Dense Matrices for Biofluids Applications

by

Liwei Chen

A Project Report

Submitted to the Faculty

of the

WORCESTER POLYTECHNIC INSTITUTE

In partial fulfillment of the requirements for the

Degree of Master of Science

in

Applied Mathematics

by

April 2014

APPROVED:

_____

Professor Sarah D. Olson, Project Advisor

_____

Professor Luca Capogna, Head of Department

Abstract

In this report, we focus on Biofluids problems, specifically the Stokes Equation. The method of regularized Stokeslets can be derived from boundary integral equations derived from the Lorentz reciprocal identity. When body forces are known, this is a direct numerical approximation of an integral, resulting in a summation to determine the fluid velocity. In certain cases, which this report is focused on, we know the velocity and want to determine the forces on a structure immersed in a fluid. This results in a linear system $A\mathbf{f} = \mathbf{u}$, where A is a square dense matrix. We study different methods to solve this system of equations to determine the force $\mathbf{f}$ on the structure. For solving a linear system with a dense coefficient matrix, the backslash command in *MATLAB* can be used. This will use an efficient and robust direct method for solving a smaller matrix, but this is not an efficient method for a large, dense coefficient matrix. For a large, dense coefficient matrix, we will explore other direct methods as well as several iterative methods to determine computation time and error on a test case with an exact solution. For direct methods, we will study backslash, LU factorization and QR factorization methods. For iterative methods, we stuied Jacobi, Gauss-Seidel, SOR, GMRES, CG, CGS, BICGSTAB and Schulz CG methods for these biofluids applications. All of these methods have different requirements. For our coefficient matrix A, we identified specific properties and then used proper methods, both direct and iterative. Result showed that iterative methods are more efficient then direct method for large size A. Schulz CG was slower but had a smaller error for the test case where there was an exact solution.

List of Figures

List of Tables

1 Introduction

Partial differential equations have become a useful tool in the fields of industry and biology. The boundary element method is also a popular tool for the solution of equations in electromagnetism and fluids applications. By discretizing only the surface of a radiating or moving object to obtain a linear system, we could get a smaller size system than the finite element and finite difference methods. Usually there are large number of unknowns in biofluids applications, leading to dense coefficient matrices A when solving the linear system $Ax = b$. We need to find good (fast, computationally efficient, low error, reliable and robust) numerical methods for these different applications.

In our report, we focus on Biofluids problems, specifically we are solving the Stokes Equation. In this formulation, we often have a linear system $Ax = b$ where A is normally a square, dense matrix. For solving a linear system with a dense coefficient matrix, the method we used in the past is the backslash command in *MATLAB*. The backslash command is an efficient and robust direct method for solving small, dense matrices. For a large dense coefficient matrix, it might be more computationally efficient to iterative methods. Also combined with a proper preconditioner, we could potentially reduce the operation time even more.

For direct methods, we will talk about backslash, LU factorization and QR factorization methods. For iterative methods, we will discuss Jacobi, Gauss-Seidel, SOR, GMRES, CG, CGS, BICGSTAB and Schulz CG methods. All of these methods have different algorithms and requirements. For our coefficient matrix A, we identified the matrix properties and used proper methods, both direct and iterative.

1.1 Biofluids - Stokes Equations

The biofluids applications used here are for solving two or three dimensional Stokes equations. Stokes equation are well know in fluids problems, especially in biology and industry. When the Stokes equation are steady, for fluid viscosity μ, pressure p, velocity $\mathbf{u}$ and the force $\mathbf{F}$, then the equations are

$$\mu\Delta\mathbf{u} = \nabla p - \mathbf{F}, \tag{1}$$

$$\nabla \cdot \mathbf{u} = \mathbf{0}. \tag{2}$$

Eq. (1) is conservation of momentum and Eq. (2) is incompressibility or mass conservation.

Then in accordance with Cortez's paper [6], we first consider the generic situation in which the forces are spread over a small ball with center point $\mathbf{x}_0$. If ϕ_δ is a radially symmetric smooth function with integral equal to one, then the force is given by

$$\mathbf{F}(\mathbf{x}) = \mathbf{f_0}\phi_\delta(\mathbf{x} - \mathbf{x_0}),$$

where $\mathbf{x}$ can be any point in the fluid domain and δ is the regularization parameter controlling radius of the ball the force $\mathbf{f_o}$ is spread around the point $\mathbf{x_o}$. Point forces are regularized in order for known fundamental solutions such as a Stokeslet to be used when determining velocity $\mathbf{u}$ [6, 8]. Then we denote $G_\delta(\mathbf{x})$ as the solution of $\Delta G_\delta(\mathbf{x}) = \phi_\delta(\mathbf{x})$ in infinite space and let B_δ be the solution of $\Delta B_\delta(\mathbf{x}) = G_\delta(\mathbf{x})$ in infinite space.

Then, taking divergence of Eq. (1) and combining with Eq. (2) we get that $\Delta p = \nabla \cdot \mathbf{F}$, with the particular solution $p = \mathbf{f_0} \cdot \nabla G_\delta$. Now with this expression we rewrite Eq. (1) as $\mu\Delta\mathbf{u} = (\mathbf{f_0} \cdot \nabla)\nabla\mathbf{G}_\delta - \mathbf{f_0}\phi_\delta$, which has particular solution

$$\mu \mathbf{u}(\mathbf{x}) = (\mathbf{f_0} \cdot \nabla)\nabla \mathbf{B}_\delta(\mathbf{x} - \mathbf{x_0}) - \mathbf{f_0}\mathbf{G}_\delta(\mathbf{x} - \mathbf{x_0}).$$

The equation above is referred to as a regularized Stokeslet velocity [6, 8]. If there are N forces $\mathbf{f_k}$ at points $\mathbf{x_k}$, then the pressure and velocity are given by

$$p(\mathbf{x}) = \sum_{\mathbf{k=1}}^{\mathbf{N}} \mathbf{f_k} \cdot \nabla \mathbf{G}_\delta(\mathbf{x} - \mathbf{x_k}),$$

$$\mathbf{u}(\mathbf{x}) = \mathbf{U_0} + \tfrac{\mathbf{1}}{\mu} \sum_{\mathbf{k=1}}^{\mathbf{N}} \{(\mathbf{f_k} \cdot \nabla)\nabla \mathbf{B}_\delta(\mathbf{x} - \mathbf{x_k}) - \mathbf{f_k}\mathbf{G}_\delta(\mathbf{x} - \mathbf{x_k})\}.$$

Finally by choosing a specific blob in two dimension $\phi_\delta(\mathbf{x}) = \dfrac{3\delta^3}{2\pi(|\mathbf{x}|^2 + \delta^2)^{5/2}}$, we get the equation that,

$$p(\mathbf{x}) = \sum_{k=1}^{N} \frac{[\mathbf{f}_k \cdot (\mathbf{x} - \mathbf{x}_k)]}{2\pi r_k^2},$$

$$\mathbf{u}(\mathbf{x}) = \sum_{k=1}^{N} \frac{-\mathbf{f}_k}{4\pi\mu} \ln(r_k) + [\mathbf{f}_k \cdot (\mathbf{x} - \mathbf{x}_k)] \frac{(\mathbf{x} - \mathbf{x}_k)}{4\pi\mu r_k^2}. \tag{3}$$

Where $r_k = \|\mathbf{x} - \mathbf{x}_k\|_2$ and $\|\cdot\|$ denotes the Euclidean norm.

1.1.1 Biofluids inverse problem

Now we need to use Eq. (3) to find the forces from velocities. We can rewrite the equation as:

$$\mathbf{u}(\mathbf{x}_i) = \sum_{j=1}^{N} M_{ij}(\mathbf{x}_1, ..., \mathbf{x}_N)\mathbf{f}_j, \tag{4}$$

or as a matrix equation, $U = MF$. Here, N is the number of points that we know the velocity at and we want to determine the force at each of these points. In two dimensions, M is a $2N \times 2N$ matrix, U and F are $2N \times 1$ vectors. In three dimensions, M is a $3N \times 3N$ matrix, U and F are $3N \times 1$ vectors.

Now, the problem becomes how to optimally solve the matrix equation $b = AX$, where A is M, U is b and x is the unknown F. We set up column vector x (F) by locating the first component of the N points in the top N rows, then the

second component of these points in the next N rows. (If in 3D, then the third component is in the bottom N rows.) The column vector b (U) is similar.

1.2 Linear Algebra Review

First, we will review some definitions and propositions from linear algebra that we will need to describe the matrix M in our biofluids application [1]. A real matrix A is a rectangular array $(a_{i,j})$, with $1 \leq i \leq n, 1 \leq j \leq p$, where $a_{i,j} \in \mathbb{R}$ is the entry in row i and column j, i.e.,

$$A = \begin{pmatrix} a_{1,1} & \cdots & a_{1,p} \\ \vdots & & \vdots \\ a_{n,1} & \cdots & a_{n,p} \end{pmatrix}$$

Here, $\mathbb{R}$ is the field of real numbers. The set of all matrices of n rows and p columns (or we can say size $n \times p$) is denoted by $M_{n,p}(\mathbb{R})$.

