Incomplete factorization preconditioners and their updates with applications - III[1,2]

Daniele Bertaccini, Fabio Durastante

Rome – September 5, 2016

Università di Roma Tor Vergata

Università degli Studi dell'Insubria
Department of Science and High Technology

Notes of the course: *"Incomplete factorization preconditioners and their updates with applications"*.
Lesson 3. Incomplete biconjugation methods.

> An "A^{-1}" in a formula almost always means "solve a linear system" and almost never means "compute A^{-1}."
>
> Golub–Van Loan

In this lecture we want to present another approach to *explicit preconditioning*. For this kind of preconditioner there exists various implementation and formulations[3], we are going to start with a first version of this strategy for a matrix $A \in \mathbb{R}^{n\times n}$ that is symmetric and positive definite, as presented in [Benzi et al., 1996].

To obtain the desired factorization of the inverse we need to generate an A-biconjugation process starting from a set of conjugate directions $\{\mathbf{z}_i\}_{i=1,\dots,n}$. In this way we will write the matrix A in the form:

$$Z^T A Z = D,\quad Z = \begin{pmatrix} \vdots & \vdots & & \vdots \\ \mathbf{z}_1 & \mathbf{z}_2 & \dots & \mathbf{z}_n \\ \vdots & \vdots & & \vdots \end{pmatrix},\quad D = \begin{pmatrix} p_1 & 0 & \cdots & 0 \\ 0 & p_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & p_n \end{pmatrix},$$

where the element $p_i = \mathbf{z}_i^T A \mathbf{z}_i$. From this decomposition follows that the required factorization of the inverse is given by $A^{-1} = ZD^{-1}Z^T$, and this is because the Z matrix is a unit and upper triangular matrix. Observe that in reality we have obtained a slight variation of the root-free Cholesky factorization[4] of A: $A = LDL^T$, where $Z = L^{-T}$. In this way the identity $AZ = LD$ can be stated. Observe that the complete A-biconjugation strategy can give rise, starting from a sparse matrix A, to a full triangular matrix Z. As we have seen for the incomplete LU factorization, to obtain a preconditioner:

$$M^{-1} = ZD^{-1}Z^T$$

from this kind of strategy some dropping strategy for going from Z to its sparsifying $\tilde{Z}$ is needed.

As we have done in the first lesson for the incomplete LU factorization to obtain the existence proof for the various dropping strategies we need to start from the definition of the full A-biconjugation procedure in algorithm (1). The incomplete factorization arises from the insertion of a dropping rule to apply to the $\mathbf{z}_j^{(i)}$ vectors. This rule can based on a positional strategy or on a drop tolerance.

[3] For different formulations see [Benzi et al., 1996, 2000a,b, Bridson and Tang, 1999, Kharchenko et al., 2001]. While for the different implementations on high performance computer see [Bertaccini and Filippone, 2016, Filippone and Buttari].

[4] Note that this way of calculating the Cholesky factorization is impractical for dense system, it does two time the amount of work of the standard implementation.

Algorithm 1: Inverse Factorization Algorithm (Right-looking)

Input: $A \in \mathbb{R}^{n\times n}$ symmetric and positive definite.
Output: $Z \in \mathbb{R}^{n\times n}$ upper triangular, $\mathbf{p}$ vector of the diagonal elements.

```
for i = 1,...,n do
    z_i^(0) ← e_i;
for i = 1,2,...,n do
    for j = i, i+1,...,n do
        p_j^(i-1) ← a_{i,:}^T z_j^(i-1);
    if i ≠ n then
        for j = 1,2,...,n do
            z_j^(i) ← z_j^(i-1) − (p_j^(i-1) / p_i^(i-1)) z_i^(i-1);
for i = 1,2,...,n do
    z_i = z_i^(i-1);
    p_i = p_i^(i-1);
```

For a generic drop tolerance and a generic M-matrix we can state the following theorem from [Benzi et al., 1996]:

