A robust limited-memory incomplete Cholesky factorization

Jennifer Scott

STFC Rutherford Appleton Laboratory

Miroslav Tůma

Institute of Computer Science
Academy of Sciences of the Czech Republic

Sparse Days at CERFACS, 17th June 2013

Introduction

Consider the large sparse symmetric linear system

$$Ax = b, \; A \in R^{n \times n}$$

An ideal preconditioner should be:

- cheap to compute
- sparse and fast to apply
- provide sufficient approximation of the algebraic problem
- result in rapidly converging preconditioned iterative method

Key target for library software is robustness

Introduction

Incomplete Cholesky factorization

$$A \simeq LL^T$$

Some entries that occur in complete factorization are ignored.

Long history ($>$ 50 years) and many possible variants:

- Structure-based $IC(\ell)$: potential fill entries allowed only if their level of fill is less than ℓ.
- Threshold-based $IC(\tau)$: entries greater than τ dropped.
- Memory-based $IC(p)$: dropping of entries based on memory available.

$IC(\ell)$

- Location of permissible fill entries using sparsity pattern of A prescribed in advance.
- Aim to mimic how pattern of A is developed during complete factorization.
- But although entries of $E = A - LL^T$ are zero inside prescribed sparsity pattern, outside can be large.
- Increasing ℓ can be prohibitive (storage requirements and time to compute and apply the preconditioner).

$IC(\tau)$

- Entries of computed factors or intermediate quantities that exceed drop tolerance τ discarded.
- Success depends on suitable τ: highly problem dependent.
- Trade-off between sparsity and quality.
- Memory not predictable.

$IC(p)$

- Prescribe maximum number of entries allowed in each column of L and retain only largest entries.
- Memory predictable.
- Example is widely-used dual threshold $ILUT(p, \tau)$ (Saad '94).
  - Designed for non symmetric problems.
  - Combines use of drop tolerance τ with prescribed maximum column and row counts.
  - Ignores symmetry in A (if A symmetric, patterns of L and U^T normally different).

ICFS

ICFS code of Lin and Moré '99:

- Given p, retains $n_j + p$ largest entries in the lower triangular part of L_j, where n_j is number of entries in lower triangular part of A_j.
- Incorporates l_2-norm based scaling.
- In the event of breakdown, uses global diagonal shift ($A + \alpha I$ factorized for some $\alpha > 0$ (Manteuffel '80)).
- Widely used for large-scale trust region subproblems.

But, as we will see, efficiency of resulting preconditioner not very sensitive to choice of p.
So how to improve preconditioner quality?

Positive semi-definite modifications I

Alternative way to prevent breakdown:

- Diagonal modification scheme first introduced by Jennings and Malik '77,'78 (also Ajiz and Jennings '84).
- Every time off-diagonal entry discarded, corresponding diagonal entries modified by adding SPSD matrix

$$
\begin{array}{cc}
 & \begin{array}{ccccc} & i & & j & \end{array} \\
\begin{array}{c} \\ i \\ \\ j \\ \\ \end{array} &
\begin{pmatrix}
\ddots & & & & \\
 & |a_{ij}| & & -|a_{ij}| & \\
 & & \ddots & & \\
 & -|a_{ij}| & & |a_{ij}| & \\
 & & & & \ddots
\end{pmatrix}
\end{array}
$$

Jennings-Malik approach

- Breakdown-free factorization that can be expressed as

$$A = LL^T + E$$

  where error matrix E is sum of SPSD matrices.
- But modifications to A can be significant.
- Popular in some engineering applications.

Positive semi-definite modifications II

- More sophisticated modification scheme due to Tismenetsky '91 (and Kaporin '98).
- Introduces use of intermediate memory that is employed during construction of L but then discarded.
- Shown to be very robust but it "has unfortunately attracted surprisingly little attention" (Benzi '02).
- Suffers from a serious drawback: memory requirements can be prohibitively high.

Our aims

- Develop generalisation of ICFS such that efficiency of preconditioner improves with prescribed memory.
- Develop memory-efficient variant of Tismenetsky-Kaporin approach using global shifts to avoid breakdown.
- Combine in "black-box" *IC* factorization code that is demonstratively robust, efficient and flexible.

New package is `HSL_MI28`.

Tismenetsky approach

Based on matrix decomposition of form

$$A = LL^T + LR^T + RL^T + \hat{E}$$

- L is lower triangular with positive diagonal entries used for preconditioning,
- R is strictly lower triangular with small entries that is used to stabilise the factorization process, and
- $\hat{E}$ has the structure
$$\hat{E} = RR^T.$$

Tismenetsky approach

- On j-th step, decompose col. 1 of Schur complement S into

$$l_j + r_j \quad \text{with} \quad |l_j|^T |r_j| = 0,$$

  where entries of l_j are retained in incomplete factorization and those in r_j are discarded.

