1}$ or $\phi(r) = r^{2k}\ln r$,

$k = 1, 2, 3, \cdots$, or for Gaussian, multiquadric, or inverse multiquadric basis functions). For a theoretical discussion on the impact of the choice of ϕ on the convergence of the interpolant (which, will have an effect on the performance of this method for computing quadrature weights) see, for example, [32] or [31]. Default choices of the number of nearest neighbors and maximum polynomial order are $n = 80$ and $m = 7$, respectively. These choices were made to balance computation time with accuracy and for comparison to the method in [12], which uses the same parameters. An illustration of this balance appears in figure 4.

(d) Step 4: Convert Quadrature Weights in the Plane to Weights for the Surface Integral

The projections developed in the previous sections amount to changes of variables in the integrals in (2.1) that relate the surface integral to an integral over an area in a plane. Denote the surface normal to S to be $\mathbf{n}_S$. This could be given by

$$\mathbf{n}_S(\mathbf{x}) := \frac{\nabla h(\mathbf{x})}{\|\nabla h(\mathbf{x})\|_2} \text{ or } \mathbf{n}_S(\mathbf{x}) := \frac{\frac{\partial}{\partial u}\mathbf{x}(u,v) \times \frac{\partial}{\partial v}\mathbf{x}(u,v)}{\left\|\frac{\partial}{\partial u}\mathbf{x}(u,v) \times \frac{\partial}{\partial v}\mathbf{x}(u,v)\right\|_2}$$

depending on whether S is described implicitly as a level surface or explicitly through a parameterization, respectively. For simplicity let $\mathbf{n}_{P_k}$ be the unit length vector in the direction of $\mathbf{n}_{A_kB_kC_k}$. Then the surface integral over an individual curved "surface" triangle τ_k is

$$\iint\limits_{\tau_{A_kB_kC_k}} f(\mathbf{x})dS = \iint\limits_{t_{A_kB_kC_k}} f(\mathbf{x}(\boldsymbol{\chi}_k))\frac{\mathbf{n}_{P_k}\cdot(\mathbf{x}(\boldsymbol{\chi}_k)-\mathbf{x}_{O_k})}{\mathbf{n}_S(\mathbf{x}(\boldsymbol{\chi}_k))\cdot(\mathbf{x}(\boldsymbol{\chi}_k)-\mathbf{x}_{O_k})}\left(\frac{\mathbf{n}_{A_kB_kC_k}\cdot(\mathbf{x}(\boldsymbol{\chi}_k)-\mathbf{x}_{O_k})}{\mathbf{n}_{A_kB_kC_k}\cdot(\mathbf{x}_{A_k}-\mathbf{x}_{O_k})}\right)^2 dA. \tag{2.4}$$

To understand the last two terms in (2.4) consider a local parameterization, $\mathbf{x}(\eta,\xi)$, of the points on S in a neighborhood of $t_{A_kB_kC_k}$, and consider the points $\mathbf{x}(\eta,\xi)$, $\mathbf{x}(\eta+\Delta\eta,\xi)$ and $\mathbf{x}(\eta,\xi+\Delta\xi)$. Then

$$\left\|\left(\frac{\mathbf{x}(\eta+\Delta\eta,\xi)-\mathbf{x}(\eta,\xi)}{\Delta\eta}\right)\times\left(\frac{\mathbf{x}(\eta,\xi+\Delta\xi)-\mathbf{x}(\eta,\xi)}{\Delta\xi}\right)\Delta\eta\Delta\xi\right\|_2$$

is the area of a parallelogram in three dimensions with these points as three of the vertices. Taking the limit of this area as $\Delta\eta$ and $\Delta\xi$ approach zero, the infinitesimal surface area element on S is given by

$$dS = \left\|\frac{\partial}{\partial\eta}\mathbf{x}(\eta,\xi)\times\frac{\partial}{\partial\xi}\mathbf{x}(\eta,\xi)\right\|_2 d\eta d\xi,$$

Further, consider the two-dimensional coordinates after projection given by (2.2) of each of the vertices of the parallelogram. The infinitesimal area element in two dimensions with these points as vertices is similarly

$$\begin{aligned} dA &= \left\|\frac{\partial}{\partial\eta}\boldsymbol{\chi}_k(\mathbf{x}(\eta,\xi))\times\frac{\partial}{\partial\xi}\boldsymbol{\chi}_k(\mathbf{x}(\eta,\xi))\right\|_2 d\eta d\xi \\ &= \frac{|\mathbf{n}_{A_kB_kC_k}\cdot(\mathbf{x}_{A_k}-\mathbf{x}_{O_k})|^2}{|\mathbf{n}_{A_kB_kC_k}\cdot(\mathbf{x}(\eta,\xi)-\mathbf{x}_{O_k})|^3}\left|\left(\frac{\partial}{\partial\eta}\mathbf{x}(\eta,\xi)\times\frac{\partial}{\partial\xi}\mathbf{x}(\eta,\xi)\right)\cdot(\mathbf{x}(\eta,\xi)-\mathbf{x}_{O_k})\right|\|\mathbf{n}_{A_kB_kC_k}\|_2, \end{aligned}$$

where the last line follows from (2.2). Noting that $dS = \frac{dS}{dA}dA$ and $\frac{\partial}{\partial\eta}\mathbf{x}(\eta,\xi)\times\frac{\partial}{\partial\xi}\mathbf{x}(\eta,\xi)$ is a vector pointing in the same direction (or opposite) as $\mathbf{n}_S$, the last two terms in (2.4) follow.

