Proceedings of the Project Review, Geo-Mathematical Imaging Group (Purdue University, West Lafayette IN), Vol. 1 (2013) pp. 157-169.

SUPERFAST ALGORITHM FOR COMPUTING ARBITRARY ENTRIES OF THE INVERSE OF A SPARSE MATRIX WITH APPLICATION TO THE HESSIAN IN TIME-HARMONIC FWI

XIAO LIU*, JIANLIN XIA†, YUANZHE XI‡, AND MAARTEN V. DE HOOP§

Abstract. We extend a structured selected inversion method to the extraction of any arbitrary entry of the inverse of a large sparse matrix A. For various discretized PDEs such as Helmholtz equations, a structured factorization yields a sequence of data-sparse factors of about $O(n)$ nonzero entries, where n is the matrix size. We are then able to extract any arbitrary entry of A^{-1} in about $O(\log^2 n)$ flops for both two and three dimensions. On the contrary, even the latest developments are either too expensive or have severe difficulties in this. Our method also uses ULV factorizations instead of explicit hierarchically semiseparable inversions to enhance the stability and accuracy. We fully take advantage of the structures (e.g., common nested bases in the blocks) to achieve the high efficiency. The method can have a substantial impact on Gauss-Newton iterations, where preliminary studies of the Hessian matrices are made. Numerical tests indicate significant advantages over exact inversions.

1. Introduction.

The computation of a subset of a large sparse matrix inverse is critical in many scientific computing and engineering problems. Applications include risk estimation in uncertainty quantification [3], the evaluation of retarded and less-than Green's functions in quantum nanodevice design [5], computing a rational approximation for Fermi-Dirac functions in the Density Functional Theory [24], and Gauss-Newton iteration in the time-harmonic full waveform inversion (FWI) [8, 18].

Suppose A is a given sparse matrix of order n. Several algorithms have been proposed to find the diagonal entries of A^{-1}. For direct methods, one of the earliest methods is proposed in [21]. It forms two equations to reveal the relation between the LDU factorization of A and its inverse. Later, the methods in [11, 12, 14, 15] are all derived from the nested dissection ordering [9] and share two common stages: factorizing the matrix in a bottom-up ordering of the separators, and then computing the entries of the inverse in a top-down traversal of the separators. The methods in [22, 23, 24] combine classical iterative solvers with sparse approximation techniques and solve this problem in an iterative way.

The work for finding the off-diagonal entries of A^{-1} is rarely done. The methods based on traditional linear system solutions [1] are suitable when a few entries are desired. If the number of required entries is large, those methods are inefficient and the worst case is equivalent to solving n linear systems. The direct methods designed for diagonal inverse entries also calculate certain off-diagonal entries during the computation process, but those entries only correspond to the non-zero pattern of the factors. It is not easy to extend them to computing an entry of A^{-1} at an arbitrary position. The inverse multifrontal method in [4] generalizes the method in [21] and provides a theoretical framework for identifying the dependence of any inverse entry in the factors. Similar to the linear system solution problem, the main drawback of direct methods is the issue of fill-in [9]. For 3D problems, they need $O(n^2)$ [28, 30] flops to factorize A and the resulting factors have about $O(n^{4/3})$ non-zero entries, which is too expensive. On the other hand, the iterative methods have economic storage but suffer the convergence issue for indefinite matrices if good preconditioners are lacking.

Recently, the structured inversion algorithm proposed in [32] provides a new approach to this problem. After the structured factorization [28, 30] with hierarchically semiseparable (HSS) structures [2, 31], the factors have only about $O(n)$ non-zero entries. Due to such data sparsity, the structured inversion algorithm can find all the diagonal entries in about $O(n)$ flops, whereas other existing inversion algorithms need about $O(n^{3/2})$ and $O(n^2)$ for 2D and 3D problems, respectively.

*Department of Mathematics, Purdue University (liu867@math.purdue.edu)
†Department of Mathematics, Purdue University (xiaj@math.purdue.edu)
‡Department of Mathematics, Purdue University (yxi@math.purdue.edu)
§Department of Mathematics, Purdue University (mdehoop@purdue.edu)

In this paper, we generalize the method in [32] to the computation of arbitrary entries of A^{-1}, and apply it to computing the Hessian matrix needed in FWI. The main contributions are mentioned as follows.

1. **Fast and stable HSS inversion scheme**. Explicit HSS inversion algorithm often relies on the recursive application of the Sherman-Morrison-Woodbury (SMW) formula, which may not be numerically stable. That is, the accumulated errors can lead to an inaccurate HSS inverse. Here, we propose a new fast and stable HSS inversion algorithm to replace the explicit one, which depends on the backward stable HSS ULV-type factorization scheme and resolves the stability issue. The algorithm traverses the HSS tree from the root to the leaf level and has the complexity of about $O(N)$, where N is the size of the HSS matrix.
2. **Arbitrary structured randomized inversion**. We use the randomized structured multifrontal method [29] to factorize the original matrix, which is more efficient and flexible than the classical one used in [32]. Moreover, the factors from randomized factorization scheme have additional internal structures, which can be utilized to reduce the computational complexity and enhance the stability significantly for our structured inversion algorithm. We show that the computation of any (i, j)th entry in the inverse matrix essentially corresponds a traversal of the path connecting these two nodes on the elimination tree. This means that the actual computation only involves the factors associated with the nodes along this path. In general, our new algorithm requires about $O(\log^2 n)$ flops to compute each inverse entry for both 2D and 3D problems.
3. **Hessian matrix**. Direct computation of the Hessian matrix in the Gauss-Newton iteration for FWI includes one explicit inversion of a Helmholtz matrix and several matrix-matrix multiplications, which is too expensive and has never been done before. We show that this problem can be converted into the other one, where the Hessian matrix equals the selected inverse of a large sparse matrix. Thus, the proposed structured inversion algorithm can be exploited to solve this problem effectively.

The remaining section of the paper is organized as follows. Section 2 presents the HSS inversion method based on ULV factorizations. The randomized structured multifrontal LDL factorization is reviewed in Section 3. Section 4 provides an arbitrary inversion algorithm to compute any entry of the inverse. Section 5 proposes a way to compute the diagonal and arbitrary entry of the Gauss-Newton Hessian. Section 6 gives some experimental results about the arbitrary inversion and the Gauss-Newton Hessian.

