Kronecker Products on Preconditioning

Thesis by

Longfei Gao

In Partial Fulfillment of the Requirements

For the Degree of

Doctor of Philosophy

King Abdullah University of Science and Technology, Thuwal,

Kingdom of Saudi Arabia

(August, 2013)

The thesis of Longfei Gao is approved by the examination committee

Committee Chairperson:	Victor Calo
Committee Member:	David Keyes
Committee Member:	Yalchin Efendiev
Committee Member:	Shuyu Sun

Copyright © 2013

Longfei Gao

All Rights Reserved

ABSTRACT

Kronecker products on preconditioning

Longfei Gao

Numerical techniques for linear systems arising from discretization of partial differential equations are nowadays essential for understanding the physical world. Among these techniques, iterative methods and the accompanying preconditioning techniques have become increasingly popular due to their great potential on large scale computation.

In this work, we present preconditioning techniques for linear systems built with tensor product basis functions. Efficient algorithms are designed for various problems by exploiting the Kronecker product structure in the matrices, inherited from tensor product basis functions.

Specifically, we design preconditioners for mass matrices to remove the complexity from the basis functions used in isogeometric analysis, obtaining numerical performance independent of mesh size, polynomial order and continuity order; we also present a compound iteration preconditioner for stiffness matrices in two dimensions, obtaining fast convergence speed; lastly, for the Helmholtz problem, we present a strategy to 'hide' its indefiniteness from Krylov subspace methods by eliminating the part of initial error that corresponds to those negative generalized eigenvalues. For all three cases, the Kronecker product structure in the matrices is exploited to achieve high computational efficiency.

ACKNOWLEDGEMENTS

Thanks to my friends for their consistent help and support.

Thanks to KAUST for such a wonderful research opportunity.

♡ ♠ ♢ ♣

Dedicated to my parents.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

Acronym	**Meaning**
ADI	Alternating Direction Implicit
CAD	Computer-Aided Design
CG	Conjugate Gradient
dofs	degrees of freedom
FEA	Finite Element Analysis
FEM	Finite Element Method
GLL	Gauss-Lobatto-Legendre
GMRES	Generalized Minimal Residual
IGA	Isogeometric Analysis
NURBS	Non-Uniform Rational B-Spline
PDE	Partial Differential Equation
SPD	Symmetric Positive-Definite

LIST OF SYMBOLS

Symbol	Meaning	First appearance
Operators		
,	Partial differentiation	Page 15
$\otimes$	Kronecker product	Page 18
Vec	Reshape a vector to a matrix	Page 23
Mat	Reshape a matrix to a vector	Page 24
$\|\cdot\|_M$	M-norm	Page 53
Mathematical quantities		
w_i	Weight for the ith NURBS basis function	Page 27
v_i	The ith (generalized) eigenvector	Page 47
λ_i^X/λ_i^Y	The ith (generalized) eigenvalue	Page 47/47
Numerical parameters		
p	Polynomial order of B-spline basis functions	Page 30
c	Continuity order of B-spline basis functions	Page 30
h	Mesh size	Page 30
N_{1D}	Number of 1D elements	Page 30
$r^{(k)}$	Acceleration parameter of the kth ADI iteration	Page 46
N_{inner}	Number of inner iteration steps	Page 64
Physical parameters		
$\boldsymbol{\kappa}/\kappa$	Diffusivity	Page 57/67
k	Wavenumber	Page 76
ρ	Density	Page 87
$\mathcal{B}$	Bulk modulus	Page 87
Other symbols		
^{s}	Symbol for stagnation	Page 70
—	Symbol for reaching maximum iterations	Page 70

Chapter 1

Introduction

High dimensional basis functions are needed for various scientific purposes, particularly on numerical simulation of partial differential equations (PDEs). One straightforward way to construct high dimensional basis functions is to build them as tensor products of one dimensional (1D) basis functions. We call these high dimensional basis functions the tensor product basis functions.

Despite of the conciseness, tensor product basis functions are not very popular for finite element method (FEM) due to the limitation that they can only be constructed on rectangular domains.

However, with the rise of isogeometric analysis (IGA), where a rectangular computational domain is linked with the complicated physical domain by a global mapping, tensor product basis functions are getting more and more attention in recent days. In fact, one of the motivations of this work is to develop fast iterative solvers for linear algebraic systems arising from IGA.

In simple cases, FEM matrices built with tensor product basis functions possess the Kronecker product property. (Kronecker product is a specific terminology for tensor product with restricted use on matrices.) This leads to efficient algorithms on matrix operations, among which the inversion for Kronecker product matrices is particularly appealing due to its low computational cost.

However, this nice Kronecker product structure is easily destroyed by the com-

plexities of PDEs, for instances, complicated geometry, complicated coefficients, complicated operators and so on. To obtain efficient PDE solvers that can take these complexities into account, we resort to preconditioned Krylov subspace methods where the Kronecker product property is exploited when applying the preconditioners.

1.1 Problem Statement and Research Approach

Mass matrices and stiffness matrices arise from finite element discretization of various PDE related problems. The attempt to understand the physical phenomena described by these PDEs often leads us to the computational challenge of inverting a mass matrix, a stiffness matrix or a linear combination of these two. We give several simple model problems in 2D as examples in the following to demonstrate the origins of these matrices and the Kronecker product structure they inherited from tensor product basis functions. These model problems can be generalized to 3D easily.

Model Problem 1.

Heat equation (2D):

$$u_{,t} = u_{,xx} + u_{,yy} + f \tag{1.1}$$

defined on a rectangular domain, where ',' indicates partial differentiation.

Discretizing (1.1) using (Galerkin's) finite element method with 2D tensor product basis functions $B = B^y \otimes B^x$, the term u_t leads us to the 2D mass matrix: $M = M^y \otimes M^x$ while $u_{,xx}$ and $u_{,yy}$ lead us to matrices $K^X = M^y \otimes K^x$ and $K^Y = K^y \otimes M^x$, respectively. The sum of K^X and K^Y gives us the 2D stiffness matrix K.

In the above expressions, M^x and M^y stand for the 1D mass matrices built with B^x and B^y, respectively; K^x and K^y stand for the 1D stiffness matrices built with B^x and B^y, respectively.

Different approaches for solving PDE (1.1) lead to different linear algebraic problems. For instance, explicit time integration schemes lead to the problem of inverting

mass matrix M; implicit time integration scheme lead to the problem of inverting a linear combination of M and K: $M + \delta K$; if only the steady state solution is concerned, where $u_{,t} = 0$, inversion of K becomes the linear algebraic problem.

Model Problem 2.

Membrane vibration (2D):

$$u_{,xx} + u_{,yy} + \lambda u = 0 \tag{1.2}$$

defined on a rectangular domain.

After finite element discretization, (1.2) leads us to the following generalized eigenvalue problem:

$$Kb = \lambda Mb,$$

where K and M have the same definitions as in Model Problem 1 while b is a vector with compatible size. Most numerical algorithms for generalized eigenvalue problems require inverting the mass matrix M, see [6, 61, 71].

Model Problem 3.

Helmholtz equation (2D):

$$-\left(u_{,xx} + u_{,yy}\right) - k^2 u = f \tag{1.3}$$

defined on a rectangular domain.

After finite element discretization, the matrix to be inverted in the linear algebraic problem is: $\left(K - k^2 M\right)$, where K and M have the same definitions as in Model Problem 1. The indefiniteness of this matrix adds significant numerical challenge, see [31, 34, 39].

The purpose of this work is to develop fast iterative solvers to apply the inverses of these matrices efficiently. The Kronecker product structure in M, K^X and K^Y can be exploited to develop fast iterative solvers by direction separation.

However, most real life problems cannot be modeled by these simple model problems and the Kronecker product structure of these matrices cannot be maintained even if tensor product basis functions are used. This is due to the appearances of complicated physical domains, non-constant coefficients or other factors that couple the different spatial directions together. In these scenarios, we resort to preconditioned Krylov subspace methods and exploit the Kronecker product property when applying the preconditioners.

1.2 Objectives and Contributions

Efficient algorithms for inverting matrices arising from finite element discretization of PDEs with tensor product basis functions can be designed by exploiting the Kronecker product structure in these matrices, inherited from the basis functions.

The contributions of this thesis include:

- Preconditioning techniques for mass matrices.
- Preconditioning techniques for stiffness matrices.
- Preconditioning techniques for matrices arising from the Helmholtz equation.

Chapter 2

Background

2.1 Kronecker product

The Kronecker product operator, denoted by $\otimes$, is a special case of the tensor product operator that is restrictively used on matrices. Symbolically, for matrices A and B with arbitrary sizes,

$$A \otimes B = \begin{bmatrix} a_{11}B & \cdots & a_{1n}B \\ \vdots & \ddots & \vdots \\ a_{m1}B & \cdots & a_{mn}B \end{bmatrix},$$

where a_{ij} is an entry of A at the ith row and jth column with i ranging from 1 to m and j ranging from 1 to n. Every entry in the first matrix A is replaced by the second matrix B and then scaled by that replaced entry.

Kronecker product has mathematically elegant and practically useful properties. We list several of them in the following:

- Mixed-product:

$$(A \otimes B)(C \otimes D) = AC \otimes BD$$

- Inverse:

$$(A \otimes B)^{-1} = A^{-1} \otimes B^{-1}$$

- Transpose:

$$(A \otimes B)^T = A^T \otimes B^T$$

- Associative:

$$(A \otimes B) \otimes C = A \otimes (B \otimes C).$$

In the mixed-product and associative properties, the matrices need to have compatible sizes so that multiplication makes sense; in the inverse property, both A and B need to be invertible.

For derivations of the above properties and more information about Kronecker products, see [75, 79]. [81] also serves as a valuable review on Kronecker products.

2.2 An algebraic view of the finite element matrices built with tensor product basis functions

For derivations of matrices M, K^X and K^Y from the model problems described in the introduction, one can consult, for instance, [15, 20, 50, 76]. Here, we only point out the entries of these matrices and the structure they inherited from tensor product basis functions.

With a set of basis functions B, listed as a column vector, the mass matrix M can be expressed in the following concise form:

$$M = \int_\Omega BB^T d_\Omega. \tag{2.1}$$

Denote the partial derivative of B with respect to x as $B_{,x}$ and its derivative with respect to y as $B_{,y}$, also listed as column vectors, K^X and K^Y can be expressed as:

$$K^X = \int_\Omega (B_{,x})(B_{,x})^T d_\Omega \quad \text{and} \quad K^Y = \int_\Omega (B_{,y})(B_{,y})^T d_\Omega, \tag{2.2}$$

correspondingly.

Take the 2D case as an example, assume B is built as: $B = B^y \otimes B^x$, where B^y and B^x are 1D basis functions, listed as column vectors. We readily have $B_{,y} = B^y_{,y} \otimes B^x$ and $B_{,x} = B^y \otimes B^x_{,x}$, where $B^x_{,x}$ and $B^y_{,y}$ stand for the derivatives of B^x and B^y, respectively. After substituting these relations into (2.1) and (2.2) and applying the mixed-product property, we have

$$M = \int_\Omega \left(B^y (B^y)^T\right) \otimes \left(B^x (B^x)^T\right) d_\Omega, \tag{2.3}$$

and

$$\begin{aligned} K^X &= \int_\Omega \left(B^y (B^y)^T\right) \otimes \left(B^x_{,x} (B^x_{,x})^T\right) d_\Omega, \\ K^Y &= \int_\Omega \left(B^y_{,y} (B^y_{,y})^T\right) \otimes \left(B^x (B^x)^T\right) d_\Omega. \end{aligned} \tag{2.4}$$

Realizing that the integration domain Ω is rectangular in the model problems, the integrals in (2.3) and (2.4) can be separated into products of 1D integrals:

$$M = \left(\int_y B^y (B^y)^T d_y\right) \otimes \left(\int_x B^x (B^x)^T d_x\right), \tag{2.5}$$

and

$$\begin{aligned} K^X = &\left(\int_y B^y (B^y)^T d_y\right) \otimes \left(\int_x B^x_{,x} (B^x_{,x})^T d_x\right), \\ K^Y = &\left(\int_y B^y_{,y} (B^y_{,y})^T d_y\right) \otimes \left(\int_x B^x (B^x)^T d_x\right). \end{aligned} \tag{2.6}$$

Notice that $\int_y B^y (B^y)^T d_y$ and $\int_x B^x (B^x)^T d_x$ are simply the 1D mass matrices built with B^y and B^x respectively while $\int_y B^y_{,y} (B^y_{,y})^T d_y$ and $\int_x B^x_{,x} (B^x_{,x})^T d_x$ are the corresponding 1D stiffness matrices, we can express M, K^X and K^Y in the following concise forms:

$$M = M^y \otimes M^x, \qquad K^X = M^y \otimes K^x, \qquad K^Y = K^y \otimes M^x, \tag{2.7}$$

with the help of the following notations:

$$M^y = \int_y B^y (B^y)^T d_y,\ M^x = \int_x B^x (B^x)^T d_x,\ K^y = \int_y B^y_{,y} (B^y_{,y})^T d_y,\ K^x = \int_x B^x_{,x} (B^x_{,x})^T d_x.$$

However, we need to bear in mind that this separation is not valid whenever there is a coupling term inside these integrals, even when the integration domain is rectangular.

2.3 Isogeometric analysis and its basis functions

Isogeometric analysis (IGA) was first introduced by Hughes *et al* [51] in 2005 with the attempt of bridging the two communities of computer-aided design (CAD) and finite element analysis (FEA) together.

The existing gap between these two communities is that the geometry generated by CAD is not necessarily suitable for FEA. The procedure of translation from CAD-generated geometry to FEA-suitable geometry might take much longer time than it takes for FEA. IGA is proposed to resolve this issue by using only one geometric model for both CAD and FEA. One approach is to adopt the basis functions for CAD directly for FEA.

B-splines and non-uniform rational B-splines (NURBS) are perhaps the most popular basis functions in CAD community, see [27, 40, 65, 69]. There has been a tremendous effort in the IGA literature on using these basis functions for FEA, for instance, see [3, 9, 16, 22, 23, 57]. Higher dimensional B-spline basis functions are built as tensor products of 1D B-spline basis functions while higher dimensional NURBS basis functions are built as weighted B-spline basis functions. Thus the algorithms that we developed for tensor product basis functions can naturally be applied on IGA.