Theorem 1

Let A be an M-matrix and let p_i the pivots produced by the algorithms (1). If $\overline{p}_i$ are the pivots computed by the incomplete factorization algorithm with any preset zero pattern in the strictly upper triangular part of Z or any value of the drop tolerance, then:

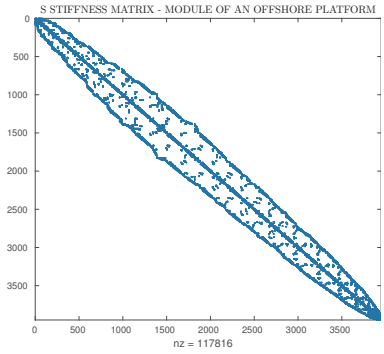
$$\overline{p}_i \geq p_i > 0.$$

Proof

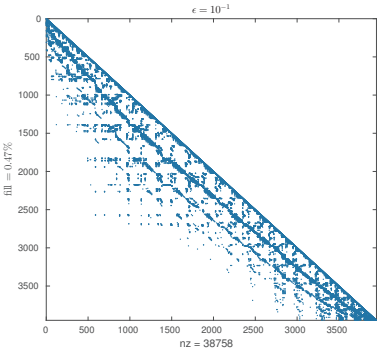
From the identity $AZ = LD$ and the fact that Z and L are unit triangular matrices it follows that the pivots p_i can be expressed in terms of the leading principal minors Δ_i of A as:

$$p_i = \frac{\Delta_i}{\Delta_{i-1}}, \quad i = 1,2,\ldots,n \quad (\Delta_0 = 1).$$

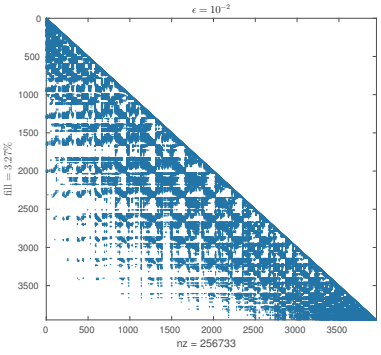
Now by the definition of M-matrix, we have that all the leading principal minors are positive and therefore $p_i > 0 \ \forall\, i > 0$. After $i-1$ steps of the algorithm (1) the column vectors $\mathbf{z}_j^{(i-1)}$ for $j = 1,\ldots,n$ are available. At the i-th step of the inverse factorization



(a) Pattern of HB/bcsstk15



(b) Pattern of $\tilde{Z}$ for $\varepsilon = 10^{-1}$



(c) Pattern of $\tilde{Z}$ for $\varepsilon = 10^{-2}$

Figure 1: Effect of dropping for the approximate inverse of the matrix HB/bcsstk15 from the Harwell-Boing collection [Davis and Hu, 2011].

scheme the element $p_j^{(i-1)}$ is calculated as:

$$p_j^{(i-1)} = \sum_{l=1}^{i-1} a_{i,l} z_{l,j}^{(i-1)} + a_{i,j}, \quad j = 1, \ldots, n$$

Now we introduce the dropping rule on $\mathbf{z}$-vector as one, or both:

Pattern Dropping $\mathbf{z}_{k,j}^{(i)} \neq 0 \Leftrightarrow (k,j) \in P$;

Tolerance Dropping Given a drop tolerance ε the element $z_{k,i}$ is discarded, namely $\mathbf{z}_{k,j}^{(i)} = 0$, if $|\mathbf{z}_{k,j}^{(i)}| < \varepsilon$.

In this way the we obtained the modified $\mathbf{z}$-vector as: $\overline{\mathbf{z}}_j^{(i-1)}$, and the pivots are now given by:

$$\overline{p}_i^{(i-1)} = \sum_{l=1}^{i-1} a_{i,l} \overline{z}_{l,i}^{(i-1)} + a_{i,i},$$

Now we show by *induction* on i that:

1. $\overline{p}_i^{(i-1)} > 0$ for $1 \leq i \leq n$;
2. $\overline{p}_j^{(i-1)} \leq 0$ for $i+1 \leq j \leq n$;
3. $\overline{\mathbf{z}}_j^{(i-1)} \geq 0$ for $1 \leq j \leq n$, $\forall i \geq 1$, componentwise.

