An Adaptive Triangulation Method for Bivariate Spline Solutions of PDEs

Ming-Jun Lai and Clayton Mersmann

Abstract We report numerical performance of our adaptive triangulation algorithms to improve numerical solutions of PDEs using bivariate spline functions. Our ultimate goal is to find a PDE-solution-dependent triangulation which improves both the accuracy and computational efficiency of the spline solution. We present little theory to guide our search for such a triangulation, but instead approach the problem numerically. Starting with some initial triangulation $\triangle$, we use the gradient values of the spline solution based on $\triangle$ to generate an updated triangulation $\triangle'$ and compute a new spline solution. We consider both refining and coarsening the initial triangulation $\triangle$ in order to make the spline solution more effective. As we add vertices to and remove vertices from the vertex set, we use a global retriangulation instead of local refinement techniques. We introduce a new concept of mesh efficiency to measure the effectiveness of a spline solution over a given triangulation. Extensive numerical experiments have been conducted and are summarized in this paper. In addition, we report a heuristic for generating an initial solution-dependent triangulation and show numerical evidence that this algorithm produces an initial triangulation which yields a better spline solution than one based on a uniform initial mesh.

Keywords Bivariate splines · Adaptive triangulation · PDEs

This research is partially supported by Simons collaboration grant 280646 and the National Science Foundation under the grant #DMS 1521537.

M.-J. Lai (✉) · C. Mersmann
Department of Mathematics, University of Georgia, Athens, GA 30602, USA
e-mail: mjlai@uga.edu

C. Mersmann
e-mail: mersmann@uga.edu

G.E. Fasshauer and L.L. Schumaker (eds.), *Approximation Theory XV: San Antonio 2016*, Springer Proceedings in Mathematics & Statistics 201,
DOI 10.1007/978-3-319-59912-0_7

1 Introduction

Besides the standard popular finite element method (cf. [9]), there are many other approaches available to solve boundary value problems of partial differential equations over an arbitrary polygonal domain. Among these are the bivariate spline method in [4, 17], the discontinuous Galerkin method (cf. [3]), the virtual element method (cf. [7]), the weak Galerkin method (cf. [22]), and etc. In this paper, we are interested in how we can solve PDEs more effectively when using the bivariate spline functions proposed in [4].

For PDEs whose solutions are difficult to approximate, uniform refinements lead to triangulations with huge numbers of triangles and unwieldy spline solutions. For these types of problems, it is necessary to use a PDE-solution-dependent triangulation to solve the PDE more effectively. Traditionally, researchers use adaptive triangulation methods to increase the accuracy of the solution. We refer to [2, 21] for classic adaptive methods for finite element solutions which are based on *a posteriori* error estimates. See [8, 11] for convergence analysis of adaptive triangulation methods based on linear and higher order finite element approximation. Also, [14] performs adaptive computations using splines over triangulations with hanging nodes, developed in [18], for numerical solution of PDEs. The goal of our research is to construct a PDE-solution-dependent triangulation over which the spline solutions exhibit better numerical performance. Our goals for improved performance can be stated as follows: (1) fixing the number of triangles in a triangulation, we seek a mesh which generates more accurate solutions; (2) fixing a level of accuracy, we seek suitable spline solutions over a smaller triangulation (i.e., fewer triangles); or (3) a suitable combination of (1) and (2).

In classic adaptive triangulation schemes, an error function is computed from an initial numerical solution to estimate the places in the domain where large errors may occur. The triangulation is then refined only in those areas. One common way to identify which parts of the domain should be refined is to compute the residual $e := |f - \mathscr{L}(u_s)|$ over each triangle where u_s is an initial approximate solution and $\mathscr{L}$ is the partial differential operator. In [6], error indicators based on energy and $||\cdot||_{1,p}$ norms of the residual are derived for FEM solutions; the main result of [5] establishes an estimation of those norms using jumps in the gradients of the numerical solution across element boundaries. Another approach, as described in [1, 23, 24], utilizes some post-processing of an initial finite element solution to produce a "recovered" gradient estimate; error indicators are then based on differences between these recovered gradients and the gradient computed from the initial solution.

The adaptive triangulation algorithms we present in this paper are not based on rigorous error estimation, but are intended for straightforward application to spline functions with smoothness $r \geq 0$. For C^r splines with $r > 0$, the jump of the gradient across any interior edge is zero, so we are unable to use those discontinuities to compute an error function. Instead, we utilize the gradient of the approximate solution at the centroids of triangles as a simple indicator of the magnitude of the change of the solution. The larger the magnitude of the gradient, the more the solution changes

over the triangle. If the gradient is large, the centroid is added to the vertex set, and if the gradient is near zero, the vertices of the triangle are removed.

Our simple gradient indicator is easy to compute, and numerical results show that it works well when constructing approximate solutions of PDEs that change far more drastically in some parts of the domain than others. One difference of our approach from many traditional adaptive methods (but not all; cf [10]) is that we retriangulate globally by a Delaunay triangulation method instead of using local refinements. For any fixed vertex set, like the ones returned by our algorithms, the Delaunay triangulation is the one with the max–min angle property. This mitigates decrease of the smallest angle of $\triangle$ due to local refinements via bisection. The small angle problem can also be avoided by using H-triangulations, i.e., triangulations with hanging nodes [18], as implemented for *hp*-finite elements in [20] and for splines in S_d^0 in [14]. Our approach uses ordinary triangulations due to their ubiquity in spline applications and to avoid the added difficulty of computing C^r splines over H-triangulations for $r \geq 1$.

This paper is organized as follows. In Sect. 2, we review the bivariate spline functions used in [4]. In Sect. 3, we describe our algorithm and introduce a simple *mesh efficiency* metric (*me*) to help measure relative improvements in a triangulation's adaptedness to a given PDE. We then define the motivating problem and present the results of our study in the context of the Poisson equation. In Sect. 3.1, we adapt the original uniform mesh by adding the locations where large "gradient" values occur to the vertex set; this amounts to a kind of *h*-refinement. In Sect. 3.2, we present an algorithm that also removes vertices of $\triangle$ from the vertex set in areas where the solution is flat. In both sections, we present extensive numerical results to demonstrate that our approach offers significant improvement over a uniform refinement scheme.

In Sect. 3.3, we present an approach that uses information from the source function f to generate an initial triangulation. This mesh produces a better *initial* spline solution than one generated over a uniform mesh with the same number of triangles and might be considered as a kind of *r*-refinement. We determine an initial number of desired triangles and generate a nonuniform vertex set whose Delaunay triangulation meets that requirement. We then apply the algorithms from Sect. 3.2 to generate a mesh that is highly adapted to the given PDE.

Finally, we comment that our adaptive approach can be extended to deal with other partial differential equations, e.g., the biharmonic equation.

2 Spline Functions on Triangulations

Given a polygonal region Ω, a collection $\Delta := \{T_1, ..., T_n\}$ of triangles is an ordinary triangulation of Ω if $\Omega = \cup_{i=1}^n T_i$ and if any two triangles T_i, T_j intersect at most at a common vertex or a common edge. There are many different possible triangulations $\triangle_j$ on a given vertex set. In some sense, the best triangulation $\triangle$ is the one that maximizes the minimum angle $\theta(\triangle)$ of the set of triangles $\{T_i\} \in \triangle_j$; this is partly because error bounds for spline approximations often depend on the minimum angle

of a triangulation, but also because computation of spline coefficients can become numerically unstable if $\theta(\triangle)$ is too small. When Ω is convex, this $max - min$ angle triangulation is equivalent to the Delaunay triangulation of the region (cf. [13]).