In many applications, it is necessary to determine the inverse of a matrix. A matrix $A \in M_{n,n}(\mathbb{R})$ is said to be invertible (or nonsingular), if there exists a matrix $B \in M_{n,n}(\mathbb{R})$ such that $AB = BA = I_n$, where I_n is the identity matrix with dimension n. This matrix B is denoted by A^{-1} and is called the inverse matrix of A. A noninvertible matrix is said to be singular.

Other important properties of a matrix include the kernel and image. The kernel, or null space, of a matrix $A \in M_{n,p}(\mathbb{R})$ is the set of vectors $x \in \mathbb{R}^p$ such that $Ax = 0$; it is denoted by $Ker(A)$. The image, or range, of A is the set of vectors $x \in \mathbb{R}^n$ such that $y = Ax$, with $x \in \mathbb{R}^p$; it is denoted by $Im(A)$. The dimension of the linear space $Im(A)$ is called the rank of A; it is denoted by $rk(A)$.

There are several properties that guarantee the existence of the inverse matrix.

For any A $\in$ M$_{n,n}$ $(\mathbb{R})$ the following statements are equivalent:

1. A is invertible;
2. $Ker(A) = 0$;
3. $Im(A) = \mathbb{R}^n$;
4. there exists $B \in M_{n,n}(\mathbb{R})$ such that $AB = I_n$;
5. there exists $B \in M_{n,n}(\mathbb{R})$ such that $BA = I_n$.

In the last two cases, the matrix B is precisely equal to the inverse of A, A^{-1}.

The determinant is also an important property while checking the singularity of a matrix. The following are properties of the determinant. Let A and B be two square matrices in $M_{n,n}(\mathbb{R})$. Then,

1. det (AB) = (det$A$)(det$B$) = det (BA);
2. det (A^T) = det (A);
3. A is invertible if and only if det(A)$\neq$ 0.

The condition number of a matrix A is a criterion of change on the output of the linear system $Ax = b$, when changing the matrix A. If the condition number is large, when we apply a small perturbation to b, the output x will change a lot. In this case, we call the matrix ill-conditioned. If the condition number is small, when we apply a small perturbation to b, the output x will change a little as well. In this case, we call the matrix well-conditioned. The condition number of the matrix A is calculated as $cond(A) = \|A\| \, \|A^{-1}\|$. Here we need the norm of a matrix to get the condition number. There are several types of matrix norms including 1-norm, 2-norm and ∞-norm. The 1-norm is the maximum of the sums of each column. The 2-norm is the square root of the maximum eigen-value of the

matrix A^*A, where A^* is the conjugate transpose of the matrix A. The ∞-norm is the maximum of the sums of the absolute value for all the entries in each row. In the *MATLAB* code we use $cond(A, 1)$, $cond(A, 2)$, $cond(A, inf)$ [13, 11].

A complex matrix U is a unitary matrix if it satisfies $U^*U = UU^* = I_n$, where U^* is the conjugate transpose of U. For an orthogonal and real matrix Q, the transpose of Q is the inverse of Q, $Q^T = Q^{-1}$.

1.2.1 Properties of matrix for biofluids application

We explore a classic test case of flow past a cylinder. Specifically, we have a 2D fluid with a cylinder of radius a moving at a constant speed of $(1, 0)$. That is, it is moving to the right and not moving up/down. As detailed in Example 3.1 in [6], this test case has an exact solution. In terms of $Ax = b$, we will have b as the velocity such that:

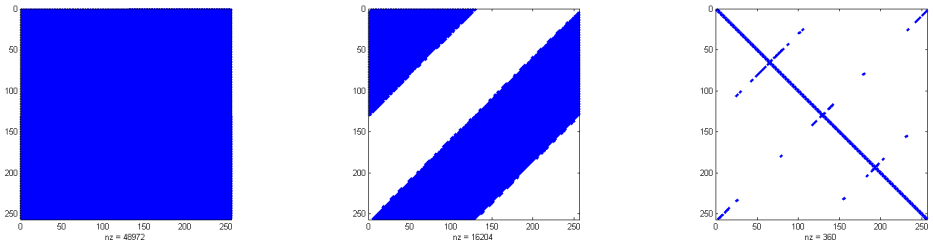
$$b = \begin{bmatrix} 1 & 1 & \cdots & 1 & 0 & 0 & \cdots & 0 \end{bmatrix}^T$$

where there are N 1's corresponding to the x- component of the velocity at each of the N points and N 0's corresponding to the y-component of the velocity at each of the N points. We show the properties in Table 1. Note that T denotes transpose and b is a $2N \times 1$ vector. We want to solve for the forces at each of the points on the cylinder, corresponding to x. The corresponding coefficient matrix will be A and determined via Eq. (4).

In Table 1, we look at several properties of the coefficient matrix for the regularization parameter $\delta = 0.5 \cdot 2\pi a/N$ where $a = 0.25$ and $N = 128$. This test case results in a 95% dense coefficient matrix. More particular, it is a nonsymmetric, positive definite dense matrix.

Table 1: Properties of coefficient matrix A.

Property	A
Density	95% , 360 zeros
Symmetric	NO
Positive Definite	Yes
Sum of each row and column	around 19
Maximum entry	0.484912861970437
Minimum entry	-0.038421931215028
Eigenvalues	not same, all positive
Condition number	$1.2937 * 10^3$
Positive entries	74.73%
Negative entries	24.73%



(a) positive entries (b) negative entries (c) zero entries

Figure 1: For original matrix A, the position of positive, negative and zero entries

For Figure 1 (a) the positive entries are placed throughout the matrix. There are two large bands where negative entries occur, shown in Figure 1 (b). In Figure 1 (c), we can see some zero entries, mainly on the diagonal.

2 Modeling Frameworks

2.1 Dense Matrix

In Biofluids applications, one must solve a system of equations with a dense coefficient matrix. Previously, people have used backslash in *MATLAB*. The backslash command is a direct method. When we solve a matrix system $Ax = b$ with the unknown x, we could use the backslash method, $x = A\backslash b$ in *MATLAB*.

In *MATLAB*, the "\" command first checks the properties of A [12]. According to the explanation of algorithm for full inputs of the "\" command on MathWorks' website [12] and outlined in [9], we have the following Table 2. In Table 2, we will show that the method *MATLAB* uses different solvers based on the properties of the matrix A. For example, if symmetric and real positive diagonal elements, Cholesky will be used to solve $Ax = b$.

Table 2: Backslash Command in *MATLAB*

Property	Solver employed
sparse and banded	banded solver
upper or lower triangular	backward substitution
symmetric and real positive diagonal elements	Cholesky
none of criteria above is fulfilled	Gaussian elimination, partial pivoting
sparse	UMFPACK library
not square	QR factorization for undetermined systems

2.2 Direct Methods

2.2.1 LU factorization

Let A be a matrix of order n where all diagonal submatrices of order k are nonsingular. There exists a unique pair of matrices (L, U), with U upper triangular and L lower triangular with a unit diagonal (i.e. diagonal entries equal one),

such that $A = LU$ [1]. The condition stipulated by the theorem is often satisfied in practice. For example, it holds true if A is positive definite. Note that the converse is not true: namely, a matrix A, such that all its diagonal submatrices are nonsingular is not necessarily positive definite.

In linear algebra, a symmetric $n \times n$ real matrix M is said to be positive definite if $z^T M z$ is positive for every non-zero column vector z of n real numbers. Here z^T denotes the transpose of z. Let M be an $n \times n$ matrix. Then all its eigenvalues are positive and its leading principal minors are all positive if M is positive definite.

2.2.2 QR factorization

QR factorization is a method for solving a least squares problem $Ax = b$ by factorizing $A = QR$. Here R is a triangular matrix and Q is an orthogonal (unitary) matrix [16]. There are three operations while doing QR factorization: Gram-Schmidt, Householder reflections and Givens rotations. While expensive for dense problems, QR is quite competitive for large, sparse problems.

2.3 Iterative Methods

Direct methods are not always efficient for solving a large system of equations with a dense coefficient matrix. This is due to the increasing operation count and memory requirements. For such problems, iterative methods, such as GMRES and BiCG can be better choices. An overview of solving these systems of equations using iterative methods is described in [1].

2.3.1 Jacobi, Gauss-Seidel, SOR

Jacobi method, Gauss-Seidel method and Successive Overrelaxation (SOR) method can all be used for solving $Ax = b$ [10]. All three of these methods are

iterative methods. For a a nonsingular matrix A, we can split A into two matrices, $A = M - N$. Here we need M to be easily invertible. Plugging in for A, we get $Mx = Nx + b$. An iterative method is obtained by having an initial guess x_0 in $\mathbb{R}$ and using the following update for x_{k+1}: $Mx_{k+1} = Nx_k + b$. Then for a given x_0 in $\mathbb{R}$, we have that

$$x_{k+1} = M^{-1}Nx_k + M^{-1}b.$$

For an iterative method, we want x_k to converge. We need the spectral radius (i.e. the maximum of eigenvalues) of $M^{-1}N$ to be less than one. That is because we define error e_k as $e_k = x_k - x$ where x is the true solution. Then:

$$\begin{aligned} e_k &= x_k - x \\ &= (M^{-1}Nx_{k-1} + M^{-1}b) - (M^{-1}Nx + M^{-1}b) \\ &= M^{-1}N(x_{k-1} - x) \\ &= M^{-1}Ne_{k-1}. \end{aligned}$$

Thus, we need the spectral radius of $M^{-1}N$ less than one to get the error to converge.