Now for $i = 1$ the inequality are obviously true, by the starting initialization of the algorithm (1). Now we fix $i \geq 2$ and assume that three inequality holds for $i-1$, so we can express the updates as:

$$\overline{\mathbf{z}}_j^{(i-1)} = \overline{\mathbf{z}}_j^{(i-2)} - \frac{\overline{p}_j^{(i-2)}}{\overline{p}_{i-1}^{(i-2)}} \overline{\mathbf{z}}_{i-1}^{(i-2)} \geq 0,$$

because we are subtracting non-positive quantity, possibly being zero by the dropping rules, from non negative quantity, and so the third inequality is proved. To show the second one, we rewrite the formula:

$$\overline{p}_j^{(i-1)} = \sum_{l=1}^{i-1} a_{i,l} \overline{z}_{l,j}^{(i-1)} + a_{i,j} \leq 0 \quad j = i+1, \ldots, n$$

and so because the $a_{i,j}$ are the extra-diagonal element of M-matrix and the previous equation we are summing up negative quantities. To prove the first inequality is sufficient to observe that $\overline{p}_i^{(i-1)} \geq p_i^{(i-1)} > 0$, because we cannot obtain smaller pivot because of dropping, and since the original pivot are positive.

Remark 1. *In the proof of the proposition the symmetry of the matrix A is not used. In the latter this will permit the generalization of the A-biconjugation process to a non symmetric matrix.*

Applying this algorithm to an M-matrix A, symmetric and posi-

tive definite, a split preconditioner M for the conjugate gradient, as:

$$M^{-1} = (ZD^{-1/2})(D^{-1/2}Z^T),$$

can be built.

At last, before treating the non symmetric case, we give another formulation of the $AINV$ algorithm, following [Bertaccini and Filippone, 2016]. In the version (1) a *right-looking* approach is performed, namely this is to use the final version of the vector $\mathbf{z}_i$ to update all the vectors $\mathbf{z}_j$ $\forall\, j > 1$. Now we report a reformulation of the algorithm to have all the updates to $\mathbf{z}_i$, involving the vector $\mathbf{z}_j$, for $j < 1$ done concurrently. Observe that in exact arithmetic the numerical behaviour of the two algorithms is *exactly* the same. On the other hand, the distribution of work in the two approach is very different. The left-looking approach groups together all the updates operation on a given column $\mathbf{z}_i$. In this way the update is based on computing sparse dot product before doing the dropping on the column, therefore the dropping is done only once at the end of the update.

Algorithm 2: Inverse Factorization Algorithm (Left-looking)

Input: $A \in \mathbb{R}^{n\times n}$ symmetric and positive definite.
Output: $Z \in \mathbb{R}^{n\times n}$ upper triangular, $\mathbf{p}$ vector of the diagonal elements.

1 $\mathbf{z}_1^{(0)} \leftarrow \mathbf{e}_1$;
2 $p_1^{(0)} \leftarrow a_{1,1}$;
3 **for** $i = 2, \ldots, n$ **do**
4 $\mathbf{z}_i^{(0)} \leftarrow \mathbf{e}_i$;
5 **for** $j = 1, \ldots, i-1$ **do**
6 $p_i^{(j-1)} \leftarrow \mathbf{a}_{:,j}^T \mathbf{z}_i^{(j-1)}$;
7 $\mathbf{z}_i^{(j)} \leftarrow \mathbf{z}_i^{(j-1)} - \frac{p_i^{(j-1)}}{p_j^{(j-1)}} \mathbf{z}_j^{(j-1)}$;
8 $p_i^{(i-1)} \leftarrow \mathbf{a}_{:,i}^T \mathbf{z}_i^{(i-1)}$;

Referring to the formulation in algorithm (2), we have that the vector product at row (6) and (7) are done before the application of the dropping to the vector $\mathbf{z}_i$. Then the dropping rule is applied at the end of the update for loop at row (5), instead of applying it every time the $\mathbf{z}_i$ are updated as in Algorithm 1.

