Journal of Abstract and Computational Mathematics

http://www.ntmsci.com/jacm

Numerical solutions to two-dimensional integration problems

Alexander Carstairs and Valerie Miller

Mathematics and Statistics, Georgia State University Atlanta, Ga, United States of America

Received: 4 February 2017, Accepted: 10 June 2017
Published online: 3 December 2017.

Abstract: This paper presents numerical solutions to integration problems with bivariate integrands. Using equally spaced nodes in Adaptive Simpson's Rule as a base case, we look at two ways of sampling the domain over which the integration will take place. Drawing from Ouellette and Fiume, we first look at Voronoi sampling along both axes of integration and use the corresponding points as nodes for an unequally spaced Simpson's Rule. Then we look at triangulating the domain of integration and use the Triangular Prism Rules discussed by Limaye. Finally, we take all of these techniques and run simulations over heavily oscillatory and monomial (up to degree five) functions over polygonal regions.

Keywords: Delaunay Triangulation, Voronoi Sampling, Simpson Rule, Adaptive Simpson Rule, Quadrature.

1 Introduction

This paper presents two ways in which the one-dimensional Voronoi diagram-based sampling technique described by Fiume and Ouellette [9] can be expanded to solve two-dimensional integration problems. The first method is to perform the one-dimensional Voronoi sampling described in [9] along each axis of integration to get two sets of n points. Then the artesian product of those two sets is taken to create an $n \times n$ grid of nodes, which is then used in a quadrature rule. The second method will be to triangulate the domain of integration using a Delaunay triangulation. Both of these methods will be described in greater detail along with some relevant background theory of each in Section 1 and Section 2, respectively. The Voronoi sampling will be implemented in an adaptive Newton-Cotes method of degree two, and the Delaunay triangulation will be implemented in the midpoint, trapezoidal and Simpson's rules described in [3]. These methods, along with adaptive Simpson's rule and Monte Carlo integration, will be used to integrate a variety of test functions described in [14] and the set of monomials given in [8]. The results of these numerical simulations will be discussed in Section 3.

1.1 Voronoi sampling

In this section, the reader is introduced to the basics of Voronoi diagrams, and the following definitions are consistent with those given in [2] and [9].

Definition 1. *Let $S \subset \mathbb{R}^2$ be a set of points $x_1, x_2, \ldots, x_n$ for $n \geq 3$ and $p \in \mathbb{R}^2$ with $d(x_i, p)$ given as some metric. For any $x_i, x_j \in S$ and $i \neq j$, let*

$$B(x_i, x_j) = \{p \mid d(x_i, p) = d(x_j, p)\}$$

* Corresponding author e-mail: alex.carstairs@gmail.com; vmiller@gsu.edu

be the bisector of x_i and x_j, i.e. $B(x_i,x_j)$ is the perpendicular line through the center of the line segment connecting x_i and x_j. Thus, the bisector separates the halfplane

$$D(x_i,x_j) = \{p \mid d(x_i,p) < d(x_j,p)\}$$

containing x_i from the halfplane $D(x_j,x_i)$ containing x_j.

Using the halfplane described above, we now define the Voronoi diagram.

Definition 2. *Let the Voronoi region of x_i be given as*

$$V_i = V(x_i,S) = \bigcap_{x_j \in S, i \neq j} D(x_i,x_j)$$

with respect to S where V_i is an open set in the topological sense. Then the Voronoi Diagram of S is defined as

$$V(S) = \bigcup_{x_i,x_j \in S, i \neq j} \overline{V_i} \cap \overline{V_j}$$

where $\overline{V}$ is the closure of set V, i.e. the open set V unioned with its boundary.

Scaling the above definitions down to a one-dimensional diagram, we can create a sampling method that will iteratively select points on an interval $[a,b]$. First, let $x_1 = a$ and $x_2 = b$ and have x_3 be a randomly chosen number from the uniform distribution over (a,b). Then we can determine the next n points by constructing the Voronoi cells corresponding to the location of the sample points that already exist in the sequence. Let V_i represent the Voronoi cell of x_i and be defined as

$$V_i = \{x \in [a,b] \mid |x-x_j| \leq |x-x_i|, \text{ for all } j \in [1,n+3]\}$$

for $i \in [1,n+3]$. Let V_M be the longest line segment as defined above with ties being broken randomly. The next sample point in the sequence would the midpoint of the line segment corresponding to V_M. The abbreviation V_m is used to denote a Voronoi sampling sequence of n additional points where $m = n+3$.

1.2 One-Dimensional Newton-cotes quadrature

We now derive a generic one-dimensional quadrature method for integration over $[a,b]$. Given three arbitrary points a, m and b, where $a < m < b$, Lagrange interpolation is used to find a degree two polynomial, $p(x)$, to approximate our function $f(x)$.

$$\int_a^b f(x)dx \approx \int_a^b p(x)dx = \sum_{j=1}^{3} w_j f(x_j) \tag{1}$$

where

$$w_j = \int_a^b L_j(x)dx.$$

Instead of completing the derivation of Simpson's rule by exploiting the equal spacing of the points, we set up a general spacing of points as follows:

$$b-a = \alpha + \alpha c = \alpha(1+c)$$

$$b-m = \alpha$$

$$a-m = -c\alpha,$$

which simplifies the right hand side of Equation (1) to get

$$\sum_{j=1}^{3} f(x_i)L_j(x) = \frac{\alpha(1+c)}{6c}\left[(2c-1)f(a)+(1+c)^2 f(m)+c(2-c)f(b)\right].$$

This in turn gives us our quadrature formula

$$I(f) = \int_a^b f(x)dx \approx \frac{\alpha(1+c)}{6c}\left[(2c-1)f(a)+(1+c)^2 f(m)+c(2-c)f(b)\right] = I(p).$$

For the error, the equation

$$R(x) = I(f) - I(p) = \int_a^b \frac{f'''(\alpha)}{3!}(x-a)(x-m)(x-b)dx$$

is examined, which yields the total error $R(x)$ to be

$$R(x) = \frac{f'''(\alpha_1)}{6}\frac{(m-a)^3}{12}[2b-m-a] + \frac{f'''(\alpha_2)}{6}\frac{(b-m)^3}{12}[2a-m-b].$$

Assuming that f''' is essentially constant on $[a,b]$, then

$$R(x) \approx \frac{A}{72}(b-a)^3[2m-b-a],$$

so $R(x) = 0$ if $f(x)$ is a polynomial of degree ≤ 2. This is to be expected since quadratic interpolation is only guaranteed to be exact if $f(x)$ is of degree ≤ 2.

1.3 Delaunay triangulation

According to Aurenhammer and Klein [2], Voronoi was the first to consider the *dual* of the Voronoi diagram, but it was later determined by Delaunay as follows:

Definition 3. *Two point sites are connected if and only if the two sites lie on a circle whose interior contains no point in S where S is the set defined in definition 1.*

Using the definitions below, a given triangulation can be checked to see if it is Delaunay.

Definition 4. *A circumcircle is the circle that passes through the endpoints x_i and x_j for the edge x_ix_j and endpoints x_i, x_j and x_k of triangle $x_ix_jx_k$ for all combinations of i, j and k.*

Definition 5. *Let T be a triangulation with m triangles and a set of n points S where each element of S is a vertex of a triangle $t_i \in T$ for $i = 1,\ldots,m$. T is considered Delaunay if and only if the circumcircle of every t_i contains no other vertex in S.*

Both definitions 3 and 5 are known as the Delaunay criterion or empty circle property for edges (3) and triangles (5), and are implemented in several algorithms used for creating Delaunay triangulations. The advantage of using a Delaunay triangulation is that it maximizes the minimum angle of all of the triangles within the triangulation of a given set of points, which helps avoid skinny triangles. As the number of triangles increases, the triangles appear more uniform in size. This reduces the risk of peaks of functions from being cut off by large skinny triangles, which improves the stability of the calculations performed on the mesh.

