Journal of Scientific Computing (2020) 85:50
https://doi.org/10.1007/s10915-020-01343-5

Hierarchical Interpolative Factorization Preconditioner for Parabolic Equations

Jordi Feliu-Fabà[1] · **Lexing Ying**[2]

Received: 13 April 2020 / Revised: 18 September 2020 / Accepted: 8 October 2020 /
Published online: 10 November 2020

Abstract
This note proposes an efficient preconditioner for solving linear and semi-linear parabolic equations. With the Crank–Nicholson time stepping method, the algebraic system of equations at each time step is solved with the conjugate gradient method, preconditioned with hierarchical interpolative factorization. Stiffness matrices arising in the discretization of parabolic equations typically have large condition numbers, and therefore preconditioning becomes essential, especially for large time steps. We propose to use the hierarchical interpolative factorization as the preconditioning for the conjugate gradient iteration. Computed only once, the hierarchical interpolative factorization offers an efficient and accurate approximate inverse of the linear system. As a result, the preconditioned conjugate gradient iteration converges in a small number of iterations. Compared to other classical exact and approximate factorizations such as Cholesky or incomplete Cholesky, the hierarchical interpolative factorization can be computed in linear time and the application of its inverse has linear complexity. Numerical experiments demonstrate the performance of the method and the reduction of conjugate gradient iterations.

Keywords Hierarchical interpolative factorization · Parabolic equations · Heat equation · Reaction–diffusion equations

Electronic supplementary material The online version of this article (https://doi.org/10.1007/s10915-020-01343-5) contains supplementary material, which is available to authorized users.

✉ Jordi Feliu-Fabà
jfeliu@stanford.edu

Lexing Ying
lexing@stanford.edu

[1] Institute for Computational and Mathematical Engineering, Stanford University, Stanford, CA, USA

[2] Department of Mathematics and Institute for Computational and Mathematical Engineering, Stanford University, Stanford, CA, USA

1 Introduction

This note is concerned with the numerical solution of parabolic equations of the form

$$\frac{\partial u(x,t)}{\partial t} = \nabla \cdot \big(a(x)\nabla u(x)\big) + r\big(u(x,t)\big), \quad x \in \Omega \subset \mathbb{R}^d, \tag{1}$$

in two and three dimensions, with appropriate boundary conditions on $\partial\Omega$ and initial conditions $u(x,0) = u_0(x)$. Here, $a(x) > 0$ is the coefficient field of the diffusion operator and $r\big(u(x,t)\big)$ is the reaction term. We are interested on approximating the unknown field $u(x,t)$. Such reaction–diffusion equations can model a great variety of physical phenomena, such as heat conduction with internal heat generation, population dynamics [7,16] or pattern formation [22] in biology.

The most common spatial discretizations for solving (1) are finite difference and finite element methods. Such a spatial discretization results in a time-dependent system of form

$$\frac{\partial u(t)}{\partial t} = Mu(t) + r(t), \tag{2}$$

where $u(t) \in \mathbb{R}^N$ and $r(t) \in \mathbb{R}^N$ are the spatial discretizations of $u(x,t)$ and $r\big(u(x,t)\big)$ at time t, respectively, and N is the number of degrees of freedom (DOFs) in the spatial discretization. The *stiffness* matrix $M \in \mathbb{R}^{N\times N}$ is the discretization of the diffusion term.

The classical approach to solve (2) consists of discretizing in time and evolving the numerical solution at successive time steps using a time marching method. For instance, an explicit scheme to approximate the solution at every time step successively can be obtained by using the forward Euler method in time,

$$u^{k+1} = (I + M\Delta t)u^k + \Delta t \cdot r^k.$$

However, such an approach would require selecting a very small time step Δt in order to satisfy the stability condition $\Delta t \leq \frac{1}{2}\Delta x^2 \max_{x\in\Omega} a(x)$, where Δx is the spacing of the spatial grid.

Alternatively, one can overcome the stability condition and use larger time steps by using an implicit scheme such as the first-order backward Euler or the second-order Crank–Nicolson methods, which are unconditionally stable. However, such methods introduce another challenge by requiring to solve a system of equations at each time step. This note focuses on the Crank–Nicolson method

$$\left(I - \frac{\Delta t}{2}M\right)u^{k+1} = \left(I + \frac{\Delta t}{2}M\right)u^k + \Delta t \cdot r^k. \tag{3}$$

There are several ways to solve the system of equations. The most direct way is to use exact factorization methods, such as Cholesky decomposition, which would be prohibitively expensive for large problem sizes. Alternatively, one can use iterative methods such as conjugate gradient (CG) which takes $O(nnz(M)) = O(N)$ cost per iterations. However, since the matrix $I - \frac{\Delta t}{2}M$ is typically ill-conditioned, the number of iterations can be quite large, thus requiring the use of a preconditioner. Finding a good preconditioner is itself a challenging task that has been vastly studied in the literature. Among others, we highlight here the incomplete Cholesky factorization [18] and multifrontal method based algorithms coupled with hierarchical matrices [26] and skeletonization [2,5,11,15].

The main goal of this note is to describe a new efficient preconditioner based on a version of the *hierarchical interpolative factorization* (HIF) described in [5] to reduce the number of CG iterations at each time step. Due to the accuracy and efficiency of this version of HIF,

the preconditioned CG iteration at each time step converges in a small number of iterations. We demonstrate the effectiveness of this approach of solving (1) by studying several 2d and 3d numerical examples.

Other approaches. In recent years, several exponential integrator based models for time discretization have been proposed to solve parabolic equations, based on the integration factor method [14] and the exponential time differencing method [13,27].

Boundary integral formulations provide an alternative to the PDE-based approach. In the case where the diffusion coefficient $a(x)$ is constant, one can make use of parabolic potential theory [20,23] to obtain a boundary integral equation including time convolution of the form

$$u(x,t) = \int_{\Omega(0)} G(x-y,t)u_0(y)dy + \int_0^t \int_{\Omega(\tau)} G(x-y,t-\tau)r(y,\tau)dyd\tau, \quad (4)$$

where $G(x,t)$ is the free-space Green's function for the heat equation in d dimensions

$$G(x,t) = \frac{e^{-\|x\|^2/4t}}{(4\pi t)^{d/2}}. \quad (5)$$

One can then use fast algorithms to evaluate layer potentials, such as the hierarchical interpolative factorization [12], the fast multipole method [8] as done by Messner et al. [19] or the fast Gauss transform [9,21] used by Wang et al. [24] in this context. Other works include [1,25]. This approach has many advantages, such as time stability, reduction of DOFs to the boundary and the availability of fast linear algorithms for the evaluation of the layer potentials. However, so far, the classical potential theory is restricted to the set of reaction–diffusion equations with constant coefficients.

