A Next Generation Partial Differential Equation Solver

Aaron Yeiser

October 30, 2016

Abstract

When solving differential equations in multiple dimensions, mesh generation is required to discretize the geometry of the domain. Current numerical techniques, such as finite element methods, are numerically unstable on meshes containing skinny triangles. Here, we develop a novel numerical method that is numerically stable even on meshes with skinny elements. Our method is spectrally accurate on each element, and the discretization size can be adapted to suit a wide variety of applications. Our algorithm alleviates the current burden on mesh generation algorithms of avoiding skinny triangles, allowing meshes to instead be optimized for the efficient solution of time-dependent partial differential equations. We can also simulate the Navier–Stokes equations at moderate Reynolds numbers with our method.

1 Introduction

Differential equations are one of the most powerful tools that we can use to analyze and model the world [1]. Phenomena as diverse as viscous fluid flow, heat dissipation, acoustic scattering, and structural deformations can all be modeled by differential equations. Unfortunately, differential equations rarely have analytic solutions, so the only feasible way to solve a differential equation is to perform numerical calculations. In two dimensions or more, it is usually necessary to discretize the geometry of the domain, and this discretization is often achieved with a mesh. A mesh is simply a collection of polygons or polyhedrons that partition a domain to a certain resolution, and those mesh elements are typically triangles or higher-dimensional analogues of triangles. Meshes are computationally convenient because they can be used to discretize a wide variety of complicated geometries. Finite element methods are currently the method of choice for solving a wide variety of engineering problems, such as electromagnetism, heat transfer, and fluid dynamics. They represent the solution as a piecewise polynomial, with each polynomial piece corresponding to a mesh element. Finite element methods use a weak formulation of the differential equation, defining a suitable inner-product to impose the equation. If the inner product is represented as $\langle \cdot, \cdot \rangle$, and the differential operator is L, then the matrix operator A is defined by

$$A_{ij} = \langle L[u_i], v_j \rangle,$$

where u_i and v_j are test and trial basis functions. By using basis functions with compact support, one can make this matrix operator extremely sparse. Adaptive finite element methods try to optimize the solution by varying the location of the mesh vertices, the degree of the piecewise polynomial basis functions, the size of the mesh elements, or a combination of the three strategies.

Finite element methods are widely used throughout industry, but they have one major drawback—they are slow and numerically unstable when the underlying mesh contains skinny triangles [6]. The skewness of a mesh element can be defined as the fraction

$$\frac{\text{optimal cell size} - \text{cell size}}{\text{optimal cell size}},$$

where the optimal cell size is the area of the equilateral triangle with the same circumcircle as the element. It is computationally expensive to generate meshes completely free of skinny triangles, and the computational cost depends on the geometry of the region. In addition, skinny triangles can actually lead to more efficient simulations, if they could be handled in a numerically stable manner. For instance, rapid fluid flow over the surface of an airfoil will only require high resolution perpendicular to the airfoil surface. Rather than representing the region around the airfoil with an enormous numbers of tiny quality triangles, one can simply use fewer skinny triangles parallel to the surface of the airfoil.

My method takes a radically different approach to solving differential equations than do finite element methods and standard spectral element methods. Finite element methods focus on solving differential equations on the entire mesh all at once, and current spectral element methods use dense, ill-conditioned matrices. The heart of my method is a fast, sparse, spectral partial differential equation (PDE) solver that operates individually on each element. We can generate almost-banded well-conditioned matrices to represent any linear differential operator. Rather than using the weak formulation of the differential equation, like finite element methods, we construct a family of sparse discrete differential operators that directly map a function in the basis of Chebyshev polynomials to its derivatives in so-called ultraspherical polynomials. In fact, we can construct a sparse discrete differential operator for any linear partial differential operator of low order. By removing rows of the operator corresponding to the highest-frequency components and replacing them with boundary condition rows, we have a well-conditioned matrix equation for any well-posed problem on a square domain. We can map the square domain onto a quadrilateral, and transform the differential equations accordingly. Even for skinny quadrilaterals, the transformed differential equations are still numerically stable to solve. Finally, we stitch three quadrilaterals together to form a triangle, using the Schur complement method, and we can use similar techniques to stitch together thousands of triangles. The Schur complement method allows us to solve for all of the interior boundary conditions of the mesh, and then independently compute the solution to the differential equation on each individual element—a process that can be easily parallelized. My method is able to handle a mesh containing skinny triangles, or a mixture of skinny and non-skinny elements. It is also able to represent solutions to differential equations extremely accurately on each mesh element, significantly reducing the number of elements required for an accurate solution.

2 Spectral methods

When solving differential equations, the problem is often simplified by viewing the solution as the sum of several basis functions. Depending on the type and domain of the function being analyzed, different sets of basis functions are used. For instance, sines and cosines are a natural basis for periodic functions. For nonperiodic functions, natural sets of basis functions include monomials or Chebyshev polynomials.

2.1 Chebyshev polynomials

Monomials are mathematically simple to handle, but they are a very numerically unstable set of basis functions for polynomials. In contrast, Chebyshev polynomials, defined as $T_k(x) = \cos(k \cos^{-1} x)$ for $x \in [-1, 1]$, are a stable set of basis functions for representing polynomials on the unit interval. Chebyshev polynomials oscillate between -1 and 1 on the unit interval, with $T_k(-1) = (-1)^n$, and $T_k(1) = 1$. We can represent any smooth function as an infinite Chebyshev series, and a Chebyshev series can be truncated to produce very accurate results. For instance, the Chebyshev series for e^x can be truncated at 16 terms to produce an approximant that is accurate to machine precision [9]. We can define the k^{th} set of Chebyshev points to be $\left\{\cos\left(\frac{\pi(k-m-1)}{k-1}\right)\middle| m \in \mathbb{Z}, 0 \leq m < n\right\}$. By sampling a function at Chebyshev points, the transformation to Chebyshev coefficients is well-conditioned even with large discretization sizes. Furthermore, this transformation can be rapidly computed using a variant of the fast Fourier transform [9].

2.2 Sparse discrete differential operators

Spectral methods are especially useful for solving linear differential equations. The goal of using spectral methods is to approximate an infinitely-dimensional linear differential operator with a finite-dimensional sparse matrix operator. We can represent a function as an infinite vector of Chebyshev coefficients, and we can truncate that vector at an appropriate size to achieve a sufficiently accurate approximation of a smooth function. If we attempted to create a differential operator D_λ that mapped a function in the Chebyshev basis to its λ^{th} derivative in the Chebyshev basis, that operator would be dense, as most of its matrix elements would be nonzero. Current spectral methods map Chebyshev polynomials to Chebyshev polynomials, giving dense, ill-conditioned differentiation matrices. In contrast, our method constructs D_λ to map a function in Chebyshev coefficients to its λ^{th} derivative in the ultraspherical basis of parameter λ. For every positive real λ, ultraspherical polynomials of parameter λ are orthogonal on $[-1,1]$ with respect to the weight function $(1-x^2)^{\lambda-\frac{1}{2}}$. Here, we will use ultraspherical polynomials with positive integer parameter. Using ultraspherical coefficients of parameter λ, the matrix λ^{th} differentiation matrix D_λ is

$$D_\lambda = 2^{\lambda-1}(\lambda-1)! \begin{pmatrix} \overbrace{0 \quad \cdots \quad 0}^{\lambda \text{ times}} & \lambda & & & \\ & & \lambda+1 & & \\ & & & \lambda+2 & \\ & & & & \ddots \end{pmatrix}. \tag{1}$$

