Numerical quadrature over smooth surfaces with boundaries

By Jonah A. Reeger [1], Bengt Fornberg [2]

This paper describes a high order accurate method to calculate integrals over curved surfaces with boundaries. Given data locations that are arbitrarily distributed over the surface, together with some functional description of the surface and its boundary, the algorithm produces matching quadrature weights. This extends on the authors' earlier methods for integrating over the surface of a sphere and over arbitrarily shaped smooth closed surfaces by also considering domain boundaries. The core approach consists again of combining RBF-FD (radial basis function-generated finite difference) approximations for curved surface triangles, which together make up the full surface. The provided examples include both curved and flat domains. In the highly special case of equi-spaced nodes over a regular interval in 1-D, the method provides a new opportunity for improving on the classical Gregory enhancements of the trapezoidal rule.

Key words quadrature, radial basis function, RBFs, RBF-FD, Gregory's method, trapezoidal rule

1 Introduction

Algorithms for numerical quadrature typically determine weights, after which the evaluation of integrals becomes a matter of multiplying these with actual function values, and adding the results. In some cases (such as with Gaussian quadrature), the nodes (data locations) have to be chosen in a very special way. In applications, numerical quadrature is usually a

[1] Major, United States Air Force, supported by the Department of Defense and the Office of Naval Research's Atmospheric Propagation Sciences of High Energy Lasers. 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. 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.

follow-up to some other task (such as collecting data, or numerically solving PDEs), making it impractical to require node locations that are specific to the quadrature method. The present algorithm is therefore designed to find the quadrature weights at whatever node locations that are specified.

For N scattered nodes over the surface of a sphere, the algorithm described in [1] offers spectral accuracy, but at the relatively high cost of $O(N^3)$ operations and $O(N^2)$ memory. The RBF (radial basis function)-based method in [2] gives quadrature errors of size $O(1/N^2)$ (or equivalently $O(h^4)$ if h is a typical node separation distance) at costs of $O(N^2)$ for both operation count and memory. The first method in the authors' recent quadrature investigations [3] achieves $O(1/N^{3.5})$ (equivalent to $O(h^7)$) accuracy at the much reduced costs of $O(N \log N)$ and $O(N)$, respectively, as was also the case for the subsequent generalization from a sphere to arbitrarily shaped smooth closed surfaces [4]. These convergence rate and cost estimates hold again for the present further generalization to curved surfaces with smooth boundaries.

The core numerical method used in this paper is known as RBF-FD (radial basis function-generated finite differences). This approach has so far mostly been used to approximate partial derivatives, with the key difference to regular finite differences that the node points no longer need to be grid-based (in particular, Cartesian node layouts are now known to be less-than-optimal [5], which can also be seen in figures 10 and 11). For surveys of RBFs and of RBF-FD methods as these are applied to PDEs, see [6], [7].

The following Section 2 describes the present quadrature method. This starts with decomposing the surface into curved surface triangles which, between them, cover the full surface without any overlaps. Each of these surface triangles is then projected to a local tangent plane, in which the RBF-FD approach provides accurate quadrature weights. The integrals required in this step can be evaluated in closed form for triangles that do not share a side with the domain boundary. For the ones that do, a line integral is instead evaluated numerically. Another key component of the algorithm is a formula for converting quadrature weights in the tangent plane to corresponding weights at the surface nodes. Section 3 describes some test examples, with illustrations of convergence rates and computational costs. A Matlab implementation of the method is available at Matlab Central's File Exchange [8].

To better motivate certain aspects of the implementation, especially focusing on the accuracy at the boundaries (for which the surface curvature is not a critical aspect), we turn in Section 4 to some test cases for flat bounded domains. We note in particular that it is beneficial at boundaries to use larger projected regions than what are needed for interior triangles (consistent with similar observations in the context of using RBF-FD for PDEs [9]).

The case for which the effect of boundaries on quadrature rules have been studied most thoroughly in the literature (and certainly for the longest time) is the enhancement to the trapezoidal rule that was developed by James Gregory in 1670 [10] (remarkably, before Leibnitz' and Newton's first publications on the topic of calculus, in 1684 and 1687, respectively). A recent discussion of these Gregory end corrections can be found in [11]. When applying the present RBF-FD method to this case of equispaced nodes in 1-D, it can not only match but will typically even improve on the Gregory procedure. Results of the present method applied to 1-D are given in section 5.

2 Description of the key steps in the algorithm

Consider computing ($\mathbf{x} \in \mathbb{R}^3$)

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

where S is a finite surface defined implicitly. That is, $\mathbf{x} \in S$ if both $h(\mathbf{x}) = 0$ and $b(\mathbf{x}) \geq 0$ (see figure 3 for examples). These equations define a surface with a boundary curve satisfying the initial value problem

$$\frac{d}{ds}\mathbf{x}(s) = \frac{\nabla h(\mathbf{x}(s)) \times \nabla b(\mathbf{x}(s))}{\|\nabla h(\mathbf{x}(s)) \times \nabla b(\mathbf{x}(s))\|_2} \tag{1a}$$

$$\mathbf{x}(0) = \mathbf{x}_0 \tag{1b}$$

where s is arc-length along the boundary curve and $\mathbf{x}_0$ is a point on the boundary curve. In this definition, both the surface h and the boundary b are described as level surfaces in such a way that the inequality for b defines whether the surface is "above" or "below" the boundary curve. Other definitions of the surface could also be used. For instance, any combination of explicit and implicit parameterizations for the surface and boundary, or boundary curve, could be used.

Here the approach described in [4] for computing quadrature weights for surface integrals will be extended to surfaces like S. In this sequel, it is still assumed that a set, $\mathcal{S}_N$, of N quadrature nodes on the surface and a triangulation $T = \{t_{A_k B_k C_k}\}_{k=1}^{K}$ (which maps one-to-one onto the set $\mathcal{T} = \{\tau_{A_k B_k C_k}\}_{k=1}^{K}$ of curved "surface triangles" that covers S) are provided by the user. Since the method in [4] considers each triangle in $\mathcal{T}$ separately so that

$$\mathcal{I}_S(f) = \sum_{k=1}^{K} \iint\limits_{\tau_{A_k B_k C_k}} f(\mathbf{x}) dS \tag{2}$$

this generalization requires only that "boundary triangles" (with at least one edge, i.e. at least two vertices on the boundary) be handled differently from "interior triangles". The method in [4] can be used without specifying $h(x, y, z)$ by approximating normal vectors to the surface. Likewise, the present method could be adapted so that only a set of nodes, a triangulation of the set of nodes, and an ordered list of the nodes on the boundary are required from the user.

This method for determining quadrature weights for approximating the surface integral can be summarized in the same five steps as in [4]:

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

However, in general steps 1 and 3 are handled differently when a boundary triangle is being considered. It should be noted that when $b(\mathbf{x}) = \mathbf{n}_b^T \mathbf{x}$ (with $\mathbf{n}_b$ a constant vector, i.e. a planar boundary) the method in [4] can easily handle such a case with only a change in the definition of the projection point in step 1 with no modifications to step 3. Similarly, in the case where $h(\mathbf{x}) = \mathbf{n}_h^T \mathbf{x}$ (with $\mathbf{n}_h$ a constant vector), that is, the surface is planar, the present method reduces to steps 3 and 5 only.

The remainder of this section will describe the steps in the algorithm and the modifications needed for steps 1 and 3 when considering boundary triangles.

2.1 Step 1: Locate a Projection Point

In the method described in [4], the first step requires that a projection point be found by first defining "cutting planes" along the edges of the curved surface triangles in $\mathcal{T}$. The left frame of figure 1 illustrates a flat triangle $t_{A_kB_kC_k}$, its curved counterpart $\tau_{A_kB_kC_k}$, and a projection point $\mathbf{x}_{O_k}$. The right frame illustrates how a flat triangle meets another triangle along each of its edges. The curved counterparts to any two adjacent triangles must meet without any gap or overlap. For each side of the central triangle, this requirement defines a plane in which $\mathbf{x}_{O_k}$ must be located. With three sides, we get three planes, and $\mathbf{x}_{O_k}$ is chosen as their intersection point. In vector notation [4]:

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

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. Here $\mathbf{n}_{A_kB_k} = \frac{1}{2}\left(\mathbf{n}_{A_kB_kC_k} + \operatorname{sign}\left(\mathbf{n}^T_{A_kB_kC_k}\mathbf{n}_{A_kB_kE_k}\right)\mathbf{n}_{A_kB_kE_k}\right)$ is 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 $\frac{\pi}{2}$). The right frame of figure 1 illustrates this definition of the projection point.

2.1.1 Modifications to Step 1: Defining the "Cutting" Plane

If $t_{A_kB_kC_k}$ is a boundary triangle and $b(\mathbf{x}_{A_k}) = 0$ and $b(\mathbf{x}_{B_k}) = 0$, then there is no triangle in T sharing the edge A_kB_k, and $\mathbf{n}_{A_kB_k}$ must be defined in a different way.

First, suppose that $b(\mathbf{x})$ is not the equation of a plane (or, at least, the algorithm does not know it is). Then $\mathbf{n}_{A_kB_k}$ is set equal to the normal vector, $\mathbf{n}_{A_kB_kC_k}$, to the triangle $t_{A_kB_kC_k}$.

Next, suppose that $b(\mathbf{x})$ is the equation of a plane which, since $\mathbf{x}_{A_k}$ satisfies $b(\mathbf{x}_{A_k}) = 0$, could be written

$$b(\mathbf{x}) = \mathbf{n}_b \cdot (\mathbf{x} - \mathbf{x}_{A_k})$$

where $\cdot$ represents the vector dot (inner) product on $\mathbb{R}^3$. In this case, the vector $\mathbf{n}_{A_kB_k}$ is defined to be perpendicular to the vector $\mathbf{x}_{A_k} - \mathbf{x}_{B_k}$ and parallel to the plane $b(\mathbf{x})$ (i.e.

