Unstructured adaptive mesh generation and sparse matrix storage applied to Stokes flow around cylinders

Cameron Bracken

Environmental Resources Engineering, Humboldt State University, Arcata, CA, USA.

Abstract. The flow field around cylindrical obstructions is computed by solving the Stokes equations. We were able to reproduce well know flow patterns around cylinders at very low Reynolds numbers. An *h*-Method adaptive meshing algorithm is developed using a heuristic error indicator. Pressure tended to produce the meshes with the lowest relative error in the least number of refinements. The meshes produced using u, v and p separately as error indicators were markedly different. A significant reduction in computation time was achieved using a sparse matrix storage and linear algebra software.

1. Introduction

The Naiver-Stokes (NS) equation is a set of vector valued nonlinear partial differential equations (PDE's) which apply to flowing fluid. Common applications for the NS equations include ocean currents, weather, river hydraulics and airplane flight [*Chung*, 2002]. The Clay Mathematical Institute lists the theory relating to the NS equations as one of the seven most important outstanding problems in modern mathematics [*Clay Mathematics Institute*, 2000]. The NS equations are the cornerstone of a field known as Computational Fluid Dynamics (CFD). The equations themselves represent conservation of momentum principals applied to a control volume. The solution to the NS equations are a set of functions $(\mathbf{V}(\mathbf{x}, t), \rho(\mathbf{x}, t), p(\mathbf{x}, t))$, where $\mathbf{v}$ is the velocity field, ρ is the density field and p is the pressure field.

The linear form of the Navier-Stokes equations are called the Stokes equations. These equations are typically only applicable under extremely low Reynolds numbers ($Re \ll 1$), also known as creeping flow [*White*, 2006]. These conditions may involve high fluid viscosities or low velocities or a combination of both.

Mesh generation is fundamental to numerical solutions of PDE's via the Finite Element (FE) method. Node placement and mesh creation is often more of an art than a science, if not an arbitrary decision of the modeler. Adaptive meshing techniques allow for efficient generation of "optimal meshes," that is meshes which minimize error and computation time. Adaptive mesh generation, while diverse in form, serves to negate the issue of manual node placement, manual mesh creation and global mesh refinement.

The issue of mesh generation comes hand in hand with the issue of efficient solutions of the resulting system of equations. The complete solution to the flow equations with a general mesh is computationally intensive if solved via direct methods. In general the resulting matrices are sparse and a large time and memory savings may be achieved by exploiting sparse matrix storage and solution methods.

2. Problem Formulation

We seek to solve the steady Stokes equations in 2D with cylindrical obstructions using an adaptively generated finite element (FE) mesh. The Stokes equations are solved via the FE method. The unknowns of interest are velocity and pressure, $(\mathbf{v}(x, y), p(x, y))$. Integrations are done in global coordinates via 4 point Gaussian quadrature. The freely available software `Triangle` is used to generate meshes given a set of nodes [*Shewchuk*, 1996]. The adaptive meshing algorithm uses an unstructured, *h*-adaptive, one-way refinement scheme based on a heuristic error indicator [*Bangerth and Rannacher*, 2003]. One-way refers to the fact that the algorithm only refines and does not derefine. Average relative error of each method is compared.

The Stokes simulation model utilizes sparse matrix storage storage for the coefficient matrix A. The corresponding system of linear equations is solved using sparse matrix library `SLAP` *Seager* [1988] as well as the dense matrix library `LAPACK` [*Anderson et al.*, 1999]. Solution times using both sparse and dense solvers are compared.

3. Literature Review

The CFD has historically been dominated by the finite difference (FD) method [*Chung*, 2002]. FD methods tend to have much simpler formulations and be less computationally intensive but they do not support unstructured grids. One of the earliest applications of the FD method in CFD was *Courant et al.* [1928].

FE formulations require more mathematical rigor than FD methods but they support unstructured meshes and irregular problem domains. FE work was first published by *Turner et al.* [1956]. 2D flow past a cylinder is a common application of the Stokes equations [*Bangerth and Rannacher*, 2003]. One aim in modeling flow around a cylinder is to calculate the drag coefficient.

3.1. Grid Generation

There are two main types of FE grid generation,

1. *Structured Grid generation* involves some pattern of isoparametric elements which exploit the geometry of a problem domain for efficient mesh generation. Structured grid generation may need to be preformed piecewise for complex geometries. Structured grids apply only in 2 and 3D.

2. *Unstructured Grid generation* involves the use of Delaunay-Voronoi triangulation of a set of points. Many algorithms for constrained of Delaunay triangulation exist including the Watson algorithm *Watson* [1981] and `Triangle` [*Shewchuk*, 1996].

3.2. Adaptive Meshing

Adaptive meshing techniques can apply to structured or unstructured mesh generation in 1, 2, or 3D. Let E_e be an estimate of the actual error $\mathscr{E}_e$ error for element e. Adaptive meshing is used to refine or unrefine (coarsen) meshes based on E_e, (usually based on the solution gradient) [*Chung*, 2002]. E_e is called an a priori error estimate because to calculate, it requires a pre-computed solution. The focus here will be in unstructured adaptive methods. There are three main methodologies in unstructured mesh refinement

Computational Methods for Engineers Spring 2008.
Humboldt State University.

1. *h-Methods* also known as mesh refinement methods, refine elements which have a high a priori error by adding nodes and vertices.

2. *r-Methods* also known as mesh movement methods, do not add any new nodes but simply move existing nodes in order to obtain the lowest possible global error. This may involve solving an optimization problem.

3. *p-Methods* also known as mesh enrichment methods, increase the degree of the polynomial interpolating function for a particular element with high a priori error estimate.

Additionally many hybrid methods exist which combine the ideas from the three basic adaptive mesh methods. Some of these hybrid methods are combined mesh refinement and mesh movement (*hr*-methods) and combined mesh refinement and mesh enrichment (*hp*-methods). For additional information on adaptive mesh generation methods, see *Chung* [2002].

For all unstructured adaptive mesh techniques, the general procedure involved is

1. Solve the simulation model.

2. Obtain an estimate of the local relative error over each element using some error indicator (typically u, v, p or ρ in a CFD problem).

3. Identify elements with high relative error.

4. Increase the accuracy of the solution over elements which have high error and

5. Repeating until a global error tolerance is met.

3.3. Sparse Matrix Storage and Computation

1D coefficient matrices resulting from the FE method are tightly banded and may be stored and solved efficiently. For structured meshes in 2D coefficient matrices are typically banded and thus banded storage and solution algorithms can be used with savings over dense solvers [*Burkardt*, 2005c]. For unstructured meshes in 2D, coefficient matrices are typically sparse and unbanded. The sparsity is especially pronounced for adaptively generated meshes. Thus, general sparse matrix storage and computation is crucial for efficient solutions to large problems.

Sparse matrix data structures come in many forms all of which attempt to only store nonzero entries of a sparse matrix. The most simple storage structure is called the coordinate format or the Triad format [*Seager*, 1988] which stores the row number, column number and nonzero entries in vectors of equal length. Other, more efficient storage structures include the compressed sparse row format, compressed sparse column format, and the sparse block storage format. The coordinate format is used in this paper.

If we let n_s be the number of nonzero entries in the matrix A of dimension N, then the coordinate format will only provide storage savings if $3n_s < N^2$. Dense matrix storage increases as N^2 where sparse storage increases linearly with N.

4. Model Formulation and Development

This section will address the simplification of the Navier-Stokes equations, finite element formulation, quadrature rule over the general triangle and the adaptive meshing algorithm.

4.1. Navier-Stokes Simplification

The NS Equations for unsteady compressible flow are

$$\rho \mathbf{V}_t - \mu \nabla^2 \mathbf{V} + \rho(\mathbf{V}\cdot\nabla)\mathbf{V} + \nabla p = 0 \tag{1}$$

where μ is the kinematic viscosity [*Chung*, 2002]. The NS equations are typically coupled with the continuity equation for fluids

$$\rho_t + \nabla\cdot(\rho\mathbf{V}) = 0. \tag{2}$$

If we assume that the flow is steady, incompressible and uniform in the z direction then $\mathbf{V}(\mathbf{x}, t) = \mathbf{V}(x, y) = (u(x, y), v(x, y))$. If we further construct a situation involving very low Reynolds number ($Re < 10$), the nonlinear term in equation 1 will be small resulting in the Stokes equations

$$-\nu\nabla^2\mathbf{V} + \nabla p = 0 \tag{3}$$

where p has been redefined as the scaled pressure $p := p/\rho$. The associated continuity equation is

$$\nabla\cdot\mathbf{V} = 0. \tag{4}$$

The Stokes equations are linear and the solution is the set of functions $(\mathbf{V}(x, y), p(x, y))$.

