EFFICIENT AND ROBUST SOLUTION STRATEGIES FOR SADDLE-POINT SYSTEMS

By

Jeremy Chiu, Lola Davidson, Aritra Dutta, Jia Gou, Kak Choon Loy, Mark Thom

Mentor: Dimitar Trenev, ExxonMobil

IMA Preprint Series #2440

(October 2014)

Efficient and Robust Solution Strategies for Saddle-Point Systems

Jeremy Chiu, Lola Davidson, Aritra Dutta, Jia Gou,
Kak Choon Loy, Mark Thom
Mentor: Dimitar Trenev, ExxonMobil

August 6-16, 2014

Mathematical Modeling in Industry XVIII
University of British Columbia

1 Introduction and Problem Statement

Linear saddle-point systems occur in many different fields of research including economics, finance, computer science, and engineering. In particular for mathematicians, finite difference schemes and finite element methods will often give rise to large and sparse saddle-point systems. Some problems in practice demand the solution of systems with hundreds of millions of unknowns, eliminating the applicability of direct solvers. Oftentimes the systems are sparse, which suggests the use of iterative schemes. Unfortunately, saddle-point systems are frequently non-symmetric, indefinite, and poorly conditioned - characteristics that pose a challenge for out-of-the-box iterative solvers.

In this workshop, we explore various kinds of iterative methods to solve linear, sparse saddle-point systems arising in the mixed finite element approximation of certain partial differential equations (PDEs). In our search for an efficient and robust solution strategy we will discuss both general methods, which will not make use of the matrix's particular structure, as well as methods tailored specifically to saddle-point systems or systems coming from a finite-element discretizations of PDEs.

In the rest of this section, we pose the problem and very briefly give some mathematical background. In Section 2, we look into the performance of some state-of-the-art iterative methods when applied to our systems. As the methods discussed are general "black box" methods for solving linear systems of equations, they do not take advantage of the extra knowledge we have of the structure of the matrices we wish to invert. To make use of that *a priori* knowledge, we will discuss a way to speed-up the methods' convergence by applying a preconditioner suitable for the particular systems at hand. In Section 3 we will discuss a different set of methods, namely the Uzawa and Augmented Lagrangian methods. These are two stationary iteration methods, which exploit the specific block structure of the saddle-point systems. Finally, in Section 4 we will fully take advantage of our *a priori* knowledge of the systems' origin by looking at several levels of mixed finite element approximations of our partial differential equation and constructing multilevel solution algorithms and preconditioners.

1.1 Mathematical Background

In this report, we will call a saddle-point system a block linear system of the form

$$\begin{bmatrix} A & B_1^T \\ B_2 & -C \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} f \\ g \end{bmatrix}, \qquad (1)$$

where

$$A \in \mathbb{R}^{n\times n},\ B_1, B_2 \in \mathbb{R}^{m\times n},\ C \in \mathbb{R}^{m\times m}$$

and the block matrices satisfy the following properties:

A1. A and C are square matrices, with $n \geq m$;

A2. $A + A^T$ is positive semidefinite;

A3. C is symmetric and semidefinite;

A4. $B_1, B_2 \neq 0$.

The system (1) can be written in the general form

$$\mathcal{A}u = b, \tag{2}$$

where $\mathcal{A}$ contains the 2×2 blocked structure of (1), $u = [x \;\; y]^T \in \mathbb{R}^{n+m}$, and $b = [f \;\; g]^T \in \mathbb{R}^{n+m}$.

We will not discuss here the solvability conditions for equation (1). We will instead refer to [1] for an excellent review on the subject.

Throughout this report, all reported numerical experiments were performed on specific saddle-point systems satisfying the following additional conditions:

B1. A is symmetric and positive-definite;

B2. $B_2 = -B_1 = B$;

B3. $C = 0$.

That is to say, the matrices we invert for are of the following form

$$\mathcal{A} = \begin{bmatrix} A & -B^T \\ B & 0 \end{bmatrix}. \tag{3}$$

Notice that such matrices are positive semi-definite, but not symmetric. While we can make the systems symmetric (by multiplying the second block-row equation by -1), such transformation makes them indefinite. Thus, we have not reduced the complexity of our problem significantly by restricting ourselves to this less general class of matrices. In addition, most of the solution strategies discussed have straightforward generalizations for systems that do not satisfy the additional conditions above (and in particular, all of the solution strategies discussed, work in the case where $C \neq 0$).

In this context we can quote a Theorem from [1].

Theorem: *Assume A is a symmetric positive-definite matrix, $B_1 = B_2 = B$, and C is symmetric positive-semidefinite. If $\mathrm{Ker}(C) \cap \mathrm{Ker}(B) = \{0\}$, then the saddle-point system matrix $\mathcal{A}$ is non-singular. In particular, $\mathcal{A}$ is invertible if B has full rank.*

1.2 Schur's Complement Reduction

Schur's complement reduction exploits the following simple LU block-factorization of a saddle-point system matrix

$$\mathcal{A} = \begin{bmatrix} A & B_1^T \\ B_2 & -C \end{bmatrix} = \begin{bmatrix} I & 0 \\ B_2A^{-1} & I \end{bmatrix} \begin{bmatrix} A & B_1^T \\ 0 & S \end{bmatrix}, \tag{4}$$

where the *Schur complement* S is given by

$$S = -(C + B_2 A^{-1} B_1^T) \in \mathbb{R}^{m\times m}.$$

It is clear that $\mathcal{A}$ is non-singular if and only if both A and S are non-singular. Multiplying both sides of (1) by the matrix

$$\begin{bmatrix} I & 0 \\ -B_2 A^{-1} & I \end{bmatrix}$$

produces

$$\begin{bmatrix} A & B_1^T \\ 0 & S \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} f \\ g - B_2 A^{-1} f \end{bmatrix}.$$

This transforms the original system of size $(m+n)\times(m+n)$ into two systems of sizes $m\times m$ and $n\times n$, the solutions of which now depend only on the inverses of S and A, respectively.