2 Algorithm

This section reviews the version of HIF introduced in [5] and then describes the new preconditioner. Throughout the note, the following notation are used: the uppercase letters (A, F, M, etc.) denote matrices; the calligraphic letters ($\mathcal{I}$, $\mathcal{B}$, $\mathcal{R}$, etc.) denote sets of indices, associated to DOFs; $A_{\mathcal{IB}}$ refers to the restriction of A to the $|\mathcal{I}| \times |\mathcal{J}|$ submatrix with rows indexed by $\mathcal{I}$ and columns indexed by $\mathcal{B}$; the notation $\{\mathcal{I}_i\}_{i=1}^p$ represents a collection of p disjoint sets of indices $\mathcal{I}_i$ for $i = 1, \ldots, p$.

Consider the PDE (1) on $\Omega = (0,1)^2$ with the Dirichlet boundary conditions and initial condition $u(x,0) = u_0(x)$. We perform finite difference dicretization via the standard five-point stencil over a uniform grid with step size h, resulting in $N = (n-1)^2$ DOFs, with $n = 1/h = 2^L m$ for some integers L and m. Each DOF corresponds to the solution $u_j = u(x_j)$ at grid points $x_j = (hj_1, hj_2)$, with $j = (j_1, j_2)$ and $1 < j_1, j_2 < n-1$. The resulting matrix M corresponding to the discretization of the diffusion term $\nabla a(x)\nabla$ in (1) is sparse and symmetric negative definite.

2.1 Crank–Nicolson Scheme

Since the diffusive term is the leading term, we use the Crank–Nicolson scheme for the diffusive term and an explicit scheme for the reaction term, with time step Δt. This leads to the algebraic system of Eq. (3), which can be solved successively to evolve the numerical solution at future time steps,

$$u^{k+1} = \left(I - \frac{\Delta t}{2} M\right)^{-1} \left[\left(I + \frac{\Delta t}{2} M\right) u^k + \Delta t \cdot r^k\right]. \tag{6}$$

Since the unconditionally stable Crank–Nicolson scheme is second order in time and space, it allows for larger time steps than Backward or Forward Euler methods. Since $A = I - \frac{\Delta t}{2} M$ is sparse and symmetric-positive-definite (SPD) one can use conjugate gradient to solve (6) as opposed to classical direct solvers which are more expensive. However, the number of iterations scale with $\kappa(A)$, the condition number of A. For small time steps, $\kappa(A)$ may be close to 1. However, for the large time steps adopted here, $\kappa(A)$ is comparable to the condition number of M, which is quite large as M is ill-conditioned. Preconditioning then becomes an essential step in order to reduce the number of CG iterations. The HIF has been shown to significantly reduce the number of CG iterations for the Poisson equation with variable coefficient [5] and we apply it to precondition $A = I - \frac{\Delta t}{2} M$.

2.2 Recursively Preconditioned Hierarchical Interpolative Factorization

Here we briefly review the *recursively preconditioned hierarchical interpolative factorization* (PHIF) proposed in [5], which is an improved version of the original HIF described in [11]. Given a sparse SPD matrix A, PHIF produces a fast factorization that can be seen as a multilevel generalized Cholesky decomposition, which alternates among block Gaussian elimination, block Jacobi preconditioning and skeletonization at different levels.

One starts by defining a uniform quad-tree with L levels, that partitions the domain Ω into $p^\ell = 2^{L-\ell} \times 2^{L-\ell}$ square cells at each level $\ell = 0, 1, \ldots, L$. The leaves correspond to level $\ell = 0$ and the root to level $\ell = L$. Throughout the factorization we denote *active* DOFs the DOFs that have not yet been decoupled. We set $A_0 = A$ and start factorizing the matrix by decoupling DOFs starting from the leaves level $\ell = 0$ up to level $L - 1$ with the following three steps at each level ℓ:

(1) *Cell elimination*. For each cell $1 < i < p_\ell$ at level ℓ we decouple the interior active DOFs indexed by $\mathcal{I}_{\ell,i}$ as follows. Up to a permutation matrix A_ℓ can be written as

$$A_\ell = \begin{bmatrix} A_{\mathcal{I}_{\ell,i}\mathcal{I}_{\ell,i}} & A^T_{\mathcal{B}\mathcal{I}_{\ell,i}} & \\ A_{\mathcal{B}\mathcal{I}_{\ell,i}} & A_{\mathcal{B}\mathcal{B}} & A^T_{\mathcal{R}\mathcal{B}} \\ & A_{\mathcal{R}\mathcal{B}} & A_{\mathcal{R}\mathcal{R}} \end{bmatrix}, \tag{7}$$

where $\mathcal{I}_{\ell,i}$ represents the DOFs inside that cell, $\mathcal{B}$ the DOFs on the boundary (i.e. edges and corners of the cell), and $\mathcal{R}$ the remaining DOFs. For clarity purposes, hereafter we use sub-index ℓ to denote matrices at level ℓ and drop the sub-index ℓ when referring to a particular block of the matrix, for instance we use $A_{\mathcal{B}\mathcal{B}}$ to refer to $A_{\ell_{\mathcal{B}\mathcal{B}}}$. Let $A_{\mathcal{I}_{\ell,i},\mathcal{I}_{\ell,i}} = L_{\mathcal{I}_{\ell,i}} L^T_{\mathcal{I}_{\ell,i}}$, Gaussian elimination leads to

$$M^T_{\mathcal{I}_{\ell,i}} A_\ell M_{\mathcal{I}_{\ell,i}} = \begin{bmatrix} I & & \\ & X_{\mathcal{B}\mathcal{B}} & A^T_{\mathcal{R}\mathcal{B}} \\ & A_{\mathcal{R}\mathcal{B}} & A_{\mathcal{R}\mathcal{R}} \end{bmatrix}, \quad M_{\mathcal{I}_{\ell,i}} = \begin{bmatrix} L^{-T}_{\mathcal{I}_{\ell,i}} & -A^{-1}_{\mathcal{I}_{\ell,i}\mathcal{I}_{\ell,i}} A^T_{\mathcal{B}\mathcal{I}_{\ell,i}} & \\ & I & \\ & & I \end{bmatrix}, \tag{8}$$

where $X_{\mathcal{B}\mathcal{B}} = A_{\mathcal{B}\mathcal{B}} - A_{\mathcal{B}\mathcal{I}_{\ell,i}} A^{-1}_{\mathcal{I}_{\ell,i}\mathcal{I}_{\ell,i}} A^T_{\mathcal{B}\mathcal{I}_{\ell,i}}$. Performing block eliminations for each cell $1 < i < p_\ell$ at level ℓ of the quadtree leads to

$$\bar{A}_\ell = M^T_\ell A_\ell M_\ell, \quad M_\ell = \prod_{i=1}^{p_\ell} M_{\mathcal{I}_{\ell,i}}. \tag{9}$$

After block elimination with respect the collection of index sets $\{\mathcal{I}_{\ell,i}\}_{i=1}^{p_\ell}$, the DOFs inside the cells have been decoupled from those in the edges and corners at level ℓ. Therefore, the remaining active DOFs are those in the edges and corners and we only need to continue factorizing the matrix $\bar{A}_\ell$ restricted on these active DOFs.