There are different criterion to determine whether an iterative method is converging. One is checking the norm of $x_{k+1} - x_k$, the norm of the iteration value. The other one is a relative criterion, meaning that

$$\frac{\|b - Ax_k\|}{\|b - Ax_0\|} \leq \varepsilon.$$

We prefer the relative criterion, though sometimes the iteration value may converge slowly. This does not necessarlly mean that x_n is close enough to our exact value. However, for the same problem $Ax = b$, checking the relative criterion may cost more time than checking iteration values. That is because each time we

calculate the relative criterion, even though we can compute $\|b - Ax_0\|$ once and save for later calculations, the $\|b - Ax_k\|$ must be computed for each iteration.

Specifically, the Jacobi method splits $A = M - N$, where $M = D$, D is diagonal matrix and $N = D - A$. The Gauss-Seidel Method (the code is shown in Appendix 5.2 and 5.3) splits A as $A = D - E - F$, where D is a diagonal matrix, $-E$ is the lower triangular part (without diagonal) of A and $-F$ is the upper triangular part (without diagonal) of A. Then $M = D - E$, $N = F$, so $M^{-1}N = (D - E)^{-1}F$. Here $(D - E)$ is easy to invert since it is a triangular matrix. SOR method (the code is shown in Appendix 5.4 and 5.5) splits $A = M - N$, where $M = \frac{D}{\omega} - E$, $N = \frac{1-\omega}{\omega}D + F$. The relaxation method can converge only if $0 < \omega < 2$. When ω=1, SOR is the same as Gauss-Seidel.

2.3.2 GMRES

GMRES [14] is short for Generalized minimal residual method, which is designed for solving nonsymmetric linear systems. While solving a linear system $Ax = b$ with $A_{n\times n}$ an invertible matrix, the order m Krylov space is $K_m = span\{b, Ab, A^2b, ..., A^{m-1}b\}$. Then the GMRES method approximates the exact solution by minimizing the residual $Ax_m - b$, where $x_m \in K_m$.

However, vectors b, Ab, ..., $A^{m-1}b$ are almost linearly dependent. Then we can use the Arnoldi method to get the standard orthogonal basis q_m for K_m. Now we could rewrite x_m as $x_m = x_0 + Q_n y_m$, where $y_m \in R^m$ and Q_m stands for the $m \times n$ matrix formed by q_m. The Arnoldi process will produce an $(m + 1)$ by m upper Hessenberg matrix (square matrix with zero entries below the first subdiagonal), denoted by $\widetilde{H}_m$. Then we have $AQ_m = Q_{m+1}\widetilde{H_m}$. Since Q_m is orthogonal, we can obtain that $\|Ax_m - b\| = \|H_m y_m - \beta e_1\|$ with $e_1 = (1, 0, ..., 0) \in R^m$ and $\beta = \|b - Ax_0\|$. Usually we take x_0 zero as our first trial initial vector. Next we find

y_m by minimizing $\|H_m y_m - \beta e_1\|$ with the QR factorization method. Then $x_m = x_0 + Q_n y_m$.

2.3.3 Conjugate Gradient

Here we introduce the Gradient method [1] first. When solving a linear system $Ax = b$, if the coefficient matrix A is positive definite, we could use the gradient method. The main idea of a gradient method is to consider the function

$$f(x) = \frac{1}{2}\langle Ax, x\rangle - \langle b, x\rangle$$

($\langle\cdot,\cdot\rangle$ is the dot product) so that $\min \|b - Ax\|^2 = \min f(x)$, considering the problem of minimizing quadratic functions. For a point x_k , the iteration is $x_{k+1} = x_k + \lambda_k d_k$. The gradient method determines the searching direction, which is the steepest descent direction d_k ($d_k = -\nabla f(x_k)$). The step length λ_k satisfies:

$$f(x_k + \lambda_k d_k) = \min f(x_k + \lambda d_k), \lambda \geq 0.$$

If the coefficient matrix A is symmetric and positive definite (SPD), other search directions can be used. For an SPD A, the Conjugate Gradient method (CG) constructs two conjugate directions with respect to A, making it faster than the Gradient method.

Here we introduce the iteration of CG. Let x_k be a sequence of approximate solutions of the Conjugate Gradient method. The associated residual sequence is $r_k = b - Ax_k$. Then there exists an A-conjugate sequence p_k such that $p_0 = r_0 = b - Ax_0$. Now we get three iterations:

$$x_{k+1} = x_k + \alpha_k p_k$$

$$r_{k+1} = r_k - \alpha_k A p_k$$

$$p_{k+1} = r_{k+1} + \beta_k p_k$$

where $\alpha_k = \frac{\|r_k\|^2}{\langle Ap_k, p_k\rangle}$ and $\beta_k = \frac{\|r_{k+1}\|^2}{\|r_k\|^2}$. CG is an effective method, however, the coefficient matrix A must be SPD. We now introduce several methods which are a modification of CG.

The first modification is the Conjugate Gradient to Normal Equations (CGN) [14]. There are also two CGN methods, CGNE and CGNR. Still solving $Ax = b$.

CGNE: $AA^Ty = b, x = A^Ty$

CGNR: $AA^Tx = \tilde{b}, \tilde{b} = A^Tb$

where the coefficient matrix A is nonsymmetric, possibly indefinite (but nonsingular).

The second one is the Bicongugate Gradient method (BiCG) [15]. Instead of searching mutually conjugate directions, the BiCG method constructs two A-conjugate directions. One is r and $\tilde{r}$, the other is p and $\tilde{p}$, so that $\langle \tilde{r}_k, r_j\rangle = 0$, for $j \neq k$; $\langle \tilde{p}_k, Ap_j\rangle = 0$, for $j \neq k$. This means that we do not need the coefficient matrix A to be symmetric. The method is:

$$p_0 = r_0 = b - Ax_0, \tilde{p}_0 = \tilde{r}_0$$

For k = 0, 1, 2, 3 . . . until convergence do

$$x_{k+1} = x_k + \alpha_k p_k$$

$$r_{k+1} = r_k - \alpha_k Ap_k, \tilde{r}_{k+1} = \tilde{r}_k + \alpha_k A^T\tilde{p}_k$$

$$p_{k+1} = r_{k+1} + \beta_k p_k, \tilde{p}_{k+1} = \tilde{r}_{k+1} + \beta_k\tilde{p}_k$$

End do

where $\alpha_k = \frac{\langle \tilde{r}_k, r_k\rangle}{\langle \tilde{p}_k, Ap_k\rangle}, \beta_k = \frac{\langle \tilde{r}_{k+1}, r_{k+1}\rangle}{\langle \tilde{r}_k, r_k\rangle}$. And the orthogonal projection are $\langle \tilde{r}_k, r_j\rangle = 0$, for $j \neq k$; $\langle \tilde{p}_k, Ap_j\rangle = 0$, for $j \neq k$.

The third method is the Conjugate Gradient Squared method (CGS) [17]. This method comes from the BiCG method. According to the BiCG method's scheme, r_{i+1} has a term Ap_i and p_i has a term r_i. So r_{i+1} has a term Ar_i. Similarly for $\tilde{r}_i$.

Then it is straightforward to show that $r_i = P_i(A)r_0$ and $\tilde{r}_i = P_i(A^T)\tilde{r}_0$, where P_i is the i^{th} polynomial of A, whose largest order is i and the corresponding term is $(A)^i$. Since we could construct $\langle \tilde{r}_k, r_j \rangle = 0$, for $i \neq k$; $\langle \tilde{p}_k, Ap_j \rangle = 0$, for $j \neq k$, now we could represent $\langle r_j, \tilde{r}_k \rangle$ as:

$$\langle r_j, \tilde{r}_k \rangle = \left\langle P_j(A)r_0, P_k(A^T)\tilde{r}_0 \right\rangle = \langle P_k(A)P_j(A)r_0, \tilde{r}_0 \rangle = 0, i < j.$$

Then we could think about the right hand side $\langle P_k(A)P_j(A)r_0, \tilde{r}_0 \rangle = 0, i < j$ to be a squared form. By doing so, we can avoid forming the vector $\tilde{r}$ and multiplication with A^T.

