Solving Symmetric Indefinite Linear Systems with a Sherman-Morrison-Woodbury-based Algorithm

Kam Chuen (Alex) Tung

The Chinese University of Hong Kong

30 August 2018

Table of Contents

Problem Background

- Solve $Ax = b$. That's it.

Problem Background

- ~~Solve $Ax = b$. That's it.~~
- Given a linear system $Ax = b$, where A is:
  - sparse,
  - symmetric,
  - indefinite, and
  - nonsingular,

  find an **accurate** and **efficient** way to solve the system.

Problem Background

- ~~Solve $Ax = b$. That's it.~~
- Given a linear system $Ax = b$, where A is:
  - sparse,
  - symmetric,
  - indefinite, and
  - nonsingular,

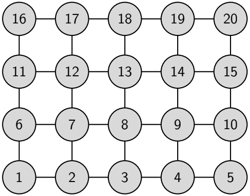
  find an **accurate** and **efficient** way to solve the system.
- Accuracy is measured by the ∞-norm relative error:

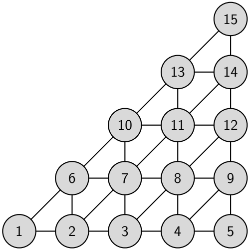
$$\epsilon_{rel} := \frac{\|\widetilde{x} - x\|_\infty}{\|x\|_\infty},$$

where $\widetilde{x}$ is the solution obtained by the solver.

Why sparse?

- Linear systems from many applications are sparse





- General dense solvers run in $O(n^3)$ and scale badly
- Sparse solvers: solvers that exploit sparsity

Exploiting Symmetry

- Symmetric matrices are simpler for factorization-based solvers.
- Generally:

$$A = LDU,$$

  where:
  - L is lower triangular,
  - D is diagonal, and
  - U is upper triangular.
- For symmetric A, it becomes:

$$A = LDL^T.$$

Focusing on indefinite matrices

- For (positive- or negative-) definite matrices, Cholesky factorization works well
- The indefinite case is more interesting!

Focusing on nonsingular matrices

- If A is singular, there can be infinitely many solutions!
- Even if A is near-singular, it is hard to measure accuracy...
- $\|Av\|_\infty$ can be small even if $\|v\|_\infty$ is large

Focusing on nonsingular matrices

- If A is singular, there can be infinitely many solutions!
- Even if A is near-singular, it is hard to measure accuracy...
- $\|Av\|_\infty$ can be small even if $\|v\|_\infty$ is large

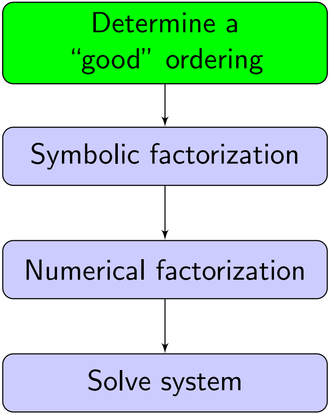
- There is a reason why the residual

$$r_{rel} := \frac{\|b - A\widetilde{x}\|_\infty}{\|b\|_\infty},$$

is **not** used to measure performance. More on that later.

General Framework

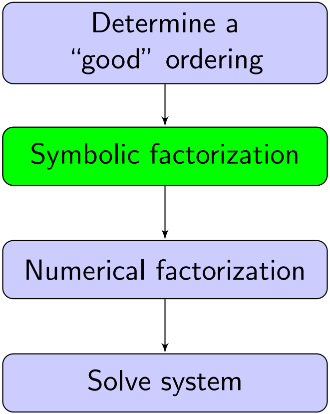
Aim: Solve $Ax = b$ based on LDL^T factorization.



- Find ordering that minimizes the number of nonzeros of L (denoted by $|L|$)

General Framework

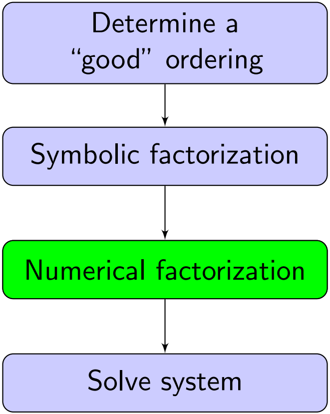
Aim: Solve $Ax = b$ based on LDL^T factorization.



- Perform symbolic factorization to determine data structure

General Framework

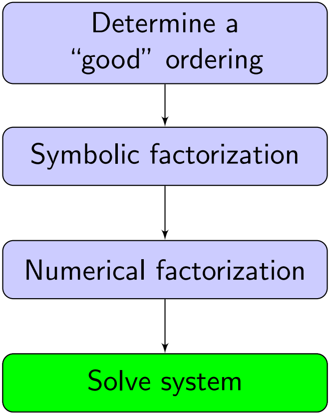
Aim: Solve $Ax = b$ based on LDL^T factorization.



- Avoid pivoting to utilize the fixed data structure
- Factorizing SPD matrices is stable without pivoting
- Factorizing indefinite matrices may fail without pivoting!

General Framework

Aim: Solve $Ax = b$ based on LDL^T factorization.



- Triangular solves only
- Possibly with iterative refinement

Previous Work

- **[Bunch, Kaufman]**: Use 1x1 and 2x2 pivots
  - Does not exploit sparsity
- **[Duff, Reid]**: Multifrontal Method
  - Sparse solver
  - Uses idea from Bunch-Kaufman to handle indefinite case
- **[Li, Demmel]**: Change the value of pivot if it is too small
  - Works for nonsymmetric A
- **[Egidi, Maponi]**: Use Sherman-Morrison formula to update solution
  - Breaks system into rank-1 components
  - Does not exploit sparsity

Overview

Properties:

- Left-looking LDL^T factorization
- Avoids pivoting

Overview

Properties:

- Left-looking LDL^T factorization
- Avoids pivoting

Key Idea:

- If pivot too small, change it and *record the change*
- In the end, we get LDL^T factorization of $B := A + UCU^T$
- C is a $k \times k$ diagonal matrix storing the changes
- U is $n \times k$; maps C from $\mathbb{R}^{k \times k}$ back to $\mathbb{R}^{n \times n}$
- Use Sherman-Morrison-Woodbury formula to compute $A^{-1}b$

The Sherman-Morrison-Woodbury (SMW) Formula

- From last slide, we have

$$A = B - UCU^T,$$

where A, B are $n \times n$, U is $n \times k$, and C is $k \times k$.

