Lecture Notes on Advances of Numerical Methods for Hubbard Quantum Monte Carlo Simulation

Part 2, July 31, 2007

Zhaojun Bai
Wenbin Chen
Richard Scalettar
Ichitaro Yamazaki

Contents

Lecture 1

Self-adaptive direct linear system solvers

1.1 Introduction

In this lecture, we consider one of the computational kernels of the QMC simulations discussed in Lecture 1: solving the linear system of equations

$$Mx = b, \tag{1.1.1}$$

where the coefficient matrix M is the Hubbard matrix as defined in Lecture 2, see equation (**??**).

One of main challenges in the multiscale QMC simulation is to develop algorithmic techniques and paradigms that can robustly and efficiently solve numerical linear algebra problems with underlying multiscale coefficient matrices in a self-adapting fashion to achieve a required simulation accuracy.

The Hubbard matrix M exhibits the form of a so-called block p-cyclic consistently ordered matrix [8]. p-cyclic matrices arise in a number of important contexts in applied mathematics, including numerical solution of boundary value problems for ordinary differential equations [7], finite-difference equations for the steady-state solution of a parabolic equation with periodic boundary conditions [6], and computing the stationary solution of Markov chains with periodic graph structure [5].

It is known that the block Gaussian elimination with and without pivoting for solving p-cyclic linear systems can be numerically unstable, similar to the case of multiple shooting method for solving two-point boundary value problems [10, 2] and Markov chain modeling [4].

Block cyclic reduction [1] is a powerful idea to solve such p-cyclic system. However, a full block cyclic reduction is applicable only for small energy scales, namely, $U \leq 1$, due to emerging ill-condition of the reduced system. A stable p-cyclic linear system solver is based on the structural orthogonal factorization [9, 2]. Unfortunately, the costs of memory requirements and flops is prohibitively expensive when the length scales N and L increase.

To take advantage of significant reduction of memory requirement and floating point computations in the block cyclic reduction and numerical stability of the orthogonal factorization method, and to carefully examine the accuracy needs in our quantum monte carlo simulation, in this lecture, we present a hybrid method, which we simply call a *Self-Adaptive Block cyclic reduction Orthogonal factorization method*, or SABO method for short.

1.2 Block cyclic reduction

Consider the following 16×16 block cyclic linear system (1.1.1):

$$Mx = b,$$

where

$$\begin{bmatrix} I & & & & & & & B_1 \\ -B_2 & I & & & & & & \\ & -B_3 & I & & & & & \\ & & -B_4 & I & & & & \\ & & & \ddots & \ddots & & & \\ & & & & -B_{15} & I & & \\ & & & & & -B_{16} & I & \end{bmatrix}.$$

Correspondingly, partitions of the vectors x and b are comformed to the blocks of M:

$$x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ \vdots \\ x_{15} \\ x_{16} \end{bmatrix}, \quad b = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ \vdots \\ b_{15} \\ b_{16} \end{bmatrix},$$

A-factor-of-four block cyclic reduction (BCR) leads to a 4×4 block cycle linear system of the form

$$M^{(4)} x^{(4)} = b^{(4)},$$

where

$$M^{(4)} = \begin{bmatrix} I & & & B_1^{(4)} \\ B_2^{(4)} & I & & \\ & B_3^{(4)} & I & \\ & & B_4^{(4)} & I \end{bmatrix}, \quad x^{(4)} = \begin{bmatrix} x_4 \\ x_8 \\ x_{12} \\ x_{16} \end{bmatrix}, \quad b^{(4)} = \begin{bmatrix} b_1^{(4)} \\ b_2^{(4)} \\ b_3^{(4)} \\ b_4^{(4)} \end{bmatrix}$$

and

$$\begin{array}{lcllcl} B_1^{(4)} & = & B_4B_3B_2B_1 & b_1^{(4)} & = & b_4 + B_4b_3 + B_4B_3b_2 + B_4B_3B_2b_1 \\ B_2^{(4)} & = & B_8B_7B_6B_5 & b_2^{(4)} & = & b_8 + B_8b_7 + B_8B_7b_6 + B_8B_7B_6b_5 \\ B_3^{(4)} & = & B_{12}B_{11}B_{10}B_9 & b_3^{(4)} & = & b_{12} + B_{12}b_{11} + B_{12}B_{11}b_{10} + B_{12}B_{11}B_{10}b_9 \\ B_4^{(4)} & = & B_{16}B_{15}B_{14}B_{13} & b_4^{(4)} & = & b_{16} + B_{16}b_{15} + B_{16}B_{15}b_{14} + B_{16}B_{15}B_{14}b_{13} \end{array}$$

Once the vector $x^{(4)}$ is computed, i.e, the block components x_4, x_8, x_{12} and x_{16} of the solution x are computed, the rest of block components x_i of the solution x can be computed by the following forward and back substitutions:

- Forward substitution:

$$\begin{aligned} x_1 &= b_1 - B_1 x_{16}, & x_2 &= b_2 + B_2 x_1, \\ x_5 &= b_5 + B_5 x_4, & x_6 &= b_6 + B_6 x_5, \\ x_9 &= b_9 + B_9 x_8, & x_{10} &= b_{10} + B_{10} x_9, \\ x_{13} &= b_{13} + B_{13} x_{12}, & x_{14} &= b_{14} + B_{14} x_{13}, \end{aligned}$$

- Back substitution:

$$\begin{aligned} x_3 &= B_4^{-1}(x_4 - b_4), & x_7 &= B_8^{-1}(x_8 - b_8), \\ x_{11} &= B_{12}^{-1}(x_{12} - b_{12}), & x_{15} &= B_{16}^{-1}(x_{16} - b_{16}), \end{aligned}$$

The use of both forward and back substitutions can minimize the propagation of rounding errors in the computed x_4, x_8, x_{12} and x_{16}.

The pattern for a general factor-of-k reduction is clear. Given an integer $k \leq L$, a-factor-of-k BCR leads to a $L^{(k)} \times L^{(k)}$ block cycle linear system:

$$M^{(k)} x^{(k)} = b^{(k)}, \tag{1.2.2}$$

where $L_k = \lceil \frac{L}{k} \rceil$,

$$M^{(k)} = \begin{bmatrix} I & & & & & B_1^{(k)} \\ -B_2^{(k)} & I & & & & \\ & -B_3^{(k)} & I & & & \\ & & \ddots & \ddots & & \\ & & & -B_{L_k}^{(k)} & I \end{bmatrix},$$

with

$$\begin{aligned} B_1^{(k)} &= B_k B_{k-1} \cdots B_2 B_1 \\ B_2^{(k)} &= B_{2k} B_{2k-1} \cdots B_{k+2} B_{k+1} \\ &\vdots \\ B_{L_k}^{(k)} &= B_L B_{L-1} \cdots B_{(L_k-1)k+1}, \end{aligned}$$

and the vectors $x^{(k)}$ and $b^{(k)}$ are

$$x^{(k)} = \begin{bmatrix} x_k \\ x_{2k} \\ \vdots \\ x_{(L_k-1)k} \\ x_L \end{bmatrix} \quad \text{and} \quad b^{(k)} = \begin{bmatrix} b_k + \sum_{t=1}^{k-1} B_k \cdots B_{t+1} b_t \\ b_{2k} + \sum_{t=k+1}^{2k-1} B_{2k} \cdots B_{t+1} b_t \\ \vdots \\ b_{(L_k-1)k} + \sum_{t=(L_k-2)k+1}^{(L_k-1)k-1} B_{(L_k-1)k} \cdots B_{t+1} b_t \\ b_L + \sum_{t=(L_k-1)k+1}^{L-1} B_L \cdots B_{t+1} b_t \end{bmatrix}.$$

After the solution vector $x^{(k)}$ of the reduced system (1.2.2) is computed, then the rest of block components x_i of the solution vector x are obtained by forward and back substitutions, that is to say, when the index t is less than $\frac{k}{2}$, the components of block x_{ik+t} are obtained by the forward substitution and otherwise by the back substitution. The pseudo-code is as the following.

1. Let $\ell = [k, 2k, \cdots, (L_k - 1)k, L]$
2. For $j = 1, 2, \cdots, L_k$
   (a) $x_{\ell(j)} = x_j^{(k)}$
   (b) forward substitution
   For $i = \ell(j-1)+1, \ell(j-1)+2, \ldots, \ell(j-1) + \lceil \frac{1}{2}(\ell(j) - \ell(j-1) - 1) \rceil$
   with $\ell(0) = 0$:
   If $i = 1$, $x_1 = b_1 - B_1 x_L$
   else $x_i = b_i + B_i x_{i-1}$
   (c) back substitution
   For $i = \ell(j) - 1, \ell(j) - 2, \ldots, \ell(j) - \lfloor \frac{1}{2}(\ell(j) - \ell(j-1) - 1) \rfloor$,
   $x_i = B_{i+1}^{-1}(x_{i+1} - b_{i+1})$.

Remark **1.2.1** *If the reduction factor $k = L$, then $L_k = 1$. The reduced system is*

$$M^{(L)} x_L = b^{(L)},$$

i.e,

$$(I + B_L B_{L-1} \cdots B_1) x_L = b_L + \sum_{t=1}^{L-1} B_L \cdots B_{t+1} b_t.$$

Remark **1.2.2** *There are a number of ways to derive the reduced system (1.2.2). For example, we can use the block Gaussian elimination. Writing the matrix M of the original system* (1.1.1) *as a L_k by L_k block matrix:*

$$M = \begin{bmatrix} D_1 & & & & \widehat{B}_1 \\ -\widehat{B}_2 & D_2 & & & \\ & -\widehat{B}_3 & D_3 & & \\ & & \ddots & \ddots & \\ & & & -\widehat{B}_{L_k} & D_{L_k} \end{bmatrix},$$

where D_i are $k \times k$ black matrices defined as

$$D_i = \begin{bmatrix} I & & & & \\ -B_{(i-1)k+2} & I & & & \\ & -B_{(i-1)k+3} & I & & \\ & & \ddots & \ddots & \\ & & & -B_{ik} & I \end{bmatrix},$$

and $\widehat{B}_i$ are $k \times k$ block matrices defined as

$$\widehat{B}_i = \begin{bmatrix} 0 & 0 & \cdots & B_{(i-1)k+1} \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & 0 \end{bmatrix}.$$

Define $\widehat{D} = \mathrm{diag}(D_1, D_2, \cdots, D_{L_k})$, *then*

$$\widehat{D}^{-1}M = \begin{bmatrix} I & & & & & D_1^{-1}\widehat{B}_1 \\ -D_2^{-1}\widehat{B}_2 & I & & & & \\ & -D_3^{-1}\widehat{B}_3 & I & & & \\ & & \ddots & \ddots & & \\ & & & -D_{L_k}^{-1}\widehat{B}_{L_k} & I & \end{bmatrix}.$$

Note that the matrix $D_i^{-1}\widehat{B}_i$ *is given by*

$$D_i^{-1}\widehat{B}_i = \begin{bmatrix} 0 & 0 & \cdots & B_{(i-1)k+2}B_{(i-1)k+1} \\ 0 & 0 & \cdots & B_{(i-1)k+3}B_{(i-1)k+2}B_{(i-1)k+1} \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & B_{(i-1)k+k}\cdots B_{(i-1)k+2}B_{(i-1)k+1} \end{bmatrix}.$$

Therefore, $M^{(k)}$ *is a submatrix of* $\widehat{D}^{-1}M$. *There exists a matrix* Π, *such that*

$$M^{(k)} = \Pi^T \widehat{D}^{-1} M \Pi,$$

where the matrix Π *is* $NL \times (NL/k)$ *matrix, whose* $(i-1)N+1$ *to* iN *columns are the* $(ik-1)N+1$ *to* ikN *columns of the identity matrix* I_{NL}.

1.3 Block orthogonal factorization method

Comparing with the Gaussian elimination (LU factorization) method, the block orthogonal factorization (BOF) method presented in this section is computationally more expensive, but numerically backward stable.

It is easy to see that by multiplying a sequence of orthogonal transformation matrices Q_i, the block cyclic matrix M of the system (1.1.1) can be transformed to an upper triangular, namely,

$$Q_{L-1}^T \cdots Q_2^T Q_1^T M = R, \tag{1.3.3}$$

where

$$R = \begin{bmatrix} R_{11} & R_{12} & & & R_{1L} \\ & R_{22} & R_{23} & & R_{2L} \\ & & \ddots & \ddots & \vdots \\ & & & R_{L-1,L-1} & R_{L-1,L} \\ & & & & R_{LL} \end{bmatrix},$$

and diagonal blocks $R_{\ell,\ell}$ are upper triangular. The orthogonal matrices Q_ℓ are defined by

$$Q_\ell = \begin{bmatrix} I & & & & & \\ & \ddots & & & & \\ & & Q_{11}^{(\ell)} & Q_{12}^{(\ell)} & & \\ & & Q_{21}^{(\ell)} & Q_{22}^{(\ell)} & & \\ & & & & \ddots & \\ & & & & & I \end{bmatrix},$$

where the blocks $Q_{ij}^{(\ell)}$ is defined by the orthogonal factor of the QR decomposition:

$$\begin{bmatrix} \widetilde{M}_{\ell\ell} \\ -B_{\ell+1} \end{bmatrix} = \begin{bmatrix} Q_{11}^{(\ell)} & Q_{12}^{(\ell)} \\ Q_{21}^{(\ell)} & Q_{22}^{(\ell)} \end{bmatrix} \begin{bmatrix} R_{\ell\ell} \\ 0 \end{bmatrix},$$

where $\widetilde{M}_{\ell\ell}$ are defined as

- for $\ell = 1$, $\widetilde{M}_{\ell\ell} = I$
- for $\ell = 2, 3, \ldots, L-2$, $\widetilde{M}_{\ell\ell} = (Q_{22}^{(\ell-1)})^T$,

except for the the last step $\ell = L-1$, we use the QR decomposition:

$$\begin{bmatrix} \widetilde{M}_{L-1,L-1} & R_{L-1,L} \\ -B_L & I \end{bmatrix} = \begin{bmatrix} Q_{11}^{(L-1)} & Q_{12}^{(L-1)} \\ Q_{21}^{(L-1)} & Q_{22}^{(L-1)} \end{bmatrix} \begin{bmatrix} R_{L-1,L-1} & \check{R}_{L-1,L} \\ 0 & R_{LL} \end{bmatrix}.$$

The following is a pseudo-code for the BOF method to solve the block cyclic system (1.1.1).