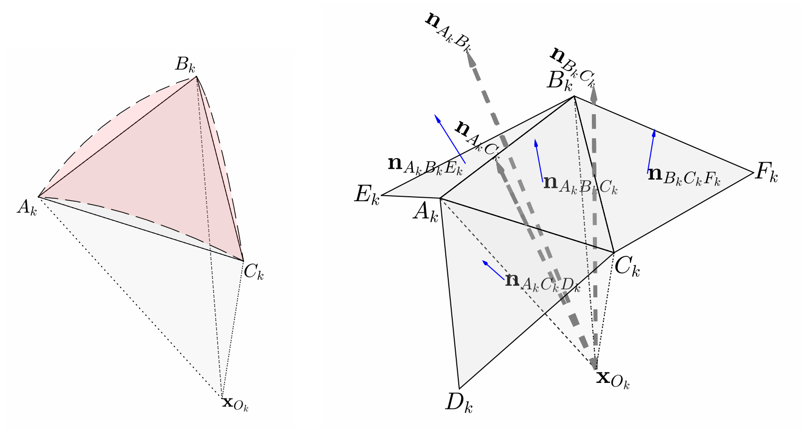

Figure 1: (left) A single flat triangle, $t_{A_kB_kC_k}$, (outlined by solid lines) and its curved counterpart, $\tau_{A_kB_kC_k}$, (outlined by dashed lines). (right) An illustration of a central flat triangle and its three immediate neighbors, all with the normal vectors shown as solid arrows. The 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 $\mathbf{n}_{A_kB_k} = \frac{1}{2}(\mathbf{n}_{A_kB_kC_k} + \text{sign}(\mathbf{n}^T_{A_kB_kC_k}\mathbf{n}_{A_kB_kE_k})\mathbf{n}_{A_kB_kE_k})$, $\mathbf{n}_{B_kC_k} = \frac{1}{2}(\mathbf{n}_{A_kB_kC_k} + \text{sign}(\mathbf{n}^T_{A_kB_kC_k}\mathbf{n}_{B_kC_kF_k})\mathbf{n}_{B_kC_kF_k})$ and $\mathbf{n}_{C_kA_k} = \frac{1}{2}(\mathbf{n}_{A_kB_kC_k} + \text{sign}(\mathbf{n}^T_{A_kB_kC_k}\mathbf{n}_{A_kC_kD_k})\mathbf{n}_{A_kC_kD_k})$.

perpendicular to $\mathbf{n}_b$). That is, $\mathbf{n}_{A_kB_k}$ should be in the direction of the vector

$$\mathbf{n}_{A_kB_k} = \frac{\mathbf{n}_b \times (\mathbf{x}_{A_k} - \mathbf{x}_{B_k})}{\|\mathbf{n}_b \times (\mathbf{x}_{A_k} - \mathbf{x}_{B_k})\|_2}.$$

2.2 Step 2: Project a neighborhood of a curved surface triangle

Once the point $\mathbf{x}_{O_k}$ is available, points $\mathbf{x}$ in a neighborhood of $\tau_{A_kB_kC_k}$, which is the region of S that projects onto $t_{A_kB_kC_k}$ (including its boundary), must be projected into the plane containing $t_{A_kB_kC_k}$ for the interpolation procedure described in section 2.3. The projection occurs by determining the intersection of this plane and the line through $\mathbf{x}_{O_k}$ and in the direction of $(\mathbf{x} - \mathbf{x}_{O_k})$. Further, a two-dimensional coordinate system is defined in the plane containing $t_{A_kB_kC_k}$ that ensures that the midpoint of this triangle is the origin of the coordinate system. This projection process can be summarized by

$$\boldsymbol{\chi}_k(\mathbf{x}(s)) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} R_k \frac{1}{\mathbf{n}_{A_kB_kC_k} \cdot (\mathbf{x}(s) - \mathbf{x}_{O_k})} (\mathbf{n}_{A_kB_kC_k} \times ((\mathbf{x}(s) - \mathbf{x}_{O_k}) \times (\mathbf{x}_{M_k} - \mathbf{x}_{O_k}))) \tag{3}$$

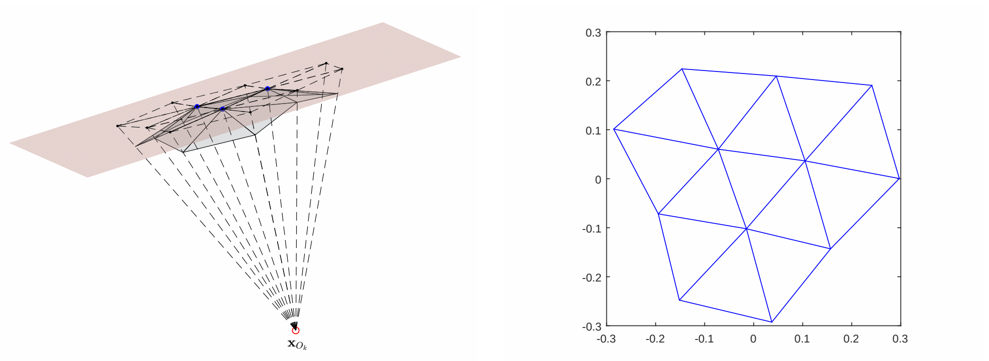

Figure 2: An illustration of the projection of a triangle and its neighbors. Left: Here $\tau_{A_kB_kC_k}$ is the central triangle and $n = 12$ of its nearest neighbors (including the 3 vertices) are projected from $\mathbf{x}_{O_k}$ into the plane containing the vertices of $\tau_{A_kB_kC_k}$. The dashed 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 (3). This figure is adapted from [4].

where $\mathbf{x}_{M_k} = 1/3(\mathbf{x}_{A_k} + \mathbf{x}_{B_k} + \mathbf{x}_{C_k})$, $\mathbf{x}_{O_k}$ is the projection point for $t_{A_kB_kC_k}$, $\mathbf{n}_{A_kB_kC_k}$ is the normal vector to $t_{A_kB_kC_k}$ with components n_{x_k}, n_{y_k} and n_{z_k}, and

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

is a rotation matrix (as long as $\sqrt{n_{x_k}^2 + n_{y_k}^2} \neq 0$ in which case $R_k = I$). An example projection neighborhood containing 12 neighboring vertices is illustrated in the left frame of figure 2. The right frame of figure 2 illustrates the two-dimensional coordinate system in the plane.

In the case of a planar surface $h(\mathbf{x})$, (3) reduces to (with $\mathbf{n}_{A_kB_kC_k} = \mathbf{n}_h$ for each k)

$$\boldsymbol{\chi}_k(\mathbf{x}(s)) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} R_k\left(\mathbf{x}(s) - \mathbf{x}_{M_k}\right). \tag{4}$$

2.3 Step 3: Find Quadrature Weights for Integrating Over a Projected Triangle

Consider the double integral of a function $g(\boldsymbol{\chi}_k)$ over a planar triangle $t_{A_kB_kC_k}$

$$I_{t_{A_kB_kC_k}}(g) := \iint\limits_{t_{A_kB_kC_k}} g(\boldsymbol{\chi}_k)dA \tag{5}$$

It can be evaluated approximately by integrating the RBF interpolant of $g(\boldsymbol{\chi}_k)$ with basis functions $\phi\left(\left\|\boldsymbol{\chi}_k - \boldsymbol{\chi}_{k,j}\right\|\right)$ centered at the projections of the n points in $\mathcal{N}_k^n$ (the set of n nearest neighbors in $\mathcal{S}_N$ to the midpoint of $t_{A_kB_kC_k}$), which are all in a neighborhood of $t_{A_kB_kC_k}$.

It is common to construct the RBF-interpolant as

$$s(\boldsymbol{\chi}_k) := \sum_{j=1}^{n} c_{k,j}^{\text{RBF}} \phi\left(\left\|\boldsymbol{\chi}_k - \boldsymbol{\chi}_{k,j}\right\|\right) + \sum_{l=1}^{M} c_{k,l}^{p} \pi_l(\boldsymbol{\chi}_k) \tag{6}$$

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 $l = 1, 2, \ldots, M$. Here the set $\{\pi_l(\boldsymbol{\chi}_k)\}_{l=1}^{M}$, with $M = \frac{(m+1)(m+2)}{2}$, contains all of the bivariate polynomial terms up through degree m.

By integrating the interpolant the approximation of the integral of g is reduced to $I_{t_{A_kB_kC_k}}(g) \approx \sum_{j=1}^{n} w_{k,j}^{RBF} g(\boldsymbol{\chi}_{k,j})$, where the weights can be found by solving the linear system $\tilde{A}_k W_k = \tilde{I}_k$ with

$$\tilde{A}_k = \begin{bmatrix} A_k^T & P_k \\ P_k^T & 0 \end{bmatrix}, W_k = \begin{bmatrix} \mathbf{w}_k^{RBF} \\ \mathbf{w}^p \end{bmatrix}, \text{ and } \tilde{I}_k = \begin{bmatrix} I_k^{RBF} \\ I_k^p \end{bmatrix}.$$

Here, $A_{k,ij} = \phi\left(\left\|\boldsymbol{\chi}_{k,i} - \boldsymbol{\chi}_{k,j}\right\|\right)$, $P_{k,il} = \pi_l(\boldsymbol{\chi}_{k,i})$, $\mathbf{w}_{k,j}^{RBF} = w_{k,j}^{RBF}$, $I_{k,j}^{RBF} = I_{t_{A_kB_kC_k}}\left(\phi\left(\left\|\boldsymbol{\chi}_k - \boldsymbol{\chi}_{k,j}\right\|\right)\right)$, and $I_{k,l}^{p} = I_{t_{A_kB_kC_k}}(\pi_l(\boldsymbol{\chi}_k))$, for $i, j = 1, 2, \ldots, n$ and $l = 1, 2, \ldots, M$ [7, Section 5.1.4].