Under the IGA framework, restricted by their tensor product structure, basis functions are built on a rectangular computational domain while the PDEs are posed

on a more complicated physical domain. As a linear combination of these basis functions, our approximate solution is also constructed on this rectangular computational domain. A global mapping links these two domains together. Change of variables in the integrals is needed when assembling the linear algebraic systems and this naturally introduces a coupling term inside the integrals, for instance, determinant of the Jacobian matrix for mass matrices.

Chapter 3

Mass matrix

As mentioned in the introduction, for PDE related problems, the need for inverting mass matrices can come from explicit dynamic evolution for time dependent PDEs or generalized eigenvalue problems. There are various techniques on inverting mass matrices, among which the most well-known one is probably the lumped-mass method, for instance, see [42, 49, 50]. Despite of its simplicity, lumped-mass method suffers from its inaccuracy. Here we propose an iterative method that can invert the original mass matrices accurately and efficiently.

3.1 The ideal case

For model problems, if tensor product basis functions are used, the 2D mass matrix is the Kronecker product of two 1D mass matrices: $M = M^y \otimes M^x$. Fast algorithm for inverting Kronecker product matrices can be developed by exploiting this structure, for instance, see [26, 63]. We restate this algorithm here for completeness. For simplicity, we first introduce operators Vec and Mat, which will become handy in the upcoming discussion. These two operators are borrowed from [81] and [58].

Definition 3.1. *Let $B \in R^{m\times n}$, $Vec(B)$ is the operator that returns a column vector*

at length mn by stacking all the columns of B together under the natural order:

$$Vec(B) \equiv \begin{bmatrix} B(:,1) \\ \vdots \\ B(:,n) \end{bmatrix}. \tag{3.1}$$

Definition 3.2. *Let b be a column vector at length mn, $Mat(b,m,n)$ is the operator that returns a matrix of size $m \times n$ by chopping b into n pieces at length m and putting these pieces into the matrix under the natural order:*

$$Mat(b,m,n) \equiv [b(1:m), \cdots, b((n-1)m+1:nm)]. \tag{3.2}$$

In the above definitions, we also borrowed the colon notation ':' from MATLAB [59].

With the above notations, we now present an algorithm to solve the following linear system:

$$(M^y \otimes M^x)\, x = b, \tag{3.3}$$

where M^y has size $N_y \times N_y$ and M_x has size $N_x \times N_x$ while x and b are column vectors at length $N_y N_x$. Since M and b are built from basis functions $B^y \otimes B^x$, the entries in x and b have the corresponding order as in $B^y \otimes B^x$.

Algorithm 1.

1: $B = Mat(b, N_x, N_y)$;
2: $T = zeros(N_x, N_y)$;
3: $X = zeros(N_x, N_y)$;
4: ***for*** $j = 1 : N_y$ ***do***
5: $\quad T(:,j) = M^x \backslash B(:,j)$;
6: ***end for***
7: ***for*** $i = 1 : N_x$ ***do***
8: $\quad X(i,:)^T = M^y \backslash T(i,:)^T$;
9: ***end for***
10: $x = Vec(X)$;

In Algorithm 1, we also borrowed the commands '*zeros*' and '\' from MATLAB.

If we use a direct method to apply the inverses of M^x and M^y in the above algorithm, and further, if M^x and M^y are dense, the factorization cost is $\mathcal{O}\left((N_x)^3 + (N_y)^3\right)$ while the substitution cost is $\mathcal{O}\left((N_x)^2 N_y + (N_y)^2 N_x\right)$; However, if M^x and M^y are banded diagonal matrices with bandwidth independent of matrix size, which is common if they arise from finite element discretization due to the local support of basis functions, the factorization cost is $\mathcal{O}\left(N_x + N_y\right)$ while the substitution cost is $\mathcal{O}\left(N_x N_y\right)$. In other words, for the case when M^x and M^y are banded diagonal, the computational cost for solving (3.3) with Algorithm 1 is linear.

An illustration of Algorithm 1 is shown in Figure 3.1: We apply first M^x for all the columns of the original data matrix from the vertical direction and then apply M^y for all the rows of the intermediate data matrix from the horizontal direction.

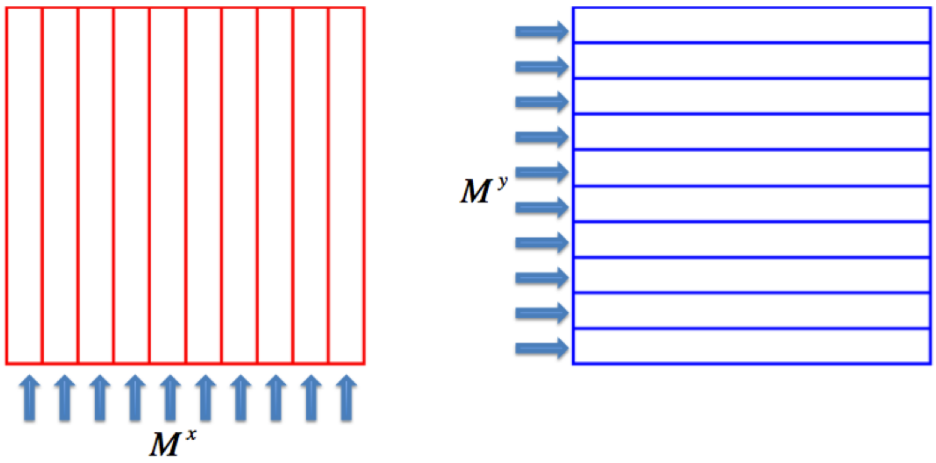


Figure 3.1: Visualization of Algorithm 1.

Algorithm 1 can be generalized to solve linear systems involving higher dimensional Kronecker product matrices:

$$\left(M^d \otimes \cdots \otimes M^1\right) x = b, \tag{3.4}$$

where M^i has size $N_i \times N_i$ for $i = 1, \cdots, d$ while x and b are column vectors at length $\left(\prod_{i=1}^{d} N_i\right)$, as shown in Algorithm 2. Again, M, x and b are related with the set of

basis functions $B^d \otimes \cdots \otimes B^1$.

Algorithm 2.

1: $B = Mat(b, N_1, \cdots, N_d)$;
2: $T = zeros(N_1, \cdots, N_d)$;
3: $X = zeros(N_1, \cdots, N_d)$;
4: $T(:, i_2, i_3, \cdots, i_d) = M^1 \backslash B(:, i_2, i_3, \cdots, i_d)$;
5: $T(i_1, :, i_3, \cdots, i_d) = M^2 \backslash B(i_1, :, i_3, \cdots, i_d)$;
6: $\vdots$
7: $T(i_1, \cdots, i_{d-1}, :) = M^d \backslash B(i_1, \cdots, i_{d-1}, :)$;
8: $x = Vec(X)$;

Corresponding generalizations of Vec and Mat to higher dimensions are implied in Algorithm 2. Again, for the case when all the M^i are banded diagonal, the computational cost for solving (3.4) with Algorithm 2 is linear.

3.2 More complicated cases

However, the nice Kronecker product property of the mass matrix: $M = M^y \otimes M^x$ is only true for simple cases, like those model problems. For more complicated situations, this property does not hold. Here we give several examples where coupling terms appear in the integrals, preventing it from being separated by directions.

Complicated geometry. Lots of real life PDEs are posed on complicated domains instead of rectangles. One approach to deal with complicated domains is isogeometric analysis (IGA) where the same set of basis functions are used to represent both the geometry and the solution.

Under this IGA framework, basis functions are defined on a rectangular computational domain (denoted as $\square$) and a geometric mapping constructed as a linear combination of these basis functions is defined to map this computational domain to the complicated physical domain (denoted as Ω), where the PDE of interest is posed. The approximate solution we are searching for is also a linear combination of these

basis functions, undergoing this geometric mapping.

When assembling the mass matrix, change of integration variables (from variables on physical domain to variables on computational domain) is required, which introduces an extra term in the integrals: the determinant of the Jacobian matrix, denoted by J, associated with the geometric mapping. In general, this extra term J cannot be separated as products of two functions, each of which depends only on one spatial variable, thus these integrals cannot be broken down as we did in (2.5). Therefore, this J becomes a coupling term that keeps the 2D integrals unfactorizable.

Non-constant coefficient. This case is relatively straightforward. Model Problem 1 deals only with u_t while in some problems a coefficient ρ appears in front of u_t. When assembling the mass matrix, ρ arises as a coupling term in the integrals.

Rational basis functions. Under the IGA framework, non-uniform rational B-spline (NURBS) basis functions are very popular due to their capability of representing complicated geometries. We briefly introduce NURBS basis functions in the following. For more details, one can consult [21, 40, 51, 64, 65, 69].

A NURBS basis set is built from a B-spline basis set in the following procedure: Given a B-spline basis set: $B = \{B_1, B_2, \ldots, B_N\}$, assign weight w_i to each B-spline basis function B_i, for $i = 1, \cdots, N$, where w_i is a positive real number. Define the weight function W_N as:

$$W_N = \sum_{i=1}^{N} w_i B_i.$$

The NURBS basis set is then defined as:

$$R = \{R_1, R_2, \ldots, R_N\},$$

where

$$R_i = \frac{B_i}{W_N}. \tag{3.5}$$

NURBS basis functions are more powerful than B-splines on representing complicated

geometries due to the freedom of choosing the weights.

Remark 3.1. (3.5) *is not the traditional definition of NURBS basis functions as one might find, for instance, in [21] or [65], where the definition is given as:*

$$R^{tra} = \{R_1^{tra}, R_2^{tra}, \ldots, R_N^{tra}\},$$

where

$$R_i^{tra} = \frac{w_i B_i}{W_N}.$$

Apparently R and R^{tra} are equivalent in the sense that they span the same function space. We prefer R because it leads to a simpler form of the mass matrices.

Along with bringing more flexibility and capacity on representing complicated geometries, these weights w_i also brings complexities to NURBS basis functions. Higher dimensional NURBS basis functions are not tensor product of 1D NURBS basis functions. However, they can be understood as weighted B-spline basis functions with W_N serving as the common weight for all basis functions. In this way, the numerators of higher dimensional NURBS basis functions (according to (3.5), they are actually B-splines) can be viewed as tensor products of 1D B-spline basis functions.

When assembling the mass matrix, $\frac{1}{W_N^2}$ becomes an extra term that appears inside the integrals, comparing with the case of assembling with B-spline basis functions. In general, this weight function is not separable, thus $\frac{1}{W_N^2}$ becomes a coupling term.

Remark 3.2. *There are cases where complicated geometries and NURBS basis functions are involved, but the 2D mass matrix still possesses the Kronecker product property. See Appendix A.6 for the example of a quarter annulus.*

Remark 3.3. *In the following section, we only concentrate on the first case, i.e., coupling due to complicated geometry. However, coupling due to non-constant coefficient or rational basis functions can be dealt with in exactly the same manner.*

3.3 Preconditioning

In cases like the three examples shown above where coupling terms arise in the integrals, the mass matrix, for instance, in the case of complicated geometry:

$$M^J = \int_{\square} \left(B^\eta (B^\eta)^T\right) \otimes \left(B^\xi (B^\xi)^T\right) J d_{\square},$$

cannot be written as the Kronecker product of 1D mass matrices as we did in (2.5). Therefore, Algorithm 1 cannot be applied on M^J directly.

Here, we use variables ξ and η in stead of x and y in order to distinguish between the parametrical domain (ξ and η) and the physical domain (x and y). B^ξ and B^η stand for the sets of 1D basis functions, defined on the parametrical domain. Denote by N_ξ and N_η the sizes of B^ξ and B^η, respectively.

3.3.1 The simplest choice: M^{-1}

The similarity between M^J and M urges us to think the following question:

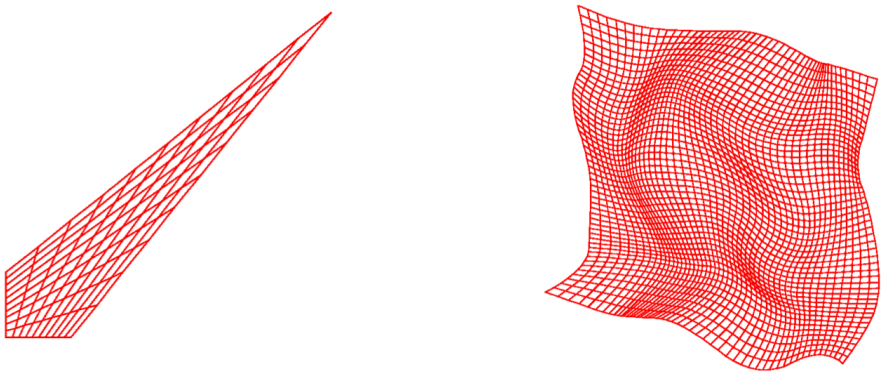
Can we still exploit the simple structure of M on the task of inverting M^J?

The answer is yes and the approach is rather natural: to use an iterative method with M^{-1} serving as the preconditioner for inverting M^J.

Among the various iterative methods, we choose the conjugate gradient (CG) method for our specific matrix M^J due to its symmetric positive definiteness. For more information about Krylov subspace methods and in general, iterative methods and preconditioning techniques, see [5, 30, 41, 46, 47, 70, 77, 83]. [4, 10, 45, 72] also serve as elegant review papers on relevant topics.

In Tables 3.1 and 3.2, we show some numerical results regarding the performance of CG with the preconditioner M^{-1} on two testing domains, namely, the stretched rectangle and the perturbed rectangle shown in Figure 3.2. Detailed information about these two testing domains can be found in Appendix A.1. The iteration process

is stopped when the relative residual (in ℓ_2 norm) is less than $1e$-12. Numerical results with the lumped-mass preconditioner are also shown in these tables for comparison.



Stretched rectangle. Perturbed rectangle.

Figure 3.2: Testing domains.