BOF method

1. Set $M_{11} = I, R_{1L} = B_1$ and $c_1 = b_1$.
2. For $\ell = 1, 2, \cdots, L-2$,
   (a) Compute the QR decomposition
   $$\begin{bmatrix} M_{\ell\ell} \\ -B_{\ell+1} \end{bmatrix} = \begin{bmatrix} Q_{11}^{(\ell)} & Q_{12}^{(\ell)} \\ Q_{21}^{(\ell)} & Q_{22}^{(\ell)} \end{bmatrix} \begin{bmatrix} R_{\ell\ell} \\ 0 \end{bmatrix}.$$
   (b) Set $\begin{bmatrix} R_{\ell,\ell+1} \\ M_{\ell+1,\ell+1} \end{bmatrix} = \begin{bmatrix} Q_{11}^{(\ell)} & Q_{12}^{(\ell)} \\ Q_{21}^{(\ell)} & Q_{22}^{(\ell)} \end{bmatrix}^T \begin{bmatrix} 0 \\ I \end{bmatrix}$
   (c) Update $\begin{bmatrix} R_{\ell L} \\ R_{\ell+1,L} \end{bmatrix} := \begin{bmatrix} Q_{11}^{(\ell)} & Q_{12}^{(\ell)} \\ Q_{21}^{(\ell)} & Q_{22}^{(\ell)} \end{bmatrix}^T \begin{bmatrix} R_{\ell L} \\ 0 \end{bmatrix}$
   (d) Set $\begin{bmatrix} c_\ell \\ c_{\ell+1} \end{bmatrix} := \begin{bmatrix} Q_{11}^{(\ell)} & Q_{12}^{(\ell)} \\ Q_{21}^{(\ell)} & Q_{22}^{(\ell)} \end{bmatrix}^T \begin{bmatrix} c_\ell \\ b_{\ell+1} \end{bmatrix}$.
3. For $\ell = L-1$,
   (a) Compute the QR decomposition
   $$\begin{bmatrix} M_{L-1,L-1} & R_{L-1,L} \\ -B_L & I \end{bmatrix} = \begin{bmatrix} Q_{11}^{(L-1)} & Q_{12}^{(L-1)} \\ Q_{21}^{(L-1)} & Q_{22}^{(L-1)} \end{bmatrix} \begin{bmatrix} R_{L-1,L-1} & R_{L-1,L} \\ 0 & R_{LL} \end{bmatrix}.$$
   (b) Set $\begin{bmatrix} c_{L-1} \\ c_L \end{bmatrix} := \begin{bmatrix} Q_{11}^{(L-1)} & Q_{12}^{(L-1)} \\ Q_{21}^{(L-1)} & Q_{22}^{(L-1)} \end{bmatrix}^T \begin{bmatrix} c_{L-1} \\ b_L \end{bmatrix}$
4. Back substitution to solve the block triangular system $Rx = c$
   (a) Solve $R_{LL}x_L = c_L$ for x_L.
   (b) Solve $R_{L-1,L-1}x_{L-1} = c_{L-1} - R_{L-1,L}x_L$ for x_{L-1}.

(c) For $\ell = L-2, L-3, \ldots, 1$, solve
$$R_{\ell\ell} x_\ell = c_\ell - R_{\ell,\ell+1} x_{\ell+1} - R_{\ell L} x_L$$
for x_ℓ.

Floating point operations of the BOF method is about $15N^3L$, with a memory requirement of $3N^2L$.

1.4 A hybrid method

To take advantages of block reduction of the BCR method and numerical stability of the BOF method, and to meet the accuracy needs in the QMC simulation, we propose to use a hybrid method:

Step 1. Perform a factor-of-k BCR of the original system (1.1.1) to derive a reduced block cyclic system (1.2.2).

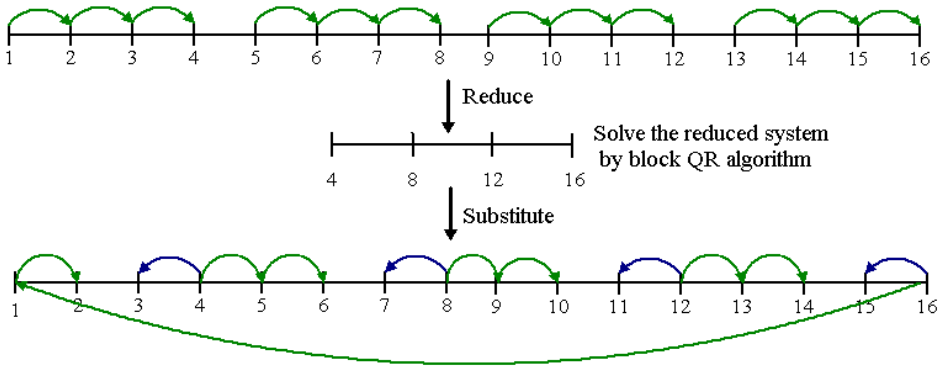
Step 2. Solve the reduced block cyclic system (1.2.2) by using the BOF method.

Step 3. Forward and back substitute to find the rest components x_i of the solution x of the original system:

$$\{x_i\} \quad \longleftarrow \quad x^{(k)} \quad \longrightarrow \quad \{x_j\}.$$

Note that we use both forward and back substitutions to minimize the propagation of rounding errors induced at steps 1 and 2.

The following is a schematic map of the hybrid method for a 16-block cyclic system with a reduction factor $k = 4$.



By step 1, the order of $M^{(k)}$ is reduced by a factor of k. Subsequently, the computational cost of the BOF method at Step 2 is reduced from $O(N^3L)$ to $O(N^3\frac{L}{k})$, a factor of k speedup. The larger k is, the better. However, on the other hand, the condition number of $M^{(k)}$ increases as k increases, which in turn that the accuracy of the computed solution decreases. A critical question is how to find a reduction factor k, such that the computed solution has the required accuracy for the simulation. Furthermore, such a reduction factor k should be determined in a *self-adapting* fashion with respect to the changes of underlying problem length and energy scales.

1.5 Self-adaptive reduction factor k

We turn to the question of how to determine the reduction factor k for the BCR step of the proposed hybrid method.

Since the BOF method is backward stable, by the well-established error analysis of the linear system, we know that the relative error of the computed solution $\widehat{x}^{(k)}$ of the reduced system (1.2.2) is bounded by $\kappa(M^{(k)})\epsilon$, i.e.,

$$\frac{\|\delta x^{(k)}\|}{\|x^{(k)}\|} \equiv \frac{\|x^{(k)} - \widehat{x}^{(k)}\|}{\|x^{(k)}\|} \leq \kappa(M^{(k)})\epsilon, \tag{1.5.4}$$

where ϵ is the machine precision. For example, see [3, p.120]. For the clearity of notation, we only use the first-order upper bound and ignore the constant coefficient, which is about 2.

Let us consider the propagation of the errors in the computed solution $\widehat{x}^{(k)}$ during the back and forward substitutions. Let us start with the computed L-th block compoinent $\widehat{x}_L$ of the solution vector x,

$$\widehat{x}_L = x_L + \delta x_L,$$

where δx_L is the computed error, with a related upper bound defined in (1.5.4). By the forward substitution, the computed first block component $\widehat{x}_1$ of the solution x satisfies

$$\widehat{x}_1 = b_1 - B_1\widehat{x}_L = b_1 - B_1(x_L + \delta x_L) = b_1 - B_1 x_L + B_1\delta x_L = x_1 + \delta x_1,$$

where $\delta x_1 = B_1\delta x_L$ is the error propagated by the error in $\widehat{x}_L$. By the relative error bound of δx_L, it yields that the error in the computed $\widehat{x}_1$ could be amplified by the factor $\|B_1\|$, namely,

$$\frac{\|\delta x_1\|}{\|x_1\|} \leq \|B_1\|\kappa(M^{(k)})\,\epsilon.$$

Similarly, we conclude that the relative error of the computed solution $\widehat{x}_2$ is bounded by

$$\frac{\|\delta x_2\|}{\|x_2\|} \leq \|B_2\|\|B_1\|\kappa(M^{(k)})\,\epsilon.$$

By the end of forward substitution starting with x_L, we have that relative error of the computed solution $x_{\frac{k}{2}}$ is bounded by

$$\frac{\|\delta x_{\frac{k}{2}}\|}{\|x_{\frac{k}{2}}\|} \leq \|B_{\frac{k}{2}}\|\cdots\|B_2\|\,\|B_1\|\kappa(M^{(k)})\,\epsilon.$$

In summary, we conclude that the errors in all computed block components $\widehat{x}_\ell$ of the solution x are bounded by

$$\begin{aligned}\frac{\|\delta x_\ell\|}{\|x_\ell\|} &\leq \|B_{\frac{k}{2}}\|\cdots\|B_2\|\,\|B_1\|\kappa(M^{(k)})\,\epsilon \\ &\leq c\,e^{\frac{1}{2}k(4t\Delta\tau+\nu)}\cdot e^{k(4t\Delta\tau+\nu)}\kappa(M)\,\epsilon \\ &= c\,e^{\frac{3}{2}k(4t\Delta\tau+\nu)}\kappa(M)\,\epsilon \end{aligned} \tag{1.5.5}$$

for $\ell = 1, 2, \ldots, L$, where for the second inequality we have use the upper bounds (**??**) and (**??**) for the norm of B_ℓ and the condition number of the matrix $M^{(k)}$.

Assume that a desired accuracy of the solution vector x is specified by

$$\frac{\|\delta x\|}{\|x\|} \leq \text{tol}. \tag{1.5.6}$$

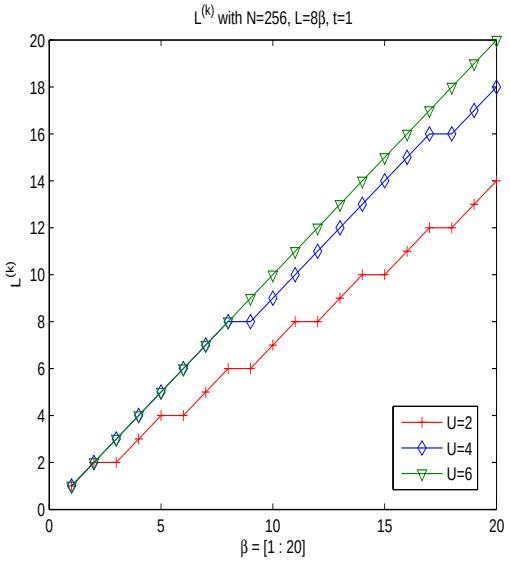
Then by combining the inequalities (1.5.5) and (1.5.6), a plausible choice of the reduction factor k is

$$k = \left\lfloor \frac{\frac{2}{3}\ln(\text{tol}/\epsilon)}{4t\tau + \nu} \right\rfloor. \tag{1.5.7}$$

In practice, to balance the number of the matrices B_ℓ in the product $B_\ell^{(k)}$, after k is computed as above, then we compute $L_k = \lceil \frac{L}{k} \rceil$. The final k is adjusted as $k = \left\lceil \frac{L}{L_k} \right\rceil$. Here we drop the factor of $\ln \kappa(M)$ in deciding reduction factor k. The reason is that, as we discussed in section 2.4, $\kappa(M)$ grows slowly in the range of parameters of interest, it is expected that $\ln \kappa(M)$ is small.

The proposed reduction factor k is determined in a *self-adaptively* fashion, since when U and other energy parameters change, the value of k is determined adaptively to achieve the desired accuracy.

The following plot shows the reduced number L_k of the blocks for different values of $L = 8\beta$ with respect to different values of U, where $\beta = 1, 2, \ldots, 20$ and $t = 1$, $N = 256$. The desired accuracy is set to be half of the machine precision, i.e., $\text{tol} = 10^{-8}$ and $\epsilon = 10^{-16}$.



1.6 Self-adapting block cyclic reduction orthogonal factorization method

A high-level outline of a self-adapting block cyclic reduction orthogonal factorization method, SABO in short, to solve the linear system (1.1.1) may be condensed as the following:

SABO method

1. Determine the reduction factor k by (1.5.7),
2. Reduce (M, b) to $(M^{(k)}, b^{(k)})$ by the BCR,
3. Solve the reduced system $M^{(k)}x^{(k)} = b^{(k)}$ by the BOF method,
4. Use forward and back substitutions to compute the remaining solution components.

1.7 Numerical experiments

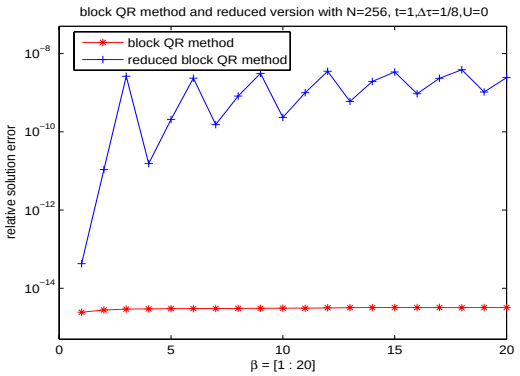
In all numerical experiments, it is set that the SABO solver (implemented in FORTRAN 90) has the relative accuracy at the order of $\sqrt{\epsilon}$, i.e.,

$$\frac{\|\delta x\|}{\|x\|} \leq \text{tol} = 10^{-8}.$$

Performance data are collected from an Intel Itanium2 workstation with 1.5GHZ CPU and 2GB core memory.

Experiment 1. In this experiment, we examine robustness, stability and performance of the SABO solver when $U = 0$. The rest of parameters of the coefficient matrix M is set as $N = 16 \times 16$, $L = 8\beta$ for $\beta = 1, 2, \ldots, 20$. and $t = 1, \Delta\tau = \frac{1}{8}$.

The following plot first shows the relative error of the computed solutions $\widehat{x}$ by the BOF method is at the full machine precision $O(10^{-15})$. It implies two facts: (1) the linear system at $U = 0$ is well-conditioned and (2) the BOF is backward stable, Second, the plot also shows that the relative errors of the computed solutions $\widehat{x}$ by the SABO method are at $O(10^{-8})$ as it is prescribed.



