A robust multilevel approximate inverse preconditioner for symmetric positive definite matrices

ANDREA FRANCESCHINI, VICTOR MAGRI, MASSIMILIANO FERRONATO AND CARLO JANNA

February 16th, 2017
Our main goal is solving real world problems arising from the discretization of PDE in various fields of application.

**Algebraic preconditioners**: robust tools which can be used knowing the coefficient matrix only, regardless of the specific problem addressed.

Incomplete factorizations, algebraic multigrid, **sparse approximate inverses** (FSAI).
**FSAI preconditioner for SPD linear systems**

- Factorized Sparse Approximate Inverse (FSAI): an almost perfectly parallel factored preconditioner [Kolotilina and Yeremin, 1993]

\[ A^{-1} = F^T F \]

with \( F \) a lower triangular matrix such that:

\[ \| I - FL \|_F \rightarrow \min \]

over the set of matrices with a prescribed lower triangular sparsity pattern \( S_L \) where \( L \) is the exact Cholesky factor of \( A \). \( L \) is not actually required for computing \( F \! \).

- Computed via the solution of \( n \) independent small dense systems and applied via matrix-vector products

- Nice features: (1) ideally perfect parallel construction and application of the preconditioner; (2) preservation of the positive definiteness of the native matrix
One of the main difficulties stems from the selection of $S_L$ as an a priori sparsity pattern for $F$

Using small powers of $A$ is a popular choice, but for difficult problems high powers may be needed and the preconditioner construction can become quite heavy.

A most efficient option relies on selecting the pattern dynamically by an adaptive procedure which uses somewhat the “best” available positions for the non-zero coefficients.

The procedure is based on the minimization of the Kaporin conditioning number $\kappa$ of an SPD matrix which is defined as:

$$\kappa(A) = \frac{\text{tr}(A)}{n \det(A)^{1/n}}$$

It can be shown that this procedure, though expensive, has some optimality properties and gives satisfactorily results for SPD systems.
The drawback of static FSAI is the choice of the sparsity pattern. The cost increases with the third power of the preconditioner density.

\[ c \propto n \cdot \bar{m}_G^3 = n \cdot \bar{m}_A^3 \mu_G^3 \]

The adaptive FSAI overcomes the choice of the pattern, but the cost increases with the fourth power of the preconditioner density.

\[ c = \sum_{i=1}^{n} c_i \propto n \cdot k_{iter}^4 = n \cdot \bar{m}_A^4 \mu_G^4 \]
Multilevel approaches for FSAI preconditioning

Basic concepts
- To improve the quality of the FSAI preconditioner we borrow the scheme of the incomplete factorization
- To reduce the cost of the dynamic selection of the pattern (the set where we look for best coefficients) in the adaptive FSAI algorithm, we developed two new approaches:
  - Block tridiagonal FSAI
  - Domain Decomposition FSAI: a particular case of the former, with $n = 2$. Blocks are defined as internal unknowns ($1^{st}$ block) and interface unknowns ($2^{nd}$ block) after a Domain Decomposition reordering
MFLR preconditioner

\[
A = \begin{bmatrix}
A_1 & B_1 \\
B_1^T & A_2 & B_2 \\
& \ddots & \ddots & \ddots \\
& & B_{n-2}^T & A_{n-1} & B_{n-1} \\
& & & B_{n-1}^T & A_n
\end{bmatrix}
\]

Factorization in the LDL^T form (Cholesky-like)

\[
A = \begin{bmatrix}
I_1 & B_1^T S_1^{-1} & I_2 \\
& \ddots & \ddots & \ddots \\
& & B_{n-1}^T S_{n-1}^{-1} & I_n
\end{bmatrix}
\begin{bmatrix}
S_1 \\
S_2 \\
\vdots \\
S_n
\end{bmatrix}
\begin{bmatrix}
I_1 & S_1^{-1} B_1 \\
I_2 & S_2^{-1} B_2 \\
& \ddots & \ddots \\
& & I_n
\end{bmatrix}
\]