(2) *Block Jacobi preconditioning*. Let r_ℓ be the number of edges and corners at level ℓ, and $\{\mathcal{I}_{\ell,i}\}_{i=1}^{r_\ell}$ be the collection of corresponding index sets for the active DOFs. For a given edge or corner i, up to a permutation, $\bar{A}_\ell$ can be written as

$$\bar{A}_\ell = \begin{bmatrix} \bar{A}_{\mathcal{I}_{\ell,i}\mathcal{I}_{\ell,i}} & \bar{A}^T_{\mathcal{B}\mathcal{I}_{\ell,i}} & \\ \bar{A}_{\mathcal{B}\mathcal{I}_{\ell,i}} & \bar{A}_{\mathcal{B}\mathcal{B}} & \bar{A}^T_{\mathcal{R}\mathcal{B}} \\ & \bar{A}_{\mathcal{R}\mathcal{B}} & \bar{A}_{\mathcal{R}\mathcal{R}} \end{bmatrix}, \tag{10}$$

where $\mathcal{I}_{\ell,i}$ represents the DOFs in the edge/corner i, $\mathcal{B}$ the DOFs on the edges and corners connected to edge/corner i in $\bar{A}_\ell$, and $\mathcal{R}$ the remaining active DOFs. Then a *rescaling* of edge/corner i can be performed using the Cholesky decomposition $\bar{A}_{\mathcal{I}_{\ell,i}\mathcal{I}_{\ell,i}} = L_{\mathcal{I}_{\ell,i}} L^T_{\mathcal{I}_{\ell,i}}$ as

$$C^T_{\mathcal{I}_{\ell,i}} \bar{A}_\ell C_{\mathcal{I}_{\ell,i}} = \begin{bmatrix} I & L^{-1}_{\mathcal{I}_{\ell,i}} \bar{A}^T_{\mathcal{B}\mathcal{I}_{\ell,i}} & \\ \bar{A}_{\mathcal{B}\mathcal{I}_{\ell,i}} L^{-T}_{\mathcal{I}_{\ell,i}} & \bar{A}_{\mathcal{B}\mathcal{B}} & \bar{A}^T_{\mathcal{R}\mathcal{B}} \\ & \bar{A}_{\mathcal{R}\mathcal{B}} & \bar{A}_{\mathcal{R}\mathcal{R}} \end{bmatrix}, \quad C_{\mathcal{I}_{\ell,i}} = \begin{bmatrix} L^{-T}_{\mathcal{I}_{\ell,i}} & & \\ & I & \\ & & I \end{bmatrix} \in \mathbb{R}^{N\times N}, \tag{11}$$

If we perform this preconditioning for each edge and corner in level ℓ, we obtain a block Jacobi preconditioning that yields

$$\widetilde{A}_\ell = C^T_\ell \bar{A}_\ell C_\ell, \quad C_\ell = \prod_{i=1}^{r_\ell} C_{\mathcal{I}_{\ell,i}}, \tag{12}$$

where C_ℓ is a block diagonal matrix, up to a permutation, since $\{\mathcal{I}_{\ell,i}\}_{i=1}^{r_\ell}$ is a collection of disjoint index sets. The diagonal blocks of the resulting matrix $\widetilde{A}_\ell$ are identity matrices and the set of active DOFs remains unchanged.

(3) *Edge skeletonization*. Let q_ℓ be the number of edges at level ℓ, and $\{\mathcal{I}_{\ell,i}\}_{i=1}^{q_\ell}$ the collection of corresponding index sets. For a given edge i with DOFs indexed by $\mathcal{I}_{\ell,i}$ skeletonization is performed as follows. For clarity we drop the subindex in $\mathcal{I}_{\ell,i}$ in the remainder of the explanation for the edge skeletonization step. Up to a permutation one can write

$$\widetilde{A}_\ell = \begin{bmatrix} I & \widetilde{A}^T_{\mathcal{R}\mathcal{I}} \\ \widetilde{A}_{\mathcal{R}\mathcal{I}} & \widetilde{A}_{\mathcal{R}\mathcal{R}} \end{bmatrix} \tag{13}$$

where I are the DOFs in edge i at level ℓ and R are the rest of active DOFs. Assume $\widetilde{A}_{\mathcal{R}\mathcal{I}} \in \mathbb{R}^{N_\mathcal{R}\times N_\mathcal{I}}$ has numerical rank k to relative precision ϵ. Then, we can use interpolative decomposition (ID) [3], to get a partition of $\mathcal{I} = \widetilde{\mathcal{I}} \cup \hat{\mathcal{I}}$ into *skeleton* $\hat{\mathcal{I}}$ and *redundant* $\widetilde{\mathcal{I}}$ DOFs, and approximate the redundant columns of $\widetilde{A}_{\mathcal{R}\mathcal{I}}$ by a linear combination of its skeleton columns such that

$$\widetilde{A}_{\mathcal{R}\widetilde{\mathcal{I}}} = \widetilde{A}_{\mathcal{R}\hat{\mathcal{I}}} T_\mathcal{I} + E_\mathcal{I}, \quad \|E_\mathcal{I}\| = O(\epsilon \|\widetilde{A}_{\mathcal{R}\mathcal{I}}\|), \tag{14}$$

with $T_{\mathcal{I}} \in \mathbb{R}^{k\times(N_{\mathcal{I}}-k)}$. Up to a permutation, using ID we can approximately zero out the redundant columns leading to

$$Z_{\mathcal{I}}^T \widetilde{A}_\ell Z_{\mathcal{I}} \approx \left[\begin{array}{cc|c} I + T_{\mathcal{I}}^T T_{\mathcal{I}} & -T_{\mathcal{I}}^T & \\ -T_{\mathcal{I}} & I & \widetilde{A}_{\mathcal{R}\hat{\mathcal{I}}}^T \\ \hline & \widetilde{A}_{\mathcal{R}\hat{\mathcal{I}}} & \widetilde{A}_{\mathcal{R}\mathcal{R}} \end{array}\right], \quad Z_{\mathcal{I}} = \left[\begin{array}{cc|c} I & & \\ -T_{\mathcal{I}} & I & \\ \hline & & I \end{array}\right] \in \mathbb{R}^{N\times N}, \quad (15)$$

