
by

Chetan Jhurani and L. Demkowicz

Part III: Algorithms for Moore-Penrose Pseudoinverse

Chetan Jhurani  L. Demkowicz
Institute for Computational Engineering and Sciences
The University of Texas at Austin
Austin, Texas 78712

Abstract
This paper is the third and last in a series to develop a numerical homogenization method for heterogeneous media and integrate it with goal-oriented finite element mesh adaptivity. The first paper developed the mathematical formulation of local homogenization. This was done using an appropriate averaging of the Moore-Penrose pseudoinverse of the element stiffness matrix. In the second one, we presented numerical verification of this approach and other numerical results. In the current paper, we present four algorithms to compute Moore-Penrose pseudoinverse that also exploit the sparsity. These are based on $QR$ factorization, $a$-priori knowledge of the null-space, a regularization based characterization, and lastly an iterative algorithm based on proper splittings of matrices. We analyze some of these algorithms in detail for homogenizing hexahedral elements in the molecular base model of densification in Step and Flash Imprint Lithography.

Contents
1 Introduction 2
2 Moore-Penrose Pseudoinverse 2

3 Fast Algorithms for Moore-Penrose Pseudoinverse 3
   3.1 Pseudoinverses and homogenization 4
   3.2 Algorithms for Moore-Penrose pseudoinverse of sparse matrices 5
      3.2.1 Pseudoinverse using Tikhonov regularization 5
      3.2.2 Pseudoinverse using a known null-space basis 7
      3.2.3 Pseudoinverse using $QR$ factorization 7
      3.2.4 Pseudoinverse using proper splittings 8
1 Introduction

In the first paper of this series, we presented mathematical formulation of a local numerical homogenization procedure to dimensionally reduce and approximately solve large nonlinear optimization problems posed on finite element meshes. This approach toward homogenization requires computation of Moore-Penrose pseudoinverse of matrices. The pseudoinverse is required for matrices with different structural properties. The most important case is of sparse, symmetric, singular finite-element stiffness matrices local to an element.

For computational efficiency of our homogenization method, we require fast and possibly approximate algorithms for computing Moore-Penrose pseudoinverse of such matrices. We present multiple algorithms for computing them. They all have quite different features and none is the best for all situations.

Moore-Penrose pseudoinverse is the most widely known special case of generalized inverses or pseudoinverses. Since we work with Moore-Penrose pseudoinverse exclusively, occasionally we use the term pseudoinverse for it.

2 Moore-Penrose Pseudoinverse

Moore-Penrose pseudoinverse was discovered by E. H. Moore in 1906 in the context of projections associated with singular and rectangular matrices [12]. It was rediscovered by Roger Penrose in 1955 as the unique matrix satisfying four algebraic matrix equations [15]. Quite a few articles and books have appeared on the subject since then. There is a large bibliography in [14] and [4].

For an arbitrary dense matrix, Singular Value Decomposition (SVD) is the most reliable algorithm to compute the pseudoinverse (or its action on a vector). It is important to determine (or already know) the numerical rank of a matrix. This is because of the unusual perturbation properties of the pseudoinverse [11]. For this reason, general purpose linear algebra packages like MATLAB® use the SVD to compute the pseudoinverse. But the cost
of computing the SVD is high. It is $O(N^3)$ where $N$ is the matrix size (assuming a square matrix). However, SVD is useful for determination of the numerical rank, which is important to reliably compute pseudoinverse of arbitrary matrices.

As is common in the field of computational mathematics, much better algorithms can be developed if one has additional information about the problem input, or if one needs only an approximate solution, or if one has to repeatedly compute something with moderate changes in input data. All these cases are present when computing pseudoinverses in the context of numerical homogenization.

Since pseudoinverses are generalizations of inverses of square nonsingular matrices, their computation usually involves solving linear equations or obtaining matrix factorizations. Analogous to the plethora of direct and iterative algorithms for solution of linear systems of equations, there is no single algorithm that is the best choice for computing generalized inverses. There is a variety of algorithms that can be used [4]. Moore-Penrose pseudoinverse can be computed using full-rank factorization. The pseudoinverse of the original matrix can be computed using pseudoinverses of the factors. There are other direct algorithms too, for example Greville’s method discussed in [4].