2. HSS LDL factorization and inversion.

2.1. HSS LDL factorization.

The HSS LDL factorization is developed in [33] for symmetric matrices with rank structures. Here we only summarize the main steps of this algorithm, for detailed explanation and illustration, the reader is referred to [33].

Here, the inputs are the HSS generators [2] D, U, B, R, and the outputs are the factors $\hat{Q}, \hat{L}, \hat{D}$.

- If i is a leaf, first compute a QL factorization of the off-diagonal basis

$$U_i \to \hat{Q}_i \begin{pmatrix} 0 \\ \bar{U}_i \end{pmatrix}.$$

Then update the diagonal block by the same orthogonal matrix and compute a partial LDL factorization.

$$\hat{Q}_i^T D_i \hat{Q}_i \to \begin{pmatrix} \hat{L}_{i;1,1} & \\ \hat{L}_{i;2,1} & I \end{pmatrix} \begin{pmatrix} \hat{D}_i & \\ & S_i \end{pmatrix} \begin{pmatrix} \hat{L}_{i;1,1}^T & \hat{L}_{i;2,1}^T \\ & I \end{pmatrix}.$$

- If i has children c_1, c_2, merge children node's reduced basis and compute a QL factorization

$$\begin{pmatrix} \bar{U}_{c_1} R_{c_1} \\ \bar{U}_{c_2} R_{c_2} \end{pmatrix} \to \hat{Q}_i \begin{pmatrix} 0 \\ \bar{U}_i \end{pmatrix}.$$

Then form the new diagonal block by merging children's information and perform a similar partial LDL factorization.

$$\hat{Q}_i^T \begin{pmatrix} S_{c_1} & \bar{U}_{c_1} B_{c_1} \bar{U}_{c_2}^T \\ \bar{U}_{c_2} B_{c_1}^T \bar{U}_{c_1}^T & S_{c_2} \end{pmatrix} \hat{Q}_i \to \begin{pmatrix} \hat{L}_{i;1,1} & \\ \hat{L}_{i;2,1} & I \end{pmatrix} \begin{pmatrix} \hat{D}_i & \\ & S_i \end{pmatrix} \begin{pmatrix} \hat{L}_{i;1,1}^T & \hat{L}_{i;2,1}^T \\ & I \end{pmatrix}$$

Note that if i reaches the root, just compute a full LDL factorization of the form $\hat{L}_i \hat{D}_i \hat{L}_i^T$.

2.2. Fast and stable HSS inversion. The HSS LDL inversion algorithm perform a top-down traversal of the HSS tree. The basic idea of HSS LDL is to invert each factor and multiply them back.

With the inputs $\hat{Q}, \hat{L}, \hat{D}$, the outputs are the generators of the inverse: $\tilde{D}, \tilde{U}, \tilde{B}, \tilde{R}$.

1. If i is the root node, compute the inverse of the reduced block from LDL factor,

$$\bar{D}_i = \hat{L}_i^{-T} \hat{D}_i^{-1} \hat{L}_i^{-1}.$$

Then split it into 4 blocks and pass the submatrices to the two children

$$\bar{D}_i \to \begin{pmatrix} T_{c_1} & \tilde{B}_{c_1} \\ \tilde{B}_{c_1}^T & T_{c_2} \end{pmatrix}.$$

2. If i is not the root, compute the inverse from the L and Q factor

$$\bar{D}_i = \hat{Q}_i \begin{pmatrix} \hat{L}_{i;1,1}^T & \hat{L}_{i;2,1}^T \\ & I \end{pmatrix}^{-1} \begin{pmatrix} \hat{D}_i^{-1} & \\ & T_i \end{pmatrix} \begin{pmatrix} \hat{L}_{i;1,1} & \\ \hat{L}_{i;2,1} & I \end{pmatrix}^{-1} \hat{Q}_i^T$$

The same split is performed to $\bar{D}_i$. In order to get the R generator, compute

$$\bar{U}_i = \hat{Q}_i \begin{pmatrix} \hat{L}_{i;1,1}^T & \hat{L}_{i;2,1}^T \\ & I \end{pmatrix}^{-1} \begin{pmatrix} 0 \\ I \end{pmatrix}$$

Then split $\bar{U}_i$

$$\bar{U}_i \to \begin{pmatrix} \tilde{R}_{c_1} \\ \tilde{R}_{c_2} \end{pmatrix}.$$

Note that if i is a leaf node there's no need to split the matrix, just let $\tilde{D}_i = \bar{D}_i, \tilde{U}_i = \bar{U}_i$.

3. Structured multifrontal block LDL factorization. We briefly review the structured multifrontal LDL factorization in [32] based on the multifrontal method [6, 16] and a structured variation [28], and mention a randomized version. To reduce the propagation of the nonzero elements in Gaussian elimination, we first reorder the original matrix by nested dissection [9]. As a result, the index set of the original matrix is partitioned into groups of indices called supernodes, and the possible positions of nonzeros in the L factor is acquired by a symbolic factorization.

The block Gaussian elimination on supernodes follows a tree structure called the *elimination tree* $\mathcal{T}$ [6]. The key idea of the multifrontal method is that the elimination at each step doesn't involve all the indices of all the groups. Let $\mathcal{N}_{\mathbf{i}}$ be the set of nodes that is involved in the row operations at node $\mathbf{i}$. Then the factorization can be performed following the order of $\mathcal{N}_{\mathbf{i}}$. The parent node par($\mathbf{i}$) is defined as the first element in $\mathcal{N}_{\mathbf{i}}$. For more detailed study within supernodes, let $t_{\mathbf{i}}$ be $\mathbf{i}$'s index set in the original matrix, $t_{\mathbf{j},\mathbf{i}}$ be the subset of $t_{\mathbf{j}}$ that will be updated by node $\mathbf{i}$. Therefore $\tilde{t}_{\mathbf{i}} = \bigcup_{\mathbf{j} \in \mathcal{N}_{\mathbf{i}}} t_{\mathbf{j},\mathbf{i}}$ is the set of rows involved in the factorization at $\mathbf{i}$. With these notations, the nonzero pattern in the factorization can be summarized in the next proposition. Figure 1 illustrates such a relationship in the matrix.