4.2. Finite Element Formulation

We begin by constructing a piecewise approximate solution to the Stokes equation $\hat{\mathbf{V}}(x, y)$ and $\hat{p}(x, y)$ where

$$\mathbf{V}(x, y) \approx \hat{\mathbf{V}}(x, y) = \sum_{j=1}^{n_v} \phi_j(x, y)\tilde{\mathbf{V}}_j \tag{5}$$

$$p(x, y) \approx \hat{p}(x, y) = \sum_{j=1}^{n_p} \psi_j(x, y)\tilde{p}_j \tag{6}$$

where $\phi_i(x, y)$ and $\psi_i(x, y)$ are the interpolating functions associated with velocity and pressure respectively, $\tilde{\mathbf{V}}_j$ and $\tilde{p}_j$ are the nodal values velocity and pressure, n_v is the number of velocity nodes and n_p is the number of pressure nodes. This formulation allows us to have a different number of velocity and pressure nodes. From this point on the explicit dependency in x and y will be dropped in the interest concision.

The necessary conditions of the FE method, which minimize global error of the approximate solution, are

$$\int_\Omega \mathbf{w}_i \mathscr{L}\{\hat{\mathbf{V}}, \hat{p}\}\, d\Omega = 0 \tag{7}$$

where Ω is the problem domain, $\mathbf{w}_i$ are weighting functions and $\mathscr{L}$ is the Stokes operator given by equations 3 and 4 [*Willis*, 1987]. The Galerkin method sets the weighting functions equal to the interpolating functions such that $\mathbf{w}_i = (\phi_i, \psi_i)$. The equations so far are not very useful to us, but if we apply the Stokes operator and multiply by the weighting functions we start to formulate the useful FE equations

$$\int_\Omega \phi_i(\nu\nabla^2\hat{\mathbf{V}} + \nabla\hat{p})\, d\Omega = 0 \tag{8}$$

$$\int_\Omega \psi_i \nabla\cdot\hat{\mathbf{V}}\, d\Omega = 0. \tag{9}$$

which are known as the strong form of the Stokes equations. A solution to these equations is a so called strong solution. A strong solution, due to the second derivative term, requires the interpolating functions be at least $C^1(\Omega)$ functions [*Burkardt*, 2005c].

We can derive the weak form of the Stokes equations via integration by parts, which states that for a continuously differential functions B and $\mathbf{F}$ defined over some region Ω

$$\iint_\Omega B\nabla^2\mathbf{F}\, d\Omega = -\iint_\Omega \nabla B\nabla\mathbf{F}\, d\Omega + \oint_{\partial\Omega} B\nabla\mathbf{F}\cdot\mathbf{n}\, ds. \tag{10}$$

where B is a general function and $\mathbf{F}$ is a vector valued function both defined on Ω and $\mathbf{n}$ is an outward pointing unit normal vector on $\partial\Omega$ [*Marsden and Tromba*, 2003].

Letting $B = \phi_i$ and $\mathbf{F} = \hat{\mathbf{V}}$

$$\iint_A \nu \nabla\phi_i \cdot \nabla\hat{\mathbf{V}} + \phi_i \nabla p \; dA = \oint_s \phi_i \nabla\hat{\mathbf{V}} \; ds. \tag{11}$$

$$\iint_A \psi_i \nabla \cdot \hat{\mathbf{V}} \; dA = 0 \tag{12}$$

which are known as the weak form of the Stokes equations. The solution to these equations are only required to be $C^0(\Omega)$ functions which will simplify the solution process and allow us to use linear basis functions *Burkardt* [2005b].

Let us examine equation 11 in order to explicitly formulate the equations associated with the weak Stokes equations. There are $2n_v + n_p$ linear equations associated with these equations 11 and 12 because equation 11 is vector valued with two components. Substitute in the piecewise approximate solutions $\hat{\mathbf{V}}$ and $\hat{p}$ into the left hand side of equation 11

$$\iint_A \nu \nabla\phi_i \nabla \sum_{j=1}^{n_v} \phi_j \tilde{\mathbf{V}}_j + \phi_i \nabla \sum_{j=1}^{n_p} \psi_j \tilde{p}_j \; dA = \oint_{\partial A} \phi_i \nabla\hat{\mathbf{V}}_j \; ds$$

$$\iint_A \nu \sum_{j=1}^{n_v} \nabla\phi_i \nabla\phi_j \tilde{\mathbf{V}}_j + \sum_{j=1}^{n_p} \phi_i \nabla\psi_j \tilde{p}_j \; dA = \oint_{\partial A} \phi_i \nabla\hat{\mathbf{V}}_j \; ds \tag{13}$$

Analogously for equation 12 we can write

$$\iint_A \sum_{j=1}^{n_v} \tilde{\mathbf{V}}_j \psi_j \nabla\phi_i \; dA = 0 \tag{14}$$

Using equations 13 and 14 we can formulate a system of linear equations with dimension $2n_v + n_p$ of the form

$$A\mathbf{g} + \mathbf{f} = 0 \tag{15}$$

where $\mathbf{g} = [\tilde{\mathbf{u}} \; \tilde{\mathbf{v}} \; \tilde{\mathbf{p}}]^T$ and T denotes transpose. It is useful to note here that the A_{ij} coefficient of $\mathbf{V}_j$ is

$$\left[\iint_A \nabla\phi_i \cdot \nabla\phi_j \; dA + \iint_A \nabla\psi_j \phi_i \; dA \right]$$

and of p_j is

$$\left[\iint_A \phi_i \nabla\psi_j \; dA \right]$$

An algorithm to assemble the stokes equations must take into account that ϕ_i and ψ_i need not be defined at the same nodes. Also note that equation 13 is vector valued. The scalar versions will be presented in Section 5.

4.3. Adaptive Meshing Algorithm

The required components of an h adaptive mesh refinement scheme are an error indicator, an error estimate and a refinement procedure. Let θ_n be the error indicator for every node n and E_e be the a priori error estimate for element e. In the stokes problem θ^n may be horizontal velocity, vertical velocity or pressure. We define the relative percent error, E_e for an element with verticies (i, j, k) as

$$E_e = \frac{\max\left(\theta_i^e, \theta_j^e, \theta_k^e\right) - \min\left(\theta_i^e, \theta_j^e, \theta_k^e\right)}{\frac{1}{nele} \sum_{n=1}^{nele} \left[\max\left(\theta_i^n, \theta_j^n, \theta_k^n\right) - \min\left(\theta_i^n, \theta_j^n, \theta_k^n\right) \right]}$$

where $nele$ is the number of elements. This relative error estimate is is known as a heuristic type error estimte[*Bangerth and Rannacher*, 2003]. There exist many other error estimation methods which will not be discussed in this paper.

If we take E_e to be a surrogate for the actual error and h_e to be a element length scale then the assumption is that as $h_e \to 0, E_e \to 0$. Therefore we introduce the parameters A_{min}, the minimum allowable element area, ϵ, the maximum allowable relative percent error, and m the maximum allowable number of elements that do not meet the error or area criteria. The algorithm for one way adaptive meshing is (Figure 1)

1. Generate an initial sparse set of points.
2. Generate mesh.
3. Solve Stokes equations to determine I_e.
4. Evaluate E_e and $A_e$for all elements e using equation 4.3.
5. For all elements e

   (i) IF $E_e > \epsilon$ and $A_e > A_{min}$, mark element e for refinment

   (ii) ELSEIF do not mark element e for refinement.
6. IF number of elements to refine is $< m$ FINISH.
7. ELSE Refine elements marked for refinement by placing a new node at the center of the element.
8. GO TO step 2.

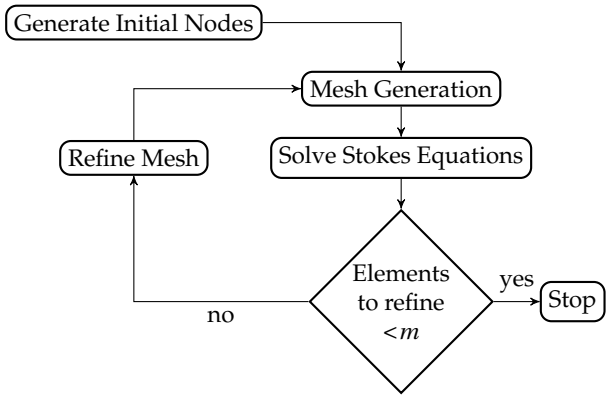


Figure 1. Mesh refinement algorithm flowchart.