This approach is not very useful in cases where the matrix A^{-1} is difficult to compute. Moreover, since the matrix S is generally full even when the matrix A is sparse, it would be expensive to find S^{-1}. Recall that the operational complexity for solving the linear system $Sy = g - B_2 A^{-1} f$ is $O(m^3)$. This cost makes a naive implementation of Schur's complement method far too slow when the dimension is large. Other cost saving techniques must be employed alongside Schur's factorization for it to be a viable scheme. It is important to note that many of our iterative methods rely on Schur's factorization to some degree.

2 Krylov Subspace Methods

Classical stationary methods, such as Gauss-Seidel or Successive Over-relaxation, are unappealing to solve a large sparse linear system of equations as their convergence rate is generally poor. In the last 30 years, fast iterative solvers designed to solve such systems mostly fall in the framework of the *Krylov Subspace Methods.*

For completeness, we will very briefly discuss the basic idea of the Krylov subspace methods here. The reader is referred to [1] for a more complete overview, and to [2] for a detailed analysis of some of the more well-known methods.

The general idea behind the Krylov subspace methods is constricting a sequence $u_1, u_2, \ldots$ of approximate solutions to a given linear system

$$\mathcal{A}u = b, \tag{5}$$

such that the residuals, given by $r_k = b - \mathcal{A}u_k$, are orthogonal to subspaces growing in dimension, whose basis is relatively easy to compute and only involves applications of the matrix $\mathcal{A}$.

More precisely, the Krylov subspaces $\mathcal{K}_k$ are defined as

$$\mathcal{K}_k(\mathcal{A}, r_0) = \text{span}\{r_0, \mathcal{A}r_0, ...\mathcal{A}^{k-1}r_0\},$$

and the iterate $u_k \in u_0 + \mathcal{K}_k(\mathcal{A}, r_0)$ is taken such that the residual $r_k \in r_0 + \mathcal{A}\mathcal{K}_k(\mathcal{A}, r_0)$ is orthogonal to a kth-dimensional subspace $\mathcal{C}_k$.

The following theorem asserts that this is always possible

Theorem: *Suppose the Krylov subspace $\mathcal{K}_k(\mathcal{A}, r_0)$ has dimension k. If*
(a) $\mathcal{A}$ is symmetric positive-definite and $\mathcal{C}_k = \mathcal{K}_k(\mathcal{A}, r_0)$, or
(b) $\mathcal{A}$ is nonsingular and $\mathcal{C}_k = \mathcal{A}\mathcal{K}_k(\mathcal{A}, r_0)$,
then there exits a uniquely defined iterate $u_k \in u_0 + \mathrm{span}\{\mathrm{r}_0, \mathcal{A}\mathrm{r}_0, ...\mathcal{A}^{\mathrm{k}-1}\mathrm{r}_0\}$ for which r_k is orthogonal to $\mathcal{C}_k$.

The most well-known Krylov subspace methods are, in a sense, algorithms for efficiently computing those uniquely defined iterates. In the case of the *Conjugate Gradient* method, one works with $\mathcal{C}_k = \mathcal{K}_k(\mathcal{A}, r_0)$; in the case of the *Generalized Minimal Residual* method - with $\mathcal{C}_k = \mathcal{A}\mathcal{K}_k(\mathcal{A}, r_0)$; and so on. One can even generalize such algorithms to non-square matrices, as in the case of the *LSQR* method, which - albeit implemented differently - is mathematically equivalent to solving the (symmetric and positive definite) system $\mathcal{A}^T\mathcal{A}u = \mathcal{A}^T b$ with a conjugate gradient method.

2.1 Preliminary Numerical Results

Among the many available methods, we restricted our tests to Matlab's eleven built-in iterative solvers:

1. Biconjugate Gradient (*bicg*)
2. Biconjugate Gradient Stabilized (*bicgstab*)
3. *l*-Generalized Biconjugate Gradient Stabilized (*bicgstabl*)
4. Conjugate Gradient Squared (*cgs*)
5. Generalized Minimal Residual (*gmres*)
6. LSQR (*lsqr*)
7. Minimal Residual (*minres*)
8. Preconditioned Conjugate Gradient (*pcg*)
9. Quasi-Minimal Residual (*qmr*)
10. Symmetric LQ (*symmlq*)
11. Transpose-Free Quasi-Minimal Residual (*tfqmr*)

Name	Best Residual	Best Iteration	Runtime
bicg	1.2755×10^{-4}	1825	2.3643
bicgstab	1.6178×10^{-5}	2000	3.2868
bicgstabl	2.985×10^{-6}	1994	10.3299
cgs	1	0	2.567
gmres	5.6851×10^{-5}	2000	236.9806
lsqr	7.4754×10^{-5}	2000	3.2685
minres	1.3476×10^{-5}	2000	2.3866
pcg	1	0	0.0261
qmr	1.8278×10^{-5}	1991	3.956
symmlq	1.1765×10^{-4}	1682	2.0208
tfqmr	1	0	0.5073

Table 1: Summary of the performance of Matlab built-in iterative solvers

To get an understanding of how our methods will perform we tested all eleven of the built-in solvers without the use of preconditioners or factorizations. The tests were performed on a matrix $\mathcal{A} \in \mathbb{R}^{10565\times 10565}$ (coming from a PDE discretized on a mesh with 4194 triangles) for a fixed 2000 iterations. Table 1 summarizes the key results.

Notice that most methods had smaller residuals with respect to a larger iteration number. On the other hand, *pcg*, *cgs*, and *tfqmr* completely failed - the best approximation was the initial guess, the zero vector. This was expected for the basic conjugate gradient, *pcg*, as our system is indefinite. *tfqmr* failed due to stagnant iterations, and the residual of *cgs* blew up (see Figure 1) due to the conditioning of our matrix (we will talk more about that in the next section).

Among the convergent methods (i.e. where the residuals were decreasing), some of the methods had the property that the best residual came from a non-final iteration. This implies that for these methods, taking the next iteration does not guarantee a smaller residual - rather, the residual will shrink over the course of many steps. We plot the various Biconjugate Gradient methods which demonstrate this "shaky convergence" in Figure 2. In the plot, we also include the Symmetric LQ Method since it's convergence is also shaky. The most general, *bicgl*, had the best convergence rate. However, in our setup, each iteration of *bicgl* involved three sub-steps, as opposed to the single step for a *bicgstab* iteration. Thus, the higher accuracy was obtained at the cost of a longer computational time.