**KEY ROLE**

Computation of the Schur complement

\[
S_i = A_i - B_i^{T} S_{i-1}^{-1} B_i, \quad i = 2 \ldots n
\]

\[
S_i^{-1} \approx F_i^{T} F_i
\]

The inverse of S is factorized with the adaptive FSAI, implemented in FSAIPACK [Janna et al., 2015]
Domain Decomposition FSAI

Following the previous idea, we split the matrix into 2x2 block structure, where the two blocks have different size. Use of domain decomposition technique. Note that this is a particular case of the Block Tridiagonal with 2 blocks.

\[
\tilde{A} = \begin{bmatrix}
A_1 & B_1 \\
B_1^T & A_2
\end{bmatrix} = \begin{bmatrix}
(A_1)_1 & \cdots & (B_1)_1 \\
\vdots & \ddots & \vdots \\
(B_1^T)_1 & \cdots & (A_2)_n
\end{bmatrix}
\]

In this case, we need to compute just one Schur complement.
Algorithm to compute and to apply the preconditioner

Algorithm 1 Construction of the BTFSAI preconditioner

1: procedure $\text{cp} \text{rBTFSAI}(A,n)$
2: Compute $A_1^{-1} \approx F_1^T F_1$
3: for $i = 2, \ldots, n$ do
4: Compute $H_{i-1} = F_{i-1} B_{i-1}$
5: Compute $S_i = A_i - H_{i-1}^T H_{i-1}$
6: Compute $S_i^{-1} = F_i^T F_i$
7: end for
8: end procedure

S can be indefinite

Algorithm 2 Calculation of $u = M^{-1} v$

1: procedure $\text{APPLYBTFSAI}(F_i, H_i, v)$
2: $w_1 = F_1 v_1$
3: for $i = 2, \ldots, n$ do
4: $w_i = F_i (v_i - H_{i-1}^T w_{i-1})$
5: end for
6: $u_n = F_n^T w_n$
7: for $i = n-1, \ldots, 1$ do
8: $u_i = F_i^T (w_i - H_i u_{i+1})$
9: end for
10: end procedure

Sequential framework

Parallel phases

Parallel phases

Sequential framework

BREAKDOWN
Results

Solver: CG
Exit tolerance: $10^{-8}$
CPU: 1 core
Robust Multilevel approaches for FSAI preconditioning

Basic concepts

- The previous implementation of the multilevel is very prone to **breakdowns**: Schur complement approximation is the difference of two SPD matrices and can be **indefinite**. This happens also in relatively well-conditioned problems.
- The reason for this probably resides in the better approximation of the lower eigenvalues usually offered by ILU over FSAI.
- These issues motivate the development of a multilevel preconditioner which is more connected to the FSAI methodology.

Comparison between the spectra of $A$, $LL^T$ and $G^{-1}G^T$ for the bcsstk16 matrix from the University of Florida sparse matrix collection
**Schur complement computation**

- The matrix is subdivided into 4 blocks $A = \begin{bmatrix} K & B \\ B^T & C \end{bmatrix}$

- The adaptive FSAI matrix $G$ of the (1,1) block $K$ is computed, so the first phase of preconditioning is done.

\[
(G^T G)^{-1} \approx K \quad \text{with} \quad \begin{bmatrix} G & 0 \\ 0 & I \end{bmatrix} \begin{bmatrix} K & B \\ B^T & C \end{bmatrix} \begin{bmatrix} G^T & 0 \\ 0 & I \end{bmatrix} = \begin{bmatrix} GKG^T & GB \\ B^T G^T & C \end{bmatrix} \approx \begin{bmatrix} I & GB \\ B^T G^T & C \end{bmatrix}
\]

- The adaptive block FSAI matrix $F$ to decouple diagonal blocks is computed. With this also the second phase of preconditioning is done and the Schur complement is defined.

\[
F \approx -(GB)^T (GKG^T)^{-1}
\]

\[
\begin{bmatrix} I & 0 \\ F & I \end{bmatrix} \begin{bmatrix} GKG^T & GB \\ B^T G^T & C \end{bmatrix} \begin{bmatrix} I & F^T \\ 0 & I \end{bmatrix} = \begin{bmatrix} GKG^T & G(KG^T F^T + B) \\ (FGK + B^T) G^T & C + FGB + B^T G^T F^T + FGKG^T F^T \end{bmatrix} \approx \begin{bmatrix} I & 0 \\ 0 & \tilde{S} \end{bmatrix}
\]