As we have observed (rmk. 1) the existence proof for M and H–matrix[5] doesn't need symmetry. So it's possible to go from a symmetric A-biconjugation algorithm to a full A-biconjugation. To achieve this for a matrix $A \in \mathbb{R}^{n\times n}$ two family of A-orthogonal direction are needed, namely the $\{\mathbf{z}_i\}_{i=1,2,\ldots,n}$ and the $\{\mathbf{w}_i\}_{i=1,2,\ldots,n}$ such that:

$$W^T A Z = D = \begin{pmatrix} p_1 & 0 & \cdots & 0 \\ 0 & p_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & p_n \end{pmatrix},$$

[5] The definition of H–matrix is derived from the definition of M–matrix in the following way,

Definition 1: H–matrix

Given a matrix $(a_{i,j}) = A \in \mathbb{R}^{n\times n}$, we says that A is an H–**matrix** if the matrix $(b_{i,j}) = B \in \mathbb{R}^{n\times n}$ with $b_{i,i} = a_{i,i}\ \forall\, i = 1, \ldots, n$ and $b_{i,j} = -|a_{i,j}|$ for $i \neq j$ is an M–matrix.

where:

$$W = \begin{pmatrix} \vdots & \vdots & & \vdots \\ \mathbf{w}_1 & \mathbf{w}_2 & \cdots & \mathbf{w}_n \\ \vdots & \vdots & & \vdots \end{pmatrix}, \quad Z = \begin{pmatrix} \vdots & \vdots & & \vdots \\ \mathbf{z}_1 & \mathbf{z}_2 & \cdots & \mathbf{z}_n \\ \vdots & \vdots & & \vdots \end{pmatrix}.$$

In this way the coefficients are given by $p_i = \mathbf{w}_i^t A \mathbf{z}_i \neq 0$, and so also the W and Z are non singular and orthogonal triangular matrix. With this decomposition the inverse of the matrix A can be expressed as:

$$A^{-1} = ZD^{-1}W^T = \sum_{i=1}^{n} \frac{\mathbf{z}_i \mathbf{w}_i^T}{p_i}.$$

While for the symmetric case the A-biconjugation process was analogous to calculate the root-free Cholesky factorization, this case is the analogue of building the LDU-factorization of the matrix A, where $L = W^{-T}$ and $U = Z^{-1}$. An implementation of the full A-biconjugation process is in algorithm (3), the notation used is of $\mathbf{a}_{i,:}$ for the row of the A matrix, and $\mathbf{c}_{i,:}$ for the row of the A^T matrix, namely $\mathbf{c}_{i,:} = \mathbf{a}_{:,i}^T$. As starting point for the W and Z matrix the choice $W = Z = I_{n\times n}$ is taken.

Algorithm 3: Full Biconjugation Algorithm (Right-looking)

Input: A matrix $A \in \mathbb{R}^{n\times n}$.

Output: Two matrix Z, W and a vector $\mathbf{p}$ such that
$A^{-1} = Z\,\mathrm{diag}(1/p)W^T$.

```
for i = 1, 2, ..., n do
    w_i^(0) = z_i^(0) = e_i;
for i = 1, 2, ..., n do
    for j = i, i+1, ..., n do
        p_j^(i-1) ← a_{i,:}^T z_j^(i-1);        // p_i = w_i^T A z_i ← z_i^T A^T w_i = q_i
        q_j^(i-1) ← c_{i,:}^T w_j^(i-1);        // One could put q_j^(i-1) ← p_j^(i-1)
    for j = i+1, ..., n do
        z_j^(i) ← z_j^(i) − (p_j^(i-1) / p_i^(i-1)) z_i^(i-1);
        w_j^(i) ← w_j^(i) − (q_j^(i-1) / q_i^(i-1)) w_i^(i-1);
for i = 1, 2, ..., n do
    z_i ← z_i^(i-1);
    w_i ← w_i^(i-1);
    p_i ← p_i^(i-1)
return Z = [z_1, z_2, ..., z_n], W = [w_1, w_2, ..., w_n] and
p = (p_1, p_2, ..., p_n)^T.
```