The last method is the BiCGSTAB [17]. This is similar to the former CGS method. However, we consider $\langle P_k(A)P_j(A)r_0, \tilde{r}_0 \rangle = \langle Q_k(A)P_j(A)r_0, \tilde{r}_0 \rangle$. Here Q is also a polynomial of A, but could be written as $Q_i(x) = (1 - \omega_1 x)(1 - \omega_2 x)...(1 - \omega_i x)$, where ω_i are constants. To get the parameters α_i and β_i, since $\alpha_k = \frac{\langle \tilde{r}_k, r_k \rangle}{\langle \tilde{p}_k, Ap_k \rangle}, \beta_k = \frac{\langle \tilde{r}_{k+1}, r_{k+1} \rangle}{\langle \tilde{r}_k, r_k \rangle}$, we need to calculate $\langle r_i, \tilde{r}_i \rangle$. By using this kind of Q, we could reduce the time of solving

$$\langle r_i, \tilde{r}_i \rangle = \left\langle P_i(A)r_0, Q_i(A^T)\tilde{r}_0 \right\rangle = \langle Q_i(A)P_i(A)r_0, \tilde{r}_0 \rangle .$$

Q_i has $i + 1$ terms and so does P_i. For calculating Q_iP_i, we have $(i + 1) \times (i + 1)$ terms. However we can only consider the highest order term of $Q_i(A)$, since the projection of lower order terms equals zero because of the A-conjugate gradient ($\langle \tilde{r}_k, r_j \rangle = 0$, for $j \neq k$; $\langle \tilde{p}_k, Ap_j \rangle = 0$, for $j \neq k$). Then we reduce the number of terms to $i + 1$ only.

2.3.4 Pseudoinverse Method

We know that the inverse matrix is only available for nonsingular square matrices. While for some full rank matrices, though not square, we can obtain the

pseudoinverse for them. If the pseudoinverse of a matrix A is available, it is denoted as $A^{\dagger}$. Then the minimum norm solution of $\min_{x\in\mathbb{R}^n} \|Ax-b\|_2$ is given by $x = A^{\dagger}b$. We use Schulz method to approximate the pseudoinverse [5]. The code is shown in Appendix 5.1.

Here are two methods combined with the Schulz method. The first one is the Richardson's PR2 and the second one is the Conjugate Gradient method. For Richardson's PR2, we obtain the following scheme. Starting from a given x_0 and $r_0 = b - Ax_0$,

$$x_{k+1} = x_k + \lambda_k C_k r_k$$

$$r_{k+1} = r_k - \lambda_k A C_k r_k$$

$$\lambda_k = \frac{(AC_k r_k)^T r_k}{\|AC_k r_k\|_2^2}$$

Here λ_k is the step length and C_k is a coefficient here. The closer C_k is to A^{-1}, the faster this method will converge. Then, we use the Schulz method to find the M_k (pseudoinverse approximation of A), and let $C_k = M_k$.

Moreover, the pseudoinverse method allows one to use the CG method for nonsymmetric coefficient matrix systems. While using conjugate gradient method to solve the linear system $Ax = b$, the matrix A must be SPD. But we could combine Schulz method with the conjugate gradient method, called Schulz CG. Schulz method is a way of finding the pseudoinverse of a full rank matrix. First we set $M_0 = \frac{A^T}{\|A\|_2^2}$. Then for each iteration, update $M_{k+1} = 2M_k - M_k A M_k$. Then we combine M_k with our original problem $Ax = b$, such that $M_k Ax = M_k b$. Here we have $M_k A$ is SPD for any nonnegative k. Then we can use the conjugate gradient method for our new problem $M_k Ax = M_k b$.

2.4 Preconditioners

When solving large size 3-D PDE problems, iterative methods can be better than direct methods. Direct methods generally need more storage and operations than iterative methods. But iterative methods may not have the reliability of direct methods since in some cases, iterative methods do not converge. In this kind of situation, preconditioners can be used. Even though not always sufficient, it could allow the method to converge in a reasonable amount of time.

Preconditioning means that the linear system is transferred to a new one with properties that may be helpful to increase the speed of convergence. In other words, preconditioning would improve the coefficient matrix's spectral properties. For an SPD system, the CG method converges faster when the distribution of the eigenvalues of the coefficient matrix A is better. Here, we want the preconditioned matrix to have a small spectral condition number and (or) the eigenvalues clustered around 1. For a nonsymmetric problem, the situation is different. Similar to the method GMRES, the eigenvalues may not describe the convergence of nonsymmetric matrix iterations. However, we want a clustered spectrum (away form zero) to get rapid convergence [4].

Suppose a matrix M is the preconditioner in our problem. Then the transferred problem to solve is $MAx = Mb$ (left preconditioning) or $AMy = b, x = My$ (right preconditioning). It is obvious that the closer M is to A^{-1}, the better it will be. To construct the matrix M, our preconditioning matrix, we need to determine the nonzero pattern of it. The large entries in A come out to be the same positions in A^{-1}. So when the nonzero pattern of A is the same for M, we try two methods here to get our nonzero pattern. These are mentioned in [2] as heuristic (1) and (4). The code is shown in Appendix 5.6 and 5.7.

For the first one mentioned as heuristic (1), we fix a positive integer k with k

far less than A's dimension n. We then find the k largest entries in each of A's columns. Then the position of those $k * n$ entries are our nonzero pattern. For the second one mentioned as heuristic (4), we find entries whose absolute value is greater than or equal to the parameter ε, with $\varepsilon \in (0, 1)$. Then we keep the position (i, j) of entries in A with $|a_{ij}| > \varepsilon \cdot \max_{1 \leq k,l \leq n} |a_{kl}|$ as our nonzero pattern.

Once the nonzero pattern is chosen, we find each m_j (column vectors of M) by minimizing the Frobenius norm, $\min \|I - AM\|_F^2 \Rightarrow \min \left\| \hat{e}_j - \hat{A}\hat{m}_j \right\|_2$. That is

$$\min \left\| \widehat{e}_j - \widehat{A}\widehat{m}_j \right\|_2,$$

where $\widehat{A}$ is the new A matrix with nonzero columns left with respect to the nonzero rows in m_j and $\widehat{e}_j$ is the the unitary vector. Since it is a least squares problem, we could use an orthogonal factorization method to solve the system.

3 Model Testing and Results

3.1 Case 1: Cylinder with $N = 128$ and $a = 0.25$

Table 3 summarizes different solvers and whether they can be used for this test case based on the properties of the coefficient matrix A. This is a test case with an exact solution for a cylinder with a given radius a moving to the right with velocity $(0, 1)$ as described in Section 1.2.1. Note since this is a 2D problem and $N = 128$, the coefficient matrix A will be $2N \times 2N$. In accordance to the properties of the coefficient matrix A, direct methods LU and QR and iterative methods Gauss Seidel, SOR, CGS, BiCHSTAB, Schulz PR2, Schulz CG and GMRES can be used. In Table 3, ρ denotes the spectral radius.

Table 3: Case 1–Solvers and criteria.

Method / Algorithm	Criteria	**Yes / No based on A matrix**
Cholesky	SPD	NO - NOT symmetric
LU	invertible, principal minor $\neq 0$	YES - invertible, PD
QR	real nonsingular matrix	YES - real nonsingular matrix
Jacobi	$\rho(M^{-1}N) < 1$	NO - $M^{-1}N > 1$
Gauss Seidel	$\rho(M^{-1}N) < 1$	YES - $M^{-1}N < 1$
SOR	$\rho(M^{-1}N) < 1$	YES - $M^{-1}N < 1$
CG	SPD	NO - NOT SPD
CGS	invertible	YES - invertible
BiCGSTAB	invertible	YES - invertible
Schulz PR2	full rank, $\rho(I - AM_0) < 1$	YES - full rank $I - AM_0 < 1$
Schulz CG	full rank, $\rho(I - AM_0) < 1$	YES - full rank $I - AM_0 < 1$
GMRES	any real matrix	YES - real matrix

Table 4 shows the number of iterations for these three direct methods. Cholesky requires the least number of operations of these three, but requires A to be symmetric. LU and QR can also be used, with LU requiring less operations than QR.

Table 5 shows the results for trying different solvers with different stopping

Table 4: Solvers and number of operations for the coefficient matrix $A_{n\times n}$.

Method / Algorithm	Operations
Cholesky	$\frac{n^3}{6}+n$
LU	$\frac{n^3}{3}+\frac{n^2}{2}$
QR	$n^3+\frac{3n^2}{2}$

criteria. When we decrease stopping criteria, as show in Table 5 for Gauss Seidel and Schulz CG, operation time increases and error decreases. By observation, direct methods are an order of magnitude faster (on average), than iterative methods with a similar error.

Table 5: Case 1–Results for different solvers with $N = 128$, $a = 0.25$ and $\delta = 0.5 * a * 2 * \pi/N$.