Now, we can decouple redundant DOFs $\widetilde{\mathcal{I}}$ in the edge by performing Gaussian elimination using the Cholesky decomposition $I + T_{\mathcal{I}}^T T_{\mathcal{I}} = L_{\hat{\mathcal{I}}} L_{\hat{\mathcal{I}}}^T$,

$$M_{\widetilde{\mathcal{I}}}^T Z_{\mathcal{I}}^T \widetilde{A}_\ell Z_{\mathcal{I}} M_{\widetilde{\mathcal{I}}} \approx \left[\begin{array}{cc|c} I & & \\ & X_{\hat{\mathcal{I}}\hat{\mathcal{I}}} & \widetilde{A}_{\mathcal{R}\hat{\mathcal{I}}}^T \\ \hline & \widetilde{A}_{\mathcal{R}\hat{\mathcal{I}}} & \widetilde{A}_{\mathcal{R}\mathcal{R}} \end{array}\right], \quad M_{\mathcal{I}} = \left[\begin{array}{cc|c} L_{\hat{\mathcal{I}}}^{-T} & (I + T_{\mathcal{I}}^T T_{\mathcal{I}})^{-1} T_{\mathcal{I}}^T & \\ & I & \\ \hline & & I \end{array}\right] \quad (16)$$

with $X_{\hat{\mathcal{I}}\hat{\mathcal{I}}} = I - T_{\mathcal{I}}(I + T_{\mathcal{I}}^T T_{\mathcal{I}})^{-1} T_{\mathcal{I}}^T$. Performing skeletonization for each edge i with set of indices $\mathcal{I}_{\ell,i}$ gives

$$A_{\ell+1} \approx K_\ell^T \widetilde{A}_\ell K_\ell, \quad K_\ell = \prod_{i=1}^{q_\ell} K_{\mathcal{I}_{\ell,i}}, \quad K_{\mathcal{I}_{\ell,i}} = Z_{\mathcal{I}_{\ell,i}} M_{\mathcal{I}_{\ell,i}}. \quad (17)$$

The remaining active DOFs are now the skeleton DOFs in the edges and the corners at level ℓ. We can now move to the following level on the tree and perform the same three steps.

After performing these three steps for all levels $\ell = 0, \ldots, L-1$, i.e. once we are in the root level of the tree, the resulting matrix is

$$A_L \approx R_{L-1}^T \cdots R_0^T A R_0 \cdots R_{L-1}, \quad R_\ell = M_\ell C_\ell K_\ell \quad (18)$$

which is everywhere the identity except in the block indexed by the remaining active DOFs at the root level. As opposed to the nested dissection multifrontal factorization, the frontal matrix at the root is small since sparsification on the separator fronts has been performed throughout all the levels. Now, we can approximate the original matrix as

$$A \approx F = R_0^{-T} \cdots R_{L-1}^{-T} A_L R_{L-1}^{-1} \cdots R_0^{-1}, \quad (19)$$

and its inverse as

$$A^{-1} \approx F^{-1} = R_0 \cdots R_{L-1} A_L^{-1} R_{L-1}^T \cdots R_0^T. \quad (20)$$

The factors $R_\ell = M_\ell C_\ell K_\ell$ are easily invertible since M_ℓ, C_ℓ, K_ℓ are block diagonal up to a permutation, with each block being triangular. Therefore, the factorization can be viewed as a generalized Cholesky decomposition.

In the 3d case, the algorithm for PHIF is similar to the 2d case, except for a few details. Instead of square cells, the domain is partitioned into cube cells. At each level ℓ, Block Jacobi preconditioning is performed in faces, edges and corners, while skeletonization is performed on faces (although it can also be performed in edges to lower the computational complexity of the factorization).

See Figs. 1 and 2 for an illustration of the active DOFs at different steps and levels of the factorization process in 2d and 3d respectively. The Block Jacobi preconditioning step has been omitted in the illustrations because it doesn't change the active DOFs.

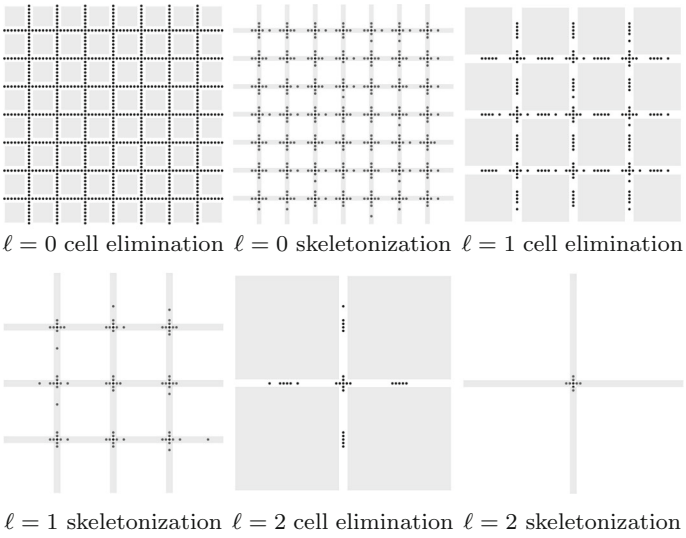


Fig. 1 Active DOFs at each level ℓ of PHIF in 2d, depicted with black dots. The square cells and edges at each level, from which DOFs have been eliminated, are represented in gray for cell elimination and skeletonization steps respectively