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. When considering interior triangles or boundary triangles whose edges are projected via (3) as straight lines, 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 detail in [3], 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

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

In the preceding expression η is the distance (the base) between $\boldsymbol{\chi}_{k,j}$ and the vertex at the right angle and ζ is the length (the height) of the opposite side. Such expressions can also be found for many of the most popular choices of RBFs, including $\phi(r) = r^l$ (l odd) and $\phi(r) = r^l \ln r$ (l even), and for Gaussian, multiquadric, and inverse multiquadric basis functions. For example, in the case of the inverse multiquadric, the counterpart to (7) is (with ϵ the shape parameter)

$$\iint_t \frac{1}{\sqrt{1+(\epsilon r)^2}} dA =$$
$$\frac{1}{\epsilon^2}\left[-\tan^{-1}\left(\frac{\eta\sqrt{\epsilon^2\left(\eta^2+\zeta^2\right)+1}}{\zeta}\right) + \eta\epsilon\sinh^{-1}\left(\frac{\zeta\epsilon}{\sqrt{\eta^2\epsilon^2+1}}\right) + \tan^{-1}\left(\frac{\eta}{\zeta}\right)\right].$$

2.3.1 Modifications to Step 3: Find Quadrature Weights for Integrating over the Projection of a Boundary Triangle

On the other hand, if $b(\mathbf{x})$ is not the equation of a plane, then it is unlikely that any definition of $\mathbf{n}_{A_k B_k}$ will lead to the portion of the boundary curve between $\mathbf{x}_{A_k}$ and $\mathbf{x}_{B_k}$ being projected as a straight line into the plane containing $t_{A_k B_k C_k}$. The projection of this portion of the curve will likely be still another curve in the plane containing $t_{A_k B_k C_k}$. Assuming that the boundary curve satisfying (1) does not self-intersect, this projected curve and the projections of the sides $B_k C_k$ and $C_k A_k$ altogether form a simple closed (piecewise) curve. Therefore, to integrate a function $g(\boldsymbol{\chi}_k)$ (for instance, a RBF) over the area contained by the simple closed curve, Green's theorem can be employed. As a reminder $\boldsymbol{\chi}_k \in \mathbb{R}^2$ are the coordinates in the 2-dimensional coordinate system resulting from the projection described in step 2 of the method in [4].

For the simplicity of discussion consider approximately evaluating the double integral of a function $g(\boldsymbol{\chi}_k)$ over an area $\mathcal{A}_k$ in the 2-dimensional coordinate system. Suppose that the boundary of this area, $\mathcal{C}$ is a piecewise simple closed curve with segments given by $\mathcal{C}_{A_kB_k}$, $\mathcal{C}_{B_kC_k}$, and $\mathcal{C}_{C_kA_k}$, where, for instance, the projections of $\mathbf{x}_{A_k}$ and $\mathbf{x}_{B_k}$ represented in the 2-dimensional coordinate system are the endpoints of $\mathcal{C}_{A_kB_k}$ and the curve is traversed from the transformation of $\mathbf{x}_{A_k}$ to that of $\mathbf{x}_{B_k}$.

If two functions $L(\boldsymbol{\chi}_k)$ and $M(\boldsymbol{\chi}_k)$ are chosen so that

$$g(\boldsymbol{\chi}_k) = \left(\nabla \times \begin{bmatrix} L(\boldsymbol{\chi}_k) \\ M(\boldsymbol{\chi}_k) \\ 0 \end{bmatrix} \right) \cdot \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix},$$

then by Stoke's or Green's theorem,

$$\iint\limits_{\mathcal{A}_k} g(\boldsymbol{\chi}_k) dA = \int\limits_{\mathcal{C}} \begin{bmatrix} L(\boldsymbol{\chi}_k) \\ M(\boldsymbol{\chi}_k) \end{bmatrix} \cdot d\boldsymbol{\chi}_k. \tag{8}$$

Suppose that $\mathcal{C}_{A_kB_k}$ is not the projection of a boundary curve. That is, $b(\mathbf{x}_{A_k}) > 0$ and/or $b(\mathbf{x}_{B_k}) > 0$. Then $\mathcal{C}_{A_kB_k}$ is a line and can be parameterized explicitly via

$$\boldsymbol{\chi}_k(t) = \boldsymbol{\chi}_{k,A_k} + t(\boldsymbol{\chi}_{k,B_k} - \boldsymbol{\chi}_{k,A_k}),\ 0 \le t \le 1 \tag{9}$$

so that

$$\int\limits_{\mathcal{C}_{A_kB_k}} \begin{bmatrix} L(\boldsymbol{\chi}_k) \\ M(\boldsymbol{\chi}_k) \end{bmatrix} \cdot d\boldsymbol{\chi}_k = \int\limits_0^1 \begin{bmatrix} L(\boldsymbol{\chi}_k(t)) \\ M(\boldsymbol{\chi}_k(t)) \end{bmatrix} \cdot (\boldsymbol{\chi}_{k,B_k} - \boldsymbol{\chi}_{k,A_k}) dt \tag{10}$$

On the other hand, if $\mathcal{C}_{A_kB_k}$ is the projection of a boundary curve, then the solution to the system (1) is an explicit parameterization with respect to arc length along the boundary curve, as long as the arc length between $\mathbf{x}_{A_k}$ and $\mathbf{x}_{B_k}$ is known. Denote this arc length by $s_{A_kB_k}$ and let $\mathbf{x}(s)$ be the solution to (1) with $\mathbf{x}_0 = \mathbf{x}_{A_k}$ and $0 \le s \le s_{A_kB_k}$. The solution can be transformed into the two coordinate system via (3) or (4).

Given (3) or (4) the differentials in (8) can be expressed in terms of the differential ds.

That is,

$$d\boldsymbol{\chi}_k = \nabla_{\mathbf{x}(s)}\boldsymbol{\chi}_k(\mathbf{x}(s))\frac{d}{ds}\mathbf{x}(s)ds,$$

where $\nabla_{\mathbf{x}(s)}\boldsymbol{\chi}_k(\mathbf{x}(s))$ is the Jacobian of $\boldsymbol{\chi}_k(\mathbf{x}(s))$ with respect to the entries of $\mathbf{x}(s)$. With this differential

$$\int\limits_{\mathcal{C}_{A_kB_k}} \begin{bmatrix} L(\boldsymbol{\chi}_k) \\ M(\boldsymbol{\chi}_k) \end{bmatrix} \cdot d\boldsymbol{\chi}_k = \int\limits_0^{s_{A_kB_k}} \begin{bmatrix} L(\boldsymbol{\chi}_k(\mathbf{x}(s))) \\ M(\boldsymbol{\chi}_k(\mathbf{x}(s))) \end{bmatrix} \cdot \left(\nabla_{\mathbf{x}(s)}\boldsymbol{\chi}_k(\mathbf{x}(s))\frac{d}{ds}\mathbf{x}(s)\right) ds. \tag{11}$$

It is unlikely that the line integrals over the curves $\mathcal{C}_{A_kB_k}$, $\mathcal{C}_{B_kC_k}$ and $\mathcal{C}_{C_kA_k}$ can be computed in closed form, so they are computed numerically. Here a Newton based root finding method and the system (1) are used to numerically determine (if needed) the arc length, $s_{A_kB_k}$, between $\mathbf{x}_{A_k}$ and $\mathbf{x}_{B_k}$ (given the Euclidean distance between the two points as a guess). This is done by finding the root of the function

$$\sigma(s_{A_kB_k}) = \|\mathbf{x}(s_{A_kB_k}) - \mathbf{x}_{B_k}\|_2^2. \tag{12}$$

Notice that from (1)

$$\begin{aligned}\frac{d}{ds_{A_kB_k}}\sigma(s_{A_kB_k}) =&2\left(\mathbf{x}(s_{A_kB_k}) - \mathbf{x}_{B_k}\right)\cdot\frac{d}{ds_{A_kB_k}}\mathbf{x}(s_{A_kB_k}) \\ =&2\left(\mathbf{x}(s_{A_kB_k}) - \mathbf{x}_{B_k}\right)\cdot\frac{\nabla h(\mathbf{x}(s_{A_kB_k}))\times\nabla b(\mathbf{x}(s_{A_kB_k}))}{\|\nabla h(\mathbf{x}(s_{A_kB_k}))\times\nabla b(\mathbf{x}(s_{A_kB_k}))\|_2},\end{aligned}$$

where $\mathbf{x}(s_{A_kB_k})$ can be found by solving (1) numerically with the initial value $\mathbf{x}_0 = \mathbf{x}_{A_k}$.

When $\mathcal{C}_{A_kB_k}$ is not the projection of a boundary curve, the value of the integral (10) is computed numerically by adaptively solving the initial value problem

$$\frac{d}{dt}\mathcal{I}_{\mathcal{C}_{A_kB_k}}(t) = \begin{bmatrix} L(\boldsymbol{\chi}_k(t)) \\ M(\boldsymbol{\chi}_k(t)) \end{bmatrix} \cdot (\boldsymbol{\chi}_{k,B} - \boldsymbol{\chi}_{k,A}) \tag{13a}$$

$$\mathcal{I}_{\mathcal{C}_{A_kB_k}}(0) =0 \tag{13b}$$

for $\mathcal{I}_{\mathcal{C}_{A_kB_k}}(1)$ or by applying numerical quadrature directly to (10). In this case, quadrature nodes are easily chosen since the parameterization (9) can be directly evaluated for any choice of t. Likewise, when $\mathcal{C}_{A_kB_k}$ is the projection of a boundary curve, the value of the integral (11) is computed numerically by adaptively solving simultaneously

$$\begin{aligned}
\frac{d}{ds}\mathcal{I}_{\mathcal{C}_{A_kB_k}}(s) =& \begin{bmatrix} L(\boldsymbol{\chi}_k(\mathbf{x}(s))) \\ M(\boldsymbol{\chi}_k(\mathbf{x}(s))) \end{bmatrix} \cdot \left(\nabla_{\mathbf{x}(s)}\boldsymbol{\chi}_k(\mathbf{x}(s))\frac{d}{ds}\mathbf{x}(s)\right) && \text{(14a)}\\
\mathcal{I}_{\mathcal{C}_{A_kB_k}}(0) =& 0 && \text{(14b)}\\
\frac{d}{ds}\mathbf{x}(s) =& \frac{\nabla h(\mathbf{x}(s)) \times \nabla b(\mathbf{x}(s))}{\|\nabla h(\mathbf{x}(s)) \times \nabla b(\mathbf{x}(s))\|_2} && \text{(14c)}\\
\mathbf{x}(0) =& \mathbf{x}_{A_k} && \text{(14d)}
\end{aligned}$$

for $\mathcal{I}_{\mathcal{C}_{A_kB_k}}(s_{A_kB_k})$ or by applying numerical quadrature to (11). In this case, the system (1) must be solved with initial value $\mathbf{x}_0 = \mathbf{x}_{A_k}$ for each choice of s needed as a quadrature node.