In Table 3.1, polynomial order (p) and continuity order (c) of the basis functions are fixed while mesh size ($h = 1/N_{1D}$) is varying; in Table 3.2, mesh size is fixed while polynomial order and continuity order of the basis functions are varying with relationship: $c = p - 1$, i.e., the so called k-refinement [22, 24, 51].

	N_{1D}	2^3	2^4	2^5	2^6	2^7	2^8	2^9	2^{10}
Stretched	M^{-1}	31	32	32	32	33	33	33	33
	Lumped-mass	168	418	516	413	320	254	206	151
Perturbed	M^{-1}	29	34	39	40	42	44	44	45
	Lumped-mass	289	512	463	380	306	245	200	152

Table 3.1: h-scaling: $p = 4, c = 3$.

	p	1	2	3	4	5	6	7	8
Stretched	M^{-1}	33	33	33	33	33	33	33	33
	Lumped-mass	25	54	107	206	373	650	682	728
Perturbed	M^{-1}	44	45	44	44	44	44	44	44
	Lumped-mass	26	55	105	200	346	459	505	599

Table 3.2: k-scaling: $N_{1D} = 2^9$, $c = p - 1$.

Loosely speaking, iterative methods will have their best performances on identity matrix. Most effective iterative methods, if not all, will converge after their first

iteration step for this case. To solve a linear system, the deviation of the matrix from identity matrix requires iterative methods to iterate. And apparently, the more variance there is between these two matrices, the more iteration steps are required to remove this difference.

The role of a preconditioner is to bring the matrix closer to the identity matrix so that less variance is left for the iterative method to remove, thus a faster convergence, in terms of number of iteration steps, can be expected. The art of designing a good preconditioner is to put into it as much correction as possible and meanwhile maintain the ability to apply it efficiently.

Simple as it is, the preconditioner M^{-1} is actually quite powerful in the sense that it corrects the part of deviation that corresponds to the structure of the matrix, which eventually comes from the complexity of basis functions. The deviation left for the conjugate gradient method to remove mainly comes from the complicated geometry. This is why we observed a performance, in terms of number of iteration steps, independent with mesh size in Table 3.1 and independent with polynomial order in Table 3.2.

Additional numerical results can be found in Appendix A.2. In Table A.4, mesh size and continuity order are fixed while polynomial order is varying; In Table A.5, mesh size and polynomial order are fixed while continuity order is varying.

3.3.2 Partial inclusion of geometric information in the preconditioners

Based on the above observations, the following question is natural:

Can we put the geometric information, at least partially, into the preconditioners so that the difficulty of the task left for iterative methods is further reduced?

The answer is, again, yes and the ingredient is the local support property of basis

functions. Considering a representative entry of M^J:

$$M^J_{AB} = \int_{\square} (B^\eta_i B^\eta_k)(B^\xi_j B^\xi_l) J d_{\square},$$

where $A = (i-1)N_\xi + j$, $B = (k-1)N_\xi + l$. If basis function B^η_i only has local support, then the integration region for M^J_{AB} on η direction is restricted on the support region for B^η_i and the 2D integration domain is restricted on a thin strip of the whole rectangle, illustrated in Figure 3.3.

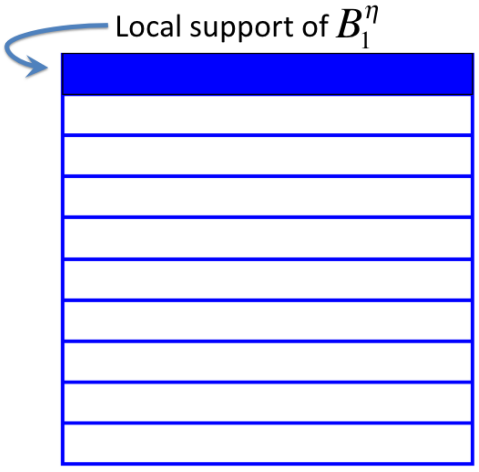


Figure 3.3: Local support property of basis functions.

It is somehow plausible to assume that $J_{\eta|i}$, meaning J restricted on the ith thin strip, does not have much variation on η direction, simply due to the shortness of the support region. Under this assumption, we approximate M^J_{AB} by

$$M^J_{AB} \approx \left(\int_\eta B^\eta_i B^\eta_k d_\eta \right) \left(\frac{1}{L^\eta_i} \int_{\square} B^\xi_j B^\xi_l J_{\eta|i} d_{\square} \right),$$

where L^η_i is the length of the support region for B^η_i. By introducing the following notations:

$$M^\xi_i = \frac{1}{L^\eta_i} \int_{\square} B^\xi (B^\xi)^T J_{\eta|i} d_{\square}, \tag{3.6}$$

for $i = 1, \cdots, N_\eta$, we can write down the corresponding approximation of M^J in the

following generalized Kronecker product form:

$$M^J \approx M^\xi = \begin{bmatrix} M^\eta(1,:) \otimes M_1^\xi \\ \vdots \\ M^\eta(N_\eta,:) \otimes M_{N_\eta}^\xi \end{bmatrix}. \tag{3.7}$$

A similar generalization of Kronecker product can be found in [68].

A fast algorithm can be developed to invert M^ξ, as shown in Algorithm 3 and illustrated in Figure 3.4.

Algorithm 3.

1: $B = Mat(b, N_\xi, N_\eta)$;
2: $T = zeros(N_\xi, N_\eta)$;
3: $X = zeros(N_\xi, N_\eta)$;
4: ***for*** $j = 1 : N_\eta$ ***do***
5: $\quad T(:, j) = M_j^\xi \backslash B(:, j)$;
6: ***end for***
7: ***for*** $i = 1 : N_\xi$ ***do***
8: $\quad X(i,:)^T = M^\eta \backslash T(i,:)^T$;
9: ***end for***
10: $x = Vec(X)$;

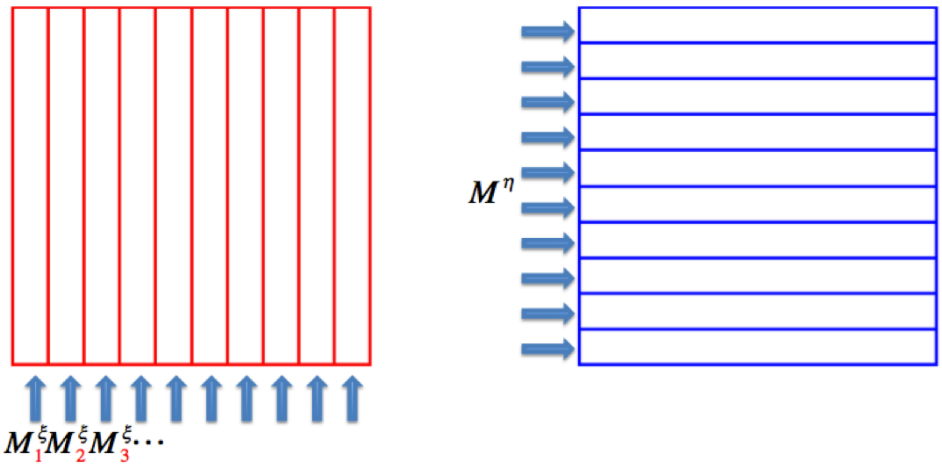


Figure 3.4: Visualization for Algorithm 3.

Algorithm 3 might be easier to understand if we apply the mix-product property of

Kronecker products and reformulate M^ξ as following:

$$
\begin{aligned}
M^J \approx M^\xi &= \begin{bmatrix} M^\eta(1,:) \otimes M_1^\xi \\ \vdots \\ M^\eta(N_\eta,:) \otimes M_{N_\eta}^\xi \end{bmatrix} \\
&= \begin{bmatrix} \left(I^1 M^\eta(1,:)\right) \otimes \left(M_1^\xi I^\xi\right) \\ \vdots \\ \left(I^1 M^\eta(N_\eta,:)\right) \otimes \left(M_{N_\eta}^\xi I^\xi\right) \end{bmatrix} \\
&= \begin{bmatrix} \left(I^1 \otimes M_1^\xi\right)\left(M^\eta(1,:) \otimes I^\xi\right) \\ \vdots \\ \left(I^1 \otimes M_{N_\eta}^\xi\right)\left(M^\eta(N_\eta,:) \otimes I^\xi\right) \end{bmatrix} \\
&= \begin{bmatrix} M_1^\xi & & \\ & \ddots & \\ & & M_{N_\eta}^\xi \end{bmatrix} \begin{bmatrix} M^\eta(1,:) \otimes I^\xi \\ \vdots \\ M^\eta(N_\eta,:) \otimes I^\xi \end{bmatrix} \\
&= \begin{bmatrix} M_1^\xi & & \\ & \ddots & \\ & & M_{N_\eta}^\xi \end{bmatrix} \begin{bmatrix} \\ M^\eta \otimes I^\xi \\ \\ \end{bmatrix}.
\end{aligned}
$$

In the above derivation, I^ξ stands for the identity matrix of the same size of M^ξ and I^1 stands for the identity matrix of size 1×1, i.e., the scalar 1.

It is now readily seen that applying the inverse of M^ξ can be decomposed to two stages: The first stage is to apply the inverse of the block diagonal matrix composed by M_i^ξ, which is taken care of by lines 4-6 in Algorithm 3. The second stage is to apply the inverse of the tensor product matrix $M^\eta \otimes I^\xi$, which is taken care of by lines 7-9 in Algorithm 3.

Although an efficient algorithm is at hand for exploitation, M^ξ cannot be combined with CG due to its asymmetry. Some modification is needed in order to regain the

symmetry. We turn to Cholesky decomposition factors for help.

Denote $\mathcal{L}^\eta$ as the lower Cholesky decomposition factor for M^η and $\mathcal{L}_i^\xi$ as the lower Cholesky decomposition factor for M_i^ξ. We present two symmetrized variants of M^ξ, namely, $M^{\xi S}$ in (3.8) and $M^{\xi M}$ in (3.9). Corresponding algorithms to apply their inverses can be found in Algorithm 4 and Algorithm 5.

$$M^{\xi S} = \begin{bmatrix} \mathcal{L}_1^\xi & & \\ & \ddots & \\ & & \mathcal{L}_{N_\eta}^\xi \end{bmatrix} \begin{bmatrix} & & \\ & M^\eta \otimes I^\xi & \\ & & \end{bmatrix} \begin{bmatrix} \left(\mathcal{L}_1^\xi\right)^T & & \\ & \ddots & \\ & & \left(\mathcal{L}_{N_\eta}^\xi\right)^T \end{bmatrix}. \tag{3.8}$$

Algorithm 4. (For $M^{\xi S}$)

1: $B = Mat(b, N_\xi, N_\eta)$;
2: $T = zeros(N_\xi, N_\eta)$;
3: $X = zeros(N_\xi, N_\eta)$;
4: ***for*** $j = 1 : N_\eta$ ***do***
5: $\quad X(:, j) = \mathcal{L}_j^\xi \backslash B(:, j)$;
6: ***end for***
7: ***for*** $i = 1 : N_\xi$ ***do***
8: $\quad T(i, :)^T = M^\eta \backslash X(i, :)^T$;
9: ***end for***
10: ***for*** $j = 1 : N_\eta$ ***do***
11: $\quad X(:, j) = \left(\mathcal{L}_j^\xi\right)^T \backslash T(:, j)$;
12: ***end for***
13: $x = Vec(X)$;

We show some numerical results regarding the quality of $M^{\xi S}$ in Table 3.3 and Table 3.4. Again, numerical results with the lumped-mass preconditioner are shown for comparison. Additional numerical results are provided in Appendix A.3, including the performance of $M^{\xi M}$.

$$M^{\xi M} = \begin{bmatrix} & & \\ & \mathcal{L}^\eta \otimes I^\xi & \\ & & \end{bmatrix} \begin{bmatrix} M_1^\xi & & \\ & \ddots & \\ & & M_{N_\eta}^\xi \end{bmatrix} \begin{bmatrix} & & \\ & (\mathcal{L}^\eta)^T \otimes I^\xi & \\ & & \end{bmatrix}. \tag{3.9}$$

Algorithm 5. (For $M^{\xi M}$)

1: $B = Mat(b, N_\xi, N_\eta)$;
2: $T = zeros(N_\xi, N_\eta)$;
3: $X = zeros(N_\xi, N_\eta)$;
4: ***for*** $i = 1 : N_\xi$ ***do***
5: $\quad X(i,:)^T = \mathcal{L}^\eta \backslash B(i,:)^T$;
6: ***end for***
7: ***for*** $j = 1 : N_\eta$ ***do***
8: $\quad T(:,j) = M_j^\xi \backslash X(:,j)$;
9: ***end for***
10: ***for*** $i = 1 : N_\xi$ ***do***
11: $\quad X(i,:)^T = (\mathcal{L}^\eta)^T \backslash T(i,:)^T$;
12: ***end for***
13: $x = Vec(X)$;

	N_{1D}	2^3	2^4	2^5	2^6	2^7	2^8	2^9	2^{10}
Stretched	$(M^{\xi S})^{-1}$	10	8	6	6	5	4	4	3
	Lumped-mass	168	418	516	413	320	254	206	151
Perturbed	$(M^{\xi S})^{-1}$	21	16	12	10	8	7	6	5
	Lumped-mass	289	512	463	380	306	245	200	152

Table 3.3: h-scaling: $p = 4, c = 3$.

	p	1	2	3	4	5	6	7	8
Stretched	$(M^{\xi S})^{-1}$	3	4	4	4	4	4	4	4
	Lumped-mass	25	54	107	206	373	650	682	728
Perturbed	$(M^{\xi S})^{-1}$	5	5	6	6	6	7	7	7
	Lumped-mass	26	55	105	200	346	459	505	599

Table 3.4: k-scaling: $N_{1D} = 2^9$, $c = p - 1$.

In Table 3.3, the number of iterations corresponding to $M^{\xi S}$ is decreasing as we refine the mesh. This is because the assumption we made before: $J_{\eta|i}$ does not have much variation on η direction, becomes more valid since the support region of B_i^η decreases as the mesh size gets smaller.