In (2.4) the value of $\mathbf{n}_S$ can be evaluated in closed form if such expressions describing the surface are available; however, at the start it was assumed that the surface may be given only as the set of points $\mathcal{S}_N$ on the surface and the triangulation T. In the case where the closed form is not available the surface normal must be approximated (known from solving PDEs on surfaces to be a particularly large source of errors). Notice that after the projection, shifts, and rotation in

section (b) a local "explicit" parameterization of each point $\mathbf{x}$ on the surface is available. That is, each point is parameterized by $\mathbf{x}_k(\boldsymbol{\chi}_k)$ where $\boldsymbol{\chi}_k$ is a point in the two dimensional coordinate system after projection.

For each of the points in $\mathcal{N}_k^n$ the parameterization is known exactly via $\mathbf{x}_{k,j} = \mathbf{x}_k(\boldsymbol{\chi}_{k,j})$. Knowledge of the parameterization at these points allows each component to be approximated through interpolation via

$$\mathbf{x}_k(\boldsymbol{\chi}_k) \approx \mathbf{s}_{\mathbf{x}_k}(\boldsymbol{\chi}_k) := \sum_{j=1}^{n} \mathbf{c}^{\text{RBF}}_{\mathbf{x}_k,j}\, \phi\left(\|\boldsymbol{\chi}_k - \boldsymbol{\chi}_{k,j}\|\right) + \sum_{l=1}^{M} \mathbf{c}^{p}_{\mathbf{x}_k,l}\, \pi_l(\boldsymbol{\chi}_k),$$

where $\mathbf{s}_{\mathbf{x}_k} : \mathbb{R}^2 \mapsto \mathbb{R}^3$. Here the vectors $\mathbf{c}^{RBF}_{\mathbf{x}_k,1}, \ldots, \mathbf{c}^{RBF}_{\mathbf{x}_k,n}, \mathbf{c}^{p}_{\mathbf{x}_k,1}, \ldots, \mathbf{c}^{p}_{\mathbf{x}_k,M} \in \mathbb{R}^3$ are chosen to satisfy the interpolation conditions $\mathbf{s}_{\mathbf{x}_k}(\boldsymbol{\chi}_{k,j}) = \mathbf{x}_k(\boldsymbol{\chi}_{k,j})$, $j = 1, 2, \ldots, n$, along with constraints $\sum_{j=1}^{n} \mathbf{c}^{RBF}_{\mathbf{x}_{k,j}} \pi_l(\boldsymbol{\chi}_{k,j}) = 0$, for $l = 1, 2, \ldots, M$. These vectors of coefficients are determined by solving $\hat{A}_k C_{\mathbf{x}_k} = X'_k$ with

$$\hat{A}_k = \begin{bmatrix} A_k & P_k \\ P_k^T & 0 \end{bmatrix}, X'_k = \begin{bmatrix} \mathbf{x}_k(\boldsymbol{\chi}_{k,1})^T \\ \mathbf{x}_k(\boldsymbol{\chi}_{k,2})^T \\ \vdots \\ \mathbf{x}_k(\boldsymbol{\chi}_{k,n})^T \\ \mathbf{0}_M \end{bmatrix} = \begin{bmatrix} \mathbf{x}_{k,1}^T \\ \mathbf{x}_{k,2}^T \\ \vdots \\ \mathbf{x}_{k,n}^T \\ \mathbf{0}_M \end{bmatrix}, \text{ and } C_{x_k} = \begin{bmatrix} {\mathbf{c}^{RBF}_{x_k,1}}^T \\ \vdots \\ {\mathbf{c}^{RBF}_{x_k,n}}^T \\ {\mathbf{c}^{p}_{x_k,1}}^T \\ \vdots \\ {\mathbf{c}^{RBF}_{x_k,M}}^T \end{bmatrix},$$

where A_k and P_k are as defined in section (c) and $\mathbf{0}_M$ is an $M \times 3$ zero matrix. Since A_k is symmetric, $\hat{A}_k$ in this section equals $\tilde{A}_k$ from section (c) allowing the vector $\mathbf{W}_k$ and the columns of $C_{\mathbf{x}_k}$ to be found efficiently with, for instance, the QR decomposition of $\tilde{A}_k$ in hand. Once the approximation for the explicit parameterization is found an approximation to the normal vector can be computed via the vector cross product between the first partial derivatives of $\mathbf{s}_{\mathbf{x}_k}(\boldsymbol{\chi}_k)$ (with respect to the entries of $\boldsymbol{\chi}_k$)).

Applying step 3 to the double integral (2.4) over a flat triangle $t_k \in T$ gives, for instance