These operators are diagonal and hence extremely sparse, but combining different-order derivative coefficients leads to a problem: different-order D_λ transform functions in the Chebyshev basis to ultraspherical bases of different parameters. The solution is to create basis conversion operators, denoted as S_λ, which convert between ultraspherical bases of different orders. Specifically, S_0 converts functions from the Chebyshev basis to the ultraspherical basis of parameter 1, and S_λ converts functions from the ultraspherical basis of parameter λ to the ultraspherical basis of parameter $\lambda+1$. For example, we can represent the differential operator $\frac{\partial u}{\partial x} + 2x$ as the matrix operator $(D_1 + 2S_0)u$. The ultraspherical conversion operators take the form

$$S_0 = \frac{1}{2}\begin{pmatrix} 2 & 0 & -1 & & \\ & 1 & 0 & -1 & \\ & & 1 & 0 & \ddots \\ & & & 1 & \ddots \\ & & & & \ddots \end{pmatrix}, \quad S_\lambda = \begin{pmatrix} 1 & 0 & -\frac{\lambda}{\lambda+2} & & \\ & \frac{\lambda}{\lambda+1} & 0 & -\frac{\lambda}{\lambda+3} & \\ & & \frac{\lambda}{\lambda+2} & 0 & \ddots \\ & & & \frac{\lambda}{\lambda+3} & \ddots \\ & & & & \ddots \end{pmatrix}, \quad \lambda > 0. \tag{2}$$

Since D_λ and S_λ are sparse for all valid λ, the final differential operator will be sparse and banded. Let L be some n-order differential operator such that $L = c_0 + c_1 D_1 + c_2 D_2 + \ldots + c_n D_n$ where $c_0, \ldots, c_n$ are constants and D_n is equivalent to $\frac{d^n}{dx^n}$. If we wish to solve the differential equation $Lu = f$, we create the matrix operator $\mathbf{L}$. We also create vectors $\vec{u}$ and $\vec{f}$ as vectors of Chebyshev coefficients approximating $u(x)$ and $f(x)$. Now, we can write $\mathbf{L} = c_0 S_{n-1} \cdots S_0 + c_1 S_{n-1} \cdots S_1 D_1 + \cdots + c_{n-1} D_{n-1} + c_n D_n$, and we can write the matrix equation $\mathbf{L}\vec{u} = S_{n-1} \cdots S_0 \vec{f}$.

In order to find a particular solution to the differential equation $Lu = f$, we need boundary conditions. Since the highest ordered coefficients of f are often very close to zero, we can simply replace the last two rows of $\mathbf{L}$ and the last two elements of $\vec{f}$ with boundary condition rows. The $(n-1)^{st}$ row of $\mathbf{L}$ is replaced by the row $[1, -1, 1, -1, \ldots, (-1)^n]$, and the $(n-1)^{st}$ element of $\vec{f}$ is replaced by $u(-1)$. Similarly, the n^{th} row of $\mathbf{L}$ is replaced by the row $[1, 1, 1, 1, \ldots]$, and the n^{th} element of $\vec{f}$ is replaced by $u(1)$. We now have a sparse, well-conditioned differential operator [7].

3 Solving PDEs in two dimensions

In two dimensions, we can write any polynomial p as its bivariate Chebyshev expansion

$$p(x, y) = \sum_{i,j=0}^{\infty} a_{ij} T_i(y) T_j(x), \qquad x, y \in [-1, 1].$$

To form a matrix operator, we can stack the columns of the coefficient matrix to form the vector

$$u = [a_{00},\ a_{10},\ a_{20}, \ldots, a_{n-1,0},\ a_{01},\ a_{11}, \ldots, a_{n-1,1}, \ldots, a_{n-1,n-1}]^T. \tag{3}$$

We use Kronecker products (denoted as '$\otimes$') in order to construct differential operators in two dimensions. For example $\frac{\partial u}{\partial x}$ is represented as $D_1 \otimes I$ and $\frac{\partial u}{\partial y}$ is represented as $I \otimes D_1$. To put together differential operators, we can use the identity $(A \otimes B)(C \otimes D) = AC \otimes BD$, so to represent an operator like $\frac{\partial^3}{\partial x \partial y^2}$, we use $D_1 \otimes D_2$. We can also add Kronecker products to construct more complicated operators; for instance,

$$\frac{\partial^2}{\partial x^2} - \frac{\partial^2}{\partial x \partial y} + 2\frac{\partial}{\partial x} \quad \Longleftrightarrow \quad D_2 \otimes S_1 S_0 - S_1 D_1 \otimes S_1 D_1 + 2 S_1 D_1 \otimes S_1 S_0. \tag{4}$$

A derivation of this two-dimensional sparse operator construction can be found in [8].

Boundary rows are slightly more complicated for two dimensions. If we have some arbitrary differential operator L with discretization size $n \times n$, the rows in L corresponding to the two highest order row and column coefficients are removed and replaced with boundary condition rows.

For $m = 1, 2, \ldots, n-2$, we find that

- Lower boundary condition row $mn-1$ is deleted and elements in columns $m(n-1)+1$ to mn are set to $1, -1, 1, -1, \dots, (-1)^n$.
- Upper boundary condition row mn is deleted and elements in columns $m(n-1)+1$ to mn are set to a row of ones.

For $m = 1, 2, \dots, n$, we see that

- Left boundary condition row $n(n-2)+m$ is deleted and elements in columns $m, m+n, m+2n, \dots, m+n(n-1)$ are set to $1, -1, 1, -1, \dots, (-1)^n$.
- Right boundary condition row $n(n-1)+m$ is deleted and elements in columns $m, m+n, m+2n, \dots, m+n(n-1)$ are set to a row of ones.

We only need $n-2$ boundary conditions for the lower and upper boundary conditions of the square because the value of u at those endpoints is already predetermined by the left and right boundary conditions.

To make the boundary condition rows sparse, we use the spectral decomposition of each boundary condition. On the square, we have the four cases, corresponding to the four boundaries:

$$\begin{aligned} u(x,1) &= \sum_{j=0}^{n-1}\left(T_j(x)\sum_{i=0}^{n-1} a_{ij}\right), & u(x,-1) &= \sum_{j=0}^{n-1}\left(T_j(x)\sum_{i=0}^{n-1}(-1)^i a_{ij}\right), \\ u(1,y) &= \sum_{i=0}^{n-1}\left(T_i(y)\sum_{j=0}^{n-1} a_{ij}\right), & u(-1,y) &= \sum_{i=0}^{n-1}\left(T_i(y)\sum_{j=0}^{n-1}(-1)^j a_{ij}\right). \end{aligned} \tag{5}$$

These equations result in the boundary condition rows shown in red and black in Figure 1. Figure 1 also shows the structure of the linear system used to solve a second-order PDE.