In Table 3.4, the number of iterations corresponding to $M^{\xi S}$ is increasing as we increase the polynomial order p. This is because the same assumption we made before becomes less valid since the support region of B_i^η increases as we increase p

while keeping $c = p - 1$.

Analogically to M^{ξ}, we can also approximate M^{J} by partially putting the geometric information into the η direction.

3.3.3 Computational cost per iteration

Tables 3.1 - 3.4 tell us only one side of the story: the number of iteration steps required for convergence is indeed reduced by the preconditioners. In the following, we briefly investigate the computational cost at each iteration step, including the cost for applying different preconditioners and the cost for matrix vector multiplication. In this section, we only deal with the special case where $c = p-1$. More general cases are dealt with in Appendix A.4.

For the preconditioner M^{-1}, the factorization cost is merely for the two 1D matrices, M^{ξ} and M^{η}, thus can be neglected. The major computational cost for applying M^{-1} is associated with the backward and forward substitutions, roughly at $4(2p+1)\cdot N$ operation counts. It is not competitive when comparing with the lumped mass preconditioner, which only requires $1 \cdot N$ operations to apply. However, it turns out that the computational cost for matrix vector multiplication is dominating at each iteration step: roughly at $2(2p+1)^2 \cdot N$ operation counts. Thus, the total computational cost per iteration is barely increased by a factor around $\dfrac{2}{2p+1}$. As we increase p, the preconditioner M^{-1} actually becomes 'cheaper' in a comparative sense.

For the preconditioner $\left(M^{\xi S}\right)^{-1}$, the computational cost for backward and forward substitutions is the same as M^{-1}. However, the factorization cost becomes more expensive. All of M_i^{ξ} and M^{η}, in total $(N_{\eta}+1)$ 1D matrices, need to be factorized, which amounts to roughly $(3p+1)(p+1)\cdot N$ operation counts. Since $(3p+1)(p+1) < 2(2p+1)^2$, we can conclude that, roughly, the factorization cost for $\left(M^{\xi S}\right)^{-1}$ amounts to 1 additional iteration cost. Similar results can be obtained for $\left(M^{\xi M}\right)^{-1}$.

To sum up, the additional computational cost for applying the preconditioner

$\left(M^{\xi S}\right)^{-1}$ or $\left(M^{\xi M}\right)^{-1}$ is marginal.

Remark 3.4. *When generalized to the 3D case, the computational cost for backward and forward substitutions is roughly at* $6(2p+1)\cdot N$ *while for matrix vector multiplication, it becomes* $2(2p+1)^3\cdot N$. *Therefore, the factor of the additional computational cost per iteration is even smaller:* $\dfrac{3}{(2p+1)^2}$.

Remark 3.5. *Our purpose of this discussion is merely to qualitatively demonstrate the marginal additional cost for applying these preconditioners. That is why in the above, we treat the four elementary arithmetic operations: addition, subtraction, multiplication and division, equivalently for simplicity, which may deviate from the reality.*

3.3.4 Hybrid preconditioning

Despite of the acceleration of convergence speed, the extra factorization cost and associated memory requirement are somehow unpleasant, particularly when it comes to time-dependent nonlinear problems, where the mass matrix can change from step to step. In this case, sometimes even the mass matrix is not explicitly formed due to the prohibitively high assembly cost. The assembly of M^{ξ} acquires 2D global information as well. Although dissected into different strips, it can still be prohibitive. This observation motivates us to look for an alternative strategy for preconditioning, which requires lower assembling cost.

Recall the numerical results in Table 3.1 and Table 3.2, where iteration steps corresponding to M^{-1} exhibit independent behavior with respect to mesh size and polynomial order. The complicated matrix structure is well 'preconditioned' by M^{-1}. If in addition, we can find another preconditioner to 'precondition' the complicated geometry solely, we might be able to achieve fast convergence speed by hybridizing these two preconditioners together.

In searching for such a preconditioner, let us first look at the numerical performance of the Jacobi preconditioner, shown in Table 3.5. Despite of its slow con-

vergence speed comparing with M^{-1}, there is some nice feature about the Jacobi preconditioner: it works better as we refine the mesh. This indicates that the Jacobi preconditioner might be able to 'precondition' the complicated geometry, in contrast with M^{-1}. For a detailed discussion on the effect of the Jacobi preconditioner on mass matrices, one can consult [89, 90].

	N_{1D}	2^3	2^4	2^5	2^6	2^7	2^8	2^9	2^{10}
Stretched	M^{-1}	31	32	32	32	33	33	33	33
	Jacobi	168	414	516	387	328	301	287	269
Perturbed	M^{-1}	29	34	39	40	42	44	44	45
	Jacobi	276	500	458	378	327	308	292	275

Table 3.5: h-scaling: $p = 4, c = 3$.

Based on the above observation, the following question naturally arises:

Can we combine these two preconditioners, M^{-1} and Jacobi, together to achieve faster convergence than with each one individually?

The answer is, again, yes, but some extra effort is needed. If we simply put the square root of the Jacobi preconditioner on both sides of M^{-1} (splitting the Jacobi preconditioner into its square roots is to preserve symmetry), instead of improvement on the performance, deterioration happens, as shown in Tables 3.6 and 3.7.

	N_{1D}	2^3	2^4	2^5	2^6	2^7	2^8	2^9	2^{10}
Stretched	Hybrid	48	43	41	38	35	35	34	31
	M^{-1}	31	32	32	32	33	33	33	33
Perturbed	Hybrid	56	51	48	45	42	40	37	36
	M^{-1}	29	34	39	40	42	44	44	45

Table 3.6: h-scaling: $p = 4, c = 3$.

	p	1	2	3	4	5	6	7	8
Stretched	Hybrid	10	17	25	34	40	47	56	66
	M^{-1}	33	33	33	33	33	33	33	33
Perturbed	Hybrid	12	21	29	37	46	54	61	72
	M^{-1}	44	45	44	44	44	44	44	44

Table 3.7: k-scaling: $N_{1D} = 2^9, c = p - 1$.

The reason behind is that M^{-1} already provides a fairly good correction on the deviation from matrix structure. Adding the Jacobi preconditioner on top results in overcorrection on the deviation from matrix structure.

Some adjustment is thus needed. Our approach is to remove the information of basis functions from the Jacobi preconditioner: when assembling the Jacobi preconditioner, we replace the involved basis functions with constant 1, and then rescale with respect to the sizes of their support region.

In Tables 3.8 and 3.9, we show some numerical results regarding the quality of this newly designed hybrid preconditioner. This time, the hybrid preconditioner demonstrates competitive performance even when comparing with $M^{\xi S}$.

Moreover, when it comes to time-dependent nonlinear problems where the mass matrix changes from step to step, we only need to update a diagonal matrix (the modified Jacobi preconditioner) in order to maintain fast convergence speed at each time step. The other component of the hybrid preconditioner, M^{-1}, can be used for all steps as long as the basis functions are unchanged. The assembling cost is thus maintained at low level. Additional numerical results are provided in Appendix A.5.

	N_{1D}	2^3	2^4	2^5	2^6	2^7	2^8	2^9	2^{10}
Stretched	Hybrid	11	9	7	6	5	4	4	3
	$(M^{\xi S})^{-1}$	10	8	6	6	5	4	4	4
Perturbed	Hybrid	26	20	15	12	10	8	7	6
	$(M^{\xi S})^{-1}$	21	16	12	10	8	7	6	5

Table 3.8: h-scaling: $p = 4, c = 3$.

	p	1	2	3	4	5	6	7	8
Stretched	Hybrid	3	4	4	4	4	4	4	4
	$(M^{\xi S})^{-1}$	3	4	4	4	4	4	4	4
Perturbed	Hybrid	5	6	7	7	7	8	8	8
	$(M^{\xi S})^{-1}$	5	5	6	6	6	7	7	7

Table 3.9: k-scaling: $N_{1D} = 2^9, c = p - 1$.

Remark 3.6. *Although only 2D preconditioners, algorithms and numerical examples are presented in Section 3.3, their extensions to 3D case are straightforward.*

3.4 Improving starting points

Not only the quality of the preconditioner affects the performance of the Krylov Subspace method, but also the quality of the starting point. A carefully chosen starting point can reduce the computational cost substantially. In this section, we propose an economic strategy to construct starting points that are close to the solutions.

Still take the complicated geometry case as an example. Suppose we are interested on solution of the following linear system:

$$M^J b^J = \mathcal{F}^J, \tag{3.10}$$

where M^J and $\mathcal{F}^J$ are the mass matrix and right hand side built for the complicated physical domain with basis functions $\{B_i\}_{i=1}^N$. We propose solution of the following linear system:

$$Mb = \mathcal{F} \tag{3.11}$$

as the starting point for linear system (3.10), where M and $\mathcal{F}$ are the mass matrix and right hand side built with the same basis functions $\{B_i\}_{i=1}^N$, but as if the physical domain is identical with the parametrical domain, i.e., a rectangle. Since M possesses the Kronecker product property, solution of linear system (3.11) is easy to obtain.

Motivation behind this proposed starting point is explained in the following. Solution of linear system (3.10) is identical to solution of minimization problem (3.12):

$$\arg\min_{\{b_i^J\}} \int_{\square} (f - \sum_{i=1}^{N} b_i^J B_i)^2 J d_{\square} \tag{3.12}$$

while the solution of linear system (3.11) is identical to the solution of minimization problem (3.13):

$$\arg\min_{\{b_i\}} \int_{\square} (f - \sum_{i=1}^{N} b_i B_i)^2 d_{\square}. \tag{3.13}$$

Solutions of these two minimization problems, (3.12) and (3.13), get closer and closer as the space of basis functions $\{B_i\}_{i=1}^{N}$ gets richer and richer. Therefore, at least when the space of basis functions is rich enough, solution of linear system (3.11) should serve as a good starting point for linear system (3.10). In the following, we refer to this starting point the 'improved starting point'.

We demonstrate some numerical results to verify this claim. Figure 3.5 and 3.6 record the initial residual (in ℓ_2 norm) of linear system (3.10) with the improved starting point. Figure 3.5 corresponds to the stretched rectangle while Figure 3.6 corresponds to the perturbed rectangle shown in Figure 3.2.

Different lines in Figure 3.5 and 3.6 correspond to different types of B-spline basis functions. For instance, P2C1 corresponds to quadratic B-spline basis functions with continuity order 1. We see from both figures that as we refine the mesh, the initial residual indeed decreases as predicted.

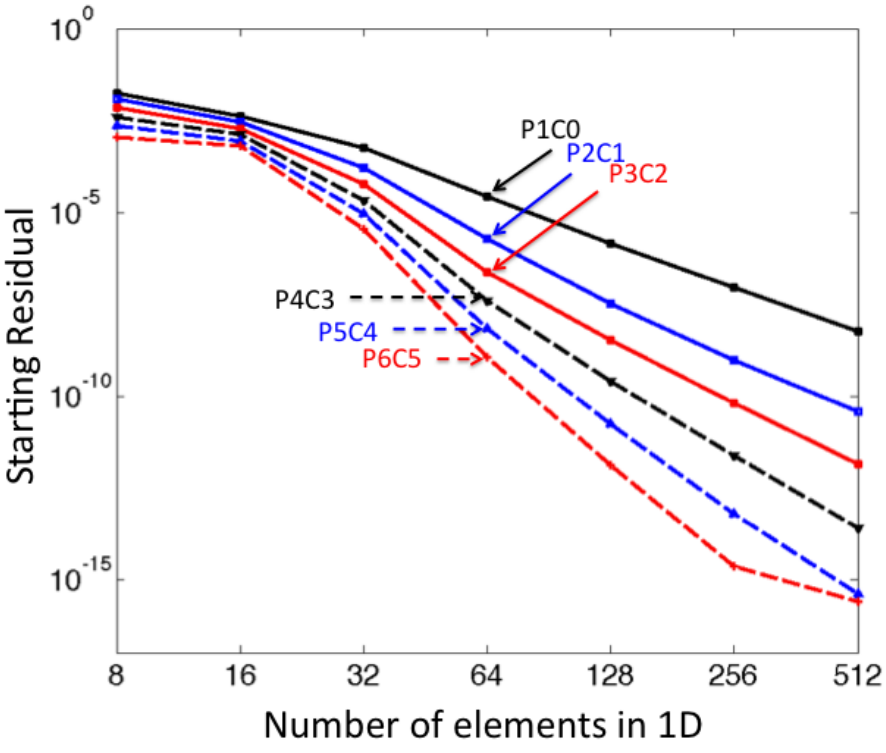


Figure 3.5: Starting residual vs mesh size: stretched rectangle.

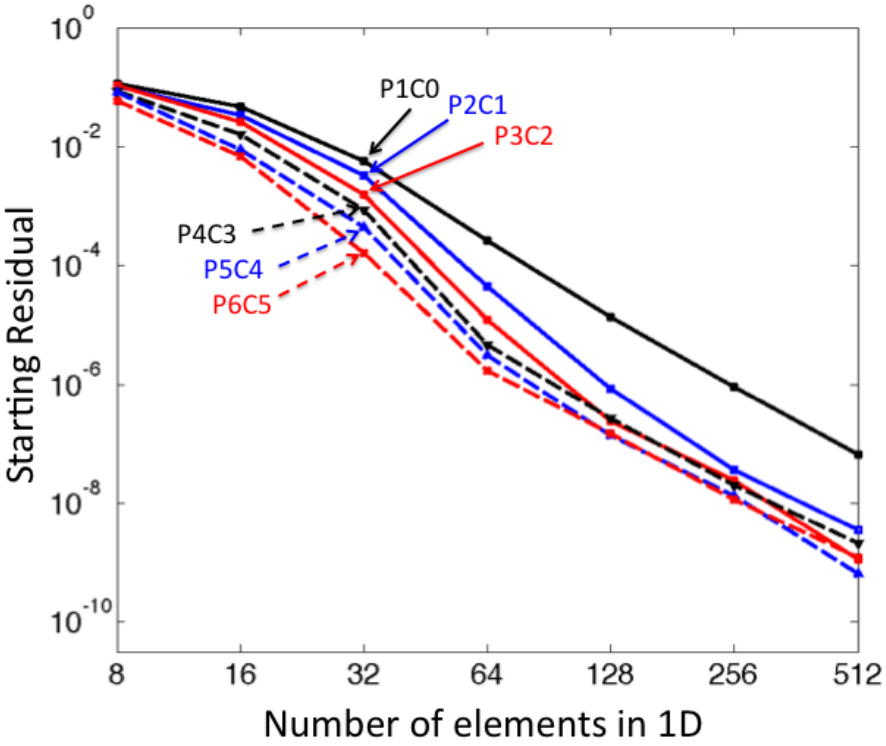


Figure 3.6: Starting residual vs mesh size: perturbed rectangle.