$$\begin{aligned} &\iint_{t_{A_kB_kC_k}} f(\mathbf{x}(\boldsymbol{\chi}_k)) \frac{\mathbf{n}_{P_k} \cdot (\mathbf{x}(\boldsymbol{\chi}_k) - \mathbf{x}_{O_k})}{\mathbf{n}_S(\mathbf{x}(\boldsymbol{\chi}_k)) \cdot (\mathbf{x}(\boldsymbol{\chi}_k) - \mathbf{x}_{O_k})} \left(\frac{\mathbf{n}_{A_kB_kC_k} \cdot (\mathbf{x}(\boldsymbol{\chi}_k) - \mathbf{x}_{O_k})}{\mathbf{n}_{A_kB_kC_k} \cdot (\mathbf{x}_{A_k} - \mathbf{x}_{O_k})} \right)^2 dA \\ &\qquad \approx \sum_{j=1}^{n} w^{\text{RBF}}_{k,j} f(\mathbf{x}_{k,j}) \frac{\mathbf{n}_{P_k} \cdot (\mathbf{x}_{k,j} - \mathbf{x}_{O_k})}{\mathbf{n}_S(\mathbf{x}_{k,j}) \cdot (\mathbf{x}_{k,j} - \mathbf{x}_{O_k})} \left(\frac{\mathbf{n}_{A_kB_kC_k} \cdot (\mathbf{x}_{k,j} - \mathbf{x}_{O_k})}{\mathbf{n}_{A_kB_kC_k} \cdot (\mathbf{x}_{A_k} - \mathbf{x}_{O_k})} \right)^2. \end{aligned} \tag{2.5}$$

(e) Step 5: Combine the Weights Over the Entire Surface

Summing over all of the curved triangles in $\mathcal{T}$ leads to the approximation of the surface integral over S

$$\mathcal{I}_S(f) \approx \sum_{k=1}^{K} \sum_{j=1}^{n} w^{\text{RBF}}_{k,j} f(\mathbf{x}_{k,j}) \frac{\mathbf{n}_{P_k} \cdot (\mathbf{x}_{k,j} - \mathbf{x}_{O_k})}{\mathbf{n}_S(\mathbf{x}_{k,j}) \cdot (\mathbf{x}_{k,j} - \mathbf{x}_{O_k})} \left(\frac{\mathbf{n}_{A_kB_kC_k} \cdot (\mathbf{x}_{k,j} - \mathbf{x}_{O_k})}{\mathbf{n}_{A_kB_kC_k} \cdot (\mathbf{x}_{A_k} - \mathbf{x}_{O_k})} \right)^2. \tag{2.6}$$

Let $\mathcal{K}_i$, $i = 1, 2, \ldots, N$, be the set of all pairs (k, j) such that $\boldsymbol{\chi}_{k,j} \mapsto \mathbf{x}_i$. Then the surface integral over S can be rewritten as

$$\begin{aligned} \mathcal{I}_S(f) &\approx \sum_{i=1}^{N} \left(\sum_{(k,j) \in \mathcal{K}_i} w^{\text{RBF}}_{k,j} \frac{\mathbf{n}_{P_k} \cdot (\mathbf{x}_{k,j} - \mathbf{x}_{O_k})}{\mathbf{n}_S(\mathbf{x}_{k,j}) \cdot (\mathbf{x}_{k,j} - \mathbf{x}_{O_k})} \left(\frac{\mathbf{n}_{A_kB_kC_k} \cdot (\mathbf{x}_{k,j} - \mathbf{x}_{O_k})}{\mathbf{n}_{A_kB_kC_k} \cdot (\mathbf{x}_{A_k} - \mathbf{x}_{O_k})} \right)^2 \right) f(\mathbf{x}_i) \\ &= \sum_{i=1}^{N} w_i f(\mathbf{x}_i). \end{aligned} \tag{2.7}$$

3 Computational Cost

Just as with the method developed in [12], the method introduced in this article requires $O(N)$ time and $O(N)$ memory to construct a set of quadrature weights, even on node sets numbering in the millions. This is illustrated in figure 3. Neglecting the construction of the triangulation of the surface of the sphere discussed in [12], the computation time for that method is provided for comparison to the method introduced in this paper. A slight increase in computation time then comes from the determination of the projection point $\mathbf{x}_{O_k}$ in the case where the exact normal to the surface is known. When the surface normals must be approximated, the cost is greater (but still scales like $O(N)$) because of the additional systems of linear equations that must be solved. For all choices of N, memory use is the same for each of the methods considered.

Since the method considers each curved triangle individually (as in [12]), the method is again embarrassingly parallel. Further, if a GPU is available, the Matlab function pagefun allows all of the linear systems to be solved at near peak GPU speeds.

The ability to parallelize this algorithm warrants that the cost in computation time be further broken down when considering each curved triangle, in turn. The greatest cost when considering each curved triangle is the solution of a linear system or linear systems of size $O((n+m^2)^3)$. In the Matlab implementation, the dense linear system appearing when the normal to the surface is available are solved using the backslash ("\") command. When the normal to the surface must be approximated, the increase in computation time is more significant, since there are three more linear systems of equations that must be solved for each curved triangle. In this case, the Matlab implementation uses the built in qr function (at a cost of $O((n+m^2)^3)$ operations) to construct the decomposition, and then solves each upper triangular linear system using Matlab's linsolve command ($O((n+m^2)^2)$ operations). The other operations performed in the calculation of the quadrature weights for an individual triangle effect the overall cost much less significantly. For instance, evaluation of the integrals of the RBFs and of the polynomial terms in closed form requires much less time than solving the systems of linear equations at approximately $O(n)$ and $O(m^{3.5})$, respectively.

The cost of locating the n nearest neighbors to each triangle should also be mentioned, since it could have a more significant effect on the overall computation time when the triangles are not considered in parallel. The method discussed in [33] requires $O(N\, n \log N)$ operations in the nearest neighbor search using the kd-tree algorithm; however, all computational tests for parallel scalability have not shown this cost to impact the effectiveness of the parallelization.