For a triangle $T_i \in \Omega$, $T_i = (v_1, v_2, v_3)$, we define the barycentric coordinates (b_1, b_2, b_3) of a point $(x_o, y_o) \in \Omega$. These coordinates are the solution to the following system of equations

$$\begin{aligned} b_1 + b_2 + b_3 &= 1 \\ b_1 v_{1,x} + b_2 v_{2,x} + b_3 v_{3,x} &= x_o \\ b_1 v_{1,y} + b_2 v_{2,y} + b_3 v_{3,y} &= y_o, \end{aligned}$$

and are nonnegative if $(x_o, y_o) \in T_i$. The barycentric coordinates are then used to define the Bernstein polynomials of degree d:

$$B^T_{i,j,k}(x, y) := \frac{d!}{i!j!k!} b_1^i b_2^j b_3^k, \qquad i + j + k = d,$$

which form a basis for the space $\mathscr{P}_d$ of polynomials of degree d. Therefore, we can represent all $p \in \mathscr{P}_d$ in B-form:

$$p = \sum_{i+j+k=d} c_{ijk} B^T_{ijk}$$

where the B-coefficients c_{ijk} are uniquely determined by p.

We define the spline space $S^0_d := \{s \in C^0(\Omega) : s|_{T_i} \in \mathscr{P}_d\}$ where T_i is a triangle in a triangulation $\triangle$ of Ω. We use this piecewise continuous polynomial space to define the space $S^r_d := C^r(\Omega) \cap S^0_d(\triangle)$. The C^r condition for splines can be guaranteed across edges of an ordinary triangulation $\triangle$ by enforcing linear conditions on the B-coefficients of neighboring triangles.

Computations involving splines written in B-form can be performed recursively using the de Casteljau's algorithm. In fact, these spline functions have numerically stable, closed-form formulas for differentiation, integration, inner products (cf. [13]), and triple products. Spline functions in $S^r_d(\Omega)$ on ordinary triangulations have optimal approximation power if $d \geq 3r + 2$. In the following, we exhibit the flexibility of spline solutions to the Poisson problem by solving over $S^r_d(\Omega)$ for various choices of d and r. For a more complete discussion on the properties of bivariate splines in B-form, see [13] or [17]. The implementations in this paper are based on those discussed in [4]. In particular, we do not work with minimal determining sets, but instead impose smoothness conditions as side constraints. We also do not enforce supersmoothness conditions around vertices, as PDE solutions themselves may not exhibit this smoothness phenomenon at those points in the domain.

3 Numerical Results for Our Adaptive Approach

3.1 Part 1: Adding Triangles

To efficiently solve PDEs which change far more drastically in some areas of the domain than others, we should add more vertices only as needed, instead of refining uniformly. To decide where we need to add more vertices, we use the gradient of the spline solution based on an initial triangulation. We introduce our first algorithm as follows:

Algorithm 1 Input a triangulation $\triangle$ with vertex set V and triangle list T, and initial approximate spline solution s. Then do the following steps:

- compute centroids $\mathbf{c}_t$ of all triangles t in T;
- compute values of $D_t(s) = |D_x s| + |D_y s|$ at the location $\mathbf{c}_t$ for all $t \in T$;
- sort the values of D_t by size;
- choose the $\mathbf{c}_t$ corresponding to the largest D_t values above a certain threshold to be added to V;
- use a triangulation method (Delaunay triangulation) to find a new triangulation of the updated vertex set;
- output V and T of the resulting triangulation.

To help measure triangulation improvement in any setting, we introduce a simple *mesh efficiency* metric, or *me*, for a spline solution s over a triangulation $\triangle$, defined by

$$me(s) := \#(\triangle) \times \text{RMSE}(s)$$

where $\#(\triangle)$ is the number of triangles in the underlying triangulation $\triangle$, and RMSE(s) is the root-mean-squared error of spline solution, computed on a grid of 1001×1001 points spread evenly over the domain.

This metric helps measure relative improvements in a triangulation's adaptedness to a given PDE; a smaller *me* value represents an improvement in the triangulation size, approximation error, or both. Therefore, comparing reduction in *me* values from one mesh to another gives us a clear idea of how the triangulation is improving. For comparing mesh 1 to mesh 2, we refer to the percent reduction in *me* values from one mesh to the other as

$$\text{Net Gain} := 1 - me_2/me_1.$$

Example 1 Our study is motivated by the Poisson problem with exact solution $u(x, y) = \tanh(40y - 80x^2) - \tanh(40x - 80y^2)$ on $[0, 1] \times [0, 1]$; its graph is shown in Fig. 1. We first use our bivariate spline method to compute numerical solutions in S_7^0 for this particular problem over uniform type-1 triangulations of varying fineness. The initial triangulations shown in Fig. 2 are examples of this type of triangulation; in the following, we simply use "uniform triangulation" to mean uniform type-1 triangulation.

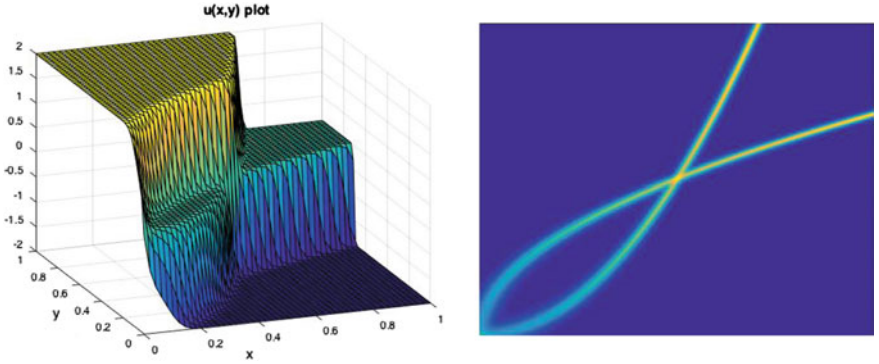


Fig. 1 A plot of the function $u(x, y) = \tanh(40y - 80x^2) - \tanh(40x - 80y^2)$ on $[0, 1] \times [0, 1]$ (*right*), and a color plot of its $|D_x u| + |D_y u|$ values (*left*). This function is hard to approximate based on a uniform triangulation