3 Fast Algorithms for Moore-Penrose Pseudoinverse

Local homogenization is the core part of the global solution method and is done in each element. Moreover, because of mesh refinement and nonlinearities, each element can possibly have different stiffness matrix after each step, and it may be required to upscale it again. Thus, critical to the computational efficiency of the whole method is fast and approximate computation of the pseudoinverse of a sparse element stiffness matrix. For these reasons, Singular Value Decomposition (SVD) is not the best choice here for computing pseudoinverses.

We present algorithms that are significantly faster for sparse matrices. The first one uses the characterization of the pseudoinverse as the limit of a Tikhonov regularized matrix [4, 16]. In the second algorithm, the a priori knowledge of the null-space of a matrix can be used to compute the pseudoinverse by direct or iterative linear solvers [10]. The third algorithm uses a sparse rank-revealing $QR$ factorization of the matrix (or of a suitable column permutation) to compute an exact pseudoinverse using the matrix factors $Q$ and $R$ [6, 9]. Lastly, in the context of mesh refinement or nonlinear problems, one might reuse an old pseudoinverse (actually its sparse factorized form) to compute pseudoinverse of a perturbed matrix. This can be done using an iterative procedure based on “proper splittings” [5, 13].

None of these algorithms is the perfect one for all situations. Depending on the problem dimension, number of vector unknowns per DOF, element size, condition numbers, and sparsity, one algorithm may be much better than the others. These choices are analyzed in
3.1 Pseudoinverses and homogenization

Our focus is on computing Moore-Penrose pseudoinverses of the following three categories of matrices.

1. A real, singular (hence square), sparse, symmetric matrix with almost all eigenvalues being positive. The rank-deficiency is known in advance and is small (typically between 1 and 6) and is independent of the matrix size. The matrix size is variable, typical value being between 500 and 3000. These are element stiffness matrices coming from a finite element discretization.

2. A real, dense, rectangular, full column-rank matrix with a small number of columns, between 2 and 24, and a variable number of rows, typical value being between 500 and 3000. These are interpolation matrices in a finite element discretization.

3. A real, singular (hence square), dense, symmetric matrix. The matrix size is small and will be between 2 and 24. These are pseudoinverses of homogenized fine-scale matrices.

For the second category (the dense, rectangular, full column-rank matrices) the pseudoinverse can be expressed and computed using the normal equations.

\[ A^\dagger = (A^T A)^{-1} A^T \]

Since the number of columns of \( A \) is fixed and small and \( A \) is well-conditioned, we can either use the normal equations for computing \( A^\dagger \) or preferably use QR factorization of \( A \) to avoid loss of precision due to squaring of the condition number. For the third category (the small, dense, symmetric, singular matrices), we use the SVD. These are implemented in LAPACK [1].

The computational bottleneck of the homogenization process is computing the pseudoinverses of the matrices of the first category (real, large, sparse, singular matrices). SVD of a matrix of size 400 takes nearly a second on current single core processor (as of year 2009). However, a sparse Cholesky factorization of matrices of similar size and with a structure and properties of the homogenization problem takes around 0.005 seconds, which is 200 times faster. If the matrix size is greater, say 2500, the sparse factorization can be 3000 times faster than computing the SVD. Thus, if we can use sparse factorization or sparse solution methods for linear systems instead of SVD for pseudoinverse, we can hope to obtain a speedup of 100 to 1000 depending on the matrix size.

Note that when we talk about computing pseudoinverse, in general we do not want it in an explicit matrix form but want only its action on a set of vectors. However, for the
categories of matrices mentioned above, we do want the pseudoinverse of the dense matrices explicitly. This is all right, since these are small matrices.

### 3.2 Algorithms for Moore-Penrose pseudoinverse of sparse matrices

As mentioned above, using a sparse algorithm for computing pseudoinverse may be 2 or 3 orders of magnitude faster than the SVD for typical sparse finite element matrices. We present three algorithms that exploit the sparsity. In the end, we present an iterative algorithm that can compute pseudoinverse of a perturbed matrix using the factorized form of the pseudoinverse of the original matrix. This is useful when the element stiffness matrices change slightly during mesh adaptivity steps and Newton step for nonlinearity.