The Sherman-Morrison-Woodbury (SMW) Formula

- From last slide, we have

$$A = B - UCU^T,$$

where A, B are $n \times n$, U is $n \times k$, and C is $k \times k$.

- Sherman-Morrison formula deals with $k = 1$ (rank-1 update):

$$A^{-1} = B^{-1} + \frac{B^{-1}uu^T B^{-1}}{c^{-1} - u^T B^{-1} u}$$

The Sherman-Morrison-Woodbury (SMW) Formula

- From last slide, we have

$$A = B - UCU^T,$$

where A, B are $n \times n$, U is $n \times k$, and C is $k \times k$.

- Woodbury formula deals with the general case:

$$A^{-1} = B^{-1} + B^{-1}UW^{-1}U^T B^{-1},$$

where

$$W := C^{-1} - U^T B^{-1} U$$

is the “Woodbury matrix”.

Proof of SMW Formula

- Based on blockwise matrix inversion
- A^{-1} is the solution X of the matrix equation

$$\begin{pmatrix} B & U \\ U^T & C^{-1} \end{pmatrix} \begin{pmatrix} X \\ Y \end{pmatrix} = \begin{pmatrix} I \\ O \end{pmatrix}$$

Proof of SMW Formula

- Based on blockwise matrix inversion
- A^{-1} is the solution X of the matrix equation

$$\begin{pmatrix} B & U \\ U^T & C^{-1} \end{pmatrix} \begin{pmatrix} X \\ Y \end{pmatrix} = \begin{pmatrix} I \\ O \end{pmatrix}$$

- Solving

$$\begin{cases} BX + UY &= I \\ U^T X + C^{-1} Y &= O \end{cases}$$

gives

$$\begin{aligned} X &= B^{-1}(I - UY) \\ \implies Y &= -(C^{-1} - U^T B^{-1} U)^{-1} U^T B^{-1} \\ \implies X &= B^{-1} + B^{-1} U (C^{-1} - U^T B^{-1} U)^{-1} U^T B^{-1} \end{aligned}$$

Factorization Step — Algorithm I

$sigma \leftarrow 10^{-3}$

$threshold \leftarrow 10^{-4}$ ▷ Parameter values are changeable

$nchanges \leftarrow 0$

$L \leftarrow tril(A)$ ▷ Lower triangular part of A

for $i \leftarrow 1$ to n **do** ▷ Currently on i-th column

  for j: $1 \leq j < i$, $l_{ij} \neq 0$ **do** ▷ **Elimination step**

    $\lambda \leftarrow d_{jj} \times l_{ij}$

    for k: $i \leq k \leq n$ **do**

      $l_{ki} \leftarrow l_{ki} - \lambda \times l_{kj}$

    end for

  end for

Factorization Step — Algorithm II

$\alpha \leftarrow l_{ii}$
if $|\alpha| < \textit{threshold}$ **then** ▷ **Change pivot value**
  if $\alpha < 0$ **then**
    $t \leftarrow -\textit{sigma}$
  else
    $t \leftarrow \textit{sigma}$
  end if
  $\textit{nchanges} \leftarrow \textit{nchanges} + 1$
  $\textit{changes}[\textit{nchanges}] \leftarrow t - \alpha$ ▷ Record change value
  $\textit{locs}[\textit{nchanges}] \leftarrow i$ ▷ Record change index
  $\alpha \leftarrow t$
end if

Factorization Step — Algorithm III

$d_{jj} \leftarrow \alpha$, $l_{jj} \leftarrow 1$ ▷ **Update D and L**

for $j \leftarrow i + 1$ to n **do**

  $l_{ji} \leftarrow \frac{l_{ji}}{\alpha}$

end for

end for

Now what?

- In the end, we obtain the LDL^T factorization of B, where B differs from A in *nchanges* diagonal entries.
- *changes*[ ] and *locs*[ ] can be used to form U and C, such that $B = A + UCU^T$.

Forming U and C

- Let $k := nchanges$. Given $changes[\,]$ and $locs[\,]$.
- Then C is just a $k \times k$ diagonal matrix with $c_{ii} = changes[i]$.
- U is a $n \times k$ binary matrix with $u_{ij} = 1 \iff locs[j] = i$.

Forming U and C

- Let $k := nchanges$. Given $changes[\,]$ and $locs[\,]$.
- Then C is just a $k \times k$ diagonal matrix with $c_{ii} = changes[i]$.
- U is a $n \times k$ binary matrix with $u_{ij} = 1 \iff locs[j] = i$.

For example,

if $n = 5$, $changes[\,] = [-0.1, 1.2, 1.0]$, $locs[\,] = [1, 4, 5]$, then

$$C = \begin{pmatrix} -0.1 & 0 & 0 \\ 0 & 1.2 & 0 \\ 0 & 0 & 1.0 \end{pmatrix} \text{ and } U = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

Solution Step — Algorithm

Let $SMW(b)$ be a subroutine that, given $B = LDL^T$ and $A = B - UCU^T$, computes $A^{-1}b$, using:

- triangular solves, and
- builtin algorithm for computing $W^{-1} = (C^{-1} - U^T B^{-1} U)^{-1}$.

Solution Step — Algorithm

Let $SMW(b)$ be a subroutine that, given $B = LDL^T$ and $A = B - UCU^T$, computes $A^{-1}b$, using:

- triangular solves, and
- builtin algorithm for computing $W^{-1} = (C^{-1} - U^T B^{-1} U)^{-1}$.

Here is the solution step:

1: Form U and C from $changes[\,]$ and $locs[\,]$
2: $x \leftarrow SMW(b)$

Solution Step — Algorithm 2.0

To improve accuracy, iterative refinement (IR) with *extended precision* is used.

Solution Step — Algorithm 2.0

To improve accuracy, iterative refinement (IR) with *extended precision* is used.

Let $SMW128(b)$ be a subroutine that, given $B = LDL^T$ and $A = B - UCU^T$, computes $A^{-1}b$, using:

- triangular solves, and
- builtin algorithm for computing W^{-1}, **to 128-bit precision**.