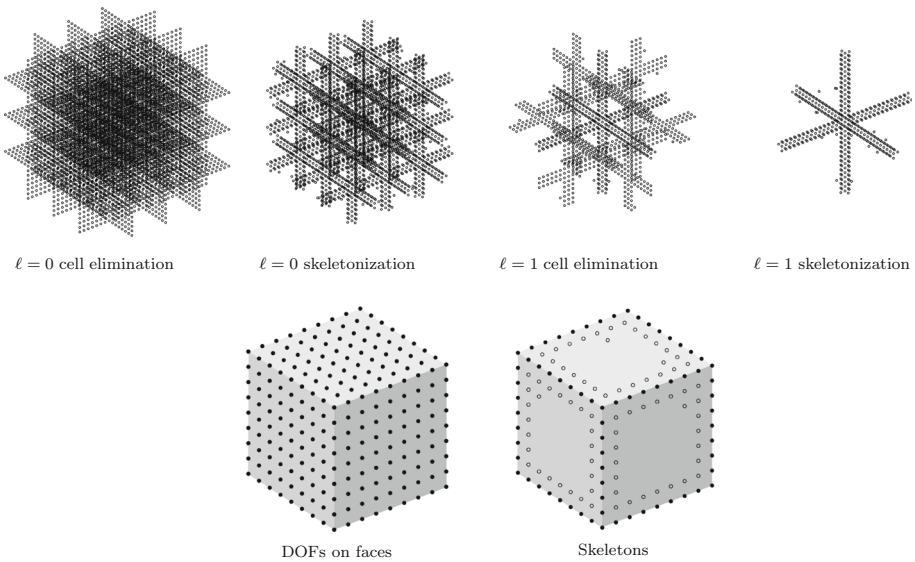


Fig. 2 Top: Active DOFs at each level ℓ of PHIF in 3d. Bottom: Detail of DOFs on the faces of a cubic cell before and after skeletonization, with skeleton DOFs represented by hollow circles

2.3 Crank–Nicolson Scheme with PHIF Preconditioned CG

We now proceed to describe the algorithm for evolving the numerical solution of (1). First, we compute the PHIF factorization $F \approx A = (I - \frac{\Delta t}{2} M)$. Since A is a sparse SPD matrix, the

PHIF can be computed in linear time to obtain $F^{-1} \approx A^{-1}$, and matrix–vector multiplications with F^{-1} require only $O(N)$ work.

In each time step of the Crank–Nicolson method, the linear system of Eq. (3) is solved by CG, with F^{-1} in its factorized form (20) used as the preconditioner. The algorithm is presented in Algorithm 1, where $\text{pcg}(A, b, P)$ represents the preconditioned conjugate gradient with preconditioner P that attempts to solve for $Ax = b$.

Algorithm 1: Numerical solution of reaction–diffusion equations

Construct PHIF F for $A = (I - \frac{\Delta t}{2} M)$ with some tolerance ϵ;
Initialize $u^0 = u_0$;
for $k = 1 : n_{steps}$ **do**
  $g = (I + \frac{\Delta t}{2} M)u^{k-1} + \Delta t \cdot r^{k-1}$;
  $u^k = \text{pcg}(A, g, F^{-1})$ (see (20) for the factorization of F^{-1})
end

3 Numerical Results

In this section, we demonstrate the performance of PHIF preconditioning for parabolic equations by solving two examples of Eq. (1): the heat equation and a logistic reaction–diffusion equation. The PHIF preconditioning is compared with incomplete Cholesky preconditioning in terms of the following quantities

- mem: the memory usage for PHIF and incomplete Cholesky factorizations;
- t_f: the factorization time;
- t_s: the average solve time for one time step, obtained from averaging over 100 time steps;
- n_i: the number of CG iterations averaged over 100 time steps, with the relative residual equal to 10^{-12}.

The only user-defined parameter in the PHIF factorization is the relative precision ϵ of the interpolative decomposition, which is set to 10^{-3} and 10^{-6} in the numerical experiments. Similarly, a drop tolerance ϵ is used for the incomplete Cholesky factorization.

The MATLAB code for PHIF used for the numerical experiments is a modified version of FLAM [10] to account for the block Jacobi preconditioning.

Example 1 *The heat equation.* Consider first the 2d heat equation

$$\frac{\partial u(x,t)}{\partial t} = \nabla \cdot (a(x)\nabla u(x,t)), \quad x \in \Omega = (0,1)^2 \tag{21}$$

with the zero Dirichlet boundary condition and the initial condition equal to the sum of two Gaussians,

$$u_0(x, y) = e^{-((x-c_1)^2+(y-c_1)^2)/\sigma^2} + e^{-((x-c_2)^2+(y-c_2)^2)/\sigma^2}, \tag{22}$$

where $c_1 = 0.35$, $c_2 = 0.65$ and $\sigma^2 = 0.05$. The diffusion coefficient field is set to

$$a(x, y) \sim \sum_{i=1}^{m} e^{-((x-x_i)^2+(y-y_i)^2)/\sigma_2^2}, \tag{23}$$

Table 1 Numerical results for the heat equation in 2d

	PHIF					Incomplete Cholesky				
N	ϵ	Mem (GB)	t_f (s)	t_s (s)	n_i	ϵ	Mem (GB)	t_f (s)	t_s (s)	n_i
511^2	10^{-3}	0.19	1.4e1	1.7	4.6	10^{-3}	0.085	1.7e−1	1.6	58.5
1023^2	10^{-3}	0.80	6.2e1	7.9	5.2	10^{-3}	0.343	4.7e−1	1.2e1	97
2047^2	10^{-3}	3.24	2.6e2	3.6e1	5.7	10^{-3}	1.37	2.1	7.4e1	152.7
4095^2	10^{-3}	13.0	9.1e2	1.7e2	5.8	10^{-3}	5.50	7.7	4.6e2	226.8
511^2	10^{-6}	0.205	1.6e1	8.6e−1	2.3	10^{-5}	0.335	1.4	1.1	11
1023^2	10^{-6}	0.834	6.6e1	4.3	2.7	10^{-5}	1.35	6.1	7.7	16.4
2047^2	10^{-6}	3.36	2.9e2	2.1e1	3	10^{-5}	5.4	2.6e1	4.3e1	24.5
4095^2	10^{-6}	13.5	9.8e2	7.9e1	3	10^{-5}	21.6	1.1e2	2.7e2	38.2

rescaled and shifted to have values in the interval $[0.1, 10]$, with $\sigma_2^2 = 0.005$, $m = 100$ and x_i and y_i being i.i.d. random variables sampled from the uniform distribution $\mathcal{U}(0, 1)$.

The time step size is set to $\Delta t = \Delta x = \frac{1}{N}$ and the numerical solution is obtained for 100 successive time steps. Note that we want to set Δt on the order of Δx to get a second order approximation. Since Crank–Nicolson is unconditionally stable we can select a large time step, and since the initial condition is smooth, we don't observe numerical spurious oscillations.