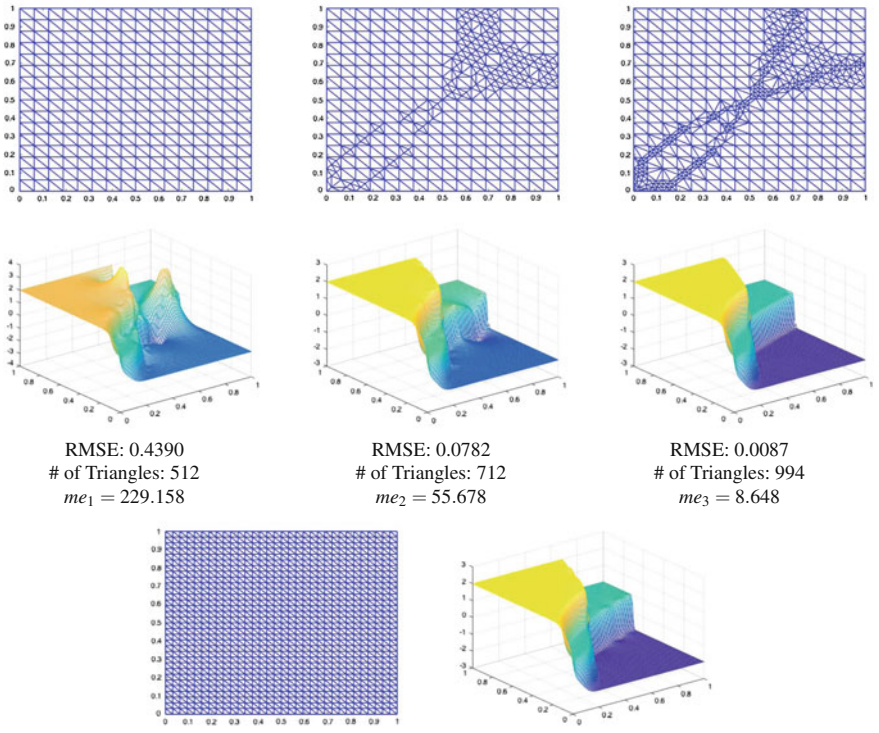


Fig. 2 Example 1: Here, we solve the Poisson problem on $[0, 1] \times [0, 1]$ with exact solution $u(x, y) = \tanh(40y - 80x^2) - \tanh(40x - 80y^2)$ over spline space S_7^0. A uniform triangulation (*top left*), a triangulation after one iteration of Algorithm 1 (*top middle*), and a triangulation after a second iteration of the algorithm (*top right*). In the *bottom row*, we include the triangulation and solution that result from a uniform refinement of the original mesh; our adapted triangulation generates a more accurate solution even though it uses less than half the number of triangles as the uniformly refined mesh

Table 1 RMSE results for 4 uniform meshes of varying fineness; Numerical results are based on spline spaces $S_7^0(\triangle)$

Number of triangles	RMSE	Number of spline coefficients
800	1.92e-1	28,800
1152	5.17e-2	41,472
1568	2.96e-2	56,488
2048	1.34e-2	73,728

From Table 1, we can see that one has to use quite a large uniform triangulation to get a reasonably accurate numerical solution. This makes it an ideal candidate for application of Algorithm 1.

Based on the implementation in [4], we present numerical data for spline solutions in the space S_7^0 to this particular Poisson problem. Here, we apply Algorithm 1 twice; the process and the resulting meshes are shown in Fig. 2. Comparison of the plots of the numerical solutions with the exact plot in Fig. 1 shows that solving the PDE over the adapted meshes produces a much more accurate *looking* approximate solution. We can also compare the numerical accuracy of the solution that results from solving over the adapted mesh shown in the top right of Fig. 2, which has 994 triangles, to the accuracy of a solution generated from a uniform mesh with many more triangles–say 2048. We look at the root-mean-squared error (RMSE) which we compute over 1001×1001 equally spaced points on $[0, 1] \times [0, 1]$. The solution from our adapted triangulation has RMSE 0.0087, while the RMSE from the (globally denser) uniform triangulation is 0.0134. This means that in this case, our mesh generates a solution that is 35% more accurate even while using a triangulation less than half the size of the uniform mesh. Using the *me* metric introduced in (Sect. 3.1) above, our adapted mesh represents a 75% *Net Gain* $= 1 - me_3/me_1$ over the uniform triangulation.

More numerical results for this Poisson problem are presented in Table 2. We control the number of vertices added into the vertex set by our algorithm so that the resulting adapted triangulation has the same number of triangles as a different uniform mesh. Then, we can compare the accuracy of the solutions that result from the distinct triangulations and claim that any error reduction is due to the adaptedness of our mesh to the particular PDE.

The meshes in the "Adapted mesh" column were formed from the uniform meshes in the previous row. The appropriate number of vertices was added to that smaller uniform triangulation to generate a triangulation of exactly the same size as the uniform mesh in the next row. For example, the first triangulation was a uniform mesh T_u over vertex set V_u with 288 triangles. An initial approximate solution u_h was computed over T_u; then, $D_t(u_h)$ values were computed, and just the right number N of centroids $\mathbf{c}_t$ was added so that the new vertex set $V_1 := V \cup_{n=1}^{N} \mathbf{c}_n$ generated a triangulation T_1 with $|T_1| = 512$. We compute the percent reduction in RMSE from the uniform error e_u and the adapted error e_a in the same manner that *Net Gain* is calculated.

Table 2 Example 1: RMSE results for the Poisson problem on $[0, 1] \times [0, 1]$ with exact solution $u(x, y) = \tanh(40y - 80x^2) - \tanh(40x - 80y^2)$. Here, the approximate solutions are in $S_7^0(\triangle)$

Compared performance of uniform mesh and Algorithm 1 adapted mesh						
Uniform mesh		Adapted mesh		Percent improvement		
Num Tri	RMSE	Num Tri	RMSE	Num Tri	RMSE reduction(%)	Net Gain(%)
288	6.01e-1	–	–	–	–	–
512	4.39e-1	512	3.29e-1	–	45.4	45.4
800	1.92e-1	800	4.70e-2	–	75.5	75.5
1152	5.18e-2	1152	1.44e-2	–	72.2	72.2
1568	2.96e-2	1568	3.31e-3	–	88.9	88.9
2048	1.34e-2	2048	8.56e-4	–	94.0	94.0

Table 3 Triangulation times for Example 1, in seconds; triangulation times make up a negligible component of the total computational time

Triangulation times for Example 1					
Initial triangulation		After 1 iteration		After 2 iterations	
Num Tri	Tri Time	Num Tri	Tri Time	Num Tri	Tri Time
512	0.1953	800	0.1783	1152	0.1355
800	0.1467	1152	0.1259	1568	0.1735
1152	0.2140	1568	0.1716	2048	0.2205
1568	0.2637	2048	0.2206	2592	0.2924

We emphasize that we retriangulate globally with a Delaunay algorithm, assuring that the triangulation we use for the fixed vertex set returned by Algorithm 1 has the max–min angle property. But does this global retriangulation have a high computational cost? Does it add significantly to computation times? To address this potential concern, we report triangulation and retriangulation times for the meshes generated in Example 1. In Table 3, we see that the time spent retriangulating does not significantly impact computational times. In fact, because the initial triangulation requires a generation of a point grid across the domain, retriangulations of the vertex set passed by Algorithm 1 are often faster than the generation of the initial triangulation. The total time required to generate the table of data for Example 1 is about 5 min on a 2009 MacBook Pro with 4GB RAM; the total triangulation and retriangulation times for all the meshes reported in the table are less than 5 s. Since these triangulation times are negligible, we do not continue to report them in the remaining examples.

Although generating Delaunay tetrahedralizations of 3-dimensional vertex sets is a more complex undertaking, we expect that remeshing times will remain small relative to total computational time. On the same computer, we computed a Delaunay

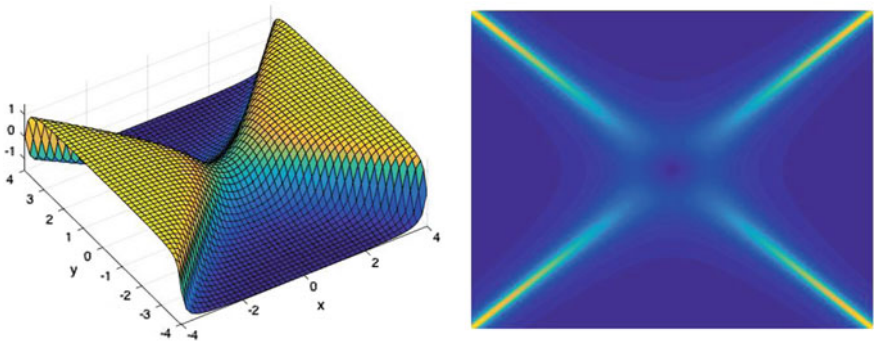


Fig. 3 A plot of the function $u(x, y) = \arctan(x^2 - y^2)$ on $[-4, 4] \times [-4, 4]$ (*left*), and a 2D color plot of its $|u_x| + |u_y|$ values (*right*)

tetrahedralization of 10,000 random vertices (x, y, z) in MATLAB; the meshing took just 0.3078 s. Thus, for reasonably sized vertex sets, we do not anticipate that retetrahedralization times will pose a difficulty in generalizing our approach to 3-dimensional elliptic problems.