#### 3.2.1 Pseudoinverse using Tikhonov regularization

We can avoid the SVD by using a limit characterization for the pseudoinverse using Tikhonov regularization [4, 16]. For an arbitrary matrix $K$

$$K^\dagger = \lim_{\delta \to 0} (K^T K + \delta I)^{-1} K^T = \lim_{\delta \to 0} K^T (K K^T + \delta I)^{-1}. \tag{1}$$

$I$ is a size-compatible identity matrix. The limit always exists. The limit has to be taken for the full expression and not just the matrix with the variable $\delta (\delta I)$ or $((K K^T + \delta I)^{-1}$ or $(K^T K + \delta I))$. The proof is in [4, 16].

We use a finite $\delta$ in Equation (1) to approximate $K^\dagger$. For an accurate approximation, $\delta$ should be a small multiple of the $\sigma^2_r$, the square of the smallest non-zero singular value of $K$. We can then use Equation (1) to compute an approximate action of $K^\dagger$ on any given vector $g$. An a priori error estimate for $\delta > 0$ can be proved [4, 7]

$$\| (K^T K + \delta I)^{-1} K^T - K^\dagger \|_2 \leq \delta \| K^\dagger \|_2^3.$$

We use the sparse direct Cholesky factorization method as implemented in CHOLMOD [8]. If $K$ is sparse or banded then $KK^T$ and $K^T K$ are sparse too (although less so). For example, if $K$ is tridiagonal, then $KK^T$ and $K^T K$ are pentadiagonal. Moreover, they are positive semi-definite. Addition of a positive diagonal makes the matrix to be inverted positive definite. Thus, Cholesky factorization is a feasible algorithm for all $K$.

It is not necessary to use a general sparse factorization method for the matrices we are interested in. If a typical nodal ordering is used to order particles in a 2-D or 3-D box, the stiffness matrices are banded. For $n^d$ particles and $p$ DOFs per particle, where $d$ is the space
dimension and \( n \) is the “edge-size” of an element, the matrix size is \( pn^d \) and bandwidth is \( O(pn^{d-1}) \). Thus banded Cholesky solvers can also be competitive with general sparse Cholesky solvers. Currently, we have used reordering based sparse direct solvers. A brief comparison of the two methods is done in Section 4.3.

Because \( \delta \) is finite, we lose a few digits in the computed vector \( K^\dagger g \). If \( \delta \) is chosen appropriately, the approximation is not detrimental to the overall algorithm. It should not be too small (relative to \( \sigma^2 \)) either since it would lead to numerical singularity of \( K^TK + \delta I \). A large \( \delta \) would lead to less accuracy. We do need an estimate of \( \sigma^2 \) for this problem, which is an \( O(N^3) \) procedure for an accurate estimate, but that is all right since it can be done once and re-used for other elements.

This algorithm squares the (pseudo-)condition number by computing \( K^TK \) (or \( KK^T \)), and then adds a small diagonal perturbation to make the matrix invertible. Hence, we do not use an iterative algorithm to invert. One can use the QR factorization to compute the pseudoinverse for this damped least squares problem [6] and avoid squaring of the condition number, but it would be slower than sparse Cholesky factorization. And if one goes through the trouble of implementing sparse QR factorization, one might as well compute the pseudoinverse “exactly” using the algorithm presented in Section 3.2.3. The advantage of sparse Cholesky factorization is its speed. For certain solution regimes, this is the fastest of all the algorithms we will discuss. It is inherently an approximate algorithm and will never obtain solutions with full accuracy. This is not a big drawback because in the context of homogenization with mesh adaptivity and nonlinearity, we need only an approximation of the upscaled Hessian matrix and not its exact upscaling.

The pseudoinverse can also be represented by the sum of a series [3].

\[
K^\dagger = \sum_{i=1}^{\infty} K^T (KK^T + I)^{-i}
\]  

(2)

Here \( K^T \) must not be removed as a factor from the series (else the sum will diverge). Using the property \( (\alpha K)^\dagger = \frac{1}{\alpha} K^\dagger \) for \( \alpha \neq 0 \), it can be derived from Equation (2) that

\[
K^\dagger = \sum_{i=1}^{\infty} K^T \delta^{i-1} (KK^T + \delta I)^{-i} \forall \delta > 0.
\]

Here \( \delta = \frac{1}{\alpha^2} \). Since \( \alpha \neq 0 \) is arbitrary, the representation is valid for all \( \delta > 0 \). Note that the first term in the summation is the last expression in Equation (1) for a finite \( \delta \).