2.4 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) that relate the surface integral to an integral over an area in a plane. Denote the surface normal to S to be

$$\mathbf{n}_S(\mathbf{x}) := \frac{\nabla h(\mathbf{x})}{\|\nabla h(\mathbf{x})\|_2}. \tag{15}$$

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 $\tau_{A_kB_kC_k}$ is

$$\iint\limits_{\tau_{A_kB_kC_k}} f(\mathbf{x})dS = \iint\limits_{t_{A_kB_kC_k}} f(\mathbf{x}(\boldsymbol{\chi}_k))\frac{\mathbf{n}_{P_k}\cdot(\mathbf{x}(\boldsymbol{\chi}_k)-\mathbf{x}_{O_k})}{\mathbf{n}_S(\mathbf{x}(\boldsymbol{\chi}_k))\cdot(\mathbf{x}(\boldsymbol{\chi}_k)-\mathbf{x}_{O_k})}\left(\frac{\mathbf{n}_{A_kB_kC_k}\cdot(\mathbf{x}(\boldsymbol{\chi}_k)-\mathbf{x}_{O_k})}{\mathbf{n}_{A_kB_kC_k}\cdot(\mathbf{x}_{A_k}-\mathbf{x}_{O_k})}\right)^2 dA. \tag{16}$$

A description of the last two factors in (16) can be found in [4]

Applying step 3 to the double integral (16) over a flat triangle $t_{A_kB_kC_k} \in T$ gives, for instance

$$\begin{aligned}\iint\limits_{t_{A_kB_kC_k}} & f(\mathbf{x}(\boldsymbol{\chi}_k))\frac{\mathbf{n}_{P_k}\cdot(\mathbf{x}(\boldsymbol{\chi}_k)-\mathbf{x}_{O_k})}{\mathbf{n}_S(\mathbf{x}(\boldsymbol{\chi}_k))\cdot(\mathbf{x}(\boldsymbol{\chi}_k)-\mathbf{x}_{O_k})}\left(\frac{\mathbf{n}_{A_kB_kC_k}\cdot(\mathbf{x}(\boldsymbol{\chi}_k)-\mathbf{x}_{O_k})}{\mathbf{n}_{A_kB_kC_k}\cdot(\mathbf{x}_{A_k}-\mathbf{x}_{O_k})}\right)^2 dA \\ &\approx \sum_{j=1}^{n} w_{k,j}^{\text{RBF}} f(\mathbf{x}_{k,j})\frac{\mathbf{n}_{P_k}\cdot(\mathbf{x}_{k,j}-\mathbf{x}_{O_k})}{\mathbf{n}_S(\mathbf{x}_{k,j})\cdot(\mathbf{x}_{k,j}-\mathbf{x}_{O_k})}\left(\frac{\mathbf{n}_{A_kB_kC_k}\cdot(\mathbf{x}_{k,j}-\mathbf{x}_{O_k})}{\mathbf{n}_{A_kB_kC_k}\cdot(\mathbf{x}_{A_k}-\mathbf{x}_{O_k})}\right)^2. \qquad (17)\end{aligned}$$

2.5 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

$$\mathcal{I}_S(f) \approx \sum_{k=1}^{K}\sum_{j=1}^{n} w_{k,j}^{\text{RBF}} f(\mathbf{x}_{k,j})\frac{\mathbf{n}_{P_k}\cdot(\mathbf{x}_{k,j}-\mathbf{x}_{O_k})}{\mathbf{n}_S(\mathbf{x}_{k,j})\cdot(\mathbf{x}_{k,j}-\mathbf{x}_{O_k})}\left(\frac{\mathbf{n}_{A_kB_kC_k}\cdot(\mathbf{x}_{k,j}-\mathbf{x}_{O_k})}{\mathbf{n}_{A_kB_kC_k}\cdot(\mathbf{x}_{A_k}-\mathbf{x}_{O_k})}\right)^2. \qquad (18)$$

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

$$\begin{aligned}\mathcal{I}_S(f) &\approx \sum_{i=1}^{N}\left(\sum_{(k,j)\in\mathcal{K}_i} w_{k,j}^{\text{RBF}}\frac{\mathbf{n}_{P_k}\cdot(\mathbf{x}_{k,j}-\mathbf{x}_{O_k})}{\mathbf{n}_S(\mathbf{x}_{k,j})\cdot(\mathbf{x}_{k,j}-\mathbf{x}_{O_k})}\left(\frac{\mathbf{n}_{A_kB_kC_k}\cdot(\mathbf{x}_{k,j}-\mathbf{x}_{O_k})}{\mathbf{n}_{A_kB_kC_k}\cdot(\mathbf{x}_{A_k}-\mathbf{x}_{O_k})}\right)^2\right) f(\mathbf{x}_i) \\ &= \sum_{i=1}^{N} w_i f(\mathbf{x}_i). \qquad (19)\end{aligned}$$

3 Test examples of quadrature over curved bounded surfaces

The test surfaces in this paper are a family of surfaces generated by rotating the Cassini ovals about the x-axis. The current default settings for the method are used with $n = 80$ (number of nearest neighbors) and $m = 7$ (maximum order of bivariate polynomial terms in interpolation) and the RBF $\phi(r) = r^7$. Consider the level surfaces defined by

$$h(\mathbf{x}) = h(x,y,z) = (x^2+y^2+z^2)^2 - 2\alpha^2(x^2-y^2-z^2) + \alpha^4 - \beta^4 = 0, \qquad (20)$$

which depend on the two parameters α and β. The demonstrations here will consider $\alpha = \lambda\beta$ for $0 < \lambda < 1$ (specifically, $\lambda = 0, 0.8, 0.95$). The parameter β in this work is chosen so that

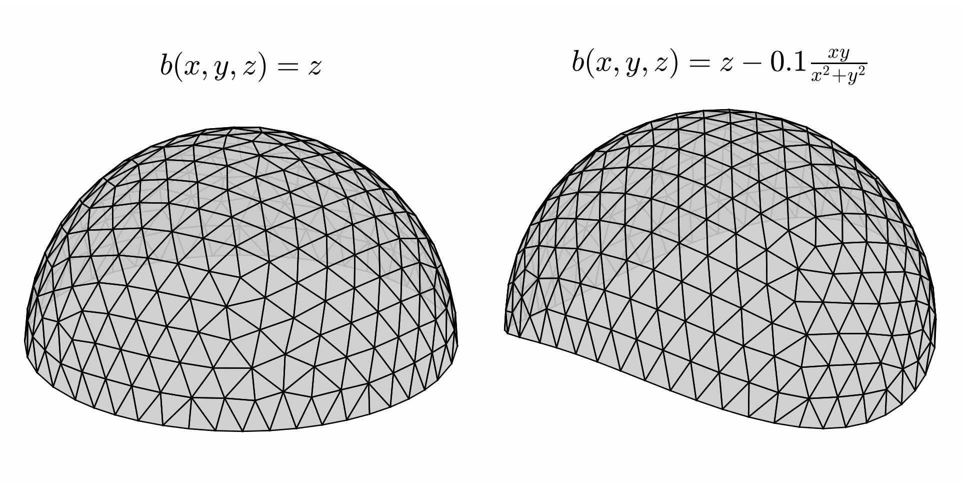

Figure 3: Examples of the surface S defined implicitly by $h(\mathbf{x}) = h(x, y, z) = (x^2 + y^2 + z^2)^2 - 2\alpha^2(x^2 - y^2 - z^2) + \alpha^4 - \beta^4 = 0$ when $\alpha = 0$ and β chosen that the surface area (of the closed surface) equals 1. The surface is shown for two different boundary surfaces $b(\mathbf{x}) = b(x, y, z)$.

the surface area (of the closed surface) is equal to 1, and since the area of this surface of revolution is not known explicitly, β is chosen numerically.

Further, two boundary surfaces $b(\mathbf{x})$ are considered. First, let $b(\mathbf{x}) = b(x, y, z) = z$. This is the xy–plane, and the surface with boundary is the upper half of the original surface. Second, take $b(x, y, z) = z - 0.1\frac{xy}{x^2+y^2}$, where for $\lambda = 0$ this creates a "scalloped" sphere. The surface area of any of these surfaces with their boundary is 0.5. The case of $\lambda = 0$ is shown for either boundary in figure 3.

There are also four test integrands considered for surfaces in this paper. These test integrands lead to integrals that can be computed in closed form over the closed surfaces. Denote S^+ to be the surface bounded by $b(x, y, z) \geq 0$ and S^- to be the surface bounded by $-b(x, y, z) \geq 0$. Then, $S^+ \cup S^-$ is a closed surface, where $S^+ \cap S^-$ is only the boundary curve specified by (1). Summing the integrals over the two halves of the closed surface also allows consideration of maximum absolute errors after 1000 random rotations of each half about the x-axis. These are the errors that will be shown in this section for the first three test functions, since the fourth test function is not amenable to any sort of rotation.