Method / Algorithm	stopping criteria	operation time	error
backslash	none	0.003917s	2.883578065903e-003
LU	none	0.003803s	2.883578065903e-003
QR	none	0.013098s	2.883578065903e-003
Gause Seidel ite	10^{-2}	0.012448s	4.001010911070e-003
Gause Seidel rel	10^{-2}	0.016991s	8.479259557713e-003
Gause Seidel ite	10^{-4}	0.066209s	2.871422306836e-003
Gause Seidel rel	10^{-4}	0.024783s	2.861511514985e-003
SOR ite $\omega = 0.4$	10^{-2}	0.011180s	3.263184449366e-003
SOR rel	10^{-2}	0.023780s	7.355444394070e-003
SOR ite	10^{-4}	0.019994s	2.884251964290e-003
SOR rel	10^{-4}	0.025819s	2.892100565886e-003
CGS	$4.2 * 10^{-15}$	0.055983s	2.883578065906e-003
BiCGSTAB	$4.9 * 10^{-16}$	0.041117s	2.883578065903e-003
Schulz CG	10^{-6}	0.043843s	2.768892208353e-003
Schulz CG	10^{-9}	0.088875s	2.451242421238e-003
Schulz CG	10^{-12}	0.083423s	2.455553760412e-003
GMRES	10^{-6}	0.081820s	2.883578065903e-003

Table 6 shows the number of iterations of Schulz CG with different stopping criteria. We note that decreasing the tolerance does not require that many more iterations. However, each iteration has a high computational cost.

Table 6: Case 1–Results for Schulz CG.

Method / Algorithm	stopping criteria	number of iterations
Schulz CG	10^{-6}	2
Schulz CG	10^{-9}	3
Schulz CG	10^{-12}	4

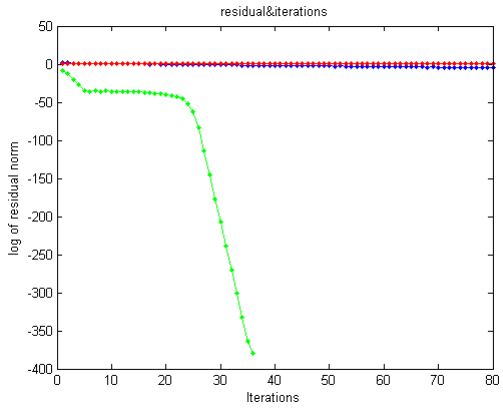


Figure 2: Schulz CG is shown in Green, Gauss seidel in blue, SOR ($\omega = 0.4$) in red. Here, $N = 128$, $a = 0.25$, $\delta = 0.5 * a * 2 * \pi / N$. We set max number of iterations to 80. For the residuals plotted, Schulz CG's residual is $\|M_K b - M_K A x\|$ and the other two are $\|b - Ax\|$.

From Figure 2, we can observe that there is almost no difference between the Gauss seidel method and SOR ($\omega = 0.4$) method for this problem. However, Schulz CG converges faster than Gauss seidel method and SOR ($\omega = 0.4$) method. It actually meets the error criteria or tolerance in a smaller number of iterations for each of the methods. We show for more iterations to see the difference of convergence speed, for each of the methods.

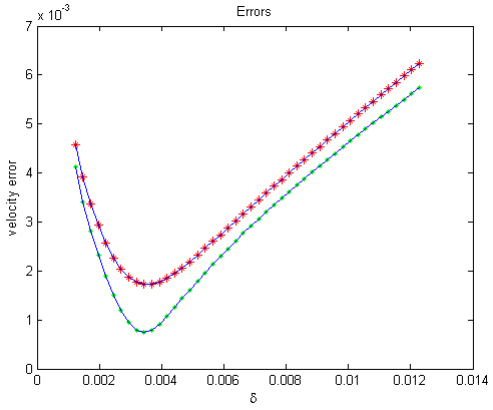
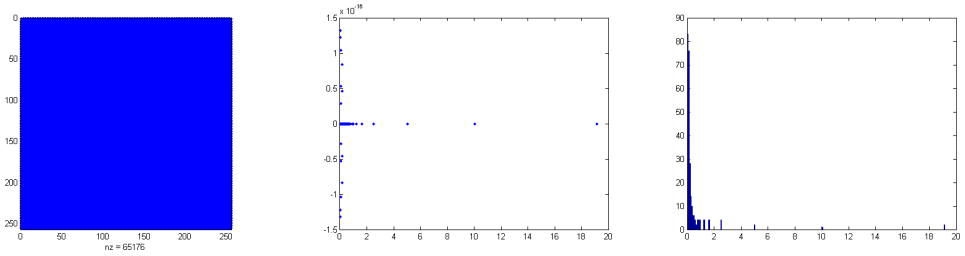


Figure 3: Velocity Error for test case 1. Schulz CG is shown in Green and backslash red. Result shown are for several different regularization parameters δ. The stopping criteria for Schulz CG is 10^{-15}.

From Figure 3, we can observe that the velocity error of the Schulz CG method is smaller than the backslash method. Here, the Schulz CG method is computationally slower for this small problem. However, for larger problems, an iterative solver may win out above backslash because it is a direct method. Note also that there is an optimal regularization parameter δ that gives the smallest velocity error for both Schulz CG and backslash in Figure 3.

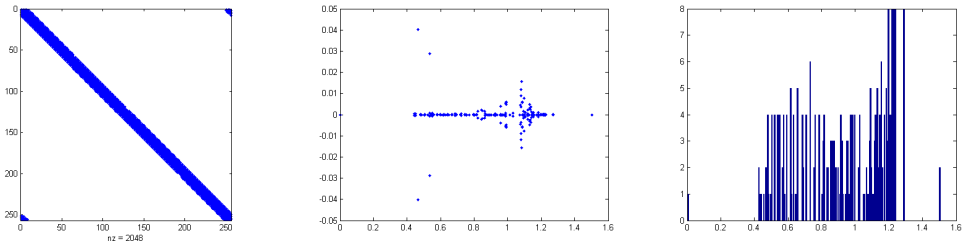
Next, we summarize some results for our problem using preconditioners. We use 'spy(A)' to get the nonzero entries of our original matrix A. Using 'plot(eig(A))' to get the graph of the eigen-value distribution. We also use 'svds(A,256)' to get all the singular values of our matrix and 'hist(svds(A,256),256)' to get our his-

togram of singular values (Note $N = 128$ and $2N = 256$). As mentioned before, for nonsymmetric problems, we want the eigenvalues to have a clustered spectrum (away form zero) to get rapid convergence [4].



(a) position of nonzero entries (b) eigen-value distribution (c) Singular value histogram

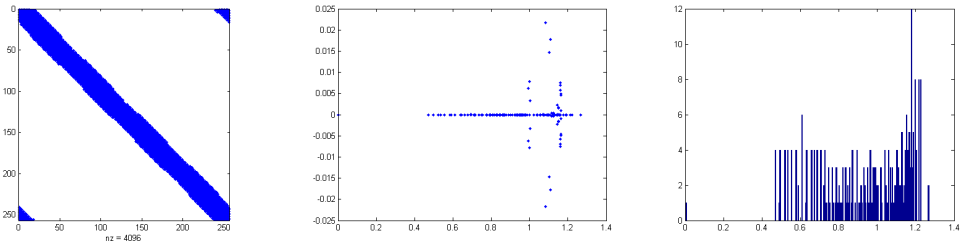
Figure 4: Properties of original matrix A.



(a) position of nonzero entries (b) eigen-value distribution (c) Singular value histogram

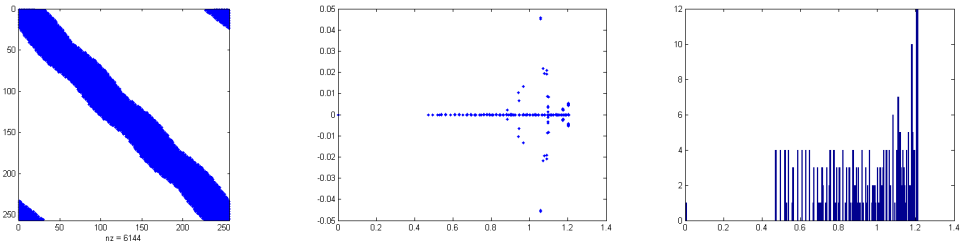
Figure 5: Properties of preconditioner NO.1, $M1_1$ with $k = 8$.

Figure 4 is a graph of nonzero entries (a), eigen-values distribution (b), and singular value histogram (c) for our original matrix A. Note that the eigen-values are mainly clustered between 0 and 2 with a few larger values going out to 20. Figure 5 and 6 are preconditioner NO.1 with parameter k=8 and 16 which made the matrix density 3.13% and 6.25%, respectively. Here, $M1_1$ is preconditioner NO.1 with $k = 8$. For the Figure captions $M1_i$ is preconditioner NO.1 with parameter k_i, where k=[8, 16, 24, 40]. Note that eigen-values are now between 0 and 1.5, 0 and 1.3 for k=8 and k=16, respectively in Figure 5, 6. Comparing Figure 5, 6, 7, 8 and 9 , we can see that when $k = 32$ in Figure 8, the eigen-values are the



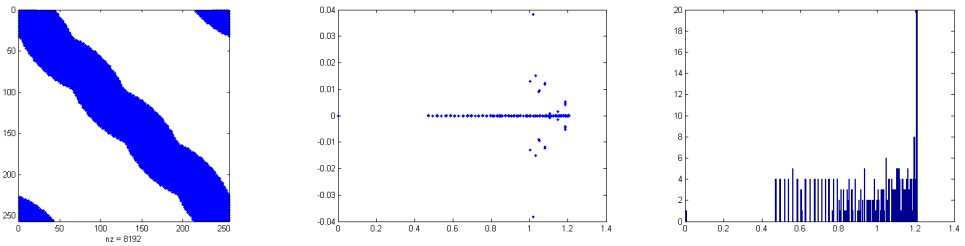
(a) position of nonzero entries (b) eigen-value distribution (c) Singular value histogram

Figure 6: Properties of preconditioner NO.1, $M1_2$ with $k = 16$.



