RCHOL: Randomized Cholesky Factorization for Solving SDD Linear Systems

Chao Chen Tianyu Liang George Biros

University of Texas at Austin

chenchao.nk@gmail.com

Sparse Days 2020
November 23

Symmetric Diagonally-dominant (SDD) Linear System

$$Ax = b, \quad A = (a_{ij}) \in \mathbb{R}^{N \times N} \tag{1}$$

- A is SDD: $A = A^\top$ and $a_{ii} \geq \sum_{j \neq i} |a_{ij}|$ (non-negative diagonal)
- Assume A is nonsingular and irreducible[1]
- Discretization of PDEs with, e.g., finite difference and finite elements

[1] A can't be permuted to be a block diagonal matrix.

Overview

- Randomized Cholesky factorization for Laplacian matrices
- Solve SDD linear systems with PCG
- Parallel algorithm based on nested-dissection multifrontal method
- Numerical results

Laplacian Matrix (a.k.a., Graph Laplacian)

Definition (Laplacian matrix $L \in \mathbb{R}^{N \times N}$)

(1) $L = L^\top$, (2) $\ell_{ij} \leq 0$ when $i \neq j$, and (3) $\sum_j \ell_{ij} = 0$.

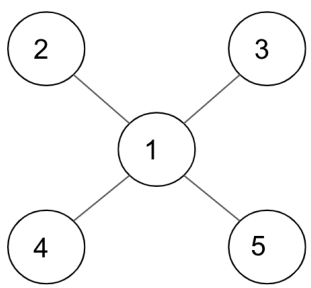
- Subclass of SDD matrices
- Singular. If L is irreducible $\rightarrow$ $\mathrm{Ker}(L) = \mathrm{span}\{\mathbf{1}\}$
- One-to-one maps to a weighted undirected graph

Classical Cholesky for Laplacian Matrix

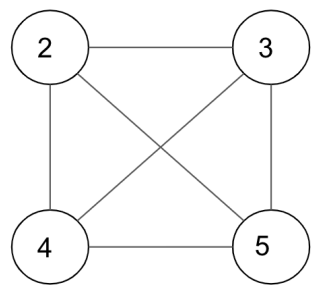
Algorithm 1

Input: irreducible Laplacian matrix $L \in \mathbb{R}^{N \times N}$
Output: lower triangular matrix $G \in \mathbb{R}^{N \times N}$

1: $G = \mathbf{0}_{N \times N}$
2: **for** $k = 1$ **to** $N - 1$ **do**
3: $\quad G(:, k) = L(:, k)/\sqrt{\ell_{kk}}$ // $\ell_{kk} > 0$
4: $\quad L = L - \frac{1}{\ell_{kk}} L(:, k)\, L(k, :)$ // dense update
5: **end for** // $\ell_{NN} = g_{NN} = 0$



(a) Graph of L



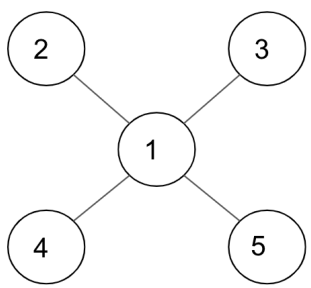
(b) Clique

Classical Cholesky for Laplacian Matrix

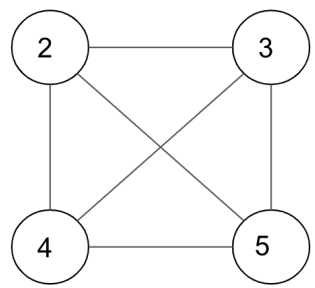
Algorithm 1

Input: irreducible Laplacian matrix $L \in \mathbb{R}^{N \times N}$
Output: lower triangular matrix $G \in \mathbb{R}^{N \times N}$

1: $G = \mathbf{0}_{N \times N}$
2: **for** $k = 1$ **to** $N - 1$ **do**
3: $\quad G(:, k) = L(:, k)/\sqrt{\ell_{kk}}$ // $\ell_{kk} > 0$
4: $\quad L = L - \mathrm{Star}(L, k) + \mathrm{Clique}(L, k)$ // dense update
5: **end for** // $\ell_{NN} = g_{NN} = 0$