This representation effectively rescales the matrix \( K \) so that computing the inverse of \( KK^T + \delta I \) is not too badly-conditioned when \( K \) is rank-deficient (and \( \delta \) is chosen appropriately). A large \( \delta \) would mean that the inverse can be accurately computed, but then more terms in the series will be needed to accurately approximate \( K^\dagger \). If a smaller \( \delta \) is chosen, then fewer terms will be needed to achieve an acceptable accuracy, but the inverse will
not be accurately computed in finite precision. Thus, there is a built-in trade-off between computation speed and high accuracy.

Once $KK^T + \delta I$ is factorized using a sparse Cholesky factorization, the factors can be reused to compute the actions of $(KK^T + \delta I)^{-i}$ for $i \geq 1$. Thus, the “inversion” step is required only once and rest of the terms can computed cheaply.

We have not used this iterative improvement strategy in the context of homogenization, but it can be used when the pseudoinverse is required with higher accuracy.

3.2.2 Pseudoinverse using a known null-space basis

A less well-known result, presented in [10], is that computing pseudoinverse, or its action on a vector, can be transformed to solving linear algebraic equations of same size (by either direct or iterative algorithms). One needs the null space of the original matrix a priori. For many mathematical models of physical problems the null-space is known from physical arguments.

For nonlinear lattice elasticity problems, the null-space of the stiffness matrix contains the rigid body translations whether the lattice is in equilibrium or not. If it is in equilibrium, rigid body rotations are also in the null-space [10]. Thus, the dimension of the null-space is independent of the size of the lattice, and its basis is known.

Let $R \in \mathbb{R}^{N \times P}$, $0 < P < N$, be a matrix of columns-vectors that form an orthonormal basis of the null-space of a symmetric $K \in \mathbb{R}^{N \times N}$. $R$ stands for rigid body motion. Thus, $P := I_n - RR^T$ is the orthogonal projector on the range of $K$. It is shown in [10] that $P(K + RR^T)^{-1} = (K + RR^T)^{-1}P$ is the Moore-Penrose pseudoinverse of $K$.

Hence, if $K$ and $R$ are known, then computing $u = K^\dagger g$, for $f \in \mathbb{R}^N$, implies solving $(K + RR^T)u = Pg$ for $u$. Even if $K$ is sparse, $(K + RR^T)$ would be dense in general. But it is a rank $p$ update of a sparse matrix and $p$ is small compared to $N$. Thus the action of $(K + RR^T)$ on any vector can be computed with a little more effort than sparse matrix vector multiplication. Thus the system can be efficiently solved by iterative solvers for symmetric matrices. We use the iterative solvers (conjugate gradient and minimum residual methods) and preconditioners (Jacobi and Incomplete Cholesky) implemented in PETSc [2]. A numerical comparison of these 4 combinations is presented in Chapter 4.

3.2.3 Pseudoinverse using QR factorization

Let $K \in \mathbb{R}^{N \times N}$ have a rank-deficiency $p$, where $0 < p < n$. In our application, $p$ will be a fixed number much smaller than $n$. We can compute a rank-revealing $QR$ factorization
$K = QR$, where $Q$ is orthonormal and $R$ is upper triangular. Partition $Q$ and $R$ as follows.

\[ Q = [Q_1 \quad Q_2]. \]

and

\[ R = \begin{bmatrix} R_{11} & R_{12} \\ 0 & 0 \end{bmatrix}. \]

Here $Q_1$ is $n \times (n - p)$, $Q_2$ is $n \times p$, $R_{11}$ is $(n - p) \times (n - p)$, and $R_{12}$ is $(n - p) \times p$.

Define auxiliary matrices $S$ and $X$.

\[ S := R_{11}^{-1}R_{12} \]

and

\[ X := (S^T S + I)^{-1}S^T R_{11}^{-1}Q_1^T. \]

$S$ is $(n - p) \times p$, and $X$ is $p \times n$. The action of $R_{11}^{-1}$ can be computed by back-substitutions since $R_{11}$ is a triangular matrix. In addition, the ‘inversion’ of a $p \times p$ system $(S^T S + I)$ is needed to compute $S$ and $X$.

Finally, using the $QR$ factorization and the matrices $S$ and $X$, it can be shown [6]

\[ K^\dagger = \begin{bmatrix} R_{11}^{-1}(Q_1^T R_{12} - R_{12}X) \\ X \end{bmatrix}. \]

Note that this is just an expression and need not be computed explicitly. It can be used to compute $K^\dagger f$ for a known vector $f$. One should form $Q_1^T f$ first to compute $X$ and $K^\dagger f$.