2 Algorithms

There are three types of algorithms used in the construction of Delaunay triangulations: incremental insertion algorithms, divide-and-conquer techniques, and a sweepline techniques. The simplest are the incremental insertion algorithms, and they can be expanded to be used in higher dimensions easily. The algorithms that use the divide-and-conquer or sweepline techniques are faster than the incremental insertion techniques in two dimensions but are difficult to generalize (if at all) to higher dimensions. To construct the Delaunay triangulation in this paper, two algorithms are combined: the method `dtris2` from the GEOMPACK package and the Bowyer-Watson algorithm. Both algorithms are incremental insertion algorithms, which means they maintain a Delaunay triangulation into which points are inserted [5]. First, `dtris2` is used to triangulate the set of initial points including the vertices along the boundary of integration and a randomly chosen point within the boundary. The centroid of the largest triangle is then inserted using the Bowyer-Watson algorithm. In the following sections, each algorithm is examined and shown how they are implemented into our integration problem.

2.1 Incremental insertion algorithms

The earliest incremental insertion algorithm was developed by Lawson [11] and is based on edge flips. When a vertex is inserted, the triangle that contains the new point is found, and the point is connected to the vertices of the containing triangle by inserting three new edges (if the new point falls on the edge of an existing triangle, the edge is deleted, and the point is connected to the four vertices of the containing quadrilateral by inserting four new edges). The edges are placed into a stack and are tested to determine if they pass the Delaunay criterion. If not, then an edge flip is performed to remove the non-Delaunay edge. With each flip two new edges are added to the stack, and the algorithm ends when the stack is empty yielding a globally Delaunay triangulation. In 1981, A. Bowyer and D. Watson simultaneously presented an algorithm that does not depend on the use of edge flips and can easily be generalized to arbitrary dimensionality [11]. Our implementation of the Bowyer-Watson algorithm is given below and starts with already having a Delaunay triangulation of n points with a new point, x_{n+1}, to be added.

(1) Determine which triangle contains x_{n+1}. Delete this triangle and add its neighbors to a stack.
(2) Pop a triangle off the stack and determine if the new point is within the circumcircle of the triangle. If yes, delete the triangle and add the neighboring triangles to the stack.
(3) Repeat 2 until stack is empty.
(4) Triangulate the deleted region (We use the method `dtris2`, which is discussed in the next section and our implementation is discussed in Section 3.).
(5) Inserting the triangulation from 4 into the space that was voided by the deleted triangles provides the new Delaunay triangulation.

The Bowyer-Watson algorithm can also be implemented from scratch with no preexisting triangulation. First, three points are chosen that create a bounding triangle that encloses all of the points that need to be triangulated. The algorithm as outlined above then follows. Once all of the points have been inserted, the bounding triangle is then deleted along with all of its connections to the inner triangulation.

As stated above, this algorithm easily generalizes to higher dimensions by replacing the triangles and circumcircles for tetrahedron and circumspheres, respectively [11].

In its simplest form, this algorithm is not robust against roundoff error, though. A degenerate case can develop in which two triangles have the same circumcircle, but only one of them is deleted due to roundoff error, and the triangle that is not deleted is between the new vertex and the triangle that was not deleted. This gives a cavity that is not empty, and the resulting triangulation of the cavity would be a “nonsensical” [11] triangulation. This problem can be avoided by using

Lawson's algorithm instead. Lawson's algorithm is not absolutely robust to roundoff error, but failures occur much more sparingly compared to the simplest Bowyer-Watson algorithm. The Bowyer-Watson algorithm can be implemented with a depth-first search of the containing triangle and will perform equally as robust as Lawson's algorithm. Another advantage of Lawson's is that it is slightly easier to implement due in part because the topological structure maintained throughout the process stays a triangulation [11]. The Bowyer-Watson was chosen due to the nice pairing that it has with the method `dtris2` given below. The method keeps track of a neighbor matrix, which virtually negates the search time for the triangles that are effected by the new insertion point. The time complexity is discussed in further detail in Section 2.3.

There are other methods that can be used to create Delaunay triangulations such as divide-and-conquer and sweepline techniques. Both can be implemented in $\mathcal{O}(n \log n)$, and can be more robust than Bowyer-Watson in two dimensions, but given our application, they each have two limitations. First, they required a set of predetermined set of points, which would mean the entire set of points would be triangulated after every new point. A possible solution would be to use one of the techniques to triangulate the subset of affected triangles after the new point is inserted. Given that this subset is always relatively small, there was no added benefit to using either of the approaches given their intricacy. Second, both also suffer from the *curse of dimensionality*. Given these two limitations, Bowyer-Watson was chosen.

2.2 Incremental Delaunay triangulation algorithm with edge flips

The method `dtris2` as described by Joe [5] is a variation of the algorithm given by Sloan [12]. Sloan's algorithm combines the techniques from the Bowyer-Watson and Lawson algorithms. First, a super triangle is created that encompasses all of the points to be triangulated. Then a point P is inserted into the triangulation. The triangle that contains P is found, and P is connected to the three vertices of the containing triangle to create three new triangles. The Lawson flip algorithm is then used to make sure the triangulation is still Delaunay. This process is repeated until all points have been inserted [12]. Joe uses the same outline for `dtris2` but disregards the initial bounding triangle and initially sorts the points lexicographically [5]. The algorithm in `dtris2` is outlined below by starting with a set of points S that needs to be triangulated.

(1) Sort S using an ascending indexed heap sort to obtain the sorted set of indices S_s.
(2) Take the first three points according to S_s and create the first triangle.
(3) Add the next point according to S_s and connect the new vertex to the vertices that are "visible" to the new point.
(4) Check to make sure the new triangles are Delaunay. If not, perform edge swaps until all triangles are Delaunay.
(5) Repeat steps 3 and 4 until all points have been added to the triangulation.

A crucial component of this algorithm is that edge swapping guarantees the new triangles created are both Delaunay. Welzl [13] provides the following proposition and proof that guarantees any four points that are not cocircular have exactly one Delaunay triangulation.

Proposition 1. *Given a set $P \subset \mathbb{R}^2$ of four points that are in convex position but not cocircular. Then P has exactly one Delaunay triangulation.*

Proof. Let $P = pqrs$ be a convex polygon. There are two triangulations of P: a triangulation $\mathcal{T}_1$ using the edge pr and a triangulation $\mathcal{T}_2$ using edge qs. Now consider the family $\mathcal{C}_1$ of circles through the edge pr, which contains the circumcircles $C_1 = pqr$ and $C_1' = rsp$ of the triangles in $\mathcal{T}_1$. By assumption s is not on C_1. If s is outside of C_1, then q is outside of C_1'. Consider the process of continuously moving from C_1 to C_1' in $\mathcal{C}_1$ then point q is "left behind" immediately when going beyond C_1 and only the final circle C_1' "grabs" the point s.

Similarly, consider the family $\mathcal{C}_2$ of circles through pq, which contains the circumcircles $C_1 = pqr$ and $C_2 = spq$, the

latter belonging to a triangle in $\mathcal{T}_2$. As s is outside of C_1, it follows that r is inside C_2. Consider the process of continuously moving from C_1 to C_2 in $\mathcal{C}_2$ (right image in Figure **??**). The point r is on C_1 and remains within the circle all the way up to C_2. This shows that $\mathcal{T}_1$ is Delaunay, where as $\mathcal{T}_2$ is not.

The case that s is located inside C_1 is symmetric; just cyclically shift the roles of *pqrs* to *qrsp*.

2.3 Complexity

Now the time complexity of the `dtris2` method and the Bowyer-Watson algorithm are analyzed. Let T be the time it takes to triangulate a set of n additional points be given as

$$T = \sum_{k=1}^{n} T_k + S_k$$

where T_k is the amount of time it takes to find the triangle that contains the new point and S_k the time it takes to find all of the triangles in the cavity. Since the neighbor relations are given in `dtris2`, $S_k = \mathcal{O}(1)$ because it is proportional to the number of triangles in the cavity not the number of points. The reason for this is that the search for the neighboring triangle becomes obsolete as the number of points in the triangulation increases. Experimentally, the number of triangles in the cavity per iteration is less than ten, so as n increases the number of triangles affected stays essentially constant; thus, making T_k the dominating factor. In the worst case, the complexity of T_k is $\mathcal{O}(1)$, which gives us $\mathcal{O}(n^2)$ [1]. This worst case scenario happens when all existing triangles' circumcircles contain the new point at every point insertion. However, in the typical case, the number of triangles to be deleted at each point insertion does not depend on the number of existing triangles as described above. Combining an $\mathcal{O}(n \log n)$ multidimensional search for the triangle that contains the new point and the saved information of the neighbor relations between triangles, the Bowyer-Watson algorithm computes the Delaunay triangulation of n points in $\mathcal{O}(n \log n)$.