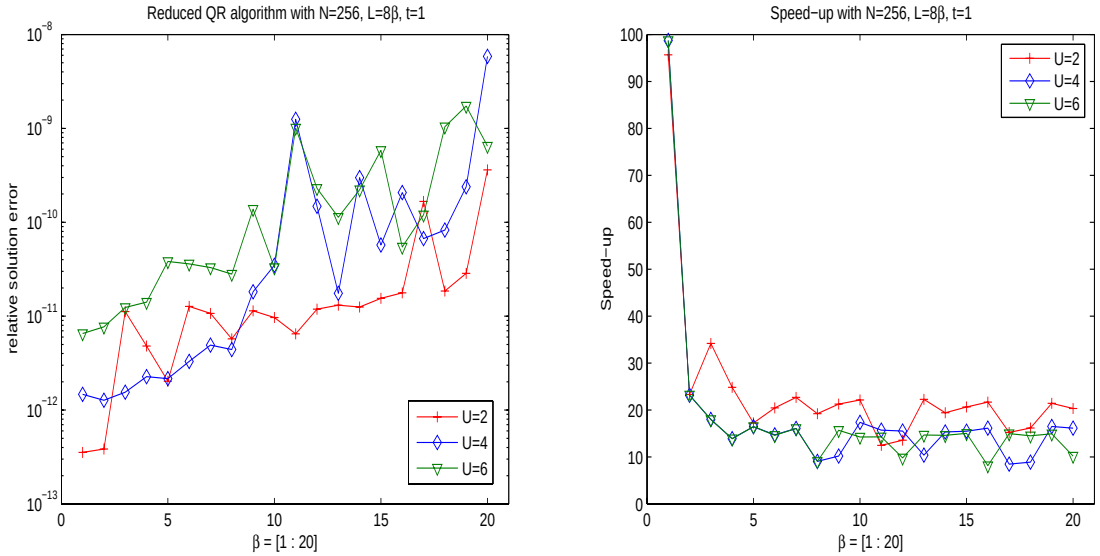
The reduction factor k, CPU time and speedups of the SABO method over the BOF method are reported in the following table:

β	$L = 8\beta$	k	L_k	BOF (sec.)	SABO (sec.)	speedup ($\times$)
1	8	8	1	3.19	0.0293	108
2	16	16	1	7.06	0.042	168
3	24	24	1	10.8	0.0547	197
4	32	16	2	14.6	0.303	48
5	40	20	2	18.6	0.326	57
6	48	24	2	23.1	0.342	67
7	56	19	3	27.2	0.666	40
8	64	22	3	31.3	0.683	45
9	72	24	3	35.1	0.675	52
10	80	20	4	38.0	1.18	32
11	88	22	4	42.0	1.18	35
12	96	24	4	46.0	1.20	38
13	104	21	5	49.9	1.28	38
14	112	23	5	54.0	1.28	42
15	120	24	5	58.2	1.32	44
16	128	22	6	62.9	1.67	37
17	136	23	6	68.3	1.72	39
18	144	24	6	73.2	1.73	42
19	152	22	7	75.3	1.98	38
20	160	23	7	80.2	2.02	39

Note that for small β, namely, $\beta \leq 3$, the SABO solver reduces the original system all the way to one block $M^{(L)}$, i.e., $k = L$. However, for large β, the reduction factor k is smaller, and the BCR reduces the original system to 6 or 7 blocks. For example, when $\beta = 20$, $L = 160$, it reduces to $L_k = \lfloor \frac{160}{23} \rfloor + 1 = 7$ with the reduction factor $k = 23$.

Experiment 2. In this experiment, we examine robustness, stability and performance of the SABO solver for $U = 2, 4, 6$. The rest of the parameters of the coefficient matrix M are $N = 256$, $L = 8\beta$ with $\beta = 1, 2, \ldots, 20$. $t = 1$ and $\Delta\tau = \frac{1}{8}$.

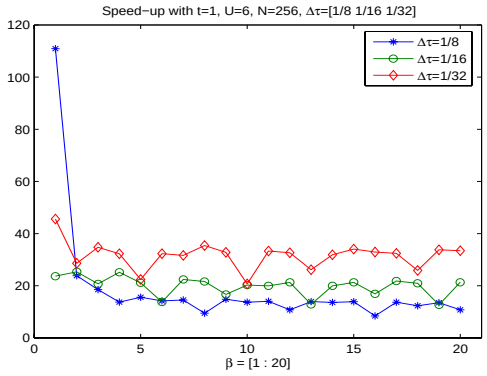
The left of the following plots shows that the relative errors of the computed solution are all under 10^{-8} as prescribed. The right plots shows the speedups of the SABO method over BOF. *can we have the detail data too?*



Experiment 3. In this experiment, we examine computational efficiency of the SABO solver with respect to the parameter $L = \beta/\Delta\tau$ with $\beta = 1, 2, \ldots, 20.$ and $\Delta\tau = \frac{1}{8}, \frac{1}{16}, \frac{1}{32}$. The dimensions of the coefficient matrices M vary from $2,048$ ($\beta = 1, \Delta\tau = \frac{1}{8}$) to $163,840$ ($\beta = 20, \Delta\tau = \frac{1}{32}$). The other parameters are $N = 256$, $U = 6$ and $t = 1$.

In Experiment 2, we have shown that the performance of the SABO solver for a fixed $\Delta\tau$. For large energy scale parameters t, β and U, small $\Delta\tau$ is necessary for the desired accuracy of the Trotter decomposition. Small $\Delta\tau$ implies large $L = \frac{\beta}{\Delta\tau}$. For the SABO solver, small $\Delta\tau$ implies a large reduction factor k. Therefore the SABO is more efficient for small $\Delta\tau$ as shown in the following table and plot:

	$\Delta\tau = 1/8$			$\Delta\tau = 1/16$			$\Delta\tau = 1/32$		
β	BOF	SABO	L_k	BOF	SABO	L_k	BOF	SABO	L_k
1	3.25	0.0293	1	7.25	0.306	2	15.5	0.34	2
2	7.28	0.305	2	15.1	0.596	3	32.9	1.15	4
3	11.2	0.605	3	23.0	1.11	4	47.3	1.36	5
4	15.1	1.10	4	32.0	1.27	5	63.6	1.97	7
5	19.2	1.23	5	39.1	1.85	7	80.3	3.58	8
6	23.0	1.62	6	47.2	3.43	8	97.9	3.03	10
7	27.2	1.87	7	55.4	2.47	9	112	3.54	11
8	32.1	3.38	8	63.4	2.93	10	140	3.95	13
9	35.3	2.38	9	71.1	4.26	12	150	4.57	14
10	39.1	2.86	10	79.3	3.91	13	167	8.06	16
11	43.2	3.08	11	87.6	4.39	14	180	5.40	17
12	47.2	4.39	12	95.7	4.50	15	196	6.00	19
13	51.7	3.71	13	103	8.00	16	209	7.99	20
14	55.3	4.06	14	112	5.61	18	224	7.03	22
15	59.2	4.26	15	120	5.64	19	240	7.05	23
16	63.5	7.54	16	128	7.58	20	258	7.83	25
17	67.3	4.92	17	136	6.23	21	273	8.42	26
18	71.2	5.78	18	144	6.88	23	290	11.2	28
19	75.3	5.58	19	152	12.0	24	305	9.03	29
20	79.3	7.36	20	160	7.49	25	321	9.60	31



Experiment 4. Finally, we examine the memory limit with respect to the increase of the lattice size parameter N. The memory requirement of the BOF method is $3N^2L = 3N_x^4L$. If $N_x = N_y = 32$, the memory storage of one $N \times N$ matrix is 8Mb. Therefore for a 1.5GB memory machine, $L < 63$. It implies that when $\Delta\tau = \frac{1}{8}$, $\beta < 8$. Therefore, the BOF method will run out of memory when $\beta \geq 8$. On the other hand, the SABO solver works for $L = 8\beta$ and $\beta = 1, 2, \ldots, 10$ as shown in the following table, where $t = 1$ and $U = 6$.

β	L	k	L_k	BOF(sec.)	SABO (sec.)	Speedup ($\times$)
1	8	8	1	148.00	2.10	70
2	16	8	2	322.00	17.8	18
3	24	8	3	509.00	40.1	12.7
4	32	8	4	689.00	64.5	10.6
5	40	8	5	875.00	88.6	9.8
6	48	8	6	1060.00	110.00	9.6
7	56	8	7	1250.00	131.00	9.5
8	64	8	8	out of memory	150.00	
9	72	8	9	out of memory	172.00	
10	80	8	10	out of memory	200.00	

Bibliography

[1] B.L. Buzbee, G.H. Golub, and C.W. Nielson. On direct methods for solving Poisson's equations. *SIAM J.Numer. Anal.*, 7:627–656, 1970.

[2] G. Fairweather and I. Gladwell. Algorithms for almost block diagonal linear systems. *SIAM Review*, 46(1):49–58, 2004.

[3] N. Higham, Accuracy and Stability of Numerical Algorithms (Second Edition). *SIAM*, 2002

[4] B. Philippe and Y. Saad W.J. Stewart. Numerical methods in Markov chain modeling. *Operations research*, 40(6):1156–1179, 1992.

[5] W. J. Stewart. *Introduction to the numerical solution of Markov chains.* Princeton University Press, 1994.

[6] G.J. Tee. An application of p-cyclic matrices for solving periodic parabolic problems. *Numerische Mathematik*, 6:142–159, 1964.

[7] U.M.Ascher, R.M.M. Mattheij, and R.D. Russell. *Numerical solution of boundary value problems for ordinary differential equations.* Prentice-Hall, Englewood Cliffs, 1988.

[8] R.S. Varga. *Matrix iterative analysis.* Prentice-Hall, Englewood Cliffs, 1962. 2nd ed., Springer, Berlin/Heidelberg,2000.

[9] S.J. Wright. Stable parallel algorithms for two-point boundary value problems. *SIAM J. Sci. Statist. Comput.*, 13(1):742–764, 1992.

[10] S.J. Wright. A collection of problems for which gaussian elimination with partial pivoting is unstable. *SIAM J. Sci. Statist. comput.*, 14(1):231–238, 1993.

Lecture 2

Preconditioned iterative linear solvers

2.1 Introduction

As discussed in Lectures 1 and 2, one of the computational kernels of the hybrid quantum Monte Carlo (HQMC) simulation is to solve the linear system of equations of the form

$$Ax = b, \tag{2.1.1}$$

where

$$A = M^T M$$

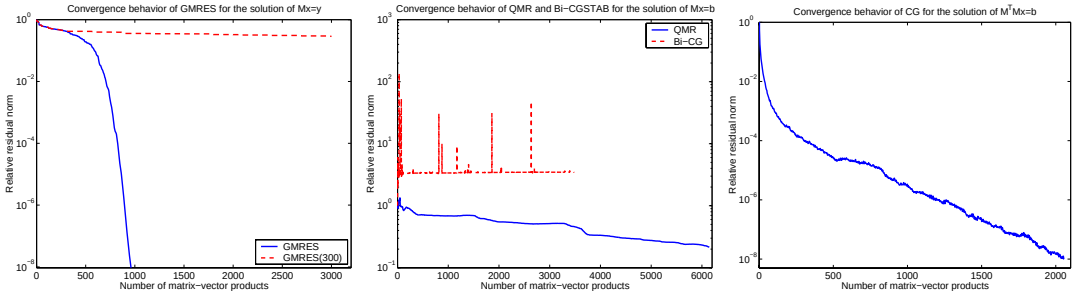
and M is the Hubbard matrix as defined in (**??**).

The SABO method introduced in Lecture 3 solves (2.1.1) with the computational complexity of $O(N^3 L/k)$, where k the reduction factor of the BCR, $k \leq L$. In this lecture, we consider precoditioned iterative solvers for the goal towarding an optimal linear-scaling solver, namely, the computational complexity increases linearly with the lattice size N, i.e., $O(NL)$.

2.2 Iterative solvers and preconditioning

We have conducted a preliminary study of applying the GMRES, QMR, and Bi-CGSTAB methods to solve the coupled linear systems $M^T y = b$ for y and $Mx = y$ for x. When there is no preconditioning, these methods suffer from slow convergence rates or erratic convergence behaviors. On the other hand, although the convergence of conjugate gradient (CG) to directly solve the original system (2.1.1) is slow, it is robust in the sense that the norm of its residual errors decrease steadly.

The following plots show the typical convergence behaviors of GMRES, QMR, Bi-CGSTAB and CG methods. The parameters of the matrices M are $(N, L, U, t, \beta, \mu) = (8 \times 8, 24, 4, 1, 3, 0)$ and the configurations $h = \pm 1$ with equal probability, and the RHS vector is chosen so that entries of x is uniformly distributed in $[0, 1]$.



In this lecture, we focus on the study of preconditioning techniques for the CG method. As it is well known, the convergence rate of CG is typically improved by a proper preconditioner R, which symmetrically preconditions the system (2.1.1):

$$R^{-1}AR^{-T} \cdot R^T x = R^{-1}b. \tag{2.2.2}$$

An ideal preconditioner R satisfies the following three conditions:

1) The cost of constructing R is affordable.

2) The application of R, i.e., solving $Rz = r$ for z, is not expensive.

3) RR^T is a good approximation of A.

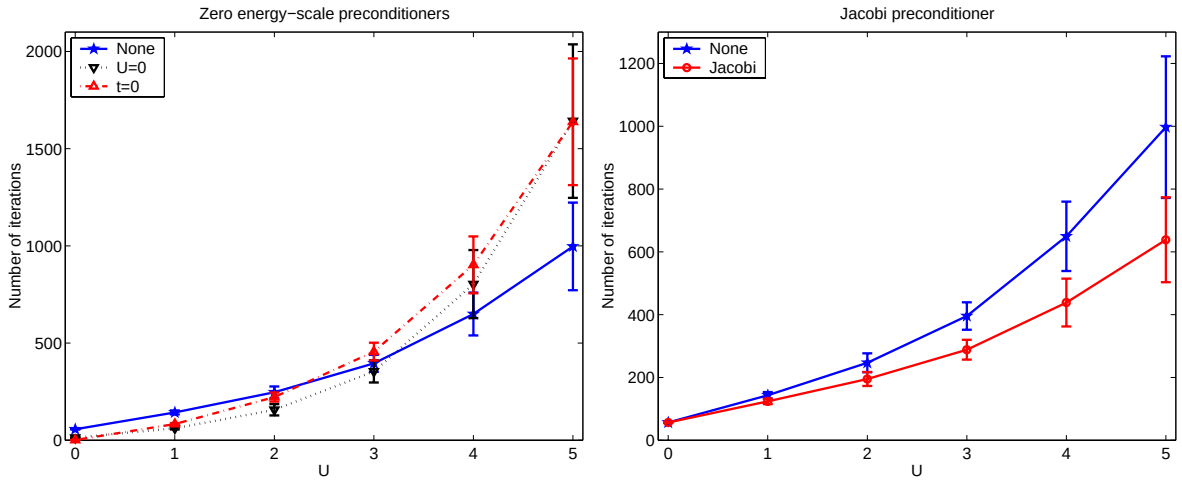
There is a trade-off between the costs (conditions 1 and 2) and the quality (condition 3). We shall search for an optimal balance between the cost and the quality for solving the HQMC system (2.1.1).