In general the factors $Q$ and $R$ are not as sparse as the original matrix $K$. Moreover, a typical $QR$ factorization algorithm will not exploit the sparsity of $K$. For this, a sparse $QR$ factorization algorithm should be used, for example as implemented in [9].

In sparse factorization, the columns of $K$ are reordered before the $QR$ factorization to reduce fill-in. Thus, we need to form the $Q$ and $R$ factors of $KP$, where $P$ is a permutation matrix, usually chosen automatically [9]. If $KP = QR$, where $P$ is orthonormal, then it can be shown using SVD that $K^\dagger = P(QR)^\dagger$. This expression is applicable for a permutation matrix $P$ also because permutations are orthonormal. $(QR)^\dagger$ can be computed as shown above. The expressions remain identical but the implementation can take advantage of the sparsity of $Q$ and $R$.

### 3.2.4 Pseudoinverse using proper splittings

In the context of homogenization for mesh-adaptivity and nonlinearities, each of the algorithms to compute Moore-Penrose pseudoinverse presented above recomputes the pseudoin-
verse of the element stiffness matrix whenever needed. However, in some cases, material
nonlinearity might not be strong enough to change the element stiffness matrix appreciably
when the mesh is refined away from the element of interest. Hence, once pseudoinverse is
computed in an earlier step, its value may be useful in the new step if we do not care for an
exact upscaling. In a few steps, one could update the pseudoinverse to avoid slower conver-
genence in Newton steps (if observed). With this motivation, we can use an algorithm presented
in [5, 13] to update the pseudoinverse of a perturbed matrix by reusing the factorized form
of the old pseudoinverse.

Analogous to additive splittings for solving nonsingular system of equations, one can use
a proper splitting for computing pseudoinverses. A splitting $K = G - H$ of a matrix $K$ is
called proper if the range and null spaces of $K$ and $G$ are equal. Using such a splitting, the
iterative algorithm

$$u^{(n+1)} = G^\dagger Hu^{(n)} + G^\dagger f$$

converges to $u = K^\dagger f$ as $n \to \infty$ iff the spectral norm of $G^\dagger H$ is less than one.

This iteration can be useful if we know $G^\dagger$, the old pseudoinverse, in factorized form using
Cholesky factorization or $QR$ factorization discussed above. $H$ is (negative of) the update
of the element stiffness matrix due to a new solution guess. One must make sure that the
range and null space of $K$ and $G$ are identical. In many cases this can be readily done using
physical arguments. In addition, if the update is small, only a few iterations will be needed
to converge to $u$.

Despite its appeal, this algorithm requires an independent algorithm that computes the
factorization of the pseudoinverse. Moreover, it is memory-intensive because of extra storage
that will be useful for just a few steps. Finally, to ensure convergence a priori one needs an
estimate of the spectral norm of $G^\dagger H$ or abort the computation if norm of $u^{(n)}$ is diverging to
infinity. We have not pursued this approach further in this research. We mention it because
using this method may lead to faster computation at the expense of more memory usage.

4 Computational time of pseudoinverse algorithms and
optimum element sizes

Based on the time spent in computing pseudoinverse, we compute an optimum element size
for homogenization of SFIL lattice. Optimum size means one that leads to fastest global
homogenization without taking loss of accuracy (due to large elements being homogenized)
into consideration. Some of the algorithms for Moore-Penrose pseudoinverse discussed in
Section 3 are analyzed for computational time as a function of space dimension, element
size, iterative method, and preconditioner. We compare the computational times of the
sparse Cholesky factorization method for approximate pseudoinverses with the exact $QR$
factorization method. Based on these results, we have created a feature matrix for different
pseudoinverse methods that could be useful for other applications.

4.1 Computational time for different element sizes

A typical mesh in the sequence of meshes generated for automatic mesh adaptivity will have elements of all sizes. For allocating different elements to different processors it is necessary to know in advance the time taken to homogenize elements as a function of element size $n$ (number of SFIL particles in each element side). We compute homogenization time for hexahedral and quadrilateral elements of polynomial degree 1 and identical number of cells in each side.

The most expensive step is computing the pseudoinverse of the element stiffness matrix. In Section 3.2 we presented a few algorithms for computing Moore-Penrose pseudoinverses. Here, we compute pseudoinverses using two kinds of algorithms — sparse Cholesky factorization of (Section 3.2.1) and iterative methods based on known null space (Section 3.2.2).