In [5], Joe discusses the time complexity of `dtris2` and determines it to be $\mathcal{O}(m \log m)$. Since the method is only used for the initial set of points and the vertices along the hull of the cavity along with the new point, then as n increases, m stays relatively constant and is significantly smaller than n. Thus, the time complexity of `dtris2` and Bowyer-Watson together is still $\mathcal{O}(n \log n)$.

Whenever implementing a geometric algorithm, two problems always need to be addressed: geometric degeneracies and numerical errors. For the Delaunay triangulation, four or more cocircular points will result in a non-unique triangulation [6]. In such a case, `dtris2` will produ ce the triangulation that comes first lexicographically. Another geometric degeneracy that needs to be considered is if three or more points are too close to being co-linear. In this case, `dtris2` checks for a "healthy" triangle when inserting a new point. It determines this by checking if the third point is to the left or right of a directed ray between the initial two points of the triangle. If the third point is within a certain tolerance of the directed ray, the method will break and return a fatal error.

Numerical errors are more difficult to handle. As discussed in Section 2.1, there is a degenerate case for roundoff error in the Bowyer-Watson algorithm. Mavriplis also discusses this issue with round-off error in [7]. In general, the nature of the problem will determine the accuracy requirements of the inputs and outputs. For our implementation, double-precision arithmetic is used, and it proves to be very robust. An occasional error occurs when inserting several thousand points in `dtris2` that appears to happen when two points are too close to each other, which illustrates the round-off error problems described in Section 2.1. Since the error rarely occurred, the iteration was simply skipped but noted during the simulation process using a try/catch block.

2.4 Delaunay integration

Now the triangulation of the integral domain is implemented in the triangular prism rules described in [3] to approximate the integral

$$\iint_D f(x,y)dxdy, \tag{2}$$

where D is the domain of integration that has been triangulated into n triangles. First, replace the function $f(x,y)$ above with a two variable polynomial function p_2 of total degree 2 that interpolates $f(x,y)$ at $p_1 = (\frac{x_1+x_2}{2}, \frac{y_1+y_2}{2})$, $p_2 = (\frac{x_2+x_3}{2}, \frac{y_2+y_3}{2})$, and $p_3 = (\frac{x_3+x_1}{2}, \frac{y_3+y_1}{2})$. Then the "signed volume" under the surface given by $p_2(x,y)$ is the "volume" of the paraboloidal triangular prism with its base D_i, the lengths of the 3 parallel edges equal to $f(x_1,y_1)$, $f(x_2,y_2)$, $f(x_3,y_3)$, and the heights at the midpoints of the sides are equal to the values of $f(x,y)$ at those midpoints. This gives us a "cubature"[3] rule in two variables that is analogous to Simpson's rule in one-variable and is given by

$$\iint_{D_i} f(x,y)dxdy \approx \frac{Area(D_i)}{3}\left[f(p_1)+f(p_2)+f(p_3)\right] \tag{3}$$

where $D_i \in D$ for all $i = 1,2,\ldots,n$. Although the 2D Simpson's rule is the main method in which our triangulation is implemented, the midpoint and trapezoidal equivalents described in [3] are also used for added performance comparison. For the midpoint rule, let (s,t) be the centroid of D_i and replace the function $f(x,y)$ from Equation 2 with the constant function $p_0(s,t)$. The "signed volume" under the surface given by p_0 is the "volume" of the triangular prism with the base D_i and height $f(s,t)$. This gives the "cubature" rule as

$$\iint_{D_i} f(x,y)dxdy \approx Area(D_i)\left[f(x_1,y_1)+f(x_2,y_2)+f(x_3,y_3)\right],$$

where $D_i \in D$ for all $i = 1,2,\ldots,n$ [3].

2.5 Monte Carlo integration

Monte Carlo methods are numerical methods that depend on taking random samples to approximate their results. Monte Carlo integration applies this process to the numerical estimation of integrals. The following estimator is given by Jarosz [4], but can be found in most sources discussing Monte Carlo methods. Suppose $f(x)$ is to be integrated over $[a,b]$:

$$F = \int_a^b f(x)dx.$$

and given a set of n uniform random variables $X_i \in [a,b)$ with corresponding PDF of $\frac{1}{b-a}$, then the Monte Carlo estimator for F is

$$\langle F^n \rangle = (b-a)\frac{1}{n-1}\sum_{i=0}^{n} f(X_i). \tag{4}$$

Using fundamental properties of random variables, it is easy to show that the expected value of the estimator is F. The proof can be found in [4]. For the one-dimensional case, the convergence rate is determined by looking at the convergence rate of the estimator's variance:

$$\sigma\left[\langle F^n \rangle\right] \propto \frac{1}{\sqrt{n}}.$$

Even though the convergence rate is slow compared to other one-dimensional techniques, it does not suffer from the *curse of dimensionality*. The estimator,$[\langle F^n \rangle]$, can easily be extended to multiple dimensions with the convergence remaining the same as above [4].

3 Implementation and numerical results

In this section the implementation of the algorithms and theory that was presented in the previous sections is given. Then test functions are defined, and the numerical results given by testing our implementations versus other known techniques discussed in Section 2 are also discussed. First, the adaptive Newton-Cotes quadrature discussed in Section 1.2 is tested against AS. Then AS is compared to Simpson's rule over the Delaunay triangulation described in equation (3). Finally, the performance of all of the techniques are compared to each other simultaneously.

3.1 Implementation

In Sections 2.1 and 2.2, the Bowyer-Watson algorithm and the method `dtris2` were investigated. These methods were combined to create a hybrid method that is based mainly on the Bowyer-Watson algorithm. Starting with an array, P, that contains the four points representing the vertices of the rectangular integration domain and one randomly chosen point within the rectangle, the initial triangulation is constructed using the `dtris2` method. From this initial triangulation, `dtris2` produces three outputs: the number of triangles, a matrix containing the vertices of each triangle, and another matrix containing the neighbor relations of the triangles. The columns of each matrix refers to a triangle in the triangulation, i.e. column one refers to triangle T_1, column two refers to triangle T_2, etc.

The centroid of the largest triangle is found, and the affected region is found by determining which triangles' circumcircles contain the new point. The boundary of the affected region is then stored in a temporary matrix with the newly inserted point. The region is then triangulated using `dtris2`. If the region is concave then the triangles that are created from bridging the concave vertices are deleted. If the region is convex then the triangulation is correct, and there is no need to delete any triangles.

Now that the affected region is triangulated, it needs to inserted back into the main triangulation. To do this, the referencing between the neighbor and vertex matrices of the affected region need to be inserted into the neighbor and vertex matrix for the main triangulation. First the point references in the vertex matrix are corrected. When triangulating the affected region with m points, `dtris2` labels the points $1,2,\ldots,m$ so the references need to be changed to their original numbering from P that consists of n points. Similar to the vertex matrix for the affected region, the neighbor matrix is also updated to reflect the numbering of the whole triangulation. While correcting the numbering for the overall neighbor matrix, the entries that are boundary edges for the affected region are set to 0 if they are not a boundary edge for the entire triangulation. If they are a boundary edge for both the affected region and larger triangulation, then the entry remains the same. Then the vertex and neighbor matrices for the affected region are merged with vertex and neighbor matrices corresponding to the overall triangulation. This is done by first noticing that if m triangles were affected, then the new triangulation consists of $m+2$ triangles, so each column in the overall neighbor matrix is filled with one from the affected region's triangulation, and the two extra triangles are placed onto the end. The corresponding columns in the vertex matrix for the affected region are added in the same manner to the over all vertex matrix. The vertex matrix is now complete and describes the triangulation with the new point added. Lastly, all of the zeros in the overall neighbor matrix are changed to their correct triangle references, and all of the negative entries are updated as well. The triangulation is now complete with correct vertex and neighbor matrices.