The remaining methods had a smoother convergence rate - up to numerics, each iteration is guaranteed to reduce the residual. Note that we took the restart parameter of *gmres* as $R = 30$, so every iteration for *gmres* actually involved R steps. This was reflected by the relatively long computation time.

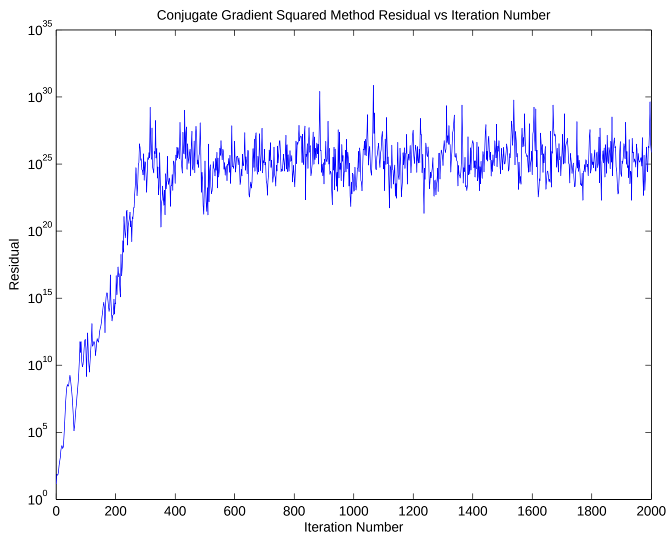


Figure 1: The residuals for *cgs* blow up as we iterate

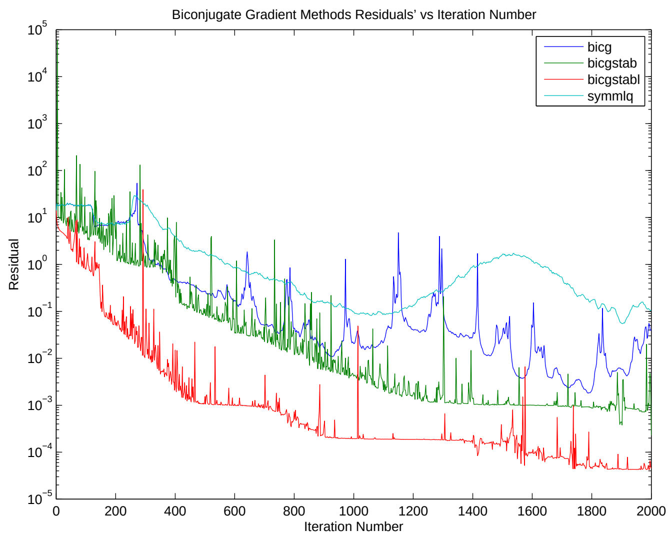


Figure 2: Biconjugrate Gradient Stabilized Method and it's extensions, and Symmetric LQ

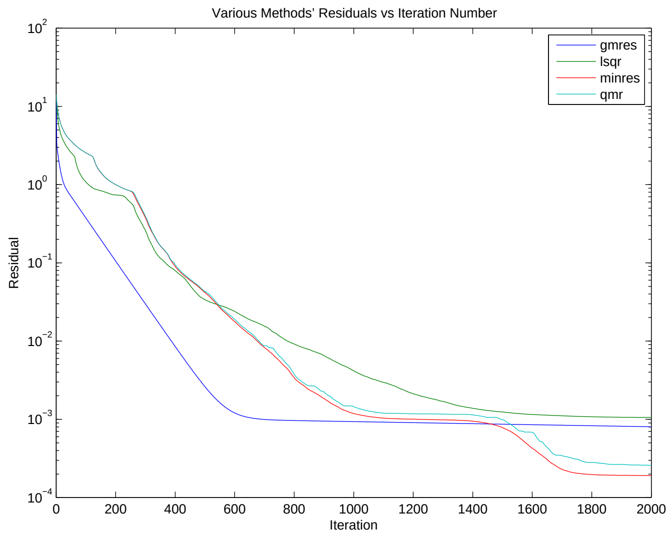


Figure 3: Minimal Residual Method and it's extensions

2.2 Systems of higher dimension

As previously noted, our problem currently deals with a mesh consisting of about 4000 triangles and 6000 edges, so the matrix we need to solve is roughly 10,000 by 10,000. As the mesh is refined, however, the matrix dimension can grow into the millions. Thus a method that can still perform reasonably fast with a large $\mathcal{A}$ is mandatory.

In Figure 4, we have plotted the computation time and number of iterations for the two most promising methods from our preliminary testing, *bicgstabl* and *minres*, on a series of problems growing in size. As is evident from the plot, both the computation time and the number of iterations grow unacceptably large as we refine the mesh. While the growth in computation time may be mitigated by the use parallel computing, the iteration number cannot because computing consecutive iterations is an innately serial process. This suggests we need to find a way to reduce the number of iterations. As we will see in the following section, this is somewhat possible with the use of preconditioners.

2.3 Preconditioners

Even with iterative methods, a linear system may be incredibly difficult to solve, especially when $\mathcal{A}$ is "close to singular". A way of measuring this for a given matrix is through its *condition number*. Recall that the condition number of a matrix A is defined by

$$\kappa(A) = \|A\|\|A^{-1}\|,$$

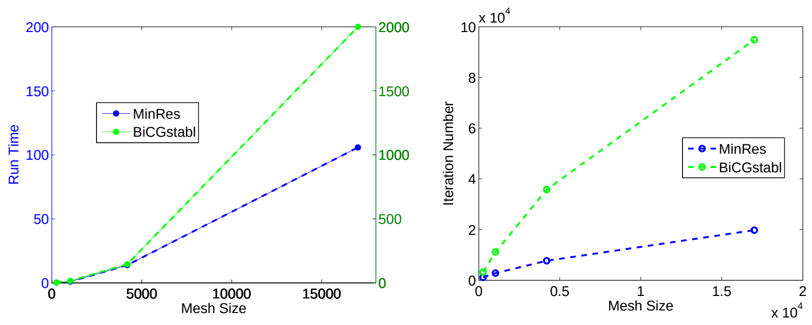


Figure 4: Computation work vs mesh size

where the matrix norm is subordinate to the vector norm used on our space (usually l_2), and is given by