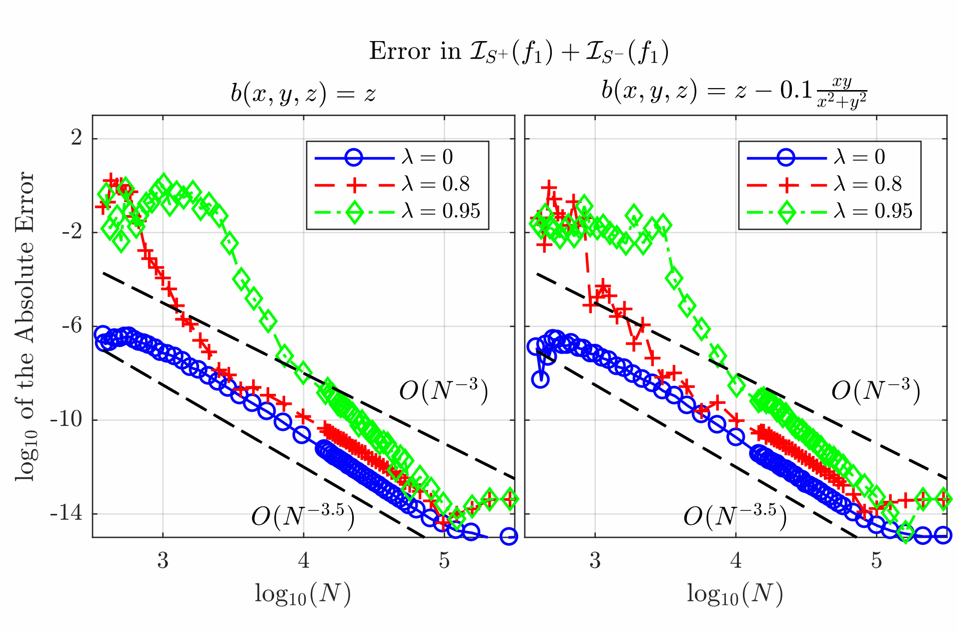

Figure 4: Maximum absolute error after 1000 random rotations about the x-axis when evaluating $\mathcal{I}_{S^+}(f_1)+\mathcal{I}_{S^-}(f_1)$ with the two different boundary surfaces $b(x,y,z)$.

First, consider

$$f_1(\mathbf{x})=\frac{1}{3}\mathbf{x}\cdot\mathbf{n}_S(\mathbf{x}),\;\; i=1,2,3$$

where $\mathbf{n}_S(\mathbf{x})$ is defined in (15). From the Divergence Theorem

$$\mathcal{I}_S^+(f_1)+\mathcal{I}_S^-(f_1)=\frac{\pi}{6\alpha}\left[2\alpha(\beta^2-2\alpha^2)\sqrt{\alpha^2+\beta^2}+3\beta^4\sinh^{-1}\left(\frac{2\alpha\sqrt{\alpha^2+\beta^2}}{\beta^2}\right)\right]$$

is the volume of the closed surface. The integrand $f_1(\mathbf{x})$ is smooth and, as with the methods discussed in [3] and [4] from which the present method is developed, the convergence order is $O(N^{-3.5})$ for either a planar or curved boundary. The absolute errors when computing the volume are shown in figure 4.

Second, the test integrand $f_2(\mathbf{x})=f_2(x,y,z)=\frac{2}{\pi}\tan^{-1}(z)$ is again smooth, but features a steep gradient near $z=0$ and $\mathcal{I}_{S^+}(f_3)+\mathcal{I}_{S^-}(f_3)=0$. The method also achieves a convergence order of $O(N^{-3.5})$ for either boundary, and the absolute errors for computing this surface integral are shown in figure 5.

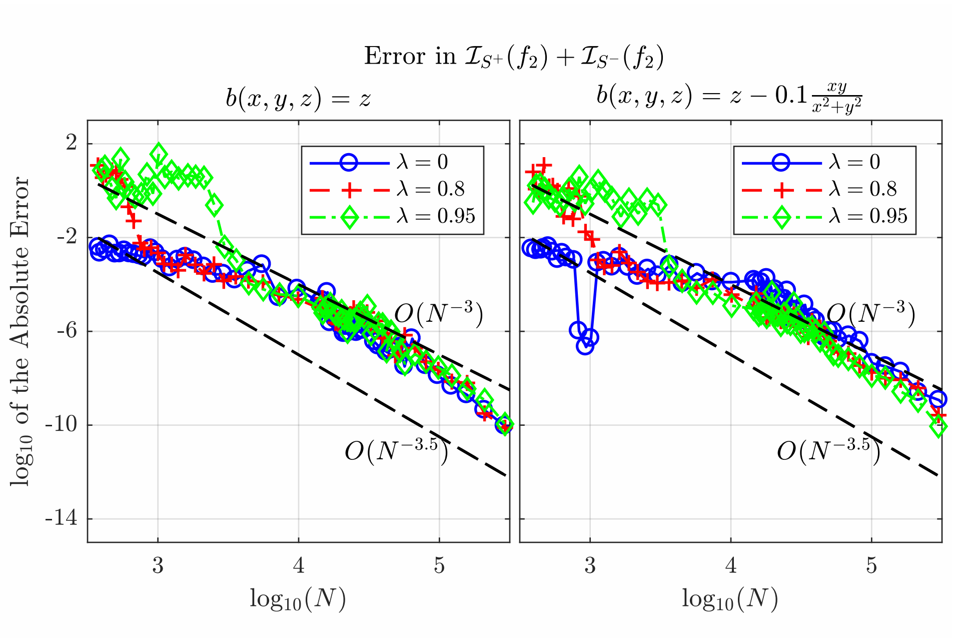

Figure 5: Maximum absolute error after 1000 random rotations about the x-axis when evaluating $\mathcal{I}_{S^+}(f_2)+\mathcal{I}_{S^-}(f_2)$ with the two different boundary surfaces $b(x,y,z)$.

Third, $f_3(\mathbf{x})=f_3(x,y,z)=\text{sign}(z)$ so that $\mathcal{I}_{S^+}(f_3)+\mathcal{I}_{S^-}(f_3)=0$. This integrand is discontinuous where $z=0$ and so it achieves a convergence order between $O(N^{-0.5})$ and $O(N^{-1})$. In this case, the errors are shown in figure 6.

Finally, to explore the impact of node clustering on convergence orders where the integrand includes rapidly changing features, the integrand

$$f_4(\mathbf{x})=\frac{1}{2}+\frac{\tan^{-1}(1000(z-\frac{9999}{10000}\frac{1}{2\sqrt{\pi}}))}{\pi} \tag{21}$$

is also considered, but only for the case where the $\lambda=0$ for the test surface. This test integrand is smooth, but features a sharp spike near the point $\left(0,0,\frac{1}{2\sqrt{\pi}}\right)$. This surface integral for this integrand evaluates exactly to

$$\mathcal{I}_{S^+}(f_4)+\mathcal{I}_{S^-}(f_4)=\frac{5000\pi+10\sqrt{\pi}\log\left(1+\frac{99980000}{1+400\pi}\right)-9999\tan^{-1}\left(\frac{9999}{20\sqrt{\pi}}\right)+\cot^{-1}\left(20\sqrt{\pi}\right)}{20000\pi}.$$

We use clustered nodes generated by a modification of the algorithm described in [12] with a node separation function $d(x,y,z)=\omega\sqrt{x^2+y^2+(z-1.2)^2}$ that are scaled to a sphere

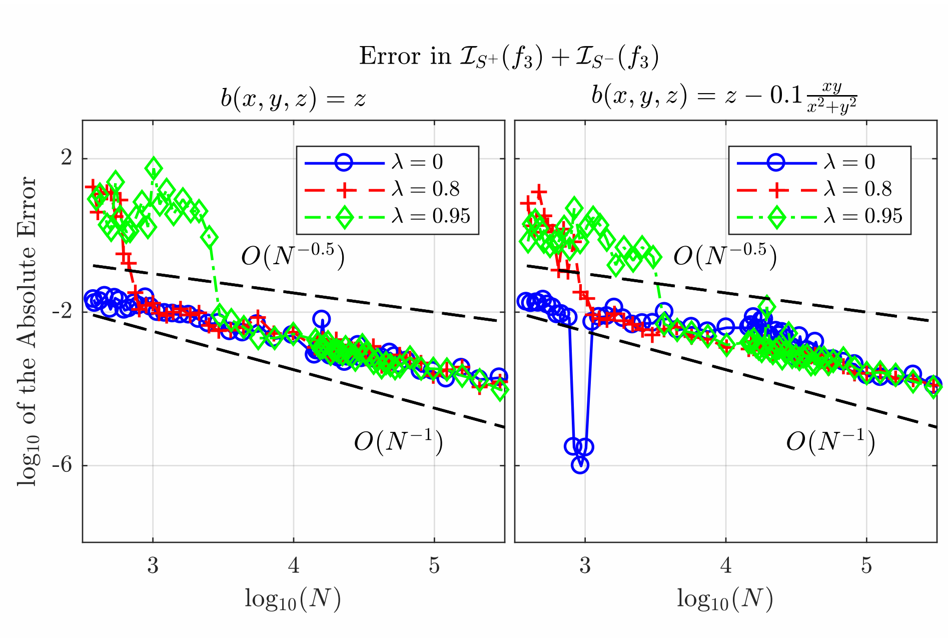

Figure 6: Maximum absolute error after 1000 random rotations about the x-axis when evaluating $\mathcal{I}_{S^+}(f_3) + \mathcal{I}_{S^-}(f_3)$ with the two different boundary surfaces $b(x, y, z)$.

of radius $\frac{1}{2\sqrt{\pi}}$ ($\lambda = 0$ in (20)). Changing the parameter ω varies the density of the nodes. In this study ω ranges from 0.005 to 0.1 in increments of 0.001. A comparison of $N = 1024$ clustered nodes against the same number of nearly uniformly spaced nodes appears in figure 7. The errors displayed in figure 8 highlight that clustering nodes near features of the integrand can lead to significant improvements on the order of accuracy of the approximate surface integral without degrading the convergence rate.