The algorithm described above is used in the cubature rules described in Section 2.4. The implementation of each of the cubature rules follows the same general outline with the only difference being at what points the function is being evaluated as given by each rule. First, an initial triangulation is found using `dtris2` along with the areas of each triangle. The areas are then placed in an array in increasing order, and a matrix containing the boundary edge information is also created. Then the functions is "integrated" using one of the three cubature rules described in [3] (Simpson's,

midpoint and trapezoid). The error is then calculated to see if it is within the given tolerance. If the volume is not within the given tolerance then the triangle with largest area is selected for refinement, and its centroid is computed. This point becomes the new point to be inserted and the algorithm above is run to determine the new triangulation. After triangulating, the areas of the triangles affected during the triangulation are deleted, and the new ones are calculated and sorted in ascending order. The two arrays of areas are then merged together. This process repeats until the volume is within the given tolerance or a maximum number of iterations is reached. This method involving the cubature rules is not implemented adaptively, so the error at each step is compared to the previous step. However, the Voronoi Newton-Cotes (VNC) method is implemented adaptively similar to our base case of the two-dimensional adaptive Simpson's (AS) Rule.

3.2 Integrands

In [14], Yu and Sheu examine solving the following double integral using the Mean-Value theorem for integrals:

$$\int_0^{2\pi}\int_0^R f(r,\theta,s,\phi,n)drd\theta,$$

where $s,\phi,R\in\mathbb{R}$, $R>0$, $n\in\mathbb{Z}^+$ and $f(r,\theta,s,\phi,n)$ is one of the following functions:

$$A(r,\theta,s,\phi,n)=r\exp\left[\sum_{k=0}^{n}\binom{n}{k}s^{n-k}r^k\cos\left[(n-k)\phi+k\theta\right]\right]\cos\left[\sum_{k=0}^{n}\binom{n}{k}s^{n-k}r^k\sin\left[(n-k)\phi+k\theta\right]\right] \tag{5}$$

$$B(r,\theta,s,\phi,n)=r\exp\left[\sum_{k=0}^{n}\binom{n}{k}s^{n-k}r^k\cos\left[(n-k)\phi+k\theta\right]\right]\sin\left[\sum_{k=0}^{n}\binom{n}{k}s^{n-k}r^k\sin\left[(n-k)\phi+k\theta\right]\right] \tag{6}$$

$$C(r,\theta,s,\phi,n)=r\sin\left[\sum_{k=0}^{n}\binom{n}{k}s^{n-k}r^k\cos\left[(n-k)\phi+k\theta\right]\right]\cosh\left[\sum_{k=0}^{n}\binom{n}{k}s^{n-k}r^k\sin\left[(n-k)\phi+k\theta\right]\right] \tag{7}$$

$$D(r,\theta,s,\phi,n)=r\cos\left[\sum_{k=0}^{n}\binom{n}{k}s^{n-k}r^k\cos\left[(n-k)\phi+k\theta\right]\right]\sinh\left[\sum_{k=0}^{n}\binom{n}{k}s^{n-k}r^k\sin\left[(n-k)\phi+k\theta\right]\right] \tag{8}$$

$$E(r,\theta,s,\phi,n)=r\cos\left[\sum_{k=0}^{n}\binom{n}{k}s^{n-k}r^k\cos\left[(n-k)\phi+k\theta\right]\right]\cosh\left[\sum_{k=0}^{n}\binom{n}{k}s^{n-k}r^k\sin\left[(n-k)\phi+k\theta\right]\right] \tag{9}$$

$$F(r,\theta,s,\phi,n)=r\sin\left[\sum_{k=0}^{n}\binom{n}{k}s^{n-k}r^k\cos\left[(n-k)\phi+k\theta\right]\right]\sinh\left[\sum_{k=0}^{n}\binom{n}{k}s^{n-k}r^k\sin\left[(n-k)\phi+k\theta\right]\right]. \tag{10}$$

Even though n can be an any integer such that $n\geq 1$, it is only chosen to be between 1 and 3. When n is increased, the results would become quite large ($\geq 10^6$) most of the time, which made it more difficult to get a good graph and harder to determine what could be causing the inaccuracies. The methods were also tested on functions of the form

$$\int_c^d\int_a^b x^i y^j dxdy$$

where $a,b,c,d\in\mathbb{R}$, $i,j\in\mathbb{Z}$ and $i+j\leq 5$. An analytical solution for each $f(r,\theta,s,\phi,n)$ is provided by Yu and Sheu, so the relative errors of each trial could easily be calculated. Similarly, the analytical solutions for the monomials can be found, so the relative error could easily be calculated.

3.3 Results

The VNC method is initially tested against AS on the set of monomials described above. During the first several runs, the Voronoi sampling only chose 3 additional points between our boundaries at each step similar to how Simpson's rule finds the three midpoints (quartiles) between the boundaries. For all of the simulations shown in the table, a tolerance of $\varepsilon = 0.0001$ is used, and the function is integrated over four randomly chosen points to create a rectangle with vertices $a = -0.00884120840760527$, $b = 2.71855632151155$, $c = 2.88900981641759$, and $d = 3.44868288240732$. The full results of one of the simulations are given in Table 11 in Appendix A. From Table 1, it is obvious that the VNC method is exact (within machine epsilon) for polynomials of degree two or less but is not always exact for the polynomials of degree three. As given in Equation 1.2, this is expected since the interpolating polynomial is only exact through degree two, and there is no additional degrees of accuracy since the error term is only proportional to $f^{(3)}$. Also, it is clear that AS is exact through degree three as expected as illustrated Table 1. For the higher degree (≥ 4) polynomials shown in Table 11, the

Table 1: Condensed Voronoi Newton-Cotes (VNC) v. Adaptive Simpson's Rule (AS) on Monomials with $a = -0.00884120840760527$, $b = 2.71855632151155$, $= 2.88900981641759$, $d = 3.44868288240732$ and $\varepsilon = 0.0001$.

i	j	AS Time	AS Rel. Error	VNC Time	VNC Rel. Error
0	0	0.008355225	1.45465E-16	0.007077289	0
0	1	0.000445179	1.83618E-16	0.000605689	1.83618E-16
0	2	0.000444923	1.15589E-16	0.000636664	1.15589E-16
0	3	0.000717303	1.45154E-16	0.028169894	4.20143E-08
1	0	0.000505850	0	0.000657144	0
1	1	0.000572153	1.35526E-16	0.000752119	2.71052E-16
1	2	0.000714487	3.41259E-16	0.000930037	3.41259E-16
2	0	0.000655608	1.18479E-16	0.000882165	1.18479E-16
2	1	0.000450042	4.48665E-16	0.000632312	1.49555E-16
3	0	0.000457722	1.16218E-16	0.498908685	8.96198E-08

VNC method performs adequately giving four additional orders of accuracy for the given epsilon in many cases. However, there are three cases that stand out: $f(x,y) = x^3y^2$, $f(x,y) = x^4y^1$ and $f(x,y) = x^5$, which are examined further in Table 2. Originally the maximum number of iterations was set to 5000, and all three of those cases reached the maximum number, so the maximum iterations was increased to see how many it would take to achieve a desirable accuracy. When increasing the maximum iterations to 15000, the following results were found and are shown in Table 2. The relative errors again give us an additional four or five digits of accuracy and even outperform AS on one of the runs. Unfortunately, the amount of time taken spiked drastically. Since the desired accuracy was finally achieved, the next step was to try to improve the speed of the method.

Table 2: 15000 Max Iteration Voronoi Newton-Cotes (VNC) v. Adaptive Simpson's Rule (AS) on Monomials with $a = -0.00884120840760527$, $b = 2.71855632151155$, $c = 2.88900981641759$, $d = 3.44868288240732$ and $\varepsilon = 0.0001$.

i	j	AS Time	AS Rel. Error	VNC Time	VNC Rel. Error	Iterations
3	2	0.000457210	0	3.063991277	1.30530E-08	6497
4	1	0.116197996	4.03868E-08	4.288516134	2.32547E-09	9069
5	0	0.211611352	7.10947E-08	7.329204530	3.50050E-08	14869

As stated above, the trials were initially run using the adaptive VNC with the Voronoi sampling only being used for three additional points along each axis. When examining the intermediate steps of both methods, the VNC had very long streaks of not adding any values to the total volume. This meant it was spending a lot of time finding an accurate enough approximation to be able to move on to the next quadrant. When looking at how the points were generally distributed

between the two values, there were large gaps on the tails of the interval giving large areas to approximate over on the ends, which would then need more refinement. Since AS uses the midpoints of each cell at every step, the empty space is filled much more evenly than with the Voronoi sampling; therefore, AS was always using significantly fewer iterations. In an effort to correct this, more points were sampled at each step (19 additional for a total of 21 with the endpoints) but would only use the first, second and third quartiles of the sampling. As shown in Table 3, the guaranteed accuracy through degree two for VNC and degree three for AS remains unchanged. In Table 12, the times for the larger degree polynomials did end up improving with the accuracy remaining roughly the same as before. Since increasing the number of points helped the speed of the algorithm and also gave us similar accuracy, the Voronoi sampling of 19 points over the three point method is used in the rest of the trials described in this paper. Even with the additional increase in speed, AS provides both better accuracy and speed on these simple functions overall, though. Now both methods are tested on more complicated functions.