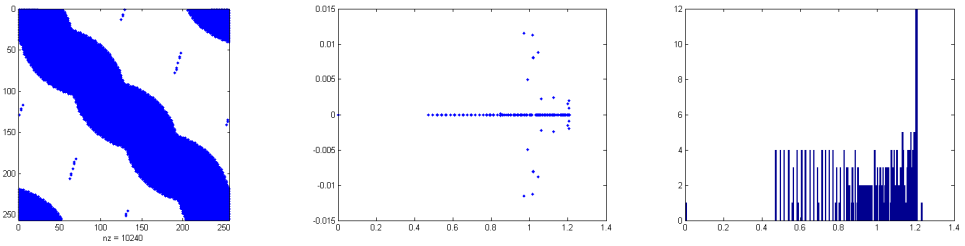
(a) position of nonzero entries (b) eigen-value distribution (c) Singular value histogram

Figure 7: Properties of preconditioner NO.1, $M1_3$ with $k = 24$.



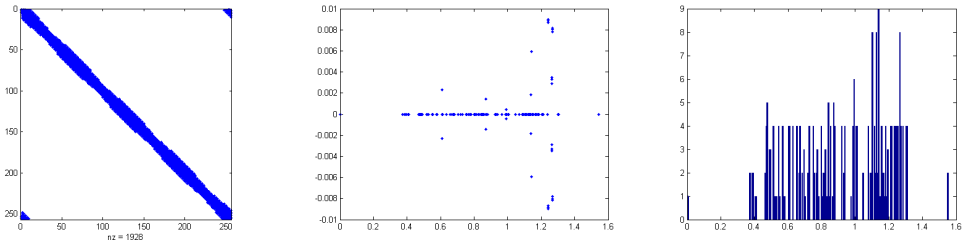
(a) position of nonzero entries (b) eigen-value distribution (c) Singular value histogram

Figure 8: Properties of preconditioner NO.1, $M1_4$ with $k = 32$.



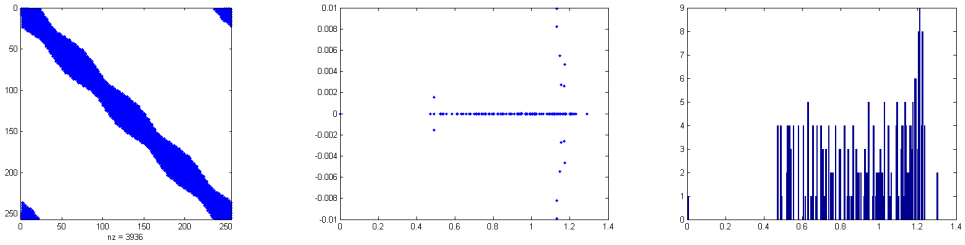
(a) position of nonzero entries (b) eigen-value distribution (c) Singular value histogram

Figure 9: Properties of preconditioner NO.1, $M1_5$ with $k = 40$.



(a) position of nonzero entries (b) eigen-value distribution (c) Singular value histogram

Figure 10: Properties of preconditioner NO.4, $M4_1$ with $tol = 0.6$.



(a) position of nonzero entries (b) eigen-value distribution (c) Singular value histogram

Figure 11: Properties of preconditioner NO.4, $M4_2$ with tol=0.48.

most clustered out of all the different k values used here. Also as k increases in Figure 5-9 (a), the zero entries are always around the diagonal and the width of the band of nonzero entries increases. Figure 10 and 11 are preconditioner NO.4 with parameter tol value 0.6 and 0.48, which made the matrix density 2.49% and 6.01%. We choose tol value 0.6 and 0.48 to make sure that the density percentages are similar to $k = 8, 16$ for preconditioner NO.1 for comparison. Figures 9-10 for preconditioner NO.4 shows similar properties to Figure 3-7 with preconditioner NO.1. In Figure Captions 9-10, $M4_i$ corresponding to preconditioner NO.4 with parameter tol_i, where tol=[0.6, 0.48]. Overall, the more nonzero entries in our matrix, the more clustered our eigen-values are, and they are closer to 1.

Table 7 shows that for this test case, the preconditioned GMRES could reduce the density of coefficient matrix A and save operation time for GMRES only. Case

NO.1 and NO.4 are described in Section 2.4 (Preconditioners) and are less dense than A. For example, NO.1 with k=8 has 3.13% density while the original A has 99.45% density.

Table 7: Preconditioner density and operation time.

Preconditioners	density	Operation times for GMRES only (Full solve)
Without, A_c	99.45%	0.081820s (0.081820s)
NO.1 k=8	3.13%	0.045495s (0.635933s)
NO.1 k=16	6.25%	0.005630s (0.700279s)
NO.4 tol=0.6	2.49%	0.003050s (0.588738s)
NO.4 tol=0.48	6.01%	0.003208s (0.849987s)

In accordance to Table 7, we now look at computational times for solving preconditioned problems. In Table 7, we observe that the GMRES solver is faster than GMRES without preconditioning. This means that the more clustered coefficient matrix A is more efficient to solve via GMRES. Also, when a different density of our preconditioner matrix M is used, the elapsed times are different. We could see that NO.1 $k = 8$ cost 10 times more than NO.1 $k = 16$. NO.4 is also faster than NO.1 but $tol = 0.6$ and $tol = 0.48$ have almost the same efficiency. However, we can not consider the elapsed time for GMRES method only, we need to include the time for preconditioning the matrix M and find out our final result by right preconditioning (here we only consider right preconditioning as left preconditioning has a similar setup). We can see that the denser our matrix M is, the more elapsed time cost. When considering the total elapsed time, preconditioned GMRES methods are slower than GMRES method without preconditioner.

The poor result may be caused by minimizing the Frobenius norm to solve for the matrix M. Notice that as mentioned in Alleon and Benzi's paper [2], the sparse A will make the $\widehat{A}$ a matrix with only a few nonzero rows and columns, so that each least square problem has small size and can be solved efficiently. Since

our matrix A is a 95% dense matrix, the time spent on solving the least squares problem can be a lot. The preconditioned system has more clustered eigenvalues, but the time spent on preconditioning makes for a poor result.

3.2 Case 2: Cylinder test case with $N = 1280$ and $a = 2.5$

Similar to Section 3.1, we use a cylinder test case from [6] with a known exact solution. We change N to 1280 and a to 2.5. Thus, our cylinder of larger radius is moving with constant speed.

Table 8 shows the properties of the larger coefficient matrix A. Direct methods LU and QR and iterative methods CGS, BiCGSTAB, Schulz PR2, Schulz CG and GMRES can be used to solve this test case.

Table 8: Case2–Solvers and criteria.

Method / Algorithm	Criteria	**Yes / No based on A matrix**
Cholesky	SPD	NO -NOT symmetric
LU	invertible, principal minor $\neq 0$	YES - invertible, positive definite
QR	real nonsingular matrix	YES - real nonsingular matrix
Jacobi	$\rho(M^{-1}N) < 1$	NO - $\rho(M^{-1}N) > 1$
Gauss Seidel	$\rho(M^{-1}N) < 11$	NO - $\rho(M^{-1}N) > 1$
SOR	$\rho(M^{-1}N) < 1$	NO - $\rho(M^{-1}N) > 1$
CG	SPD	NO - NOT SPD
CGS	invertible	YES - invertible
BiCGSTAB	invertible	YES - invertible
Schulz PR2	full rank, $\rho(I - AM_0) < 1$	YES - full rank, $\rho(I - AM_0) < 1$
Schulz CG	full rank, $\rho(I - AM_0) < 1$	YES - full rank, $\rho(I - AM_0) < 1$
GMRES	any real matrix	YES - real matrix

Table 9 shows the results for this cylinder test case with a larger coefficient matrix A. Highlighted are results for using different solvers with different stopping criteria. The operation time and the error from the exact solution are reported. We can see that backslash is a little slower than LU and faster than QR with similar error. Also, backslash cost 20 times more in operation time than iterative methods CGS and BiCGSTAB. Backslash also cost 10 times more in operation time than GMRES with similar error. Comparing Case 2 (shown in Table 9) to Case 1 (shown in Table 5), we can see that the operation time increases as N gets

larger. But operation time for direct methods increase much more than iterative methods. In other words, computational time for iterative methods increases in a reasonable range. Schulz CG method has a much larger computing time but it reduces the error. However, stopping criteria decreases here but the error increases. This happens because the stopping criteria 10^{-6} only has two iterations and the stopping criteria 10^{-9} and 10^{-12} has three iterations. M_k updates each iteration, so for different iterations, M_k is different. Then, the residual norm will vary as well.