The methods developed in [3] and [4] both require $O(N\log N)$ time and $O(N)$ memory to construct a set of quadrature weights, even on node sets numbering in the millions. The computation time is $O(N)$ when the number of nodes is low enough so that the $O(N\log N)$ cost of finding nearest neighbors does not dominate. Figure 9 illustrates similar cost in computation time and memory when boundaries are introduced. Note, however, that there is the possibility of greater variability in computation time over different sets of nodes when boundaries are present due to the iterative method required to find the root of (12) and the initial value problem solvers required to solve (1), (13) or (14) (since adaptive solvers are recommended in this case). Since the method considers each curved triangle individually, the method is again embarrassingly parallel [3]. Further, if a GPU is available, the Matlab

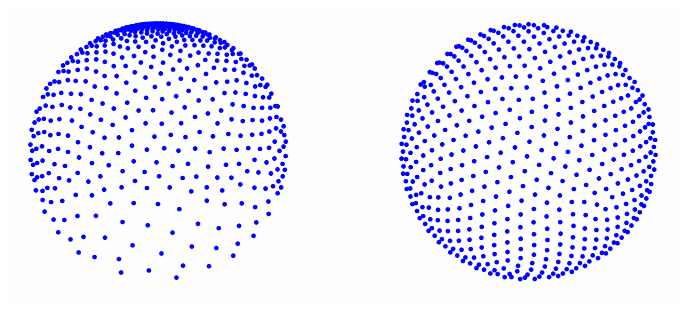

Figure 7: Left: 1024 nodes generated by a modification of the algorithm described in [12] with a node separation function $d(x, y, z) = \omega\sqrt{x^2 + y^2 + (z - 1.2)^2}$ that are scaled to a sphere of radius $\frac{1}{2\sqrt{\pi}}$ ($\lambda = 0$ in (20)). Right: 1024 Nearly uniformly spaced nodes on the sphere of radius $\frac{1}{2\sqrt{\pi}}$ ($\lambda = 0$ in (20)).

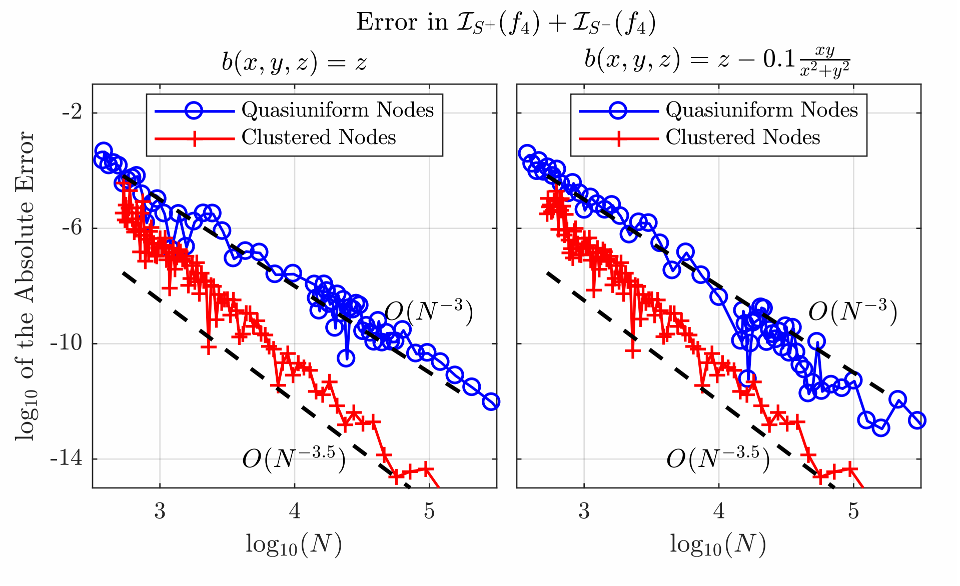

Figure 8: Maximum absolute error when evaluating $\mathcal{I}_{S^+}(f_4)+\mathcal{I}_{S^-}(f_4)$ with the two different boundary surfaces $b(x, y, z)$, but only for the choice of $\lambda = 0$ in (20).

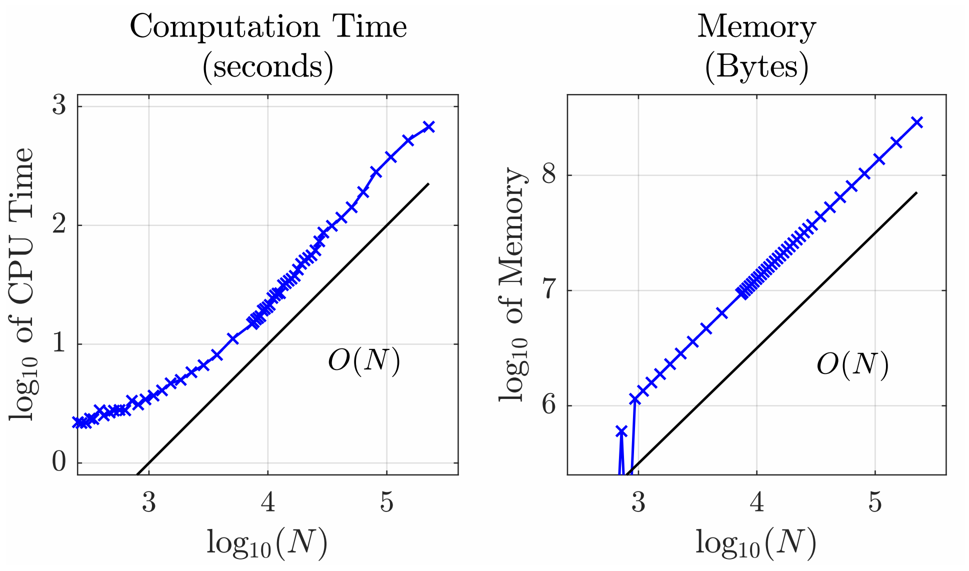

Figure 9: Left: CPU time (in seconds) to compute quadrature weights for evaluting $\mathcal{I}_S(f)$ where f is a scalar function. Right: Required memory (in Bytes) required to compute the quadrature weights. The computations in this figure were performed in Matlab on a laptop with 32 GB of DDR3 1500MHz memory and an Intel Core i7-4900MQ processor featuring four cores at 2.80 GHz.

function pagefun allows all of the linear systems to be solved at near peak GPU speeds.

The errors illustrated in figures 4, 5, 6 and 8 and the computation time and memory use illustrated in figure 9 are similar to those illustrated in [4] for varying N. To understand the effect of different choices of n and m on errors, CPU time and memory use, the reader should refer to [4], figure 4.

4 Test examples in planar (flat) bounded regions, focusing on boundary effects

To illustrate the effects that node placement and the choices of bivariate polynomial order, m, and number of nearest neighbors, n, can have on the present method consider the bounded planar region that is the unit square centered at the origin, i.e., the square with the vertices $\left(-\frac{1}{2}, -\frac{1}{2}\right)$, $\left(-\frac{1}{2}, \frac{1}{2}\right)$, $\left(\frac{1}{2}, \frac{1}{2}\right)$, and $\left(\frac{1}{2}, -\frac{1}{2}\right)$. Here $h(x, y, z) = z$ and $b(x, y, z)$ is defined piecewise by four planes whose normals lie in the xy-plane. Next consider the test integrand

$$f(\mathbf{x}) = \frac{1}{1 + 25((x+0.45)^2 + (y-0.4)^2)}, \tag{22}$$

(peaked near the top left corner of the domain). Also, consider three different node sets. The first are evenly spaced nodes on a lattice generated using Matlab's meshgrid command. The second are nearly uniformly spaced and are generated using distmesh2d [13] with equally spaced points on the boundary. The third node set is a subset of the Halton set in 2-dimensions supplemented by equally spaced points on the boundary [14]. Figure 10 displays the relative error when evaluating the integral of (22) over each triangle in T *individually* with m and N fixed and for three different choices of n. Notice that for $n = 36$ nearest neighbors, below which the matrix $\tilde{A}_k$ will be singular, the errors in the case of the meshgrid–generated nodes are many orders of magnitude larger than for the distmesh–generated and the Halton nodes. Also, in the case of the distmesh–generated nodes many of the triangles along the boundary exhibit larger errors. This is most apparent where the alignment of the triangles is much more regular along the top and bottom boundary. As n increases to 43, the large errors persist near the boundary in the case of the meshgrid–generated and distmesh–generated nodes, while the interior triangles in the meshgrid–generated case have errors comparable to the interior triangles of the other two node sets. It might seem that the boundary errors prohibit the use of this method in the presence of regularly aligned points near the boundary; however, if n is increased enough–to $n = 81$ in this case–the large errors on the boundary disappear. This is in agreement with the observations made in [9], that as the number of nearest neighbors (or stencil size) increase, the RBFs in (6) play a prominent role in mitigating the Runge-phenomenon (see also section 5).

This behavior is also visible in figure 11 which displays the relative errors when integrating (22) for fixed N and various choices of m and n over the entire unit square. Considering a fixed choice of m in the cases of meshgrid–generated or distmesh–generated nodes, errors decrease sharply when $n \gtrsim (m+1)(m+3)$, which is in agreement with the choice of $n = 81$ in figure 10. Because of this behavior near the boundary it is recommended to increase n near the boundary while using smaller stencil sizes further from the boundary to reduce computational cost.