3.1 Quadrilateral domains

The method of constructing matrices described above works only on square domains. We now adapt existing spectral methods to solve differential equations on quadrilaterals, triangles, and finally meshes. To solve differential equations on a triangle, the most obvious solution is to map the triangle onto the square $[-1,1]^2$. Unfortunately, this approach maps one vertex of the triangle onto an entire edge of the square, or it produces a hanging node on the triangle that becomes a

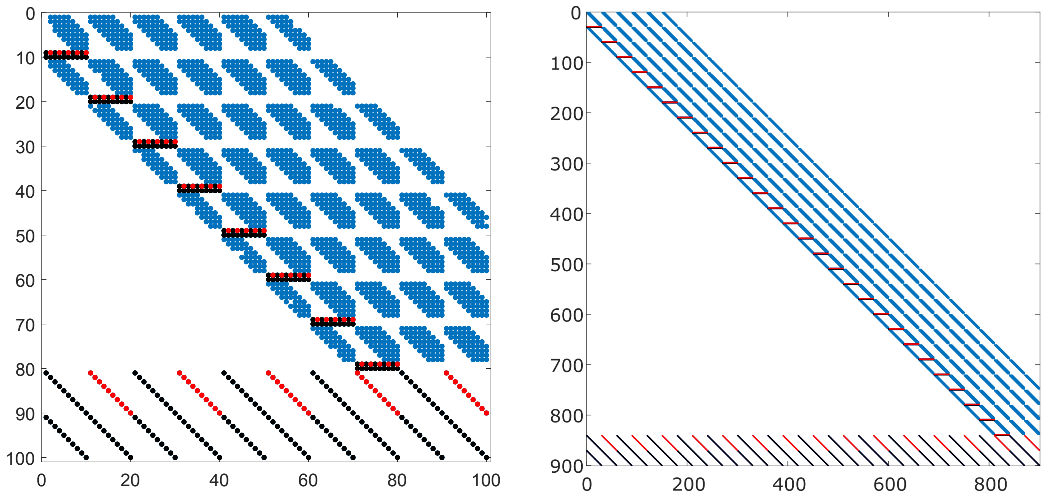

Figure 1: The leftmost figure shows the matrix structure of a second-order two-dimensional partial differential operator with a discretization size 10×10, and the rightmost figure shows the same operator with discretization size 30×30. Nonzero elements of the matrix are colored, and black and red elements correspond to boundary rows. Black elements are equivalent to 1 and red elements are equivalent to -1. With a discretization size of $n \times n$, the matrix has dimensions $n^2 \times n^2$, but the bandwidth only grows linearly with n.

vertex of the square. Either method of directly mapping the triangle to the square is prone to numerical instability, as well as impracticality when stitching multiple triangles together.

The solution that I found was to partition each triangle of the mesh into three quadrilaterals using the medians of the triangle. Let r and s be coordinates on the square, and let x and y be the coordinates on the (convex) quadrilateral. Also, let the quadrilateral have coordinates (x_1, y_1), (x_2, y_2), (x_3, y_3), and (x_4, y_4), with (x_1, y_1) mapped to $(1, 1)$ and (x_2, y_2) mapped to $(-1, 1)$.

I found the following bilinear map from the square to the quadrilateral:

$$x = a_1 + b_1 r + c_1 s + d_1 rs, \qquad y = a_2 + b_2 r + c_2 s + d_2 rs, \tag{6}$$

where

$$\begin{aligned} a_1 &= \frac{1}{4}\left(x_1 + x_2 + x_3 + x_4\right), \qquad & b_1 &= \frac{1}{4}\left(x_1 - x_2 - x_3 + x_4\right), \\ c_1 &= \frac{1}{4}\left(x_1 + x_2 - x_3 - x_4\right), \qquad & d_1 &= \frac{1}{4}\left(x_1 - x_2 + x_3 - x_4\right), \end{aligned} \tag{7}$$

and a_2, b_2, c_2, and d_2 are similarly defined with y_1, y_2, y_3, and y_4.

The transformation from the quadrilateral to the square is more complicated, but it is well-defined over the entire domain of the quadrilateral and it will not be needed.

When the quadrilateral domain is mapped onto the square, the differential equations on the quadrilateral are also distorted. Through the use of the chain rule, we obtain

$$\begin{aligned}
u_x &= u_r r_x + u_s s_x,\\
u_y &= u_r r_y + u_s s_y,\\
u_{xx} &= u_{rr}(r_x^2) + 2u_{rs}(r_x s_x) + u_{ss}(s_x^2) + u_r r_{xx} + u_s s_{xx},\\
u_{xy} &= u_{rr}(r_x r_y) + u_{rs}(r_x s_y + s_x r_y) + u_{ss}(s_x s_y) + u_r r_{xy} + u_s s_{xy},\\
u_{yy} &= u_{rr}(r_y^2) + 2u_{rs}(r_y s_y) + u_{ss}(s_y^2) + u_r r_{yy} + u_s s_{yy}.
\end{aligned} \tag{8}$$

We still need to evaluate derivatives r_x, r_y, s_x, s_y, r_{xx}, r_{xy}, r_{yy}, s_{xx}, s_{xy}, and s_{yy}. Using the Inverse Function Theorem, we can invert the Jacobian of the transformation to obtain

$$\begin{pmatrix} r_x & r_y \\ s_x & s_y \end{pmatrix} = \begin{pmatrix} x_r & x_s \\ y_r & y_s \end{pmatrix}^{-1} = \frac{1}{\det(x,y)} \begin{pmatrix} y_s & -x_s \\ -y_r & x_r \end{pmatrix}, \qquad \det(x,y) = x_r y_s - x_s y_r. \tag{9}$$

Using these equations, we can also find second derivatives r_{xx}, r_{xy}, r_{yy}, s_{xx}, s_{xy}, and s_{yy}. These derivatives can be represented as a bivariate cubic polynomial divided by $\det(x,y)^3$.

3.2 Triangular element construction

Since it is not practical to map a triangular domain to a single square, we can stitch together three quadrilateral domains to form a triangle. We can impose Dirichlet conditions on the exterior of the triangle, and Dirichlet and Neumann conditions along the interior boundaries for continuity and differentiability of the solution. The process of stitching together quadrilaterals is most easily done by treating the interior edges as independent variables. Although extra rows and columns are added to the differential operator, the resulting matrix equation can easily be simplified using the

Schur complement method. The structure of the linear system is

$$\begin{bmatrix} A_{11} & 0 & 0 & A_{1\Gamma} \\ 0 & A_{22} & 0 & A_{2\Gamma} \\ 0 & 0 & A_{33} & A_{3\Gamma} \\ A_{\Gamma 1} & A_{\Gamma 2} & A_{\Gamma 3} & A_{\Gamma\Gamma} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_\Gamma \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ f_\Gamma \end{bmatrix}. \tag{10}$$

The A_{ii} blocks are standard matrices for solving differential equations on a quadrilateral. The $A_{i\Gamma}$ blocks effectively replace some of the boundary condition elements of the right hand side, enforcing continuity along the internal boundaries. The $A_{\Gamma i}$ and $A_{\Gamma\Gamma}$ blocks use the transformation from the square to the quadrilateral to compute the normal derivative of the solution along the boundaries, enforcing differentiability. The vectors u_1, u_2, and u_3 correspond to the interior regions of each of the three quadrilaterals. The vector u_Γ corresponds to the interior boundary conditions, and it is broken into three parts, with one part corresponding to each interior boundary condition.