The numerical solution is evolved by solving (6) at each time step using CG. The matrix $A = I - \frac{\Delta t}{2} M$ is especially ill-conditioned for large time steps, therefore preconditioning becomes necessary to reduce the number of CG iterations. Numerical results in Table 1 show a decrease on the number of CG iterations when using PHIF preconditioning instead of incomplete Cholesky factorization, with the similar memory footprint. The computation time per time step is approximately halved with the use of PHIF compared to incomplete Cholesky. Additionally, PHIF exhibits constant and problem size independent number of CG iterations, while the number of CG iterations for incomplete Cholesky scale with $O(N^{1/4})$. This results in almost linear $O(N)$ scaling of t_s with PHIF preconditioning and $O(N^{1.25})$ scaling with incomplete Cholesky preconditioning. PHIF also provides a good approximation to the inverse for $\epsilon = 10^{-6}$, thus one could use the factorization to directly solve the system of equations and bypass CG. For instance with $\epsilon = 10^{-6}$ and $N = 4095^2$, the PHIF factorization gives a solve error estimated as $\|I - AF^{-1}\| = 7.9 \times 10^{-6}$ with randomized power iteration [4,17] to 10^{-2} relative precision.

Let us consider an analogous problem in three dimensions with $\Omega = (0, 1)^3$ and the initial condition equal to a Gaussian function,

$$u_0(x, y, z) = e^{-((x-c)^2+(y-c)^2+(z-c)^2)/\sigma^2}, \tag{24}$$

where $c = 0.5$ and $\sigma^2 = 0.05$. The coefficient field is generated by

$$a(x, y, z) \sim \sum_{i=1}^{m} e^{-((x-x_i)^2+(y-y_i)^2+(z-z_i)^2)/\sigma_2^2} \tag{25}$$

Table 2 Numerical results for the heat equation in 3d

	PHIF					Incomplete Cholesky				
N	ϵ	Mem (GB)	t_f (s)	t_s (s)	n_i	ϵ	Mem (GB)	t_f (s)	t_s (s)	n_i
63^3	10^{-3}	1.33	2.5e2	3.8	5.2	10^{-4}	0.444	6.0	3.4	14.5
127^3	10^{-3}	14.1	3.2e3	3.7e1	5.9	10^{-4}	3.66	1.9e1	2.6e1	22.9
255^3	10^{-3}	137	1.3e4	3.7e2	6	10^{-4}	29.7	1.9e2	3.8e2	35
63^3	10^{-6}	1.78	8.7e1	1.9	3	10^{-6}	3.54	5.2e1	6.0	5.8
127^3	10^{-6}	21.2	1.5e3	2.0e1	3	10^{-6}	29.2	8.2e2	8.1e1	8
255^3	10^{-6}	223	2.2e4	1.8e2	3	10^{-6}	473	1.1e4	1.4e3	11

rescaled and shifted to be within the interval $[0.05, 20]$, with $\sigma_2^2 = 0.005$, $m = 1000$ and x_i, y_i and z_i being i.i.d. random variables sampled from the uniform distribution $\mathcal{U}(0, 1)$. The time step is set to $\Delta t = 0.1\Delta x$.

Numerical results are shown in Table 2. Similarly to the 2d example, PHIF leads to a reduction of CG iterations when compared to the threshold-based incomplete Cholesky. For instance for $N = 255^3$ with a tolerance $\epsilon = 10^{-6}$, CG with PHIF takes an average of 3 iterations, while CG with incomplete Cholesky takes 11. While the factorization time of PHIF is high for this 3d example, the solve time per time step can be approximately halved with the use of PHIF as opposed to incomplete Cholesky. For instance, the solve time per time step for PHIF is 1.8×10^2 with $\epsilon = 10^{-6}$, while for incomplete Cholesky it is 3.8×10^2 for with $\epsilon = 10^{-4}$. Additionally, experimentally PHIF exhibits constant number of CG iterations, while the number of CG iterations for incomplete Cholesky increases with the problem size N. This results in better scaling of t_s with PHIF preconditioning than with incomplete Cholesky preconditioning, making PHIF better suited for large problem sizes.

Example 2 *A logistic reaction–diffusion equation.* Consider now a 2d reaction–diffusion equation with logistic growth

$$\frac{\partial u(x,t)}{\partial t} = \nabla \cdot (a(x)\nabla u(x,t)) + k_1 u(x,t)\left(1 - \frac{u(x,t)}{k_2}\right), \quad x \in \Omega = (0,1)^2, \tag{26}$$

with $k_1 = 1$ and $k_2 = 10$, the zero Dirichlet boundary conditions and initial condition

$$u_0(x, y) = \frac{2}{3\sqrt{2\pi\sigma^2}} e^{-((x-c)^2+(y-c)^2)/\sigma^2} \tag{27}$$

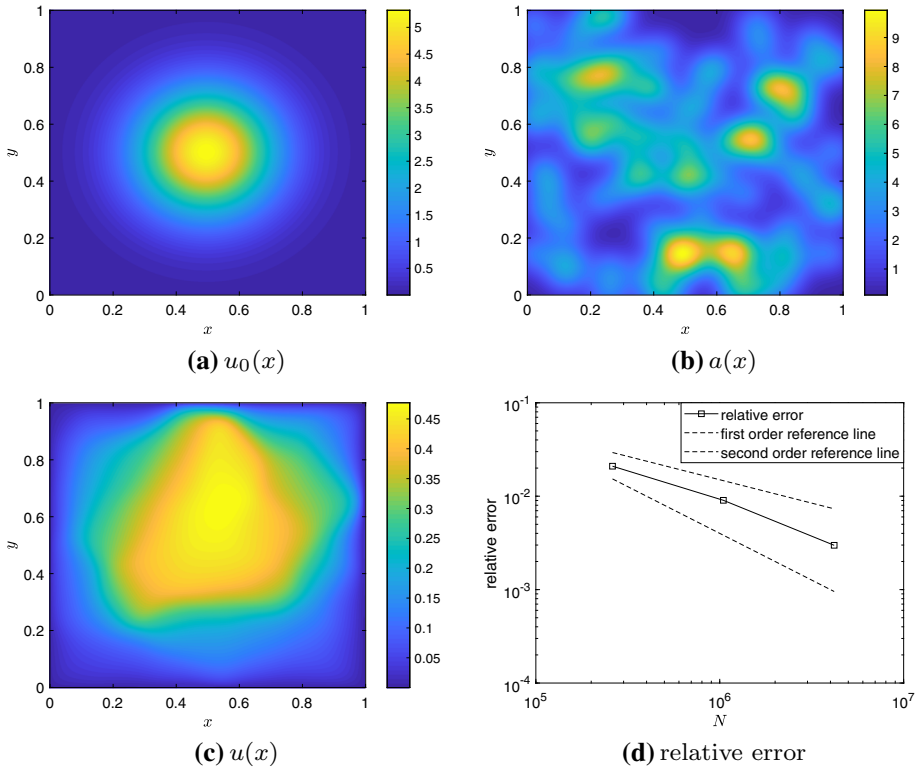
with $c = 0.5$, $\sigma^2 = 0.05$. The coefficient field is set analogously to Example 1. This problem is run with the same time step and the same number of time steps as in Example 1 and we observe no numerical spurious oscillations in the numerical solution.