Approximate Schur complement that becomes new matrix $A$ (multilevel)
Algorithm to compute the MFLR preconditioner

Multilevel FSAI preconditioner with Low-Rank corrections

Algorithm 2.2 Multilevel FSAI Set-up

1. Set $A_0 = A$;
2. for all $l = 0, \ldots, n_l - 2$ do
   3. Partition $A_l$ as $\begin{bmatrix} K & B \\ BT & C \end{bmatrix}$;
   4. Compute the adaptive FSAI approximation $G$ of $K$ such that $G^T G \simeq K^{-1}$;
   5. Set $P_a = \begin{bmatrix} G & 0 \\ 0 & I \end{bmatrix}$;
   6. Compute $P_b = \begin{bmatrix} I & 0 \\ F & I \end{bmatrix}$ the adaptive Block FSAI approximation of $P_a A_l P_a^T$;
   7. Compute $\tilde{S} = C + FGB + B^T G^T F^T + FGK G^T F^T$;
   8. Form $M_l^{-1} = P_b P_a$;
   9. Set $A_{l+1} = \tilde{S}$;
3. end for
11. Compute $M_{n_l-1}^{-1}$ as the adaptive FSAI approximation of $A_{n_l-1}$;
12. Set $M^{-1} = \{M_0^{-1}, M_1^{-1}, \ldots, M_{n_l-1}^{-1}\}$;

S is SPD
NO BREAKDOWN

Sequential framework
Parallel phases
Main Schur complement property

The Schur complement is an SPD matrix, indeed, with some algebra:

\[
\tilde{S} = C + FGB + B^T G F + FGK G F = C - B^T K^{-1} B + W^T KW
\]

where: \( W = K^{-1} B + G^T F \)

Real Schur complement (SPD)

Spectrum of \( \tilde{S} - S \) matrix for the \textit{bc}stress\textit{k38} matrix from the University of Florida sparse matrix collection
Low-Rank corrections

To improve the precondition quality, two different low rank corrections are developed:

- Descending Low-Rank: locally improvement of adaptive FSAI of (1,1) block
- Ascending Low-Rank: global improvement of adaptive FSAI of Schur complement [Xi et al., 2016]

Both ideas are based on follows observations:

- An approximation $G$ for the matrix $A$, such that $I - GAG^T = H \rightarrow 0$, is available
- We want to improve $A^{-1}$ as $A^{-1} = G^T G + G^T X G$
- Matrices $X$ and $H$ are linked by the simple relation: $X = H[I - H]^{-1}$
- To reduce ill-conditioning, eigenvalues of $A$ that we want to correct are ones close to 0, that are eigenvalues of $H$ close to 1. For these values, eigenvalues of $X$ tends to infinity
- Due to this fact, just a low-rank representation of matrix $H$ is needed. Moreover, computation of eigenvalues of matrix $X$ (the correction) is straightforward.
Low-Rank corrections

\( H \) matrix is symmetric, so we can write its eigen-decomposition (where \( \Sigma \) is diagonal):

\[
H = U \Sigma U^T
\]

\( X \) matrix is linked to \( H \), so we have:

\[
X = U \left[ \Sigma (I - \Sigma)^{-1} \right] U^T
\]

For each eigenvalue of \( H \), eigenvalue of \( X \) is:

\[
\sigma_X = \frac{\sigma_H}{1 - \sigma_H}
\]

A small number of eigenvalues gives rise to a big correction on the approximate inverse. Moreover, it can be shown that the Low-Rank corrections do not propagate along the levels.
Low-Rank target

Target of Ascending Low Rank is to make the available factorized inverse more similar to the inverse of one of two matrices:

- approximate Schur complement \( \tilde{S} = C + FGB + B^T G F^T + FGK G^T F^T \)
- real Schur complement \( S = C - B^T K^{-1} B \)