Using the Schur complement method, we can solve for u_Γ as follows:

$$\begin{aligned} \Sigma &= A_{\Gamma\Gamma} - A_{\Gamma 1}A_{11}^{-1}A_{1\Gamma} - A_{\Gamma 2}A_{22}^{-1}A_{2\Gamma} - A_{\Gamma 3}A_{33}^{-1}A_{3\Gamma}, \\ \Psi &= f_\Gamma - f_1 A_{11}^{-1}A_{1\Gamma} - f_2 A_{22}^{-1}A_{2\Gamma} - f_3 A_{33}^{-1}A_{3\Gamma}, \\ u_\Gamma &= \Sigma^{-1}\Psi. \end{aligned} \tag{11}$$

Then, we can solve for the rest of u_1, u_2, u_3 as follows:

$$u_1 = A_{11}^{-1}(f_1 - A_{1\Gamma}u_\Gamma), \qquad u_2 = A_{22}^{-1}(f_2 - A_{2\Gamma}u_\Gamma), \qquad u_3 = A_{33}^{-1}(f_3 - A_{3\Gamma}u_\Gamma). \tag{12}$$

By using the Schur complement method, we are essentially performing blockwise elimination on the linear system in Equation 11. Chapter 3 of [5] gives a guide to using the Schur complement method for domain decomposition. The entire process can be parallelized, which would be very useful for a mesh containing thousands or millions of triangles.

Figure 2 provides a logical numbering scheme for edges, vertices, and quadrilateral regions. For each interior region u_j, vertex j of the triangle is mapped to the upper right corner of the square $[-1, 1]^2$. The vector u_Γ is split into three subvectors of length n corresponding to the three interior boundaries of the triangle.

When stitching together two different domains, it is necessary to ensure that the solution is continuous and differentiable across the boundaries. The $A_{i\Gamma}$ blocks provide most of the boundary

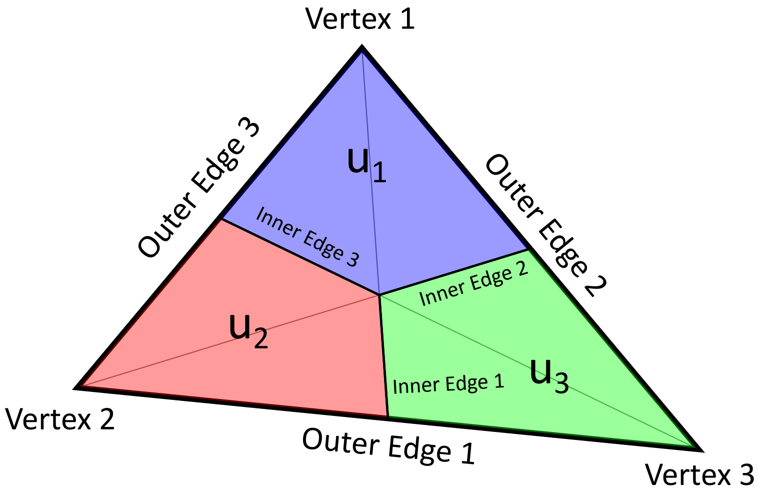

Figure 2: Each colored region u_1, u_2, and u_3 is given its own matrix. Vector u_Γ is split between inner edges 1, 2, and 3.

conditions for continuity of the solution. As an example, we will construct the $A_{1\Gamma}$ block. The lower boundary of region u_1 corresponds to inner edge 2, and the left boundary of u_1 corresponds to inner edge 3. Columns $2n+1$ through $3n$ of $A_{1\Gamma}$ correspond to the constraints between inner edge 3 and u_1. There are n rows in A_{11} that constrain the left boundary of u_1: rows $n(n-2)+1$ through $n(n-1)$. Therefore, these elements of f_1 are set to zero, and the elements in row $n(n-2)+m$ and column $m+2n$ for $m = 1, 2, \ldots, n$ of $A_{1\Gamma}$ are set to -1. Thus, inner edge 3 is entirely constrained to the left edge of u_1.

To constrain the lower edge of u_1 to inner edge 2, we use columns $n+1$ through $2n$ of $A_{1\Gamma}$. Unlike for the left edge, there are only $n-2$ boundary condition rows in A_{11} for the lower edge. We set the elements in row $mn-1$ and column $m+n$ for $m = 1, 2, \ldots, n-2$ of $A_{1\Gamma}$ to -1. This procedure constrains the lowest frequencies of inner edge 2 to the lower edge of u_1, but it leaves two degrees of freedom. In order to create a well-conditioned matrix, these additional degrees of freedom must be handled as special cases in the $A_{\Gamma i}$ and $A_{\Gamma\Gamma}$ blocks. Otherwise, we would have redundant boundary conditions at the centroid of the triangle.

The $A_{\Gamma i}$ and $A_{\Gamma\Gamma}$ blocks force the solution to be differentiable across interior boundaries, as well as being continuous at the endpoints of the interior boundaries. We will demonstrate how to construct Neumann boundary conditions for interior edge 1.

Suppose that interior edge 1 has endpoints (x_1, y_1) and (x_2, y_2). Let us define $\alpha = \frac{\Delta x}{\sqrt{(\Delta x)^2+(\Delta y)^2}}$ and $\beta = \frac{\Delta y}{\sqrt{(\Delta x)^2+(\Delta y)^2}}$, so that $\alpha^2 + \beta^2 = 1$. We can now define the derivative perpendicular to the boundary as

$$D_\perp = \beta u_x - \alpha u_y = \beta(u_r r_x + u_s s_x) - \alpha(u_r r_y + u_s s_y) = u_r(\beta r_x - \alpha r_y) + u_s(\beta s_x - \alpha s_y). \tag{13}$$

Substituting for the inverse derivatives, we have

$$\det(r, s) D_\perp = u_r(\beta y_s + \alpha x_s) - u_s(\beta y_r + \alpha x_r). \tag{14}$$

With some more substitution, we obtain

$$\det(r, s) D_\perp = u_r[(\beta c_2 + \alpha c_1) + r(\beta d_2 + \alpha d_1)] - u_s[(\beta b_2 + \alpha b_1) + s(\beta d_2 + \alpha d_1)], \tag{15}$$

where $x = a_1 + b_1 r + c_1 s + d_1 rs$ and $y = a_2 + b_2 r + c_2 s + d_2 rs$.

Setting constants $q_b = \beta b_2 + \alpha b_1$, $q_c = \beta c_2 + \alpha c_1$, and $q_d = \beta d_2 + \alpha d_1$ gives

$$\det(r, s) D_\perp = u_r(q_c + q_d r) - u_s(q_b + q_d s). \tag{16}$$

Interestingly, the determinant for these quadrilateral domains is very simple and symmetrical. In Appendix A, we can show that for any quadrilateral defined by a triangle as shown in Figure 2, we have

$$\det(r, s) = \frac{a\big(r + s + 4\big)}{48}, \tag{17}$$

where a is the area of the triangle. This fact allows us to factor out the determinant when stitching together two quadrilaterals in the same triangle, and it allows us to factor out the determinant up to a constant when stitching together two adjacent triangles.

We can define constants q_{b_2}, q_{c_2}, and q_{d_2} on u_2, and q_{b_3}, q_{c_3}, and q_{d_3} on u_3.