Numerical settings in this lecture is as the following: all Hubbard matrices M are generated with the configurations $h_{\ell,i} = \pm 1$ with equal probability. The RHS vector b is chosen so that entries of the solution vector x is uniformly distributed in $[0, 1]$. The stopping criteria used for the PCG iteration is $\|x_k - x\|_2/\|x\|_2 \leq 10^{-3}$. Numerical experiments reported are performed on a HP Itanium2 workstation with 1.5GHz CPU and 2GB of main memory.

2.3 Previous work

There have been a few previous studies on preconditioning techniques to improve the convergence rate of preconditioned CG (PCG) for the HQMC application.

One attempt is made to precondition the system with the matrix $R = M_{(U=0)}$ [4, 19]. By using fast Fourier transform (FFT), computational cost of applying this preconditioner is of the order $\Theta(NL\log(NL))$. However, the quality of the preconditioner is poor. For strongly interacting systems with $U \geq 4$, the convergence rate can be worsened, as shown in the left of the following plots (plots are the average of 50 solutions, bars indicate the standard deviation.



It is also suggested to use the preconditioner R such that $R = M_{(t=0)}$ [4]. The B_ℓ in the matrix M becomes diagonal when $t = 0$. The application of the preconditioner is efficient with the computational complexity of $\Theta(NL)$. However, this preconditioner is also poor quality. The convergence rate is again worsened for strongly interacting systems, as shown in the left of the above plots.

Jacobi preconditioner $R = \sqrt{\text{diag}(A)}$, that has the application cost of $\Theta(NL)$, is used [12]. The PCG convergence rate is improved consistently as shown in the right of the above plots. However, the improvement is still insufficient. For example, when $(N, L, U, t, \beta) = (8 \times 8, 40, 4, 1, 5)$, PCG solver requires $1,225$ iterations and total CPU time of 0.46 seconds. If $100,000$ solutions are required per HQMC simulation, then each simulation requires 12.8 hours of CPU time. When N is increased to 32×32, the number of PCG iterations increases to $2,013$, and the total solution time becomes 11.12 seconds, thus making the simulation time to be 309 hours.

It is proposed to use an incomplete Cholesky (IC) preconditioner R such that $A \approx RR^T$ and R is lower triangular and has the same block structure of A [19]. Although the PCG convergence rate is improved considerably, IC preconditioner results in a high memory cost, i.e., $\Theta(N^2L)$ fill-ins in R, and the application of the preconditioner is expensive, i.e., $\Theta(N^2L)$ computation. Furthermore, such a preconditioner is not robust due to the pivot breakdown.

2.4 Cholesky factorization

The goal of this lecture is to search for robust and efficient IC preconditioning techniques. We begin with a review of Cholesky factorization of an $n \times n$ symmetric positive definite (SPD) matrix A:

$$A = RR^T, \tag{2.4.3}$$

where R is lower-triangular with positive diagonal entries, and it is referred to as Cholesky factor.

Algorithms. We follow the presentation in [8]. Cholesky factorization (2.4.3) can be computed by using the following partition and factorization:

$$A = \begin{bmatrix} a_{11} & \widehat{a}_1^T \\ \widehat{a}_1 & A_1 \end{bmatrix} = \begin{bmatrix} r_{11} & 0 \\ \widehat{r}_1 & I_{n-1} \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & A_1 - \widehat{r}_1 \widehat{r}_1^T \end{bmatrix} \begin{bmatrix} r_{11} & \widehat{r}_1^T \\ 0 & I_{n-1} \end{bmatrix}. \tag{2.4.4}$$

By the first columns of the both sides of the factorization, we have

$$\begin{aligned} a_{11}^2 &= r_{11}, \\ \widehat{a}_1 &= \widehat{r}_1 r_{11}. \end{aligned}$$

Therefore,

$$\begin{aligned} r_{11} &= \sqrt{a_{11}}, \\ \widehat{r}_1 &= \widehat{a}_1 / r_{11}. \end{aligned}$$

If we have a Cholesky factorization of the $(n-1) \times (n-1)$ matrix $A_1 - \widehat{r}_1 \widehat{r}_1^T$:

$$A_1 - \widehat{r}_1 \widehat{r}_1^T = R_1 R_1^T, \tag{2.4.5}$$

then the Cholesky factor R of A is given by

$$R = \begin{bmatrix} r_{11} & 0 \\ \widehat{r}_1 & R_1 \end{bmatrix}.$$

Therefore, the Cholesky decomposition can be obtained through the repeated application of (2.4.4) on (2.4.5). The resulting algorithm is referred to as *right-looking* Cholesky algorithm because after the first column r_1 of R is computed, it is used to update the matrix A_1 to compute the remaining columns of R, which are on the right side of r_1.

There is a left-looking version of the Cholesky algorithm. By comparing the jth column of the factorization (2.4.3), we have

$$a_j = \sum_{k=1}^{j} r_{jk} r_k.$$

This says that

$$r_{jj} r_j = a_j - \sum_{k=1}^{j-1} r_{jk} r_k.$$

Hence, to compute the jth column r_j of R, one first computes

$$v = a_j - \sum_{k=1}^{j-1} r_{jk} r_k,$$

and then

$$r_j(j:n) = v(j:n) / \sqrt{v(j)}.$$

The Cholesky factorizaiton can be computed starting from the 1st column through the nth colulmn. It is a *left-looking* implementation since the jth column r_j of R is computed through referencing the computed columns $r_1, r_2, \ldots, r_{j-1}$ of R, which are on the left of r_j.

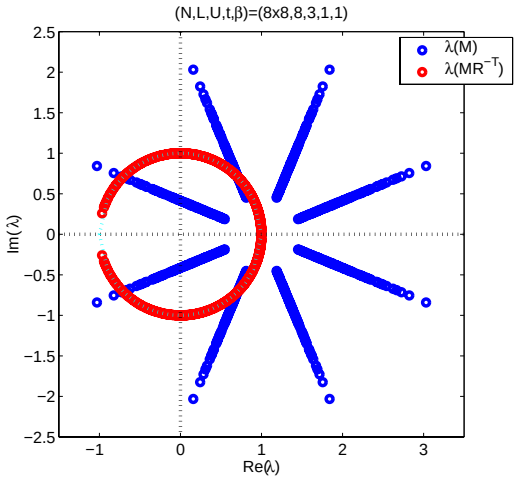
Cholesky factorization can fail due to a *pivot breakdown*, namely, at the jth step, the diagonal element a_{jj} is non-positive. Otherwise, the operations $r_{jj} = \sqrt{a_{jj}}$ and the division by r_{jj} becomes invalid. When A is SPD, the diagonal element a_{11} is always positive. Furthermore, $A_1 - \widehat{r}_1 \widehat{r}_1^T$ is SPD because it is a principal submatrix of the SPD matrix $X^T A X$, where

$$X = \begin{bmatrix} 1 & -\widehat{r}_1^T \\ 0 & I_{n-1} \end{bmatrix}.$$

Thus, when A is SPD, there is no pivot breakdown.

HQMC application. When Cholesky factor R of $A = M^T M$ is used to precondition the HQMC linear system (2.2.2), the preconditioned system is an identity system, i.e., $I \cdot R^T x = R^{-1} b$. The PCG takes one iteration to convergence. However, the cost to construct and to apply R become impractical for a large system. It is $O(N^3 L)$ flops to compute R and $O(N^2 L)$ memory to store R. For example, when $(N, L, U, t, \beta, \mu) = (16 \times 16, 40, 4, 1, 5, 0)$, PCG solver requires about 16 seconds of CPU time and about 66MB of memory to store the Cholesky factor in a sparse format. When $N = 32 \times 32$, it is estimated to require about 17 minutes of CPU time and 1GB of memory. Therefore, it is not practical to use the exact Cholesky decomposition as the linear system solver in HQMC simulation.

Note that the preconditioned matrix MR^{-T} becomes orthogonal. The eigenvalues of MR^{-T} are on the unit circle, as shown in the following plot:



Therefore, one of ways to assess the quality of a preconditioner R is to see how close the eigenvalues of the preconditioned matrix MR^{-T} are to the unit circle. However, it is prohibitive for large scale systems.

2.5 Incomplete Cholesky factorization

To reduce the computational and storage costs of Cholesky factorization (2.4.3), a preconditioner R can be constructed based on the following incomplete Cholesky (IC) factorization:

$$A = RR^T + S + S^T, \tag{2.5.6}$$

where R is lower-triangular and is referred to as an IC factor, S is a strictly lower-triangular matrix[1]. $E = S + S^T$ is the error matrix. The sparsity of R is controlled by a *sparsity set* $\mathcal{Z}$, a set of ordered pairs of integers from $\{1, 2, \ldots, n\}$ containing no pairs of the form (i, i) and $r_{ij} \neq 0$ if $(i, j) \in \mathcal{Z}$.

Algorithms. The IC factor R can be computed based on the following partition and factorization:

$$\begin{bmatrix} a_{11} & \widehat{a}_1^T \\ \widehat{a}_1 & A_1 \end{bmatrix} = \begin{bmatrix} r_{11} & 0 \\ \widehat{r}_1 & I_{n-1} \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & A_1 - \widehat{r}_1 \widehat{r}_1^T \end{bmatrix} \begin{bmatrix} r_{11} & \widehat{r}_1^T \\ 0 & I_{n-1} \end{bmatrix} + \begin{bmatrix} 0 & \widehat{s}_1^T \\ \widehat{s}_1 & 0 \end{bmatrix}. \tag{2.5.7}$$

Multiplying out the first column of the both sides, we have

$$\begin{aligned} a_{11} &= r_{11}^2, \\ \widehat{a}_1 &= \widehat{r}_1 r_{11} + \widehat{s}_1. \end{aligned}$$

Therefore, if $a_{11} > 0$,

$$r_{11} = \sqrt{a_{11}}.$$

The vector $\widehat{r}_1$ (and $\widehat{s}_1$) is computed as the following: for $i \leq n - 1$,

$$\begin{cases} \widehat{r}_1(i) = \widehat{a}_1(i)/r_{11}, & \widehat{s}_1(i) = 0, & \text{if } (i,1) \in \mathcal{Z} \\ \widehat{r}_1(i) = 0, & \widehat{s}_1(i) = \widehat{a}_1(i)/r_{11}, & \text{otherwise.} \end{cases} \tag{2.5.8}$$

If we have an IC factorization of the $(n-1) \times (n-1)$ matrix $A_1 - \widehat{r}_1 \widehat{r}_1^T$:

$$A_1 - \widehat{r}_1 \widehat{r}_1^T = R_1 R_1^T + S_1 + S_1^T, \tag{2.5.9}$$

then the IC factorization (2.5.6) of A is given by

$$R = \begin{bmatrix} r_{11} & 0 \\ \widehat{r}_1 & R_1 \end{bmatrix} \quad \text{and} \quad S = \begin{bmatrix} 0 & 0 \\ \widehat{s}_1 & S_1 \end{bmatrix}.$$

Note that when non-zero element $\widehat{a}_1(i)$ is discarded, i.e., $\widehat{r}_1(i) = 0$, in (2.5.8), the operations to update A_1 with $\widehat{r}_1(i)$ in (2.5.9) are eliminated, thus reducing the cost of both storing and computing the preconditioner R.

The following algorithm computes the IC factor R in a right-looking fashion. On the first line, $R = \text{lower}(A)$, R is initialized as the lower-triangular part of A, and the update of A_1 is performed directly in R.

[1]Therefore, the diagonal elements of A and RR^T are the same.

```
RIGHT-LOOKING IC
R = lower(A)
for j = 1, ..., n do
  r_jj = sqrt(r_jj), if r_jj > 0
  for i = j + 1, ..., n do
    if (i, j) ∈ Z then
      r_ij = r_ij / r_jj
    else
      r_ij = 0
    end if
  end for
  for k = j + 1, ..., n do
    r_k(k : n) = r_k(k : n) − r_kj r_j(k : n)
  end for
end for
```

It is assumed that all computational steps presented in the previous and the rest of algorithms presented in this lecture are performed with regard to the sparsity of matrices and vectors involved.

Alternately, there is a left-looking IC algorithm. By comparing the jth column in factorization (2.5.6), we have

$$a_{jj} = \sum_{k=1}^{j} r_{jk}^2, \tag{2.5.10}$$

$$a_j(j+1:n) - s_j(j+1:n) = \sum_{k=1}^{j} r_{jk} r_k(j+1:n). \tag{2.5.11}$$

This says that

$$r_{jj}^2 = a_{jj} - \sum_{k=1}^{j-1} r_{jk}^2,$$

$$r_{jj} r_j(j+1:n) + s_j(j+1:n) = a_j(j+1:n) - \sum_{k=1}^{j-1} r_{jk} r_k(j+1:n).$$

Thus, to compute the jth column in IC factorization, one first computes

$$v = a_j - \sum_{k=1}^{j-1} r_{jk} r_k. \tag{2.5.12}$$

Then, the jth pivot is

$$r_{jj} = \sqrt{v(j)},$$

and the rest of non-zero elements in r_j (and s_j) is computed based on the sparsity constraint $\mathcal{Z}$, i.e., for $i \geq j+1$,

$$\begin{cases} r_{ij} = v(i)/r_{jj}, & s_{ij} = 0, & (i,j) \in \mathcal{Z}, \\ r_{ij} = 0, & s_{ij} = v(i)/r_{jj}, & \text{otherwise.} \end{cases}$$

A pseudo-code of the left-looking IC algorithm is as the following:

```
LEFT-LOOKING IC ALGORITHM
for j = 1, ..., n do
  v(j : n) = a_j(j : n)
  for k = 1, .., j − 1 do
    v(j : n) = v_j(j : n) − r_jk r_k(j : n)
  end for
  r_jj = √a_jj
  for i = j + 1, ..., n do
    if (i, j) ∈ Z then
      r_ij = v(i)/r_jj
    else
      r_ij = 0
    end if
  end for
end for
```

The following conditions are often used to define the sparsity set $\mathcal{Z}$:

1. *Fixing sparsity pattern (FSP) in advance*, i.e., the IC factor R has a prescribed sparsity pattern $\mathcal{Z} = \{(i,j)\}$. $r_{ij} = 0$ if $(i,j) \notin \mathcal{Z}$.

   A popular sparsity pattern of R is that of the original matrix A, i.e., $\mathcal{Z} = \{(i,j) : a_{ij} \neq 0 \text{ and } i > j\}$.

2. *Dropping small elements (DSE)*, namely, the magnitude of the non-zero elements in the error factor S is small. In other words, all small elements of R with the relative magnitude less than or equal to a specified drop threshold σ are discarded. The sparsity pattern $\mathcal{Z}$ of R is not known in advance. The fill-ins in R need to be computed and compared with the drop threshold σ. The construction of R could be expensive. Furthermore, the number of fill-ins in R cannot be estimated in advance.