- On next step, S updated by subtracting

$$(l_j + r_j)(l_j + r_j)^T.$$

- Tismenetsky omits the term

$$\hat{E}_j = r_j r_j^T. \tag{1}$$

- Thus, SPSD matrix implicitly added to A.

Kaporin's use of drop tolerances

- Obvious choice for r_j are smallest off-diagonal entries in col j.
- Controls size of L but not memory required to compute it.
- Kaporin '98: entries of magnitude at least τ_1 kept in L and those smaller than τ_2 are dropped from R.
- Now $\hat{E}$ has structure

$$\hat{E} = RR^T + F + F^T,$$

F strictly lower triangular matrix that is not computed;
R used in computation of L but discarded.

Problems of Tismenetsky-Kaporin approach

- How to choose tolerances τ_1 and τ_2? Problem dependent.
- Method not guaranteed breakdown free ... combine with diagonal compensation or global shift.
- With no restriction on size of L and R, can achieve high quality preconditioner but memory demands high.
- Also too expensive. Impractical for the very large problems iterative methods designed for.

Remedy: impose memory limit on L and R.

Limited memory Tismenetsky-Kaporin approach

- `lsize`: max. number of fill entries in each col. of L

$$nz(L) \le nz(A) + \texttt{lsize} * (n-1)$$

- `rsize`: max. number of entries in each col. of R.
  Amount of intermediate memory and work involved in computing preconditioner depends on `rsize`.
  Note: if $\texttt{rsize} = 0$, R not used.

- Retain largest entries in l_j, provided at least τ_1 in magnitude; then retain next largest entries in r_j, provided at least τ_2 in magnitude.

Left-looking algorithm outline

Input: A, `lsize`, `rsize`, τ_1, τ_2

Set $w(1:n) = 0$
for $j = 1 : n$ **do**
  Scatter col. A_j into w
  Apply $LL^T + RL^T + LR^T$ updates from columns $1 : j-1$ to w
  (Partially) sort entries in w by magnitude
  Keep $n_j +$ `lsize` entries of largest magnitude in l_j provided
    they are at least τ_1
  Keep `rsize` additional entries that are next largest in magnitude
    in r_j provided they are at least τ_2
  Reset entries of w to zero
  end do
end do

Output: L

Coping with breakdown

- When using limited memory (and/or dropping), factorization may breakdown.
- We hold a copy of diagonal entries and, at each step j, keep them updated. If any becomes zero or negative, restart factorization with

$$A \leftarrow A + \alpha I$$

  for some positive α.
- More than one restart may be required.

Test environment

- Problems from University of Florida Collection.
- Selected all non-diagonal SPD matrices with $n > 1000$.
- Removed those with duplicate sparsity patterns.
- Following initial experiments, 8 problems discarded as unable to achieve convergence without large amount of fill.
- Test set of 145 problems.
- CG used with $x_0 = 0$, b computed so that $x = 1$, and stopping criteria

  $$\|Ax_k - b\| \leq 10^{-10}\|b\|$$

  with limit of 2000 iterations.

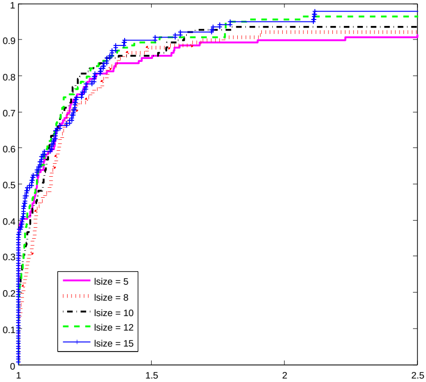
Test environment (continued)

- What to measure? iteration counts? timings? sparsity of L?
- We define the efficiency of preconditioner to be

$$iter \times nz(L)$$

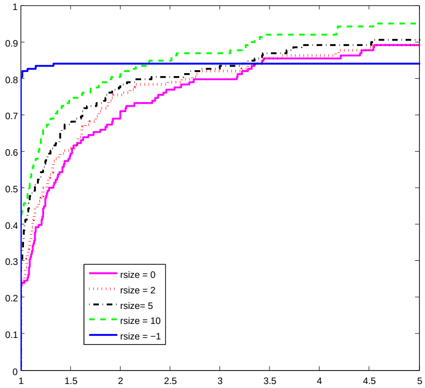
- Performance profiles (Moré, Dolan '02) used to assess performance.
- All software written in Fortran.