Here is the solution step:

Form U and C from $changes[\,]$ and $locs[\,]$
$x \leftarrow \mathbf{0}$
$residual \leftarrow \frac{\|Ax-b\|_\infty}{\|b\|_\infty}$
$tolerance \leftarrow 10^{-16}$, $maxit \leftarrow 10$ ▷ Parameters, changeable
$numit \leftarrow 0$
while $numit < maxit$ **do**
  $r \leftarrow b - Ax$
  $correction \leftarrow SMW128(r)$ ▷ Using extended precision
  $x \leftarrow x + correction$
  $newresidual \leftarrow \frac{\|Ax-b\|_\infty}{\|b\|_\infty}$
  if $newresidual < tolerance$ **or** $2 \cdot newresidual > residual$ **then**
    break
  end if
  $residual \leftarrow newresidual$
  $numit \leftarrow numit + 1$
end while

Computing $SMW(b)$ and $SMW128(b)$

For $SMW(b)$,

1: $v \leftarrow L^T \setminus D \setminus L \setminus b$ ▷ $v = B^{-1}b$
2: $Y \leftarrow L^T \setminus D \setminus L \setminus U$ ▷ $Y = B^{-1}U$
3: $W \leftarrow C^{-1} - U^T Y$ ▷ Forming Woodbury matrix
4: $z \leftarrow W \setminus (U^T v)$
5: $x \leftarrow v + Yz$

For $SMW128(b)$,

1: $v \leftarrow L^T \setminus D \setminus L \setminus b$ ▷ $v = B^{-1}b$
2: $Y \leftarrow L^T \setminus D \setminus L \setminus U$ ▷ $Y = B^{-1}U$
3: $W \leftarrow mp(C)^{-1} - mp(U)^T mp(Y)$ ▷ Forming W in 128-bit
4: $z \leftarrow W \setminus (mp(U)^T mp(v))$
5: $x \leftarrow v + Yz$

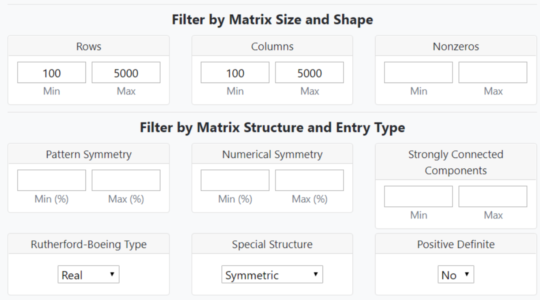
Here, $mp()$ converts a matrix to 128-bit precision.

Software and Tools

- MATLAB R2018a Student License
- Advanpix Multiprecision Computing Toolbox, 7-day trial license

Test Matrices

- 78 matrices from SuiteSparse (https://sparse.tamu.edu/)
- Selection criteria:



- Manually removed matrices that are:
  - singular, or
  - "not meant to be solved" (e.g. random graphs)

Test Matrices

- For each SuiteSparse matrix A, a random symmetric matrix with the *same nonzero patterns* and fixed (approximate) condition number *cond* is generated
- Command used: `sprandsym(A, [], cond, 3)`

Parameter Overview

Parameter	Description
threshold	Pivots smaller than *threshold* is too small
sigma	Small thresholds will be changed to $\pm sigma$
tolerance	IR should stop if residual smaller than *tolerance*
maxit	Maximum number of IR steps
Use 128 bit?	Whether $SMW128()$ is used in lieu of $SMW()$

Notice that $maxit = 1$ is equivalent to not using IR.

Parameter Choice

A total of $1 \times 2 \times 2 \times 7 = 28$ parameter sets are tested.

- $threshold = 10^{-16}$
- $maxit = 1$ (no IR) or $maxit = 10$ (with IR)
- Use $SMW()$ or $SMW128()$
- As for *threshold* and *sigma*,

No.	*threshold*	*sigma*
1	10^{-3}	10^{-3}
2	10^{-4}	10^{-3}
3	10^{-6}	10^{-6}
4	10^{-9}	10^{-9}
5	$10^{-8} \times \|A\|$	$10^{-8} \times \|A\|$
6	$10^{-12} \times \|A\|$	$10^{-12} \times \|A\|$
7	$10^{-16} \times \|A\|$	$10^{-16} \times \|A\|$

The Competition

1. Bunch-Kaufman
   - The default MATLAB full matrix solver in our case
2. MA57 algorithm
   - Multifrontal method
   - With scaling and pivoting
   - The default MATLAB sparse matrix solver in our case

For fairness, we test the algorithms both with and without IR.

Comparing the Algorithms

- Relative residual is used for IR terminating condition
- Compare **relative error** instead
- Performance profile (**[Dolan, More]**) is used for visualizing the results

Performance Profile

- Suppose there are X algorithms and T tests.

Performance Profile

- Suppose there are X algorithms and T tests.
- The i-th algorithm gives relative error ϵ_{ij} on test j.

Performance Profile

- Suppose there are X algorithms and T tests.
- The i-th algorithm gives relative error ϵ_{ij} on test j.
- If "fail", ϵ_{ij} is set to ∞.

Performance Profile

- Suppose there are X algorithms and T tests.
- The i-th algorithm gives relative error ϵ_{ij} on test j.
- If "fail", ϵ_{ij} is set to ∞.
- For $1 \leq j \leq T$, set $best_j := \min_i \epsilon_{ij}$.

Performance Profile

- Suppose there are X algorithms and T tests.
- The i-th algorithm gives relative error ϵ_{ij} on test j.
- If "fail", ϵ_{ij} is set to ∞.
- For $1 \leq j \leq T$, set $best_j := \min_i \epsilon_{ij}$.
- Set $ratio_{ij} := \frac{\epsilon_{ij}}{best_j}$. (Assume $\frac{\infty}{\infty} = \infty$)

Performance Profile

- Suppose there are X algorithms and T tests.
- The i-th algorithm gives relative error ϵ_{ij} on test j.
- If "fail", ϵ_{ij} is set to ∞.
- For $1 \leq j \leq T$, set $best_j := \min_i \epsilon_{ij}$.
- Set $ratio_{ij} := \frac{\epsilon_{ij}}{best_j}$. (Assume $\frac{\infty}{\infty} = \infty$)
- For each algorithm i, plot the cumulative frequency of the data $\log_{10}(ratio_{i1}), \log_{10}(ratio_{i2}), \ldots, \log_{10}(ratio_{iT})$.