4 Test Integrands and Results

The present method is demonstrated on a two parameter family of surfaces and on three test integrands featuring varying degrees of regularity. The current default settings for the method are $n = 80$ (number of nearest neighbors) and $m = 7$ (maximum order of bivariate polynomial terms in interpolation). Figure 4 illustrates the effects of different choices of m and n on accuracy and computational cost and suggests that the default parameter choices balance these competing attributes. When using the approximate normal the computational cost is roughly double that of when using the exact normal. Further, an order of magnitude is lost in accuracy when approximating the surface normal, although the convergence rate is unaffected.

Consider the level surfaces defined by

$$h(\mathbf{x}) = h(x,y,z) = (x^2+y^2+z^2)^2 - 2a^2(x^2-y^2-z^2) + a^4 - b^4 = 0, \tag{4.1}$$

which can also be parameterized explicitly via

$$\begin{aligned} x(\theta,\phi) &= \rho(\phi)\cos(\phi) \\ y(\theta,\phi) &= \rho(\phi)\sin(\phi)\sin(\theta) \\ z(\theta,\phi) &= \rho(\phi)\sin(\phi)\cos(\theta) \end{aligned}$$

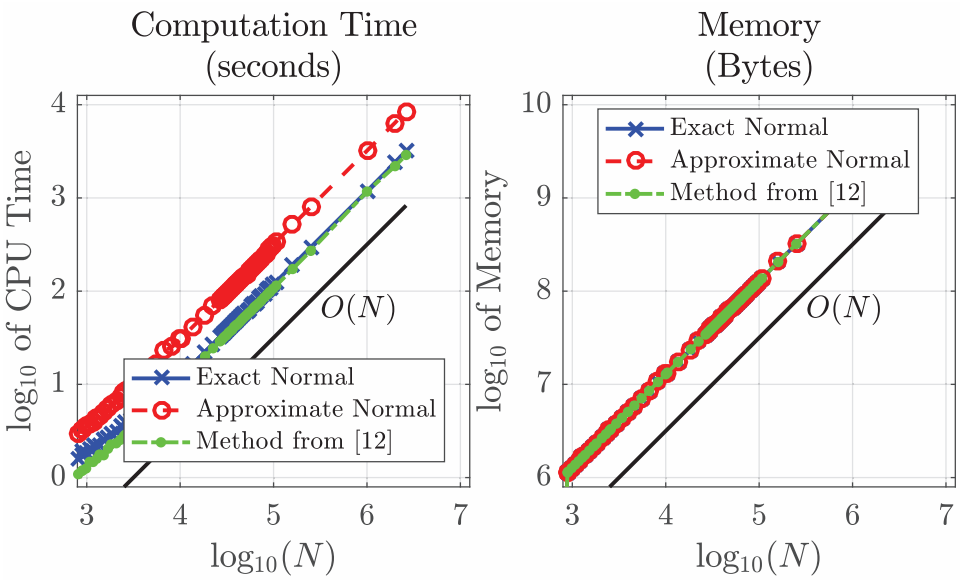

Figure 3: Left: CPU time (in seconds) to compute quadrature weights for evaluting $\mathcal{I}_S(f)$ where f is a scalar function. Right: Required memory (in Bytes) required to compute the quadrature weights. In both frames the method from [12] is only applicable to the sphere. The computations in this figure were performed in Matlab on a laptop with 32 GB of DDR3 1500MHz memory and an Intel Core i7-4900MQ processor featuring four cores at 2.80 GHz.

with $\theta \in [0, 2\pi)$, $\phi \in [0, \pi]$, and $\rho(\phi) = \sqrt{\sqrt{(b^4 - a^4) + a^4\cos^2(2\phi)} + a^2\cos(2\phi)}$. This family is a set of surfaces of revolution generated by rotating the Cassini Oval in the (x, y)-plane about the x-axis. In either case, the surface depends on the two parameters a and b, and the demonstrations here will consider $a = \lambda b$ for $0 < \lambda < 1$ (specifically, $\lambda = 0, 0.8, 0.95$). Notice that in the case of $a = 0$ ($\lambda = 0$) the surface reduces to a sphere of radius b. In the case of $a = b$ ($\lambda = 1$) the surface self intersects, and the slice in the (x, y)-plane forms the Bernoulli lemniscate. Finally, if $a > b$ ($\lambda > 1$) is considered, two separate surfaces are generated. The parameter b in this work is chosen so that the surface area is equal to 1. Since the area of this surface of revolution is not known explicitly, b is chosen by finding the root of

$$R(b) = 1 - 8 \int_0^{\sqrt{a^2+b^2}} \int_0^{\sqrt{\sqrt{b^4+4a^2x^2}-a^2-x^2}} \sqrt{\frac{a^2b^4 - (b^4+4a^2x^2)^{\frac{3}{2}} + 4a^2x^2\sqrt{b^4+4a^2x^2}}{a^2b^4 - (b^4+4a^2x^2)^{\frac{3}{2}} + 4a^2x^4 + 4a^4x^2 + b^4x^2 + b^4y^2 + 4a^2x^2y^2}}\, dy dx$$

with $a = \lambda b$ using Matlab's fsolve command paired with its quad2d function to an absolute and relative tolerance of $O(10^{-15})$. Figure 5 illustrates the three test surfaces. Node sets in all cases were generated using a modified version of "distmeshsurface", a Matlab code made available at [34] and motivated in [35]. This code generates a set of N points (and triangulation) on the surface with nearly uniform spacing between each point and its neighbors (in Euclidean distance in $\mathbb{R}^3$).