Table 3: Condensed Voronoi Newton-Cotes (VNC) v. Adaptive Simpson's Rule (AS) on Monomials with additional sampling and $a = -0.00884120840760527$, $b = 2.71855632151155$, $c = 2.88900981641759$, $d = 3.44868288240732$ and $\varepsilon = 0.0001$.

i	j	AS Time	AS Rel. Error	VNC Time	VNC Rel. Error
0	0	0.001667055	0	0.005648840	1.45465E-16
0	1	0.000532987	0	0.001388274	3.67237E-16
0	2	0.000424188	2.31179E-16	0.001337843	0
0	3	0.000500475	1.45154E-16	0.022388000	4.90503E-08
1	0	0.000477947	0	0.001335283	4.29461E-16
1	1	0.000468731	1.35526E-16	0.001387250	0
1	2	0.000577018	0	0.001409522	1.70629E-16
2	0	0.000573178	2.36958E-16	0.001332467	4.73917E-16
2	1	0.000468475	2.99110E-16	0.001327091	1.49555E-16
3	0	0.000555770	1.16218E-16	0.432514594	4.00847E-08

Next tests were run on the functions described above from [14] using VNC and AS. For all of the simulations shown in Table 4, again a tolerance of $\varepsilon = 0.0001$ is used, and the function is integrated over the rectangle given by $a = 0$, $R = 5.480255137$, $c = 0$, $d = 2\pi$ with R being a randomly chosen point. We also randomly choose $s = 2.444171059$, $\phi = 5.69125859039527$ and $n = 1$. It is clear to see from Table 4 that neither AS nor the VNC method performs well on the six functions. These functions have fairly sharp high and low peaks and are also oscillatory. Simpson's rule is known to break down with oscillatory functions, e.g. integrating $|sin(x)|$ from $[0, 2\pi]$ using Simpson's rule arrives at an area of 0 when the true area should be 4. Considering most of the functions appear to have equally high and low peaks, a similar cancellation could be affecting the results. It is easy to see that the VNC method could also run into a similar problem given an appropriate function and "midpoint."

Table 4: Voronoi Newton-Cotes (VNC) v. Adaptive Simpson's Rule (AS) on Functions A-F with $a = 0$, $R = 5.480255137$, $c = 0$, $d = 2\pi$, $s = 2.444171059$, $\phi = 5.69125859039527$, $n = 1$ and $\varepsilon = 0.0001$.

f Type	AS Time	AS Rel. Error	VNC Time	VNC Rel. Error
A	11.8374406	2.729767377	16.11166813	1.182096082
B	12.04885046	1.418534859	16.38065796	0.842777814
C	12.13505427	1.611736343	16.61918614	1.067504856
D	13.03777242	2.358085297	17.13948942	0.634025761
E	12.37735201	2.191400672	18.07931543	0.806867337
F	12.78167719	1.697312935	17.24927674	0.899201229

Similar to our simulations comparing the methods on monomials, the results displayed in Table 4 have a maximum number of iterations of 5000. As before, the maximum number of iterations was increased, this time all the way to 20000, but this did not improve the accuracy by a significant amount (rarely getting even one additional order of accuracy). Ignoring the accuracy and looking at the times it took to complete the simulation, AS still trumps the Newton-Cotes method. Since both of the simulations are now executing the same number of iterations (5000), it is highly likely the time difference is attributed to the extra work the VNC has to do to perform the extra sampling.

Now that the VNC method and AS have been compared against each other, AS is now compared against the Simpson's rule analog using the Delaunay triangulation, which will be called Simpson's cubature (SC) rule, that was discussed in Section 2.4. Similar to the analysis performed above for the VNC and AS, the performances of AS and SC will first be compared over the monomials to make sure the expected guaranteed accuracies hold. Then they will be tested on the higher degree monomials. Both methods use a tolerance of $\varepsilon = 0.0001$ over the rectangle $a = -1.62110966800282$, $b = -1.37432067059289$, $c = -3.3239379751915$, $d = -1.72003265166653$. The full results of this simulation are given in Table 13 in Appendix A. Looking at Table 5, both AS and SC perform well through degree three and two polynomials, respectively. SC does not give us the extra third order of accuracy as shown earlier when comparing the VNC and AS, but it does yield an additional two to three extra orders of accuracy, which is still quite adequate. When looking at their differences in speed, SC is quite slow, even for functions for which it is exact, compared to AS. Similar to the VNC method, SC has to take the extra time to triangulate. The triangulation is slightly more time consuming than the Voronoi sampling, so this is why there is a larger gap on average between SC and AS than the gap between VNC and AS.

Table 5: Simpson's Cubature Rule (SC) v. Adaptive Simpson's Rule (AS) on Monomials with $a = -1.62110966800282$, $b = -1.37432067059289$, $c = -3.3239379751915$, $d = -1.72003265166653$ and $\varepsilon = 0.0001$.

i	j	SC Time	SC Rel. Error	AS Time	AS Rel. Error
0	0	0.016313693	2.80482E-16	0.000848376	2.80482E-16
0	1	0.005531337	7.78505E-16	0.000447996	6.67290E-16
0	2	0.006637502	5.11924E-16	0.000467707	5.11924E-16
0	3	0.025780223	2.19831E-05	0.000461051	2.54077E-16
1	0	0.005419978	1.87274E-16	0.000468731	1.87274E-16
1	1	0.005155789	1.48513E-16	0.000481531	1.48513E-16
1	2	0.009143973	3.05773E-06	0.000441084	6.83606E-16
2	0	0.005003726	4.99029E-16	0.000488443	7.48543E-16
2	1	0.005474249	2.78244E-06	0.000503547	3.95743E-16
3	0	0.004566226	1.31798E-07	0.000488187	0

Looking at the higher degree (≥ 4) polynomials in Table 13, SC performs about the same as it did for degree three in terms of accuracy. In the majority of cases, it gives at least one additional order of accuracy but does not perform as well as AS. There are a couple exceptions where they perform equally as well or the SC performs better such as $f(x,y) = x^4$ and $f(x,y) = x^5$. Since SC is always slower or the same speed as AS, it is easy to see that it does an adequate job, but AS outperforms it in all categories for the simple functions, which is to be expected. SC and AS are now compared on the functions A-F. Table 6 shows that even though AS gives much better accuracy, SC is either quicker or the same speed as AS. The graphs still have peaks and valleys like the previous example, but they are much less steep (only reaching as high as ten as opposed to several thousand in the previous example) and do not seem to be as oscillatory as the previous example. This definitely helps the accuracy of each method, but mainly helps AS. Over the many simulations run, it was noticed that the convergence for these functions is quite slow. Since SC is not implemented adaptively, the method will stop when the current iteration and the previous iteration are within ε of each other. These two facts combined lead to the method terminating too early, which also explains its performance in speed.