Eigenvalue bound with approximate Schur complement as target:

\[
|\lambda - 1| \leq \frac{\epsilon_1 + \sqrt{\epsilon_1^2 + 4\eta_1^2}}{2} \tag{1}
\]

Eigenvalue bound with real Schur complement as target:

\[
|\lambda - 1| \leq \frac{\epsilon_1 + \eta_1^2 \kappa_n^{-1} + \sqrt{(\epsilon_1 - \eta_1^2 \kappa_n^{-1})^2 + 4\eta_1^2}}{2} \tag{2}
\]

It can be shown that bound (1) is narrower or equal to the bound (2).

\[
\|E_K\|_2 = \lambda_1(E_K) = \epsilon_1, \quad \|Q\|_2 = \sqrt{\lambda_1(Q^T Q)} = \eta_1, \quad \|\tilde{Q}\|_2 = \sqrt{\lambda_1(\tilde{Q}^T \tilde{Q})} = \tilde{\eta}_1, \quad \|(I + E_K)^{-1}\|_2 = \lambda_n^{-1}(GK G^T) = \kappa_n^{-1}
\]
Algorithm 4.1 MFLR Set-up

1. Recursive Function MFLR_SETUP(I,A)
2. if \( I < (n_l - 1) \) then
3. Partition \( A_l \) as \(
\begin{bmatrix}
K & B \\
B^T & C
\end{bmatrix}
\)
4. Compute the adaptive FSAI approximation \( G \) of \( K \) such that \( G^T G \simeq K^{-1} \);
5. Set \( P_a = \begin{bmatrix} G & 0 \\ 0 & I \end{bmatrix} \);
6. Compute \( P_b = \begin{bmatrix} I & 0 \\ F & I \end{bmatrix} \) as the adaptive Block FSAI approximation of \( P_a A_l P_a^T \);
7. Compute \( \tilde{S} = C + FGB + B^TGT^T + FGKGT^T F^T \);
8. Compute a rank \( k \) approximation \( \tilde{U}_k \tilde{\Sigma}_k \tilde{U}_k^T \) of \( \tilde{Y} = I - GKG^T \);
9. Set \( \tilde{G} = (I + \tilde{U}_k \tilde{\Sigma}_k \tilde{U}_k^T) G \) with \( \tilde{\Sigma}_k = (I - \tilde{\Sigma}_k)^{-1/2} - I \);
10. Set \( P = \begin{bmatrix} \tilde{G} & 0 \\ FG & I \end{bmatrix} \);
11. \([Q_{alr}, Q_M] = MFLR_SETUP(I + 1, \tilde{S})\);
12. Use \( Q_{alr} \) and \( Q_M \) to compute the rank \( k \) correction \( \tilde{W}_k \) and \( \tilde{\Theta}_k \) as in (3.23);
13. Set \( P = \begin{bmatrix} I & 0 \\ 0 & I + \tilde{W}_k \tilde{\Theta}_k \tilde{W}_k^T \end{bmatrix} \);
14. Push \( \tilde{P} \) in the head of \( Q_{alr} \);
15. Push \( P \) in the head of \( Q_M \);
16. return \( Q_{alr}, Q_M \);
17. else
18. Compute the adaptive FSAI approximation \( G \) of \( A \) such that \( G^T G \simeq A \);
19. Set \( Q_{alr} = \emptyset \);
20. Set \( Q_M = \{G\} \);
21. return \( Q_{alr}, Q_M \);
22. end if
Algorithm 4.2 MFLR Apply

1. Recursive Function $MFLR\_APPLY(l, Q_{alr}, Q_M, x)$
2. if $l < (m-1)$ then
3. Pop $P$ from the head of $Q_M$;
4. Pop $\tilde{P}$ from the head of $Q_{alr}$;
5. Compute $y = P \cdot x$;
6. Partition $y = \{y_1^T, y_2^T\}^T$;
7. Compute $z_2 = MFLR\_APPLY(l+1, Q_{alr}, Q_M, y_2)$;
8. Compose $z = \{y_1^T, z_2^T\}^T$;
9. Compute $y = \tilde{P} \cdot z$;
10. Compute $y = P^T \cdot y$;
11. Push $P$ in the head of $Q_M$;
12. Push $\tilde{P}$ in the head of $Q_{alr}$;
13. return $y$;
14. else
15. Pop $G$ from the head of $Q_M$;
16. Compute $y = G^T \cdot G \cdot x$;
17. Push $G$ in the head of $Q_M$;
18. return $y$;
19. end if
Results: test case