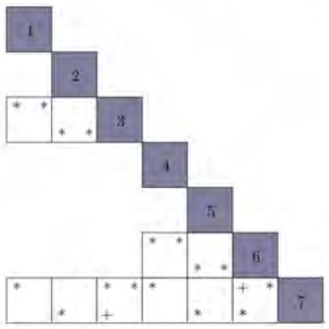

FIG. 1. *Nonzero structure after nested dissection*

PROPOSITION 3.1. *Theorem 3.1 and 3.2 in [16]* $\mathcal{N}_{\mathbf{i}}$ *is a subset of* $\mathbf{i}$*'s ancestors. Moreover if* $\mathbf{p} = \mathrm{par}(\mathbf{i})$*, then it follows that*

$$\begin{aligned}&\mathcal{N}_{\mathbf{i}} \setminus \{\mathbf{p}\} \subset \mathcal{N}_{\mathbf{p}},\\ &\qquad t_{\mathbf{j},\mathbf{i}} \subset t_{\mathbf{j},\mathbf{p}}, \forall \mathbf{j} \in \mathcal{N}_{\mathbf{i}} \setminus \{\mathbf{p}\}.\end{aligned}$$

The multifrontal method at each node only works on related submatrices and therefore transforms global sparse matrix operations into local dense operations.

The Gaussian elimination is performed locally following two steps: first eliminate the off-diagonal block of the current node, then send a low-rank update matrix to the parent node. For the leaf node, the frontal matrix $\mathbf{F}_{\mathbf{i}}$ is taken from the corresponding nonzero block columns and rows of the original matrix.

$$\mathbf{F}_{\mathbf{i}} = \mathbf{F}_{\mathbf{i}}^0 = \begin{pmatrix} A|_{t_{\mathbf{i}} \times t_{\mathbf{i}}} & (A|_{\tilde{t}_{\mathbf{i}} \times t_{\mathbf{i}}})^T \\ A|_{\tilde{t}_{\mathbf{i}} \times t_{\mathbf{i}}} & 0 \end{pmatrix}$$

For the nonleaf nodes, the frontal matrix is a combination of the original blocks and update matrices representing the influence of Gaussian elimination at children nodes.

$$\mathbf{F}_{\mathbf{i}} = \mathbf{F}_{\mathbf{i}}^0 \updownarrow\!\!\!\!\leftrightarrow \mathbf{U}_{\mathbf{c}_1} \updownarrow\!\!\!\!\leftrightarrow \mathbf{U}_{\mathbf{c}_2}$$

With these notations, both the exact and the structured factorization can be written as

$$\mathbf{F}_{\mathbf{i}} = \begin{pmatrix} F_{\mathbf{i},\mathbf{i}} & (F_{\mathcal{N}_{\mathbf{i}},\mathbf{i}})^T \\ F_{\mathcal{N}_{\mathbf{i}},\mathbf{i}} & F_{\mathcal{N}_{\mathbf{i}},\mathcal{N}_{\mathbf{i}}} \end{pmatrix} = \begin{pmatrix} I & \\ L_{\mathcal{N}_{\mathbf{i}},\mathbf{i}} & I \end{pmatrix} \begin{pmatrix} F_{\mathbf{i},\mathbf{i}} & \\ & \mathbf{U}_{\mathbf{i}} \end{pmatrix} \begin{pmatrix} I & (L_{\mathcal{N}_{\mathbf{i}},\mathbf{i}})^T \\ & I \end{pmatrix},$$

$$\begin{aligned}L_{\mathcal{N}_{\mathbf{i}},\mathbf{i}} &= F_{\mathcal{N}_{\mathbf{i}},\mathbf{i}} F_{\mathbf{i},\mathbf{i}}^{-1},\\ \mathbf{U}_{\mathbf{i}} &= F_{\mathcal{N}_{\mathbf{i}},\mathcal{N}_{\mathbf{i}}} - F_{\mathcal{N}_{\mathbf{i}},\mathbf{i}} F_{\mathbf{i},\mathbf{i}}^{-1} (F_{\mathcal{N}_{\mathbf{i}},\mathbf{i}})^T.\end{aligned}$$

In many 2D and 3D discretized models, the frontal matrix has low-rank off-diagonal blocks. Hence a HSS compression on $\mathbf{F}_{\mathbf{i}}$ can significantly reduce the computation and storage cost. The HSS form of $\mathbf{F}_{\mathbf{i}}$ is demonstrated in Figure 2.

The fast way to compute update matrix $\mathbf{U}_{\mathbf{i}}$ has been established in [28, 33]. The computation of update matrix only needs the last step reduced matrix of HSS factorization. Note that $F_{\mathcal{N}_{\mathbf{i}},\mathbf{i}}$ is the top level off-diagonal block and has the low-rank form $F_{\mathcal{N}_{\mathbf{i}},\mathbf{i}} = U_{k+1} B_k^T U_k^T$.

$$\mathbf{U}_{\mathbf{i}} = F_{\mathcal{N}_{\mathbf{i}},\mathcal{N}_{\mathbf{i}}} - (U_{k+1} B_k^T U_k^T) F_{\mathbf{i},\mathbf{i}}^{-1} (U_k B_k U_{k+1}^T).$$

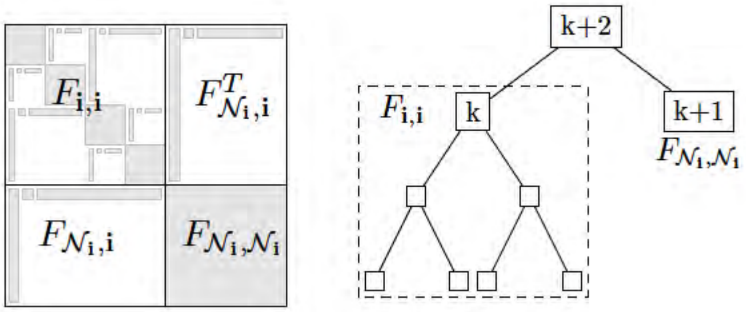
$F_{\mathbf{i},\mathbf{i}}$
$F^T_{\mathcal{N}_\mathbf{i},\mathbf{i}}$
$F_{\mathcal{N}_\mathbf{i},\mathbf{i}}$
$F_{\mathcal{N}_\mathbf{i},\mathcal{N}_\mathbf{i}}$
k+2
$F_{\mathbf{i},\mathbf{i}}$
k
k+1
$F_{\mathcal{N}_\mathbf{i},\mathcal{N}_\mathbf{i}}$