For the lower degree (≤ 3) polynomials given in Table 14, AS still remains the superior choice in both time and

Table 6: Simpson's Cubature Rule (SC) v. Adaptive Simpson's Rule (AS) on Functions A-F with $a = 0$, $R = 1$, $c = 0$, $d = 2\pi$, $s = 1.83468664481796$, $\phi = 5.71912370455419$, $n = 1$ and $\varepsilon = 0.0001$.

f Type	SC Time	SC Rel. Error	AS Time	AS Rel. Error
A	0.181542123	0.008030536	2.271972965	3.01322E-08
B	0.162510249	0.009020936	2.453636943	2.98341E-08
C	0.091698284	0.000826563	1.233137896	3.37518E-07
D	0.263396794	0.125639992	1.315914670	1.77746E-05
E	0.447086736	0.054360408	1.355982622	2.10402E-06
F	0.071517238	0.025073541	1.362870489	5.59632E-08

accuracy. The VNC method and SC perform giving several digits of accuracy and occasionally matching AS. As shown in Table 15, both still lag behind in speed, which is consistent with the analysis provided above. Now the methods are compared on monomials with degree ≥ 4. Two interesting cases are examined further with their results presented in Table 7 and Table 8 as well. When $f(x,y) = x^2y^2$, the VNC method and AS provide exact solutions with MC and SC performing respectably giving three digits of accuracy. Even though AS and the VNC provide the same level of accuracy, AS is roughly three times faster than the VNC, so AS is still a superior choice. On the other hand, VNC still provides better accuracy and speed than the other four methods. When $f(x,y) = x^5$, it is clear that with respect to accuracy AS, SC and MC perform the best with MDT and TDT performing about the same and VNC performing the worst. However, when reviewing each method's performance with respect to time, AS is the second worst performer with SC and MC performing the best. As discussed earlier with both AS and the VNC method, additional iterations can be expensive with respect to time. The VNC and AS methods are still capped at only 5000 iterations, but in this case, that amount is still too expensive. If needed, the iterations could be increased to gain more accuracy with AS, but given its performance on this example, the added digits of accuracy could be extremely expensive with time. The VNC method performs the worst in both accuracy and speed, which could be improved by increasing the number of sampled points to greater than 19, but based on previous results, this could marginally increase the accuracy but not improve the speed at all.

Lastly, all of the methods presented in the previous sections including Monte Carlo (MC) integration, midpoint rule, and

Table 7: Condensed accuracy only for Midpoint Delaunay triangulation (MDT), Trapezoid Delaunay triangulation (TDT), Simpson's cubature (SC), Adaptive Simpson's (AS), Voronoi Newton-Cotes (VNC) and Monte Carlo (MC) on Monomials with $a = 2.51778949114543$, $b = 5.67194769326589$, $c = -2.98410546965195$, $d = 5.22175955533465$ and $\varepsilon = 0.0001$.

i	j	MDT Rel. Error	TDT Rel. Error	SC Rel. Error	AS Rel. Error	VNC Rel. Error	MC Rel. Error
0	0	1.37263E-16	1.37263E-16	0	1.37263E-16	2.74525E-16	0
0	1	1.22684E-16	1.22684E-16	1.22684E-16	0	1.22684E-16	0.005362692
1	0	1.34083E-16	0	1.34083E-16	0	0	0.001456460
2	2	0.041796026	0.391609471	0.003020576	2.90959E-16	2.90959E-16	0.006233964
5	0	0.022003814	0.087866804	0.001089610	0.009776169	0.888547317	0.004548625

trapezoid rule are compared against one another. Tables 14 and 15 in Appendix A contain the accuracy and run times for each method. Table 7 confirms that the midpoint Delaunay triangulation (MDT) is accurate through constant functions (with a bonus of accuracy through degree one), and the trapezoidal Delaunay triangulation (TDT) method is accurate through degree one, but both of their accuracies dip significantly as the polynomials have higher degree.

The MC method is also shown in Table 7 and was run for 50000 iterations. This number proved to be large enough to give a competitive accuracy without taking a significant amount of time. Since the MC method is not a deterministic quadrature like the other methods, there is no guaranteed exactness for a specific degree (except when $f(x,y)$ is constant), so it does not perform as well with regards to accuracy for the lower degree polynomials when the other methods are exact. However, Table 7 illustrates that it still does a good job of approximating the functions and consistently provides three digits of accuracy.

For the lower degree (≤ 3) polynomials given in Table 14, AS still remains the superior choice in both time and

Table 8: Condensed time only for Midpoint Delaunay triangulation (MDT), Trapezoid Delaunay triangulation (TDT), Simpson's cubature (SC), Adaptive Simpson's (AS), Vorono Newton-Cotes (VNC) and Monte Carlo (MC) on Monomials with $a = 2.51778949114543, b = 5.67194769326589, c = -2.98410546965195, d = 5.22175955533465$ and $\varepsilon = 0.0001$.

i	j	MDT Time	TDT Time	SC Time	AS Time	VNC Time	MC Time
0	0	0.264221362	0.048522268	0.021801254	0.006768572	0.017867854	0.352805694
0	1	0.010175386	0.009185188	0.005659591	0.000578810	0.004228822	0.308191995
1	0	0.005873861	0.00499195	0.005412554	0.000433404	0.001324019	0.305286168
2	2	0.172706884	0.024697353	0.053885158	0.000398588	0.001289971	0.303935782
5	0	0.159462088	0.113784976	0.038072452	1.719201267	6.147717949	0.30707687

accuracy. The VNC method and SC perform giving several digits of accuracy and occasionally matching AS. As shown in Table 15, both still lag behind in speed, which is consistent with the analysis provided above. Now the methods are compared on monomials with degree ≥ 4. Two interesting cases are examined further with their results presented in Table 7 and Table 8 as well. When $f(x,y) = x^2y^2$, the VNC method and AS provide exact solutions with MC and SC performing respectably giving three digits of accuracy. Even though AS and the VNC provide the same level of accuracy, AS is roughly three times faster than the VNC, so AS is still a superior choice. On the other hand, VNC still provides better accuracy and speed than the other four methods. When $f(x,y) = x^5$, it is clear that with respect to accuracy AS, SC and MC perform the best with MDT and TDT performing about the same and VNC performing the worst. However, when reviewing each method's performance with respect to time, AS is the second worst performer with SC and MC performing the best. As discussed earlier with both AS and the VNC method, additional iterations can be expensive with respect to time. The VNC and AS methods are still capped at only 5000 iterations, but in this case, that amount is still too expensive. If needed, the iterations could be increased to gain more accuracy with AS, but given its performance on this example, the added digits of accuracy could be extremely expensive with time. The VNC method performs the worst in both accuracy and speed, which could be improved by increasing the number of sampled points to greater than 19, but based on previous results, this could marginally increase the accuracy but not improve the speed at all.

Lastly, each of the methods is compared over the functions A-F. As seen previously and now in Table 9, none of the methods provide much accuracy on the oscillatory functions. MC is a little bit of an exception since it provides two to three digits of accuracy for all but function A. From Table 9, AS and the VNC both perform extremely poorly with regards to accuracy and time. The functions again have fairly steep peaks and valleys with a couple of graphs maxing out in the low thousands. As discussed earlier, the high peaks and valleys that appear in all of the graphs could cause issues with the Newton-Cotes based methods.The maximum iterations could also be increased to greater than 5000, but this would only increase the run times of AS and VNC, which are already extremely long compared to the other methods as shown in Table 10.

Table 9: Accuracy only for Midpoint Delaunay triangulation (MDT), Trapezoid Delaunay triangulation (TDT), Simpson's cubature (SC), Adaptive Simpson's (AS), Vorono Newton-Cotes (VNC) and Monte Carlo (MC) on functions *A*-*F* with $a = 0$, $R = 4.310689426030381$, $c = 0$, $d = 2\pi$, $s = 2.35651382285138$, $\phi = 0.387434275655817$, $n = 1$ and $\varepsilon = 0.0001$.

f Type	MDT Rel. Error	TDT Rel. Error	SC Rel. Error	AS Rel. Error	VNC Rel. Error	MC Rel. Error
A	0.683714451	10.03800011	0.352786127	1.554339606	1.061044439	0.132485128
B	0.6990499	8.567750035	0.471139253	1.418966123	1.049486153	0.020218433
C	0.607982921	2.107507132	0.437277931	1.263018939	0.72749257	0.081858071
D	0.620570225	0.185445349	0.026569987	2.142429562	1.288711192	0.009542655
E	0.017144236	2.145806657	3.902967839	1.81321621	1.228863182	0.002194894
F	4.43223692	2.388462115	3.178847867	1.369362554	1.176581187	0.03489333

Table 10: Time only for Midpoint Delaunay triangulation (MDT), Trapezoid Delaunay triangulation (TDT), Simpson's cubature (SC), Adaptive Simpson's (AS), Vorono Newton-Cotes (VNC) and Monte Carlo (MC) on functions *A*-*F* with $a = 0$, $R = 4.310689426030381$, $c = 0$, $d = 2\pi$, $s = 2.35651382285138$, $\phi = 0.387434275655817$, $n = 1$ and $\varepsilon = 0.0001$.