Example 2 Here, we solve the Poisson problem with exact solution $u(x, y) = \arctan(x^2 - y^2)$. This function is also fairly difficult to approximate if one uses only a uniform triangulation. Its graph and a 2D color plot of its D_t values are shown in Fig. 3.

Here, the solution is studied over $\Omega = [-4, 4] \times [-4, 4]$ using spline space S_9^1. Again, we shall use the RMSE and number of triangles to measure the goodness of the performance of our spline method.

We show the algorithm's performance visually in Fig. 4. For comparison, a uniform refinement of the original mesh (the first one in the top row of Fig. 4) yields a triangulation with 800 triangles and produces a spline solution with a RMSE of 0.0030. The adaptive method achieves this level of accuracy while using a triangulation that less than half the size. Comparing these two results shows that the adapted mesh produces a *Net Gain* of 55%. As in the first example, we present more numerical results for this Poisson problem in Table 4 below.

3.2 Part 2: Adding and Removing Triangles

We now want to consider removing vertices from the triangulation as needed. This approach is worth consideration for reasons of portability and ease of use. If our spline solution contains more coefficients than necessary in parts of the domain where the function is not changing rapidly, then we would like to create a smaller spline solution by reducing the density of triangles in those regions. If we can do this

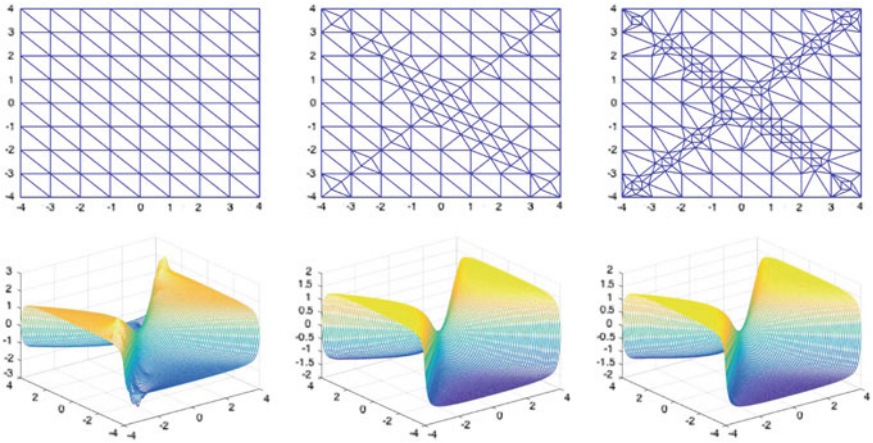


Fig. 4 Example 2: Here, we solve the Poisson problem on $[-4, 4] \times [-4, 4]$ with exact solution $u(x, y) = \arctan(x^2 - y^2)$ over spline space S_9^1. A uniform triangulation (*top left*), a triangulation produced after 1 iteration of Algorithm 1 (*top middle*), and the triangulation resulting from a second pass of the algorithm (*top right*). Spline solutions (*on the second row*) based on triangulations shown in the *top row*. The RMSEs, numbers of triangles, and *me* are explained in the *third row*

Table 4 Example 2: RMSE results for the Poisson problem on $[-4, 4] \times [-4, 4]$ with exact solution $u(x, y) = \arctan(x^2 - y^2)$. Here, the approximate solutions are in S_9^1

Compared performance of uniform mesh and Algorithm 1 adapted mesh						
Uniform mesh		Adapted mesh		Percent improvement		
Num Tri	RMSE	Num Tri	RMSE	Num Tri	RMSE reduction	Net Gain
288	2.71e-2	–	–	–	–	–
512	8.72e-3	512	6.57e-4	–	92.5%	92.5%
800	3.03e-3	800	4.56e-4	–	84.9%	84.9%
1152	1.13e-3	1152	7.66e-5	–	93.2%	93.2%
1568	4.72e-4	1568	2.17e-5	–	95.4%	95.4%
2048	2.08e-4	2048	1.33e-5	–	93.6%	93.6%

without affecting numerical accuracy, the resulting spline solution would be easier to transmit, store, and perform calculations with. To this end, our algorithm is presented as follows:

Algorithm 2 Input a triangulation $\triangle$ with vertex set V and triangle list T, and initial spline approximation s, and tolerance TOL. Then do the following steps:

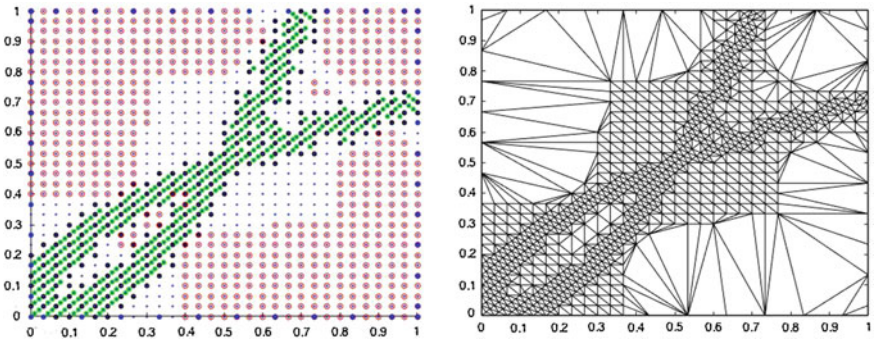


Fig. 5 Shown *left*, a color-coded vertex set demonstrating Algorithm 2 applied to the Poisson problem with exact solution $u(x, y) = \tanh(40y - 80x^2) - \tanh(40x - 80y^2)$, shown *left*. The dark vertices around the boundary form the list $\mathscr{P}$; the dark vertices in the interior of the domain form the list $\mathscr{L}$; the vertices forming the *ribbon shape* through the diagonal of the domain are the new vertices $\mathbf{c}_t$; the other *red-circled vertices* are to be removed (*left*). The resulting triangulation (*right*)

- compute centroids $\mathbf{c}_t$ of all triangles t in T;
- compute values of $D_t(s) = |D_x s| + |D_y s|$ at the location $\mathbf{c}_t$ for all $t \in T$;
- sort values of D_t by size;
- choose the $\mathbf{c}_t$ corresponding to the largest D_t values above a certain threshold to be added to the vertex set V;
- make a list $\mathscr{L}$ of the vertices of all triangles to which new points $\mathbf{c}_t$ were added;
- make a list $\mathscr{P}$ of a certain proportion of boundary vertices of V to be protected from removal;
- remove all vertices not in $\mathscr{L}$ or $\mathscr{P}$ with D_t value less than TOL;
- find a Delaunay triangulation nT of the new vertex set $\tilde{V}$, and output both.

This algorithm has great utility in applications where the solution is flat over large portions of the domain. We exhibit the process in Fig. 5.

Frequently, deleting vertices from the original mesh results in a new triangulation with a smaller minimum angle than that of the original one. Having highly acute triangles in a mesh is problematic for accurate numerical approximation in theory and in practice; constants in many error bounds depend on this minimum angle, and computing spline coefficients over especially skinny triangles leads to numerical instability.