$$\|A\| = \sup\{\|Ax\| : x \in \mathbb{R}^n, \|x\| = 1\}.$$

If the condition number of A is of manageable size, the matrix is said to be well-conditioned. On the other hand, a matrix with a large condition number is said to be ill-conditioned. For an ill-conditioned matrix A, there will be cases in which the solution of the system $Ax = b$ will be very sensitive to small changes in the vector b. In other words, to attain a certain precision in determining x, we shall require a much higher precision in b. In the context of the iterative solvers, this means that the tolerance - or how big the residual $||b - Ax||$ is allowed to be - needs to be lowered significantly (and sometimes infeasibly, due to numerical errors) if we want to get below a certain tolerance level for the error of our approximation.

Krylov Space methods work reasonably well when $\mathcal{A}$ is well-conditioned and suitable for the method at hand (say, symmetric for *minres*). But for an ill-conditioned matrix their performance can deteriorate immensely. A way to tackle this problem is to transform the original system (5) into a better-conditioned one, or one more suitable for the numerical method

$$\hat{\mathcal{A}}\hat{u} = \hat{b}. \tag{6}$$

Typically this is done by multiplying both sides of the equation (5) by a matrix P^{-1} (we shall call P the *preconditioner*) and solving $P^{-1}\mathcal{A}u = P^{-1}b$ instead. In this case $\hat{\mathcal{A}} = P^{-1}\mathcal{A}$, $\hat{b} = P^{-1}b$, and the solution $\hat{u}$ of the system (6) is the same as the solution u of (5). If the matrix P^{-1} is "close" to the inverse of our original matrix $\mathcal{A}$, we will have $\kappa(\hat{\mathcal{A}}) < \kappa(\mathcal{A})$, and the preconditioned system will be better-conditioned.

Note that even if we don't change the condition number we may use this preconditioning technique to modify our matrices in a favorable way. For example, multiplying our test matrices

$$\mathcal{A} = \begin{bmatrix} A & -B^T \\ B & 0 \end{bmatrix}$$

by a “preconditioner”

$$P = P^{-1} = \begin{bmatrix} I & 0 \\ 0 & -I \end{bmatrix}$$

makes our matrix symmetric. We have already made use of that when testing the performance of the *minres* method.

We should note that apart from the technique described above, called left preconditioning, one can use right preconditioning, where the system solved is $\mathcal{A}P^{-1}Pu = b$ i.e. $\hat{\mathcal{A}} = \mathcal{A}P^{-1}$, $\hat{b} = b$, and $\hat{u} = Pu$ (or, equivalently, $u = P^{-1}\hat{u}$). More generally, one can use split preconditioning solving

$$P_L^{-1}AP_R^{-1}P_Ru = P_L^{-1}b.$$

In practice Krylov space solvers are almost always applied with some kind of preconditioning.

When looking for a suitable preconditioner for our saddle point systems, we started from the Schur factorization of $\mathcal{A}$ given in (4) which we restate here in the symmetric case

$$\mathcal{A} = \begin{bmatrix} A & B^T \\ B & 0 \end{bmatrix} = \begin{bmatrix} I & 0 \\ BA^{-1} & 0I \end{bmatrix} \begin{bmatrix} A & 0 \\ 0 & S \end{bmatrix} \begin{bmatrix} I & A^{-1}B^T \\ 0 & I \end{bmatrix}.$$

The idea is to take the center matrix in the above equation as a good approximation for $\mathcal{A}$, and use it as a preconditioner. Thus all of the preconditioners that we tried were of the form

$$P = \begin{pmatrix} \hat{A} & 0 \\ 0 & \hat{S} \end{pmatrix},$$

where $\hat{A}$ is an approximation of A and $\hat{S}$ is an approximation of S.

2.3.1 The SIMPLE Scheme

The first preconditioner that we tested took

$$\hat{A} = \operatorname{diag}(A) = D,\ \hat{S} = B_2D^{-1}B_1^T.$$

This has been referred to as the *SIMPLE* scheme (Semi-Implicit Method for Pressure-Linked Equations). Implementing this preconditioner in any built-in solver requires either explicitly finding P^{-1} or writing a function that applies P^{-1} to a given vector. We chose the latter for two reasons. First, since we are dealing with large matrices, we do not wish to store P^{-1}. Second, finding P^{-1} requires explicitly computing $\hat{S}^{-1}$, which could be costly. On the other hand, since in our trial $B_2 = B_1$, we could use *pcg* to quickly approximate $\hat{S}^{-1}x$ for

Name		Residual	Iterations	Runtime
bicgstab	with P	3.8023×10^{-5}	22	51.9105
	no P	4.2722×10^{-5}	1959	5.3098
bicgstabl	with P	1.3017×10^{-6}	14.25	57.6201
	no P	1.9807×10^{-6}	1995	13.8173
minres	with P	4.1417×10^{-6}	41	40.9778
	no P	1.3246×10^{-5}	2000	3.8629
symmlq	with P	9.7868×10^{-5}	34	33.3071
	no P	1.1852×10^{-4}	1690	3.0107

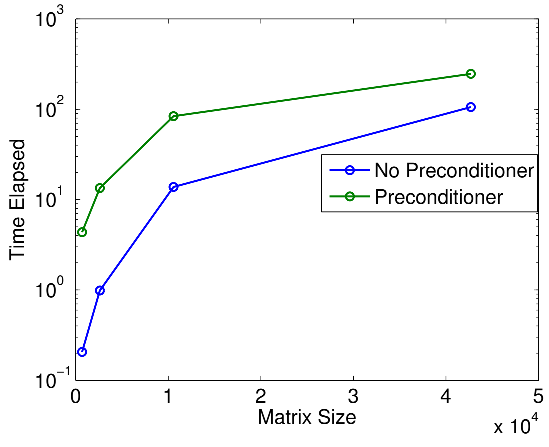
Table 2: Highlighting the difference between methods with and without the SIMPLE preconditioner P

a given vector x. This SIMPLE scheme was the best preconditioner that we tested, both in terms of run time and number of iterations of the outer solver for $\mathcal{A}$.