Tables 3.1 and 3.2 record the number of iteration steps required for convergence with the improved starting points, in comparison with zero starting points, for preconditioned conjugate gradient method with the preconditioners M^{-1} and $(M^{\xi S})^{-1}$, respectively.

The iteration process is stopped when relative residual (in ℓ_2 norm) is less than 1e-12. In both tables, polynomial order (p) and continuity order (c) of the basis functions are fixed while mesh size ($h = {}^1/_{N_{1D}}$) is varying.

M^{-1}	N_{1D}	2^3	2^4	2^5	2^6	2^7	2^8	2^9	2^{10}
Stretched	I.S.P.	28	20	8	4	2	1	0	0
	Z.S.P.	31	32	32	32	33	33	33	33
Perturbed	I.S.P.	27	31	29	23	19	16	12	8
	Z.S.P.	29	34	39	40	42	44	44	45

Table 3.10: h-scaling: $p = 4, c = 3$.
I.S.P.: improved starting points; Z.S.P.: zero starting points.

$(M^{\xi S})^{-1}$	N_{1D}	2^3	2^4	2^5	2^6	2^7	2^8	2^9	2^{10}
Stretched	I.S.P.	8	5	4	2	1	1	0	0
	Z.S.P.	10	8	6	6	5	4	4	4
Perturbed	I.S.P.	19	15	9	6	4	3	2	1
	Z.S.P.	21	16	12	10	8	7	6	5

Table 3.11: h-scaling: $p = 4, c = 3$.
I.S.P.: improved starting points; Z.S.P.: zero starting points.

Similar to the complicated geometry case, for the case of complicated coefficient, the resulting linear system $M^\beta b^\beta = \mathcal{F}^\beta$ corresponds to minimization problem

$$\arg\min_{\{b_i^\beta\}} \int_\square (\frac{f}{\beta} - \sum_{i=1}^N b_i^\beta B_i)^2 \beta d_\square.$$

We propose solution of the following minimization problem:

$$\arg\min_{\{b_i\}} \int_\square (\frac{f}{\beta} - \sum_{i=1}^N b_i B_i)^2 d_\square$$

as our 'improved starting point'. For the case of complicated basis functions, the resulting linear system $M^w b^w = \mathcal{F}^w$ corresponds to minimization problem

$$\arg\min_{\{b_i^w\}} \int_\square (fW_N - \sum_{i=1}^N b_i^w B_i)^2 \frac{1}{W_N^2} d_\square.$$

We propose solution of the following minimization problem:

$$\arg\min_{\{b_i\}} \int_\square (fW_N - \sum_{i=1}^N b_i B_i)^2 d_\square$$

as our 'improved starting point'. Moreover, for the situation where more than 1 of the above difficulties are involved, one can easily write down the minimization problems and the corresponding 'improved starting point'.

Chapter 4

Stiffness matrix

Comparing with mass matrices, stiffness matrices are numerically more challenging due to their increasing condition number as the computational mesh gets refined [30, 38]. There are plenty of existing works offering efficient algorithms for solving the Poisson type equations, see for instance, [7, 17, 48, 53, 60, 73, 78]. Here we propose a specific approach by exploiting the Kronecker product structure in the stiffness matrices.

4.1 The ideal case

Let us first focus on the linear algebraic system arising from the stationary state of Model Problem 1. In Section 3.1, we presented Algorithm 1 to invert the mass matrix M. However, for the stiffness matrix $K = K^X + K^Y$ arising from Model Problem 1, this simple algorithm does not work anymore. This is because albeit $K^X = M^y \otimes K^x$ and $K^Y = K^y \otimes M^x$ have the Kronecker product structure individually, their sum does not possess this structure anymore.

Thus we need a more sophisticated strategy to exploit the structures in K^X and K^Y. Early attempts can be found in [8] and [82], for instance. Here we find our strategy by shifting towards iterative methods again. Specifically, we resort to the alternating direction implicit algorithm.

As the name of the algorithm indicates, we split the stiffness matrix K into two parts: K^X and K^Y, and deal with only one of them at a single iteration step. The iterative scheme couples these two parts together and leads us to the convergence of solution with respect to the original matrix K.

4.1.1 Alternating direction implicit method: origin

The Alternating Direction Implicit (ADI) method was proposed in [62] as a numerical method for parabolic and elliptic PDEs. Thorough explanations of the ADI method can also be found in [11, 83, 87, 88, 92]. Here we only sketch the general idea briefly.

For the linear algebraic system:

$$(K^X + K^Y)b = \mathcal{F}, \tag{4.1}$$

the following scheme can be used to solve (4.1) iteratively:

$$(r^{(k)}\Sigma + K^X)b^{(k+\frac{1}{2})} = (r^{(k)}\Sigma - K^Y)b^{(k)} + \mathcal{F}, \tag{4.2a}$$

$$(r^{(k)}\Sigma + K^Y)b^{(k+1)} = (r^{(k)}\Sigma - K^X)b^{(k+\frac{1}{2})} + \mathcal{F}. \tag{4.2b}$$

The motivation behind the development of this iterative scheme is that if K^X and K^Y come from the finite difference discretization of $u_{,xx}$ and $u_{,yy}$, respectively, with the five-point stencil, they can be reformulated as tridiagonal matrices after suitable permutations. An efficient algorithm (the tridiagonal matrix algorithm, also knows as the Thomas algorithm, see [66, 67]) can be applied to invert non-singular tridiagonal matrices efficiently.

Therefore, if adding the extra term $r^{(k)}\Sigma$ does not destroy this tridiagonal structure in K^X and K^Y, (4.2a) and (4.2b) can be solved efficiently. This is why in most early literatures on ADI, Σ is chosen as the identity matrix, or diagonal matrices [88].

Adding $r^{(k)}\Sigma$ can ensure that the matrices to be inverted in (4.2a) and (4.2b) are non-singular, but more importantly, it accelerates the convergence speed.

To see why it is so, we set Σ to the identity matrix I and subtract the following equivalent forms of (4.1):

$$\begin{aligned}(r^{(k)}I + K^X)b &= (r^{(k)}I - K^Y)b + \mathcal{F},\\ (r^{(k)}I + K^Y)b &= (r^{(k)}I - K^X)b + \mathcal{F},\end{aligned}$$

from (4.2a) and (4.2b), respectively, obtaining:

$$(r^{(k)}I + K^X)e^{(k+\frac{1}{2})} = (r^{(k)}I - K^Y)e^{(k)}, \tag{4.3a}$$

$$(r^{(k)}I + K^Y)e^{(k+1)} = (r^{(k)}I - K^X)e^{(k+\frac{1}{2})}, \tag{4.3b}$$

where $e^{(i)} = b^{(i)} - b$ for $i = k$, $k + \frac{1}{2}$ and $k + 1$.

Substituting $e^{(k+\frac{1}{2})}$ from (4.3a) into (4.3b), we get

$$e^{(k+1)} = \left[(r^{(k)}I + K^Y)^{-1}(r^{(k)}I - K^X)(r^{(k)}I + K^X)^{-1}(r^{(k)}I - K^Y)\right] e^{(k)}. \tag{4.4}$$

Defining $\mathcal{P}^{(k)} = (r^{(k)}I + K^Y)^{-1}(r^{(k)}I - K^X)(r^{(k)}I + K^X)^{-1}(r^{(k)}I - K^Y)$, we have

$$e^{(k+1)} = \left(\prod_{j=0}^{k} \mathcal{P}^{(j)}\right) e^{(0)}, \tag{4.5}$$

where $\left(\prod_{j=0}^{k} \mathcal{P}^{(j)}\right) = \mathcal{P}^{(k)} \cdots \mathcal{P}^{(0)}$.

Now let us further simplify the problem by making the following assumption: there exists a set of orthonormal eigenvectors $\{v_i\}_{i=1}^{N}$ such that $K^X v_i = \lambda_i^X v_i$ and $K^Y v_i = \lambda_i^Y v_i$, where $N \times N$ is the size of K^X and K^Y. Under this assumption, we have:

Theorem 4.1. *If K^X and K^Y are $N \times N$ symmetric positive definite matrices that*

share a same set of orthonormal eigenvectors $\{v_i\}_{i=1}^N$ *with corresponding eigenvalues* $\{\lambda_i^X\}_{i=1}^N$ *and* $\{\lambda_i^Y\}_{i=1}^N$*, respectively, and* $\{r^{(j)}\}_{j=0}^{k+1}$ *is a set of positive real numbers, then*

$$\left\|e^{(k+1)}\right\|_2 \leq \left\|\prod_{j=0}^{k}\mathcal{P}^{(j)}\right\|_2 \cdot \left\|e^{(0)}\right\|_2 \tag{4.6}$$

with

$$\left\|\prod_{j=0}^{k}\mathcal{P}^{(j)}\right\|_2 = \max_{1\leq i\leq N}\left\{\prod_{j=0}^{k}\left|\frac{r^{(j)}-\lambda_i^X}{r^{(j)}+\lambda_i^X}\right|\cdot\left|\frac{r^{(j)}-\lambda_i^Y}{r^{(j)}+\lambda_i^Y}\right|\right\} < 1 \tag{4.7}$$

and

$$\left\|\prod_{j=0}^{k+1}\mathcal{P}^{(j)}\right\|_2 < \left\|\prod_{j=0}^{k}\mathcal{P}^{(j)}\right\|_2. \tag{4.8}$$

The requirement on K^X and K^Y in Theorem 4.1 can be relaxed so that only one of them is symmetric positive definite while the other can be symmetric positive semi-definite. Allowing this relaxation is important for the case when Neumann boundary conditions are imposed, for instance, on two parallel edges of a rectangular domain.

For proof of Theorem 4.1 and further discussions, please refer to [83]. We also include two comments from [83] here:

1. The requirement that K^X and K^Y share a same set of eigenvectors is not needed for (4.8);

2. If either $\{\lambda_i^X\}_{i=1}^N$ or $\{\lambda_i^X\}_{i=1}^N$ is known *a priori*, a direct method for (4.1) can be designed by letting $r^{(j)}$ going through all elements of the known eigenvalue set.

(4.6) and (4.7) motivates the investigate on the following min - max problem in order to accelerate the convergence:

$$\{s^{(j)}\}_{j=0}^k = \underset{\{r^{(j)}\}_{j=0}^k}{\arg\min}\left\{\max_{1\leq i\leq N}\left\{\prod_{j=0}^{k}\left|\frac{r^{(j)}-\lambda_i^X}{r^{(j)}+\lambda_i^X}\right|\cdot\left|\frac{r^{(j)}-\lambda_i^Y}{r^{(j)}+\lambda_i^Y}\right|\right\}\right\}. \tag{4.9}$$

However, (4.9) is not practical since it is hard to obtain all the eigenvalues $\{\lambda_i^X\}_{i=1}^N$

and $\{\lambda_i^X\}_{i=1}^N$. Instead, the following relaxed problem:

$$\{s^{(j)}\}_{j=0}^k = \underset{\{r^{(j)}\}_{j=0}^k}{\arg\min} \left\{ \max_{0<\alpha\leq x\leq\beta} \left\{ \left| \prod_{j=0}^k \frac{r^{(j)}-x}{r^{(j)}+x} \right| \right\} \right\} \tag{4.10}$$

is thoroughly discussed in [85] under the assumption that lower bound α and upper bound β for $\{\lambda_i^X\}_{i=1}^N$ and $\{\lambda_i^X\}_{i=1}^N$ can be found such that $0 < \alpha \leq \lambda_i^X, \lambda_i^Y \leq \beta, 1 \leq i \leq N$. In the sense of (4.10), a set of optimal parameters $\{s^{(j)}\}_{j=0}^k$ can be selected, based on the following theorem (again from [83]):

Theorem 4.2. *A unique set of distinct parameters* $\{s^{(j)}\}_{j=0}^k$ *can be found as the solution of the* min - max *problem* (4.10).

For proof of Theorem 4.2, please consult [83], or [85, 87] for direct information and [1, 19] for background knowledge on approximation theory.

More importantly, [85] gives a theoretically sound and numerically elegant algorithm to find these optimal parameters for the special case where $k = 2^i$ with i a nonnegative integer. [83] is also a good reference for understanding the algorithm and the effects of these optimal parameters.

In reality, the assumption that K^X and K^Y share a same set of eigenvectors is rarely satisfied, see [83]. There are various attempts on extending the theory and practice of ADI to more general cases, among which we find the 'compound iteration' idea presented in [86] particularly appealing. In [86], ADI is used for the 'inner iteration' to approximately invert an approximate matrix of K that satisfies this assumption. This iteration scheme serves as the preconditioner while the conjugate gradient method is used for the 'outer iteration' to solve the preconditioned linear algebraic system. It is worth mentioning that this compound iteration idea might be the first attempt to use an iterative scheme as a preconditioner for the conjugate gradient method, see [45].

4.1.2 Alternating direction implicit method: generalization

A brief introduction to ADI has been given in Section 4.1.1, focusing on the linear algebraic systems arising from finite difference discretization with a five point stencil.

However, the linear algebraic systems that we are dealing with are slightly different from those in the finite difference discretization case. Coming from finite element discretization, K^X and K^Y are generally not tridiagonal matrices (even after permutations), thus the Thomas algorithm is not applicable to these matrices.

Nevertheless, K^X and K^Y possess the Kronecker product structure if tensor product basis functions are used in the finite element discretization. Thus the inverses of $K^X = M^y \otimes K^x$ and $K^Y = K^x \otimes M^y$ (both are assumed to be symmetric positive-definite here) can still be applied efficiently with Algorithm 1.