Along with the three different test surfaces, demonstrations are performed on three different test integrands. First, recall that the surface S being considered is a surface of revolution about the x-axis, and in the (x, y)-plane $r(x) := y(x) = \sqrt{-a^2 - x^2 + \sqrt{b^4 + 4a^2x^2}}$, which comes from

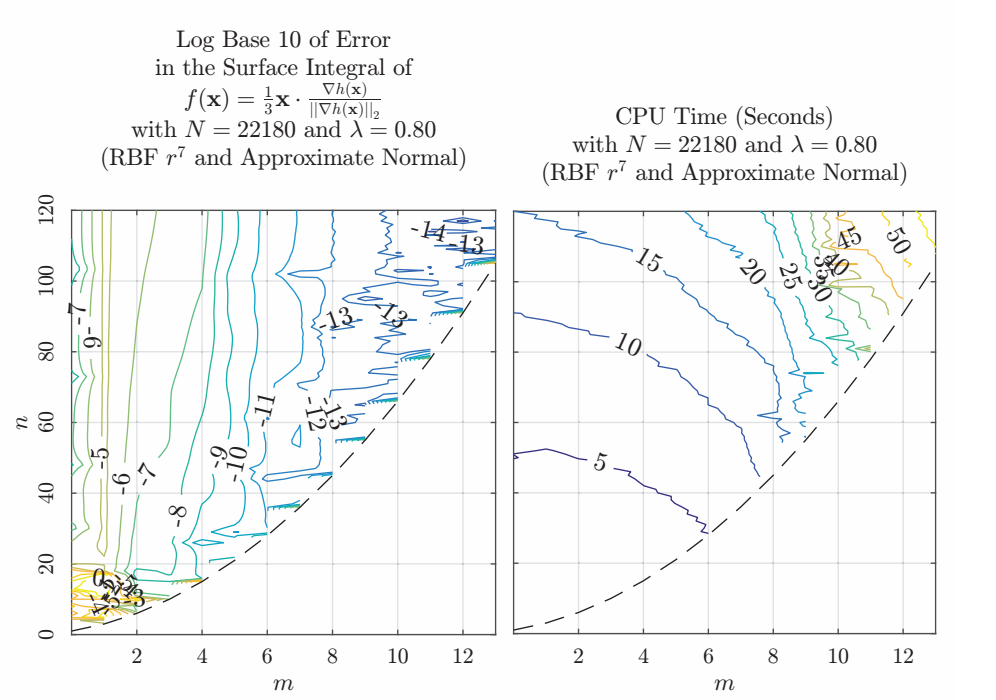

Figure 4: Left: Contour plot of log base 10 of the error when approximating the surface integral of (4.2) over the surface described implicitly by (4.1) when $a = \lambda b$ and b chosen that the surface area equals 1 for various m and n. In this case, the approximate normal is used. Right: Contour plots of computation time in seconds when computing the quadrature weights for various m and n. The dashed parabola in both plots represent the boundary $n = M$ below which the linear system $\tilde{A}_k W_k = \tilde{I}_k$ becomes singular. The vertical axes are the same in the two plots. These computations were performed in Matlab on a workstation with 2 Intel Xeon E5-2697 v3 2.6GHz processors that have 14 cores each (a total of 28 cores) and 256 GB of DDR4 2133MHz memory.

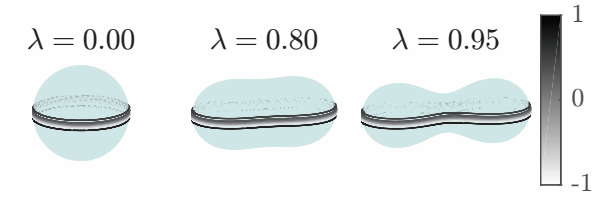

Figure 5: Examples of the surface S defined implicitly by (4.1) when $a = \lambda b$ and b chosen that the surface area equals 1. On each surface ten contours are shown indicating the value of $f_2(\mathbf{x})$ evaluated at the points on the surface.

solving $h(x, y, 0) = 0$. Therefore, if

$$f_1(\mathbf{x}) := \frac{1}{3}\mathbf{x} \cdot \frac{\nabla h(\mathbf{x})}{\|\nabla h(\mathbf{x})\|_2} \tag{4.2}$$