For A an M-matrix no breakdown of the algorithm, meaning that $p_i^{(i-1)} = q_i^{(i-1)} = 0$, can occur, also if we include a dropping strategy. This is a direct consequence of what the we have proved with the

proposition (1). And this is because the algorithm (3) is nothing more than the algorithm (1) applied two time, one to the A matrix and one to the A^T matrix. In case of non M matrix, having this algorithm an algebraic equivalence to the LDU-factorization, to ensure the absence of breakdown, for the version without dropping, is sufficient to have all that all the principal minor of A are nonzero. Namely this is due to the relation:

$$p_i = \frac{\Delta_i}{\Delta_{i-1}}, \;\; i = 1, \ldots, n \text{ and } \Delta_0 = 1.$$

Some variants of the algorithm

We will not go into detail about all the possible variants of this class of algorithms, however, to allow for further study, we mention some general strategies that have been developed.

We start focusing our attention on the breakdown case for symmetric and positive definite matrix, namely the appearance of a non-positive pivot during the A-orthogonalization process. The case of negative pivot is not, properly speaking, a breakdown of the algorithm, but taking into account the use of a non positive definite preconditioner for a symmetric and positive matrix is not a feasible strategy. Also having a pivot with absolute value around the machine error is not a breakdown, but it gives back all the same a poor quality preconditioner.

The first solution that the authors in [Benzi et al., 2000a] take into account is the so called SAINV, or *stabilized* AINV. This is a rewriting of the algorithm (1) in an algebraic equivalent form with a supposed better numerical behavior for a generic symmetric and positive definite matrix. The way to avoid nonpositive pivots is to restart with the matrix equation $Z^T AZ = D$ and then observe that the pivots can be written as:

$$p_i^{(i-1)} = \mathbf{z}_i^T A \mathbf{z}_i > 0, \;\; 1 \leq i \leq n,$$

by this we have that in the exact process, where for exact we mean without dropping, the pivots can be calculated as:

$$p_i^{(i-1)} = \mathbf{z}_i^T A \mathbf{z}_i = \mathbf{a}_{i,:}^T \mathbf{z}_i,$$

and the equality holds because Z is an upper triangular matrix and AZ a lower triangular matrix. On the other hand, applying the dropping rule cause the precedent equality to transform in:

$$\mathbf{a}_{i,:}^T \bar{z}_i \leq \bar{\mathbf{z}}_i^T A \bar{\mathbf{z}}_i.$$

In this way having bigger, and positive, pivots for the algorithm is possible. In the following doing the calculation for the $\bar{p}_i^{(i-1)}$ for $i = 1, 2, \ldots, n$ with the bilinear form $\bar{p}_i^{(i-1)} = \bar{\mathbf{z}}_i^T A \bar{\mathbf{z}}_i$ is considered. Then a formulation for computing the $\bar{p}_j^{(i-1)}$ for $i + 1 \leq j \leq n$ in the internal cycle is needed. Calculating it as the dot product of the precedent algorithm is not feasible. Doing it that way drives back to

a possible decrease of the pivots, so it need to be calculated in two steps as:

$$\overline{v}_i^T = \overline{z}_i^T A, \;\; \overline{p}_j = \overline{v}_i^T \overline{\mathbf{z}}_j.$$

Using this variant of the algorithm (4) is clear that the cost will increase, going from a dot product to the application of a bilinear form. Nevertheless the increasing is not the full increase of going from a complete dot product to a complete matrix vector product, i.e. the $\text{nnz}(\overline{\mathbf{z}}_i) << n$, and so the construction of the $\overline{v}_i^T$ and of the $\overline{p}_i^{(i-1)}$ is just an operation linear in n.