First consider finding the derivative perpendicular to the left edge of u_2. Since $r = -1$, we can write $\det(-1, s) D_\perp(-1, s) = (q_{c_2} - q_{d_2}) u_r - (q_{b_2} + q_{d_2} s) u_s$. We can easily compute u_s by taking the derivative of u_Γ along the edge. The factor of $-(q_{b_2} + q_{d_2} s) u_s$ is represented by the matrix operator

$$A_{\Gamma\Gamma}(1\!:\!n, 1\!:\!n) = -(q_{b_2} I + q_{d_2} M_1[x]) D_1. \tag{18}$$

The u_r term requires a derivative independent of u_Γ. We can reorganize the equation for u to get

$$u(r, s) = \sum_{i=0}^{n-1} T_i(s) \left(\sum_{j=0}^{n-1} a_{ij} T_j(r) \right), \qquad u_r(r, s) = \sum_{i=0}^{n-1} T_i(s) \left(\sum_{j=0}^{n-1} a_{ij} T_j'(r) \right). \tag{19}$$

We know that $T_i'(1) = i^2$ and $T_i'(-1) = (-1)^{i+1}i^2$. We will modify the submatrix $A_{\Gamma 2}(1:n,:)$, which corresponds to inner edge 1 and u_2. Let us create a $n \times n^2$ matrix $\mathbf{D}_{\perp_r}$. Row j of $\mathbf{D}_{\perp_r}$ has elements $j, n+j, 2n+j, \ldots, n(n-1)+j$ set to $[0, 1, -4, 9, -16, \ldots, (-1)^n(n-1)^2]$. Finally, we set $A_{\Gamma 2}(1\!:\!n,:) = (q_{c_2} - q_{d_2})S_0\mathbf{D}_{\perp_r}$.

Now, we must consider the lower edge of u_3. We set the submatrix $A_{\Gamma 3}(1:n,:) = (q_{c_3} - q_{d_3})S_0\mathbf{D}_{\perp_s}$. Row j of $\mathbf{D}_{\perp_s}$ has elements $n(j-1)+1, n(j-1)+2, n(j-1)+3, \ldots, nj$ set to $[0, 1, -4, 9, -16, \ldots, (-1)^n(n-1)^2]$.

Finally, we set $A_{\Gamma\Gamma}(1\!:\!n, 1\!:\!n) = -((q_{b_2} + q_{c_3})I + (q_{d_2} + q_{d_3})M_1[x])D_1$.

The highest order derivative rows of u_Γ are sacrificed in order to set continuity boundary conditions. Rows $n, 2n, 3n$ of $A_{\Gamma i}$ and $A_{\Gamma\Gamma}$ are set to zero, as well as rows $n-1$ and $2n-1$. Rows $n, 2n, 3n$ of $A_{\Gamma\Gamma}$ use to set the outside nodes of u_Γ to the exterior boundary conditions. Rows $n-1$ and $2n-1$ are used to set the first and second inside nodes of u_Γ to the third inside node of u_Γ. One derivative boundary condition is left in place.

3.3 Constructing differential operators on meshes

The key to designing a mesh solver is strict bookkeeping. Every triangle in the mesh is given a unique number, as is every vertex and every edge. We can define our mesh with a list of coordinates of vertices, and a list of the vertices contained in each triangle. It is important that all of the vertices are numbered in counterclockwise order. Next, each edge can be given a unique number. We must also list the two vertices and triangles adjacent to each edge, as well as the outer edge number of each triangle that contains that edge. Finally, we must complete the vertex list by listing whether the vertex is on the interior of the mesh, and also listing one edge that contains that vertex.

Now that we have organized the mesh, we can begin constructing matrices. We can generate A_{ii} matrices for each quadrilateral, and we can construct the $A_{\Gamma i}$, $A_{i\Gamma}$, and $A_{\Gamma\Gamma}$ matrices for each triangle. Running through the edge list, we handle two separate cases—exterior boundary edges and interior boundary edges. For each exterior boundary edge, we set the outside boundaries of the quadrilaterals and the center node to the specified boundary conditions. For each interior boundary edge, we set continuity and differentiability conditions across the edge and at the center nodes, similarly to inside of each triangle. Finally, we run through the vertex list. Outside vertices have interior edge endpoints set to the exterior boundary conditions. Interior vertices have all of the interior edge endpoints containing that vertex set to the one edge endpoint listed in the vertex list. We use the Schur complement method to solve the problems of the universe, and voilà, we have a solver that even works on skinny triangles.

4 Stability and Conditioning

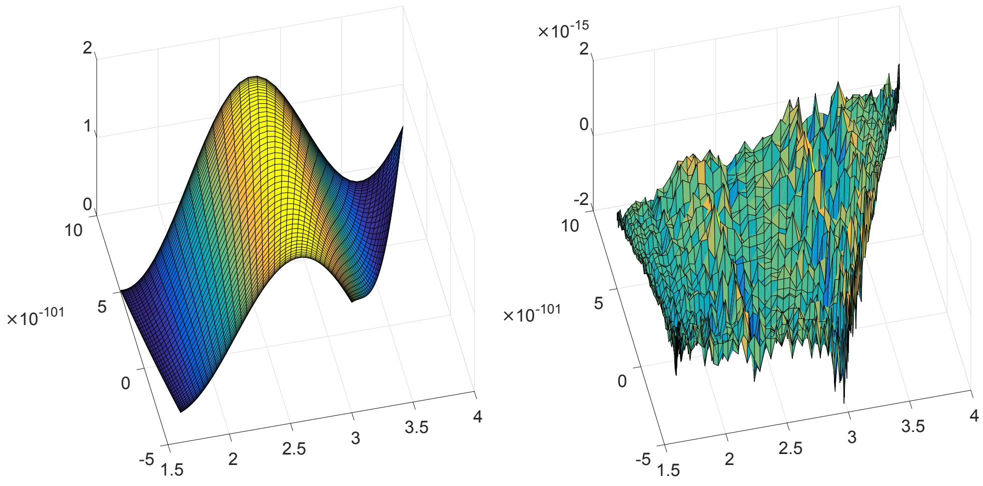

Figure 3: Even on skinny quadrilaterals, our method is still numerically stable. Our only limitation on the skinniness of mesh elements is roundoff error in actually defining the element.

We will prove that Poisson's equation $\nabla^2 u = f$ and the screened Poisson equation $\nabla^2 u - k^2 u = f$ are numerically stable on any quadrilateral. The key to the following proof is the maximum principle. The following two lemmas can be found as Theorems 1 and 2 in section 6.5.1 in [1].

Lemma 1. *Let the uniformly elliptic partial differential operator*

$$L[u] = \sum_{i,j=1}^{n} a_{ij} u_{x_i x_j} + \sum_{i=1}^{n} b_i u_{x_i},$$

with coefficients a_{ij} and b_i continuous. Let $L[u] = f$ on domain Ω with boundary $\partial\Omega$. If $f \geq 0$ on Ω, then the maximum value of u occurs on $\partial\Omega$. If $f \leq 0$ on Ω, then the minimum value of u occurs on $\partial\Omega$.