Now, we need an appropriate matrix Σ to accelerate the convergence speed (also ensure the non-singularity), but without destroying the Kronecker product structure. Identity matrix is not an option anymore. Luckily, the mass matrix $M = M^y \otimes M^x$ is ready to be summoned. Preservation of the Kronecker product structure can be easily verified using the following properties of Kronecker product, see [75]:

$$
\begin{aligned}
&(cA) \otimes B = c(A \otimes B) = A \otimes (cB), \\
&(A + B) \otimes (C + D) = A \otimes C + A \otimes D + B \otimes C + B \otimes D,
\end{aligned}
$$

where c is a scalar while A, B, C, D are matrices with compatible sizes.

With the ability of solving linear algebraic systems (4.2a) and (4.2b) efficiently, the following question is: how to choose a set of parameters $\{r^{(k)}\}$ to accelerate the convergence speed. Fortunately, all the theories developed for the finite difference case in Section 4.1.1 are still applicable after a slight modification on the measurement.

Under the same procedure as in Section 4.1.1, but setting Σ to the mass matrix

M in (4.2) leads us to the following relationship:

$$e^{(k+1)} = \left[(r^{(k)}M + K^Y)^{-1}(r^{(k)}M - K^X)(r^{(k)}M + K^X)^{-1}(r^{(k)}M - K^Y)\right] e^{(k)}. \quad (4.11)$$

Adapting the definition of $\mathcal{P}^{(k)}$ to

$$\mathcal{P}^{(k)} = (r^{(k)}M + K^Y)^{-1}(r^{(k)}M - K^X)(r^{(k)}M + K^X)^{-1}(r^{(k)}M - K^Y),$$

we can still write:

$$e^{(k+1)} = \left(\prod_{j=0}^{k} \mathcal{P}^{(j)}\right) e^{(0)}. \quad (4.12)$$

To determine a set of acceleration parameters $\{r^{(k)}\}$ for the case $\Sigma = M$, we assume that there exists a set of M-orthonormal generalized eigenvectors $\{v_i\}_{i=1}^N$ such that

$$K^X v_i = \lambda_i^X M v_i, \quad (4.13a)$$

$$K^Y v_i = \lambda_i^Y M v_i, \quad (4.13b)$$

where $N \times N$ is the size of K^X, K^Y and M. For two vectors v_i and v_j to be M-orthonormal, we mean:

$$(v_i)^T M v_j = \begin{cases} 1 & \text{if } i = j, \\ 0 & \text{if } i \neq j. \end{cases}$$

For matrices arising from the model problems: $M = M^y \otimes M^x$, $K^X = M^y \otimes K^x$ and $K^Y = K^y \otimes M^x$, the above assumption is actually satisfied. We demonstrate this in a constructive manner:

Lemma 4.1. *Assume $K^x V^x = M^x V^x D^x$ and $K^y V^y = M^y V^y D^y$ where D^x and D^y are diagonal, then for matrices $M = M^y \otimes M^x$, $K^X = M^y \otimes K^x$ and $K^Y = K^y \otimes M^x$,*

we have the following relationships:

$$K^X V = MVD^X,$$
$$K^Y V = MVD^Y,$$

with $V = V^y \otimes V^x$, $D^X = I^y \otimes D^x$ *and* $D^Y = D^y \otimes I^x$ *where* I^x *and* I^y *stand for the identity matrices with the same sizes as* D^x *and* D^y, *respectively.*

If we further have

$$(V^x)^T M^x V^x = I^x, \tag{4.14a}$$
$$(V^y)^T M^y V^y = I^y, \tag{4.14b}$$

then $V^T MV = I^y \otimes I^x$ *is also true.*

Proof.

$$\begin{aligned}
K^X V &= (M^y \otimes K^x)(V^y \otimes V^x) \\
&= (M^y V^y) \otimes (K^x V^x) \\
&= (M^y V^y I^y) \otimes (M^x V^x D^x) \\
&= \big((M^y V^y) \otimes (M^x V^x)\big)(I^y \otimes D^x) \\
&= (M^y \otimes M^x)(V^y \otimes V^x)(I^y \otimes D^x) \\
&= MVD^X;
\end{aligned}$$

$$\begin{aligned}
K^Y V &= (K^y \otimes M^x)(V^y \otimes V^x) \\
&= (K^y V^y) \otimes (M^x V^x) \\
&= (M^y V^y D^y) \otimes (M^x V^x I^x) \\
&= \big((M^y V^y) \otimes (M^x V^x)\big)(D^y \otimes I^x) \\
&= (M^y \otimes M^x)(V^y \otimes V^x)(D^y \otimes I^x) \\
&= MVD^Y;
\end{aligned}$$

$$\begin{aligned} V^T M V &= (V^y \otimes V^x)^T (M^y \otimes M^x)(V^y \otimes V^x) \\ &= \left((V^y)^T \otimes (V^x)^T\right)\left((M^y V^y) \otimes (M^x V^x)\right) \\ &= \left((V^y)^T M^y V^y\right) \otimes \left((V^x)^T M^x V^x\right) \\ &= I^y \otimes I^x. \end{aligned}$$

□

To clarify, in the following, when using the terminology 'generalized eigenvalue/eigenvector' of K^X or K^Y, we refer to the generalized eigenvalue/eigenvector defined by (4.13a) or (4.13b), respectively. Similarly, the generalized eigenvalue/eigenvector of K^x or K^y corresponds to the eigen-decomposition $K^x V^x = M^x V^x D^x$ or $K^y V^y = M^y V^y D^y$, respectively.

Lemma 4.1 also tells us that the generalized eigenvalues of K^X are repeated: K^X has the same set of distinct generalized eigenvalues as K^x. The analogical result holds for K^Y and K^y.

Moreover, coming from the finite element discretization, K^x and K^y are symmetric positive semi-definite while M^x and M^y are symmetric positive definite. This guarantees that all the diagonal entries of D^x and D^y are non-negative real numbers and there exist full rank matrices V^x and V^y such that (4.14a) and (4.14b) are satisfied, see [46, 71]. Additionally, this also tells us that columns of V form a basis for vector space R^N. When this happens, we refer to V as the generalized eigen-matrix.

With all these properties established above, we can now proceed to analyze the relationships between errors at different iteration steps and the selection of acceleration parameters. We first introduce the M-norm:

Definition 4.1. *Given a symmetric positive definite matrix M, the M-norm of a vector v with compatible size is defined as:*

$$\|v\|_M = \left(v^T M v\right)^{\frac{1}{2}}.$$

For matrix A with compatible size, the induced matrix M-norm is defined as:

$$\|A\|_M = \sup_{v \neq 0} \frac{\|Av\|_M}{\|v\|_M}.$$

For brevity, we omit the verifications of whether the above definitions meet the requirements of a norm. See [77] if one has concerns.

The above definitions immediately lead us to the following inequality, analogous to (4.6):

$$\left\|e^{(k+1)}\right\|_M \leq \left\|\prod_{j=0}^{k} \mathcal{P}^{(j)}\right\|_M \cdot \left\|e^{(0)}\right\|_M. \tag{4.15}$$

Now we can focus on the term $\left\|\prod_{j=0}^{k} \mathcal{P}^{(j)}\right\|_M$. For any generalized eigenvector v_i (a column of V), we have the following relations that can be easily verified:

$$(r^{(k)}M - K^X)v_i = (r^{(k)} - \lambda_i^X)Mv_i,$$
$$(r^{(k)}M - K^Y)v_i = (r^{(k)} - \lambda_i^Y)Mv_i,$$

and

$$\frac{1}{(r^{(k)} + \lambda_i^X)}v_i = (r^{(k)}M + K^X)^{-1}Mv_i,$$
$$\frac{1}{(r^{(k)} + \lambda_i^Y)}v_i = (r^{(k)}M + K^Y)^{-1}Mv_i,$$

given that $r^{(k)}$ is a positive real number. These relations enable us to look through the effect of applying $\mathcal{P}^{(k)}$ on a generalized eigenvector v_i:

$$\mathcal{P}^{(k)}v_i = \left(\frac{r^{(k)} - \lambda_i^X}{r^{(k)} + \lambda_i^X}\right) \cdot \left(\frac{r^{(k)} - \lambda_i^Y}{r^{(k)} + \lambda_i^Y}\right) v_i; \tag{4.16}$$

and similarly for $\prod_{j=0}^{k} \mathcal{P}^{(j)}$:

$$\left(\prod_{j=0}^{k} \mathcal{P}^{(j)}\right) v_i = \left(\prod_{j=0}^{k} \left(\frac{r^{(j)} - \lambda_i^X}{r^{(j)} + \lambda_i^X}\right) \cdot \left(\frac{r^{(j)} - \lambda_i^Y}{r^{(j)} + \lambda_i^Y}\right)\right) v_i. \tag{4.17}$$

Since $\{v_i\}_{i=1}^N$ are M-orthonormal with each other, we have the following relationships, analogous to (4.7) and (4.8):

$$\left\|\prod_{j=0}^{k}\mathcal{P}^{(j)}\right\|_M = \max_{1\le i\le N}\left\{\prod_{j=0}^{k}\left|\frac{r^{(j)}-\lambda_i^X}{r^{(j)}+\lambda_i^X}\right|\cdot\left|\frac{r^{(j)}-\lambda_i^Y}{r^{(j)}+\lambda_i^Y}\right|\right\} < 1 \tag{4.18}$$

and

$$\left\|\prod_{j=0}^{k+1}\mathcal{P}^{(j+1)}\right\|_M < \left\|\prod_{j=0}^{k}\mathcal{P}^{(j)}\right\|_M. \tag{4.19}$$

We summarize the above results in the following theorem:

Theorem 4.3. *If K^X, K^Y and M are $N\times N$ symmetric positive definite matrices and there exists a generalized eigen-matrix V such that $K^XV = MVD^X$ and $K^YV = MVD^Y$ hold, then*

$$\left\|e^{(k+1)}\right\|_M \le \left\|\prod_{j=0}^{k}\mathcal{P}^{(j)}\right\|_M \cdot \left\|e^{(0)}\right\|_M \tag{4.20}$$

with

$$\left\|\prod_{j=0}^{k}\mathcal{P}^{(j)}\right\|_M = \max_{1\le i\le N}\left\{\prod_{j=0}^{k}\left|\frac{r^{(j)}-\lambda_i^X}{r^{(j)}+\lambda_i^X}\right|\cdot\left|\frac{r^{(j)}-\lambda_i^Y}{r^{(j)}+\lambda_i^Y}\right|\right\} < 1 \tag{4.21}$$

and

$$\left\|\prod_{j=0}^{k+1}\mathcal{P}^{(j+1)}\right\|_M < \left\|\prod_{j=0}^{k}\mathcal{P}^{(j)}\right\|_M. \tag{4.22}$$

Once again, the requirement on K^X and K^Y in Theorem 4.3 can be relaxed so that only one of them is symmetric positive definite while the other can be symmetric positive semi-definite. The same as for Theorem 4.1, we have the following two analogical comments for Theorem 4.3:

1. The requirement that K^X and K^Y share a same set of generalized eigenvectors is not needed for (4.22);

2. If either $\{\lambda_i^X\}_{i=1}^N$ or $\{\lambda_i^X\}_{i=1}^N$ is known *a priori*, a direct method for (4.1) can be designed by letting $r^{(j)}$ going through all elements of the known generalized

eigenvalue set. Based on Lemma 4.1, $r^{(j)}$ only need to go through all the generalized eigenvalues of K^x or K^y.

Theorem 4.3 also shows that min-max problem (4.9) and its relaxed version: (4.10) can still provide valuable information on selecting acceleration parameters for the case $\Sigma = M$. Specifically, the set of acceleration parameters provided by (4.10) is still optimal, but in terms of error reduction under the M-norm.

Remark 4.1. *For linear algebraic system* $\left(\delta M + K^X + K^Y\right) b = \mathcal{F}$, $\delta > 0$, *which arises, possibly, from a finite element discretization in space coupled with an implicit time discretization for Model Problem 1, or from a reaction diffusion equation, the above ADI algorithm, along with the associated theories, is still applicable. The only adjustment needed is a shift of the acceleration parameters according to* δ.

Remark 4.2. *The generalization of ADI presented in Section 4.1.2 has already been discussed in [28], where Tensor Product Generalized Alternating Direction Implicit (TPGADI) iterative method is proposed for linear algebraic systems having the form* $(A_1 \otimes B_2 + A_2 \otimes B_1)C = F$. *There are only minor differences in between [28] and what we presented in 4.1.2:*

1. *We adopt a slightly different approach for the derivation and proof of the method, namely, we use the generalized eigen-pairs while [28] uses eigen-pairs of the transformed matrices* $B_1^{-1}A_1$ *and* $B_2^{-1}A_2$;

2. *We obtain the relationships between errors at different steps in terms of vector norms (Theorem 4.3) while [28] obtains the relationships in a component-wise sense;*

3. *Selection of acceleration parameters is not discussed in [28] while we discuss it as a natural following up of Theorem 4.3 and show that the solution of* min-max *problem* (4.10) *is still optimal in the* M*-norm sense.*

In the next section, we will use the solution technique developed in Section 4.1.2 for the ideal Model Problem 1 as a preconditioning technique for more general cases.

But first, let us show some numerical results for the ideal case. We apply the ADI algorithm to linear algebraic systems $(M + K^X + K^Y)b = \mathcal{F}$ discretized with 32×32, 128×128 and 512×512 elements meshes with tensor product linear basis functions for various numbers of optimal acceleration parameters, denoted as k.

The actual error reduction rates (columns tagged as 'observed') and their upper bounds provided from [85] (columns tagged as 'predicted') for these optimal acceleration parameters are recorded in Table 4.1. Appendix B.1 provides a brief description of this upper bound. If one has further interest, a thorough discussion can also be found in [83].