5. Model Application

This section gives a formal statement of the problem in question, discusses basis functions and quadrature, defines parameter values, discusses the software used and the programming framework.

5.1. Formal Problem Statement

The Stokes equations in scalar form are

$$-\nu \frac{\partial^2 u}{\partial x^2} - \nu \frac{\partial^2 u}{\partial y^2} + \frac{\partial p}{\partial x} = 0 \tag{16}$$

$$-\nu \frac{\partial^2 v}{\partial x^2} - \nu \frac{\partial^2 v}{\partial y^2} + \frac{\partial p}{\partial y} = 0 \tag{17}$$

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0 \tag{18}$$

With the associated set of boundary conditions:

$$0 \le x \le L_x, \; 0 \le y \le L_y \tag{19}$$

$$v(x, 0) = k\,x\,(1 - x), \; u(x, 0) = 0 \tag{20}$$

$$\left.\frac{\partial \mathbf{V}}{\partial x}\right|_{(x,L_y)} = 0 \tag{21}$$

$$\mathbf{V}(0,y) = 0,\ \mathbf{V}(L_x,y) = 0 \tag{22}$$

$$p(x_o,y_o) = 0 \tag{23}$$

where (x_o, y_o) is an arbitrary point. Inflow boundary conditions are a parabolic vertical velocity profile perpindicular to the boundary. Outflow boundary conditions require that flow be perpendicular to the boundary. The left and right side and obstruction boundary conditions are no slip (zero Dirchlet). Pressure is a potential function and thus need only to be specified at a single point (x_o, y_o).

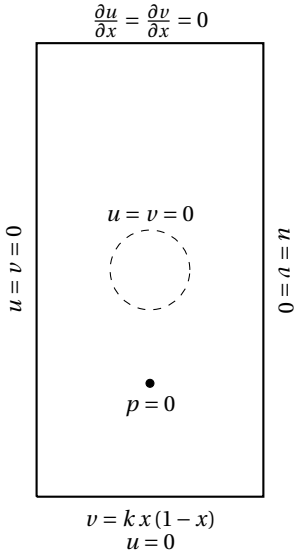


Figure 2. Problem domain with cylindrical obstruction which may or may not be present.

The FE formulation written in scalar form for any internal node

$$\iint_A \nu \sum_{j=1}^{n_v} \left\{ \frac{\partial \phi_i}{\partial x}\frac{\partial \phi_j}{\partial x} + \frac{\partial \phi_i}{\partial y}\frac{\partial \phi_j}{\partial y} \right\} \tilde{u}_j + \sum_{j=1}^{n_p} \phi_i \frac{\partial \psi_j}{\partial x} \tilde{p}_j \ dA = 0 \tag{24}$$

$$\iint_A \nu \sum_{j=1}^{n_v} \left\{ \frac{\partial \phi_i}{\partial x}\frac{\partial \phi_j}{\partial x} + \frac{\partial \phi_i}{\partial y}\frac{\partial \phi_j}{\partial y} \right\} \tilde{v}_j + \sum_{j=1}^{n_p} \phi_i \frac{\partial \psi_j}{\partial y} \tilde{p}_j \ dA = 0 \tag{25}$$

$$\iint_A \sum_{j=1}^{n_v} \left\{ u_j \frac{\partial \psi_j}{\partial x} + v_j \frac{\partial \psi_j}{\partial y} \right\} \phi_i \ dA = 0 \tag{26}$$

5.2. Basis Functions

In this section we use the terms interpolating function and basis function interchangeably. We saw in section 4.2 that we need two basis functions, ψ_i and ϕ_i, one for the momentum equation and one for the continuity equation. For this problem we will use Taylor-hood elements *Burkardt* [2005b]. These elements use linear basis functions for pressure and quadratic basis functions for a single element. Figure 3 shows an archetypal linear pressure element with nodes defined at the vertices. Figure 4 shows an archetypal quadratic velocity element with nodes defined at the vertices as well as at the midside of the vertices.

The linear basis functions for pressure as given in *Lapidus and Pinder* [1982] is

$$\psi_i(x,y) = \frac{(x-x_j)(y_k-y_j) - (x_k-x_j)(y-y_j)}{(x_i-x_j)(y_k-y_j) - (x_k-x_j)(y_i-y_j)} \tag{27}$$

Quadratic basis functions for a triangle are derived in *Burkardt* [2005b]; here we will state them without proof. For the three vertex nodes

$$\phi_i(x,y) = \frac{g_v(x,y)\,h_v(x,y)}{g_v(x_i,y_i)\,h_v(x_i,y_i)} \tag{28}$$

where

$$g_v(x,y) = (x-x_j)(y_k-y_j) - (x_k-x_j)(y-y_j) \tag{29}$$

and

$$h_v(x,y) = (x-x_{ij})(y_{ik}-y_{ij}) - (x_{ik}-x_{ij})(y-y_{ij}) \tag{30}$$

where (x_{ij}, y_{ij}) is the coordinate associated with the midside node between nodes i and j.

As given in *Burkardt* [2005b], the basis functions associated with the midside nodes are

$$\psi_{ij}(x,y) = \frac{g_m(x,y)\,h_m(x,y)}{g_m(x_{ij},y_{ij})\,h_m(x_{ij},y_{ij})} \tag{31}$$

where

$$g_m(x,y) = (x-x_k)(y_i-y_k) - (x_i-x_k)(y-y_k) \tag{32}$$

and

$$h_m(x,y) = (x-x_k)(y_j-y_k) - (x_j-x_k)(y-y_k). \tag{33}$$

It is interesting to note that

$$\psi_i(x,y) + \psi_j(x,y) + \psi_k(x,y) + \psi_{ij}(x,y) + \psi_{jk}(x,y) + \psi_{kj}(x,y) = 1 \tag{34}$$

will hold true for any non-degenerate quadratic element.

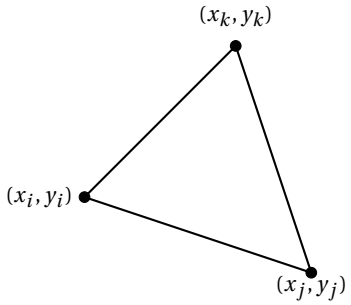


Figure 3. Archetypal linear pressure element.

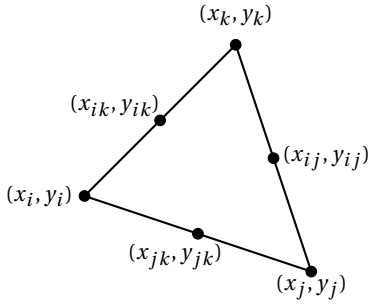


Figure 4. Archetypal quadratic velocity element.

5.3. Quadriture Over a General Triangle

The reference triangle T_r has vertices (0,0), (0,1) and (1,0) in dimensionless coordinates (ξ, η). Gaussian quadrature for the reference triangle is

$$\int_{T_r} f(\xi,\eta)\ d\xi d\eta = \sum_{p=1}^{n} w_p f(\xi_p, \eta_p). \tag{35}$$

The values of w_p, ξ_p and η_p are tabulated in *Lapidus and Pinder* [1982].

Since our basis functions (and subsequent integrals) are defined in global coordinates, we need a way to map quadrature points and weights in the reference triangle to points in the actual triangle T. The mapping is developed in *Burkardt* [2005b]. To map any point (ξ_p, η_p) to a point (x_p, y_p) in the triangle T we use

$$\begin{bmatrix} x_p \\ y_p \end{bmatrix} = \begin{bmatrix} x_i \\ y_i \end{bmatrix} + \begin{bmatrix} x_j - x_i & x_k - x_i \\ y_j - y_i & y_k - y_i \end{bmatrix} \begin{bmatrix} \xi_p \\ \eta_p \end{bmatrix} \tag{36}$$

We can write our quadrature rule as

$$\int_T f(x,y)\ dxdy = A_T \sum_{p=1}^{n} w_p f(x_p, y_p). \tag{37}$$

Where A_T is the area of the triangle which is the artifact of a constant Jacobian for our linear mapping. The area of the triangle is given by

$$A_T = \begin{bmatrix} 1 & x_i & y_i \\ 1 & x_j & y_j \\ 1 & x_k & y_k \end{bmatrix} \tag{38}$$

5.4. Software

The software library `SLAP` (Sparse Linear Algebra Package) [*Seager*, 1988] is used to solve the linear system of equations. `SLAP` is a set of matrix algebra routines, written in `Fortran 77` for large sparse linear systems and is available for free at `netlib.org`. Specifically the routine `DGMRES` is used where the coefficient matrix is stored in the `SLAP` triad format. The triad format identical to the coordinate format described in section 3.3. `DGMRES` solves the linear system $Ax = b$ using the generalized minimum residual method [*Seager*, 1988].