FIG. 2. *HSS compression of the frontal matrix*

$U_k^T F_{\mathbf{i},\mathbf{i}}^{-1} U_k$ can be reduced to the last step factors of $F_{\mathbf{i},\mathbf{i}}$. Similarly, since the computation of $L_{\mathcal{N}_\mathbf{i},\mathbf{i}}$ needs $U_k^T F_{\mathbf{i},\mathbf{i}}^{-1}$, which can also be reduced on one side.

$$L_{\mathcal{N}_\mathbf{i},\mathbf{i}} = U_{k+1} B_k^T (U_k^T F_{\mathbf{i},\mathbf{i}}^{-1}).$$

Algorithm 1 Structured multifrontal block LDL factorization based on [28, 32]

```
for node i from 1 to root do
    Form frontal matrix $\mathbf{F_i}$
    if node i is below switching level then                ▷ Exact factorization
        $L_{\mathcal{N}_\mathbf{i},\mathbf{i}} = F_{\mathcal{N}_\mathbf{i},\mathbf{i}} F_{\mathbf{i},\mathbf{i}}^{-1}$
        $\mathbf{U_i} = F_{\mathcal{N}_\mathbf{i},\mathcal{N}_\mathbf{i}} - L_{\mathcal{N}_\mathbf{i},\mathbf{i}} F^T_{\mathcal{N}_\mathbf{i},\mathbf{i}}$
    else                                                   ▷ Structured factorization
        Compute HSS compression of $\mathbf{F_i}$
        Perform partial HSS LDL factorization of $\mathbf{F_i}$
        Compute $L_{\mathcal{N}_\mathbf{i},\mathbf{i}}$, $\mathbf{U_i}$ using HSS LDL factors
    end if
end for
```

The scheme can be conveniently modified based on the randomized sparse factorization algorithm in [29]. That is, we avoid the dense update matrices, and instead, only matrix-vector products are passed along the assembly tree.

4. Sparse arbitrary inversion.

As a direct method, our inversion method is based on factorizations. If A is the symmetric sparse matrix, then all the information of A^{-1} is included in the factors of A. Assume we are only concerned with a few elements of A^{-1}, then the computation can be localized based on the sparsity of A and the elimination tree structure. From now on denote the inverse of A by

$$C = A^{-1}.$$

4.1. Diagonal of the inverse.

It has been established in [structured selected inversion] that the computation of the diagonal of C can be performed by a top-down traversal of the elimination tree. The submatrices involved in the computation of a diagonal block of C surprisingly match with the frontal matrix in the factorization.

$$C_{\mathbf{i},\mathbf{i}} = F_{\mathbf{i},\mathbf{i}}^{-1} + L^T_{\mathcal{N}_\mathbf{i},\mathbf{i}} C_{\mathcal{N}_\mathbf{i},\mathcal{N}_\mathbf{i}} L_{\mathcal{N}_\mathbf{i},\mathbf{i}} = F_{\mathbf{i},\mathbf{i}}^{-1} - L^T_{\mathcal{N}_\mathbf{i},\mathbf{i}} C_{\mathcal{N}_\mathbf{i},\mathbf{i}}.$$

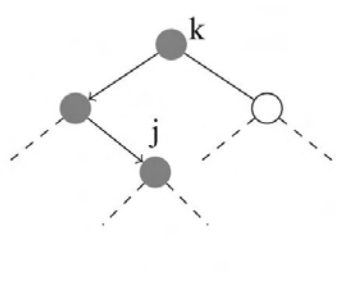

FIG. 3. *The path to get a diagonal block*

After the top-down traversal process, essentially we have computed $C_{\mathbf{i},\mathbf{i}}, C_{\mathcal{N}_\mathbf{i},\mathbf{i}}, C_{\mathcal{N}_\mathbf{i},\mathcal{N}_\mathbf{i}}$, i.e. the inverse corresponding to each frontal matrix. To see it clearly, write

$$\mathbf{C}_\mathbf{i} = \begin{pmatrix} C_{\mathbf{i},\mathbf{i}} & (C_{\mathcal{N}_\mathbf{i},\mathbf{i}})^T \\ C_{\mathcal{N}_\mathbf{i},\mathbf{i}} & C_{\mathcal{N}_\mathbf{i},\mathcal{N}_\mathbf{i}} \end{pmatrix}.$$

Furthermore, for a fixed node $\mathbf{j}$, $C_{\mathbf{j},\mathbf{j}}$ only depends on the path from root($\mathcal{T}$) to $\mathbf{j}$. The path is shown in Figure 3. The computation restricted to this path is summarized for future comparison with off-diagonal algorithm.

Algorithm 2 Compute $C_{\mathbf{j},\mathbf{j}}$

```
$\mathbf{k} = \text{root}(\mathcal{T})$
Locate the path $\mathbf{k} \to \mathbf{i}_1 \to \ldots \to \mathbf{i}_l = \mathbf{j}$
$C_{\mathbf{k},\mathbf{k}} = F_\mathbf{k}^{-1}$                                ▷ Direct inversion at the root node
for node $\mathbf{i}$ from $\mathbf{i}_1$ to $\mathbf{i}_l$ do
    Select $C_{\mathcal{N}_\mathbf{i},\mathcal{N}_\mathbf{i}}$ from the parent node
    $C_{\mathcal{N}_\mathbf{i},\mathbf{i}} = -C_{\mathcal{N}_\mathbf{i},\mathcal{N}_\mathbf{i}} L_{\mathcal{N}_\mathbf{i},\mathbf{i}}$          ▷ Inverse at the frontal matrix 2-1 block
    $C_{\mathbf{i},\mathbf{i}} = F_{\mathbf{i},\mathbf{i}}^{-1} - L_{\mathcal{N}_\mathbf{i},\mathbf{i}}^T C_{\mathcal{N}_\mathbf{i},\mathbf{i}}$      ▷ Inverse at the frontal matrix 1-1 block
end for
diagonal block
```

Note that $C_{\mathcal{N}_\mathbf{i},\mathbf{i}}$ are off-diagonal elements of the inverse. In the computation of the diagonal at node $\mathbf{j}$, we have already computed some off-diagonal elements along the path.