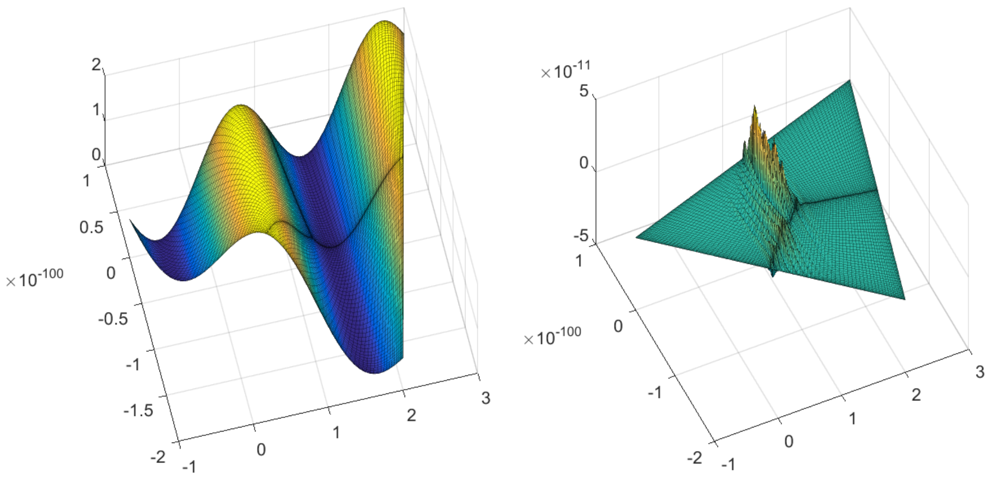

Figure 4: Using the Schur complement method, quadrilateral domains can be stitched together to solve differential equations on triangles. The solution is numerically stable even for skinny triangles.

Lemma 2. *Let the uniformly elliptic partial differential operator*

$$L[u] = \sum_{i,j=1}^{n} a_{ij} u_{x_i x_j} + \sum_{i=1}^{n} b_i u_{x_i} + cu,$$

with coefficients a_{ij}, b_i, and c continuous. Let $L[u] = f$ on domain Ω with boundary $\partial\Omega$, and let $c < 0$ on Ω. If $f \geq 0$ on Ω, then the maximum positive value of u occurs on $\partial\Omega$. If $f \leq 0$ on Ω, then the minimum negative value of u occurs on $\partial\Omega$.

We will show that Poisson's equation and the screened Poisson equation are numerically stable under perturbations to the boundary conditions and perturbations to the right hand side.

Theorem 1. *Let $L[u] = \sum_{i,j=1}^{n} a_{ij} u_{x_i x_j} + \sum_{i=1}^{n} b_i u_{x_i} + cu$, with $c \leq 0$. Let u be the solution to $L[u] = f$ on domain Ω with boundary $\partial\Omega$, and $u(\partial\Omega) = g$. If v is the solution to $L[u] = f$ with boundary conditions $u(\partial\Omega) = g + \epsilon$, then* $\max |v - u| \leq \max |\epsilon|$.

Proof. Let us write $v = u + \delta$. Since L is a linear operator, we know that $L[v] = L[u] + L[\delta] = f$. Therefore, $L[\delta] = 0$, with $\delta(\partial\Omega) = \epsilon$. By Lemma 2, we know that if δ achieves a nonnegative maximum on the interior of Ω, then δ is constant. Similarly, if δ achieves a nonpositive minimum on the interior of Ω, then δ is constant. If $\max |\delta| > \max |\epsilon|$, then δ will achieve a nonpositive minimum or a nonnegative maximum on the interior of Ω. However, Lemma 2 states that δ must then be constant, so therefore $\max |\delta| = \max |\epsilon|$, hence a contradiction. Therefore, $\max |\delta| \leq \max |\epsilon|$. $\square$

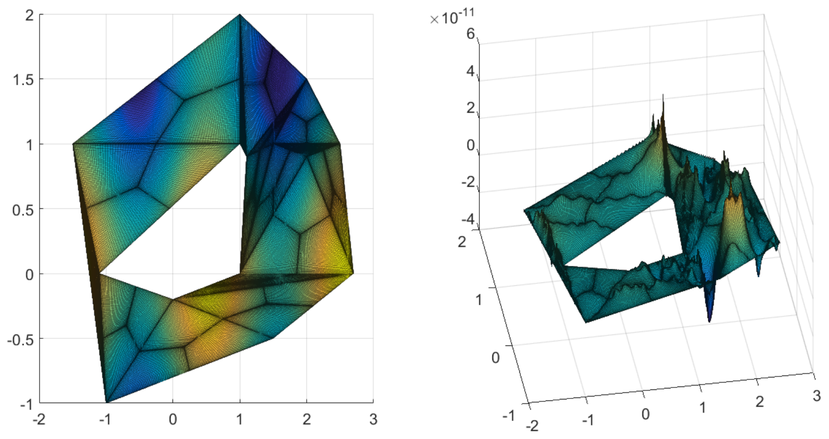

Figure 5: Our method is numerically stable even on a mesh containing both skinny and fat triangles.

Theorem 1 proves that a perturbation to the boundary conditions of any elliptic partial differential equation of the form $\sum_{i,j=1}^{n} a_{ij}u_{x_ix_j} + \sum_{i=1}^{n} b_i u_{x_i} + cu$ with $c \leq 0$ will not be amplified in the solution, even on a skinny domain. We now need to show that perturbations to the right hand side will not be amplified in the solution to the screened Poisson equation.

Theorem 2. *Let $L[u] = u_{xx} + u_{yy} - k^2u$, and let u be the solution to $L[u] = f$ on domain Ω with boundary $\partial\Omega$, and $u(\partial\Omega) = g$. Let r be the radius of the smallest circle completely containing Ω. If s is the solution to $L[s] = f + \epsilon$ with boundary conditions $s(\partial\Omega) = g$, then $\max|s-u| \leq \frac{\max|\epsilon|r^2}{4}$*

Proof. First, consider the smallest disk ω with radius r that completely contains region Ω. Let this disk have center (a, b). For $(x, y) \in \omega$ and $c < 0$, the solution $t = c\left((x-a)^2 + (y-b)^2 - r^2\right)$ is nonnegative, $\max t = -r^2c$, and $t_{xx} + t_{yy} = 4c$.

Next, consider solution w such that $w_{xx} + w_{yy} = 4c$ for $c < 0$ and $w = 0$ on $\partial\Omega$. The solution $w - t$ satisfies $(w-t)_{xx} + (w-t)_{yy} = 0$ and $w - t \leq 0$ on $\partial\Omega$, since $t \geq 0$ for all $(x, y) \in \Omega$. By Lemma 1, $(w - t) \leq 0$ for all $(x, y) \in \Omega$. Therefore, $\max w \leq -r^2c$.

Finally, we will consider the solution v such that $v_{xx} + v_{yy} = \epsilon$ with $v = 0$ on $\partial\Omega$ and $\max|\epsilon| = c$. By Lemma 1, v is bounded above by w_+ satisfying $\nabla^2 w_+ = -c$ and $w_+ = 0$ on $\partial\Omega$, and v is bounded below by w_- satisfying $\nabla^2 w_- = c$ and $w_- = 0$ on $\partial\Omega$.

We can write $\epsilon = \epsilon_+ + \epsilon_-$, where $\epsilon_+ = \max(\epsilon, 0)$, and $\epsilon_- = \min(\epsilon, 0)$.