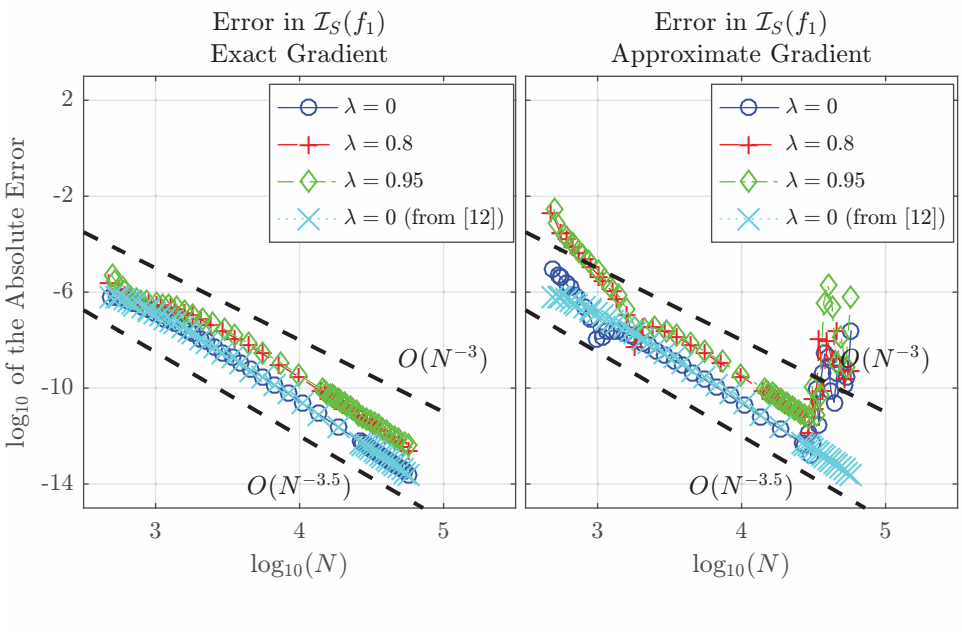

Figure 6: Absolute error in the surface integral of f_1. For each N the error presented here is the maximum over 1000 random rotations of the node set about the x-axis. The label "from [12]" indicates that the weights used were generated using the method from [12]. This curve was included for comparison. The errors presented in the left frame are for the case where the exact normal is available, the right considers when the normal is approximated.

(and $0 < a \leq b$), then the volume contained by the surface (recalling the divergence theorem) is

$$\mathcal{I}_S(f_1) = 2\int_0^{\sqrt{a^2+b^2}} \pi r(x)^2 dx = \frac{\pi}{6a}\left(2a(b^2-2a^2)\sqrt{a^2+b^2} + 3b^4 \sinh^{-1}\left(\frac{2a\sqrt{a^2+b^2}}{b^2}\right)\right).$$

Figure 6 illustrates the error when computing the $\mathcal{I}_S(f_1)$ for various λ. In the case of $\lambda = 0$ two curves are included. The first is the error when using the weights from the method presented in this paper. The second, labeled "from [12]" uses the weights as computed in that paper. It is clear that when considering an integrand that is $C^\infty(S)$ (with no sharp gradients), convergence rates of $O(N^{-3.5})$ or better can be achieved when the nodes in $\mathcal{S}_N$ are nearly uniformly spaced on S. This is true in both of the cases where the exact normal of the surface is known for use in (2.4) and where the normal needs to be approximated.

The only difference between the methods that produced the errors in the left and right frames is that an approximate surface normal, $\mathbf{n}_S$, was used in (2.7) for the construction of the weight sets in the right frame. This approximate surface normal is the result of RBF-based approximations to the partial derivatives of the local parameterization of the surface that is a consequence of the projection presented in section (b). As with other methods of approximating the derivative, the one presented here falls victim to machine rounding errors. That is, the large increases in error as N moves beyond roughly $10^{4.5}$ are a results of working in double precision floating point arithmetic. Computational experiments in extended precision (34 digit quadruple precision) using the Advanpix multiprecision computing toolbox [36] available for Matlab resolve these large increases in error at the expense of significantly greater computation time.

Second, consider $f_2(x,y,z) = \frac{2}{\pi}\tan^{-1}(100z)$, such that $\mathcal{I}_S(f_2) = 0$. This test integrand is represented in figure 5 by ten level curves (contours) evenly spaced between -1 and 1. While

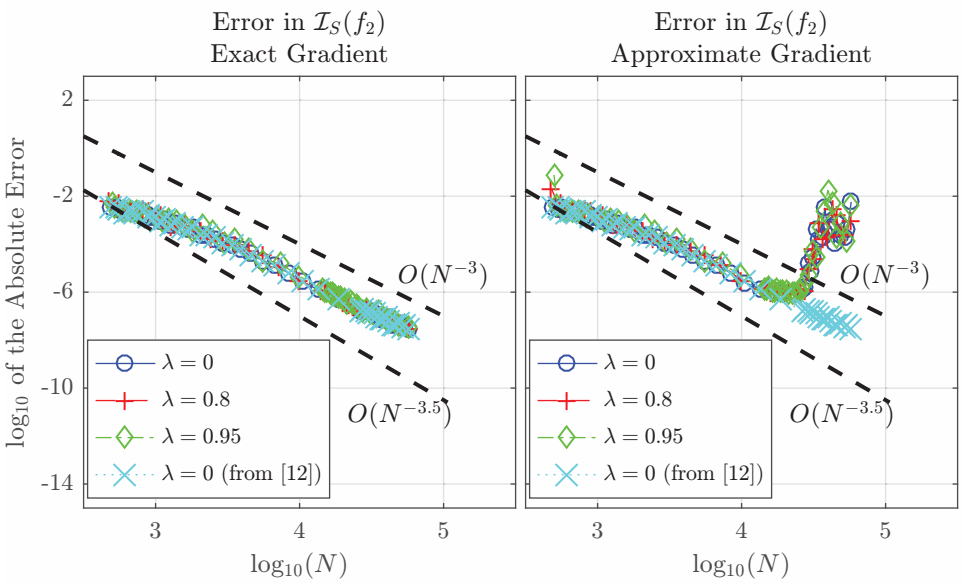

Figure 7: Absolute error in the surface integral of f_2. For each N the error presented here is the maximum over 1000 random rotations of the node set about the x-axis. The label "from [12]" indicates that the weights used were generated using the method from [12]. This curve was included for comparison. The errors presented in the left frame are for the case where the exact normal is available, the right considers when the normal is approximated.