To overcome the difficulty, we implement ideas from Ruppert's algorithm to improve the minimum angle characteristic of the mesh, without reintroducing too many vertices. Ruppert's algorithm works roughly by alternately splitting "encroached" edges and inserting circumcenters of skinny triangles into the vertex set. Given some reasonable assumptions about the domain to be triangulated, Ruppert's algorithm is guaranteed to converge to a mesh with minimum angle greater than 20°. See [19] for more details.

Adapting these ideas from Ruppert's algorithm, we describe our mesh quality improvement algorithm as follows:

Algorithm 3 Input a triangulation $\triangle$ with vertex set V and triangle list T, minimum angle tolerance *ANGTOL*, and distance tolerance *DTOL*. Then, we do the following steps:

- make a list T_s of all triangles with minimum angle less than *ANGTOL*;
- calculate circumcenters cr_t of triangles in T_s;
- if any circumcenters lie outside the domain, project them onto boundary;
- if a projected circumcenter is closer to existing boundary vertices than *DTOL*, delete from list;
- iterate: if any pair of circumcenters cr_i and cr_j are closer than *DTOL*, replace the pair with their midpoint;
- add cr_t to the vertex set V;
- retriangulate and output the new vertex set $\tilde{V}$ and the new triangle list $\tilde{T}$.

During our experiments, numerical accuracy suffered when computing over a triangulation with minimum angle below 4°. Therefore, we implemented Algorithm 3 with $ANGTOL = 4$. We choose one triangle from the original mesh and set *DTOL* to be half than the minimum edge length of this triangle; because the initial triangulation is uniform, the functionality of *DTOL* does not depend on which triangle we choose. In all cases tested with $ANGTOL \leq 4$, the output of Algorithm 3 was a triangulation with a greater minimum angle characteristic. The resulting meshes also often substantially improve numerical results, especially as applied in Sect. 3.3. The process is shown in Fig. 6.

Example 3 We return to the Poisson problem with exact solution $u(x, y) = \tanh(40y - 80x^2) - \tanh(40x - 80y^2)$ from Example 1 to show the effectiveness of adding *and* removing points from the vertex set. We again solve the problem in the spline space S_7^0. Because of the added variability in the size of the vertex set produced by Algorithms 2 and 3, we make good use of the *mesh efficiency* metric

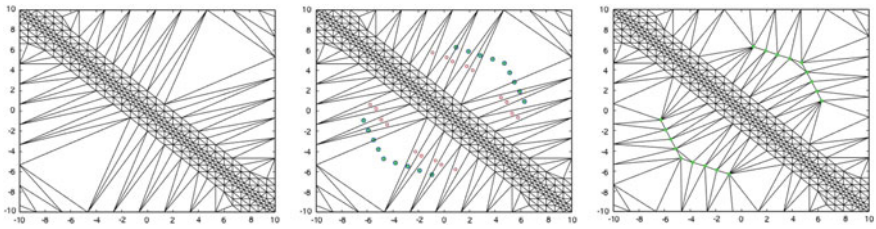


Fig. 6 An output triangulation of Algorithm 2 applied to the problem in Example 4 with minimum angle 2.5° (*left*) is input into Algorithm 3; the centroids of the triangles with angles below 4° marked by *small red circles* and the circumcenters of those triangles shown as *larger green dots* (*middle*); those *larger green dots* become vertices in the new triangulation (*right*) which has minimum angle 5.1°

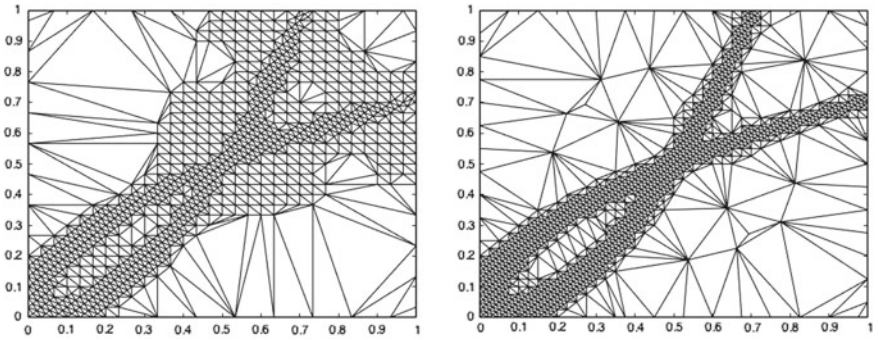


Fig. 7 A triangulation from Algorithm 2 with minimum angle above 4° (*left*); a different triangulation resulting from Algorithms 2 and 3, right, where the minimum angle was raised above 4° (*right*)

to gauge relative improvements in the adaptedness of our triangulation to the PDE. Figure 7 shows a few triangulations produced by the algorithms (from different initial uniform meshes) in this setting.

Table 5 compares numerical results for uniform meshes and adapted triangulations resulting from the algorithms. This table shows the number of triangles and RMSE for an initial uniform mesh $\triangle_u$ and initial solution $s_u \in S_7^0$, along with data for the solution computed over an adapted mesh of similar size produced by Algorithms 2 and 3. The *Net Gain* of the adapted mesh over the original uniform mesh is also shown.

While it is no surprise that these algorithms produce a better mesh for solving this PDE than the uniform approach, it is noteworthy that they also offer an improvement over Algorithm 1. Numerical results demonstrating this are shown in Table 6; here again, we made an effort to compare *me* data for solutions produced over similarly sized triangulations.

The percent improvement columns in Tables 5 and 6 are calculated in the same way as the *Net Gain* percentage described above. This number is the percentage of triangles that must be removed from $\triangle_u$ in order to produce a triangulation the same size as $\triangle_a$.

Example 4 We present one more numerical example—the Poisson problem on $[-10, 10] \times [-10, 10]$ with exact solution $u = \dfrac{1}{1 + \exp^{-10(x+y)}}$. In this example, we solve for an approximate spline solution $s \in S_8^1$. Figure 8 shows a plot of the solution together with a 2D color plot of its derivative values. The triangulations shown in Fig. 6 resulted from applying Algorithms 2 and 3 to this problem. As in the previous example, we compare performance of the mesh generated by Algorithms 2 and 3 both to uniform meshes and to meshes of similar size from Algorithm 1. The results are shown in Tables 7 and 8, respectively.

Table 5 Example 3: RMSE results for the Poisson problem on $[0, 1] \times [0, 1]$ with exact solution $u(x, y) = \tanh(40y - 80x^2) - \tanh(40x - 80y^2)$. Approximate solutions are in S_0^7. The first two columns (left) show the number of triangles and the RMSE for the uniform triangulations; the next two columns contain the same data for adapted meshes from Algorithms 2 and 3 of approximately the same size as the uniform mesh in the same row

Compared performance of uniform mesh and Algorithms 2 and 3 meshes						
Uniform		Algorithms 2 and 3		Percent improvement		
Num Tri	RMSE	Num Tri	RMSE	Num Tri reduction (%)	RMSE reduction (%)	Net Gain (%)
800	1.92e-1	794	3.29e-2	0.8	79.6	79.7
1152	5.18e-2	1147	3.33e-3	0.4	93.6	93.6
1568	2.96e-2	1533	5.27e-4	2.2	98.2	98.2
2048	1.34e-2	1978	7.14e-5	3.4	99.5	99.5

Table 6 Example 3: RMSE results for the Poisson problem on $[0, 1] \times [0, 1]$ with exact solution $u(x, y) = \tanh(40y - 80x^2) - \tanh(40x - 80y^2)$. Approximate solutions are in S_7^0. The first two columns (left) triangulation data and RMSE for meshes and spline solutions resulting from Algorithm 1 where points are merely added to the vertex set; the next two columns show the same data for meshes and splines produced by Algorithms 2 and 3 where vertices may added to and removed from the original vertex set