f Type	MDT Time	TDT Time	SC Time	AS Time	VNC Time	MC Time
A	0.155683822	0.160757179	0.173916472	12.23576955	17.99852238	2.672148681
B	0.240922523	0.183981267	0.16088441	12.23258725	17.25799038	2.696071642
C	0.268317065	0.204237321	0.124068137	12.82676935	17.25403164	2.747159004
D	0.476965478	0.105453539	0.230742016	12.74109161	17.56966932	2.702983829
E	0.296183923	0.241752978	0.029039326	12.81275861	17.46210201	2.725147064
F	0.085615529	0.193632883	0.052914928	12.95576287	17.38970056	2.761395790

4 Conclusions and future work

This paper presents two methods for solving a numerical integration problem: a second degree Newton-Cotes method combined with a Voronoi sampling technique and using a Delaunay triangulation to divide the integration domain into triangles to integrate over. These two methods are compared to a midpoint and trapezoid rule over triangles, adaptive Simpson's rule and Monte Carlo integration. In 3 the results are presented and show that the Voronoi Newton-Cotes method and Delaunay triangulation Simpson's rule perform adequately on simple functions such as monomials, but neither performs nearly as well as adaptive Simpson's with regards to accuracy and speed. When comparing their performances over more complicated functions such as those found in the first part of Section 3.2, all of the methods perform poorly in accuracy and run time and are not viable methods for solving these problems. In the end, the Voronoi Newton-Cotes and Delaunay triangulation methods can provide adequately accurate results most of the time, but adaptive Simpson's is still more reliable in both accuracy and speed.

There are a couple improvements that could be made to the Simpson's rule with Delaunay triangulation. To improve the accuracy of the Delaunay cubature rule, it could be implemented adaptively by comparing locally instead of globally after each step. In the hybrid algorithm the `dtris2` method and the Bowyer-Watson algorithm are combined. Since the triangulation puts the method at a disadvantage compared to adaptive Simpson's rule, the algorithm could be improved to attempt to reduce the time taken to triangulate. To do this, the Bowyer-Watson algorithm could be implemented using an object-oriented language such as Java and create a data structure that could hold all of the information for the triangle such as its vertices, neighbors and centroid. This would eliminate the use of the `dtris2` method entirely, which could improve the run time of the triangulation. The next step would be to perform a similar analysis in higher dimensions to see how our particular method handles the *curse of dimensionality*.

Competing interests

The authors declare that they have no competing interests.

Authors' contributions

All authors have contributed to all parts of the article. All authors read and approved the final manuscript.

References

[1] C.A. Arens, *The Bowyer-Watson Algorithm: An efficient implementation in a database environment*, TU Delft, July 2002.
[2] F. Aurenhammer and R. Klein. "Voronoi Diagrams," *Handbook of Computational Geometry*, Ed. J.R. Sack and J. Urrutia, North Holland, 2000, pp. 203-292.
[3] S. R. Ghorpade and B. V. Limaye, *A Course in Multivariable Calculus and Analyis*, pp. 346-361, Springer, 2010.
[4] W. Jarosz, *Efficient Monte Carlo Methods for Light Transport in Scattering Media*, University of California, San Diego, 2008.
[5] B. Joe, GEOMPACK - A Software Package for the Generation of Meshes using Geometric Algorithms, *Advanced Engineering Software*, Vol. 13, No. 5/6, pp. 325-331, 1991.
[6] D. Lischinski, Incremental Delaunay Triangulation, *Graphics Gems IV*, pp. 47-59, 1994.
[7] D.J. Mavriplis, Front Delaunay Triangulation Algorithm Designed for Robustness, Institute for Computer Applications in Science and Engineering, ICASE Report No. 92-49, October 1992.
[8] S.E. Mousavi, H. Xiao and N. Sukumar, Generalized Gaussian Quadrature Rules on Arbitrary Polygons, *International Journal for Numerical Methods in Engineering*, Vol. 82, Issue 1, pp. 99-113, 2010.
[9] M. J. Ouellette and E. Fiume, On Numerical Solutions to One-Dimensional Integration Problems with Applications to Linear Light Sources, *ACM Transactions on Graphics*, Vol. 20, No. 4, pp. 232-279, 2001.
[10] S. Rebay, Efficient Unstructured Mesh Generation by Means of Delaunay Triangulation and Bowyer-Watson Algorithm, *Journal of Computational Physics*, Vol. 106, pp. 125-138, 1993.
[11] J. R. Shewchuk, Lecture Notes on Delaunay Mesh Generation, Department of Electrical Engineering and Computer Science, UC Berkeley, September 1999.
[12] S. W. Sloan, A fast algorithm for constructing Delaunay triangulations in the plane, *Adv. Eng. Software*, Vol. 9, pp. 34-55, 1987.
[13] E. Welzl, Lecture Notes, Chapter 6: Delaunay Triangulations, Department of Computer Science, Swiss Federal Institute of Technology Zurich 2013.
[14] C. Yu and S. Sheu, Using Mean Value Theorem to Solve Some Double Integrals, *Turkish Journal of Analysis and Number Theory*, Vol. 2, No. 3, pp. 75-79, 2014.
[15] H. Zimmer, Voronoi and Delaunay Techniques, *Proceedings of Lecture Notes, Computer Sciences*, No. 8, 2005.

A Full results tables

Table 11: Voronoi Newton-Cotes (VNC) v. Adaptive Simpson's Rule (AS) on Monomials with $a = -0.00884120840760527$, $b = 2.71855632151155$, $c = 2.88900981641759$, $d = 3.44868288240732$ and $\varepsilon = 0.0001$.

i	j	AS Time	AS Rel. Error	VNC Time	VNC Rel. Error
0	0	0.008355225	1.45465E-16	0.007077289	0
0	1	0.000445179	1.83618E-16	0.000605689	1.83618E-16
0	2	0.000444923	1.15589E-16	0.000636664	1.15589E-16
0	3	0.000717303	1.45154E-16	0.028169894	4.20143E-08
0	4	0.005256895	3.11878E-08	0.116979043	3.11815E-08
0	5	0.018198816	9.64697E-09	0.484475327	5.47195E-09
1	0	0.00050585	0	0.000657144	0
1	1	0.000572153	1.35526E-16	0.000752119	2.71052E-16
1	2	0.000714487	3.41259E-16	0.000930037	3.41259E-16
1	3	0.000505338	2.14273E-16	0.025437127	1.05478E-07
1	4	0.005495996	9.21119E-09	0.130504124	1.02836E-09
2	0	0.000655608	1.18479E-16	0.000882165	1.18479E-16
2	1	0.000450042	4.48665E-16	0.000632312	1.49555E-16
2	2	0.000574457	0	0.000780278	1.88292E-16
2	3	0.000786678	1.18226E-16	0.03968156	7.85136E-08
3	0	0.000457722	1.16218E-16	0.498908685	8.96198E-08
3	1	0.000620536	0	1.49217904	5.65809E-08
3	2	0.00045721	0	2.354864375	0.006500369
4	0	0.121101104	4.03868E-08	1.890572353	3.33763E-07
4	1	0.116197996	4.03868E-08	2.345452395	0.028265236
5	0	0.211611352	7.10947E-08	2.349108286	0.184686147

Table 12: Voronoi Newton-Cotes (VNC) v. Adaptive Simpson's Rule (AS) on Monomials with $a = -0.00884120840760527$, $b = 2.71855632151155$, $c = 2.88900981641759$, $d = 3.44868288240732$ and $\varepsilon = 0.0001$.