Efficiency performance profile, `rsize`=0



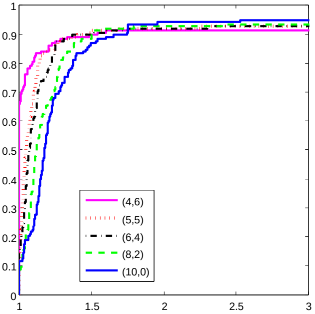
Note: rather insensitive to choice of `lsize` (ICFS).

Efficiency (= iteration) performance profile, `lsize`=5



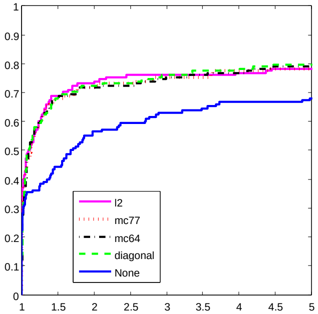
`rsize`=`-1` is unlimited memory for R (not practical).

Efficiency performance profile `lsize+rsize` constant



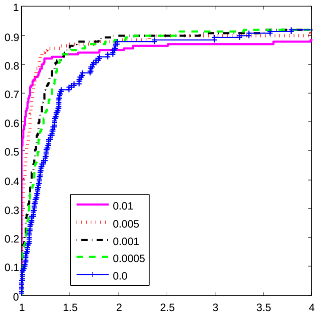
Pairs (`lsize`,`rsize`)
Intermediate memory (`rsize` > 0) can compensate for `lsize`.

Effect of scaling on efficiency (`lsize` = `rsize` = 5)



`HSL_MI28` default is l_2 scaling.

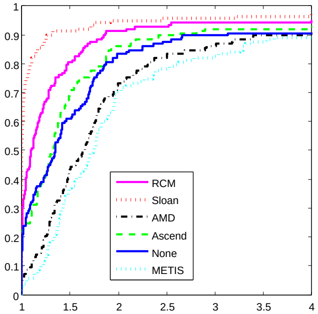
Effect of dropping on efficiency ($\texttt{lsize} = \texttt{rsize} = 5$)



Often advantageous to use small drop tolerance.

Default $\tau_1 = 0.001$.

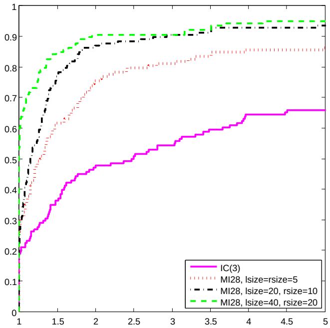
Effect of ordering on efficiency

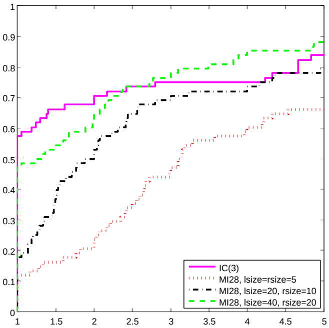


Sloan profile-reduction ordering is the winner.

Comparison with level-based approach ($IC(3)$)

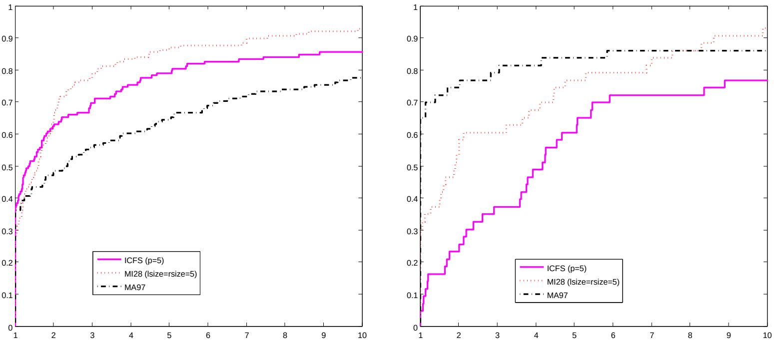
Efficiency (left) and iterations (right).





Comparison with direct solver HSL_MA97

Total time: all problems (left) and large problems (right).



HSL_MI28 can sometimes compete with direct solver
(and succeeds when HSL_MA97 runs out of memory).

Concluding remarks

- We have developed a new *IC* code HSL_MI28 that may be used as a "black box" or tuned for a particular problem.
- Memory usage is under the user's control.
- Using restricted intermediate memory improves efficiency.
- The intermediate memory can compensate for the preconditioner size.
- Based on extensive experimentation, HSL_MI28 appears robust and efficient.

Note: at the Preconditioning Conference, my talk will focus more on the use of positive semidefinite modification schemes.

Thank you!

HSL_MI28 is available (without charge) as part of HSL 2013.

Technical Reports RAL-P-2013-004 and RAL-P-2013-005.

Supported by EPSRC grant EP/I013067/1
Grant Agency of the Czech Republic Project No. P201/13-06684