The results are summarized in Table 3. Similarly to Example 1, we observe a decrease on the number of CG iterations n_i and on the computation time per time step t_s when preconditioning with PHIF. The initial condition and diffusion coefficients are illustrated in Fig. 3 together with the numerical solution after 128 time steps for $N = 4095^2$ and the relative error of the numerical solution $u^{(k=128)}$ for different problem sizes, which is close to second order asymptotically.

For the three dimensional case with $\Omega = (0, 1)^3$, we generate the initial condition $u_0(x)$ and the coefficient field in the same way as the 3d case from Example 1, with a multiplicative

Table 3 Numerical results for the 2d logistic reaction–diffusion equation

	PHIF					Incomplete Cholesky				
N	ϵ	Mem (GB)	t_f (s)	t_s (s)	n_i	ϵ	Mem (GB)	t_f (s)	t_s (s)	n_i
511^2	10^{-3}	0.19	1.4e2	2.3	5.1	10^{-3}	0.085	1.5e−1	1.8	64.6
1023^2	10^{-3}	0.81	5.7e1	8.7	5.6	10^{-3}	0.343	5.7e−1	1.3e1	107.1
2047^2	10^{-3}	3.24	2.3e2	5.0e1	6.5	10^{-3}	1.37	2.4	8.7e1	167.1
4095^2	10^{-3}	13.0	8.5e2	3.3e2	6.3	10^{-3}	5.50	7.8	5.1e2	252.9
511^2	10^{-6}	0.205	1.5	9.2e−1	2.6	10^{-5}	0.335	1.4	1.3	12.2
1023^2	10^{-6}	0.834	6.0e1	4.3	3	10^{-5}	1.347	6.2	8.6	18.5
2047^2	10^{-6}	3.36	2.9e2	2.8e1	3.4	10^{-5}	5.39	2.9e1	5.7e1	29.5
4095^2	10^{-6}	13.5	9.8e2	1.9e2	4	10^{-5}	21.5	1.2e2	3.2e2	40.8

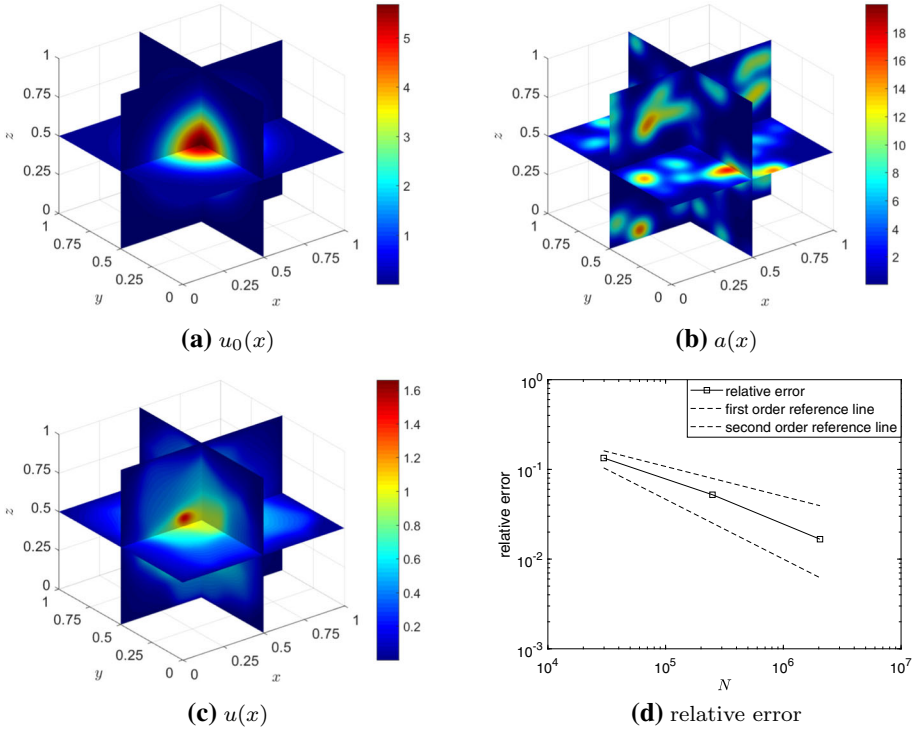


(a) $u_0(x)$ **(b)** $a(x)$ **(c)** $u(x)$ **(d)** relative error

Fig. 3 Initial conditions (**a**), coefficient distribution (**b**), numerical solution (**c**) and relative error plot (**d**) for the 2d logistic reaction–diffusion equation

Table 4 Numerical results for 3d logistic reaction–diffusion equation

N	PHIF					Incomplete Cholesky				
	ϵ	Mem (GB)	t_f (s)	t_s (s)	n_i	ϵ	Mem (GB)	t_f (s)	t_s (s)	n_i
63^3	10^{-3}	1.31	1.7e2	4.1	5	10^{-4}	0.443	2.4	1.6	14
127^3	10^{-3}	13.9	9.6e2	4.0e1	6	10^{-4}	3.66	1.6e1	2.4e1	21
255^3	10^{-3}	135	1.2e4	3.7e2	6	10^{-4}	29.7	1.5e2	3.6e2	32
63^3	10^{-6}	1.76	1.8e2	2.1	3	10^{-6}	2.77	3.0e1	4.2	5.7
127^3	10^{-6}	21.2	1.5e3	2.3e1	3	10^{-6}	29.1	5.2e2	6.2e1	7
255^3	10^{-6}	223	2.3e4	1.9e2	3	10^{-6}	237	7.6e3	8.6e2	10



(a) $u_0(x)$ **(b)** $a(x)$ **(c)** $u(x)$ **(d)** relative error

Fig. 4 Initial conditions (**a**), coefficient distribution (**b**), numerical solution (**c**) and relative error plot (**d**) for the 3d logistic reaction–diffusion equation

factor of $(2\pi\sigma)^{-3/2}$ in (24) and $m = 200$. We also set the time step to $\Delta t = 0.1\Delta x$ and the solution is evolved for 100 time steps.

Numerical results are shown in Table 4. The initial conditions, coefficient field, solution after 64 time steps for $N = 255^3$ and relative error of the numerical solution $u^{(k=64)}$ for increasing problem sizes are depicted in Fig. 4. We observe that the error is close to second order asymptotically and CG converges with very few iterations using PHIF, independently of N.