4.2. Off-diagonal of the inverse. Any off-diagonal block can be easily computed using the result from the diagonal blocks. An off-diagonal block is related to two different nodes in the elimination tree.

4.2.1. Examples.

1. A 3×3 example:

$$C = \begin{pmatrix} C_{11} & C_{12} & C_{13} \\ C_{21} & C_{22} & C_{23} \\ C_{31} & C_{32} & C_{33} \end{pmatrix} = A^{-1} = \begin{pmatrix} A_{11} & & A_{13} \\ & A_{22} & A_{12} \\ A_{31} & A_{32} & A_{33} \end{pmatrix}^{-1},$$

$$C_{32} = -A_{33}^{-1} L_{32} = -C_{33} L_{32},$$
$$C_{21} = L_{32}^T A_{33}^{-1} L_{31} = -C_{32}^T L_{31}.$$

C_{32} is the multiplication of a block from the parent and the local L factor. It has been partly computed already in the computation of the diagonal during $C_{\mathcal{N}_\mathbf{2},\mathcal{N}_\mathbf{2}} L_{\mathcal{N}_\mathbf{2},\mathbf{2}}$. Here the only difference is to get more rows. Same result holds for C_{31}.

The off-diagonal block C_{21} is the multiplication of a block from the inverse at node **2** and a block of the L factor at the sibling node **1**. The inverse part should be computed according to the nonzeros of the L factor at the sibling node.

2. A 7×7 example:

$$A = \begin{pmatrix} A_{11} & & A_{13} & & & & A_{17} \\ & A_{22} & A_{23} & & & & A_{27} \\ A_{31} & A_{32} & A_{33} & & & & A_{37} \\ & & & A_{44} & & A_{46} & A_{47} \\ & & & & A_{55} & A_{56} & A_{57} \\ & & & A_{64} & A_{65} & A_{66} & A_{67} \\ A_{71} & A_{72} & A_{73} & A_{74} & A_{75} & A_{76} & A_{77} \end{pmatrix}$$

The primary difference is the upper-level 3×3 off-diagonal blocks (the biggest blank). The result is written in Matlab notation:

C(4:6,1:3)=-C(7,4:6)'*[L(7,1)-L(7,3)*L(3,1),L(7,2)-L(7,3)*L(3,2),L(7,3)].

The last matrix suggests an interaction within the L factor. This is a top-down update from node **3** to the leaves.

4.2.2. General algorithm. There are two different situations for the two nodes involved in a given off-diagonal block.

1. One node is an ancestor of the other. Examples are C_{31} in the 3×3 case; C_{31}, C_{71}, C_{73} in the 7×7 case. In general, let **k** be an ancestor of **j**, the computation of $C_{\mathbf{k},\mathbf{j}}$ follows the top-down path from **k** to **j**, as is shown in Figure 4. See Algorithm 3.

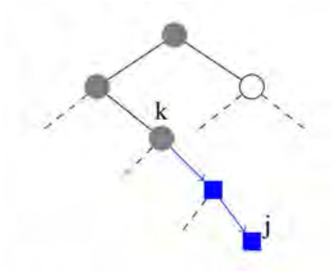

FIG. 4. *Off-diagonal block, the first case*

The complexity of this algorithm mainly depends on the length of the path and the size of each $\mathcal{N}_\mathbf{i}$ in the multiplication. If we compare this case with the computation of the diagonal, we can see that this is essentially computing more off-diagonal rows on a shorter path.

Algorithm 3 Compute $C_{\mathbf{k},\mathbf{j}}$, **k** is an ancestor of **j**

```
Locate the path $\mathbf{k} \to \mathbf{i}_1 \to ... \to \mathbf{i}_l = \mathbf{j}$
for node $\mathbf{i}$ from $\mathbf{i}_1$ to $\mathbf{i}_l$ do
    Select $C_{\mathbf{k},\mathcal{N}_\mathbf{i}}$ from the parent node
    $C_{\mathbf{k},\mathbf{i}} = -C_{\mathbf{k},\mathcal{N}_\mathbf{i}} L_{\mathcal{N}_\mathbf{i},\mathbf{i}}$        ▷ Compute the inverse on those rows
end for
```

2. The two nodes are not direct relatives. Examples are C_{21} in the 3×3 case; C_{21}, C_{41}, C_{43} in the 7×3 case. Let **j**, **k** be the nodes, **p** be the ancestor connecting them. The computation starts from **p** following two different paths to the two nodes. This is shown in Figure 5. See Algorithm 4.

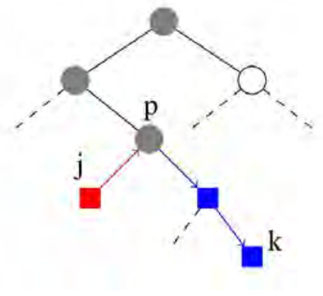

FIG. 5. *Off-diagonal block, the second case*

The complexity mainly depends on the length of two paths, the size of par($\mathbf{i}$) on the first path and the size of $\mathcal{N}_\mathbf{i}$ on the second path. This case is usually more expensive than the first case.

Algorithm 4 Compute $C_{\mathbf{k},\mathbf{j}}$, $\mathbf{j},\mathbf{k}$ connected by an ancestor $\mathbf{p}$

```
Locate the path $\mathbf{p} \to \mathbf{i}_1 \to ... \to \mathbf{i}_l = \mathbf{j}$
$\mathbf{q} = \mathbf{i}_1$
$\bar{L}_{\mathcal{N}_\mathbf{q},\mathbf{p}} = 0$
for node $\mathbf{i}$ from $\mathbf{i}_1$ to $\mathbf{i}_l$ do
    $\bar{L}_{\mathcal{N}_\mathbf{q},\mathbf{i}} = L_{\mathcal{N}_\mathbf{q},\mathbf{i}} - \bar{L}_{\mathcal{N}_\mathbf{q},\mathrm{par}(\mathbf{i})} L_{\mathrm{par}(\mathbf{i}),\mathbf{i}}$    ▷ process the information in the L factor
end for
Locate the path $\mathbf{p} \to \tilde{\mathbf{i}}_1 \to ... \to \tilde{\mathbf{i}}_{\tilde{l}} = \mathbf{k}$
for node $\mathbf{i}$ from $\tilde{\mathbf{i}}_1$ to $\tilde{\mathbf{i}}_{\tilde{l}}$ do    ▷ Compute the rows needed by the sibling
    Select $C_{\mathcal{N}_\mathbf{q},\mathcal{N}_\mathbf{i}}$ from the parent node
    $C_{\mathcal{N}_\mathbf{q},\mathbf{i}} = -C_{\mathcal{N}_\mathbf{q},\mathcal{N}_\mathbf{i}} L_{\mathcal{N}_\mathbf{i},\mathbf{i}}$    ▷ Compute the inverse on those rows
end for
$C_{\mathbf{k},\mathbf{j}} = C^T_{\mathcal{N}_\mathbf{q},\mathbf{k}} \bar{L}_{\mathcal{N}_\mathbf{q},\mathbf{j}}$    ▷ Get the final result
```

The cost of the method is shown as follows.