`LAPACK` is an industry standard library of dense and banded solvers for systems of linear equations [*Anderson et al.*, 1999]. `LAPACK` has been optimized for many computer systems and is based on the `BLAS` (Basic linear algebra subroutines) library. Here we use the version of `LAPACK` optimized by Apple computers as part of the `Accelerate` framework *Apple* [2006]. This framework uses parallel processing for maximizing computational resources. The specific routine used is the `DGETRS` which uses LU decomposition to factor the coefficient matrix.

`Triangle` is freely available software for creating 2D quality constrained meshes. By quality we mean that the mesh generator will add vertices to the mesh where appropriate to ensure that a minimum angle requirement is met [*Shewchuk*, 1996]. `Triangle` creates a conforming constrained triangulation. Each triangle may not necessarily be Delaunay but that inconsequential a finite element mesh. The software is written in `C` and therefore difficult to interface directly with `Fortran`. This issue is circumvented by writing the program framework as a bash shell script which deals in I/O files.

John Burkhardt created a Navier-Stokes simulation model called `free_fem_navier_stokes` from which some subroutines were used for this problem [*Burkardt*, 2007]. Specifically the subroutines for evaluating the basis functions and assembling the coefficient matrix.

The initial node generation program is called `regmeshgen`, the main simulation program is called `stokes_sim` the error analysis program is called `adaptive` and the mesh refining program is called `nodegen`. A flowchart of the software framework is shown in figure 5.

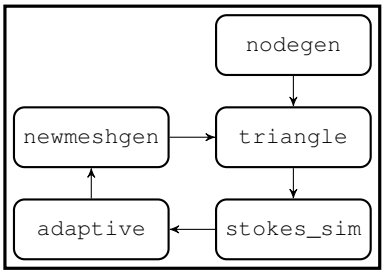


Figure 5. Software framework controlled by `BASH` shell script.

5.5. Parameters

All of the parameters presented here have been previously defined. The prameter values are given in Table 1. In addition to the parameters in table 1 the are the parameters associated with the cylindrical obstructions, namely the radii, vertices and the number of nodes used to define the obstruction.

Table 1. Parameter Values

Parameter	Value	Units
ν	1	L^2T^{-1}
L_x	1	L
L_y	2	L
k	1	$(TL)^{-1}$
ϵ	15%	–
A_{min}	0.003	L^2
m	2	–

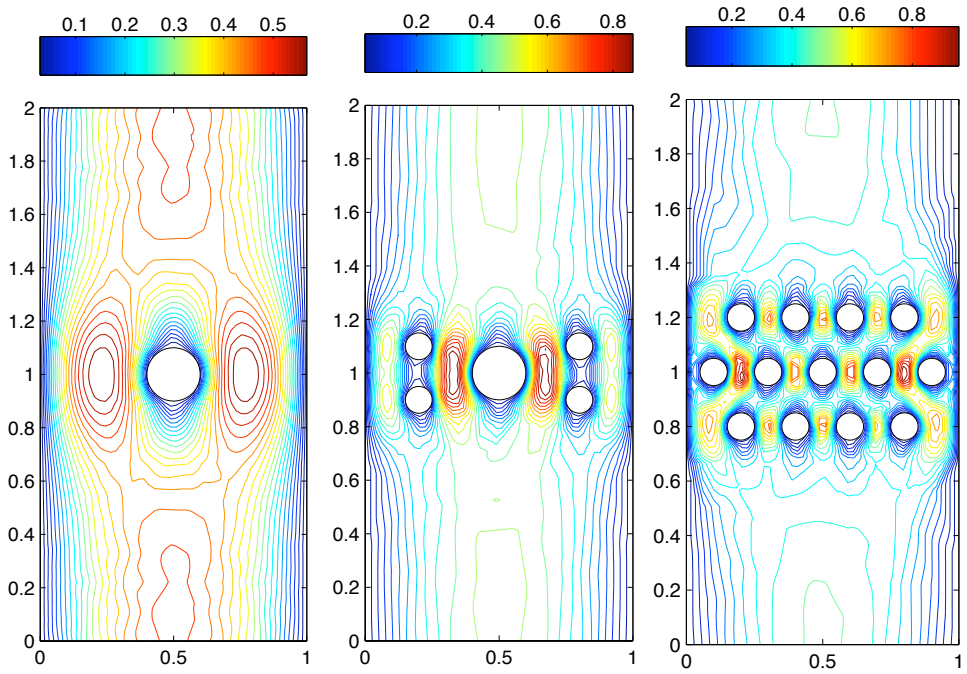
6. Model Results

6.1. Simulation Results

Simulations were carried out with the previously given set of parameters. The Reynolds number is $Re = 1/6$. Figure 6 shows the computed velocity fields for 1, 5 and 13 obstructions. Figure 7 shows velocity magnitude contours. Figure 12 shows pressure contours. The pressure solution with no obstructions is included as verification that the model is working; with no obstructions the flow is purely vertical and the pressure is approximately constant.

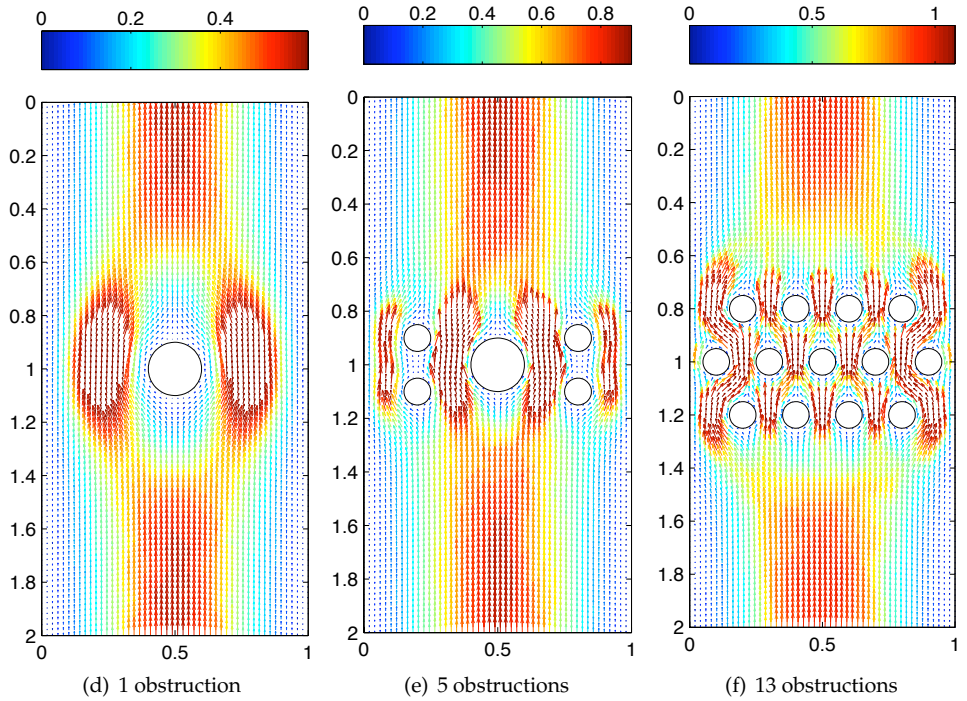
With one obstruction, flow moves around the cylinder and a stagnation point occurs at the front of the cylinder. The single cylinder caused a pressure drop of ~ 25 pressure units from the inlet to the outlet. Pressure contours show the familiar and expected pattern [*White*, 2006]. Velocity increases as the fluid moves around the cylinder.

With five obstructions, flow moves around the cylinders and a stagnation point occurs at the front of the middle cylinder. The single cylinder caused a pressure drop of ~ 90 pressure units from the inlet to the outlet. More of the flow moves around center cylinder causing increased velocity in that region.



(a) 1 obstruction (b) 5 obstructions (c) 13 obstructions

Figure 6. Velocity magnitude contours, $Re = 1/6$



(d) 1 obstruction (e) 5 obstructions (f) 13 obstructions

Figure 7. Velocity magnitude contours, $Re = 1/6$

An array of 13 cylinders caused the largest pressure drop of all (~ 250 pressure units). The flow is highly restricted causing the velocity to increase in between the cylinders. The velocity increase here is also the greatest because the flow is restricted to the smallest area. The highest velocities occurs in between the cylinders in the middle row.