$f_2 \in C^\infty(S)$, the integrand features a sharp gradient where S intersects the plane $z = 0$. Figure 7 displays the absolute error in $\mathcal{I}_S(f_2)$. Notice that, while $f_2 \in C^\infty(S)$, in the presence of the steep gradient the convergence rate is slightly degraded. For N-values beyond the present range, the rate would be expected to again become $O(N^{-3.5})$.

Finally, consider $f_3(x, y, z) = \text{sign}(z)$, where sign represents the signum function. Therefore, f_3 is discontinuous where $z = 0$ intersects S. Figure 8 displays the absolute error in $\mathcal{I}_S(f_3)$. In this case, the integrand is not even continuous, and, as expected, the convergence rate is degraded to less than $O(N^{-1})$.

5 Conclusions

The method presented in this paper allows the computation of quadrature weights for evaluating the surface integral of a scalar function over smooth, closed surfaces. These quadrature weights can be computed in $O(N)$ time and $O(N)$ memory. When applied to $C^\infty(S)$ integrands without sharp gradients, the convergence rate will be roughly $O(N^{-3.5})$. Sharp gradients in the integrand only degrade the convergence marginally. Future work will inlcude tests with still higher N-values, and also consider generalizations to surfaces with boundaries, thereby also allowing improved accuracy in cases of piecewise smooth surfaces and integrands.

Data Accessibility. All presented data can be reproduced with the publicly available codes at Matlab Central's File Exchange [24] or on Github for Python or Julia implementations (see, [25] or [26].)

Authors' Contributions. JAR contributed to the conception and design of the research, acquisition and analysis of the data, and drafting and revising the article; BF contributed to the conception and design of the research and revising the article; MLW contributed to acquisition of the data and drafting the article. All authors gave final approval for publication.

Competing Interests. We have no competing interests.

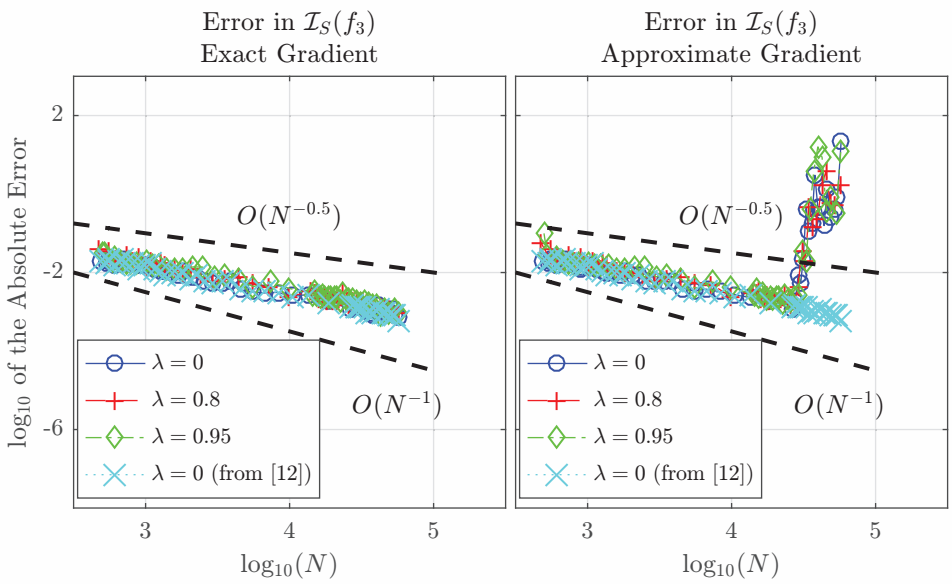

Figure 8: Absolute error in the surface integral of f_3. For each N the error presented here is the maximum over 1000 random rotations of the node set about the x-axis. The label "from [12]" indicates that the weights used were generated using the method from [12]. This curve was included for comparison. The errors presented in the left frame are for the case where the exact normal is available, the right considers when the normal is approximated.

Funding. JAR and MLW were supported by the Department of Defense.

Acknowledgements. We would like to thank the referees for their helpful comments and critiques.

Disclaimer. The views expressed in this article are those of the authors and do not reflect the official policy or position of the United States Air Force, Department of Defense, or U.S. Government.

References

1. R. L. Hardy.
Theory and Applications of the Multiquadric-Biharmonic Method.
Comput. Math. Appl., 19(8-9):163–208, 1990.
2. J. P. Bardhan.
Efficient Numerical Algorithms for Surface Formulations of Mathematical Models for Biomolecule Analysis and Design.
PhD thesis, Massachusetts Institute of Technology, 2006.
3. B. Fornberg and N. Flyer.
A Primer on Radial Basis Functions with Applications to the Geosciences.
SIAM, 2015.
4. G. E. Fasshauer.
Meshfree Approximation Methods with MATLAB, volume 6 of *Interdisciplinary Mathematical Sciences*.
World Scientific, 2007.
5. W. Chen, Z.-J. Fu, and C. S. Chen.
Recent advances in Radial Basis Function collocation methods.
Springer-Verlag Berline Heidelberg, 2014.
6. N. Flyer, G.B. Wright, and B. Fornberg.
Radial basis function-generated finite differences: A mesh-free method for computational

geosciences.
In W. Freeden, M. Z. Nashed, and T. Sonar, editors, *Handbook of Geomathematics*. Springer Verlag Berlin Heidelberg.
doi: 10.1007/978-3-642-27793-1 61-1.