3. *Fixing number of non-zero elements (FNE) per column.* It is similar to the DSE constraint except that it keeps a fixed number of non-zero elements of the largest magnitudes in each column of R.

The existence of IC factorization, for an arbitrary sparsity set $\mathcal{Z}$, is theoretically proven only for special classes of matrices [14, 13, 23]. For a general SPD matrix A, the non-zero elements introduced into the error matrix E could result in the loss of positive-definiteness of the matrix $A - E$, and the IC factorization does not exist.

HQMC application. Let us show the numerical results of the left-looking IC preconditioner R with DSE sparsity $\mathcal{Z}$. The Hubbard matrix M is generated with $(N, L, t, \beta, \mu) = (16 \times 16, 80, 1, 10, 0)$. The reported data is an average of successful solutions over 10 trials. The following table shows memory requirements in MB for storing R, with respect to different interaction parameter U and the drop tolerance σ. "$--$" indicates that all of 10 solutions failed due to the pivot breakdown of IC.

▷ $(N, L, t, \beta) = (16 \times 16, 80, 1, 10)$. IC with DSE, Memory requirements ◁

U	0	2	4	6
Chol	133.31	133.31	133.31	133.31
$\sigma = 10^{-6}$	126.60	127.39	127.24	127.13
10^{-5}	103.33	117.57	115.94	$--$
10^{-4}	43.80	64.75	$--$	$--$
10^{-3}	$--$	$--$	$--$	$--$
10^{-2}	4.36	$--$	$--$	$--$
10^{-1}	$--$	$--$	$--$	$--$
Jacobi	0.33	0.33	0.33	0.33

Correspondingly, the number of PCG iterations are shown below:

▷ $(N, L, t, \beta) = (16 \times 16, 80, 1, 10)$. IC with DSE, PCG iters. ◁

U	0	2	4	6
$\sigma_1 = 10^{-6}$	1	2	4	7
10^{-5}	2	3	13	$--$
10^{-4}	6	16	$--$	$--$
10^{-3}	$--$	$--$	$--$	$--$
10^{-2}	32	$--$	$--$	$--$
10^{-1}	$--$	$--$	$--$	$--$
Jacobi	157	1,292	10,843	19,560

Finally, the following table records the CPU time, where in each cell, the first number is the time for the construction of the IC preconditioner R and the second number is the time of PCG iteraiton.

▷ $(N, L, t, \beta) = (16 \times 16, 80, 1, 10)$. IC with DSE, CPU time ◁

U	0	2	4	6
Chol	20.37/0.07	20.43/0.07	20.44/0.07	20.43/0.07
$\sigma_1 = 10^{-6}$	18.97/0.07	19.17/0.14	19.09/0.30	19.07/0.48
10^{-5}	13.20/0.11	16.92/0.20	16.41/0.80	$--$
10^{-4}	2.95/0.17	5.72/0.58	$--$	$--$
10^{-3}	$--$	$--$	$--$	$--$
10^{-2}	0.01/0.16	$--$	$--$	$--$
10^{-1}	$--$	$--$	$--$	$--$
Jacobi	0.00/0.23	0.00/1.90	0.00/15.92	0.00/28.86

By these tables, we observe that the IC factorization breaks down frequently, especially in the case of strongly interactive system, i.e., $U \geq 4$. It clearly indicates that the IC preconditioner R is not a robust preconditioner for the HQMC simulation.

Modified IC. To overcome the pivot breakdown, one can try to first make a small perturbation of A, say by simple diagonal perturbation, and then compute its IC factorization:

$$A + \alpha D_A = RR^T + S + S^T, \tag{2.5.13}$$

where $D_A = \text{diag}(A)$. The scalar α is chosen to avoid the breakdown of IC. R is referred to as an IC_p preconditioner [13].

If the shift α is chosen is such that $A + \alpha D_A$ is diagonally dominant, then it is provable that the IC factorization (2.5.13) exists. The following table records the performance of IC_p based on the left-looking algorithm:

$\triangleright$ $(N, L, t, \beta) = (16 \times 16, 80, 1, 10)$. IC_p, diagonal dominant, $\sigma = 0.007$ $\triangleleft$

U	0	1	2	3	4	5	6
Shift α	0.74	1.63	2.11	2.49	3.11	3.46	3.90
Storage R (MB)	3.53	3.21	3.15	3.04	2.81	2.80	2.80
Workspace W (MB)	1.48	1.37	1.35	1.31	1.24	1.23	1.23
PCG iters	69	304	826	3,184	7,753	11,331	14,756
P-time	0.07	0.06	0.06	0.12	0.05	0.05	0.05
S-time	0.25	1.07	2.88	1.77	25.62	37.48	48.88
Total CPU	0.32	1.13	2.93	1.89	25.67	37.53	48.94

With the choice of the shift α such that $A + \alpha \cdot \text{diag}(A)$ is diagonally dominant, the pivot breakdown is avoided. However, the quality of the resulting preconditioner R is poor. In practice, better performance can be easily achieved with the shift α, which is much smaller than the one to make $\widehat{A}$ diagonally dominant. The following table records the significant performance improvements of the IC_p preconditioner R computed with the shift $\alpha = 0.007$:

$\triangleright$ $(N, L, t, \beta) = (16 \times 16, 80, 1, 10)$. IC_p, $\alpha = 0.007$, $\sigma = 0.007$ $\triangleleft$

U	0	1	2	3	4	5	6
Storage R (MB)	4.52	5.34	5.48	5.53	5.54	5.53	5.50
Workspace W (MB)	1.81	2.08	2.13	2.14	2.15	2.14	2.13
PCG iters	16	41	99	314	643	932	1,089
P-time	0.09	0.12	0.12	0.12	0.12	0.12	0.12
S-time	0.08	0.23	0.55	1.77	3.82	5.25	6.11
Total CPU	0.18	0.35	0.68	1.89	3.95	5.37	6.23

Unfortunately, there is no general approach for an optimal choice of the shift α. It is computed by a trial-and-error approach in PETSc [17].

2.6 Robust Incomplete Cholesky preconditioners

In the IC factorization (2.5.6), the discarded elements of R are simply moved to the error matrix E. It may result in the loss of the positive definiteness of the matrix $A - E$, and lead to the pivot breakdown.

To avoid the pivot breakdown, the error matrix E needs to come into the picture. It should be updated dynamically during the construction of the IC factorization such that the matrix $A - E$ is preserved to be symmetric positive definite. Specifically, we seek an IC factorization satisfying that

> for an arbitrary sparsity set $\mathcal{Z}$, there exists a nonsingular lower triangular matrix R of the sparsity pattern $\mathcal{Z}$, such that
>
> $$A = RR^T + E, \tag{2.6.14}$$

i.e. $A - E > 0$.

In the rest of this lecture, we will discuss several approaches to construct an IC factor R to satisfy (2.6.14). The resulting preconditioner R is referred to as a *robust incomplete Cholesky (RIC) preconditioner*.

2.6.1 RIC1

A sufficient condition for the existence of an IC factorization is to ensure that the error matrix $-E$ is symmetric positive semi-definite, i.e. $-E = -E^T \geq 0$. If we write

$$E = S - D + S^T,$$

where S is strictly lower-triangular and D is diagonal, then an IC factorization can be constructed such that

$$\begin{cases} & A = RR^T + S - D + S^T \\ \text{s.t.} & -(S - D + S^T) \geq 0, \end{cases} \tag{2.6.15}$$

where R is lower-triangular. The factorization (2.6.15) is referred to as RIC–version 1, or RIC1 in short.

Algorithms. The RIC1 factorization can be computed by using the following partition and factorization:

$$\begin{bmatrix} a_{11} & \widehat{a}_1^T \\ \widehat{a}_1 & A_1 \end{bmatrix} = \begin{bmatrix} r_{11} & 0 \\ \widehat{r}_1/r_{11} & I_{n-1} \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & C_1 \end{bmatrix} \begin{bmatrix} r_{11} & \widehat{r}_1^T/r_{11} \\ 0 & I_{n-1} \end{bmatrix} + \begin{bmatrix} -d_{11} & \widehat{s}_1^T \\ \widehat{s}_1 & -D_1 \end{bmatrix}, \tag{2.6.16}$$

where $C_1 = A_1 + D_1 - \widehat{r}_1\widehat{r}_1^T/r_{11}^2$. By the factorization, we see that $\widehat{r}_1$ (and $\widehat{s}_1$) can be computed by dropping small elements of $\widehat{a}_1$, i.e., for $i \leq n-1$,

$$\begin{cases} \widehat{r}_1(i) = \widehat{a}_1(i), & \widehat{s}_1(i) = 0, & \text{if } \tau_{i1} > \sigma, \\ \widehat{r}_1(i) = 0, & \widehat{s}_1(i) = \widehat{a}_1(i), & \text{otherwise}, \end{cases} \tag{2.6.17}$$

where σ is the drop threshold,

$$\tau_{i1} = \left[\frac{\widehat{a}_1(i)^2}{(a_{11} + d_{11})(a_{ii}^{(1)} + d_{ii}^{(1)})} \right]^{1/2},$$

and $a_{ii}^{(1)}$ and $d_{ii}^{(1)}$ are the ith diagonal elements of A_1 and D_1, respectively.

To ensure $-E = -(S - D + S^T) \geq 0$, when there is a discarded element $\widehat{a}_1(i)$ assigned to $\widehat{s}_1(i)$, the diagonal elements d_{11} and $d_{ii}^{(1)}$ are updated

$$d_{11} := d_{11} + \delta_{11}, \quad d_{ii}^{(1)} := d_{ii}^{(1)} + \delta_{ii}, \tag{2.6.18}$$

where δ_{11} and δ_{ii} are chosen such that $\delta_{11}, \delta_{ii} > 0$ and $\delta_{11}\delta_{ii} = \widehat{s}_1(i)^2$. Initially, it is set that $d_{11} = d_{ii}^{(1)} = 0$.

Subsequently, the pivot r_{11} is determined by

$$r_{11} = \sqrt{a_{11} + d_{11}}.$$

If we have RIC1 factorization of the $(n-1) \times (n-1)$ matrix C_1:

$$C_1 = R_1 R_1^T + S_1 - \widehat{D}_1 + S_1^T, \tag{2.6.19}$$

then the RIC1 factorization (2.6.15) is given by

$$R = \begin{bmatrix} r_{11} & 0 \\ \widehat{r}_1/r_{11} & R_1 \end{bmatrix}, \quad D = \begin{bmatrix} d_{11} & 0 \\ 0 & D_1 + \widehat{D}_1 \end{bmatrix}, \quad S = \begin{bmatrix} 0 & 0 \\ \widehat{s}_1 & S_1 \end{bmatrix}.$$

Thus, RIC1 decomposition can be obtained through the repeated application of (2.6.16) on (2.6.19).

The following algorithm computes RIC1 factor R in right-looking fashion. In the algorithm, $\delta_{ii} = \tau_{ij}(a_{ii} + d_{ii})$ and $\delta_{jj} = \tau_{ij}(a_{jj} + d_{jj})$ are chosen so that they result in a same factor of increase, i.e., $(1 + \tau_{ij})$, in the corresponding diagonal elements.

RIGHT-LOOKING RIC1 ALGORITHM
$R = \text{lower}(A)$
$D = 0$ (default)
for $j = 1, \ldots, n$ **do**
  for $i = j+1, \ldots, n$ **do**
    $\tau_{ij} = |r_{ij}|/[(r_{ii} + d_{ii})(r_{jj} + d_{jj})]^{1/2}$
    if $\tau_{ij} \le \sigma$ **then**
      $r_{ij} = 0$
      $d_{ii} = d_{ii} + \tau_{ij}(r_{ii} + d_{ii})$
      $d_{jj} = d_{jj} + \tau_{ij}(r_{jj} + d_{jj})$
    end if
  end for
  $r_{jj} = \sqrt{r_{jj} + d_{jj}}$
  for $i = j+1, \ldots, n$ **do**
    $r_{ij} = r_{ij}/r_{jj}$
  end for
  for $k = j+1, \ldots, n$ **do**
    $r_k(k:n) = r_k(k:n) - r_{kj}\, r_j(k:n)$
  end for
end for

Alternatively, RIC1 can be computed by a left-looking algorithm. By comparing the jth column of (2.6.15), we have

$$\begin{aligned} a_{jj} &= \sum_{k=1}^{j} r_{jk}^2 - d_{jj}, \\ a_j(j+1:n) &= \sum_{k=1}^{j} r_{jk} r_k(j+1:n) + s_j(j+1:n). \end{aligned}$$

This says that

$$r_{jj}^2 - d_{jj} = a_{jj} - \sum_{k=1}^{j-1} r_{jk}^2,$$

$$r_{jj} r_j(j+1:n) + s_j(j+1:n) = a_j(j+1:n) - \sum_{k=1}^{j-1} r_{jk} r_k(j+1:n).$$

Thus one first computes

$$v = a_j - \sum_{k=1}^{j-1} r_{jk}\, r_k.$$

Then, small elements of v are kept in the jth column s_j, i.e., for $i \geq j+1$,

$$\begin{cases} s_{ij} = 0, & \text{if } \tau_{ij} > \sigma, \\ s_{ij} = v(i), \quad v(i) = 0, & \text{otherwise}, \end{cases} \tag{2.6.20}$$

where

$$\tau_{ij} = \left[\frac{v(i)^2}{(a_{ii} + d_{ii})(a_{jj} + d_{jj})} \right]^{1/2}.$$

To ensure $-E = -(S - D + S^T) \geq 0$, when there is a discarded element assigned to s_{ij}, the corresponding diagonal elements d_{ii} and d_{jj} are updated

$$d_{ii} := d_{ii} + \delta_{ii}, \quad d_{jj} := d_{jj} + \delta_{jj}, \tag{2.6.21}$$

where δ_{ii} and δ_{jj} are chosen such that $\delta_{ii}, \delta_{jj} > 0$ and $\delta_{ii}\delta_{jj} = s_{ij}^2$. Initially, $d_{11} = d_{ii}^{(1)} = 0$. Subsequently, the jth column r_j of the IC factor R is given by

$$\begin{aligned} r_{jj} &= \sqrt{a_{jj} + d_{jj}}, \\ r_j(j+1:n) &= v(j+1:n)/r_{jj}. \end{aligned}$$

Finally, for every non-zero element r_{ij} in r_j, the corresponding diagonal element a_{ii} is updated, i.e., for our illustration, this update is performed on d_{ii},

$$d_{ii} := d_{ii} - r_{ij}^2.$$

The following algorithm computes RIC1 factor R in a left-looking fashion.