Since the block A is a “well-behaved” matrix (in this case, symmetric and positive-definite), we also decided to try using $\hat{A} = A$ while keeping $\hat{S} = B_2 D^{-1} B_1^T$. This time we had to use *pcg* as an inner solver to approximate A^{-1} in addition to $\hat{S}^{-1}$. For this reason, this preconditioner did not perform as fast as the SIMPLE scheme and the small gain in accuracy was not worth the additional computational time.

We took the SIMPLE preconditioner P and used it with select Matlab built-in iterative methods. It is worth noting that with P, *cgs* and *tfqmr* managed to converge (recall without any preconditioner, both failed). Table 2 summarizes our findings for the best performing methods. Notice that in all cases the iteration number absolutely plummets. On the other hand, even though there are much fewer iterations, the run time grows since the cost per iteration grew drastically. This time-cost is incurred from running *pcg* to solve $\hat{S}^{-1}$ at every iteration.

Figure 5 demonstrates that when we apply our preconditioner, the iteration number scales well as we refine the mesh. This tells us that the SIMPLE preconditioner is quite powerful. In fact applying this preconditioner to the *gmres* method, we were able to speed up the run-time of the method more than twenty times! Unfortunately, the time cost is still scaling poorly due to the costly *pcg* when applying the preconditioner each iteration. Indeed, it should be noted that while the outer iterations stay more-or-less constant as we have seen, the inner iteration count (i.e. the iterations needed to apply the preconditioner) rises with the mesh refinements. This increase is lower, however, than the increase of the outer iterations without a preconditioner, proving that this preconditioning technique is still viable. We should, however, explore other preconditioners, or ways to speed-up the application of the SIMPLE preconditioner.



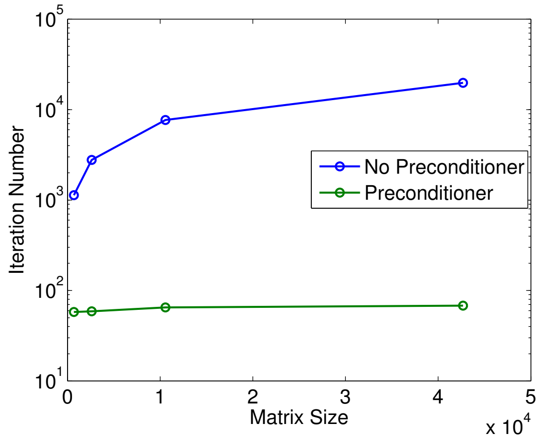


Figure 5: *minres* computation work vs mesh size

Name		Residual	Iterations	Runtime
bicgstab	$\hat{A}_D$	5.7674×10^{-7}	29	49.9066
	$\hat{A}_T$	9.7044×10^{-7}	34	166.7375
bicgstabl	$\hat{A}_D$	5.6136×10^{-7}	14.5	53.3905
	$\hat{A}_T$	6.89×10^{-7}	15.5	159.968
cgs	$\hat{A}_D$	1.1643×10^{-7}	30	51.7201
	$\hat{A}_T$	1.4245×10^{-7}	30	150.8966
gmres	$\hat{A}_D$	9.6838×10^{-7}	1	19.6727
	$\hat{A}_T$	9.8626×10^{-7}	1	56.1166
minres	$\hat{A}_D$	6.6804×10^{-7}	46	41.0382
	$\hat{A}_T$	9.2504×10^{-7}	45	106.7194
symmlq	$\hat{A}_D$	7.3971×10^{-7}	45	39.8289
	$\hat{A}_T$	5.0801×10^{-7}	45	112.3874
tfqmr	$\hat{A}_D$	4.9867×10^{-7}	29	51.7291
	$\hat{A}_T$	4.1646×10^{-7}	29	137.9618

Table 3: Summary of the performance differences using the diagonal $\hat{A}_D$ or the tridiagonal $\hat{A}_T$ in the SIMPLE preconditioner.

2.3.2 Banded Approximations and Reclustering

An approach to encode more information about A into an easily invertible approximation is using a band matrix. For example, instead of taking the diagonal, we can take $\hat{A} = T$, a tridiagonal matrix with entries equal to the corresponding entries in A. In the same spirit, we could take more bands of A to have an even closer approximation. We can also use the idea of reclustering the sparse matrix A so that it becomes a banded matrix with a relatively small width. Although we did not have the time to test any reclustering ideas, we were able to test the performance of the preconditioner obtained by using the tridiagonal matrix $\hat{A} = T$ and $\hat{S} = B_2 T^{-1} B_1^T$. As with the SIMPLE scheme, we used *pcg* to estimate $\hat{S}^{-1}$.

We tested using the tridiagonal band and compared the results to just the diagonal. The results are summarized in Table 3. Notice that the iteration count was about the same, but the run time was expectedly greater. The cost was, once again, dominated by the *pcg* solve for $\hat{S}^{-1}$. Thus we did not achieve a gain in efficiency by using this preconditioner. We suspect this is due to the fact that, in our case, the tridiagonal entries in A were somewhat arbitrary since local information on the mesh was not reflected as local in the columns of A. We believe that in other cases, where the superdiagonal and subdiagonal contain more information, using $\hat{A} = T$ instead of $\hat{A} = D$ may give rise to a better preconditioner.

3 Stationary Methods

Most classical stationary methods are too slow for large matrices. However, they can still be useful. For instance, simple stationary methods, such as the Jacobi method, may be used as a subroutine to smooth a system. Also, stationary methods used alongside Schur's complement may be quite powerful.

In this section we will look at two stationary iteration methods tailored specifically for solving saddle point systems. As before we will be considering the simplified case (3), but the generalization to (1) is mostly straightforward.