Failure Condition

Say the algorithm fails, if at least one of the following happens:

1. Relative error is too big
2. *nchanges* is too big (for SMW-based algorithms only)
   - Need to take inverse of W, where $dim(W) = nchanges$
   - $nchanges \approx n \rightarrow$ forced to solve a huge dense system!
3. Runtime is too long
   - For our test matrices, $n \leq 5000$
   - Consider > 3 minutes as too long

Part 1: Effect of improving the SMW algorithm

- First, we compare SMW algorithms with:
  1. No IR, no 128-bit
  2. IR, no 128-bit
  3. IR, 128-bit
- Parameters:
  - $\textit{threshold} = 10^{-4}$, $\textit{sigma} = 10^{-3}$
  - $\textit{tolerance} = 10^{-16}$
  - $\textit{maxit} = 1$ (no IR) or $\textit{maxit} = 10$ (with IR)
- Fail conditions:
  - Relative error > 1
  - Runtime > 3 minutes
- Test matrices: 78 SuiteSparse matrices

Figure 1-1: No IR, no 128-bit

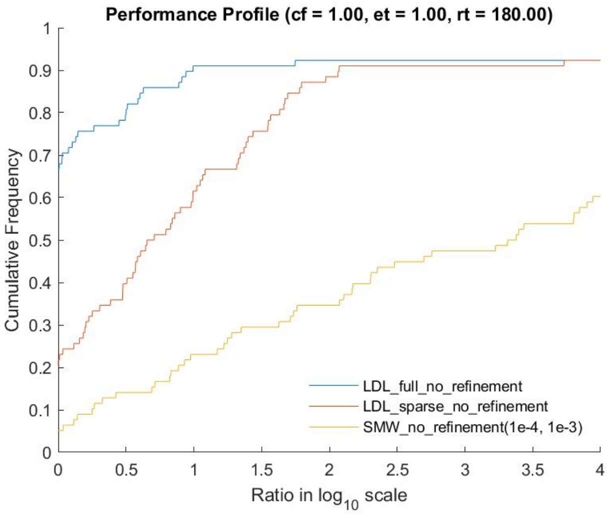


Figure 1-2: IR, no 128-bit

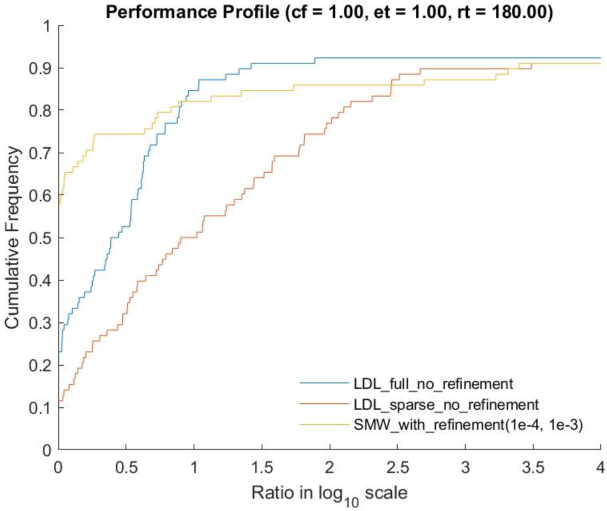


Figure 1-3: IR, 128-bit

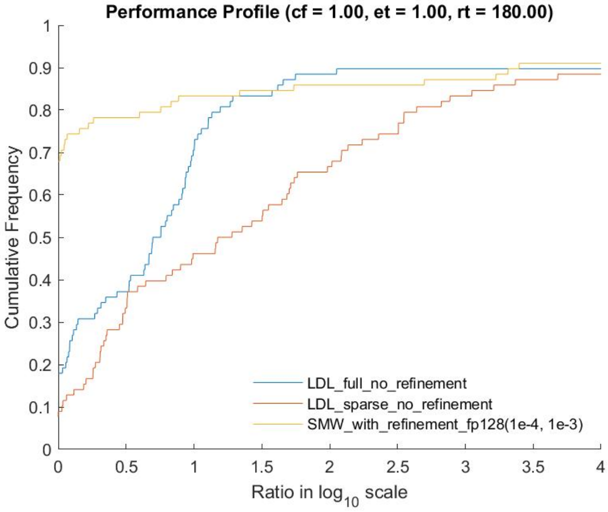


Figure 1-4: IR, 128-bit, versus LDL with IR

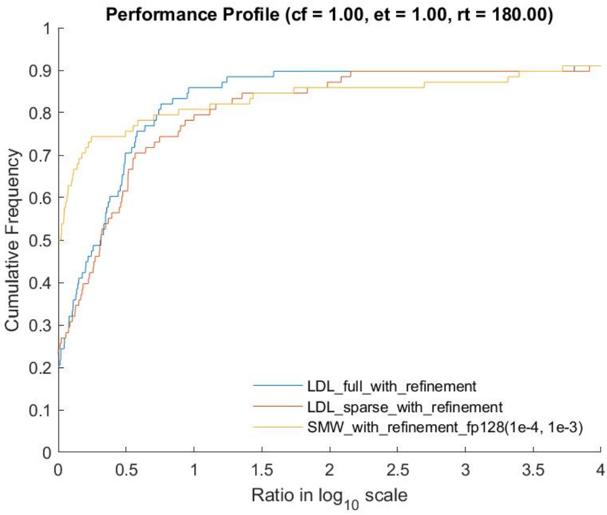
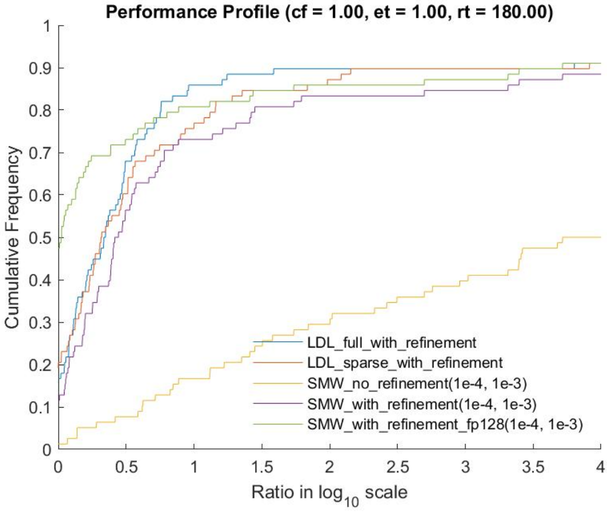


Figure 1-5: All three settings, versus LDL with IR



Part 2: Comparing different *sigma* and *threshold*

- Parameters:
  - $tolerance = 10^{-16}$
  - $maxit = 10$ (with IR)
  - Use $SMW128()$ whenever applicable
- Choose different values of *threshold* and *sigma*
- Fail conditions:
  - Relative error > 1
  - Runtime > 3 minutes
- Test matrices: 78 SuiteSparse matrices