For the iterative case, we choose the conjugate gradient (CG) and minimum residual (MINRES) methods. As preconditioners, we use Jacobi preconditioner and Incomplete Cholesky (ICC). MINRES is also tried because stiffness matrices in SFIL can be indefinite in a particular iteration. These are implemented in PETSc [2]. The sparse Cholesky factorization is implemented in CHOLMOD [8]. Thus, we analyze four different iterative schemes and one direct scheme. The trends in homogenization times presented below can be taken as trends in times taken to compute pseudoinverse (of the element stiffness matrix) using these 5 different schemes.

Figures 1 and Figure 2 show the results for 2-D and 3-D respectively. The computation time increases as a power law. For $n$ as the element size, we compute the best-fit homogenization time as a power law $10^A n^B$. The best-fit constants $A$ and $B$ are in Figure 3. We mention a few observations

For small elements, sparse Cholesky is better than iterative schemes. In 2-D, it is much better, and never loses to the iterative schemes. In 3-D, for small elements sparse Cholesky is better but not by much. Iterative schemes are better beyond $n = 8$. This might be due to higher fill-in in 3-D.

Between the four iterative schemes, depending on element size $n$, the best iterative scheme keeps on changing, but they are all very close together. For small $n$, Jacobi preconditioner is better than ICC for both 2-D and 3-D even though it leads to slower convergence (uses more iterations, not shown here). ICC is more complicated to setup so it does not help much compared to Jacobi unless $n$ increases and the number of iterations taken increase a lot with Jacobi.

10
4.2 Optimum element size for homogenization of a cubical lattice

Given a cubical geometry with a fixed number of points \( p \) on each side, we might want to divide the lattice into \( m \) elements on each side. If there are \( n \) points in each element, we have a relation between \( p, m, \) and \( n \).

\[
p = m n - m + 1 \quad \text{or equivalently} \quad m = \frac{p - 1}{n - 1}
\]

Figure 4 shows a 2-D cartoon of the fine and coarse lattices. We want to decide \( n \) to minimize the computational cost of homogenizing the whole cubical lattice into a coarse mesh.

Let \( H(n, p) \) be the total time to compute the homogenized Newton step for the full lattice. It consists of \( m^d \) homogenizations and 1 linear solve using the homogenized Hessian.

\[
H(n, p) = m^d 10^A n^B + \frac{10^A'}{d 2^d} (m + 1)^{B'}
\]
Figure 2: The figure shows computation time to homogenize a 3-D hexahedral element with 24 DOFs (3 per corner) as the element size changes. The element size is number of particles on each side of the cube. Different pseudoinverse methods lead to different times.

$A, B$ are best-fit constants related to homogenization of a single element as described in Section 4.1. $A', B'$ are constants for inverting the global homogenized Hessian. They would be different in general. In the analysis ahead, we have chosen $A' = A$ and $B' = B$ so that someone implementing a code can reuse a single scheme for all purposes instead of implementing multiple schemes together.

Substituting $m = \frac{p-1}{n-1}$, we get

$$H(n, p) = \left(\frac{p - 1}{n - 1}\right)^d 10^A n^B + \frac{10^{A'}}{d 2^d} \left(\frac{p - 1}{n - 1} + 1\right)^{B'}$$

The objective is to choose an $n$ that minimizes $H(n, p)$ for a given $p$. If $n$ is 2, we did not homogenize an SFIL lattice really since $m = p - 1$. Then, the first term is ignored since it does not require any work. We will spend more time in solving the full system as well as doing more Newton iterations. If $n = p$, it means the lattice is homogenized into a single element. Although the global system will be very small, this will require linear solves for pseudoinverse of a huge matrix. Additionally, the accuracy will be very poor.
Space dimension = 2  

<table>
<thead>
<tr>
<th>Method</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>CG ICC</td>
<td>-4.95</td>
<td>2.73</td>
</tr>
<tr>
<td>CG Jacobi</td>
<td>-4.82</td>
<td>2.72</td>
</tr>
<tr>
<td>Minres ICC</td>
<td>-4.96</td>
<td>2.77</td>
</tr>
<tr>
<td>Minres Jacobi</td>
<td>-4.90</td>
<td>2.82</td>
</tr>
<tr>
<td>Cholesky Sparse Direct</td>
<td>-5.09</td>
<td>2.63</td>
</tr>
</tbody>
</table>