Table 9: Case 2–Results for different solvers with $N = 1280$, $a = 2.5$ and $\delta = 0.5 * a * 2 * \pi/N$.

Method / Algorithm	stopping criteria	operation time	error from the exact sln
backslash	none	1.465171s	2.531431081399e-003
LU	none	1.358789s	2.531431081391e-003
QR	none	6.786379s	2.531431081399e-003
CGS	$2 * 10^{-15}$	0.084721s	2.531431081400e-003
BiCGSTAB	$1.8 * 10^{-15}$	0.070674s	2.531431081400e-003
Schulz CG	10^{-6}	43.039379s	8.031222226275e-004
Schulz CG	10^{-9}	50.761244s	1.693959743035e-003
Schulz CG	10^{-12}	50.139065s	1.693959743035e-003
GMRES	10^{-15}	0.182988s	2.531431081388e-003

3.3 Case 3: Filament in Stokes fluid, N=501

Here we know the velocity of movement of a filament immersed in a Stokes fluid and solve for the force on the filament. Table 10 highlights criteria for certain methods. For this coefficient matrix A, direct methods LU, QR and cholesky and iterative methods CG, CGS, BiCHSTAB, Schulz PR2, Schulz CG and GMRES could be used.

Table 10: Case 3–Solvers and criteria.

Method / Algorithm	Criteria	**Yes / No based on A matrix**
Cholesky	SPD	YES - SPD
LU	invertible, principal minor $\neq 0$	YES - invertible, principal minor $\neq 0$
QR	real nonsingular matrix	YES - real nonsingular matrix
Jacobi	$\rho(M^{-1}N) < 1$	NO - $\rho(M^{-1}N) > 1$
Gauss Seidel	$\rho(M^{-1}N) < 1$	YES - $\rho(M^{-1}N) < 1$
SOR	$\rho(M^{-1}N) < 1$	YES - $\rho(M^{-1}N) < 1$
CG	SPD	YES - SPD
CGS	invertible	YES - invertible
BiCGSTAB	invertible	YES - invertible
GMRES	any real matrix	YES - real matrix

Table 11 shows the results for the filament in Stokes with different solvers with different stopping criteria. The operation time and the error from the exact solution are reported. Cholesky is the fastest direct method, even faster than backslash. GMRES method is the fastest method among the iterative methods. We can see that most of the iterative methods are slower than the direct methods except for GMRES. Computational time increases for iterative methods as the stopping criteria decreases.

Table 11: Case 3–Results for different solvers with $N = 501$ and $\delta = 1/(3 * N/4)$.

Method / Algorithm	stopping criteria	operation time
backslash	none	0.094110s
LU	none	0.110517s
QR	none	0.492061s
cholesky	none	0.086663s
Gause Seidel ite	10^{-2}	0.168700s
Gause Seidel rel	10^{-2}	0.230589s
Gause Seidel ite	10^{-4}	0.194310s
Gause Seidel rel	10^{-4}	0.299491s
SOR ite $\omega = 0.6$	10^{-2}	0.195088s
SOR rel	10^{-2}	0.235641s
SOR ite	10^{-4}	0.220053s
SOR rel	10^{-4}	0.277532s
CG	$1.6 * 10^{-4}$	0.105268s
CGS	$6.5 * 10^{-6}$	0.120575s
BiCGSTAB	$1.7 * 10^{-5}$	0.130737s
GMRES	$1.2 * 10^{-3}$	0.077741s

3.4 Case 4: Filament in a Brinkman fluid with N=501

Similar to the previous case, we know the velocity of the filament and need to solve for the force on the filament. In this case, we are solving Brinkman equation (vs. Stokes) and Eq. (1) has an extra term. Brinkman flow is given by

$$\mu \Delta \mathbf{u} - \frac{\mu}{k} \mathbf{v} = \nabla p - \mathbf{F},$$

and can also be solved by regularized fundamental solutions [7].

Table 12 highlights properties of the A matrix for this test case. Direct methods LU, QR and cholesky, iterative methods CG, CGS, BiCHSTAB, Schulz PR2, Schulz CG and GMRES could be used to solve for forces.

Table 13 shows results for the filament in a Brinkman fluid with coefficient matrix A. By observation, only one iterative method, GMRES, is faster than direct

Table 12: Case 4–Solvers and criteria.

Method / Algorithm	Criteria	**Yes / No based on A matrix**
Cholesky	SPD	YES - SPD
LU	invertible, principal minor $\neq 0$	YES - invertible, principal minor $\neq 0$
QR	real nonsingular matrix	YES - real nonsingular matrix
Jacobi	$\rho(M^{-1}N) < 1$	NO - $\rho(M^{-1}N) > 1$
Gauss Seidel	$\rho(M^{-1}N) < 1$	YES - $\rho(M^{-1}N) < 1$
SOR	$\rho(M^{-1}N) < 1$	YES - $\rho(M^{-1}N) < 1$
CG	SPD	YES - SPD
CGS	invertible	YES - invertible
BiCGSTAB	invertible	YES - invertible
GMRES	any real matrix	YES - real matrix

methods. In general, direct methods are faster than iterative methods. However, all the methods used here have more elapsed computational time than Case 3, for a filament in Stokes fluid, as shown is Table 11.

Table 13: Case 4–Results for different solvers with $N = 501$ and $\delta = 1/(3 * N/4)$.

Method / Algorithm	stopping criteria	operation time
backslash	none	0.098777s
LU	none	0.108533s
QR	none	0.443145s
cholesky	none	0.087869s
Gause Seidel ite	10^{-2}	0.147021s
Gause Seidel rel	10^{-2}	0.203388s
Gause Seidel ite	10^{-4}	0.180132s
Gause Seidel rel	10^{-4}	0.267056s
SOR ite $\omega = 0.6$	10^{-2}	0.183691s
SOR rel	10^{-2}	0.207920s
SOR ite	10^{-4}	0.196745s
SOR rel	10^{-4}	0.273506s
CG	$2.4 * 10^{-4}$	0.118171s
CGS	$6.4 * 10^{-6}$	0.118292s
BiCGSTAB	$3.3 * 10^{-5}$	0.127653s
GMRES	$1.7 * 10^{-3}$	0.080534s

4 Discussion

As we assumed, the iterative methods worked faster than direct methods if the coefficient matrix A of the linear system was large as shown in test Case 2. By observation, the backslash would check the property of the matrix A first, then decide which method to use. For our matrix A, it uses the LU factorization method.

For test Case 2, GMRES method was more efficient than the backslash command with similar error. For test Case 1 and 2, Schulz CG method could reduce the error but was not as computationally efficient. For problems that we need more accuracy, Schulz CG method is a good choice.

Then we tried some preconditioning with GMRES as well. The preconditioner like FROB made the GMRES faster but the total cost of time is longer than the GMRES method without preconditioner. Since the QR factorization in solving

the preconditioner matrix M cost too much time. It is obvious that based on computational time, the preconditioner did not work well for our problem. Preconditioning methods are suited well for other methods and may not be good for our linear system. We need to keep looking for efficient methods.

In Alleon's paper [3], he showed that for Krylov subspace methods (e.g. GMRES) combined with a preconditioner could dramatically reduce the operation time. In section 2 we described the FROB preconditioner which chose the nonzero pattern with the same position as large entries in A^{-1}. Results were presented for this in section 3.1 and 3.2 for the cylinder test case in a 2-D fluid. This also corresponds to the 3rd level nearest neighbours for a geometric smooth object (3rd far-away object for nonsmooth or disconnected, or far-away edges). Alleon instead used a fast multipole method (FMM, an algorithm for computing approximate matrix-vecter products for electromagnetic scattering problems). The basic idea of the algorithm is to first generate the mesh by dividing former rectangles to get new leaf-boxes. Then, depending on the physical distance, compute iterations amongst degrees of different level are used. The new idea of the algorithm is making an outer and inner solver scheme. The outer solver uses a high accuracy FMM preconditioned FGMRES method and the inner solver uses FROB and a low accuracy FMM preconditioned GMRES method. At the end of the paper he mentioned spectral low-rank updates. Since in many cases, removing the smallest eigenvalues can greatly improve the convergence. Then, by combining FROB preconditioner with a rank-k spectral update, he got faster convergence. This type of method could be used in the biofluids application in future work.

References

[1] G Allaire and S Mahmoud Kaber. *Numerical Linear Algebra*. Springer, France, 2008.

[2] G Alleon, M Benzi, and L Giraud. Sparse approximate inverse preconditioning for dense linear systems arising in computational electromagnetics. *Numerical Algorithms*, 15:16,1–15, 1997.

[3] G Alleon, B Carpentieri, IS Duff, L Giraud, J Langou, E Artin, and G Sylvand. Efficient parallel iterative solver for the solution of large dense linear system arising from the boundary element method in electromagnetism. *CERFACS Technical Report TR/PA/03/65*, 15:1–15.