LEFT-LOOKING RIC1 ALGORITHM
$D = 0$ (default)
for $j = 1, \ldots, n$ **do**
  $v(j:n) = a_j(j:n)$
  for $k = 1, \ldots, j-1$ **do**
    $v(j+1:n) = v(j+1:n) - r_{jk} r_k(j+1:n)$
  end for
  for $i = j+1, \ldots, n$ **do**
    $\tau_{ij} = |v(i)|/[(a_{ii} + d_{ii})(a_{jj} + d_{jj})]^{1/2}$
    if $\tau_{ij} \le \sigma$ **then**
      $v(i) = 0$
      $d_{ii} = d_{ii} + \tau_{ij}(a_{ii} + d_{ii})$
      $d_{jj} = d_{jj} + \tau_{ij}(a_{jj} + d_{jj})$
    end if
  end for
  $r_{jj} = \sqrt{a_{jj} + d_{jj}}$
  for $i = j+1, \ldots, n$ **do**
    $r_{ij} = v(i)/r_{jj}$
    $d_{ii} = d_{ii} - r_{ij}^2$
  end for
end for

The RIC1 preconditioning techniques are first studied in [1, 9]. It is a simple diagonal updating scheme. The construction cost is only slightly higher than the IC preconditioner. To measure the quality of RIC1 preconditioner R, we note that the norm of the residue

$$R^{-1} A R^{-T} - I = -R^{-1}(S - D + S^T)R^{-T} \tag{2.6.22}$$

could be amplified by a factor of $\|R^{-1}\|^2$ of the error matrix $E = S - D + S^T$. When a large number of diagonal updates are introduced, The norm of E could be large. Therefore, the quality of the RIC1 preconditioner could be poor as we have seen in our HQMC application.

HQMC application. We examine the performance of the RIC1 preconditioner R for different interaction parameter U. The dropping threshold for the small element is $\sigma = 0.005$. With this dropping tolerance, the resulting RIC1 preconditioner R is of about the same sparsity as IC_p preconditioner in Section 2.5.

$\triangleright$ $(N, L, t, \beta) = (16 \times 16, 80, 1, 10)$. RIC1, right-looking, $\sigma = 0.005$ $\triangleleft$

U	0	1	2	3	4	5	6
Storage R (MB)	4.48	5.20	5.25	5.25	5.21	5.15	5.09
Workspace (MB)	1.63	1.87	1.89	1.89	1.87	1.85	1.83
PCG itrs.	21	57	135	491	1,086	1,466	1,873
P-time	0.21	0.24	0.25	0.25	0.25	0.24	0.24
S-time	0.10	0.30	0.71	2.55	5.62	7.55	9.59
Total CPU	0.32	0.54	0.96	2.80	5.86	7.79	9.83

We note that the RIC1 preconditioner based PCG is slower than the IC_p preconditioner based PCG, see section 2.7. However, RIC1 is provable robust.

2.6.2 RIC2

Tismenetsky proposed a way to improve the quality of the RIC1 preconditioner R [21]. Instead of writing the error matrix E as $E = S - D + S^T$, it begins with setting the error matrix

$$E = RF^T + FR^T,$$

and compute an IC decomposition of the form

$$A = RR^T + RF^T + FR^T, \tag{2.6.23}$$

where R is lower-triangular, and F is strictly lower-triangular. Note that the factorization (2.6.23) can be equivalently written as

$$A + FF^T = (R + F)(R + F)^T$$

Therefore, the existence of R for an arbitrary sparsity constrant $\mathcal{Z}$ is guaranteed.

With the factorization (2.6.23), the residue becomes

$$R^{-1}AR^{-T} - I = -FR^{-T} - R^{-1}F^T.$$

Hence, the norm of the residue is amplified by the order of $\|R^{-1}\|$, instead of $\|R^{-1}\|^2$ as in the RIC1. We refer (2.6.23) as RIC–version 2 factorization, or RIC2 in short.

Algorithms. RIC2 factorization (2.6.23) can be constructed by using the following partition and factorization:

$$\begin{aligned} A &= \begin{bmatrix} a_{11} & \widehat{a}_1^T \\ \widehat{a}_1 & A_1 \end{bmatrix} \\ &= \begin{bmatrix} r_{11} & 0 \\ \widehat{r}_1 & I_{n-1} \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & C_1 \end{bmatrix} \begin{bmatrix} r_{11} & \widehat{r}_1^T \\ 0 & I_{n-1} \end{bmatrix} + \begin{bmatrix} r_{11} & 0 \\ \widehat{r}_1 & 0 \end{bmatrix} \begin{bmatrix} 0 & \widehat{f}_1^T \\ 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ \widehat{f}_1 & 0 \end{bmatrix} \begin{bmatrix} r_{11} & \widehat{r}_1^T \\ 0 & 0 \end{bmatrix} \end{aligned} \tag{2.6.24}$$

where $C_1 = A_1 - \widehat{r}_1\widehat{r}_1^T - \widehat{r}_1\widehat{f}_1^T - \widehat{f}_1\widehat{r}_1^T$. By the factorization, we have

$$r_{11} = \sqrt{a_{11}}.$$

The vectors $\widehat{r}_1$ and $\widehat{f}_1$ are computed by dropping small elements of $\widehat{a}_1(i)/r_{11}$, i.e., for $i \le n-j$,

$$\begin{cases} \widehat{r}_1(i) = \widehat{a}_1(i)/r_{11}, & \widehat{r}_1(i) = 0, & \text{if } |\widehat{a}_1(i)|/r_{11} > \sigma, \\ \widehat{f}_1(i) = 0, & \widehat{f}_1(i) = \widehat{a}_1(i)/r_{11}, & \text{otherwise,} \end{cases}$$

If an RIC2 factorization of the $(n-1) \times (n-1)$ matrix C_1 is given by

$$C_1 = R_1R_1^T + R_1F_1^T + F_1R_1^T, \tag{2.6.25}$$

then the RIC2 factorization of A is given by

$$R = \begin{bmatrix} r_{11} & 0 \\ \widehat{r}_1 & R_1 \end{bmatrix}, \qquad F = \begin{bmatrix} 0 & 0 \\ \widehat{f}_1 & F_1 \end{bmatrix}.$$

Thus, RIC2 decomposition can be obtained through the repeated application of (2.6.24) on (2.6.25).

The following algorithm computes the RIC2 factor R in a right-looking fashion, where f_1 is discarded after it is used to update A_1:

RIGHT-LOOKING RIC2 ALGORITHM
$R = \text{lower}(A)$
for $j = 1, \ldots, n$ **do**
  $r_{jj} = \sqrt{r_{jj}}$
  for $i = j+1, \ldots, n$ **do**
    if $|a_{ij}|/r_{jj} > \sigma$ **then**
      $r_{ij} = r_{ij}/r_{jj}$
      $f_{ij} = 0$
    else
      $r_{ij} = 0$
      $f_{ij} = r_{ij}/r_{jj}$
    end if
  end for
  for $k = j+1, \ldots, n$ **do**
    $r_k(k:n) = r_k(k:n) - r_{kj}\,(r_j(k:n) + f_j(k:n))$
    $r_k(k:n) = r_k(k:n) - f_{kj}\, r_j(k:n)$
  end for
end for

Alternatively, the RIC2 can be computed by a left-looking algorithm. By comparing the jth column in factorization (2.6.23), we have

$$a_j = \sum_{k=1}^{j} (r_{jk} r_k + r_{jk} f_k + f_{jk} r_k). \tag{2.6.26}$$

This says that

$$r_{jj}(r_j + f_j) = a_j - \sum_{k=1}^{j-1} (r_{jk} r_k + r_{jk} f_k + f_{jk} r_k).$$

Thus, to compute the jth column in RIC2 factorization, one first computes

$$v = a_j - \sum_{k=1}^{j-1} (r_{jk} r_k + r_{jk} f_k + f_{jk} r_k). \tag{2.6.27}$$

Then, the jth pivot is given by

$$r_{jj} = \sqrt{v(j)},$$

and the rest of the non-zero elements in r_j and f_j are computed by dropping small elements of v, i.e., for $i \geq j+1$,

$$\begin{cases} r_{ij} = v(i)/r_{jj}, & f_{ij} = 0, & \text{if } |v(i)|/r_{jj} > \sigma, \\ r_{ij} = 0, & f_{ij} = v(i)/r_{ii}, & \text{otherwise.} \end{cases}$$

The following algorithm computes the RIC2 factor R in a left-looking fashion.

LEFT-LOOKING RIC2 ALGORITHM
for $j = 1, \ldots, n$ **do**
  $v(j:n) = a_j(j:n)$
  for $k = 1, .., j-1$ **do**
    $v(j:n) = v(j:n) - r_{jk}(r_k(j:n) + f_k(j:n))$
    $v(j:n) = v(j:n) - f_{jk} r_k(j:n)$
  end for
  $r_{jj} = \sqrt{v(j)}$
  for $i = j+1, \ldots, n$ **do**
    if $|v(i)|/r_{jj} > \sigma$ **then**
      $r_{ij} = v(i)/r_{jj}$
      $f_{ij} = 0$
    else
      $r_{ij} = 0$
      $f_{ij} = v(i)/r_{jj}$
    end if
  end for
end for

Notice that in the above algorithm, the columns $f_1, \ldots, f_{j-1}$ are required to compute r_j.

When A is SPD, the diagonal element $a_{11} > 0$ and

$$C_1 = A_1 - \frac{\widehat{a}_1 \widehat{a}_1^T}{a_{11}} + \widehat{f}_1 \widehat{f}_1^T > 0.$$

Thus, the pivot breakdown is avoided. In other words, the matrix $A - E$ of RIC2 factorization (2.6.23) is SPD,

$$A - RF^T - FR^T > 0, \tag{2.6.28}$$

and the factorization is robust.

HQMC application. The following table shows the numerical results with RIC2 preconditioner computed by the left-looking algorithm. It is under the same setting as in RIC1, except the drop threshold for the small elements is $\sigma = 0.012$. With this drop threshold, the resulting RIC2 preconditioner R is of about the same sparsity as the RIC1 preconditioner.

$\triangleright$ $(N, L, t, \beta) = (16 \times 16, 80, 1, 10)$. RIC2, left-looking, $\sigma = 0.012$ $\triangleleft$

U	0	1	2	3	4	5	6
Storage R (MB)	4.82	5.08	5.18	5.26	5.30	5.29	5.32
Worksspace (MB)	128.90	128.64	128.53	128.45	127.41	127.26	127.60
PCG iters	16	36	73	194	344	453	539
P-time	1.79	1.86	1.88	1.90	1.90	1.90	1.88
S-time	0.12	0.27	0.57	1.52	2.70	3.57	4.24
Total CPU	1.91	2.13	2.45	3.42	4.60	5.47	6.11

We note that the quality of RIC2 is much better than RIC1, as indicated by the number of PCG iterations and total CPU time. However, we also note that the costs of CPU and

workspace to construct RIC2 preconditioner increase significantly. This potentially limits the applicability of the RIC2 for large scale systems.

The right-looking algorithm reduces the workspace by a factor of more than 10 as shown in the following table, but with significantly more CPU time in computing the preconditioner. Therefore, the right-looking algorithm is not competitive in terms of the total CPU cost.

$\triangleright$ $(N, L, t, \beta) = (16 \times 16, 80, 1, 10)$. RIC2, right-looking, $\sigma = 0.012$ $\triangleleft$

U	0	1	2	3	4	5	6
Storage R (MB)	4.82	5.08	5.18	5.26	5.30	5.29	5.32
Workspace (MB)	10.65	10.63	10.63	10.63	10.63	10.63	10.17
PCG iters	16	36	73	194	344	453	539
P-time	19.29	19.29	19.31	19.31	19.34	19.28	19.19
S-time	0.12	0.27	0.57	1.52	2.70	3.57	4.24
Total CPU	19.43	19.58	19.90	20.84	22.06	22.87	23.45

Note that the left-looking and right-looking RIC2 result in the same preconditioner. Therefore, the storage requirement for the preconditioner R and the CPU time (S-time) of the PCG iterations are the same in the above two tables.

2.6.3 RIC3

Kaporin proposed a scheme to reduce the cost of computing the RIC2 factorization (2.6.23) by additional sparsity on F with a secondary dropping threshold $\sigma_2 \ll \sigma_1$ [10]. Specifically, Kaporin proposes to write the error matrix

$$E = RF^T + FR^T + \widehat{E},$$

where $\widehat{E}$ represents the error from imposing the secondary sparsity constraint. To maintain the robustness, similar to RIC1 factorization, by writing

$$\widehat{E} = S - D + S^T,$$

the diagonal elements D are updated to guarantee the semi-positive definiteness of $-\widehat{E}$.

In summary, one constructs a preconditioner R based on the following factorization

$$\begin{cases} A = RR^T + RF^T + FR^T + S - D + S^T, \\ \text{s.t.} \quad -(S - D + S^T) > 0, \end{cases} \tag{2.6.29}$$

where R is lower-triangular, F and S are strictly lower-triangular, and D is diagonal. The sparsity of R and F are controlled by the primary drop threshold σ_1 and the secondary drop threshold σ_2, respectively. We called this as the RIC–version 3, or RIC3 in short.

The RIC3 factorization (2.6.29) often results in a good preconditioner R when $\|S\|$ and $\|D\|$ are small enough. Specifically, when $\|\widehat{E}\| \leq \|F\|/\|R^{-1}\|$, the norm of the residue

$$R^{-1}AR^{-T} - I = -FR^{-T} - R^{-1}F^T - R^{-1}(S - D + S^T)R^{-T}$$

is amplified at most by the factor of about $\|R^{-1}\|$ of the norm $\|F\|$. At the same time, the cost of constructing the RIC3 preconditioner is significantly reduced from that to construct the RIC2 preconditioner.