Compared performance of Algorithm 1 meshes and Algorithms 2 and 3 Meshes						
Algorithm 1		Algorithms 2 and 3		Percent improvement		
Num Tri	RMSE	Num Tri	RMSE	Num Tri reduction (%)	RMSE reduction (%)	Net Gain (%)
800	4.70e-2	794	3.29e-2	0.8	30.0	30.5
1152	1.44e-2	1147	3.33e-3	0.4	76.8	76.9
1568	3.31e-3	1533	5.27e-4	2.2	84.1	84.1
2048	8.56e-4	1978	7.14e-5	3.4	91.7	91.9

3.3 Part 3: A Good Initial Guess Triangulation

We now present a heuristic method to generate a better initial triangulation for solving Poisson's equation with a nonzero source function. We produce a mesh that is denser in regions where the source function is large. The previous algorithms can then be applied to spline solutions produced over this initial mesh. The algorithm is presented as follows:

Algorithm 4 Input a source function f for Poisson's equation $-\Delta u = f$, and a parameter n related to desired mesh fineness. Then,

- with density specified by parameter n, generate 2 point grids and G_2 so that G_2 is twice as dense as G_1 and $G_1 \subset G_2$;

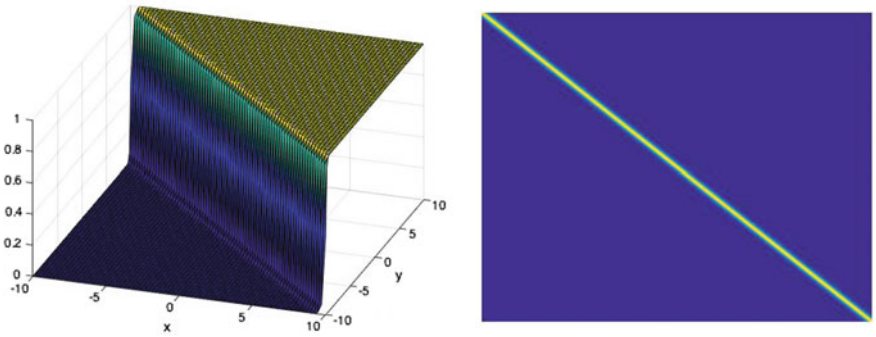


Fig. 8 A plot of the function $u = \dfrac{1}{1+\exp^{-10(x+y)}}$ on $[-10, 10] \times [-10, 10]$ (*left*), and a 2D color plot of its $|u_x| + |u_y|$ values (*right*)

Table 7 Example 4: RMSE results for the Poisson problem on $[-10, 10] \times [-10, 10]$ with exact solution $u(x, y) = \dfrac{1}{1+\exp^{-10(x+y)}}$. Approximate solutions are in S_8^1. The first two columns (left) show the number of triangles and the RMSE for the uniform triangulations; the next two columns contain the same data for adapted meshes from Algorithms 2 and 3 of approximately the same size as the uniform mesh in the same row

Compared performance of uniform meshes and Algorithm 2 & 3 adapted meshes						
Uniform		Algorithm 2 and 3		% improvement		
Num Tri	RMSE	Num Tri	RMSE	Tri Red. (%)	RMSE Red. (%)	Net Gain (%)
392	2.06e-2	386	2.84e-3	1.5	86.2	86.4
512	7.26e-3	494	2.78e-4	3.5	96.2	96.3
648	7.75e-4	640	1.31e-4	1.2	83.1	83.3
800	3.55e-3	738	6.05e-5	7.8	98.3	98.4

Table 8 Example 4: RMSE results for the Poisson problem on $[-10, 10] \times [-10, 10]$ with exact solution $u(x, y) = \dfrac{1}{1+\exp^{-10(x+y)}}$. Approximate solutions are in S_8^1. The first two columns (left) triangulation data and RMSE for meshes and spline solutions resulting from Algorithm 1; the next two columns show the same data for meshes and splines produced by Algorithms 2 and 3

Compared performance of Algorithm 1 meshes and Algorithms 2 and 3 meshes						
Algorithm 1		Algorithms 2 and 3		Percent improvement		
Num Tri	RMSE	Num Tri	RMSE	Num Tri reduction (%)	RMSE reduction (%)	Net Gain (%)
800	2.05e-3	590	1.32e-4	26.3	93.6	95.3
1080	8.70e-4	660	6.17e-5	38.9	92.9	95.7
1568	3.83e-4	1148	4.65e-6	26.8	98.8	99.1
2048	1.27e-4	1887	2.29e-6	7.8	98.2	98.3

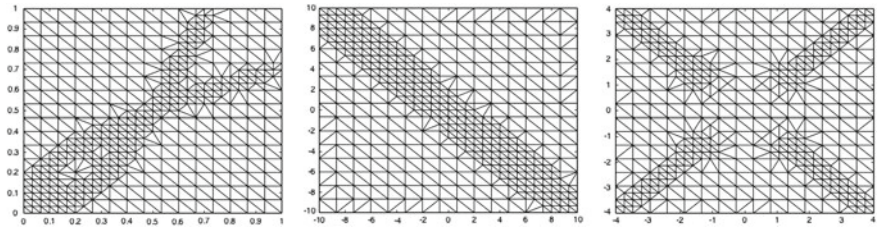


Fig. 9 Triangulations from Algorithm 4; a mesh for Poisson problem with exact solution $u(x, y) = \tanh(40y - 80x^2) - \tanh(40x - 80y^2)$ (*left*); mesh for exact solution $u = \frac{1}{1+\exp^{-10(x+y)}}$ (*middle*); mesh for exact solution $u(x, y) = \arctan(x^2 - y^2)$ (*right*)

- take a point $g \in G_2$, and let its horizontally and vertically adjacent neighbors be g_1, g_2, g_3, g_4;
- assign to g the value $L(g) = \frac{1}{5}(|f(g)| + |f(g_1)| + ... + |f(g_4)|)$;
- find the quartile of points Q of G_2 with the largest L values;
- build the vertex set $V := G_1 \bigcup Q$ and triangulate using Delaunay triangulation.

This algorithm places smaller triangles around areas where the values of the source function f are the greatest. The values of f obviously tell us something about changes in the solution function u, and while the correspondence between large values of f and large changes in u is not explicit, making the initial guess that they occur in the same regions of the domain results in improved numerical accuracy. Examples of triangulations produced by Algorithm 4 are shown in Fig. 9. We refer to these types of "initial guess" triangulations as *ig*-meshes in the following.

Example 5 In this example, we compare the numerical performance of spline solutions over the *ig*-meshes to those generated by uniform initial triangulations.

We begin with the Poisson problem with exact solution $u(x, y) = \tanh(40y - 80x^2) - \tanh(40x - 80y^2)$; Table 9 compares the mesh efficiencies of uniform meshes and *ig*-meshes from Algorithm 4 of similar size. For all the numerical results in this section, we use splines $s \in S_5^1(\triangle)$.

Table 9 shows that the *ig*-mesh offers improved numerical results in comparison with an initial uniform triangulation. Tables 10 and 11 show similar results for the other PDE solutions tested in Sects. 3.1 and 3.2. This finding has great potential utility, since computing the *ig*-mesh is far computationally cheaper than producing adapted meshes via Algorithms 1 and 2. Here, no initial solution is required; we simply use source function values to give an indication where the solution will be changing most rapidly. Of course, for some PDEs, the source functions are 0 or constant. In these settings, the algorithm will be useless.