	32×32		128×128		512×512	
k	observed	predicted	observed	predicted	observed	predicted
1	9.75E-01	9.75E-01	9.94E-01	9.94E-01	9.98E-01	9.98E-01
2	6.35E-01	6.35E-01	7.98E-01	7.98E-01	8.93E-01	8.93E-01
4	1.28E-01	1.28E-01	2.48E-01	2.48E-01	3.80E-01	3.80E-01
8	4.16E-03	4.16E-03	1.58E-02	1.58E-02	3.89E-02	3.89E-02
16	4.33E-06	4.33E-06	6.25E-05	6.25E-05	3.78E-04	3.78E-04
32	5.03E-12	4.68E-12	9.74E-10	9.75E-10	3.56E-08	3.57E-08
64	3.03E-13	5.49E-24	9.69E-13	2.38E-19	1.10E-10	3.19E-16

Table 4.1: Error reduction rates and upper bounds.
Zero initial points and random right hand sides are used.

From Table 4.1, we can see that upper bound for the error reduction rate provided from [85] is extremely tight, thus can serve as a good indicator for deciding how many optimal acceleration parameters shall be selected, given a desired error reduction rate.

4.2 Orthotropic inhomogeneous coefficients

Model Problem 1 is a simplification for more general cases, for instance:

$$u_t = \nabla \cdot \big(\boldsymbol{\kappa}(x, y)\nabla u(x, y)\big) + f(x, y), \tag{4.23}$$

where $\boldsymbol{\kappa}(x,y)$ is a 2×2 tensor:

$$\boldsymbol{\kappa}(x,y) = \begin{bmatrix} \kappa_{11}(x,y) & \kappa_{12}(x,y) \\ \kappa_{21}(x,y) & \kappa_{22}(x,y) \end{bmatrix}$$

with $\kappa_{12} = \kappa_{21}$. In this section, we concern ourselves with the static solution of (4.23), governed by

$$-\nabla\cdot\big(\boldsymbol{\kappa}(x,y)\nabla u(x,y)\big) = f(x,y), \tag{4.24}$$

with orthotropic coefficients, i.e., $\kappa_{12} = \kappa_{21} = 0$.

Discretized with the finite element method using tensor product basis functions $B^y \otimes B^x$, (4.24) leads to the following linear algebraic system:

$$(\tilde{K}^X + \tilde{K}^Y)b = \mathcal{F}, \tag{4.25}$$

where

$$\tilde{K}^X = \int_{\square} \left(B^y (B^y)^T\right) \otimes \left(B^x_{,x} (B^x_{,x})^T\right) \kappa_{11} d_{\square}, \tag{4.26a}$$

$$\tilde{K}^Y = \int_{\square} \left(B^y_{,y} (B^y_{,y})^T\right) \otimes \left(B^x (B^x)^T\right) \kappa_{22} d_{\square}. \tag{4.26b}$$

Similar as the term J in the mass matrix case (see Section 3.3), κ_{11} and κ_{22} are now the coupling terms that prevent splitting the 2D integrals. Thus $\tilde{K}^X$ and $\tilde{K}^Y$ do not possess the Kronecker product structure and the ADI method shown in Section 4.1.2 is no longer applicable. Once again, we resort to iterative methods and preconditioning.

4.2.1 The simplest choice: $(K^X + K^Y)^{-1}$

The simplest idea would be: Construct K^X and K^Y from (4.26a) and (4.26b), respectively, by replacing κ_{11} and κ_{22} with constants; Apply $(K^X + K^Y)^{-1}$ with ADI

method (the inner iteration) to precondition linear system (4.25); Solve the preconditioned system with Krylov subspace methods (the outer iteration). This is precisely the idea of compound iteration that has been proposed in [86].

A clarification of terminology is necessary here. $(K^X + K^Y)^{-1}$ is actually not the preconditioner in the strict sense; instead, the approximate inverse of $(K^X + K^Y)$ induced by ADI iterations is the preconditioner. However, for simplicity, we still call $(K^X + K^Y)^{-1}$ the preconditioner if there is no ambiguity. Besides, we may also refer to the ADI iterations (or the inner iterations in general) as the preconditioner.

There are still several questions left regarding this compound iteration idea:

1. What are we really using as the preconditioner and does it have a matrix form?
2. If so, is the matrix symmetric positive definite such that conjugate gradient method can be applied for the outer iteration?
3. How many inner iteration steps are needed?

Let us answer these questions one by one in the following.

1. Matrix form of the preconditioner. Recall the relationship between errors at different ADI iteration steps, as shown in (4.12):

$$e^{(k+1)} = \left(\prod_{j=0}^{k} \mathcal{P}^{(j)} \right) e^{(0)}.$$

Denote by b the true solution of $(K^X + K^Y)b = \mathcal{F}$, we have:

$$\begin{aligned} e^{(0)} &= b^{(0)} - b, \\ e^{(k+1)} &= b^{(k+1)} - b. \end{aligned}$$

Substituting these two equations to (4.12) gives that:

$$b^{(k+1)} = \left(I - \left(\prod_{j=0}^{k} \mathcal{P}^{(j)} \right) \right) b + \left(\prod_{j=0}^{k} \mathcal{P}^{(j)} \right) b^{(0)}.$$

If we start with zero initial guess for the inner iteration, i.e., $b^{(0)} = 0$, the above relation can be simplified to:

$$b^{(k+1)} = \left(I - \left(\prod_{j=0}^{k} \mathcal{P}^{(j)} \right) \right) b. \tag{4.27}$$

Substituting the true solution $b = (K^X + K^Y)^{-1}\mathcal{F}$ into (4.27) leads to:

$$b^{(k+1)} = \left(I - \left(\prod_{j=0}^{k} \mathcal{P}^{(j)} \right) \right) (K^X + K^Y)^{-1}\mathcal{F}. \tag{4.28}$$

Now it is clear that the matrix form of the preconditioner can be written as:

$$\left(I - \left(\prod_{j=0}^{k} \mathcal{P}^{(j)} \right) \right) (K^X + K^Y)^{-1}. \tag{4.29}$$

If we fix the number of inner iteration steps k for each outer iteration step, this preconditioner is fixed as well, thus suitable for Krylov subspace methods. Relevant discussions can be found in [91], where the concept of 'linear preconditioner' is introduced. With a fixed number of inner iteration steps and zero initial guess for each compound of inner iterations, (4.29) is a linear preconditioner.

2. Symmetry and positive definiteness of the preconditioner. Now it comes the question of choosing a method for the outer iteration. Since matrix $(\tilde{K}^X + \tilde{K}^Y)$ is symmetric positive definite (SPD), the conjugate gradient (CG) method is the most desirable one. However, CG requires the preconditioner to be SPD as well, which is yet unclear for this inner iteration preconditioner.

Recall the effect of applying $\left(\prod_{j=0}^{k} \mathcal{P}^{(j)} \right)$ on a generalized eigenvector v_i, as shown in (4.17):

$$\left(\prod_{j=0}^{k} \mathcal{P}^{(j)} \right) v_i = \left(\prod_{j=0}^{k} \left(\frac{r^{(j)} - \lambda_i^X}{r^{(j)} + \lambda_i^X} \right) \cdot \left(\frac{r^{(j)} - \lambda_i^Y}{r^{(j)} + \lambda_i^Y} \right) \right) v_i.$$

Therefore, v_i is an eigenvector of $\left(\prod_{j=0}^{k}\mathcal{P}^{(j)}\right)$. Denote by $D_{\mathcal{P}}$ the diagonal matrix with its ith diagonal entry the eigenvalue of $\left(\prod_{j=0}^{k}\mathcal{P}^{(j)}\right)$ corresponding to v_i, we have:

$$\left(\prod_{j=0}^{k}\mathcal{P}^{(j)}\right)V = VD_{\mathcal{P}}. \tag{4.30}$$

Similarly, we can write:

$$\left(K^X + K^Y\right)V = MVD \tag{4.31}$$

where $D = (I^y \otimes D^x + D^y \otimes I^x)$ is also diagonal.

We can rewrite (4.30) and (4.31), respectively, as

$$\left(I - \left(\prod_{j=0}^{k}\mathcal{P}^{(j)}\right)\right) = V\left(I - D_{\mathcal{P}}\right)V^{-1}$$

and

$$\left(K^X + K^Y\right)^{-1} = VD^{-1}V^{-1}M^{-1},$$

which together lead us to:

$$\left(I - \left(\prod_{j=0}^{k}\mathcal{P}^{(j)}\right)\right)\left(K^X + K^Y\right)^{-1} = V\left(\left(I - D_{\mathcal{P}}\right)D^{-1}\right)(MV)^{-1}.$$

Recall the relation $V^TMV = I$ from Lemma 4.1. We have:

$$\left(I - \left(\prod_{j=0}^{k}\mathcal{P}^{(j)}\right)\right)\left(K^X + K^Y\right)^{-1} = V\left(\left(I - D_{\mathcal{P}}\right)D^{-1}\right)V^T. \tag{4.32}$$

From (4.32), it is clear that if all the diagonal entries of $(I - D_{\mathcal{P}})$ and D are positive, the preconditioner, as shown in (4.29), is SPD. Since all the acceleration parameters are positive, the above assumption is actually satisfied and (4.29) is indeed SPD.

To sum up, the preconditioner obtained from applying a fixed number of ADI iterations with zero initial guess to $\left(K^X + K^Y\right)^{-1}$ is symmetric positive definite, thus can be combined with the conjugate gradient method.

3. Number of inner iteration steps. Yet to be answered is the number of inner iteration steps. Apparently, $(K^X + K^Y)$ is just an approximation of $(\tilde{K}^X + \tilde{K}^Y)$. Applying its inverse super-accurately will not benefit us much on eliminating the 'outer-error'. Rather, only an approximate inverse is needed.

Our experience is that a number of ADI iterations that can achieve error reduction rate around $1e$-3 is enough for most cases and adding more inner iteration steps can hardly help reducing the number of outer iterations.

Given matrix $(K^X + K^Y)$, we go through the following procedures to determine the number of inner iteration steps: Estimate its smallest and largest generalized eigenvalues and set them as the lower bound α and upper bound β in min-max problem (4.10), respectively; Choose the number of inner iteration steps such that the upper bound of error reduction rate associated with the solution of (4.10) is smaller than $1e$-3.

4.2.2 Partial inclusion of coefficient variations in the preconditioners

Now let us examine a more sophisticated idea. Similar as in the mass matrix case (see Section 3.3.2), we can construct approximations of $\tilde{K}^X$ and $\tilde{K}^Y$ by taking advantage of the local support property of the basis functions, as shown in the following:

$$\tilde{K}^X \approx \hat{K}^X = \begin{bmatrix} M^y(1,:) \\ \vdots \\ zeros(1, N_y) \end{bmatrix} \otimes K_1^x + \cdots + \begin{bmatrix} zeros(1, N_y) \\ \vdots \\ M^y(N_y,:) \end{bmatrix} \otimes K_{N_y}^x, \qquad (4.33)$$

and

$$\tilde{K}^Y \approx \hat{K}^Y = K_1^y \otimes \begin{bmatrix} M^x(1,:) \\ \vdots \\ zeros(1, N_x) \end{bmatrix} + \cdots + K_{N_x}^y \otimes \begin{bmatrix} zeros(1, N_x) \\ \vdots \\ M^x(N_x,:) \end{bmatrix}, \quad (4.34)$$

where K_i^x, $i = 1, \ldots, N_y$, each contains partial information of κ_{11} while K_j^y, $j = 1, \ldots, N_x$, each contains partial information of κ_{22}.

An inner iteration preconditioner for $\left(\tilde{K}^X + \tilde{K}^Y\right)$ can be constructed by applying the compound iteration idea to its approximation: $\left(\hat{K}^X + \hat{K}^Y\right)^{-1}$. However, due to the asymmetry of $\left(\hat{K}^X + \hat{K}^Y\right)$, the induced inner iteration preconditioner is also asymmetric, thus cannot be combined with CG. We need to use some other Krylov subspace method, for instance, the generalized minimal residual method, or to symmetrize the preconditioner. We choose the second path due to the efficiency of CG on SPD matrices.

It takes two steps to symmetrize the preconditioner. First, similar as in the mass matrix case (see Section 3.3.2), we obtain the symmetrized approximations of $\hat{K}^X$ and $\hat{K}^Y$, with the help of the Cholesky factors of M^y and M^x, respectively and denoted by $\bar{K}^X$ and $\bar{K}^Y$, respectively. Second, we want to ensure that the preconditioner, induced from the symmetrized approximation $\left(\bar{K}^X + \bar{K}^Y\right)$ by ADI iterations, still possesses the symmetry. For this purpose, given a set of selected acceleration parameters, we apply the cycle of ADI iterations twice, with a reversed order in the second time.

More details regarding the mentioned preconditioner and proof of its symmetry and positive definiteness can be found in Appendix B.2.

Moreover, the selection of acceleration parameters becomes a question again since the condition in Theorem 4.3 that $\bar{K}^X$ and $\bar{K}^Y$ share the same set of generalized eigenvectors is no longer satisfied. Besides, estimation of the smallest and largest generalized eigenvalues also becomes tricky.

However, what we need is merely to apply $\left(\bar{K}^X + \bar{K}^Y\right)^{-1}$ approximately. Certain amount of deviation from the dogma shall not undermine the hope completely. In this case, we estimate the smallest and largest generalized eigenvalues of K^X and K^Y, scaled by the magnitude of κ_{11} and κ_{22}, respectively, and then choose the lower bound α and upper bound β for min-max problem (4.10) accordingly. The corresponding solution of (4.10) is selected as the set of acceleration parameters.

4.2.3 Numerical results

In this section, we show some numerical results to demonstrate the performances regarding the preconditioners discussed in Sections 4.2.1 and 4.2.2. We apply the compound iterations to linear algebraic system

$$(\tilde{K}^X + \tilde{K}^Y)b = \mathcal{F}$$

discretized from (4.24) with full Dirichlet boundary conditions for different orthotropic coefficients $\boldsymbol{\kappa}$, as shown in Figures 4.1 - 4.3; different mesh sizes $h = 1/N_{1D}$, as shown in Tables 4.2, 4.4 and 4.6; and different numbers of inner iteration steps (denoted by N_{inner}), as shown in Tables 4.3, 4.5 and 4.7.