Algorithms. The RIC3 factorization (2.6.29) can be constructed by using the following partitioning and factorization:

$$\begin{aligned} A &= \begin{bmatrix} a_{11} & \widehat{a}_1^T \\ \widehat{a}_1 & A_1 \end{bmatrix} = \begin{bmatrix} r_{11} & 0 \\ \widehat{r}_1 & I_{n-1} \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & C_1 \end{bmatrix} \begin{bmatrix} r_{11} & \widehat{r}_1^T \\ 0 & I_{n-1} \end{bmatrix} + \\ & \begin{bmatrix} r_{11} & 0 \\ \widehat{r}_1 & 0 \end{bmatrix} \begin{bmatrix} 0 & \widehat{f}_1^T \\ 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ \widehat{f}_1 & 0 \end{bmatrix} \begin{bmatrix} r_{11} & \widehat{r}_1^T \\ 0 & 0 \end{bmatrix} + \begin{bmatrix} -d_{11} & \widehat{s}_1^T \\ \widehat{s}_1 & -D_1 \end{bmatrix} \end{aligned} \tag{2.6.30}$$

where $C_1 = A_1 + D_1 - \widehat{r}_1\widehat{r}_1^T - \widehat{r}_1\widehat{f}_1 - \widehat{f}_1\widehat{r}_1^T$. Multiplying out the first column, we have

$$\begin{aligned} a_{11} &= r_{11}^2 - d_{11}, \\ \widehat{a}_1 &= \widehat{r}_1 r_{11} + \widehat{f}_1 r_{11} + \widehat{s}_1. \end{aligned}$$

If we let

$$v_1 = \widehat{r}_1 + \widehat{f}_1,$$

then the vector v_1 (and $\widehat{s}_1$) are computed by imposing the sparsity constraint on $\widehat{a}_1$ with the secondary drop tolerance σ_2, i.e., for $i \le n-1$,

$$\begin{cases} v_1(i) = \widehat{a}_1(i), & \widehat{s}_1(i) = 0, & \text{if } \tau_{i1} > \sigma_2, \\ v_1(i) = 0, & \widehat{s}_1(i) = \widehat{a}_1(i), & \text{otherwise,} \end{cases}$$

where

$$\tau_{i1} = \left[\frac{\widehat{a}_1(i)^2}{(a_{ii}^{(1)} + d_{ii}^{(1)})(a_{11} + d_{11})} \right]^{1/2},$$

and $a_{ii}^{(1)}$ and $d_{ii}^{(1)}$ denote the ith diagonal elements of A_1 and D_1, respectively. To ensure $-\widehat{E} = -(S - D + S^T) \ge 0$, when a discarded element $\widehat{a}_1(i)$ is assigned to the position $\widehat{s}_1(i)$, the corresponding diagonal element d_{11} and $d_{ii}^{(1)}$ are updated,

$$d_{11} := d_{11} + \delta_{11}, \quad d_{ii}^{(1)} := d_{ii}^{(1)} + \delta_{ii},$$

where δ_{11} and δ_{ii} are chosen such that $\delta_{11}, \delta_{ii} > 0$ and $\delta_{11}\delta_{ii} = \widehat{s}_1(i)^2$. Initially, it is set that $d_{ii} = d_{jj} = 0$.

Subsequently, the pivot r_{11} is given by

$$r_{11} = \sqrt{a_{11} + d_{11}}.$$

Finally, $\widehat{r}_1$ and $\widehat{f}_1$ is updated by imposing the primary sparsity constraint on v_1 with the dropping threshold σ_1, i.e., for $i < n - j$,

$$\begin{cases} \widehat{r}_1(i) = v_1(i)/r_{11}, & \widehat{f}_1(i) = 0, & \text{if } |\widehat{f}_1(i)|/r_{11} > \sigma_1, \\ \widehat{r}_1(i) = 0, & \widehat{f}_1(i) = v_1(i)/r_{11}, & \text{otherwise,} \end{cases}$$

Therefore, if we have RIC3 factorization of the $(n-1) \times (n-1)$ matrix C_1,

$$C_1 = R_1 R_1^T + R_1 F_1^T + F_1 R_1^T + S_1 - \widehat{D}_1 + S_1^T, \tag{2.6.31}$$

then the RIC3 factorization is given by

$$R = \begin{bmatrix} r_{11} & 0 \\ \widehat{r}_1 & R_1 \end{bmatrix}, \quad F = \begin{bmatrix} 0 & 0 \\ \widehat{f}_1 & F_1 \end{bmatrix}, \quad S = \begin{bmatrix} 0 & 0 \\ \widehat{s}_1 & S_1 \end{bmatrix}, \quad D = \begin{bmatrix} d_{11} & 0 \\ 0 & D_1 + \widehat{D}_1 \end{bmatrix}.$$

Thus, the RIC3 decomposition can be computed by the repeated application of (2.6.30) on (2.6.31).

The following algorithm computes the RIC3 factor R in a right-looking fashion, where δ_{ii} and δ_{jj} are chosen in the same way as in the RIC1 algorithms. Therefore, it results in the same factor of increase of the diagonal elements.

RIGHT-LOOKING RIC3 ALGORITHM
$R = \text{lower}(A)$, $D = 0_n$
for $j = 1, \ldots, n$ **do**
  for $i = j+1, \ldots, n$ **do**
    $\tau_{ij} = |r_{ij}|/[(a_{ii} + d_{ii})(a_{jj} + d_{jj})]^{1/2}$
    if $\tau_{ij} \le \sigma_2$ **then**
      $r_{ij} = 0$
      $d_{ii} = d_{ii} + \tau_{ij}(a_{ii} + d_{ii})$
      $d_{jj} = d_{jj} + \tau_{ij}(a_{jj} + d_{jj})$
    end if
  end for
  $r_{jj} = \sqrt{a_{jj} + d_{jj}}$
  for $i = j+1, \ldots, n$ **do**
    if $|r_{ij}|/r_{jj} > \sigma_1$ **then**
      $r_{ij} = r_{ij}/r_{jj}$
      $f_{ij} = 0$
    else
      $r_{ij} = 0$
      $f_{ij} = r_{ij}/r_{jj}$
    end if
  end for
  for $k = j+1, \ldots, n$ **do**
    $r_k(k:n) = r_k(k:n) - r_{kj}(r_j(k:n) + f_j(k:n))$
    $r_k(k:n) = r_k(k:n) - f_{kj} r_j(k:n)$
  end for
end for

In the right-looking RIC3 algorithm, the jth column f_j of F can be discarded after it is used to update the remaining column $r_{j+1}, \ldots, r_n$.

Alternatively, the RIC3 factorization can be computed by a left-looking algorithm. By comparing the jth column in factorization (2.6.29), we have

$$a_{jj} = \sum_{k=1}^{j} (r_{jk}^2 + 2 f_{jk} r_{jk}) - d_{jj},$$

$$a_j(j+1:n) = s_j(j+1:n) + \sum_{k=1}^{i} (r_{jk} r_k(j+1:n) + r_{jk} f_k(j+1:n) + f_{jk} r_k(j+1:n)).$$

This says that

$$r_{jj}^2 + d_{jj} = a_{jj} - \sum_{k=1}^{j-1}(r_{jk}^2 + 2f_{jk}r_{jk}),$$

$$r_{jj}t_j(j+1:n) + s_j(j+1:n) = a_j(j+1:n) - \sum_{k=1}^{j-1}(r_{jk}t_k(j+1:n) + f_{jk}r_k(j+1:n)),$$

where $t_k = r_k + f_k$. Thus, to compute the jth column of R, one first computes,

$$v = a_j - \sum_{k=1}^{j-1}(r_{jk}(r_k + f_k) + f_{jk}r_k).$$

Then, the sparsity of v is imposed with the secondary drop threshold σ_2, i.e. for $i \geq j+1$,

$$\begin{cases} s_{ij} = 0, & \text{if } \tau_{ij} > \sigma_2, \\ s_{ij} = v(i), \quad v(i) = 0, & \text{otherwise,} \end{cases}$$

where

$$\tau_{ij} = \left[\frac{v(i)^2}{(a_{ii} + d_{ii})(a_{jj} + d_{jj})}\right]^{1/2}.$$

To ensure $-\widehat{E} = -(S - D + S^T) \geq 0$, if a discarded element $\widehat{a}_1(i)$ is entered into the position $\widehat{s}_1(i)$, the diagonal elements d_{ii} and d_{jj} are updated,

$$d_{ii} := d_{ii} + \delta_{ii}, \quad d_{jj} := d_{jj} + \delta_{jj},$$

where δ_{ii} and δ_{jj} are chosen such that $\delta_{ii}, \delta_{jj} > 0$ and $\delta_{ii}\delta_{jj} = s_{ij}^2$. Initially, it is set that $d_{ii} = d_{jj} = 0$.

Subsquently, the jth pivot is given by

$$r_{jj} = \sqrt{v(j) + d_{jj}},$$

and the rest of non-zero elements in r_j and f_j are computed by imposing the primary sparsity constraint on v with the primary drop threshold σ_1, i.e. for $i \geq j+1$,

$$\begin{cases} r_{ij} = v(i)/r_{jj}, & f_{ij} = 0, & \text{if } |v(i)|/r_{jj} > \sigma_1 \\ r_{ij} = 0, & f_{ij} = v(i)/r_{jj}, & \text{otherwise.} \end{cases}$$

The following algorithm to compute RIC3 factor R in left-looking fashion.

LEFT-LOOKING RIC3 ALGORITHM
$D = 0_n$
for $j = 1, \ldots, n$ **do**
  $v(j:n) = a_j(j:n)$
  for $k = 1, \ldots, k-1$ **do**
    $v(j:n) = v(j:n) - r_{jk}(r_k(j:n) + f_k(j:n))$
    $v(j:n) = v(j:n) - f_{jk} r_k(j:n)$
  end for
  for $i = j+1, \ldots, n$ **do**
    $\tau_{ij} = |v(i)|/[(a_{ii} + d_{ii})(a_{jj} + d_{jj})]^{1/2}$
    if $\tau_{ij} \le \sigma_2$ **then**
      $v(i) = 0$
      $d_{ii} = d_{ii} + \tau_{ij}(a_{ii} + d_{ii})$
      $d_{jj} = d_{jj} + \tau_{ij}(a_{jj} + d_{jj})$
    end if
  end for
  $r_{jj} = \sqrt{v(j) + d_{jj}}$
  for $i = j+1, \ldots, n$ **do**
    if $|v(i)|/r_{jj} > \sigma_1$ **then**
      $r_{ij} = v(i)/r_{jj}$
      $f_{ij} = 0$
    else
      $r_{ij} = 0$
      $f_{ij} = v(i)/r_{jj}$
    end if
  end for
end for

Note that in the above algorithm, the columns $f_1, f_2, \ldots, f_{j-1}$ are needed to compute the jth column r_j.

HQMC application. The following table shows the numerical results with the RIC3 preconditioner computed by the left-looking algorithm. The experimental setting is the same as used for RIC1 and RIC2, except the drop thresholds are $\sigma_1 = 0.01$ and $\sigma_2 = \sigma_1^2$. With these drop thresholds, the resulting RIC3 preconditioner R is of about the same sparsity as the RIC1 and RIC2 preconditioners.

$\triangleright$ $(N, L, t, \beta) = (16 \times 16, 80, 1, 10)$. RIC3, left-looking, $\sigma_1 = 0.01$, $\sigma_2 = \sigma_1^2$ $\triangleleft$

U	0	1	2	3	4	5	6
Storage R (MB)	4.85	5.17	5.23	5.23	5.23	5.21	5.20
Workspace (MB)	42.15	46.17	43.47	41.09	39.13	37.36	35.85
PCG iters	13	35	80	253	500	666	803
P-time	0.86	1.14	1.08	1.03	0.99	0.95	0.92
S-time	0.07	0.19	0.44	1.38	2.73	3.63	4.37
Total CPU	0.92	1.33	1.52	2.41	3.72	4.58	5.29

The right-looking RIC3 algorithm significantly reduces the workspace as shown in the following table. but with the increase of CPU time for computing the preconditioner. Therefore, the right-looking algorithm is not competitive in term of total CPU time requriement.

$\triangleright$ $(N, L, t, \beta) = (16 \times 16, 80, 1, 10)$. RIC3, right-looking, $\sigma_1 = 0.01$, $\sigma_2 = \sigma_1^2$ $\triangleleft$

U	0	1	2	3	4	5	6
Storage R (MB)	4.85	5.17	5.23	5.23	5.23	5.21	5.20
Workspace (MB)	9.34	9.30	9.29	9.29	9.29	9.30	9.30
P-time	3.96	4.35	4.12	3.91	3.74	3.58	3.44
S-time	0.07	0.19	0.44	1.38	2.73	3.63	4.37
Total CPU	4.07	4.45	4.57	5.20	6.43	6.15	7.79

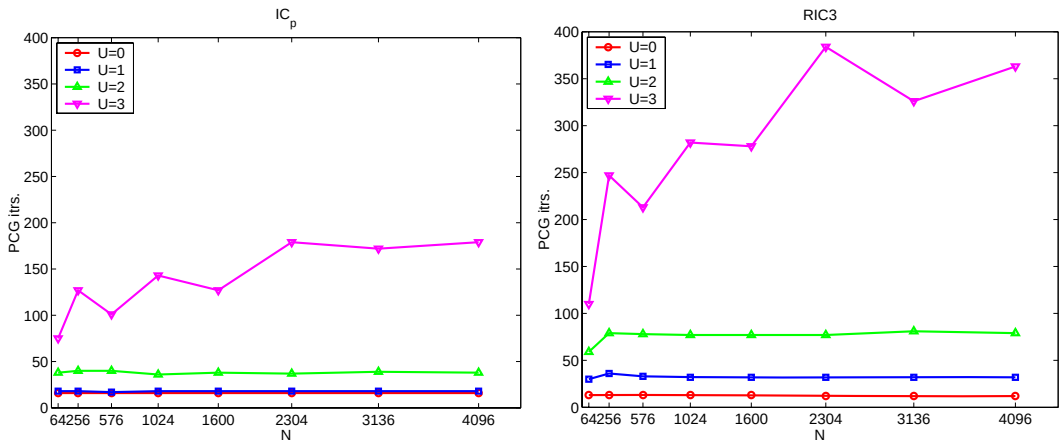
Note that the left-looking and right-looking RIC3 algorithms result in the same preconditioner. Therefore, the storage requirement for the preconditioner R and the CPU time of the PCG iterations are the same in the above two tables.

2.7 Performance evaluation

The numerical results presented in the previous sections indicate that the IC_p and RIC3 preconditioners are the most competitive ones for solving the HQMC linear system (2.1.1). In this section, we evaluate the performance of the IC_p and RIC3 preconditioners for solving HQMC linear systems (2.1.1) with respect to the different parameters.