The improvements from the *ig*-mesh seem to be maintained even when that triangulation is fed into Algorithms 2 and 3. A large *Net Gain* is observed when comparing against uniform meshes, but also when compared to the adapted triangulations produced by the algorithms in the previous sections. The tables below contain numerical data that supports this claim.

Table 9 Comparison of the RMSE for spline solutions produced over uniform meshes and *ig*-meshes of approximately the same size. This data is from the Poisson problem on $[0, 1] \times [0, 1]$ with exact solution $u(x, y) = \tanh(40y - 80x^2) - \tanh(40x - 80y^2)$. Solutions are in spline space S_5^1

Compared performance of uniform meshes and *ig*-meshes						
Uniform		*ig*-mesh		% improvement		
Num Tri	RMSE	Num Tri	RMSE	Tri Red.	RMSE Red. (%)	Net Gain (%)
800	1.44e-1	800	2.34e-2	–	83.7	83.7
1250	5.72e-2	1146	7.88e-3	8.3%	86.2	87.4
1800	2.36e-2	1706	3.08e-3	5.2%	87.0	87.6

Table 10 Comparison of the RMSE for spline solutions produced over uniform meshes and *ig*-meshes of approximately the same size. This data is from the Poisson problem on $[-10, 10] \times [-10, 10]$ with exact solution $u = \frac{1}{1+\exp^{-10(x+y)}}$. Solutions are in spline space S_5^1

Comparison of the performance of uniform meshes and *ig*-meshes						
Uniform		*ig*-mesh		% improvement		
Num Tri	RMSE	Num Tri	RMSE	Tri Red. (%)	RMSE Red. (%)	Net Gain (%)
1152	1.34e-2	1147	2.28e-3	0.4	83.0	83.0
1568	3.79e-3	1558	2.29e-3	0.6	39.6	40.0
2592	2.28e-3	2568	8.99e-4	0.9	60.6	60.9

Table 11 Comparison of the RMSE for spline solutions produced over uniform meshes and *ig*-meshes of approximately the same size. This data is from the Poisson problem on $[-4, 4] \times [-4, 4]$ with exact solution $u(x, y) = \arctan(x^2 - y^2)$. Solutions are in spline space S_5^1

Comparison of the performance of uniform meshes and *ig*-meshes						
Uniform		*ig*-mesh		% improvement		
Num Tri	RMSE	Num Tri	RMSE	Tri Red.	RMSE Red. (%)	Net Gain (%)
800	1.24e-2	800	2.04e-3	–	83.5	83.5
1250	5.17e-3	1150	7.04e-4	8.0%	86.4	87.5
1800	2.06e-3	1706	2.56e-4	5.2%	87.5	88.2

Example 6 Now, we compare the performance of the *ig*-mesh with the application of Algorithms 2 and 3 to the performance of a uniform mesh with the application of the algorithms. The results below show that starting with an *ig*-mesh offers improved numerical performance over starting with a uniform triangulation. We test all three of the Poisson problems mentioned in the previous sections and solve over spline space S_5^1. The data is presented in Tables 12, 13, and 14.

Table 12 RMSE comparison for spline solutions over triangulations from Algorithms 2 and 3 applied to uniform meshes and from Algorithms 2 and 3 applied to *ig*-meshes. This data is for the Poisson problem on $[0, 1] \times [0, 1]$ with exact solution $u(x, y) = \tanh(40y - 80x^2) - \tanh(40x - 80y^2)$. Solutions are in spline space S_5^1

Performance of Algorithms 2 and 3 applied to uniform and *ig*-meshes

Uniform + Alg		*ig*-mesh + Alg		% improvement		
Num Tri	RMSE	Num Tri	RMSE	Tri Red. (%)	RMSE Red. (%)	Net Gain (%)
1064	5.72e-2	892	5.30e-3	16.2	90.7	92.2
1385	1.39e-2	1213	2.49e-3	12.4	82.1	84.3
1496	6.89e-3	1860	1.22e-3	−24.3	82.4	78.1

Table 13 RMSE comparison for spline solutions over triangulations from Algorithms 2 and 3 applied to uniform meshes and from Algorithms 2 and 3 applied to *ig*-meshes. This data is for the Poisson problem on $[-10, 10] \times [-10, 10]$ with exact solution $u = \frac{1}{1+\exp^{-10(x+y)}}$. Solutions are in spline space S_5^1

Performance of Algorithms 2 and 3 applied to uniform and *ig*-meshes

Uniform + Alg		*ig*-mesh + Alg		% improvement		
Num Tri	RMSE	Num Tri	RMSE	Tri Red. (%)	RMSE Red. (%)	Net Gain (%)
574	3.37e-3	588	7.08e-4	−2.4	79.0	78.5
802	5.44e-4	800	3.73e-4	0.2	31.4	31.6
1068	5.09e-4	1236	5.32e-5	−15.7	89.6	87.9

Table 14 RMSE comparison for spline solutions over triangulations from Algorithms 2 and 3 applied to uniform meshes and from Algorithms 2 and 3 applied to *ig*-meshes. This data is for the Poisson problem on $[-4, 4] \times [-4, 4]$ with exact solution $u(x, y) = \arctan(x^2 - y^2)$. Solutions are in spline space S_5^1

Performance of Algorithms 2 and 3 applied to uniform and *ig*-meshes

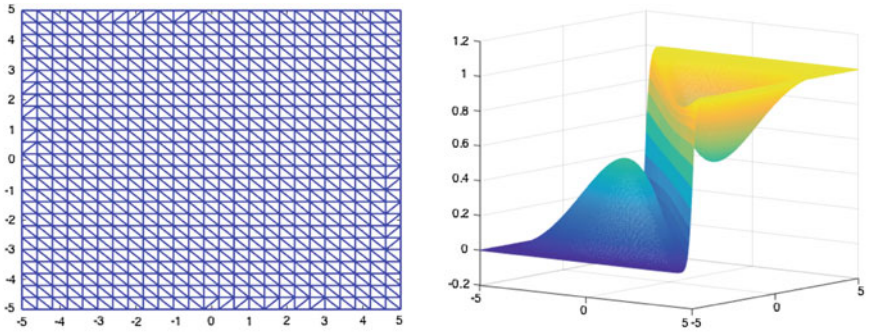
Uniform + Alg		*ig*-mesh + Alg		% improvement		
Num Tri	RMSE	Num Tri	RMSE	Tri Red. (%)	RMSE Red.(%)	Net Gain (%)
910	3.23e-3	1027	5.29e-4	−12.9	83.6	81.5
1441	5.36e-4	1620	1.61e-4	−12.4	70.1	66.3
2036	3.41e-4	2474	4.92e-5	−21.5	85.6	82.5

4 Remarks and Conclusions

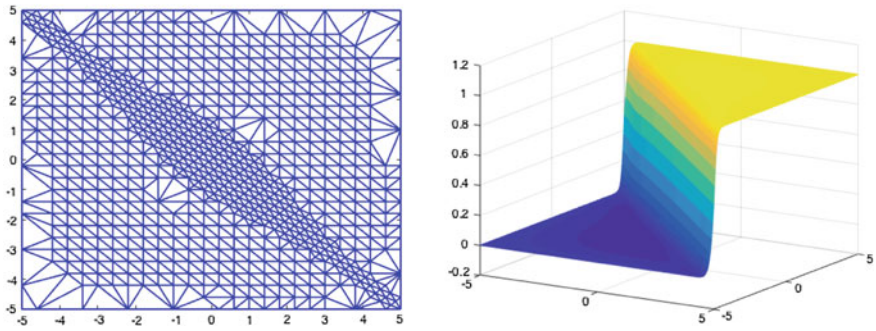
We have presented numerical results of our algorithms applied to three Poisson problems whose solutions are difficult to approximate accurately. For the examples shown in Sect. 3.1, we adjusted Algorithm 1 to terminate when the adapted mesh had

the same number of triangles as a uniform mesh for easy comparison of performance; for the numerical experiments in Sects. 3.2 and 3.3, the tolerance which centroids $\mathbf{c}_t$ are added to the vertex set is determined within the algorithm. We computed the mean m and standard deviation σ of the middle 80% of the D_t values and added only the $\mathbf{c}_t$ with D_t above $m + 2\sigma$ to the vertex set. When applying Algorithm 2 to the *ig*-mesh, we added *all* boundary vertices of Q to the protected list $\mathscr{P}$. Although it can often improve accuracy to apply the algorithms multiple times, for ease and consistency in this paper, the numerical results in the data tables resulted from only one iteration of the given algorithm.