Space dimension = 3  

<table>
<thead>
<tr>
<th>Method</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>CG ICC</td>
<td>-4.16</td>
<td>4.09</td>
</tr>
<tr>
<td>CG Jacobi</td>
<td>-4.68</td>
<td>4.50</td>
</tr>
<tr>
<td>Minres ICC</td>
<td>-4.19</td>
<td>4.14</td>
</tr>
<tr>
<td>Minres Jacobi</td>
<td>-4.78</td>
<td>4.69</td>
</tr>
<tr>
<td>Cholesky Sparse Direct</td>
<td>-5.69</td>
<td>5.91</td>
</tr>
</tbody>
</table>

Figure 3: The best-fit power law constants $A$ and $B$ for homogenization time in 2-D (Figure 1) and 3-D (Figure 2). The computation time is modeled by $T(n) = 10^{-A}n^B$, where $n$ is the element size.

We have observed that the optimum element size $n$ depends weakly on changes in $p$. We will work with $p = 101$. Note that in practice we may choose to not homogenize every element and instead reuse the homogenized Hessian from other equal sized elements. We may also reuse the homogenized Hessian for multiple Newton steps. This analysis is for a single homogenized Newton step in which all elements are homogenized separately without reusing data from other elements.

Figures 5 and Figure 6 show the plots for the computation time for a 2-D and 3-D lattice.
Figure 5: The time taken to homogenize a 2-D lattice of size $101 \times 101$ once as a function of the coarse element size. Cholesky factorization to compute pseudoinverse is the fastest for all element sizes. Compare Figure 6.

Figure 6: The time taken to homogenize a 3-D lattice of size $101 \times 101 \times 101$ once as a function of the coarse element size. Cholesky factorization to compute pseudoinverse is the fastest for small element sizes only. Compare Figure 5.

respectively. Each lattice had 101 points on each side. The results are plotted for different methods to compute the pseudoinverses of stiffness matrices. As discussed in Section 4.1, we have analyzed 5 schemes. CG means conjugate gradient, MR means minimum residual, ICC means Incomplete Cholesky preconditioner, JAC means Jacobi preconditioner, and DIRECT means sparse Cholesky factorization.

As we can see, in 3-D, it takes around 4 minutes to homogenize if element size is 4 using sparse Cholesky factorization. For larger 3-D elements, iterative schemes are faster than the sparse direct method. For 2-D, the sparse direct method is always faster. The main reason why a sparse direct solver came out as the best is due to multiple right hand sides. In 3-D,
with 3 DOFs per node, 24 (or 25 depending on if we use a known load) linear systems are solved. For the given data, we see that the best idea is to choose \( n = 5 \) (4 cells on each side), and use a sparse direct solver for local homogenization. Since \( n - 1 \) is a power of 2, it will be beneficial for mesh adaptivity and refinements. Thus, if initial mesh elements are too large, it is better to break them using the optimum element size as a guideline.

### 4.3 A comparison of Cholesky factorization and QR factorization for computing pseudoinverse

We discussed sparse direct factorization based algorithms for computing the Moore-Penrose pseudoinverse in Sections 3.2.1 and 3.2.3. The Cholesky factorization algorithm was inherently approximate unlike the QR factorization based algorithm, which was exact. The details of these algorithms are discussed in [8, 9]. We use the stiffness matrix \( K \) of a 3-D box with 13 particles on each side (total 2197 particles), and each particle connected to the 26 nearest neighbors. Each particle has 1 DOF. In the ordering, starting from origin, the \( x \) direction index is the fastest and \( z \) direction index is the slowest.

![Stiffness matrix K](image1)

![K^T K + \delta I](image2)

Figure 7: (a) The non-zero structure of a stiffness matrix \( K \) of a 3-D box with 13 particles on each side (total 2197 particles), and each particle connected to the 26 nearest neighbors. Each particle has 1 DOF. (b) The non-zero structure of \( K^T K + \delta I \).

Figures 7(a) and (b) show the stiffness matrix and the matrix \( K^T K + \delta I \) discussed in Section 3.2.1 used for Tikhonov regularization. Figures 8(a) and (b) show the permutation matrix \( P \) and the upper triangular factor \( R \) such that \( P^T(K^T K + \delta I)P = R^T R \). Figure 9(a) shows the upper triangular factor \( R \) when no permutation is used \((P = I)\). The factor has the same bandwidth as the matrix \((K^T K + \delta I)\) and is dense within the band.