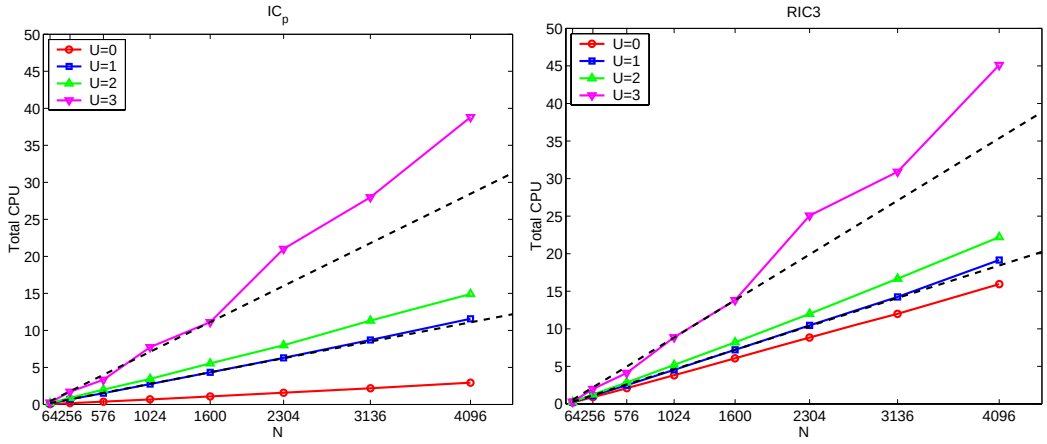
2.7.1 Moderately interacting systems with $U < 4$

We examine the performance of the PCG solver for moderate interacting systems, namely $U < 4$. The following plots show the number of PCG iterations with the IC_p preconditioner (left) and the RIC3 preconditioner (right). The plots show that for the both preconditioners, with respect to the lattice size N, the number of PCG iterations stays the same when $U = 0, 1, 2$, and grows slowly whtn $U = 3$ with the changes of the lattice size N.



Consequently, it indicates the linear-scaling of total CPU of PCG iterations. The following plots show the total CPU time of the PCG solvers for different lattice sizes N. The black dash

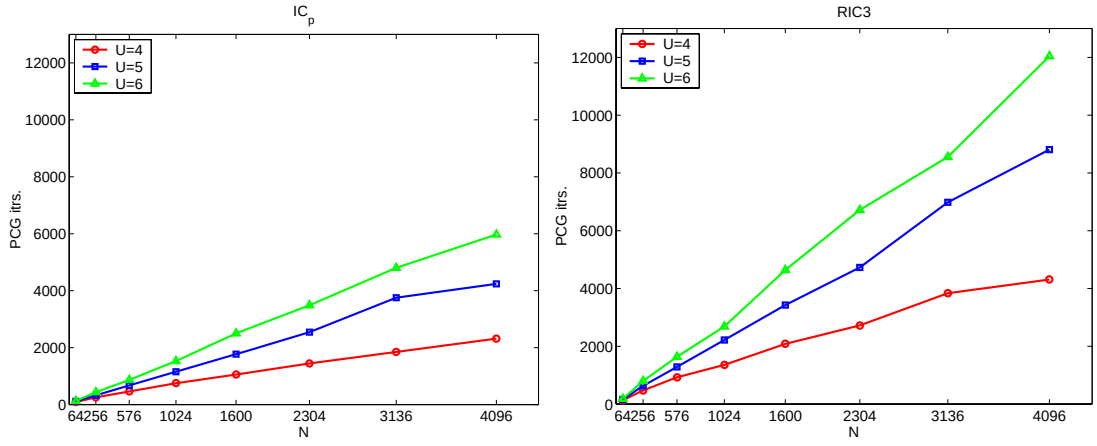
lines indicate the linear-scaling when the CPU time at $N = 40 \times 40$ is used as the reference point. The rest of parameters are $(L, t, \beta, \mu) = (80, 1, 10, 0)$.



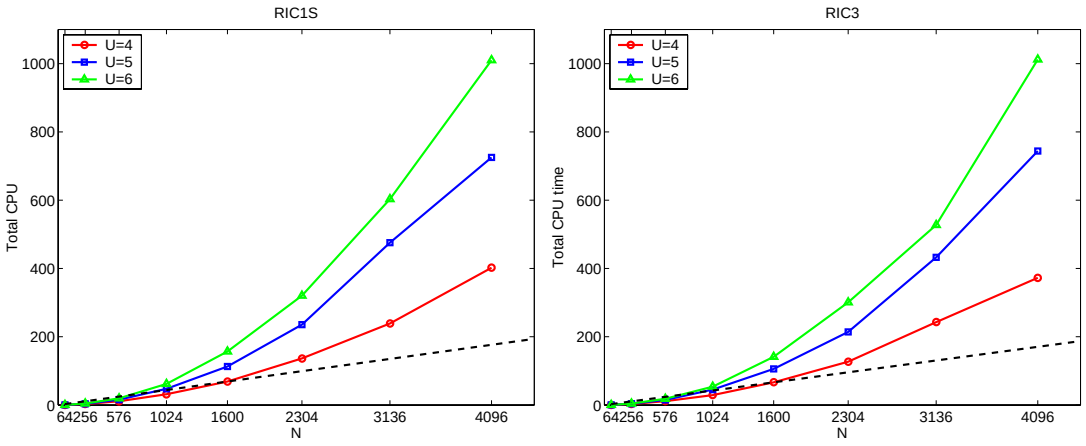
From these performance data, we conclude that when $U < 4$, the HQMC linear system (2.2.2) is relatively well-conditioned. The quality of IC_p and RIC3 preconditioners are comparable. The IC_p has twice as much fill-ins as RIC3, but the number of PCG iterations with IC_p is only half of that with RIC3. IC_p slightly outperforms RIC3.

2.7.2 Strongly interacting systems with $U \geq 4$

For *strongly interacting systems*, namely $U \geq 4$, we observe that RIC3 slightly outperforms the IC_p in terms of total CPU time of PCG solver. The following plots show the number of PCG iterations grows linearly with respect to the lattice sizes N. The rest of parameters are $(L, t, \beta, \mu) = (80, 1, 10, 0)$.



Subsequently, the total CPU time of PCG solver scales in the order of N^2, as shown in the following plots (left – IC_p, and right – RIC3). The dash line indicates the desired linear-scaling when the CPU time at $N = 40 \times 40$ is used as the reference point.



In summary, for strongly interacting system, namely $U = 4, 5$, or 6, the linear system of equations is ill-conditioned. A linear-scaling PCG solver remains an open problem.

2.7.3 Extra data

IC$_p$. For the record, the following tables are the observed optimal performance of the IC$_p$ based PCG solver with the proper chosen dropping threshold values.

$\triangleright$ IC$_p$: $(N, L, t, \beta) = (16 \times 16, 80, 1, 10)$ $\triangleleft$

U	0	1	2	3	4	5	6
Storage R (MB)	4.31	13.68	13.58	13.51	13.39	13.25	13.13
Workshpace (MB)	1.74	4.86	4.83	4.80	4.77	4.72	4.68
PCG iters.	16	18	40	127	254	325	436
P-time	0.09	0.50	0.49	0.49	0.49	0.48	0.47
S-time	0.07	0.18	0.39	1.22	2.42	3.06	4.07
Total CPU	0.16	0.69	0.88	1.71	2.90	3.54	4.54

$\triangleright$ IC$_p$: $(N, L, t, \beta) = (32 \times 32, 80, 1, 10)$ $\triangleleft$

U	0	1	2	3	4	5	6
Storage R (MB)	45.85	54.26	54.13	53.66	53.12	52.63	52.26
Workspace (MB)	16.48	19.28	19.25	19.09	18.91	18.75	18.62
PCG iters.	22	18	36	143	748	1,153	1,527
P-time	1.72	2.00	1.98	1.96	1.93	1.92	1.91
S-time	0.81	0.74	1.47	5.77	29.78	45.58	59.98
Total CPU	2.53	2.75	3.45	7.73	31.71	47.50	61.88

RIC3. For the record, the following tables are the observed optimal performance of the RIC3 based PCG solver with the proper chosen dropping threshold values.

▷ RIC3: $(N, L, t, \beta) = (16 \times 16, 80, 1, 10)$ ◁

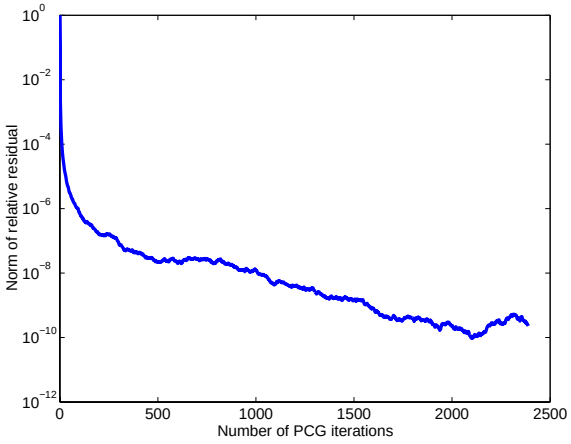
U	0	1	2	3	4	5	6
Storage R (MB)	4.85	5.17	5.22	5.24	5.22	5.21	5.22
Workspace MB)	42.15	46.23	43.32	41.07	39.09	37.41	35.93
PCG iters.	13	36	79	247	471	628	797
P-time	0.86	0.95	0.89	0.86	0.81	0.78	0.75
S-time	0.07	0.16	0.36	1.13	2.16	2.87	3.64
Total CPU	0.92	1.11	1.26	1.99	2.97	3.65	4.40

▷ RIC3: $(N, L, t, \beta) = (32 \times 32, 80, 1, 10)$ ◁

U	0	1	2	3	4	5	6
Storage R (MB)	19.56	20.75	20.98	20.96	20.91	20.87	20.86
Workspace (MB)	166.99	183.54	172.82	163.42	155.28	148.73	143.13
PCG iters.	13	32	77	282	1,355	2,217	2,686
P-time	3.57	3.92	3.72	3.52	3.35	3.21	3.10
S-time	0.24	0.61	1.47	5.24	25.63	41.90	50.61
Total CPU	3.81	4.53	5.19	8.87	28.98	45.11	53.71

2.8 Concluding remarks

It remains an open problem to search a linear-scaling preconditioner for strongly interaction systems. We have observed that in this situtation, the residual norm stagnates after initial rapid decline. The following plotthe relative residual norm of the PCG iteration to achieve the solution error $\|x_k - x\|_2/\|x\|_2 < 10^{-3}$, when $(N, L, U, t, \beta, \mu) = (32 \times 32, 80, 6, 1, 10, 0)$.



The plateau is largely due to the the slow decay of the components of the resiudal vector associated with the small eigenvalues of the preconditioned matrix $R^{-1}AR^{-T}$. Several techniques have been proposed to deflate these components from the residual vector as a way to avoid the plateau of the convergence, see [2, 6, 5, 15, 16] and references within. It remains to be studied about the applicability of these techniques to our HQMC applications.

Bibliography

[1] Ajiz, M., and Jennings, A. A robust incomplete choleski-conjugate gradient algorithm. *Internat. J. Numer. Methods Engrg. 20*, 5 (1984), 949–966.

[2] Arioli, M., and Ruiz, D. A Chevyshev-based two-stage iterative method as an alternative to the direct solution of linear systems. *Technical Report, RAL-TR-2002-021, Rutherford Appleton Laboratory.*

[3] Axelsson, O., and Kolotilina, L. Y. Diagonally compensated reduction and related preconditioning methods. *Numer. Linear Algebra Appl. 1* (1994), 155–177.

[4] Bai, Z., and Scalettar, R. T. Itr: Advances of core numerical linear algebra techniques for quantum simulation in solid state physics. NSF proposal.

[5] Bollhoefer, M., and Mehrmann, V. A New Approach to Algebraic Multilevel Methods Based on Sparse Approximate Inverses. Preprint, Numerische Simulation auf massiv parallelen Rechnern. (1999).

[6] Capentieri, B., Duff, I. S., and Giraud, L. A class of spectral two-level preconditioners. *SIAM J. Scient. Comp. 25* (2003), 749–765.

[7] Eijkhout, V. On the existence problem of incomplete factorization methods. Technical report LAPACK working note 144, UT-CS-99-435, Computer science department, University of Tennessee.

[8] Golub, G. H., and Loan, C. F. V. *Matrix Computations*, 3 ed. Johns Hopkins University Press, Baltimore and London, 1996.

[9] Jennings, A., and Malik, G. M. Partial elimination. *J. Inst. Math. Applics. 20*, (1977), 307–316.

[10] Kaporin, I. E. High quality preconditioning of a general symmetric positive definite matrix based on its $u^t u + u^t r + r^t u$-decomposition. *Numer. Linear Algebra Appl. 5* (1998), 483–509.

[11] Kershaw, D. S. The incomplete Cholesky-Conjugate Gradient method for the iterative solution of systems of linear equations. *J. of Comp. Phys. 26* (1978), 43–65.

[12] Lancaze, R., Morel, A., Petersson, B., and Schroper, J. An investigation of the 2D attractive Hubbard model. *Eur. Phys. J. B. 2* (1998), 509–523.

[13] Manteuffel, T. A. An incomplete factorization technique for positive definite linear systems. *Mathematics of Computation 34*, 150 (1980), 473–497.

[14] Meijerinkand, J., and van der Vorst, H. A. An iterative solution method for linear systems of which the coefficient matrix is a symmetric m-matrix. *Math. Comput 31*, 137 (1977), 134–155.

[15] Nicolaides, R. A. Deflation of conjugate gradients with application to boundary value problems. *SIAM J. Numer. Anal.*, 24:355–365, 1987.

[16] Padiy, A., Axelsson, O., and Polman, B. Generalized augmented matrix preconditioning approach and its application to iterative solution of ill-conditioned algebraic systems. *SIAM J. Matrix Anal. Appl.*, 22:793–818, 2000.

[17] Portable, Extensible toolkit for scientific computation (PETSc). http://www-unix.mcs.anl.gov/petsc/petsc-2/.

[18] Scalettar, R. T., Scalapino, D. J., and Sugar, R. L. A new algorithm for numerical simulation of fermions. *Phys. Rev. B, Condens Matter 34* (1986), 7911–7917.

[19] Scalettar, R. T., Scalapino, D. J., Sugar, R. L., and Tousaint, D. A hybrid-molecular dynamics algorithm for the numerical simulation of many electron systems. *Phys. Rev. B Condens Matter 36* (1987), 8632–8641.

[20] Schnabel, R. B. and Eskow, E. A new modified Cholesky factorization. *SIAM J. Sci. Compu. 11* (1990), 1136–1158.

[21] Tismenetsky, M. A new preconditioning technique for solving large sparse linear systems. *Linear Algebra Appl. 154–156* (1991), 331–353.

[22] van der Vorst, H. A. Iterative solution methods for certain sparse linear systems with a non-symmetric matrix arising from PDE-problems. *J. of Comp. Phys. 44* (1981), 1–19.

[23] Varga, R. S., Saff, E. B., and Mehrmann, V. Incomplete factorizations of matrices and connections with h-matrices. *SIAM J. Numer. Anal. 17* (1980), 787–793.