Algorithm 4: SAINV

Input: $A \in \mathbb{R}^{n\times n}$ symmetric and positive definite.
Output: $Z \in \mathbb{R}^{n\times n}$ upper triangular, $\mathbf{p}$ vector of the diagonal elements.

```
for i = 1, ..., n do
    z_i^(0) ← e_i;
for i = 1, 2, ..., n do
    v_i ← A z_i^(i-1);
    for j = i, i+1, ..., n do
        p_j^(i-1) ← v_i^T z_j^(i-1);
    if i ≠ n then
        for j = 1, 2, ..., n do
            z_j^(i) ← z_j^(i-1) − (p_j^(i-1) / p_i^(i-1)) z_i^(i-1);
for i = 1, 2, ..., n do
    z_i = z_i^(i-1);
    p_i = p_i^(i-1);
```

Other strategies that we want to mention are

Diagonal shifting for SPD matrix miming the strategy used to avoid the non-positive pivots for the incomplete Cholesky factorization, namely if the pivots became to near to the machine error, or negative, one can try to calculate the AINV preconditioner on a matrix $A_\alpha = A + \alpha\,\text{diag}(A)$, instead on calculating it onto the matrix A.

Diagonally compensated reduction the kernel of the idea is a diagonally compensate reduction of the extra-diagonal entries of the matrix A, see again [Benzi et al., 2000a].

Reordering algorithms a class of strategies for performing the AINV factorization over a generic matrix $A \in \mathbb{R}^{n\times n}$. As a general idea we want to find a permutation matrix P such that the resulting diagonal entry of the permuted matrix PA is large relative to the absolute values of the off-diagonal entries in that row and in that

column, namely this is making as large as possible the ratio:

$$\frac{|a_{j,j}|}{\max\limits_{i \neq j} |a_{i,j}|}, \quad \forall j = 1, 2, \ldots, n. \tag{1}$$

To make possible having this ratio greater than one for all j is necessary to scale the matrix before computing the permutation, i.e. an appropriate scaling can be to scale the column so that the largest entry in each column is 1, then an algorithm maximizing the ratio can exists.

Block variants is also possible to formulate the AINV algorithm (2) or the SAINV algorithm (4) also in a full block mode, as has been done in [Benzi et al., 2001], namely it consist in first recognize a block decomposition of the matrix $A \in \mathbb{R}^{n \times n}$, such that:

$$A = \begin{pmatrix} A_{1,1} & A_{1,2} & \cdots & A_{1,N} \\ A_{2,1} & A_{2,2} & \cdots & A_{2,N} \\ \vdots & \vdots & \ddots & \vdots \\ A_{N,1} & A_{N,2} & \cdots & A_{N,N} \end{pmatrix}, \quad \dim(A_{i,j}) = n_i \times n_j \;\; 1 \leq n_i \leq N.$$

Then we denote $m_i = \sum_{j<i} n_j$, that is the offset of the i-th block. We also denote the block rows of A by:

$$\mathbf{A}_i^T = (A_{i,1}, \ldots, A_{i,N}), \quad i = 1, 2, \ldots, N.$$

Then if we start from an A matrix symmetric and positive definite we have that the diagonal blocks $A_{i,i}$ are square symmetric and positive definite matrices, and that $A_{j,i} = A_{i,j}^T$ for $i \neq j$[6].