3.1 Uzawa's Algorithm

One way to solve (3) is to minimize the constrained optimization problem

$$\begin{cases} \min_x \frac{1}{2}\langle Ax, x\rangle - \langle f, x\rangle \\ \text{subject to } \; Bx = g. \end{cases}$$

The unconstrained version of the above problem can be written as

$$L(x, y) = \frac{1}{2}\langle Ax, x\rangle - \langle f, x\rangle + \langle y, Bx - g\rangle,$$

where $y \in \mathbb{R}^m$ is the Lagrange multiplier vector. Uzawa's method consists of a coupled iteration, starting with initial guesses x_0 and y_0, and obtaining consecutive approximations by

$$\begin{cases} Ax_{k+1} = f - B^T y_k \\ y_{k+1} = y_k + \omega(Bx_{k+1} - g). \end{cases} \tag{7}$$

Here $\omega > 0$ is a chosen relaxation parameter. The saddle point system can be written as a split matrix

$$\mathcal{A} = \mathcal{P} - \mathcal{Q},$$

where

$$\mathcal{P} = \begin{bmatrix} A & 0 \\ B & \frac{-1}{\omega}I \end{bmatrix}, \; \mathcal{Q} = \begin{bmatrix} 0 & -B^T \\ 0 & \frac{-1}{\omega}I \end{bmatrix}, \text{ and } u_k = \begin{bmatrix} x_k \\ y_k \end{bmatrix},$$

and the iteration can be expressed as

$$\mathcal{P}u_{k+1} = \mathcal{Q}u_k + b,$$

or

$$u_{k+1} = \mathcal{P}^{-1}\mathcal{Q}u_k + \mathcal{P}^{-1}b.$$

To get a sense of how to chose the relaxation parameter ω, note that, using the first equation of (7) to solve the second, we have

$$y_{k+1} = y_k + \omega(BA^{-1}f - g - BA^{-1}B^T y_k).$$

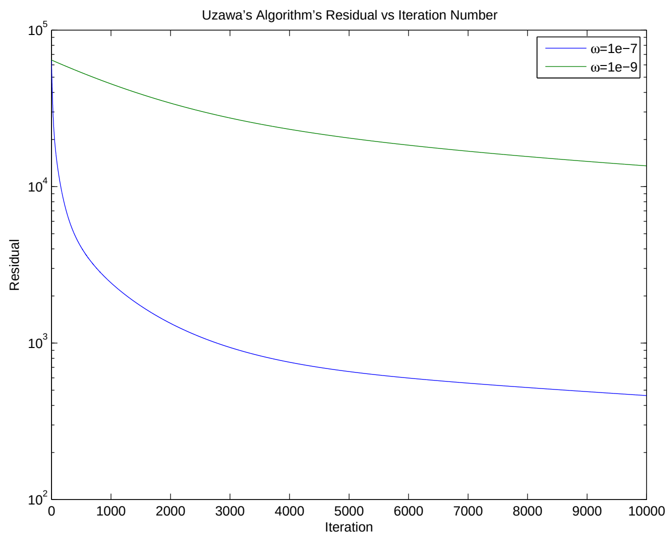


Figure 6: Uzawa Algorithm's residual vs iteration

The above equation shows that Uzawa's method is equivalent to a stationary Richardson iteration applied to a given Schur complement system, which has good characteristics if other properties hold. Particularly, convergence is guaranteed for $0 < \omega < \frac{2}{\lambda_{\max}}$, where $\lambda_{\max}$ is the largest eigenvalue of the (symmetric and positive definite) matrix $BA^{-1}B^T$. Also, the parameter that gives the optimal convergence rate, can be expressed through the extreme eigenvalues as

$$\omega_* = \frac{2}{\lambda_{\max} + \lambda_{\min}}.$$

In some cases, one can get good estimates for those eigenvalues. In others, one needs to experiment with the parameter ω to obtain better convergence. Figure 6 shows the effect of that parameter in the convergence rate of Uzawa's Algorithm. For our matrices, the straightforward application of the algorithm was not competitive with the previously described iterative methods. One can, however, look for ways to speed up the algorithm by preconditioning the matrix A (the only matrix that we need to invert in an Uzawa iteration), or by reducing the number of iterations (through obtaining a better estimate of the optimal parameter ω_*).

3.2 Augmented Lagrangian Method

We can extend Uzawa's Algorithm by including an extra term

$$L(x, y, \gamma) = \frac{1}{2}\langle Ax, x\rangle - \langle f, x\rangle + \langle y, Bx - g\rangle + \frac{\gamma}{2}\|Bx - g\|_2^2,$$

where $\gamma > 0$ is a positive penalty (regularization) parameter. The critical point for this new Lagrangian can be approximated by using an alternating strategy of minimizing the function with respect to each component iteratively

$$x_{k+1} = \arg\min_x L(x, y_k, \gamma)$$

and

$$y_{k+1} = \arg\min_y L(x_{k+1}, y, \gamma).$$

This produces the following iterative scheme

$$\begin{cases} (A + \gamma B^T B)x_{k+1} & = f - B^T y_k \\ y_{k+1} & = y_k + \gamma(Bx_{k+1} - g) \end{cases}.$$

Clearly, $A + \gamma B^T B$ is a symmetric (and, in our case, positive-definite) matrix. When the first equation of the above matrix system has been solved (for example, by a preconditioned conjugate gradient method), the term y_{k+1} can be computed directly from the second equation.

The convergence of the matrix system is guaranteed for suitably chosen $\gamma > 0$, and the convergence is fast if γ is chosen to be very large [8]. On the other hand, when γ gets larger the matrix $A + \gamma B^T B$ becomes more and more ill-conditioned and, therefore, harder to invert. Thus, once again, the parameter in our stationary iteration method needs to be carefully chosen.

We ran the Augmented Lagrangian method using *minres* as a subroutine to help solve the system for u. As seen in Figure 7, the residual drops quite quickly, but then stagnates. The problem was found in the *minres* configuration - the inner tolerance of 10^{-10} could not be achieved within the specified number of maximum iterations. We already know from the past that *minres* can achieve small tolerances quickly, yet here it failed. This goes to show the ill-conditioning coming from the addition of the penalty term can compromise the numerical accuracy. It is promising that we have not applied preconditioners at all. Using preconditioners along with the Augmented Lagrangian method could definitely robustify the algorithm, although we never managed to test this due to lack of time.