(a) Graph of L



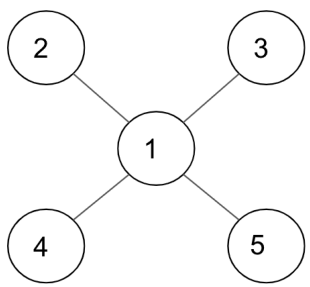
(b) Clique

Randomized Cholesky[2] for Laplacian Matrix

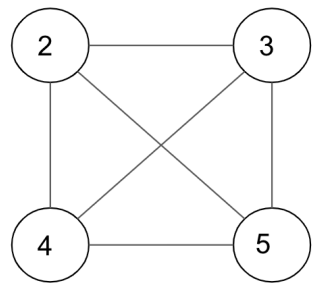
Algorithm 2 Randomized Cholesky (`rchol`)

Input: irreducible Laplacian matrix $L \in \mathbb{R}^{N\times N}$
Output: lower triangular matrix $G \in \mathbb{R}^{N\times N}$

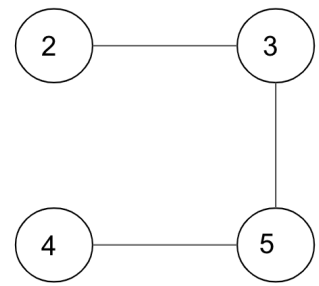
1: $G = \mathbf{0}_{N\times N}$
2: **for** $k = 1$ **to** $N-1$ **do**
3: $\quad G(:,k) = L(:,k)/\sqrt{\ell_{kk}}$ // $\ell_{kk} > 0$
4: $\quad L = L - \mathrm{Star}(L,k) + \mathrm{SampleClique}(L,k)$ // sparse update
5: **end for**



(a) Graph of L



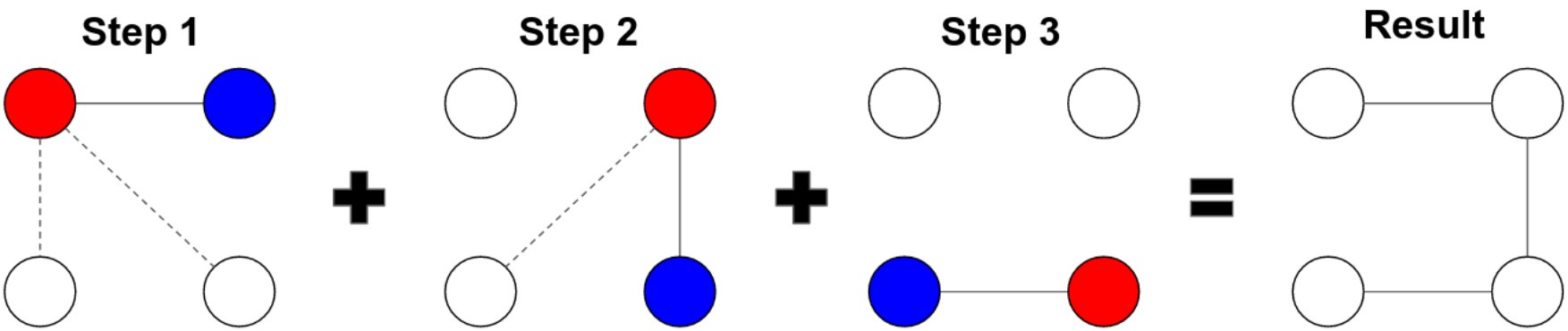
(b) Clique



(c) Sampled edges

[2] Kyng, Rasmus, and Sushant Sachdeva. "Approximate gaussian elimination for laplacians—fast, sparse, and simple." FOCS, 2016.

Clique Sampling[3]



Algorithm 3

1: Sort neighbors of the eliminated vertex
2: **for** red vertex **in** neighbors **do**
3: sample blue vertex from remaining neighbors
4: select the edge ("red", "blue") and assign a proper weight
5: remove "red" from neighbors
6: **end for**

[3] D. A. Spielman. "Laplacians.jl, Version 1.2.0, https://github.com/danspielman/Laplacians.jl/blob/master/docs/src/usingSolvers.md#sampling-solvers-of-kyng-and-sachdeva," 2020.

Robustness of `rchol`

Theorem (Spanning Tree on Clique)

Sampled edges form a spanning tree of the clique.