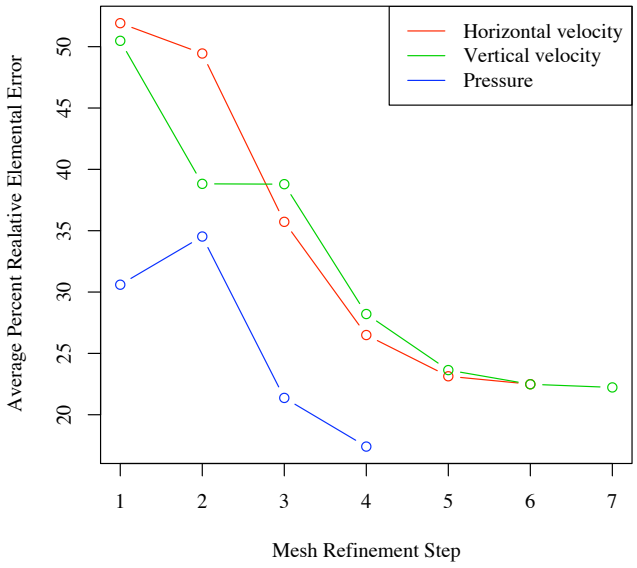


Figure 8. Average relative element error as a function of refinement step for 1 obstructions.

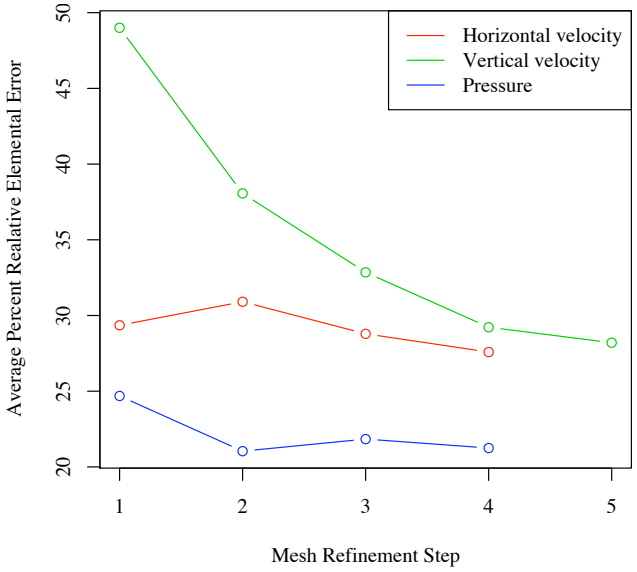


Figure 10. Average relative element error as a function of refinement step for 13 obstructions.

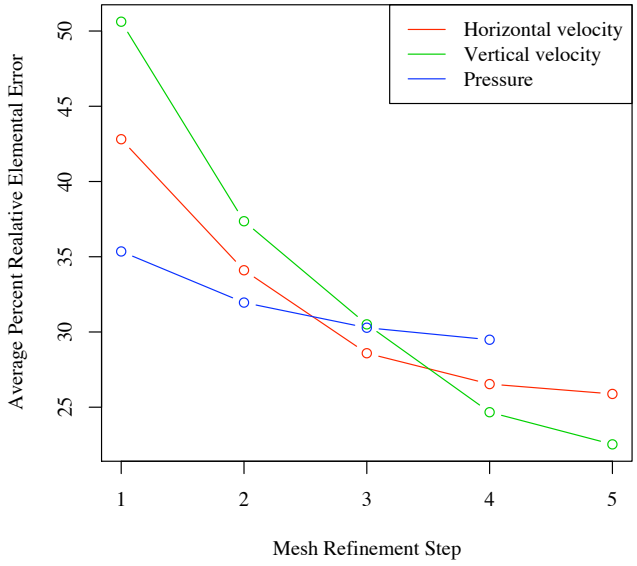


Figure 9. Average relative element error as a function of refinement step for 5 obstructions.

6.2. Adaptive Mesh Results

Table 2, 3, 4 show the details of the mesh refinement process with 1, 5 and 13 obstructions respectively. With $\theta = p$ an optimal mesh was created in the fewest refinement steps with an average relative error on the final step of ~18% (Figure 8). With pressure as the error indicator, the mesh tended to be refined only locally around the cylinder (Figure 16). Using $\theta = v$ required 3 more refinement steps and over twice as many elements in the final mesh. Using horizontal velocity as the error indicator tends to refine the mesh in a broad area around the cylinder (Figure 14). With $\theta = v$ the mesh tended to be refined near the left and right edges which are the no slip boundaries (Figure 15).

The plots of relative error (Figure 8,9 and 10) show that the average element was area limited ($A < A_{min}$) and not limited by the choice of ϵ. The imposition of a minimum area turned out to be crucial because in many elements the program ran out of precision before the acceptable error was achieved.

For the arrangement of 5 cylinders using vertical velocity as the indicator generated the mesh containing the most elements but also the mesh with the lowest relative error (Figure 9). Once again using $\theta = u$ took the fewest refinement steps. The generated meshes follow similar trends as with a single cylinder with the pressure indicator producing the most local refinement. Fewer refinement steps were taken here because

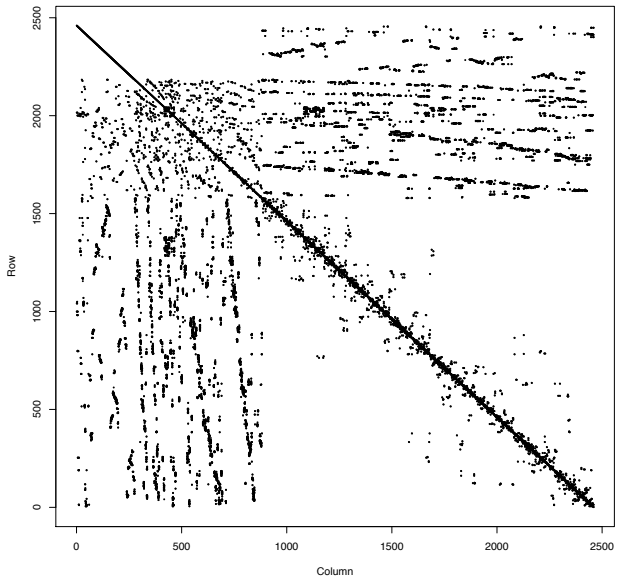
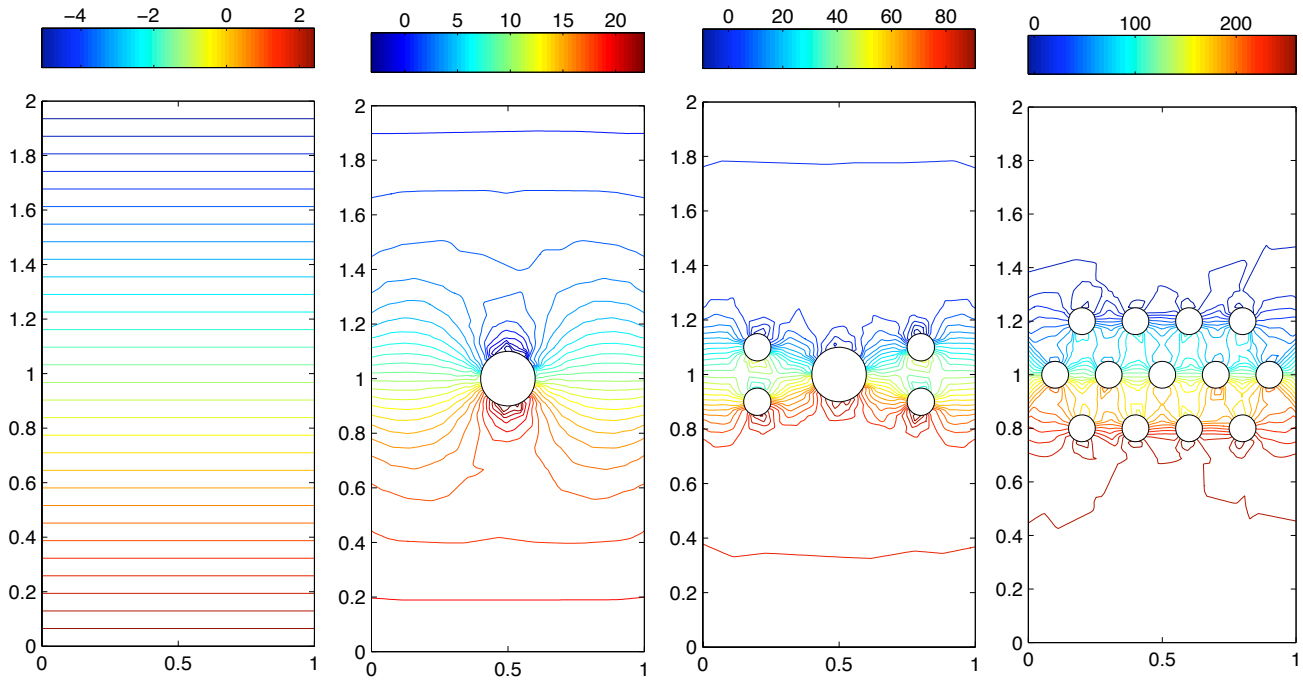