7. C. Piret.
The orthogonal gradients method: A radial basis functions method for solving partial differential equations on arbitrary surfaces.
J. Comput. Phys., 231:4662–4675, 2012.
8. E.J. Fuselier and G.B. Wright.
A high-order kernel method for diffusion and reaction-diffusion equations on surfaces.
J. Sci. Comput., 56:535–565, 2013.
9. Z. P. Bažant and B. H. Oh.
Efficient Numerical Integration on the Surface of a Sphere.
Zamm-Z. Angew. Math. ME., 66:37–49, 1986.
10. A. Sommariva and R. Womersley.
Integration by RBF over the Sphere.
Appl. Math. Report AMR05/17, 2005.
11. R. S. Womersley and I. H. Sloan.
Interpolation and Cubature on the Sphere.
http://web.maths.unsw.edu.au/ rsw/Sphere/, 2003/2007.
12. J. A. Reeger and B. Fornberg.
Numerical quadrature over the surface of a sphere.
Stud Appl Math, 2016.
doi: 10.1007/978-3-642-27793-1 61-1.
13. P. Keast and J. Diaz.
Fully Symmetric Integration Formulas for the Surface of the Sphere in S Dimensions.
SIAM J. Numer. Anal., 20(2):406–419, 1983.
14. V. Lebedev.
Quadratures on a Sphere.
USSR Comp. Math. Math.+, 16:1–23, 1976.
15. A. Stroud.
Approximate Calculation of Multiple Integrals.
Prentice-Hall, 1971.
New Jersey, USA.
16. B. Fornberg and J. M. Martel.
On spherical harmonics based numerical quadrature over the surface of a sphere.
Adv. Comput. Math., 40:1169–1184, 2014.
17. K. E. Atkinson.
Numerical Integration on the Sphere.
J. Aust. Math. Soc. B, 23:332–347, 1982.
18. E. Fuselier, T. Hangelbroek, F. J. Narcowich, J. D. Ward, and G. B. Wright.
Kernel based quadrature on spheres and other homogeneous spaces.
Numer. Math., 127:57–92, 2014.
19. D. Chien.
Numerical Evaluation of Surface Integrals in Three Dimensions.
Math. Comput., 64(210):727–743, 1995.
20. J. D'Elía, L. Battaglia, A. Cardona, and M. Storti.
Full Numerical Quadrature of Weakly Singular Double Surface Integrals in Galerkin Boundary Element Methods.
Int. J. Numer. Meth. Eng., 27(2):314–334, 2011.
21. G. Green.
An Essay on the Application of Mathematical Analysis to the Theories of Electricity and Magnetism.
Author, 1828.
Nottingham.
22. C. Ahrens and G. Beylkin.
Rotationally invariant quadratures for the sphere.
Proc. Roy. Soc. A, 465:3103–3125, 2009.
23. R.I. Saye.

High-order quadrature methods for implicitly defined surfaces and volumes in hyperrectangles.
SIAM J. Sci. Comput., 37:A993–A1019, 2015.

24. J. A. Reeger.
Smooth_Closed_Surface_Quadrature_RBF(Quadrature_Nodes,Triangles,h,gradh) (2016).
(https://www.mathworks.com/matlabcentral/fileexchange/57082-smooth-closed-surface-quadrature-rbf), MATLAB Central File Exchange. Accessed 11 May 2016.
25. J. A. Reeger.
Smooth_Closed_Surface_Quadrature_RBF-python: v1.0, May 2016.
(https://github.com/jareeger/Smooth_Closed_Surface_Quadrature_RBF-python/releases/tag/v1.0), Github. Accessed 19 May 2016. doi: 10.5281/zenodo.51687.
26. J. A. Reeger.
Smooth_Closed_Surface_Quadrature_RBF-julia: v1.0, May 2016.
(https://github.com/jareeger/Smooth_Closed_Surface_Quadrature_RBF-julia/releases/tag/v1.0), Github. Accessed 19 May 2016. doi: 10.5281/zenodo.51686.
27. MATLAB®, version 9.0.0.341360 (R2016a).
The MathWorks Inc., Natick, Massachusetts, 2016.
28. J. P. Snyder.
Map Projections: A Working Manual.
Geological Survey (U.S.), Washington, DC, U.S., 1987.
29. N. Flyer, E. Lehto, S. Blaise, G. B. Wright, and A. St-Cyr.
A guide to RBF-generated finite-differences for nonlinear transport: Shallow water simulations on a sphere.
J. Comput. Phys., 231(11):4078–4095, 2012.
30. B. Fornberg and N. Flyer.
Solving PDEs with radial basis functions.
Acta Numerica, 24:215–258, 2015.
31. H. Wendland.
Scattered Data Approximation, volume 17.
Cambridge University Press, 2004.
United Kingdom.
32. M. D. Buhmann.
Radial Basis Functions.
Acta Numerica, 9:1–38, 2000.
33. J. H. Friedman, J. L. Bentley, and R. A. Finkel.
An algorithm for finding best matches in logarithmic expected time.
ACM Trans Math Softw, 3(3):209–226,, 1977.
34. P. Persson.
Distmesh - a simple mesh generator in MATLAB (2012).
http://persson.berkeley.edu/distmesh/. Accessed 10 June 2015.
35. P. Persson and G. Strang.
A Simple Mesh Generator in MATLAB.
SIAM Rev., 46(2):329–345, June 2004.
36. ADVANPIX: Multiprecision Computing Toolbox, version 3.4.2.3222.
Advanpix.com, Yokohama, Japan, 2016.