THEOREM 4.1. *Assume A is a discretized matrix from a 2D ($N \times N$) or 3D ($N \times N \times N$) regular mesh. Also assume the same rank properties as in [28, 32] holds. Then if the computation of an off-diagonal block of A^{-1} includes total k entries, the total cost for finding such k entries is about $O(m \log^2 n)$.*

5. Application in the Hessian of time-harmonic FWI.

5.1. Hessian in full waveform inversion.

In domain Ω, receivers are placed on the set $\Sigma_r \subset \Omega$. FWI seeks to minimize the misfit between the synthetic data and the real data on the receivers. Let q be the model parameter to be recovered in Ω. Define the forward mapping F_s from q to the wavefield restricted to the receivers.

$$F_s : q \to Ru_s$$

R is the restriction operator from Ω to Σ_r. u_s is the wavefield in Ω which solves the state equation with source term f_s

$$L(q)u_s = f_s. \tag{5.1}$$

Consider complex L^2 inner-product in this model, the objective functional takes the form

$$J(q) = \frac{1}{2}\sum_{s=1}^{N_s} \|F_s(q) - d_s\|^2.$$

d_s is the real data collected on Σ_r. Then the linearized forward operator ∇F_s is the so-called Born approximation.

In optimization theory, first order algorithms require gradient and second order algorithms make use of Hessian or approximate Hessian. The adjoint-state method plays a significant role in the computation of gradient and Hessian-vector product. [18] has derived the formulas. It is helpful to rewrite the results using our notation. The gradient is the Frechet derivative of $J(q)$

$$\nabla J(q)(\cdot) = \sum_{s=1}^{N_s} \mathrm{Re}\left(\lambda_s, \nabla L(q)(\cdot) u_s\right),$$

in which u_s, λ_s solve

$$\begin{aligned} L(q)u_s &= f_s \\ L(q)^*\lambda_s &= -R^*(Ru_s - d_s). \end{aligned}$$

For a fixed v, the Hessian-vector product is

$$H(v,\cdot) = \nabla^2 J(q)(v,\cdot) = \sum_{s=1}^{N_s} \left[\mathrm{Re}\left(\mu_s, \nabla L(q)(\cdot) u_s\right) + \mathrm{Re}\left(\lambda_s, \nabla L(q)(\cdot)\alpha_s + \nabla^2 L(q)(v,\cdot)\right)\right],$$

in which α_s, μ_s solve

$$\begin{aligned} L(q)\alpha_s &= -\nabla L(q)(v)u_s \\ L(q)^*\mu_s &= -R^*R\alpha_s - \nabla L(q)(v)^*\lambda_s. \end{aligned}$$

The Gauss-Newton Hessian ignores the second order derivative of F_s, in this case we have

$$\tilde{H}(q) = \sum_{s=1}^{N_s} \nabla F_s(q)^* \nabla F_s(q).$$

Note that the operator $\nabla F_s(q)^* \nabla F_s(q)$ is not only related to optimization theory, but also an important subject in linearized scattering theory. Therefore we are interested in computing it.

5.2. Computing approximate Hessian in Gauss-Newton iterations. Starting from this point we restrict ourselves to the isotropic Helmholtz equation. Similar techniques can be established on general state equation.

$$L(q)u = (-\Delta - q(x))u.$$

From $F_s(q) = Ru_s$, we have $\nabla F_s(q)(\delta q) = R\partial_q u_s(\delta q)$. $\partial_q u_s$ can be computed by taking derivative with respect to q on both sides of Helmholtz equation.

$$L\partial_q u_s(\delta q) + \nabla L(q)(\delta q)u_s = 0.$$

The derivative of Helmholtz operator is simply a multiplication by $-\delta q$, so we have

$$L\partial_q u_s(\delta q) = u_s \delta q$$

To solve the equation above consider finite difference discretization, it gives a simple discretization of the Helmholtz equation. Let $\mathbf{A}$ be the matrix representation of Helmholtz equation. Let $\mathbf{R}$ be the projection to the receiver, namely identity matrix on receiver points, zero elsewhere. Let $\mathbf{U}_s$

be the point-wise multiplication by u_s, i.e. a diagonal matrix generated by the entries of u_s. Then $\nabla F_s \approx \mathbf{R}\mathbf{A}^{-1}\mathbf{U}_s$.

$$\begin{aligned}\tilde{H}(q) &= \sum_{s=1}^{N_s} \nabla F_s{}^* \nabla F_s \\ &\approx \sum_{s=1}^{N_s} \mathbf{U}_s^* (\mathbf{A}^{-1})^* \mathbf{R}^* \mathbf{R} \mathbf{A}^{-1} \mathbf{U}_s \\ &= \sum_{s=1}^{N_s} \mathbf{U}_s^* (\mathbf{A}^*)^{-1} \mathbf{R} \mathbf{A}^{-1} \mathbf{U}_s\end{aligned}$$

Note that $\mathbf{U}_s$ are diagonal matrices and are already computed in the gradient. Selected entries of $\tilde{H}$ are directly related to the same entries in $\mathbf{M} = (\mathbf{A}^*)^{-1}\mathbf{R}\mathbf{A}^{-1}$.