Figure 11. Sample coefficient matrix with points in the nonzero entries



(a) 0 obstructions (b) 1 obstruction (c) 5 obstructions (d) 13 obstructions

Figure 12. Pressure contours., $Re = 1/6$

Table 2. One obstruction, refinement details

	$\theta = u$			$\theta = v$			$\theta = p$		
Refinement Step	Elements	Nodes	Unknowns	Elements	Nodes	Unknowns	Elements	Nodes	Unknowns
1	88	214	491	88	214	491	88	214	491
2	161	361	822	180	406	925	178	402	916
3	300	642	1455	346	752	1707	332	722	1639
4	482	1008	2279	687	1467	3324	465	1003	2275
5	597	1241	2804	939	1971	4458	–	–	–
6	619	1285	2903	1013	2119	4791	–	–	–
7	–	–	–	1027	2147	4854	–	–	–

Table 3. Five obstructions, refinement details

	$\theta = u$			$\theta = v$			$\theta = p$		
Refinement Step	Elements	Nodes	Unknowns	Elements	Nodes	Unknowns	Elements	Nodes	Unknowns
1	189	455	1041	189	455	1041	189	455	1041
2	301	683	1555	318	718	1634	285	651	1483
3	405	897	2038	440	978	2233	309	705	1606
4	440	970	2203	598	1308	2969	315	717	1633
5	450	990	2248	674	1464	3321	–	–	–

Table 4. Thirteen obstructions, refinement details

	$\theta = u$			$\theta = v$			$\theta = p$		
Refinement Step	Elements	Nodes	Unknowns	Elements	Nodes	Unknowns	Elements	Nodes	Unknowns
1	299	735	1682	299	735	1682	299	735	1682
2	438	1016	2315	506	1162	2646	380	900	2054
3	488	1120	2550	616	1392	3166	392	924	2108
4	516	1176	2676	712	1596	3628	398	936	2135
5	–	–	–	741	1659	3771	–	–	

the mesh was more refined to begin with due to the nodes needed to define the cylinders.

For the array of 13 cylinders, the mesh was well refined to begin with and so the refinement process using pressure and horizontal velocity was not able to make much improvement on the average relative error. Despite this, pressure still achieved the lowest average relative error in the fewest refinement steps.

6.3. Sparseness and Computation Time

The assumption that the coefficient matrix resulting from an unstructured adaptive mesh refinement process would be sparse and unbanded turned out to be a good one. Figure 11 is a visualization of a sample coefficient matrix with points in the nonzero entries. The coefficient matrix contained > 99.99% zero entries, that is < .01% of the entries were zero. This was typical for the Stokes stiffness matrices.

The assertion earlier in the paper was that dense matrix storage and therefore solution time increases as N^2 where sparse storage increases linearly with N. We see from the solution times that this is indeed the case (Figure 13). Times are from 64-bit Intel Core II Duo processor. Figure 13 does not clearly show it but the LAPACK dense matrix solver is actually faster when the $N \lesssim 1000$. In this case though solution times for both sparse and dense solvers were less than 1 second.

7. Conclusions

1. We were able to reproduce well know flow patterns around cylinders at very low Reynolds numbers.

2. We implemented an unstructured adaptive meshing algorithm.

3. Pressure performed well as an error indicator

4. Using u, v and p as error indicators produced markedly different meshes.

5. Sparse matrix storage and linear algebra software drastically reduced solution time.

Notation

ρ	Fluid Density.
ρ_t	Time derivative of density.
$\mathbf{V}$	Velocity vector.
$\mathbf{V}_t$	Time derivative of velocity vector.
∇	Partial derivative operator.
∇^2	Second derivative operator.
p	Fluid pressure.
μ	Dynamic Viscocity.
ν	kinematic Viscocity.
$\mathscr{E}_e$	Actual error over element e.
E_e	Error estimate over element e.
A	Finite element coefficient matrix or alternately the area of a given finite element.
n_s	Number of nonzero entries in A.
N	Order of A.
E_e	Error estimate over element e.
Re	Reynolds number.
$\hat{\mathbf{V}}$	Piecewise continuous approximation of $\mathbf{V}$.
$\tilde{\mathbf{V}}$	Nodal value of velocity vector.
ϕ_i	Velocity interpolating function.
$\hat{p}$	Piecewise continuous approximation of p.
$\tilde{p}$	Nodal value of pressure.
ψ_i	Pressure interpolating function.
n_v	number of velocity nodes.
n_p	number of pressure nodes.
$\mathbf{w}_i$	FE weighting functions.
$\mathscr{L}$	Stokes operator.
Ω	Problem domain.
$\partial\Omega$	Boundary of problem domain.
B	General function defined on Ω.
$\mathbf{F}$	General vector field defined on Ω.
$\mathbf{n}$	Outward pointing unit normal vector on $\partial\Omega$.
$\mathbf{g}$	Right hand side vector.
$\mathbf{g}$	Vector of unknown nodal values.
$\tilde{\mathbf{u}}$	Vector of unknown nodal values of horizontal velocity.
$\tilde{\mathbf{v}}$	Vector of unknown nodal values of vertical velocity.
$\tilde{\mathbf{p}}$	Vector of unknown nodal values of pressure.
θ_n	Error indicator at node n.
h_e	length scale of element e.
A_{min}	Minimum allowable element area.
ϵ	Maximum allowable percent relative error.
m	Maximum allowable number of elements which do not meet the error or area criteria.
L_x	Length of domain in x direction.
L_y	Length of domain in y direction.
u	Horizontal component of velocity.
v	Vertical component of velocity.
k	Inlet velocity scaling coefficient.
(x_o, y_o)	Point at which pressure is specified.
T_r	Reference triangle in dimensionless coordinates.
T	Actual triangle in global cartesian coordinates.
(ξ, η)	Dimensionless (natural coordinates).
(ξ_p, η_p)	Qquadriture point in dimensionless coordinates.
w_p	Gaussian quadriture weights.
A_T	Area of triangle T.

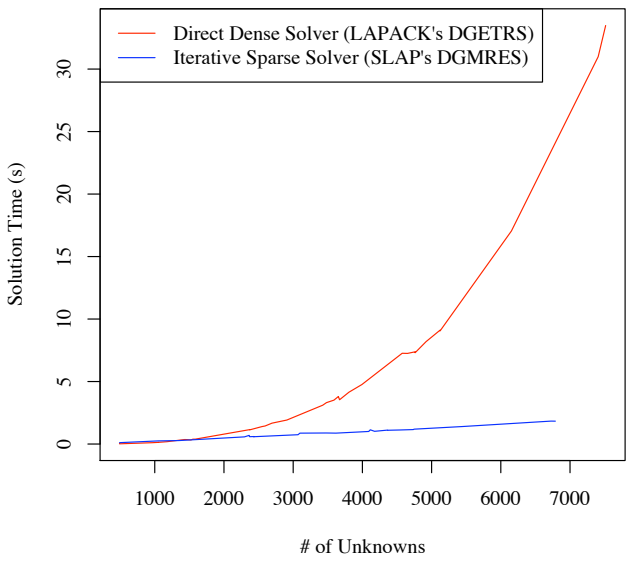


Figure 13. Solution times of direct versus sparse method

Acknowledgments. This template was typeset in pdfLaTeX and was modified from the AGU template available at http://www.agu.org/pubs/helpdesk/index.html

References

Anderson, E., Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. D. Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen (1999), *LAPACK Users' Guide, Third Edition*, SIAM.

Apple (2006), Taking advantage of the accelerate framework, *Tech. rep.*, Apple Incorporated.