We extensively tested Cube matrix, that has 190,581 rows and 7,531,389 non-zeroes

- Increasing the number of levels, number of iterations needed to reach convergence always decreases, but iteration time increases
- With Low-Rank corrections we can reduce both the number of iterations and the iteration time. In particular, the most promising technique is the Ascending Low-Rank

<table>
<thead>
<tr>
<th>levels</th>
<th>iter</th>
<th>ρ</th>
<th>TP</th>
<th>TS</th>
<th>TT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>686</td>
<td>0.363</td>
<td>8.49</td>
<td>14.44</td>
<td>22.93</td>
</tr>
<tr>
<td>10</td>
<td>640</td>
<td>0.774</td>
<td>34.45</td>
<td>21.87</td>
<td>56.32</td>
</tr>
<tr>
<td>20</td>
<td>591</td>
<td>1.173</td>
<td>265.92</td>
<td>29.30</td>
<td>295.22</td>
</tr>
<tr>
<td>50</td>
<td>503</td>
<td>2.038</td>
<td>441.67</td>
<td>41.62</td>
<td>483.30</td>
</tr>
<tr>
<td>100</td>
<td>464</td>
<td>3.674</td>
<td>612.91</td>
<td>66.96</td>
<td>679.87</td>
</tr>
</tbody>
</table>

Results for Cube matrix. All parameters are fixed, expect the number of levels

<table>
<thead>
<tr>
<th>rank</th>
<th>iter</th>
<th>ρ</th>
<th>TDLR</th>
<th>TP</th>
<th>TS</th>
<th>TT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>648</td>
<td>0.696</td>
<td>0.00</td>
<td>30.84</td>
<td>22.64</td>
<td>53.48</td>
</tr>
<tr>
<td>5</td>
<td>539</td>
<td>0.807</td>
<td>9.44</td>
<td>39.66</td>
<td>23.27</td>
<td>62.93</td>
</tr>
<tr>
<td>10</td>
<td>484</td>
<td>0.918</td>
<td>11.73</td>
<td>42.44</td>
<td>21.89</td>
<td>64.34</td>
</tr>
<tr>
<td>20</td>
<td>445</td>
<td>1.139</td>
<td>16.76</td>
<td>46.42</td>
<td>21.66</td>
<td>68.09</td>
</tr>
<tr>
<td>50</td>
<td>397</td>
<td>1.803</td>
<td>36.20</td>
<td>66.05</td>
<td>24.40</td>
<td>90.46</td>
</tr>
</tbody>
</table>

Results for Cube matrix. All parameters are fixed, expect the Ascending Low-Rank size
Results: test case