4 Multilevel Methods

Multigrid algorithms are applied to a wide range of problems, primarily to solve linear and nonlinear boundary value problems. Much has been written on the topic of the multigrid method of solving systems (cf. [3],[4]). In this section we give a very brief overview of the basic components of the multigrid strategy and show preliminary results of applying a multigrid method to solve the saddle point system (3). We also discuss a preconditioner, originally developed for finite element discretizations of the homogeneous Poisson problem, and its possible applications to our systems.

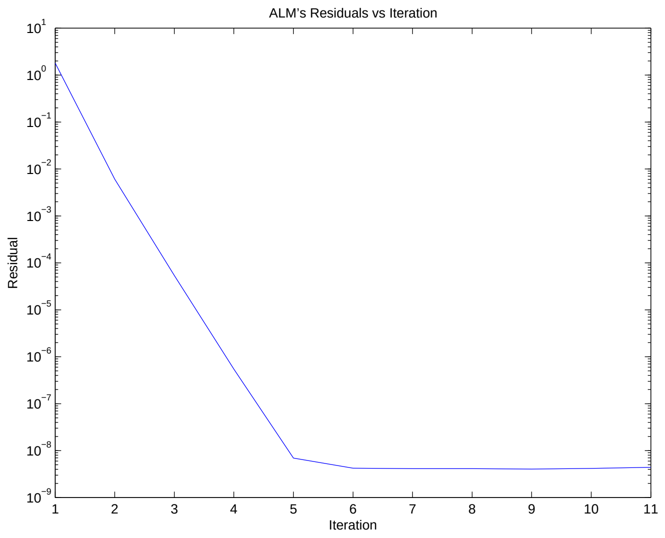


Figure 7: Augmented Lagrangian method residual

4.1 Brief overview of the multigrid algorithm

We should recall that the saddle-point systems we use for our experiments come from a mixed-finite element discretization of a Poisson problem on a series of triangular meshes. The size of our matrix depends on the number of elements (triangles) in a given mesh. The finer a mesh is (i.e. the more triangles it has) the bigger, and harder to solve, is our system.

To explain the basic idea of the multigrid algorithm, we shall first describe the simple two-level setup. Consider two meshes - a coarse one, where the triangles are of size H, and a fine one with triangles of size $h < H$.

We need a way of mapping functions on the coarse gird (vectors in the smaller dimensional space) to functions on the fine grid (vectors in the larger dimensional) space. We will call this operator the *prolongation* $I_{H\to h}$. The corresponding *restriction* operator (mapping functions on the finer grid to the coarse one) is then given by $I_{h\to H} = I_{H\to h}^T$. Figure 4.1 shows a graphical representation of the prolongation operator for our mixed finite element pair, by depicting how given basis functions on the coarser grid are represented through (several) basis functions on the finer one.

The two-level multigrid algorithm is now quite simple. We want to solve the system $\mathcal{A}_h x = b_h$ on the finer level. To do that we start with an initial guess x_0 and find a new approximation $x_{\frac{1}{3}}$ of the solution x by doing several iterations of a simple iterative scheme. This first step in our algorithm is called *Pre-Smoothing*.

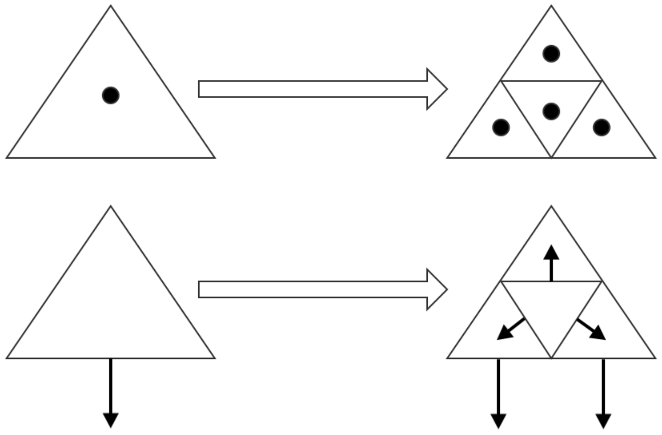


Figure 8: Graphical representation of the prolongation operator for P0/RT0 mixed finite elements

The error in this new approximation will satisfy the residual equation

$$\mathcal{A}_h e_h = \mathcal{A}_h(x - x_{\frac{1}{3}}) = b_h - \mathcal{A}_h x_{\frac{1}{3}} = r_h.$$

We attempt to solve this equation on the coarse gird (where the matrix $\mathcal{A}_H$ has smaller dimension). That is to say we solve the equation

$$\mathcal{A}_H e_H = r_H, \tag{8}$$

where the coarse grid residual is given by $r_H = I_{h\to H} r_h$. Mapping the error back to the fine grid and correcting, we obtain our new approximation

$$x_{\frac{2}{3}} = x_{\frac{1}{3}} + I_{H\to h} e_H.$$

This second step in the algorithm is called the *Coarse Grid Correction*.

Finally, in the *Post-Smoothing* step, we perform several more iterations of our simple iteration scheme for the fine grid equation (starting from our current approximation $x_{\frac{2}{3}}$) to obtain the new iterate x_1.

If more than two levels of refinement are given, then the coarse grid correction step in the two level algorithm above can be replaced by a nested multigrid to solve equation (8), leading to the standard multigrid *V-cycle* algorithm.

We were able to implement the necessary restriction and prolongation operators for a set of nested meshes of different size, and we tested the multigrid using various levels, yielding the results in Figure 9. Notice that the residuals plummet for a just a few iterations. The rate of convergence then decreases,

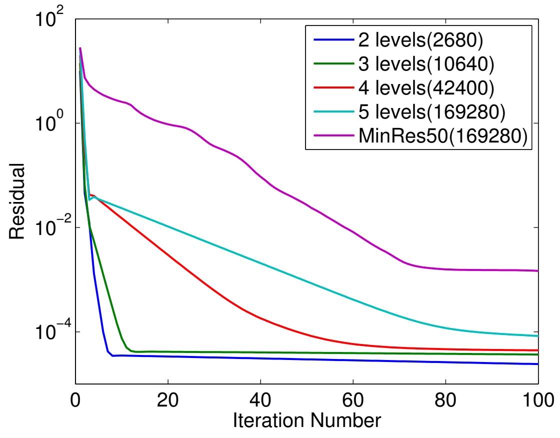


Figure 9: Multigrid results. For comparison, we plot every 50th residual of *minres*

although we believe that more appropriate smoother choices would remedy that. To further demonstrate the efficiency of the multigrid algorithm, we have also plotted, for our biggest case of approximately 1.7 million unknowns, the performance of the *minres* method. To ensure a fair-comparison in terms of the computational cost, we have taken 50 *minres* iterations for each multigrid iteration (i.e. one iteration of the plotted "*MinRes50* method" consists of 50 regular *minres* iterations). We can see from the decrease in the residual that the multigrid algorithm significantly outperforms the *minres* method.