Figure 2-1: Constant values

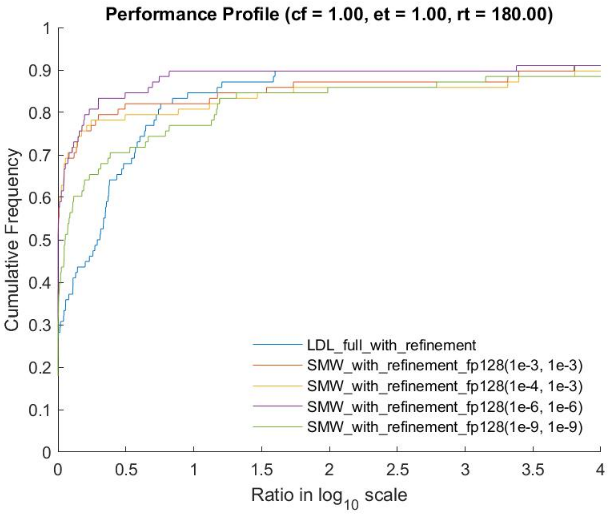


Figure 2-2: Nonconstant values (depends on $\|A\|_\infty$)

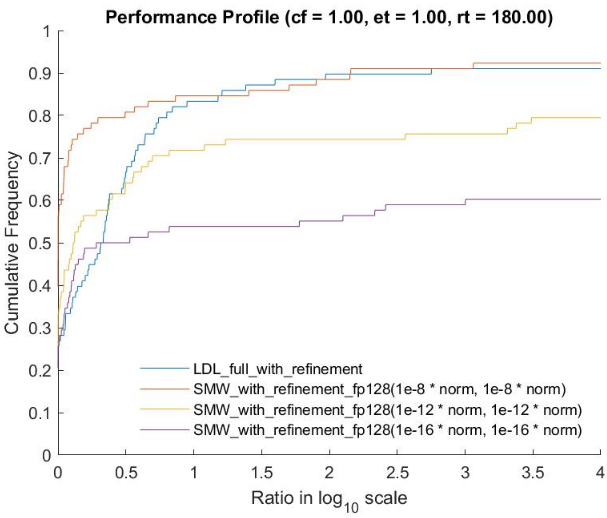
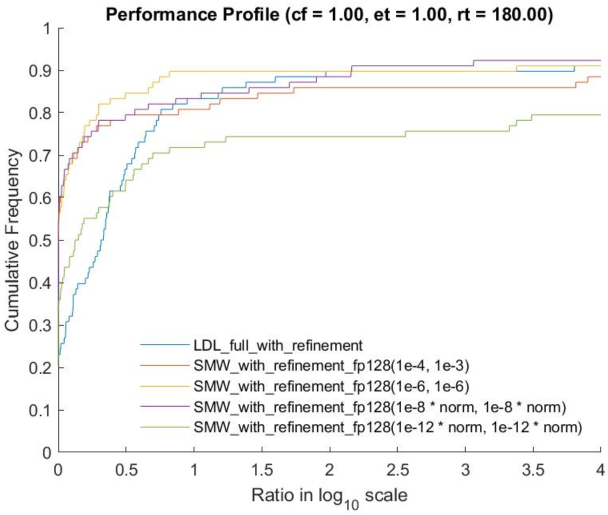


Figure 2-3: Some constant & some nonconstant



Part 3: Taking *nchanges* into account

- Parameters:
  - $tolerance = 10^{-16}$
  - $maxit = 10$ (with IR)
  - Use $SMW128()$ whenever applicable
- Choose different values of *threshold* and *sigma*
- Fail conditions:
  - Relative error > 1
  - Runtime > 3 minutes
  - **(NEW!)** $\frac{nchanges}{n} > cf$, where $cf \in \{0.1, 0.25, 0.5, 1.0\}$
- Test matrices: 78 SuiteSparse matrices