The direct inversion and multiplication is too expensive. The key is to link $\mathbf{M}$ to the inverse of a sparse matrix. Since $\mathbf{R}$ is not full-rank, the sparse matrix $\mathbf{A}\mathbf{R}\mathbf{A}^*$ is not very helpful because it is neither invertible nor the pseudoinverse of $(\mathbf{A}^*)^{-1}\mathbf{R}\mathbf{A}^{-1}$. But consider a larger sparse matrix

$$\begin{pmatrix} \mathbf{A} & \\ -\mathbf{R} & \mathbf{A}^* \end{pmatrix} = \begin{pmatrix} \mathbf{A}^{-1} & \\ (\mathbf{A}^*)^{-1}\mathbf{R}\mathbf{A}^{-1} & (\mathbf{A}^*)^{-1} \end{pmatrix}^{-1},$$

therefore $\mathbf{M}$ is the lower-left block of the inverse of a double-sized sparse matrix.

The Helmholtz solver already generates the nested dissection ordering of the matrix $\mathbf{A}$. We can reuse that information to get a nested dissection ordering of the double-sized matrix. Suppose $\mathbf{1}, \mathbf{2}, ..., \mathbf{k}$ are the supernodes for $\mathbf{A}$, $\mathbf{1}', \mathbf{2}', ..., \mathbf{k}'$ are the supernodes for $\mathbf{A}^*$, then let $\mathbf{1} \cup \mathbf{1}', \mathbf{2} \cup \mathbf{2}', ..., \mathbf{k} \cup \mathbf{k}'$ be the supernodes for the double-sized matrix.

After this reordering, the diagonal of $\mathbf{M}$ will be permuted into the diagonal blocks of the double-sized matrix. Thus, the algorithm for computing the diagonal of the inverse is enough to compute the diagonal of the Hessian. If more entries are needed, then we can use the algorithm for arbitrary entries.

6. Numerical experiments.

Our code is written in Matlab and tested on a server with 32GB memory and Quad-Core AMD Opteron(tm) Processor 2380.

6.1. Arbitrary inversion.

Since our algorithm is based on paths in the elimination tree, different off-diagonal entries will have different cost. The worst case is the block related to the left-most leaf and the right-most leaf. They have the longest distance in the elimination tree.

Consider the model problem of solving Poisson's equation on a 2D regular grid, let the grid be $N \times N$ and the matrix A be $n \times n$, then $n = N^2$. The primary concern is the comparison between the inversion based on exact multifrontal method (`MF`) and our new structured algorithm (`NEW`). The results are shown in Table 6.1 and Figure 6.

Table 6.1
Flops and time (in seconds) for processing the L factor

$n\ (= N^2)$		128^2	256^2	512^2	1024^2
$\mathbf{l}_{\max}$		11	13	15	17
`Flops`	`MF`	$3.55e5$	$3.24e6$	$3.36e7$	$3.13e8$
	`NEW`	$3.40e5$	$9.33e5$	$3.67e6$	$1.60e7$
`Time(s)`	`MF`	0.0036	0.0061	0.0143	0.0495
	`NEW`	0.0077	0.0061	0.0105	0.0220

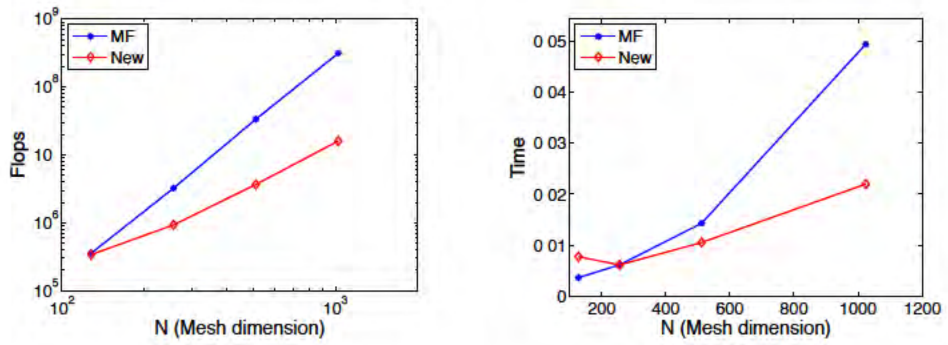

FIG. 6. *Flops and time (in seconds) for NEW and MF with different mesh size.*

6.2. Diagonal blocks of Gauss-Newton Hessian. We test our code for computing the diagonal of Gauss-Newton Hessian on a 2D 100×100 regular grid. The width of the PML layer is 10. The receivers are placed on the second row after PML layer. The diagonal blocks are computed and we plot the condition number of each diagonal block. See Figures 7 and 8.

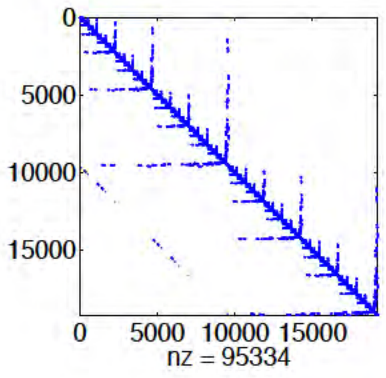

FIG. 7. *Nonzero pattern of the double-sized matrix.*

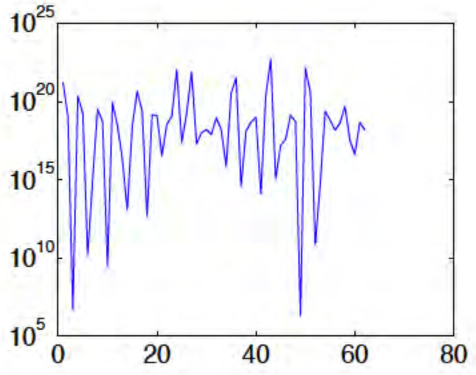

FIG. 8. *The condition number of each diagonal block of the Hessian.*

7. Acknowledgements. This research was supported in part by the members of GMIG at Purdue University, BGP, ExxonMobil, PGS, Statoil and Total. The research of Jianlin Xia was supported in part by NSF grants DMS-1115572 and CHE-0957024.

REFERENCES

[1] P. Amestoy, I. Duff, J. L'Excellent, Y. Robert, F. Rouet and B. Uçar, *On computing inverse entries of a sparse matrix in an out-of-core environment*, SIAM J. Sci. Comput., 34 (2012), pp. 1975–1999.

[2] S. Chandrasekaran, P. Dewilde, M. Gu, and T. Pals, *A fast ULV decomposition solver for hierarchically semiseparable representations*, SIAM J. Matrix Anal. Appl., 28 (2006), pp. 603–622.