Let δ_+ be the solution to $L[\delta_+] = \epsilon_-$ with zero Dirichlet boundary conditions and let δ_- be the solution to $L[\delta_-] = \epsilon_+$ with the same boundary conditions. By Lemma 2, $\delta_+ \geq 0$ on Ω, and $\delta_- \leq 0$.

We can write that $\nabla^2 \delta_+ = k^2\delta_+ + \epsilon_-$. Since $\delta_+ \geq 0$, $\min(k^2\delta_+ + \epsilon_-) \geq \min(\epsilon_-)$, so δ_+ is bounded above by the solution of $\nabla^2 v = \epsilon_-$ with zero Dirichlet boundary conditions by Lemma 1. Similarly, δ_- is bounded below by the solution of $\nabla^2 v = \epsilon_+$ with zero Dirichlet boundary conditions. Therefore, $s = u + \delta_+ + \delta_-$. Thus, $\max|s-u| = \max(\max\delta_+,\ \max -\delta_-) \leq \frac{\max|\epsilon| r^2}{4}$. $\square$

The general case of Theorem 2 with implicitly defined constants can be found as Theorem 3.7 of [2]. However, Theorem 3.7 of [2] does not explicitly provide bounds on the maximum amplitude of the perturbation.

Theorem 1 proves that any change to the boundary conditions of the screened Poisson equation will not be amplified in the solution on any domain. Theorem 2 proves that any change to the right hand side of the screened Poisson equation will induce a perturbation in the solution of a dependent on the radius of the domain. Therefore, a numerically stable algorithm exists that can solve the screened Poisson equation accurately on any domain, so solving the screened Poisson equation on a skinny domain is a well-posed problem.

In order to demonstrate that the condition number is bounded, let us define a skinny triangle with coordinates $(0,0),(1,1+\epsilon),(2,2-\epsilon)$. We can now define a skinny quadrilateral with vertices $(0,0),(0.5,0.5+0.5\epsilon),(1,1),(1,1-0.5\epsilon)$. When we construct a differential operator matrix on that skinny quadrilateral, we still need a preconditioner. In fact, we found that row scaling so that each row has a supremum norm of 1 is fairly close to optimal. Figure 4 shows the condition number (κ) of the normalized matrix for solving Poisson's equation as ϵ approaches zero. Even as the quadrilateral becomes very skinny, the condition number of the matrix is bounded from above.

ϵ	1	10^{-1}	10^{-2}	10^{-3}	10^{-6}	10^{-9}	10^{-12}
$\kappa\ \ (\times 10^4)$	0.159793	0.316152	0.963208	1.076368	1.083209	1.083215	1.083215

Figure 6: As ϵ approaches 0, the condition number is bounded from above.

5 The Navier–Stokes Equations

We decided to demonstrate our method with the direct numerical simulation of the Navier–Stokes equations in two dimensions at moderately high Reynolds numbers. For incompressible flows, the Navier–Stokes equation are

$$\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla)\mathbf{u} + \nabla p = \nabla^2 u, \qquad \nabla \cdot \mathbf{u} = 0, \tag{20}$$

where $\mathbf{u}$ is the velocity vector field and p is the internal pressure field. Note that these equations set density and viscosity to 1, so the Reynolds number can be varied solely by changing the velocity of the fluid or the dimensions of the domain. Since these equations are coupled and nonlinear, we must use a multistep method. We can use a first-order projection method to linearize and decouple the Navier–Stokes equations. With no-slip boundary conditions, we have

$$\begin{aligned}
&\nabla^2 \mathbf{u}^{n+1/2} - \frac{\mathbf{u}^{n+1/2}}{\triangle t} = (\mathbf{u}^n \cdot \nabla)\mathbf{u}^n - \frac{\mathbf{u}^n}{\triangle t}, && \mathbf{u}^{n+1/2} = 0, \quad \text{on } \partial\Omega \\
&\nabla^2 p^{n+1} = \frac{\mathbf{u}^{n+1/2}}{\triangle t}, && \frac{\partial p^{n+1}}{\partial \mathbf{n}} = 0, \quad \text{on } \partial\Omega \\
&\mathbf{u}^{n+1} = \mathbf{u}^{n+1/2} - \triangle t \nabla p^{n+1}, &&
\end{aligned} \tag{21}$$

where $\mathbf{n}$ is the normal vector to the boundary [4].

Navier–Stokes simulations are extremely useful in the form of wind tunnel simulations, where a test object is placed in a steady flow and analyzed. For an incompressible flow, different boundary conditions are applied at the inlet, outlet, walls, and test object. Typically, the test object will have no-slip boundary conditions, and the walls will have free-slip boundary conditions. The inlet has a constant fluid velocity and zero pressure gradient, and the outlet has a constant pressure and zero velocity gradient.

Figure 5 shows a wind tunnel simulation of an object in a moving fluid. A tilted ellipse was chosen to generate asymmetries in the flow, leading to vortex shedding earlier in the simulation. Future simulations will incorporate skinny elements and higher Reynolds numbers.

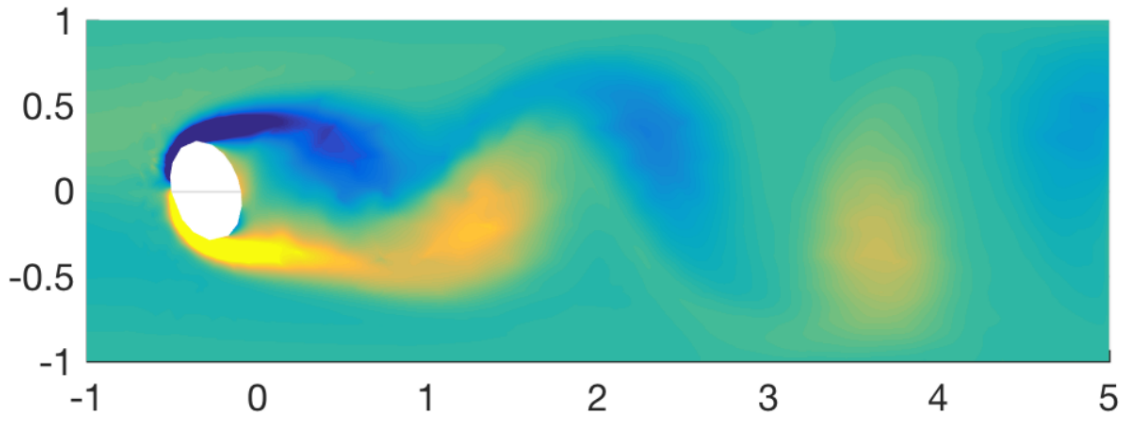

Figure 7: At a moderately high Reynolds number (Re $\approx$ 400), vortex shedding is clearly visible. Positive vorticity is shown as yellow, and negative vorticity is dark blue. This mesh has 188 triangles, with each triangle composed of three quadrilaterals with 20×20 discretization size. The color in this image is scaled to bring out more detail in the trailing vortices. The simulation is equivalent to air flow through a 2 cm $\times$ 6 cm rectangle at 0.15 m/s. In total, the simulation took about 4 hours to run 4000 time steps.