Bangerth, W., and R. Rannacher (2003), *Adaptive Finite Element Methods for Differential Equations*, Birkhauser.

Burkardt, J. (2005a), Finite element treatment of the navier stokes equations: Part i, *Tech. rep.*, Florida State University.

Burkardt, J. (2005b), Finite element treatment of the navier stokes equations: Part ii, *Tech. rep.*, Florida State University.

Burkardt, J. (2005c), Finite element treatment of the navier stokes equations: Part iii, *Tech. rep.*, Florida State University.

Burkardt, J. (2007), FREE_FEM_NAVIER_STOKES steady incompressible navier stokes equations in 2d finite element solution banded storage, *Tech. rep.*, Virginia Tech.

Chung, T. J. (2002), *Computational Fluid Dynamics*, Cambridge University Press.

Clay Mathematics Institute (2000), Millennium problems, *Tech. rep.*, Clay Mathematics Institute.

Courant, R., K. O. Friedrichs, and H. Lewy (1928), Uber die partiellen differenz-gleichungen der mathematisschen physik, *Mathematische Annalen*, *100*, 32–74.

Lapidus, L., and G. E. Pinder (1982), *Numerical Solution of Partial Differential Equations in Science and Engineering*, John Wiley and Sons.

Marsden, J. E., and A. J. Tromba (2003), *Vector Calculus*, W.H. Freeman and Company.

Seager, M. (1988), A SLAP for the masses, *Tech. Rep. UCRL-100267*, Lawrence Livermore National Laboratory.

Shewchuk, J. R. (1996), Triangle: Engineering a 2d quality mesh generator and delaunay triangulator, *Applied Computational Geometry: Towards Geometric Engineering*, *1148*, 203–222.

Turner, M. J., R. Clough, H. C. Martin, and L. Topp (1956), Stiffnes and deflection analysis of complex structures, *Aerinautical Science Journal*, *23*, 805–23.

Watson, D. F. (1981), Computing the n-dimensional delaunay tesselation with applications to voronoi polytopes, *Journal of Computation*, *24*, 167–72.

White, F. M. (2006), *Fluid Mechanics 6th Edition*, McGraw Hill.

Willis, R. (1987), *Groundwater Systems*, Prentice-Hall.

C. W. Bracken, Environmental Resources Engineering, Humboldt State University, Arcata, CA, USA. (cwb12@humboldt.edu)

Appendix A: Refinement plots

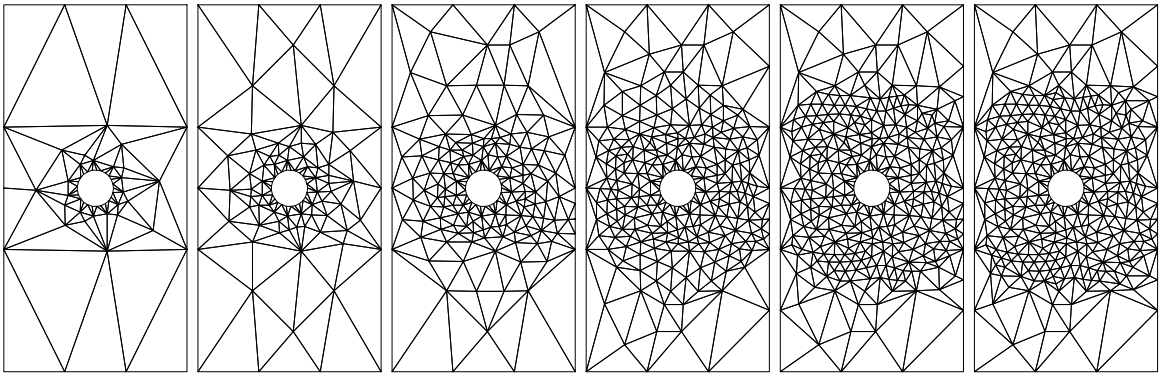


Figure 14. Sample Refinement process using horizonal component of velocity as the error indicator.

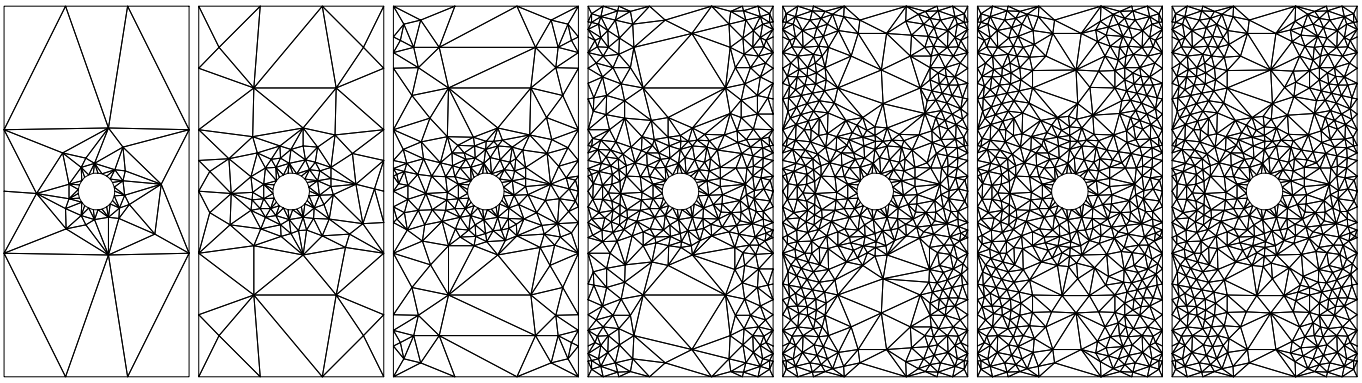


Figure 15. Sample Refinement process with one cylinder using vertical component of velocity as the error indicator.

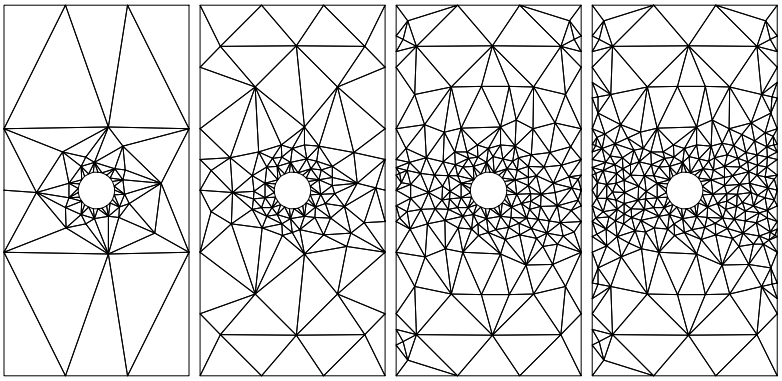


Figure 16. Sample Refinement process using pressure as the error indicator.

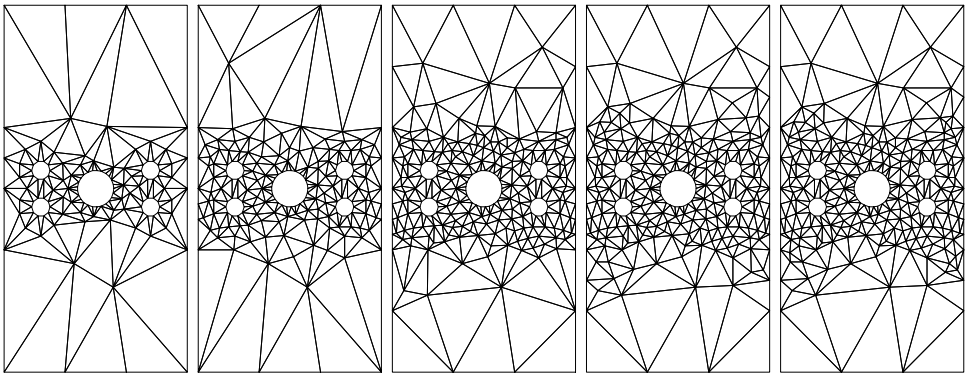


Figure 17. Sample Refinement process with 5 obstructions using horizonal component of velocity as the error indicator.