[4] M Benzi. Preconditioning techniques for large linear systems: A survey. *Journal of Computational Physics*, 60:182,418–477, 2002.

[5] O Cahuenas, L Hernandez-Ramos, and M Raydan. Pseudoinverse preconditioners and iterative methods for large dense linear least-squares problems. *Lecturas en Ciencias de la Computacion*, 16:1316–6239–16, 2012.

[6] R Cortez. The method of regularized Stokeslets. *SIAM Journal of Scientific Computing*, 23:1204–1225, 2001.

[7] R Cortez, B Cummins, K Leiderman, and D Varela. Computation of three-dimensional brinkman flows using regularized methods. *J Comp Phys*, 229:7609–7624, 2010.

[8] R Cortez, L Fauci, and A Medovikov. The method of regularized Stokeslets in three dimensions: Analysis, validation, and application to helical swimming. *Physics of Fluids*, 17:031504–1–14, 2005.

[9] TA Davis. *Direct methods for sparse linear systems*. Society for Industrial and Applied Mathematics, Philadelphia, PA, 2006.

[10] J Demmel. *Applied numerical linear algebra*. Society for Industrial and Applied Mathematics, Philadelphia, 1997.

[11] Mathworks. Cond - condition number of matrix, 2014.

[12] Mathworks. mldivide, solve systems of linear equations, 2014.

[13] Mathworks. Norm - vector and matrix norms, 2014.

[14] Y Saad. *Iterative Methods for Sparse Linear Systems, Second Edition*. Society for Industrial and Applied Math, Minneapolis, Minnesota, 2003.

[15] C Smith, AF Peterson, and R Mittra. The biconjugate gradient method for electromagnetic scattering. *IEEE TRANSACTIONS ON ANTENNAS AND PROPAGATION*, 3:938–940, 1990.

[16] L Trefethen and D Bau. *Numerical Linear Algebra*. Society for Industrial and Applied Math, Cornell University Ithica, New York, 1997.

[17] H Vorst. Bi-cgstab: A fast and smoothly converging variant of bi-cg for the solution of nonsymmetric linear systems. *SIAM Journal of Scientific Statistic Computing*, 14:631–644, 1992.

5 Appendix

5.1 Code for method Schulz CG

```
function x=Schulz_CG(A,b,tol,Nmax)
%initialization
x=zeros(length(b),1);

M=A'/norm(A,2)^2;%pseudoinverse. If A full rank then M*A is SPD
r=M*b-M*A*x;%original is r=b-A*x;but we need to solve MAx=Mb
p=r;
gama=norm(r,2)^2;
n=1;
while gama>tol&&n<Nmax
y=M*A*p;
%size(y)
alpha=gama/dot(y,p);
x=x+alpha*p;
r=r-alpha*y;
beta=norm(r,2)^2/alpha;
gama=norm(r,2)^2;
p=r+beta*p;
M=2*M-M*A*M;
n=n+1;
tol;
end
n;
end
```

5.2 Code for method Gauss Seidel with iterative residual

```
function x=gaussseidel(A,b,p,tol)      %gauss-seide(matrix A,RHS b,
    initial,tolerance)
M2=tril(A);                            %M=D-E
N2=diag(diag(A))-triu(A);              %N=F

G=M2\N2;                               %G=M^-1*N=(E-L)^-1*F
b_new=M2\b;                            %b_new=M^-1*b=(E-L)^-1*b

x=G*p+b_new;                           %x_n+1=G*x_n+b_new
n=1;
while norm(x-p,2)>=tol          %if 2-norm of( x_n-x_n-1)>=tolerance
    p=x;                               %then caculate one more time
        using x_n
    x=G*p+b_new;                       %then we get x_n+1
    n=n+1;                             % # of iterations
end
n
```

5.3 Code for method Gauss Seidel with relative residual

```
function x=gaussseidel2(A,b,p,tol)      %gauss-seide(matrix A,RHS b,
    initial,tolerance)
M2=tril(A);                            %M=D-E
N2=diag(diag(A))-triu(A);              %N=F

G=M2\N2;                               %G=M^-1*N=(E-L)^-1*F
b_new=M2\b;                            %b_new=M^-1*b=(E-L)^-1*b

```

```
x=G*p+b_new;                                %x_n+1=G*x_n+b_new
n=1;
p1=p;
tol_0=norm(b-A*p1,2);
while norm(b-A*x,2)/tol_0>=tol              %if 2-norm of( x_n-x_n-1)>=
    tolerance
    p=x;                                    %then caculate one more time
        using x_n
    x=G*p+b_new;
    %then we get x_n+1
    %x_k+1=M^-1*N*x_k+M^-1b
    n=n+1;                                  % # of iterations
end
n
```

5.4 Code for method SOR with iterative residual

```
function x=SOR2(A,b,p,w,tol)
D=diag(diag(A));
E=diag(diag(A))-tril(A);
F=diag(diag(A))-triu(A);

M=D/w-E;
N=(1-w)/w*D+F;

G=M\N;
b_new=M\b;                          %b_new=M^-1*b=(E-L)^-1*b

x=G*p+b_new;                                %x_n+1=G*x_n+b_new
```

```
n=1;
p1=p;
tol_0=norm(b-A*p1,2);
while norm(b-A*x,2)/tol_0>=tol          %if 2-norm of( x_n-x_n-1)>=
    tolerance
    p=x;                                %then caculate one more time
        using x_n
    x=G*p+b_new;                        %then we get x_n+1
    n=n+1;                              % # of iterations
end
n
```

5.5 Code for method SOR with relative residual

```
function x=SOR2(A,b,p,w,tol)
D=diag(diag(A));
E=diag(diag(A))-tril(A);
F=diag(diag(A))-triu(A);

M=D/w-E;
N=(1-w)/w*D+F;

G=M\N;
b_new=M\b;                          %b_new=M^-1*b=(E-L)^-1*b

x=G*p+b_new;                        %x_n+1=G*x_n+b_new
n=1;
p1=p;
tol_0=norm(b-A*p1,2);
```

```
while norm(b-A*x,2)/tol_0>=tol          %if 2-norm of( x_n-x_n-1)>=
    tolerance
    p=x;                                %then caculate one more time
        using x_n
    x=G*p+b_new;                        %then we get x_n+1
    n=n+1;                              % # of iterations
end
n
```

5.6 Code for preconditioner NO.1

```
function x=preconditionerNO1(A,k)
n_row=size(A,1);
n_column=size(A,2);
M=zeros(size(A));
M1=zeros(k,size(A,2));
E=eye(size(A));
E_new=zeros(k,size(A,2));
for j=1:n_column
    A_new=zeros(size(A,1),k);
    Q_new=zeros(size(A,1),k);
    [b,m]=sort(A(:,j));
    position=m(n_row-k+1:n_row);
    [b1,m1]=sort(position(:));
    A_new(:,1:k)=A(:,b1(:));%get full rank matrix first
    E_new(1:k,:)=E(b1(:),:);
    [Q,R]=qr(A_new);
    Qtemp=Q';
    Q_new(:,1:k)=Qtemp(:,b1(:));
```

```
19      M1(:,j)=R\(Q_new*E_new(:,j));
        M(b1(:),j)=M1(:,j);
21  end
    x=M;
23  end
```

5.7 Code for preconditioner NO.4

```
1   function x=preconditionerNO4(A,eps)
    n_row=size(A,1);
3   n_column=size(A,2);
    maximum=max(max(abs(A)));
5   tol=eps*maximum;
    M=zeros(size(A));
7   E=eye(size(A));
    num_tot=0;
9   num=zeros(1,size(A,2));
    for n=1:n_column
11      num(1,n)=0;
    b1=zeros(1,size(A,2));
13      for m=1:n_row
            if abs(A(m,n))>=tol
15              num(1,n)=num(1,n)+1;
                b1(1,num(1,n))=m;
17              A_new(:,num(1,n))=A(:,m);
                num_tot=num_tot+1;
19          end
        end
21      if num(1,n)>0
```

```
        M1=zeros(num(1,n),size(A,2));
        E_new=zeros(num(1,n),size(A,2));
        b1(1,:);
        b1(b1(1,:)==0)=[];
        E_new(1:size(b1,2),:)=E(b1(1,:),:);
        if n>=2&&num(1,n)<num(1,n-1)
        A_new(:,num(1,n)+1:num(1,n-1))=[];
        end
        [Q,R]=qr(A_new);
        Qtemp=Q';
        Q_new=zeros(size(A,1),size(b1,2));
        Q_new(:,1:size(b1,2))=Qtemp(:,b1(1,:));
        M1(:,n)=R\(Q_new*E_new(:,n));
        M(b1(1,:),n)=M1(:,n);
        else M(:,n)=0;
        end
    end
    x=M;
    min_mj=min(num)
    max_mj=max(num)
    num_tot;
    nonzeros=num_tot/(n_row*n_column)
    end
```