[3] C. Bekas, A. Curioni, and I. Fedulova, *Low cost high performance uncertainty quantification*, in Proceedings of the 2nd Workshop on High Performance Computational Finance, ACM, (2009), p. 8.

[4] Y. E. Campbell and T. A. Davis, *Computing the sparse inverse subset: an inverse multifrontal approach*, Tech. Report TR-95-021, CIS Dept., Univ. of Florida, (1995).

[5] S. Cauley, J. Jain, C. K. Koh, and V. Balakrishnan, *A scalable distributed method for quantum-scale device simulation*, J. Appl. Phys., 101 (2007), p. 123715.

[6] I. S. Duff and J. K. Reid, *The multifrontal solution of indefinite sparse symmetric linear equations*, ACM Bans. Math. Software, 9 (1983), pp. 302–325.

[7] A. M. Erisman and W. F. Tinney,*On computing certain elements of the inverse of a sparse matrix*, Comm. ACM, 18 (1975), pp. 177–179.

[8] A. Fichtner and J. Trampert *Hessian kernels of seismic data functionals based upon adjoint techniques*, Geophys. J. Int., 185 (2011), pp. 775–798.

[9] J. A. George, *Nested dissection of a regular finite element mesh*, SIAM J. Numer. Anal., 10 (1973), pp. 345–363.

[10] J. R. Gilbert and S.-H. Teng, *MESHPART, A Matlab Mesh Partitioning and Graph Separator Toolbox*, http://aton.cerfacs.fr/algor/Softs/MESHPART/.

[11] S. Li, S. Ahmed, G. Klimeck, and E. Darve, *Computing entries of the inverse of a sparse matrix using the FIND algorithm*, J. Comput. Phys., 227 (2008), pp. 9408–9427.

[12] S. Li and E. Darve, *Extension and optimization of the FIND algorithm: Computing Green's and less-than Green's functions*, J. Comput. Phys., 231 (2012), pp. 1121–1139.

[13] L. Lin, J. Lu, L. Ying, R. Car, and W. E, *Fast algorithm for extracting the diagonal of the inverse matrix with application to the electronic structure analysis of metallic systems*, Commun. Math. Sci. 7 (2009), pp. 755–777.

[14] L. Lin, C. Yang, J. Lu, L. Ying, and W. E, *A fast parallel algorithm for selected inversion of structured sparse matrices with application to 2D electronic structure calculations*, SIAM J. Sci. Comput., 33 (2010), pp. 1329—1351.

[15] L. Lin, C. Yang, J. Meza, J. Lu, L. Ying, and W. E, *SelInv—An algorithm for selected inversion of a sparse symmetric matrix*, ACM Trans. Math. Software, 37 (2011), pp. 40:1—40:19.

[16] J. W. H. Liu, *The Multifrontal Method for Sparse Matrix Solution: Theory and Practice*, SIAM Rev., 1 (1992), pp. 82–109.

[17] METIS, *Family of Multilevel Partitioning Algorithms*, http://glaros.dtc.umn.edu/gkhome /views/metis.

[18] L. Métivier, R. Brossier, J. Virieux, S. Operto *The truncated Newton method for Full Waveform Inversion*, J. Phys.: Conf. Ser., 386 (2012), 012013.

[19] H. Niessner and K. Reichert, *On computing the inverse of a sparse matrix*, International Journal for Numerical Methods in Engineering, 19 (1983), pp. 1513–1526.

[20] S. V. Parter, *The use of linear graphs in gaussian elimination*, SIAM Rev., 3 (1961), pp. 119–130.

[21] K. Takahashi, J. Fagan, and M. Chin, *Formation of a sparse bus impedance matrix and its application to short circuit study*, in Proceedings 8th PICA Conference, Minneapolis, Minnesota, 1973.

[22] J. Tang and Y. Saad, *A probing method for computing the diagonal of the matrix inverse*, Technical report umsi-2010-42, Minnesota Supercomputer Institute, University of Minnesota, Minneapolis, 2009.

[23] J. Tang and Y. Saad, *Domain-decomposition-type methods for computing the diagonal of a matrix inverse*, SIAM J. Sci. Comput., 33 (2011), pp. 2823–2847.

[24] R. B. Sidje and Y. Saad, *Rational approximation to the Fermi-Dirac function with applications in density functional theory*, Tech. Rep. umsi-2008-279, Minnesota Supercomputer Institute, University of Minnesota, (2008).

[25] Y. Xi, J.Xia, V.Balakrishnan, and S.Cauley, *Superfast solutions for Toeplitz least squares via randomized sampling*, SIAM J. Matrix Anal. Appl., submitted, 2012.

[26] J. Xia, Y. Xi, and M. Gu, *A superfast structured solver for Toeplitz linear systems via randomized sampling*, SIAM J. Matrix Anal. Appl., 33 (2012), pp. 837–858.

[27] J. Xia, *On the complexity of some hierarchical structured matrix algorithms*, SIAM J. Matrix Anal. Appl., 33 (2012), pp. 388–410.

[28] J. Xia, *Efficient structured multifrontal factorization for general large sparse matrices*, SIAM J. Sci. Comput., 35 (2013), pp. A832–A860.

[29] J. Xia, *Randomized sparse direct solvers*, SIAM J. Matrix Anal. Appl., 34 (2013), pp. 197–227.

[30] J. Xia, S. Chandrasekaran, M. Gu, and X. S. Li, *Superfast multifrontal method for large structured linear systems of equations*, SIAM J. Matrix Anal. Appl., 31 (2009), pp. 1382–1411.

[31] J. Xia, S. Chandrasekaran, M. Gu, and X. S. Li, *Fast algorithms for hierarchically semiseparable matrices*, Numer. Linear Algebra Appl., 17 (2010), pp. 953–976.

[32] J. Xia, Y.Xi, V.Balakrishnan, and S.Cauley, *Superfast structured selected inversion for large sparse matrices*, SIAM J. Sci. Comput., to be submitted, (2013).

[33] Y. Xi, J. Xia and R. Chan, *A fast structured eigensolver for symmetric Toeplitz matrices and more via adaptive randomized sampling*, SIAM J. Matrix Anal. Appl., submitted, (2013).