Figure 9(b) show the permutation (produced by COLAMD [9]) for a fill-in reducing
sparse QR factorization of $K$. Figures 10(a) and (b) show the sparse orthogonal factor $Q$ and the sparse upper triangular factor $R$ such that $KP = QR$.

Figure 8: (a) The non-zero structure of the permutation matrix $P$ that reduces fill-in for Cholesky factorization of $K^T K + \delta I$. (b) The non-zero structure of the upper triangular Cholesky factor $R$ of $K^T K + \delta I$ such that $P^T (K^T K + \delta I) P = R^T R$.

Figure 9: (a) The non-zero structure of the upper triangular Cholesky factor $R$ of $K^T K + \delta I$ such that $K^T K + \delta I = R^T R$. No permutation matrix is used. (b) The non-zero structure of the permutation matrix $P$ for a sparse QR decomposition of $K$ such that $KP = QR$. See Figure 10.
Figure 10: (a) The non-zero structure of the orthogonal matrix $Q$ for a sparse $QR$ decomposition of $KP$. See Figure 9(b) for $P$. (b) The non-zero structure of the upper triangular matrix $R$ where $KP = QR$. 
On a 1.5 GHz single core processor, the sparse Cholesky factorization with reordering took 0.235 seconds. The banded Cholesky factorization took 0.238 seconds. The difference is insignificant. However, the upper triangular factor had 16% more non-zeroes for the banded factor. This would make back-substitution slightly expensive.

The sparse $QR$ factorization took 1.196 seconds, nearly 5 times slower than Cholesky factorization. In addition, the number of non-zeroes in $Q$ and $R$ combined was 4 times the number of non-zeroes in the Cholesky factor $R$.

### 4.4 Features of sparse algorithms for pseudoinverse

Finally, based on the results presented here and in Sections 4.1 and 4.3, we have created a feature classification matrix for the various pseudoinverse algorithms. This is presented in Figure 11. At least for the initial iterations, the approximate pseudoinverse method using Tikhonov regularization presented in Section 3.2.1 may be the best because of its speed.

We have not integrated sparse $QR$ factorization with homogenization yet. The features and classifications are based on analytical formulas and preliminary code testing using [9].

<table>
<thead>
<tr>
<th>Criterion and its importance (to us)</th>
<th>Regularize + Cholesky</th>
<th>Sparse QR</th>
<th>Null space based</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fast (although this is element dependent)</td>
<td>Fast</td>
<td>Good</td>
<td>Depends (If multiple RHS)</td>
</tr>
<tr>
<td>Multiple RHS</td>
<td>Factorize once</td>
<td>Factorize once</td>
<td>Multiple solves</td>
</tr>
<tr>
<td>Software complexity</td>
<td>Medium</td>
<td>High</td>
<td>Medium</td>
</tr>
<tr>
<td>Exact (up to floating point)</td>
<td>No</td>
<td>Yes</td>
<td>Depends (iterative)</td>
</tr>
<tr>
<td>Prior information needed</td>
<td>Smallest singular value magnitude</td>
<td>No</td>
<td>Null space</td>
</tr>
<tr>
<td>Conditioning</td>
<td>Bad (little better if Sparse QR used)</td>
<td>Good</td>
<td>Good</td>
</tr>
</tbody>
</table>

![Figure 11: Features of algorithms for computing Moore-Penrose pseudoinverse discussed in Section 3.2. None of the algorithms is the best amongst all depending on the kind of problem to be homogenized.](image-url)
These results are for a specific SFIL lattices, for a single step in the Newton iterations, specific libraries (PETSc, CHOLMOD), and a typical recent CPU (Intel Core 2 Duo, 1.5 GHz). However, for any changes in the configuration, the time analysis can be done automatically for a few typical elements and the best method can be chosen accordingly at run-time.

Acknowledgements

This work was supported by the Department of Energy under Grant No. DE-FG02-05ER25701. The authors would also like to thank Dr. J. Tinsley Oden, Dr. C. Grant Willson, Dr. Jon Bass, Dr. Serge Prudhomme, Dr. Paul T. Bauman, and Dr. Elizabeth Collister for many helpful discussions.

References