4 Conclusions

This note proposed an efficient preconditioner for solving linear and semi-linear parabolic equations based on the hierarchical interpolative factorization in [5]. The preconditioned CG iteration enjoys several advantages: (1) it provides a good approximate inverse that can be applied very rapidly, (2) one only needs to construct HIF factorization once at the beginning, and (3) applying the inverse approximation at each time step has linear cost. Computing the factorization can be done in linear time as opposed to other more expensive factorizations such as Cholesky or incomplete Cholesky. Well-suited for ill-conditioned matrices associated with large time steps, the new preconditioner reduce the number of CG iterations significantly.

This approach can also be extended to solve other parabolic equations, for instance the time-dependent fourth-order differential equations for studying the buckling plate or the clamping plate problems in the plate theory [6]. In such cases, HIF needs to use separators twice as wide, when using the 9-point and 13-point finite differences stencils in 2d and 3d, respectively.

If the PDE has time-dependent coefficients or moving geometries, instead of constructing the PHIF factorization once at the beginning, one would need to compute the factorization at each time step. Such a computation can be expensive, especially for 3d geometries. However, if the coefficients change slowly with time, one can reuse the result by computing the factorization once every few time steps.

Acknowledgements The work of J.F. is partially supported by Stanford Graduate Fellowship. The work of L.Y. is partially supported by U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, Scientific Discovery through Advanced Computing (SciDAC) program and the National Science Foundation under award DMS-1818449.

Data availability The datasets generated during and/or analysed during the current study are available in the github repository https://github.com/jordifeliu/PHIF_parabolic and from the corresponding author on reasonable request.

Code availability The main code used is based on FLAM [10], with appropriate modifications to add the Block Jacobi preconditioning step in PHIF.

Compliance with ethical standards

Conflict of interest All authors declare that they have no conflict of interest.

References

1. Barnett, A., Epstein, C., Greengard, L., Jiang, S., Wang, J.: Explicit unconditionally stable methods for the heat equation via potential theory. Pure Appl. Anal. **1**(4), 709–742 (2019)
2. Cambier, L., Boman, E.G., Chen, C., Rajamanickam, S., Tuminaro, R.S., Darve, E.: An algebraic sparsified nested dissection algorithm using low-rank approximations. arXiv:1901.02971 (2019)
3. Cheng, H., Gimbutas, Z., Martinsson, P.G., Rokhlin, V.: On the compression of low rank matrices. SIAM J. Sci. Comput. **26**(4), 1389–1404 (2005)
4. Dixon, J.D.: Estimating extremal eigenvalues and condition numbers of matrices. SIAM J. Numer. Anal. **20**(4), 812–814 (1983)
5. Feliu-Fabà, J., Ho, K.L., Ying, L.: Recursively preconditioned hierarchical interpolative factorization for elliptic partial differential equations. Commun. Math. Sci. **18**(1), 91–108 (2020)
6. Fishelov, D.: Semi-discrete time-dependent fourth-order problems on an interval: error estimate. In: Numerical Mathematics and Advanced Applications—ENUMATH 2013. Springer, Cham, pp. 133–142 (2015)
7. Fisher, R.A.: The wave of advance of advantageous genes. Ann. Eugen. **7**(4), 355–369 (1937)

8. Greengard, L., Rokhlin, V.: A fast algorithm for particle simulations. J. Comput. Phys. **73**(2), 325–348 (1987)
9. Greengard, L., Strain, J.: The fast gauss transform. SIAM J. Sci. Comput. **12**, 79–94 (1991)
10. Ho, K.L.: FLAM: Fast linear algebra in MATLAB. http://doi.org/10.5281/zenodo.1253581 (2018)
11. Ho, K.L., Ying, L.: Hierarchical interpolative factorization for elliptic operators: differential equations. Comm. Pure Appl. Math. **69**(8), 1415–1451 (2016)
12. Ho, K.L., Ying, L.: Hierarchical interpolative factorization for elliptic operators: integral equations. Comm. Pure Appl. Math. **69**(7), 1314–1353 (2016)
13. Huang, J., Ju, L., Wu, B.: A fast compact exponential time differencing method for semilinear parabolic equations with Neumann boundary conditions. Appl. Math. Lett. **94**, 257–265 (2019)
14. Ju, L., Zhang, J., Zhu, L., Du, Q.: Fast explicit integration factor methods for semilinear parabolic equations. J. Sci. Comput. **62**, 431–455 (2015)
15. Klockiewicz, B., Darve, E.: Sparse hierarchical preconditioners using piecewise smooth approximations of eigenvectors. arXiv:1907.03406 (2019)
16. Kolmogorov, A., Petrovskii, I., Piskunov, N.: A study of the diffusion equation with increase in the amount of substance, and its application to a biological problem. Bull. Moscow Univ. Math. Mech. **17**(6), 1–26 (1937)
17. Kuczyński, J., Woźniakowski, H.: Estimating the largest eigenvalue by the power and Lanczos algorithms with a random start. SIAM J. Matrix Anal. Appl. **13**(4), 1094–1122 (1992)
18. Manteuffel, T.A.: An incomplete factorization technique for positive definite linear systems. Math. Comput. **34**, 473–497 (1980)
19. Messner, M., Schanz, M., Tausch, J.: A fast galerkin method for parabolic space-time boundary integral equations. J. Comput. Phys. **258**, 15–30 (2014)
20. Tausch, J.: A fast method for solving the heat equation by layer potentials. J. Comput. Phys. **224**(2), 956–969 (2007)
21. Tausch, J., Weckiewicz, A.: Multidimensional fast gauss transforms by Chebyshev expansions. SIAM J. Sci. Comput. **31**, 3547–3565 (2009)
22. Turing, A.M.: The chemical basis of morphogenesis. Philos. Trans. R. Soc. Lond. **237**(641), 37–72 (1952)
23. Wang, J., Greengard, L.: An adaptive fast Gauss transform in two dimensions. SIAM J. Sci. Comput. **40**, A1274–A1300 (2018)
24. Wang, J., Greengard, L., Jiang, S., Veerapaneni, S.: Fast integral equation methods for linear and semilinear heat equations in moving domains. arXiv:1910.00755 (2019)
25. Wang, S., Jiang, S., Wang, J.: Fast high-order integral equation methods for solving boundary value problems of two dimensional heat equation in complex geometry. J. Sci. Comput. **79**, 787–808 (2019)
26. Xia, J., Chandrasekaran, S., Gu, M., Li, X.S.: Superfast multifrontal method for large structured linear systems of equations. SIAM J. Matrix Anal. Appl. **31**(3), 1382–1411 (2009)
27. Zhu, L., Ju, L., Zhao, W.: Fast high-order compact exponential time differencing Runge–Kutta methods for second-order semilinear parabolic equations. J. Sci. Comput. **67**, 1043–1065 (2016)