i	j	AS Time	AS Rel. Error	VNC Time	VNC Rel. Error
0	0	0.001667055	0	0.005648840	1.45465E-16
0	1	0.000532987	0	0.001388274	3.67237E-16
0	2	0.000424188	2.31179E-16	0.001337843	0
0	3	0.000500475	1.45154E-16	0.022388000	4.90503E-08
0	4	0.001801966	3.11878E-08	0.082919108	3.34052E-08
0	5	0.007804850	9.64697E-09	0.432353059	2.63289E-09
1	0	0.000477947	0	0.001335283	4.29461E-16
1	1	0.000468731	1.35526E-16	0.001387250	0
1	2	0.000577018	0	0.001409522	1.70629E-16
1	3	0.000557306	2.14273E-16	0.031654084	1.24042E-08
1	4	0.004736465	9.21119E-09	0.124463397	6.37093E-09
2	0	0.000573178	2.36958E-16	0.001332467	4.73917E-16
2	1	0.000468475	2.99110E-16	0.001327091	1.49555E-16
2	2	0.000460539	1.88292E-16	0.001327347	5.64876E-16
2	3	0.000472827	3.54679E-16	0.036043672	2.80358E-08
3	0	0.000555770	1.16218E-16	0.432514594	4.00847E-08
3	1	0.000464379	1.46700E-16	0.974934778	8.84301E-08
3	2	0.000470011	1.84698E-16	1.876273619	6.47114E-08
4	0	0.118157412	4.03868E-08	1.116981616	1.10403E-05
4	1	0.117699177	4.03868E-08	2.882673811	1.24821E-07
5	0	0.203343378	7.10947E-08	3.297244264	2.83738E-08

Table 13: Simpson's Cubature Rule (SC) v. Adaptive Simpson's Rule (AS) on Monomials with $a = -1.62110966800282$, $b = -1.37432067059289$, $c = -3.3239379751915$, $d = -1.72003265166653$ and $\varepsilon = 0.0001$.

i	j	SC Time	SC Rel. Error	AS Time	AS Rel. Error
0	0	0.016313693	2.80482E-16	0.000848376	2.80482E-16
0	1	0.005531337	7.78505E-16	0.000447996	6.67290E-16
0	2	0.006637502	5.11924E-16	0.000467707	5.11924E-16
0	3	0.025780223	2.19831E-05	0.000461051	2.54077E-16
0	4	0.005628104	0.000980107	0.008406700	2.76362E-07
0	5	0.025711359	0.000504705	0.030613710	7.71966E-08
1	0	0.005419978	1.87274E-16	0.000468731	1.87274E-16
1	1	0.005155789	1.48513E-16	0.000481531	1.48513E-16
1	2	0.009143973	3.05773E-06	0.000441084	6.83606E-16
1	3	0.027169265	0.000164893	0.000472827	1.69643E-16
1	4	0.024940551	0.000247897	0.031728579	1.72726E-08
2	0	0.005003726	4.99029E-16	0.000488443	7.48543E-16
2	1	0.005474249	2.78244E-06	0.000503547	3.95743E-16
2	2	0.004514515	3.36712E-05	0.000525819	1.06261E-15
2	3	0.009087397	0.000268450	0.000467451	7.91086E-16
3	0	0.004566226	1.31798E-07	0.000488187	0
3	1	0.006894267	1.19500E-05	0.000463611	0
3	2	0.012163975	1.11948E-05	0.000596730	2.01799E-16
4	0	0.005361354	4.93102E-07	0.000466939	3.78813E-07
4	1	0.006172866	3.78024E-05	0.000483579	3.78813E-07
5	0	0.005350859	7.47998E-07	0.000449276	1.87723E-06

Table 14: Accuracy only for Midpoint Delaunay triangulation (MDT), Trapezoid Delaunay triangulation (TDT), Simpson's cubature (SC), Adaptive Simpson's (AS), Vorono Newton-Cotes (VNC) and Monte Carlo (MC) on Monomials with $a = 2.51778949114543$, $b = 5.67194769326589$, $c = -2.98410546965195$, $d = 5.22175955533465$ and $\varepsilon = 0.0001$.

i	j	MDT Rel. Error	TDT Rel. Error	SC Rel. Error	AS Rel. Error	VNC Rel. Error	MC Rel. Error
0	0	1.37263E-16	1.37263E-16	0	1.37263E-16	2.74525E-16	0
0	1	1.22684E-16	1.22684E-16	1.22684E-16	0	1.22684E-16	0.005362692
0	2	0.031254337	0.312565299	3.20000E-16	3.20000E-16	0	0.001030788
0	3	0.079913262	0.071087959	0.000673081	0	0.288846650	0.000739701
0	4	0.085085066	0.349330409	0.000541126	0.000170947	0.832750855	0.005001589
0	5	0.164348054	0.250576017	0.005319045	0.482681292	0.929758393	0.003924581
1	0	1.34083E-16	0	1.34083E-16	0	0	0.001456460
1	1	0.000343677	0.001646319	0	1.19842E-16	1.19842E-16	0.002003265
1	2	0.030968972	0.071611407	0.000628227	1.56293E-16	1.56293E-16	0.001598617
1	3	0.046836029	0.097188218	0.003659738	0	0.552294340	0.007234298
1	4	0.087317344	0.247612927	0.004180156	0.439828056	0.910577689	0.011503918
2	0	0.004460387	0.012879597	1.24805E-16	2.49611E-16	3.74416E-16	0.001587096
2	1	0.002454287	0.019768922	4.02520E-05	1.11550E-16	3.34651E-16	0.002064355
2	2	0.041796026	0.391609471	0.003020576	2.90959E-16	2.90959E-16	0.006233964
2	3	0.052590909	0.075861068	0.000576465	0	0.764040384	0.014456091
3	0	0.013892921	0.024434128	0.000445991	1.11416E-16	5.39003E-09	0.001646359
3	1	0.005542517	0.016485043	0.001896887	1.99165E-16	2.22670E-08	0.002018411
3	2	0.055330400	0.135658417	0.002939728	0	0.299282761	0.005762480
4	0	0.020762319	0.083940060	0.000124715	1.34399E-10	0.594516070	0.004245154
4	1	0.016455419	0.021512025	0.001381476	1.15998E-10	0.367237377	0.005712344
5	0	0.022003814	0.087866804	0.001089610	0.009776169	0.888547317	0.004548625

Table 15: Time only for Midpoint Delaunay triangulation (MDT), Trapezoid Delaunay triangulation (TDT), Simpson's cubature (SC), Adaptive Simpson's (AS), Vorono Newton-Cotes (VNC) and Monte Carlo (MC) on Monomials with $a = 2.51778949114543, b = 5.67194769326589, c = -2.98410546965195, d = 5.22175955533465$ and $\varepsilon = 0.0001$.

i	j	MDT Time	TDT Time	SC Time	AS Time	VNC Time	MC Time
0	0	0.264221362	0.048522268	0.021801254	0.006768572	0.017867854	0.352805694
0	1	0.010175386	0.009185188	0.005659591	0.000578810	0.004228822	0.308191995
0	2	0.212448951	0.038722941	0.004221398	0.000389372	0.001326323	0.304616223
0	3	0.069382987	0.323772511	0.068109400	0.000453883	6.1294469	0.305803539
0	4	0.108419526	0.111599014	0.054363361	1.694423275	6.034388905	0.305586709
0	5	0.076242439	0.180678132	0.028078824	1.599314592	5.991035482	0.305959953
1	0	0.005873861	0.00499195	0.005412554	0.000433404	0.001324019	0.305286168
1	1	0.112953752	0.094920268	0.004018392	0.000424956	0.001363698	0.300387145
1	2	0.168781675	0.225280823	0.028803296	0.000450556	0.001321203	0.303793959
1	3	0.094807373	0.209948624	0.052410613	0.000420092	6.003799002	0.30607208
1	4	0.106156252	0.135796660	0.079992545	1.650970269	5.999080969	0.305938450
2	0	0.089998973	0.106370010	0.004385236	0.000401660	0.001311475	0.355044391
2	1	0.189325214	0.094818125	0.057345731	0.000575738	0.001439474	0.303675688
2	2	0.172706884	0.024697353	0.053885158	0.000398588	0.001289971	0.303935782
2	3	0.129739761	0.298044512	0.071032123	0.000481275	5.997509913	0.307859966
3	0	0.087928978	0.188436391	0.019048514	0.000416508	3.934356623	0.303384875
3	1	0.160555453	0.220046699	0.032886456	0.000463099	5.764459568	0.305458710
3	2	0.162005423	0.195561824	0.100761618	0.000406780	5.976562154	0.323290212
4	0	0.178353931	0.080940248	0.004267989	0.483936032	6.037025678	0.313836483
4	1	0.115505535	0.355869727	0.046673454	0.454968130	5.989358442	0.309507822
5	0	0.159462088	0.113784976	0.038072452	1.719201267	6.147717949	0.30707687