6 Future research

In its current form, our method is capable of solving time-dependent, coupled, and nonlinear differential equations stably on a mesh with skinny triangles. For a quadrilateral with an $n \times n$ discretization size, our method has a time complexity of $\mathcal{O}(n^4)$ for LU decomposition and $\mathcal{O}(n^3)$ for back-substitution for a single right hand side. Dan Fortunado, a graduate student at Harvard, is currently attempting to create a faster method with a time complexity of $\mathcal{O}(n^2 \log n)$ per element.

In addition, solving for the interior boundaries of large meshes currently has a time complexity of $\mathcal{O}(n^3 N^3)$, where N is the number of elements, each with discretization size $n \times n$. A hierarchical method can be used to divide the mesh into many different sub-domains, and then stitch the sub-domains back together using the Schur complement method. This approach has a complexity of $\mathcal{O}(n^3 N^{1.5})$.

Our method can also be adapted to operate on 3-dimensional meshes, where each tetrahedron would be composed of four hexahedra.

7 Acknowledgments

My mentor Dr. Alex Townsend has been incredibly enthusiastic and accessible. He has given me great advice, and he has been a wonderful sounding board for my ideas. In addition, Dr. Pavel Etingof, Dr. Slava Gerovitch, and the MIT PRIMES program have provided providing the opportunity for this research, and Dr. Tanya Khovanova provided astute feedback on my work. I would like to thank Dr. Grady Wright for helping me debug my Navier–Stokes simulation. I would also like to thank my teachers and administrators for incubating my desire to learn. Finally, I would like to thank my parents for allowing me to prioritize mathematical research above everything else.

A Determinants

Let (x_1, y_1), (x_2, y_2), and (x_3, y_3) be the vertices of an arbitrary triangle in counterclockwise order. By the shoelace formula, the area of the triangle is

$$A = \frac{1}{2}\Big[\big(x_1y_2 + x_2y_3 + x_3y_1\big) - \big(x_2y_1 + x_3y_2 + x_1y_3\big)\Big]. \tag{22}$$

We can construct three quadrilaterals from the triangle by partitioning the triangle along the line segments connecting the centroid of the triangle to the midpoints of the sides. Without loss of generality, let our quadrilateral contain vertex 1 of the triangle, so it has the following vertices in counterclockwise order:

$$(x_1, y_1), \quad \left(\frac{x_1+x_2}{2}, \frac{y_1+y_2}{2}\right), \quad \left(\frac{x_1+x_2+x_3}{3}, \frac{y_1+y_2+y_3}{3}\right), \quad \left(\frac{x_1+x_3}{2}, \frac{y_1+y_3}{2}\right). \tag{23}$$

Using equations (7) and (23), we have

$$x = \frac{1}{24}\Big[\big(14x_1 + 5x_2 + 5x_3\big) + \big(4x_1 - 5x_2 + x_3\big)r + \big(4x_1 + x_2 - 5x_3\big)s + \big(2x_1 - x_2 - x_3\big)rs\Big], \tag{24}$$

and similarly for y. We can use equations (6) and (9) to find that

$$\det(r, s) = \big(b_1c_2 - b_2c_1\big) + \big(b_1d_2 - b_2d_1\big)r + \big(c_2d_1 - c_1d_2\big)s. \tag{25}$$

Plugging in values for b_1, c_1, d_1, b_2, c_2, and d_2, we have

$$\begin{aligned}\det(r, s) = \frac{1}{576}\Big[&\big((4x_1 - 5x_2 + x_3)(4y_1 + y_2 - 5y_3) - (4y_1 - 5y_2 + y_3)(4x_1 + x_2 - 5x_3)\big) \\ &+\big((4x_1 - 5x_2 + x_3)(2y_1 - y_2 - y_3) - (4y_1 - 5y_2 + y_3)(2x_1 - x_2 - x_3)\big)r \\ &+\big((4y_1 + y_2 - 5y_3)(2x_1 - x_2 - x_3) - (4x_1 + x_2 - 5x_3)(2y_1 - y_2 - y_3)\big)s\Big].\end{aligned} \tag{26}$$

We can note that

$$\left(\sum_{i=1}^{n} v_i x_i\right)\left(\sum_{i=1}^{n} w_i y_i\right) - \left(\sum_{i=1}^{n} w_i x_i\right)\left(\sum_{i=1}^{n} v_i y_i\right) = \sum_{1\le i<j\le n} (x_i y_j - x_j y_i)(v_i w_j - v_j w_i), \tag{27}$$

and use this identity to simplify (26) to obtain

$$\det(r,s) = \frac{1}{96}\Big[\big(x_1y_2 + x_2y_3 + x_3y_1\big) - \big(x_2y_1 + x_3y_2 + x_1y_3\big)\Big]\big(4 + r + s\big). \tag{28}$$

Substituting in (22) and letting A be the area of the triangle gives the desired result of

$$\det(r,s) = \frac{A\big(4 + r + s\big)}{48}, \tag{29}$$

B Optimization

When solving systems of equations, banded matrices are typically efficient to solve. Unfortunately, our matrices are "almost banded," meaning that most of the elements lie close to the main diagonal, but a few rows extend the whole length of the matrix. Therefore, sparse LU decomposition and Gaussian elimination have large backfill—they are forced to set many zero matrix elements to nonzero values, resulting in a more computationally intensive solve. We found that we can use the Woodbury matrix identity to replace the rows lying outside of the bandwidth. The Woodbury matrix identity can compute the inverse a matrix given the inverse of another matrix and a rank-k correction. It satisfies

$$(A + UCV)^{-1} = A^{-1} - A^{-1}U(C^{-1} + VA^{-1}U)^{-1}VA^{-1}, \tag{30}$$

where A is n-by-n, U is n-by-k, V is k-by-n, and C is k-by-k [3].

In our case, we choose V to be a block of boundary condition rows, U to be a sparse binary matrix with exactly one '1' in each column and no more than one '1' in each row, and C to be the identity matrix. The replacement boundary condition rows are 1's in columns corresponding to the lowest-order coefficients of the solution. Our matrix A with new boundary conditions is now sparse and banded, so fast sparse LU decomposition is possible. Therefore, once the LU decomposition is completed, back-substitution can be evaluated for each individual right hand side extremely rapidly, allowing a system of variables with over 60,000 degrees of freedom to be solved in under a second.

References

[1] L. C. Evans, *Partial Differential Equations.*

[2] D. Gilbarg and N. S. Trudinger, *Elliptic partial differential equations of second order*, Springer, 2015.

[3] N. J. Higham, *Accuracy and stability of numerical algorithms*, SIAM, 2002.

[4] J. G. Liu, *Projection method I: convergence and numerical boundary layers*, SIAM journal on numerical analysis, 32 (1995), pp. 1017–1057.

[5] T. Mathew, *Domain decomposition methods for the numerical solution of partial differential equations*, vol. 61, Springer Science & Business Media, 2008.

[6] J. Ruppert, *A new and simple algorithm for quality 2-dimensional mesh generation*, in SODA, vol. 93, 1993, pp. 83–92.

[7] A. Townsend, *Computing with Functions in Two Dimensions*, PhD thesis, Oxford University, 2014.

[8] A. Townsend and S. Olver, *The automatic solution of partial differential equations using a global spectral method*, Journal of Computational Physics, 299 (2015), pp. 106–123.

[9] L. N. Trefethen, *Approximation theory and approximation practice*, SIAM, 2013.