Figure 3-1: $cf = 1.0$

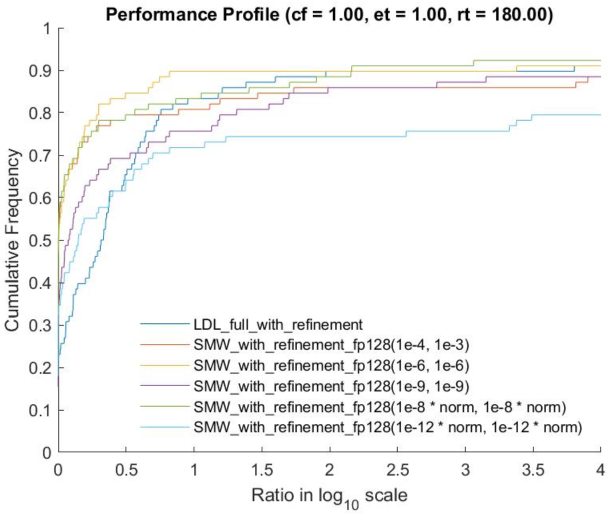


Figure 3-2: $cf = 0.5$

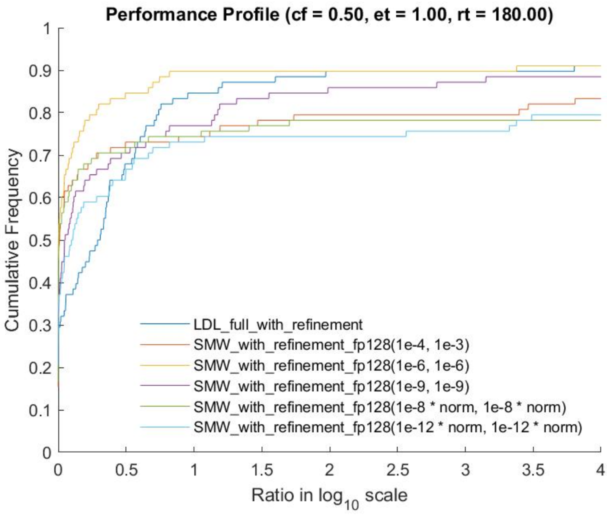


Figure 3-3: $cf = 0.25$

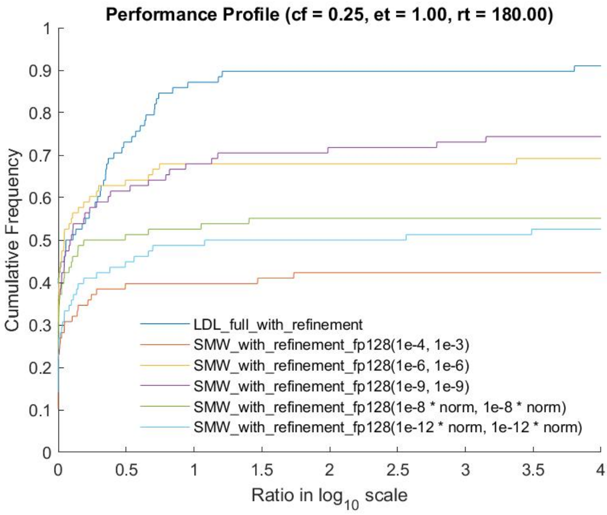
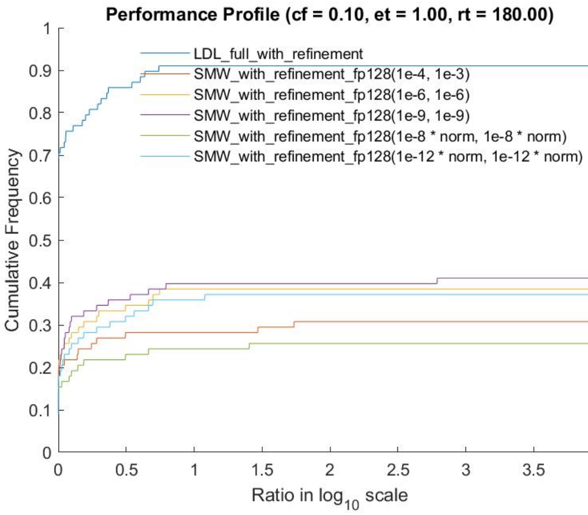


Figure 3-4: $cf = 0.1$



Part 4: Testing on `sprandsym()` Matrices

- We shall repeat the previous parts on matrices generated using `sprandsym(A, [], cond, 3)` command.
- Choices of *cond*: 10^8, 10^{10}.

Figure 4-1-1: *cond* $\approx 10^8$, focus on IR / 128-bit

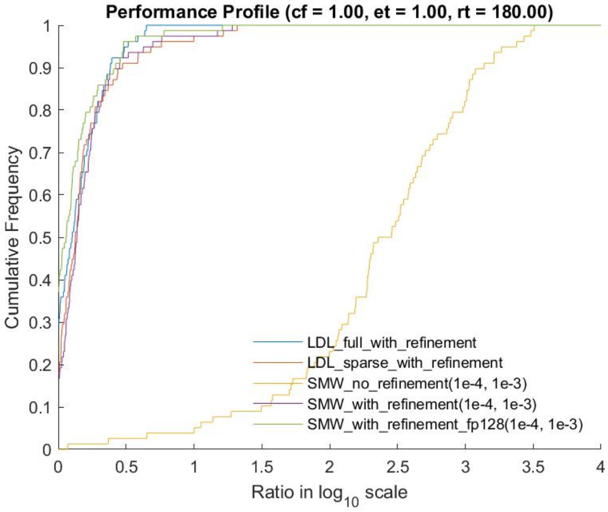


Figure 4-1-2: *cond* $\approx 10^{10}$, focus on IR / 128-bit

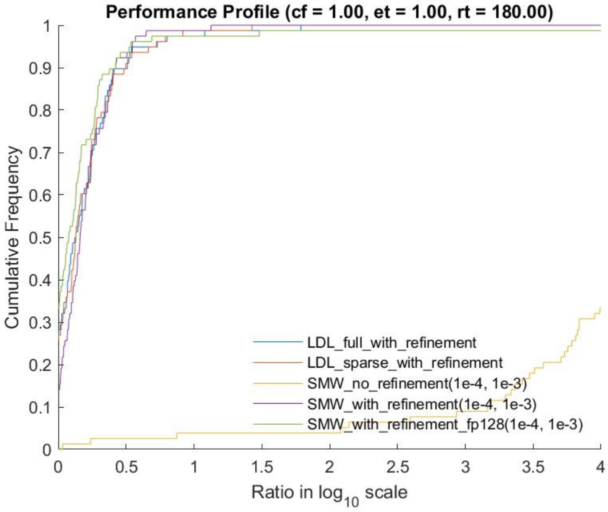


Figure 4-2: *cond* $\approx 10^{10}$, focus on *sigma* and *threshold*

- Notice that the choice of paramters matters little!
- Same for *cond* $\approx 10^8$ and other parameter choices