With this lecture we have concluded the overview of the preconditioning methods we wanted to propose in this course. The material we have covered till now is a classical topic, excepting for some of the new details we have covered regarding GPU implementations[7], with many possible applications. Among this set we will pick up some selected one for our last lesson. We will focus our attention, mainly, on the topics of updating preconditioners when facing sequences of linear systems. Namely we will deal with the problem of obtaining a sequence of preconditioners $\{P^{(k)}\}_{k \geq 0}$ for a sequence of linear systems $\{A_n^{(k)} \mathbf{x}^{(k)} = \mathbf{b}^{(k)}\}_{k \geq 0}$, with $A_n^{(k)} \in \mathbb{R}^{n \times n}$ $\forall k \geq 0$, without recomputing a new one for each element of the sequence. In this we are going to focus mainly on the approach in [Benzi and Bertaccini, 2003, Bertaccini, 2004, Bertaccini and Durastante, 2016].

[6] Methods to select the dropping rule are discussed in [Bridson and Tang, 2000], for simplicity one can consider a dropping based on $\|\cdot\|_\infty$ norm.

[7] Daniele Bertaccini and Salvatore Filippone. Sparse approximate inverse preconditioners on high performance gpu platforms. *Computers & Mathematics with Applications*, 71(3):693 – 711, 2016. ISSN 0898-1221

References

Michele Benzi and Daniele Bertaccini. Approximate inverse preconditioning for shifted linear systems. *BIT Numerical Mathematics*, 43, 2003. 10.1023/a:1026089811044.

Michele. Benzi, C. Meyer, and Miroslav Tuma. A sparse approximate inverse preconditioner for the conjugate gradient method. *SIAM Journal on Scientific Computing*, 17(5):1135–1149, 1996. 10.1137/S1064827594271421. URL http://dx.doi.org/10.1137/S1064827594271421.

Michele Benzi, Jane K. Cullum, and Miroslav Tuma. Robust approximate inverse preconditioning for the conjugate gradient method. *SIAM Journal on Scientific Computing*, 22(4):1318–1332, 2000a.

Michele Benzi, J. Haws, and Miroslav Tuma. Preconditioning highly indefinite and nonsymmetric matrices. *SIAM Journal on Scientific Computing*, 22(4):1333–1353, 2000b. 10.1137/S1064827599361308. URL http://dx.doi.org/10.1137/S1064827599361308.

Michele Benzi, Reijo Kouhia, and Miroslav Tuma. Stabilized and block approximate inverse preconditioners for problems in solid and structural mechanics. *Computer methods in applied mechanics and engineering*, 190(49):6533–6554, 2001.

Daniele Bertaccini. Efficient preconditioning for sequences of parametric complex symmetric linear systems. *ETNA. Electronic Transactions on Numerical Analysis*, 18:49–64, 2004. URL http://eudml.org/doc/124865.

Daniele Bertaccini and Fabio Durastante. Interpolating preconditioners for the solution of sequence of linear systems. *Comp. Math. Appl.*, 72:1118–1130, 2016.

Daniele Bertaccini and Salvatore Filippone. Sparse approximate inverse preconditioners on high performance gpu platforms. *Computers & Mathematics with Applications*, 71(3):693 – 711, 2016. ISSN 0898-1221.

Robert Bridson and Wei-Pai Tang. Ordering, anisotropy, and factored sparse approximate inverses. *SIAM Journal on Scientific Computing*, 21(3):867–882, 1999.

Robert Bridson and Wei-Pai Tang. Refining an approximate inverse. *Journal of computational and applied mathematics*, 123(1):293–306, 2000.

Timothy A Davis and Yifan Hu. The university of florida sparse matrix collection. *ACM Transactions on Mathematical Software (TOMS)*, 38(1):1, 2011.

Salvatore Filippone and Alfredo Buttari. Parallel sparse basic linear algebra subroutines. URL http://people.uniroma2.it/salvatore.filippone/psblas/.

SA Kharchenko, L Yu Kolotilina, AA Nikishin, and A Yu Yeremin. A robust ainv-type method for constructing sparse approximate inverse preconditioners in factored form. *Numerical linear algebra with applications*, 8(3):165–179, 2001.