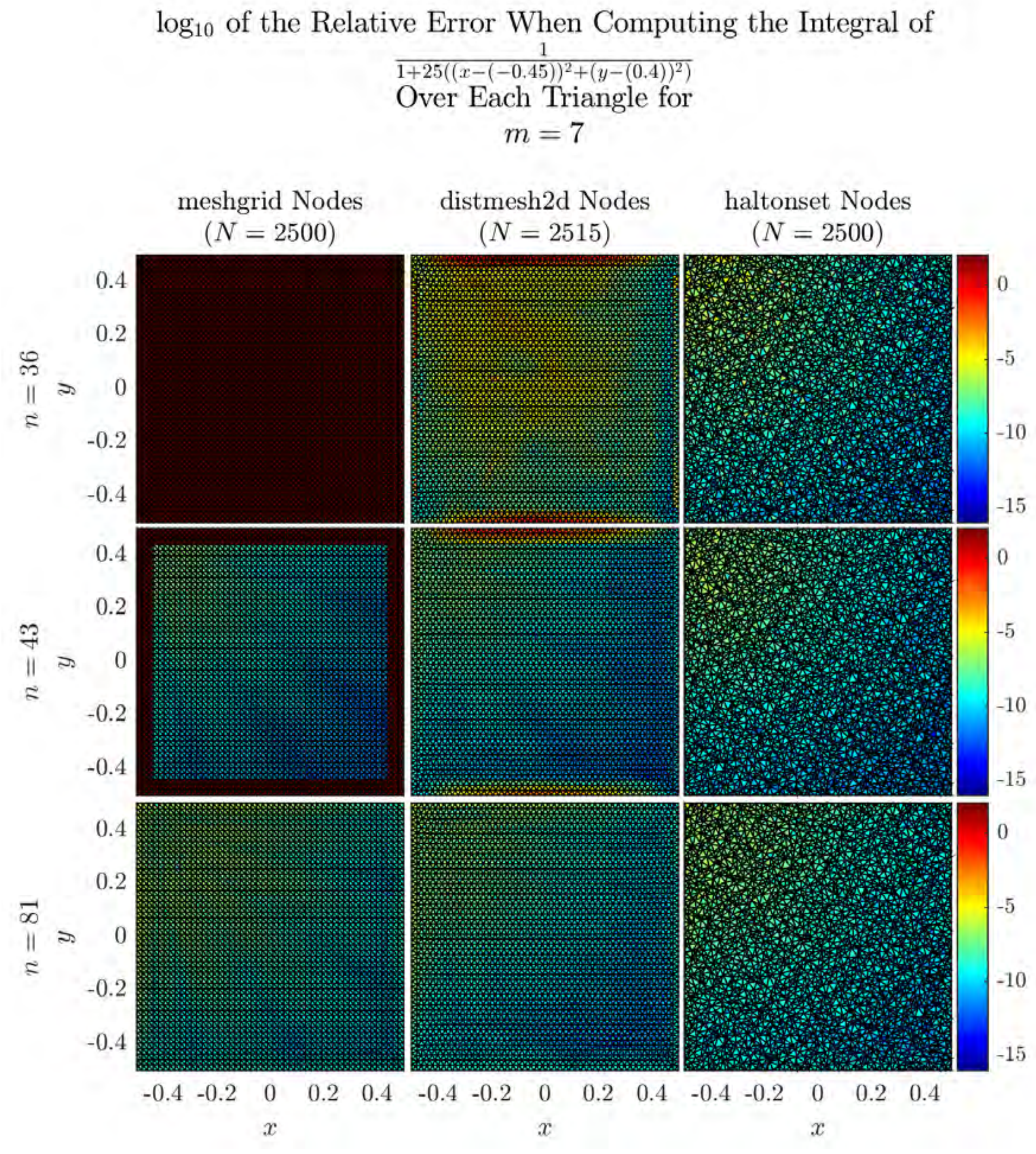

Figure 10: $\log_{10}$ of the relative error when evaluating the integral of (22) over the unit square for various node sets and choices of n. Notice that as n increases beyond $n = 81$, boundary errors in the presence of regularly spaced nodes decrease.

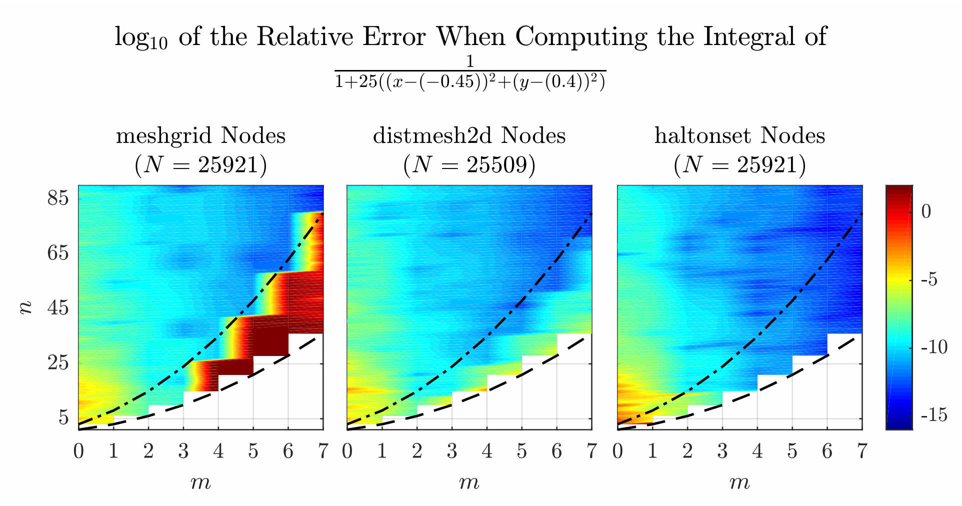

Figure 11: $\log_{10}$ of the relative error when evaluating the integral of (22) over the unit square for three different node sets. The lower dashed parabola in both plots represent the boundary $n = (m+1)(m+2)/2$ below which the linear system $\tilde{A}_k W_k = \tilde{I}_k$ becomes singular. The upper dashed–dotted parabola represents $n = (m+1)(m+3)$ above which boundaries no longer pose an issue. The roles of these parabolas are discussed also in [9], c.f. its figure 2.

5 1-D equispaced case: Comparison with Gregory's method

To better understand the observations of the previous section, concerning the number of nearest neighbors (stencil sizes) close to boundaries, this section considers the approach applied to the highly special case of equispaced nodes in 1-D. In this case, the method not only becomes essentially equivalent to the classical Gregory method [10], but it further improves on it in two ways:

1. The 1-D nodes need not be equispaced (this is not a novelty; such Gregory generalizations are discussed for a one order improvement of the trapezoidal rule in [15], in more conceptual form in [16], and for arbitrary orders of accuracy in [11]).
2. For increasing orders of accuracy, the Gregory weights become large and oscillatory, with some becoming negative beyond order 9. Inclusion of RBFs in the present approach avoids these adverse effects while maintaining the increased orders of accuracy.

5.1 Summary of the classical Gregory scheme for equispaced nodes in 1-D

For notational simplicity, consider the case of approximating $\int\limits_0^\infty f(x)dx$ with trapezoidal type approximations on the semi-infinite unit-spaced node set x_i, $i = 0, 1, 2, 3, \ldots$. Knowing that the dominant trapezoidal rule errors for smooth functions comes from the ends of the interval, it is natural to try to find correction terms for the ends. For the remainder, let the left end be at $x = 0$ and the right end far to the right.

With the notation $\Delta f(x_i) = f(x_{i+1}) - f(x_i)$, powers of the Δ operator can be represented as differences:

$$\begin{aligned}
\Delta^0 f(0) &= f(0) \\
\Delta^1 f(0) &= f(1) - f(0) \\
\Delta^2 f(0) &= f(2) - 2f(1) + f(0) \\
\Delta^3 f(0) &= f(3) - 3f(2) + 3f(1) - f(0) \\
\vdots \qquad & \qquad\qquad \vdots
\end{aligned}$$

The coefficients can be recognized from Pascal's triangle. Thus, consider the trapezoidal rule enhancement

$$\int_0^\infty f(x)dx = \sum_{i=0}^{\infty} f(i) + \left[b_0\Delta^0 + b_1\Delta^1 + b_2\Delta^2 + b_3\Delta^3 + \cdots\right] f(0) \qquad (23)$$

Substituting $f(x) = \mathrm{e}^{-zx}$ into (23)[3] gives, after a few quick simplifications,

$$\frac{1}{z} = \frac{1}{1-\mathrm{e}^{-z}} + \left[b_0 - b_1(1-\mathrm{e}^z) + b_2(1-\mathrm{e}^z)^2 + b_3(1-\mathrm{e}^z)^3 + \cdots\right].$$

Changing variables $w = 1 - e^z$, i.e. $z = -\log(1-w)$ gives

$$\frac{1}{\log(1-w)} + \frac{1}{w} = -b_0 + b_1 w - b_2 w^2 + b_3 w^3 - + \cdots ,$$

and the unknown coefficients can therefore be picked up by Taylor expanding the function

[3]A general function on $x \in [-\infty, \infty]$ can be thought of as a linear combination of Fourier modes $exp(-zx)$ with z purely imaginary. For a semi-infinite interval, the Laplace transform suggests allowing z to also have a real part. In either case, Taylor expanding errors around $z = 0$ gives accuracy orders.

$\frac{1}{\log(1-w)} + \frac{1}{w}$ around $w = 0$. This gives

$$b_0 = -\frac{1}{2}, b_1 = \frac{1}{12}, b_2 = -\frac{1}{24}, b_3 = \frac{19}{720}, b_4 = -\frac{3}{160}, b_5 = \frac{863}{60480}, b_6 = -\frac{275}{24192}, \cdots$$

For each coefficient that is included, the overall accuracy will increase with one power of h (in the case of using a spacing of h rather than just the special case of $h = 1$).

5.2 The sequence of Gregory weights

When converted to weights at node points, extending from the left boundary, the Gregory procedure gives for orders 2-10 the weight sets:

0.5000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
0.4167	1.0833	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
0.3750	1.1667	0.9583	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
0.3486	1.2458	0.8792	1.0264	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
0.3299	1.3208	0.7667	1.1014	0.9813	1.0000	1.0000	1.0000	1.0000	1.0000
0.3156	1.3922	0.6240	1.2441	0.9099	1.0143	1.0000	1.0000	1.0000	1.0000
0.3042	1.4604	0.4535	1.4714	0.7394	1.0825	0.9886	1.0000	1.0000	1.0000
0.2949	1.5259	0.2570	1.7989	0.4119	1.2790	0.9231	1.0094	1.0000	1.0000
0.2870	1.5890	0.0360	2.2409	-0.1406	1.7209	0.7021	1.0725	0.9921	1.0000

Table 1: Weights for the Gregory procedure for orders 2 (top row) to 10 (bottom row).

The top row amounts to the regular trapezoidal case. The bottom row shows the first instance of negative weights. In the comparison that follows, consider the weights in the next-to-last row

0.2949 1.5259 0.2570 1.7989 0.4119 1.2790 0.9231 1.0094 1.0000 1.0000

or in exact form

$$\frac{1070017}{3628800} \quad \frac{5537111}{3628800} \quad \frac{103613}{403200} \quad \frac{261115}{145152} \quad \frac{298951}{725760} \quad \frac{515677}{403200} \quad \frac{3349879}{3628800} \quad \frac{3662753}{3628800} \quad 1 \quad 1 \quad \cdots.$$

Figure 12 illustrates the data in table 1, with the top table row in front, then the next row behind, etc., finishing with the next-to-last table row. The bars that are colored white indicate trivial weights that are equal to one. In figure 12, where color is available, the front row displaying the nontrivial weight $\frac{1}{2}$ of the trapezoidal rule is indicated with a green bar and the nontrivial weights for the bottom row of table 1 are illustrated in red.