Corollary (Breakdown Free)

With the sampling, `rchol` never breaks down.

Complexity of `rchol`

Theorem (Running Time and Storage)

With a random elimination ordering,

$$\mathbb{E}[\textit{running time}] = \mathbb{E}[\#\ \textit{of fill-in}] = \mathcal{O}(M \log N)$$

where M is the # of non-zeros.

Theorem (Unbiased Estimator)

The clique sampling algorithm returns an unbiased estimator.

Conjecture (Concentration Result)

With high probability, the error introduced by `rchol` is small and depends weakly on N.

Notations for Numerical Results

Solve $Ax = b$ with PCG using the preconditioner computed by the randomized Cholesky factorization (`rchol`).

- N: matrix size of A
- n_{it}: number of the PCG iterations with tolerance 10^{-10}
- $$\text{fill/nnz} = \frac{\#\text{ of non-zeros in the preconditioner}}{\#\text{ of non-zeros in } A}$$

  (relative storage of the preconditioner)
- `ichol`: incomplete Cholesky with drop tolerance in MATLAB

Performance with Different Orderings

Table: *Orderings were computed by Matlab commands in parentheses.*
Poisson equation in a cube with Dirichlet boundary condition, discretized with 7-point stencil on a $256 \times 256 \times 256$ *grid.*

Ordering	fill/nnz	t_p	t_f	t_s
no reordering	**10.2**	0	97	207
reverse Cuthill-McKee (symrcm)	**7.9**	5	74	172
random ordering (randperm)	3.3	0.8	46	**337**
nested dissection (dissect)	3.3	**206**	26	147
approximate minimum degree (amd)	3.5	38	29	139

- t_p: time (seconds) for computing the ordering
- t_f: time (seconds) for constructing the preconditioner
- t_s: total PCG time (seconds) for solving a random RHS

amd: default option in `rchol`

Concentration Results

`rchol` behaves like a deterministic method.

Table: *Three trials of* `rchol`. *Poisson equation in a cube with Dirichlet boundary condition, discretized with 7-point stencil on a* $256 \times 256 \times 256$ *grid.*

trial	fill/nnz	t_f	t_s	n_{it}
1^{st}	3.5398	28	126	59
2^{nd}	3.5428	26	121	57
3^{rd}	3.5426	28	126	60

Concentration Results: Scalability with Problem Size

Table: PCG iterations with tolerance 10^{-10}. Poisson equation in a cube with Dirichlet boundary condition, discretized with 7-point stencil on regular grids.

N	128^3	256^3	512^3	1024^3
`rchol`	50	57	67	75
`ichol`[4]	100	185	341	-

[4] `ichol` performed best without any reordering; `ichol` was manually tuned to have slightly more fill-in.

SPD Matrices from SuiteSparse Matrix Collection

Name	N	nnz	no preconditioner	`ichol`		`rchol`	
			n_{it}	fill/nnz	n_{it}	fill/nnz	n_{it}
ecology2	1.0e+5	5.0e+6	> 2500	2.72	798	2.41	89
parabolic_fem	5.3e+5	3.7e+6	> 2500	2.29	411	2.27	65
apache2	7.2e+5	4.8e+6	> 2500	2.96	322	2.93	60
G3_circuit	1.6e+6	7.7e+6	> 2500	2.75	379	2.68	96

Parallel Scalability of `rchol`

Table: *Poisson equation in a cube with Dirichlet boundary condition, discretized with 7-point stencil on a regular grid.*

p	$N = 1024^3$		
	fill/nnz	t_f (sec)	n_{it}
1	4.31	2523	78
2	4.37	1279	79
4	4.39	664	75
8	4.38	388	75
16	4.38	258	76
32	4.39	197	71
64	4.38	184	75

- p: number of threads/cores on Intel Xeon Platinum 8280M ("Cascade Lake"), which has 112 cores on four sockets (28 cores/socket)
- t_f: time (seconds) for constructing the preconditioner
- (Single precision) 130 GB storage for the preconditioner
- (Single precision) 14× speedup using 64 cores

Code & Preprint

- Code with C++/MATLAB/Python interfaces at

  `https://github.com/ut-padas/rchol`

- Preprint at

  `https://arxiv.org/abs/2011.07769`