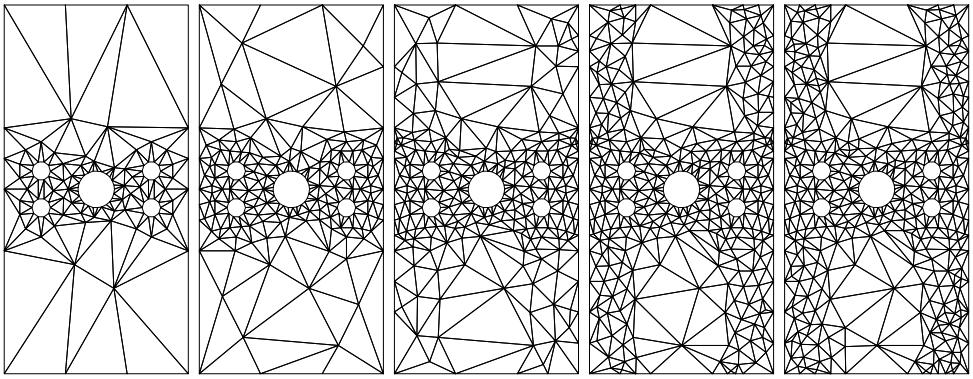


Figure 18. Sample Refinement process with 5 obstructions using vertical component of velocity as the error indicator.

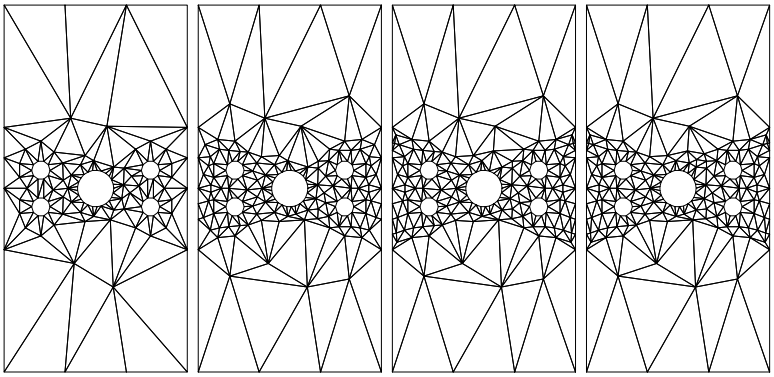


Figure 19. Sample Refinement process with 5 obstructions using pressure as the error indicator.

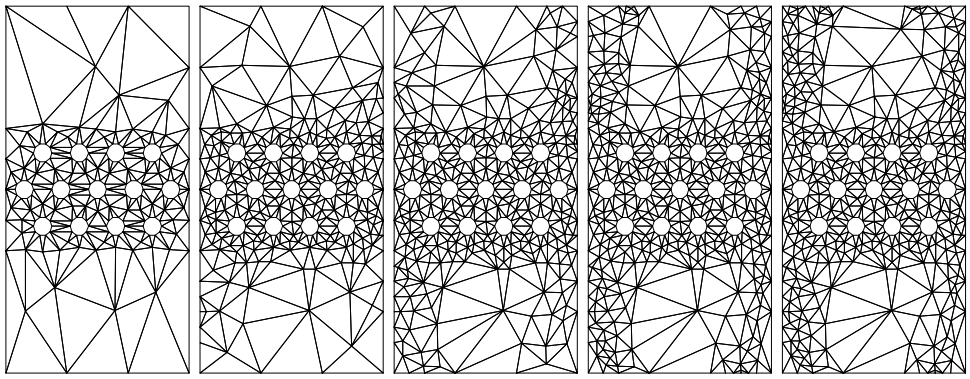


Figure 20. Sample Refinement process with 13 obstructions using vertical component of velocity as the error indicator.

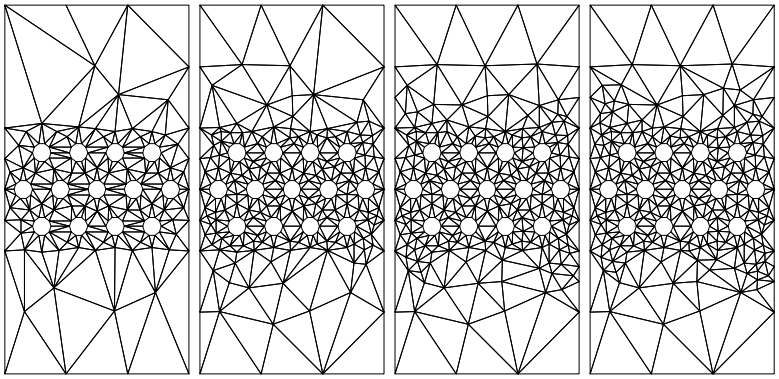


Figure 21. Sample Refinement process with 13 obstructions using horizonal component of velocity as the error indicator.

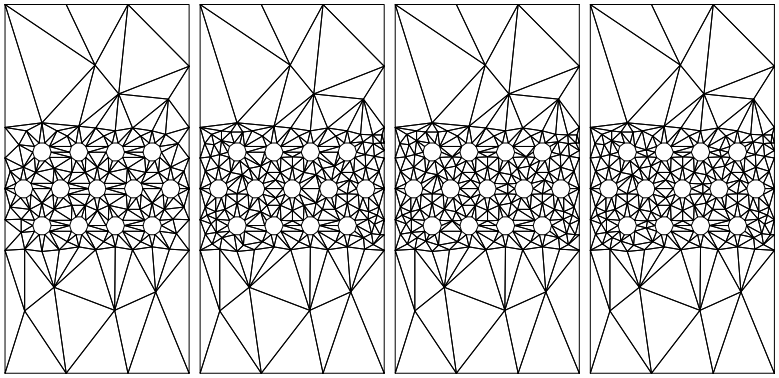


Figure 22. Sample Refinement process with 13 obstructions using pressure as the error indicator.

Appendix B: Stokes Coefficient Matrix Assembly Routine Source Code

This routine was modified from John Burkardt's original routine [*Burkardt,* 2005a].

```
subroutine assemble_jb(A,f,node_xy,element_node,node_num,element_num,&
                       quad_num,variable_num,nu,node_u_variable,&
            node_v_variable,node_p_variable)

            f = 0d0
                A = 0d0

                !  Get the quadrature weights and nodes.
  call quad_rule (quad_num,quad_w,quad_xy)

 do element=1,element_num

                !  Extract the nodes of the linear and quadratic triangles.
    t3(1:2,1:3) = node_xy(1:2,element_node(1:3,element))
    t6(1:2,1:6) = node_xy(1:2,element_node(1:6,element))


        !  Map the quadrature points QUAD_XY to points XY in the physical triangle.
    call reference_to_physical_t6(t6,quad_num,quad_xy,xy)
    area = abs(triangle_area_2d(t3))
    w(1:quad_num) = area*quad_w(1:quad_num);

        !  Consider the QUAD-th quadrature point.

    do quad = 1,quad_num

      point(:,1) = xy(1:2,quad)

                !Evaluate the test functions.
      call basis_mn_t6(t6,1,point,bi,dbidx,dbidy)
      call basis_mn_t3(t3,1,point,qi,dqidx,dqidy)

      do test = 1 , 6

        test_node = element_node(test,element);

        iu = node_u_variable(test_node);
        iv = node_v_variable(test_node);
        ip = node_p_variable(test_node);

                !  Consider the basis functions.
        call basis_mn_t6(t6,1,point,bj,dbjdx,dbjdy)
                call basis_mn_t3(t3,1,point,qj,dqjdx,dqjdy)

        do basis = 1,6

          basis_node = element_node(basis,element);

          ju = node_u_variable(basis_node);
          jv = node_v_variable(basis_node);
          jp = node_p_variable(basis_node);

                        !  Add terms to the horizonal momentum equation.
          a(iu,ju) = a(iu,ju) + w(quad)*nu*(dbidx(test,1)*dbjdx(basis,1)+&
                                            dbidy(test,1)*dbjdy(basis,1))

          if (jp>0)then !we are at a corner node
            a(iu,jp) = a(iu,jp) + w(quad)*bi(test,1)*dqjdx(basis,1)
          end if
```

```
                    !  Add terms to the vertical momentum equation.
          a(iv,jv) = a(iv,jv) + w(quad)*nu*(dbidx(test,1)*dbjdx(basis,1)+&
                                            dbidy(test,1)*dbjdy(basis,1))

          if(jp>0)then !we are at a corner node
            a(iv,jp) = a(iv,jp) + w(quad)*bi(test,1)*dqjdy(basis,1)
          end if

                    !  Add terms to the continuity equation.
          if(ip>0)then !we are at a corner node
            a(ip,ju) = a(ip,ju) + w(quad)*qi(test,1)*dbjdx(basis,1)
            a(ip,jv) = a(ip,jv) + w(quad)*qi(test,1)*dbjdy(basis,1)
          end if

        end do

      end do

    end do

  end do

return
end subroutine
```