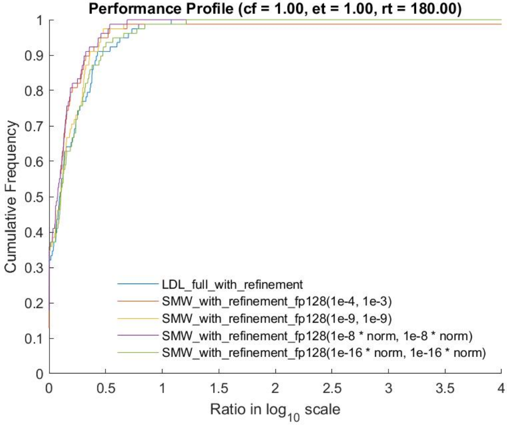


Figure 4-3-1: $cond \approx 10^{10}$, $cf = 1.0$, focus on $\frac{nchanges}{n}$

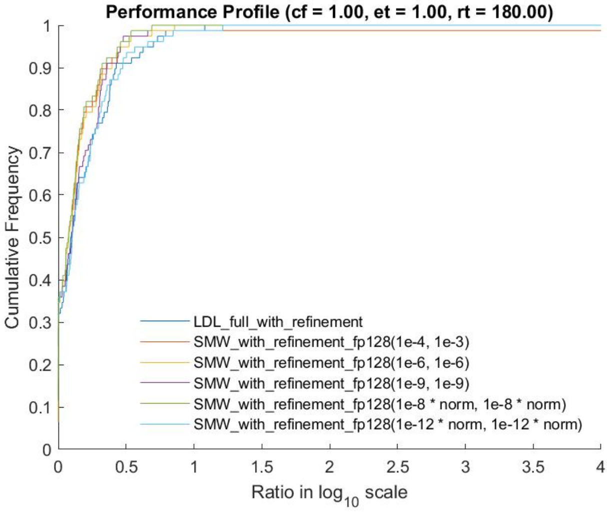


Figure 4-3-2: $cond \approx 10^{10}$, $cf = 0.5$, focus on $\frac{nchanges}{n}$

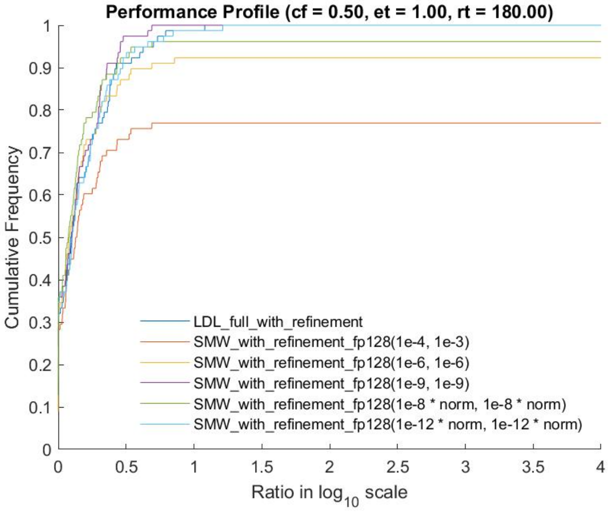


Figure 4-3-3: $cond \approx 10^{10}$, $cf = 0.25$, focus on $\frac{nchanges}{n}$

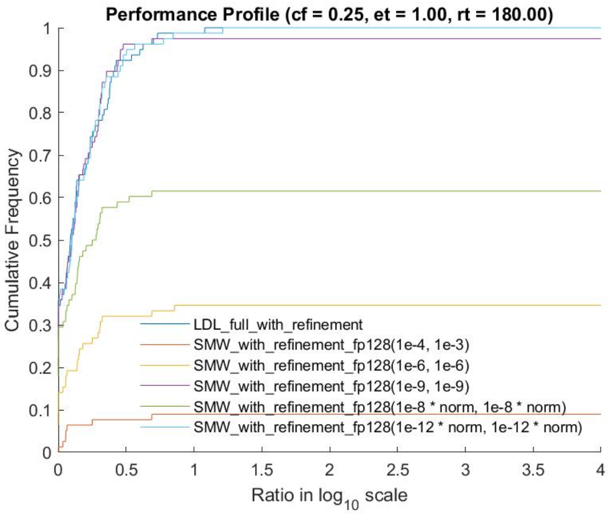
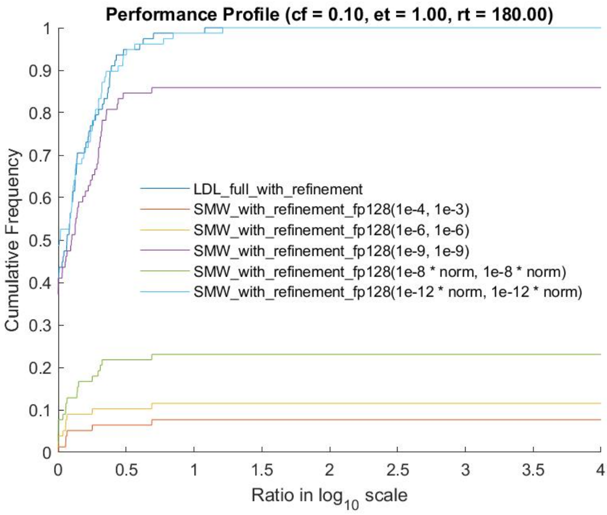


Figure 4-3-4: $cond \approx 10^{10}$, $cf = 0.1$, focus on $\frac{nchanges}{n}$



Part 5: The effect of *tolerance*

- What if *tolerance* $= 10^{-14}$, instead of 10^{-16}?