It is important to note that the adapted meshes produced by our algorithms are only effective when applied to functions that are especially difficult to approximate in a certain region of the domain. If, after an initial approximation, errors are somewhat uniformly distributed across the domain, there will be little gained by refining the



MAX ERR: 0.960, # of Triangles: 1250, $me_1 = 1200.25$



MAX ERR 0.036, # of Triangles: 1473, $me_2 = 53.028$.

Fig. 10 This image shows spline solutions in $S_5^1(\triangle)$ of the biharmonic equation over $[0, 1] \times [0, 1]$ with exact solution $u(x, y) = \tanh(40y - 80x^2) - \tanh(40x - 80y^2)$. A uniform triangulation (*top left*), a triangulation after one iteration of Algorithms 2 and 3 (*bottom left*). The solution images (*right*) are based on the triangulations shown in the corresponding row. Here, maximum errors are reported. The adapted mesh results in a *Net Gain* of 96.2%; a uniform refinement of does not reach the accuracy achieved by this adapted mesh until the number of triangles in the mesh is about 4500

mesh in some areas more than others. However, when solutions experience drastic change in certain areas, like the examples shown in this paper, or others like the steep wavefront problem in [15], our adapted meshes have great utility. Our approach is not only more efficient than uniform refinement, but it also offers a straightforward way to improve numerical performance when uniform refinement is unfeasible due to the size of the existing triangulation.

These ideas can be expanded to a wider scope; for example, we have also used this adaptive method when solving more complicated PDEs like the biharmonic equation,

$$\begin{cases} \Delta^2 u = f, & \text{on } \Omega, \\ u(x) = g(x), & \text{on } \partial\Omega, \\ \frac{\partial u}{\partial \mathbf{n}} = h, & \text{on } \partial\Omega. \end{cases}$$

Figure 10 shows an initial approximation and an improved solution generated over an adapted mesh resulting from Algorithms 2 and 3.

References

1. M. Ainsworth, J.Z. Zhu, A.W. Craig, O.C. Zienkiewicz, Analysis of the zienkiewicz zhua-posteriori error estimator in the finite element method. Int. J. Numer. Methods Eng **28**(9), 2161–2174 (1989)
2. M. Ainsworth, J.T. Oden, A posteriori error estimation in finite element analysis. Comput. Methods Appl. Mech. Eng. **142**(1–2), 1–88 (1997)
3. D.N. Arnold, F. Brezzi, B. Cockburn, D. Marini, *Discontinuous Galerkin Methods*, Discontinuous Galerkin Methods for Elliptic Problems (Springer, Berlin, 2000), pp. 89–101
4. G. Awanou, M.-J. Lai, P. Wenston, in *The Multivariate Spline Method, for Scattered Data Fitting and Numerical Solution of Partial Differential Equations*, ed. By G. Chen, M.J. Lai. Wavelets and Splines (Nashboro Press, 2006), pp. 24–74
5. I. Babuška, A. Miller, A feedback finite element method with a posteriori error estimation: Part i. the finite element method and some basic properties of the a posteriori error estimator. Comput. Methods Appl. Mech. Eng. **61**(1), 1–40 (1987)
6. I. Babuška, W.C. Rheinboldt, A-posteriori error estimates for the finite element method. Int. J. Numer. Methods Eng. **12**(10), 1597–1615 (1978)
7. L. Beirão da Veiga, F. Brezzi, A. Cangiani, G. Manzini, L.D. Marini, A. Russo, Basic principles of virtual element methods. Math. Models Methods Appl. Sci. **23**(1), 199–214 (2013)
8. P. Binev, W. Dahmen, R. DeVore, Adaptive finite element methods with convergence rates. Numer. Math. **97**(2), 219–268 (2004)
9. P.G. Ciarlet, *The Finite Element Method for Elliptic Problems* (North–Holland, Amsterdam, 1978)
10. P. Fernandes, P. Girdinio, M. Repetto, G. Secondo, Refinement strategies in adaptive meshing. IEEE Trans. Mag. **28**(2), 1739–1742 (1992)
11. F.D. Gaspoz, P. Morin, Approximation classes for adaptive higher order finite element approximation. Math. Comp. **83**(289), 2127–2160 (2014)
12. X. Hu, D. Han, M.-J. Lai, Bivariate splines of various degrees for numerical solution of PDE. SIAM J. Sci. Comput. **29**, 1338–1354 (2007)
13. M.-J. Lai, L.L. Schumaker, *Spline Functions on Triangulations* (Cambridge University Press, Cambridge, 2007)

14. S. Li, L.L. Schumaker, Adaptive Computation with Splines on Triangulations with Hanging Vertices, in *Approximation Theory XV: San Antonio 2016*, Springer Proc. in Math. and Stat. vol. 201, ed. by G.E. Fasshauer, L.L. Schumaker (Springer-Verlag, 2017), pp. 197–218
15. W.F. Mitchell, A collection of 2d elliptic problems for testing adaptive grid refinement algorithms. Appl. Math. Comput. **220**, 350–364 (2013)
16. J.-M. Mirebeau, A. Cohen, Greedy bisection generates optimally adapted triangulations. Math. Comput. **81**(278), 811–837 (2012)
17. L.L. Schumaker, *Spline Functions: Computational Methods* (SIAM Publication, Philadelphia, 2015)
18. L.L. Schumaker, L. Wang, Splines on triangulations with hanging vertices. Constr. Approx. **36**(3), 487–511 (2012)
19. J.R. Shewchuk, Delaunay refinement algorithms for triangular mesh generation. Comput. Geom. **22**(1–3), 21–74 (2002)
20. P. Solin, L. Dubcova, J. Cerveny, I. Dolezel, Adaptive hp-fem with arbitrary-level hanging nodes for maxwells equations. Adv. Appl. Math. Mech **2**(4), 518–532 (2010)
21. R. Verfürth, A posteriori error estimation and adaptive mesh-refinement techniques. J. Comput. Appl. Math. **50**(1–3), 67–83 (1994)
22. J. Wang, X. Ye, A weak Galerkin mixed finite element method for second-order elliptic problems. Math. Comp. **83**, 2101–2126 (2014)
23. O.C. Zienkiewicz, J.Z. Zhu, The super convergent patch recovery and a posteriori error estimates. part 1: the recovery technique. Int. J. Numer. Methods Eng. **33**(7), 1331–1364 (1992)
24. O.C. Zienkiewicz, J.Z. Zhu, The superconvergent patch recovery and a posteriori error estimates. part 2: errror estimates and adaptivity. Int. J. Numer. Methods Eng. **33**(7), 1365–1382 (1992)