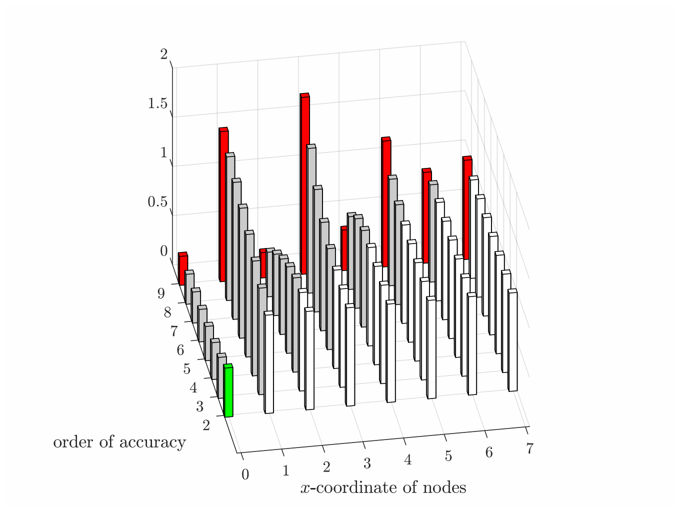

Figure 12: The Gregory weights near the left end of an interval, for increasing orders of accuracy $(2, 3, 4, \ldots, 9)$. Trivial weights (with the value one) are shown as white bars that also continue to the right of what is shown. Where color is available, the front row displaying the nontrivial weight $\frac{1}{2}$ of the trapezoidal rule is indicated with a green bar and the nontrivial weights for the bottom row of table 1 are illustrated in red.

It is graphically obvious that the entries for successive rows (orders of accuracy) become increasingly oscillatory.

5.2.1 Weights produced near the left boundary with the present RBF-FD approach

In the RBF-FD approach using $\phi(r) = r^l$ (l odd) or $\phi(r) = r^l \log r$ (l even) with polynomial terms, there are three parameters: n (the number of nodes in each stencil, i.e. the number of nearest neighbors), l (the power in the RBFs), and m (the degree of polynomials included in (6)). In the 1-D case of $n = m + 1$, the weights become purely polynomial based. Choosing as such an example $n = 8$, $l, m = 7$ produces the quadrature weight set

$$0.2942 \quad 1.5307 \quad 0.2425 \quad 1.8230 \quad 0.3878 \quad 1.2934 \quad 0.9183 \quad 1.0100 \quad 1.0000 \quad 1.0000$$

with exact form

$$\tfrac{278}{945} \quad \tfrac{185153}{120960} \quad \tfrac{3667}{15120} \quad \tfrac{8167}{4480} \quad \tfrac{733}{1890} \quad \tfrac{156451}{120960} \quad \tfrac{2777}{3024} \quad \tfrac{905}{896} \quad 1 \quad 1 \quad \cdots$$

This weight set differs from the Gregory coefficient set of the same width by (exactly)

$$-\frac{2497}{3628800} \times \left\{ \begin{array}{cccccccc} 1 & -7 & 21 & -35 & 35 & -21 & 7 & -1 \end{array} \right\}$$

This difference is a small coefficient times an approximation for $h^7 f^{(7)}(\xi)$, i.e. for smooth integrands, an $O(h^7)$-sized quantity. The present scheme is here constructed to integrate polynomials of up to degree 7 exactly. As discussed in [11], the Gregory scheme in this case will also integrate polynomials of degree 7 exactly. The small coefficient difference between the schemes involves no contradiction since, when integrating x^7, this difference term will cancel between the two ends of the interval.

The two approaches (Gregory and the present) are essentially equivalent for equivalent widths of end corrections in the purely 1-D equispaced node case.

5.2.2 Advantages of the present scheme in the 1-D case

The back row in figure 12, with nontrivial weights displayed by red bars, indicates a shortcoming in case corrections to increasing high orders are desired. Not surprisingly (being purely polynomial-based), a version of the Runge phenomenon enters, and the weights start to grow increasingly large in magnitude. The present approach offers a remedy for this if one keeps m fixed while increasing n (as was noted earlier in [9] for the task of approximating derivatives near a boundary). The front row in figure 13, with nontrivial weights displayed as red bars, is the (essentially equivalent) counterpart to the last row in figure 12–very highly oscillatory. Just including more nodes while keeping the polynomial order fixed preserves the accuracy order but is nevertheless seen to rapidly eliminate the oscillations in the weights. The non-trivial weights revert quite closely back to the trapezoidal rule weights of $\{\frac{1}{2}, 1, 1, 1, \ldots\}$. The back row of figure 13 illustrates the nontrivial weights in green for the far less oscillatory case of $n = 20$. Present Gregory-based literature offers no counterpart to this type of high order oscillation-free end corrections.

6 Conclusions

The key novelty in the present study is that the authors have supplemented their previous RBF-FD based approach for smooth surfaces [4] with a strategy for incorporating boundaries. As is well-know from interpolation and numerical differentiation, errors are usually

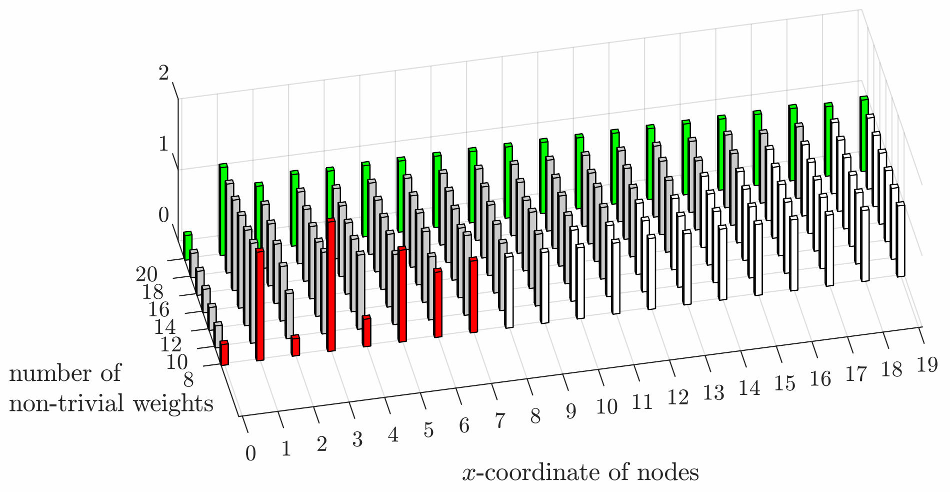

Figure 13: The non-trivial RBF-FD weights near the left end of an interval, for quadrature accuracy held fixed at $O(h^9)$ while increasing the number of nodes in each RBF-FD stencil (by including cubic RBFs). Trivial weights (with the value one) are shown as white bars that also continue to the right of what is shown. The front row, with nontrivial weights displayed as red bars, is the (essentially equivalent) counterpart to the last row in Figure 12–very highly oscillatory. The back row illustrates the nontrivial weights in green for the far less oscillatory case of $n = 20$.

larger near boundaries (where data is available from only one side, bordering on notoriously unstable extrapolation) than it is in domain interiors. The fact that the present method improves on the classical Gregory method in 1-D indicates that it is very effective in its boundary treatment. In fact, it does not seem to have been previously noted in the literature that quadrature methods can achieve high orders of accuracy near boundaries, without either local node clustering or weights becoming large and highly oscillatory.

References

[1] 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.

[2] 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.

[3] J. A. Reeger and B. Fornberg. Numerical quadrature over the surface of a sphere. *Stud. Appl. Math.*, 137:174–188, 2015.

[4] J. A. Reeger, B. Fornberg, and M. L. Watts. Numerical quadrature over smooth closed surfaces. *Proc. Royal Soc. A*, 472:20160401, 2016. (doi:10.1098/rspa.2016.0401).

[5] N. Flyer, G. A. Barnett, and L. J. Wicker. Enhancing finite differences with radial basis functions: Experiments on the Navier–Stokes equations. *J. Comput. Phys.*, 316:39–62, 2016.

[6] B. Fornberg and N. Flyer. Solving PDEs with radial basis functions. *Acta Numerica*, 24:215–258, 2015.

[7] B. Fornberg and N. Flyer. *A primer on radial basis functions with applications to the geosciences.* SIAM, Philadelphia, U.S., 2015.

[8] J. A. Reeger. Bounded_Smooth_Surface_Quadrature_RBF (2017). (https://www.mathworks.com/matlabcentral/fileexchange/63938-bounded-smooth-surface-quadrature-rbf), MATLAB Central File Exchange. Accessed 27 July 2017.

[9] V. Bayona, N. Flyer, B. Fornberg, and G. A. Barnett. On the role of polynomials in RBF-FD approximations: II. Numerical solution of elliptic PDEs. *J. Comput. Phys*, 332:257–273, 2017.

[10] J. Gregory. *Letter to J. Collins, 23 November 1670.* Oxford University Press, 1841. in Rigaud: Correspondence of Scientific Men.

[11] M. Javed and L.N. Trefethen. Euler-Maclaurin and Gregory interpolants. *Numer. Math.*, 132:201–216, 2016.

[12] B. Fornberg and N. Flyer. Fast generation of 2-D node distributions for mesh-free PDE discretizations. *Comp. Math. Applic.*, 69:531–544, 2015.

[13] P. Persson. Distmesh - a simple mesh generator in matlab. Website. http://persson.berkeley.edu/distmesh/. Accessed 29 June 2015.

[14] J. H. Halton. On the efficiency of certain quasi-random sequences of points in evaluation multi-dimensional integrals. *Numer. Math.*, 2(1):84–90, 1960.

[15] S.A. De Swardt and J.M. De Villiers. Gregory type quadrature based on quadratic nodal spline interpolation. *Numer. Math.*, 85:129–153, 2000.

[16] L.N. Trefethen and J.A.C. Weideman. The exponentially convergent trapezoidal rule. *SIAM Review*, 56:384–458, 2014.