Figure 5-1-1: *tolerance* $= 10^{-16}$, SuiteSparse

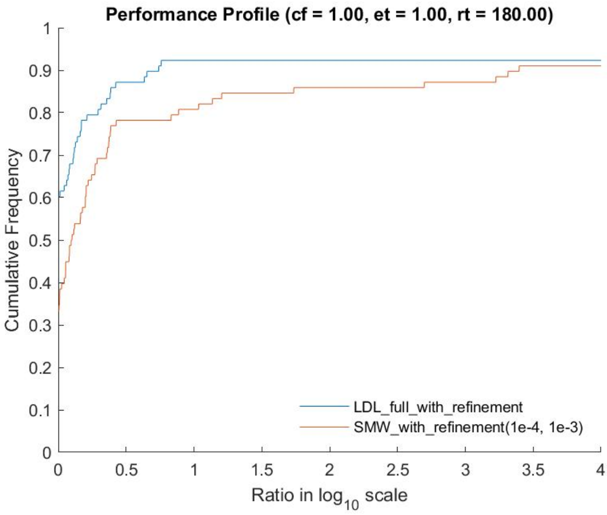


Figure 5-1-2: *tolerance* $= 10^{-14}$, SuiteSparse

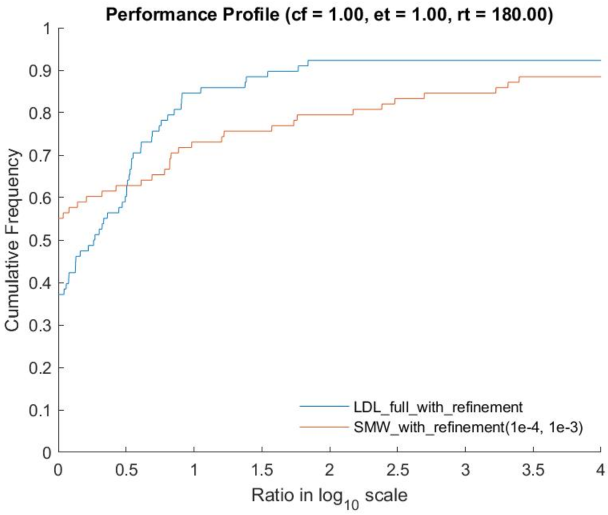


Figure 5-2-1: *tolerance* $= 10^{-16}$, sprandsym, *cond* $\approx 10^{10}$

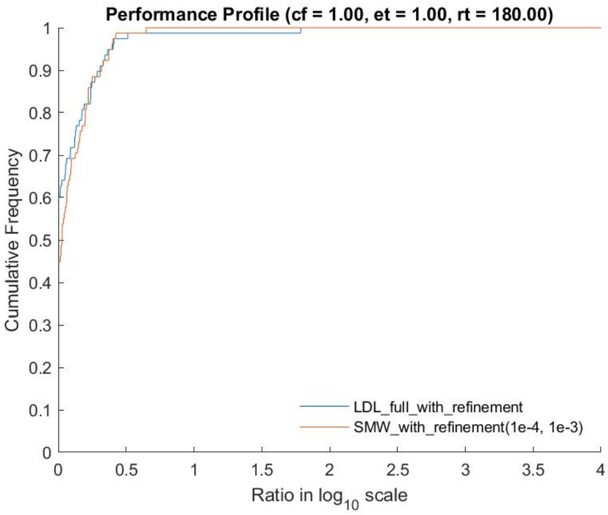
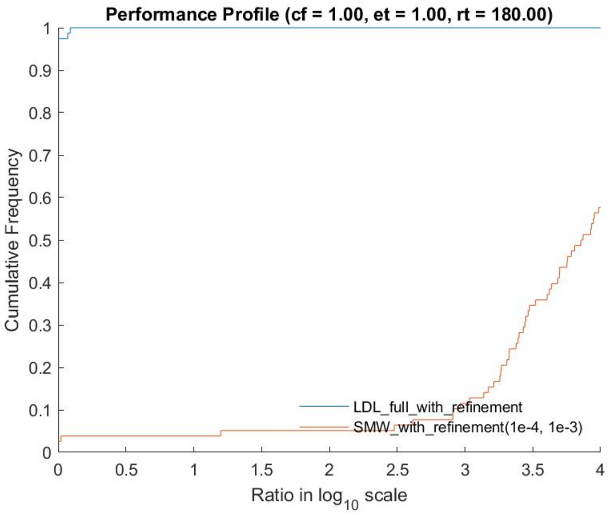


Figure 5-2-2: $tolerance = 10^{-14}$, sprandsym, $cond \approx 10^{10}$



Summary

What does each part tells us?

1. SMW with IR performs competitively against built-in algorithms. $SMW128()$ is nice to have, but not strictly necessary.

Summary

What does each part tells us?

1. SMW with IR performs competitively against built-in algorithms. $SMW128()$ is nice to have, but not strictly necessary.
2. Smaller *sigma* and *threshold* $\implies$ slightly worse accuracy.

Summary

What does each part tells us?

1. SMW with IR performs competitively against built-in algorithms. $SMW128()$ is nice to have, but not strictly necessary.
2. Smaller *sigma* and *threshold* $\implies$ slightly worse accuracy.
3. Smaller *sigma* and *threshold* $\implies$ slightly smaller *nchanges*.

Summary

What does each part tells us?

1. SMW with IR performs competitively against built-in algorithms. $SMW128()$ is nice to have, but not strictly necessary.
2. Smaller *sigma* and *threshold* $\implies$ slightly worse accuracy.
3. Smaller *sigma* and *threshold* $\implies$ slightly smaller *nchanges*.
4. On `sprandsym()` matrices:
   - Parameters do not affect accuracy by much;
   - Parameters do affect *nchanges* significantly.

Summary

What does each part tells us?

1. SMW with IR performs competitively against built-in algorithms. $SMW128()$ is nice to have, but not strictly necessary.
2. Smaller *sigma* and *threshold* $\implies$ slightly worse accuracy.
3. Smaller *sigma* and *threshold* $\implies$ slightly smaller *nchanges*.
4. On `sprandsym()` matrices:
   - Parameters do not affect accuracy by much;
   - Parameters do affect *nchanges* significantly.
5. Choice of *tolerance* may have a tremendous impact on relative performance.

Future Work

1. Parameters
   - ✗ Only a small subset tested
   - ✓ Test more parameters to find the best parameter?

Future Work

1. Parameters
   - ✗ Only a small subset tested
   - ✓ Test more parameters to find the best parameter?
2. Data Representation
   - ✗ Some matrices may be under-represented in performance profiles
   - ✓ Combine with alternative methods for data representation?

Future Work

1. Parameters
   - ✗ Only a small subset tested
   - ✓ Test more parameters to find the best parameter?
2. Data Representation
   - ✗ Some matrices may be under-represented in performance profiles
   - ✓ Combine with alternative methods for data representation?
3. Evaluation Metric
   - ✗ CPU time using $SMW128()$ is ~ 10 times that of $SMW()$
   - ✓ More comprehensive metric needed?

Future Work

1. Parameters
   - ✗ Only a small subset tested
   - ✓ Test more parameters to find the best parameter?
2. Data Representation
   - ✗ Some matrices may be under-represented in performance profiles
   - ✓ Combine with alternative methods for data representation?
3. Evaluation Metric
   - ✗ CPU time using $SMW128()$ is ~ 10 times that of $SMW()$
   - ✓ More comprehensive metric needed?
4. In-depth Analysis
   - ✗ The conclusions are based on empirical evidence
   - ✓ Find provable error bounds?

Future Work

1. Parameters
   - ✗ Only a small subset tested
   - ✓ Test more parameters to find the best parameter?
2. Data Representation
   - ✗ Some matrices may be under-represented in performance profiles
   - ✓ Combine with alternative methods for data representation?
3. Evaluation Metric
   - ✗ CPU time using $SMW128()$ is ~ 10 times that of $SMW()$
   - ✓ More comprehensive metric needed?
4. In-depth Analysis
   - ✗ The conclusions are based on empirical evidence
   - ✓ ~~Find provable error bounds?~~
     Investigate the major source of error?

References I

James R. Bunch, Linda Kaufman.
Some stable methods for calculating inertia and solving symmetric linear systems.
Mathematics of Computation, 31 (1977), 163-179.

I. S. Duff, J. K. Reid.
The Multifrontal Solution of Indefinite Sparse Symmetric Linear Equations.
ACM Transactions on Mathematical Software, Vol. 9, No. 3, September 1983, 302-325.

References II

Xiaoye S. Li, James W. Demmel.
SuperLU_DIST: A Scalable Distributed-Memory Sparse Direct Solver for Unsymmmetric Linear Systems.
ACM Transactions on Mathematical Software, Vol. 29, No. 2, June 2003.

N. Egidi, P. Maponi.
A Sherman-Morrison approach to the solution of linear systems.
Journal of Computational and Applied Mathematics, 189 (2006), 703-718.

Elizabeth D. Dolan, Jorge J. More.
Benchmarking optimization software with performance profiles.
Mathematical Programming, Ser. A 91: 201-213 (2002).