- Combining together Descending and Ascending Low-Rank we reach the best result

<table>
<thead>
<tr>
<th>Desc. rank</th>
<th>Asc. rank</th>
<th>iter</th>
<th>$\rho$</th>
<th>$T_{DLR}$</th>
<th>$T_{ALR}$</th>
<th>$T_P$</th>
<th>$T_S$</th>
<th>$T_T$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>562</td>
<td>0.807</td>
<td>8.09</td>
<td>21.45</td>
<td>57.88</td>
<td>29.55</td>
<td>87.43</td>
</tr>
<tr>
<td>1</td>
<td>10</td>
<td>228</td>
<td>1.604</td>
<td>8.41</td>
<td>45.83</td>
<td>82.60</td>
<td>13.20</td>
<td>95.79</td>
</tr>
<tr>
<td>1</td>
<td>20</td>
<td>212</td>
<td>2.490</td>
<td>8.77</td>
<td>92.26</td>
<td>130.01</td>
<td>13.87</td>
<td>143.88</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>393</td>
<td>1.006</td>
<td>10.79</td>
<td>23.08</td>
<td>62.35</td>
<td>22.25</td>
<td>84.59</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>163</td>
<td>1.803</td>
<td>10.89</td>
<td>48.41</td>
<td>87.75</td>
<td>10.37</td>
<td>98.13</td>
</tr>
<tr>
<td>10</td>
<td>20</td>
<td>147</td>
<td>2.689</td>
<td>12.01</td>
<td>93.58</td>
<td>134.93</td>
<td>10.63</td>
<td>145.56</td>
</tr>
<tr>
<td>20</td>
<td>1</td>
<td>357</td>
<td>1.227</td>
<td>16.93</td>
<td>26.31</td>
<td>72.42</td>
<td>22.11</td>
<td>94.53</td>
</tr>
<tr>
<td>20</td>
<td>10</td>
<td>145</td>
<td>2.025</td>
<td>16.91</td>
<td>54.96</td>
<td>101.03</td>
<td>9.84</td>
<td>110.87</td>
</tr>
<tr>
<td>20</td>
<td>20</td>
<td>133</td>
<td>2.910</td>
<td>15.62</td>
<td>91.69</td>
<td>135.93</td>
<td>9.89</td>
<td>145.82</td>
</tr>
</tbody>
</table>

Results for Cube matrix. Descending Low-Rank and Ascending Low-Rank are combined together.
Results: real world problems

We tested some matrices from the University of Florida Sparse Matrix Collection.

<table>
<thead>
<tr>
<th>Matrix</th>
<th>Size</th>
<th>Number of non-zeroes</th>
</tr>
</thead>
<tbody>
<tr>
<td>af_shell13</td>
<td>504,855</td>
<td>17,588,875</td>
</tr>
<tr>
<td>af_shell18</td>
<td>504,855</td>
<td>17,579,155</td>
</tr>
<tr>
<td>Emilia_923</td>
<td>923,136</td>
<td>40,373,538</td>
</tr>
<tr>
<td>Geo_1438</td>
<td>1,437,960</td>
<td>60,236,322</td>
</tr>
<tr>
<td>StocF_1465</td>
<td>1,465,137</td>
<td>21,005,389</td>
</tr>
</tbody>
</table>

Test matrices (from UF Sparse Matrix Collection)

<table>
<thead>
<tr>
<th>Matrix</th>
<th>Adaptive FSAI</th>
<th>MFLR</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( n_{it} )</td>
<td>( \rho )</td>
</tr>
<tr>
<td>af_shell13</td>
<td>963</td>
<td>0.89</td>
</tr>
<tr>
<td>af_shell18</td>
<td>1033</td>
<td>0.86</td>
</tr>
<tr>
<td>Emilia_923</td>
<td>1513</td>
<td>0.11</td>
</tr>
<tr>
<td>Geo_1423</td>
<td>491</td>
<td>0.31</td>
</tr>
<tr>
<td>StocF_1465</td>
<td>937</td>
<td>0.93</td>
</tr>
</tbody>
</table>

Computational performance of the adaptive FSAI and MFLR preconditioners for the test matrices (different configurations)
Results: strong scalability

With a Laplacian 300 x 300 x 300 we tested strong scalability

We used MARCONI cluster at the CINECA Center of High Performance Computing, Bologna. It has 1,512 nodes. Every node has 128 Gbyte of RAM memory and is equipped with 2 Intel Xeon E5-2697v4 Broadwell processors at 2.3 GHz with 36 cores.
Conclusions

- The use of FSAI in the framework of multilevel preconditioner may arise some difficulties, as indefinite Schur complements. With the MFLR preconditioner, Schur complements are ensured to be **positive definite**.

- The multilevel FSAI preconditioner is further enhanced by introducing Low-Rank corrections at both a local and a global level, namely **Descending** and **Ascending Low-Rank corrections**, respectively.

- Some theoretical properties and bounds on eigenvalues can be computed for the case with just two levels.

Further research

- To explore various degrees of sparsity in matrix products to reduce the cost of preconditioner set-up (matrix sparsification, matrix compression, etc.).
Thank you for your attention