4.2 The BPX Preconditioner

Another multilevel preconditioner we wanted to test is the Bramble-Pasciak-Xu (BPX) preconditioner. On a sequence of nested triangulations $\mathcal{T}_1, \cdots, \mathcal{T}_J$, the BPX preconditioner is defined as

$$\mathcal{B}_{BPX}^{-1} = \sum_{j=0}^{J} I_j I_J^T,$$

where I_j is the prolongation matrix that represents the degrees of freedom associated with the triangulation $\mathcal{T}_j$ in terms of the degrees of freedom associated with the finest triangulation mesh J. Since, while implementing the miltigrid algorithm described in the previous section, we had already built the grid transfer operators $I_{j\to(j+1)}$ for two consecutive refinements, it was really easy to obtain the matrices I_j and therefore to compute $\mathcal{B}_{BPX}^{-1}$. Indeed,

$$I_j = I_{(J-1)\to J)} I_{(J-2)\to(J-1)} \cdots I_{j\to(j+1)}.$$

	No BPX	with BPX
iterations	622	289
run time	0.1978 secs.	0.239 secs.
residual	10^{-4}	10^{-4}

Table 4: *minres* results using BPX

	No BPX	with BPX
iterations	9575	1701
run time	408 secs.	76 secs.
residual	10^{-6}	10^{-6}

Table 5: *gmres* results using BPX

Note that an advantage of the BPX preconditioner over the preconditioners discussed in Section 2 is that, since we already have an explicit formula for $\mathcal{B}_{BPX}^{-1}$, there is no matrix inversion in the application of the BPX preconditioner.

Due to our time constraints we were unable to test the performance of this preconditioner thoroughly and compare it with the other proposed solution strategies. We do, however, include in this report the results we obtained using this preconditioner on one of our smaller systems using *minres* and *gmres*. The results are given in Tables 4 and 5 respectively. Note that in both cases the number of iterations needed for convergence decreased. Additionally, in the case of *gmres* the use of preconditioner also substantially decreased the computational cost.

5 Summary

In this report we described our experiments and learnings regarding numerically solving large saddle-point systems. Such systems can become increasingly difficult to solve when the dimension grows too large, as is the case when the systems come from the numerical discretization of certain PDEs of interest. While these systems' sparsity suggests the use of iterative solvers, their poor conditioning and unfavorable spectral properties result in very slow convergence rates.

In Section 2 we examined the application of a wide variety of iterative methods to our set of test systems, concluding that the best performing among them were *minres* (where applicable) and *gmres*. The drawbacks of these methods were that *minres* was only applicable to symmetric systems, and *gmres*, while robust, had a very poor convergence rate. We investigated preconditioning of such methods, and demonstrated that a good preconditioner can drastically reduce the number of iterations required to achieve a small residual.

In Section 3 we looked at two of the most popular stationary iteration methods, designed specifically for saddle-point systems. Our finding was that, when properly tuned, both the Uzawa method and the Augmented Lagrangian method were able to compete with the out-of-the-box iterative methods. The drawback, however, was that case-specific parameter tuning was required to obtain the best performance for these methods. Due to lack of time, we were not able to test preconditioners for those methods and to compare their performance to the performance of the preconditioned iterative solvers.

Finally, since we were interested specifically in systems coming from numerical discretization of PDEs, in Section 4 we considered the application of multilevel algorithms. We were able to implement a basic geometric multigrid algorithm and show that it outperformed the benchmark set by the *minres* iterative method. This was a very promising result, especially since there were a lot of areas where our multigrid algorithm could potentially be improved. We were not able to investigate these potential improvements, due to the lack of time, but we believe this would be a fruitful avenue for future work.

References

[1] M. Benzi, G.H. Golub, J. Liesen, Numerical Solution of Saddle Point Problems, Acta Numerica (14), pp. 1-137.

[2] Y. Saad, Iterative Methods for Sparse Linear Systems, SIAM, ISBN 978-0898715347.

[3] J. H. Bramble (1993). Multigrid Methods: Pitman Research Notes and Mathematics Series. Longman Scientific and Technical, New York, ISSN 0269-3674.

[4] W.L. Briggs, V. E. Henson and S.F. McCormick (2000). A Multigrid Tutorial: Second Edition, SIAM, ISBN 978-0-898-714-62-3.

[5] W. Hackbusch (2003). Multi-Grid Methods and Applications: Volume 4 of Springer Series in Computational Mathematics, Springer Berlin Heidelberg.

[6] S. C. Brenner and R. Scott (2008).The Mathematical Theory of Finite Element Methods, Springer Science and Business Media, ISBN 978-0-387-75933-3, pp 183.

[7] M. Fortin (1989). Some iterative methods for incompressible flow problems, Comput. Phys. Comm. 53, pp. 393-399.

[8] M. Fortin and R. Pierre (1992). Stability analysis of discrete generalized Stokes problems, Numer. Meth. Partial Differ. Eq. 8, pp. 303-323.

[9] M. Fortin and R. Glowinski (1983). Augmented Lagrangian Methods: Applications to the Numerical Solution of Boundary-Value Problems, North Holland, Amsterdam.

[10] R. Glowinski, Q.V. Dinh and J. Periaux (1983).Domain decomposition methods for nonlinear problems in fluid dynamics, Comput. Meth. Appl. Mech. Eng. 40, pp. 27 - 109.