Exact formulae for those coefficients shown in Figures 4.1 - 4.3 can be found in Appendix B.3.

Tables 4.2, 4.4 and 4.6 record the numbers of outer iteration steps required for convergence with respect to different mesh sizes. Two choices of inner iteration preconditioners, induced from $(K^X + K^Y)$ and $(\bar{K}^X + \bar{K}^Y)$, respectively, are tested. To illustrate better the behaviors of these two preconditioners undergoing h-refinement, the number of inner iteration steps is fixed at 64 for all mesh sizes, which is very high.

Tables 4.3, 4.5 and 4.7 record the numbers of outer iteration steps required for convergence with respect to varying numbers of inner iteration steps. Mesh size is

fixed at $1/128$. The last rows in these Tables, tagged as 'bound', show the upper bounds of the error reduction rates associated with the selected acceleration parameters for linear algebraic system $(K^X + K^Y)b = \mathcal{F}$.

In all the demonstrated numerical results, CG is applied as the method for the outer iteration and forced to stop whenever the relative residual (in ℓ_2 norm) is smaller than $1e$-7 or 200 iteration steps has been reached.

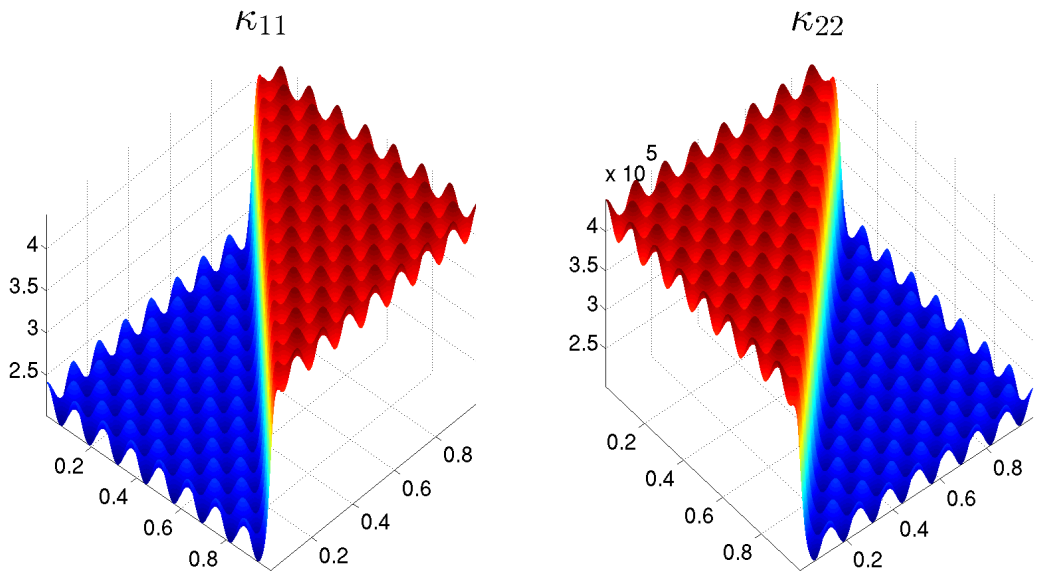


Figure 4.1: Orthotropic.

N_{1D}	32	64	128	256	512
$\left(K^X + K^Y\right)^{-1}$	11	12	12	13	13
$\left(\bar{K}^X + \bar{K}^Y\right)^{-1}$	6	5	5	4	4

Table 4.2: h-scaling: $p = 1,\ c = 0,\ N_{inner} = 64$.

Inner iterations	4	8	16	32	64
$\left(K^X + K^Y\right)^{-1}$	18	13	12	12	12
$\left(\bar{K}^X + \bar{K}^Y\right)^{-1}$	12	8	5	5	5
Upper bound	1.21E-01	3.66E-03	3.35E-06	2.81E-12	1.97E-24

Table 4.3: Effect of inner iterations: $p = 1,\ c = 0,\ N_{1D} = 128$.

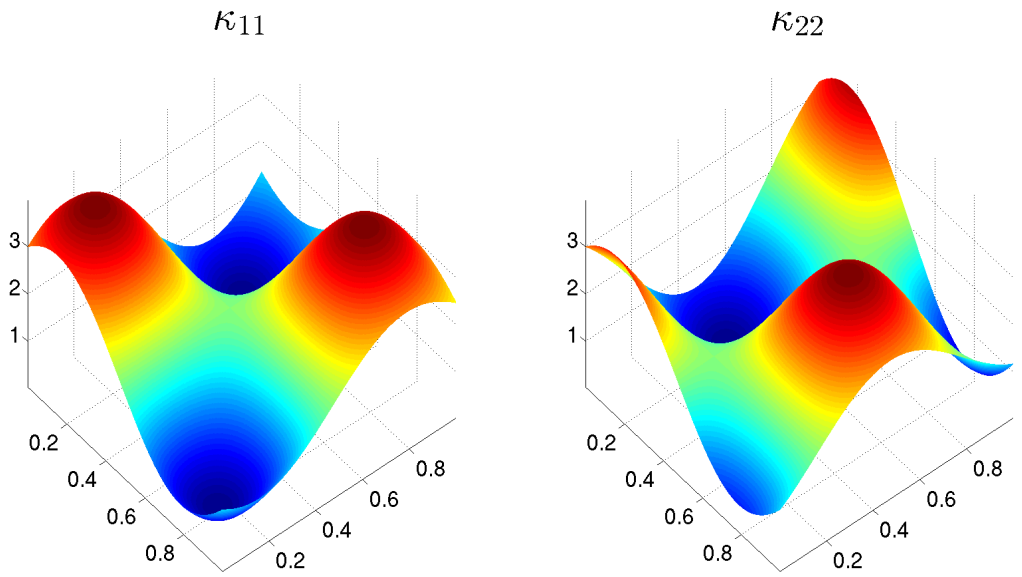


Figure 4.2: Low frequency oscillation.

N_{1D}	32	64	128	256	512
$\left(K^X + K^Y\right)^{-1}$	43	58	76	88	93
$\left(\bar{K}^X + \bar{K}^Y\right)^{-1}$	6	5	4	4	3

Table 4.4: h-scaling: $p = 1,\ c = 0,\ N_{inner} = 64$.

Inner iterations	4	8	16	32	64
$\left(K^X + K^Y\right)^{-1}$	79	77	77	77	77
$\left(\bar{K}^X + \bar{K}^Y\right)^{-1}$	8	5	4	4	4
Upper bound	1.21E-01	3.66E-03	3.35E-06	2.81E-12	1.97E-24

Table 4.5: Effect of inner iterations: $p = 1,\ c = 0,\ N_{1D} = 128$.

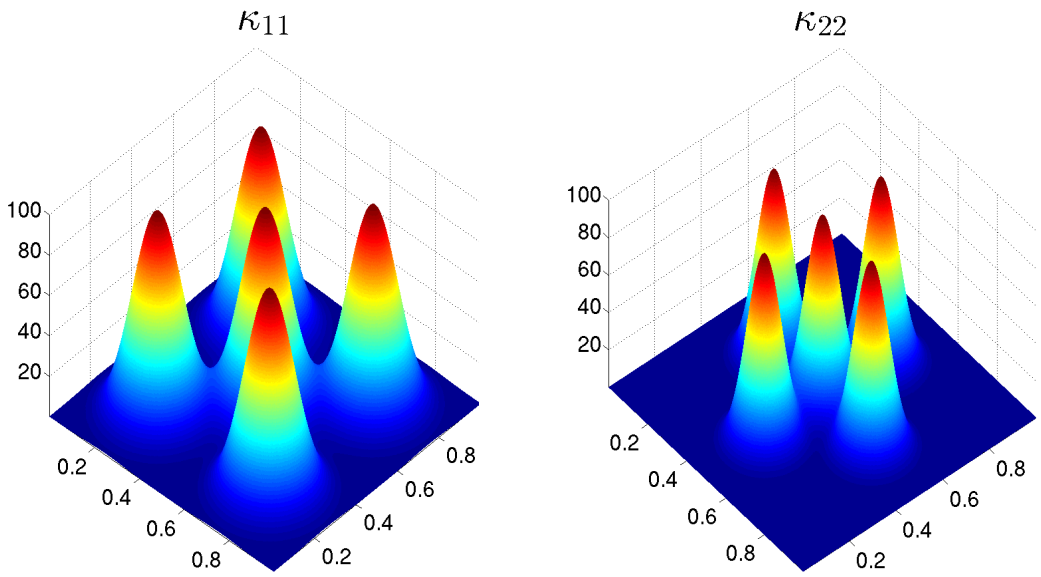


Figure 4.3: Gaussian spikes.

N_{1D}	32	64	128	256	512
$\left(K^X + K^Y\right)^{-1}$	200	200	200	200	200
$\left(\bar{K}^X + \bar{K}^Y\right)^{-1}$	16	10	10	9	10

Table 4.6: h-scaling: $p = 1,\ c = 0,\ N_{inner} = 64$.

Inner iterations	4	8	16	32	64
$\left(K^X + K^Y\right)^{-1}$	200	200	200	200	200
$\left(\bar{K}^X + \bar{K}^Y\right)^{-1}$	200	200	19	13	10
Upper bound	1.21E-01	3.66E-03	3.35E-06	2.81E-12	1.97E-24

Table 4.7: Effect of inner iterations: $p = 1,\ c = 0,\ N_{1D} = 128$.

In Table 4.2, we can see that for coefficient shown in Figure 4.1, both preconditioners work very well. The high orthotropic ratio is well handled by both preconditioners. For the case of $(\bar{K}^X + \bar{K}^Y)$, we see a slight decrease of the outer iteration numbers as we refine the mesh. This is due to a better approximability of the preconditioner since it is constructed based on the local support property.

Part of the reason that $(K^X + K^Y)^{-1}$ works so well for the coefficient shown in Figure 4.1 is because of the high frequent uniform oscillation. Recall Figure 3.3, in

this case, the coefficient variation inside different strips that correspond to support regions of different basis functions are similar to each other, thus using $(\bar{K}^X + \bar{K}^Y)^{-1}$, a preconditioner adapts itself towards the coefficient variation from strip to strip, does not show significant benefit. The coefficient shown in Figure 4.2 is manufactured such that $(\bar{K}^X + \bar{K}^Y)^{-1}$ will show obvious advantage against $(K^X + K^Y)^{-1}$.

The coefficient shown in Figure 4.3 is designed to show that $(\bar{K}^X + \bar{K}^Y)$ has the ability to handle rougher situations.

On the other hand, from Tables 4.3, 4.5 and 4.7, we can tell that little can be gained by further increasing the number of inner iteration steps after the recorded upper bound of error reduction rate has been pushed to the level of $1e$-3, except for the very rough case shown in Figure 4.3.

4.3 Isotropic coefficients with high contrasts

In this section, we focus on the Poisson equation with isotropic inhomogeneous coefficients in the following form:

$$-\nabla \cdot \big(\kappa(x,y)\nabla u(x,y)\big) = f(x,y), \tag{4.35}$$

where $\kappa(x,y)$ is a scalar function. The simplification from orthotropic coefficients to isotropic coefficients enables us to use the hybrid preconditioning technique, similar to what we did to the mass matrices in Section 3.3.4.

Our interest is particularly focused on the cases where coefficient κ has high contrast variation among different regions. In these cases, the 2D preconditioner $(\bar{K}^X + \bar{K}^Y)^{-1}$ is ill-conditioned due to the high contrast coefficient and therefore, require a significant amount of ADI iterations. Thus $(\bar{K}^X + \bar{K}^Y)^{-1}$ is no longer an economic choice and we shift our attention to hybrid preconditioning.

For more information on numerical techniques for high contrast problems, one can

consult [2, 29, 43, 44, 84] and the references therein.

4.3.1 Hybrid preconditioning

After finite element discretization with tensor product basis functions $B^y \otimes B^x$, (4.35) leads us to the following linear algebraic system:

$$(\tilde{K}^X + \tilde{K}^Y)b = \mathcal{F}, \tag{4.36}$$

where

$$\tilde{K}^X = \int_{\square} \left(B^y (B^y)^T\right) \otimes \left(B^x_{,x} (B^x_{,x})^T\right) \kappa d_{\square}, \tag{4.37a}$$

$$\tilde{K}^Y = \int_{\square} \left(B^y_{,y} (B^y_{,y})^T\right) \otimes \left(B^x (B^x)^T\right) \kappa d_{\square}. \tag{4.37b}$$

In order to build the hybrid preconditioner, $(K^X + K^Y)^{-1}$ is definitely one necessary component. Next, we want to find another preconditioner that can capture the variation in the coefficient κ. Denote this preconditioner as D_κ^{-1}. We build D_κ in a similar manner as we did in the mass matrix case, that is, assemble the diagonal of the stiffness matrix, but with the basis functions and their derivatives replaced by constant 1, and then rescale with respect to the sizes of their support region.

Due to the complexity of the high contrast stiffness matrix, the following symmetrically hybridized preconditioner $D_\kappa^{-1/2}(K^X + K^Y)^{-1} D_\kappa^{-1/2}$ does not work. Instead, we define our hybrid preconditioner as $D_\kappa^{-1}(K^X + K^Y)^{-1}$ and combine it with the Generalized Minimal Residual (GMRES) method. Specifically, we use right preconditioning since in this way, residual of the original linear system is minimized in the ℓ_2 norm.

The order of D_κ^{-1} and $(K^X + K^Y)^{-1}$ in the hybrid preconditioner is chosen in accordance with the choice of right preconditioning. Since D_κ^{-1} contains high contrast

variation, we want to apply it after $(K^X+K^Y)^{-1}$ in order to ease the affect of round-off error.

4.3.2 Numerical results

In this section, we test the quality of the hybrid preconditioner on three examples, namely, the 'Island', 'Block' and 'Borehole', with the comparison of applying preconditioner $(K^X + K^Y)^{-1}$ or D_κ^{-1} individually. In order to have a clear comparison, we incorporate $(K^X+K^Y)^{-1}$ and D_κ^{-1} with GMRES as well, albeit they can be combined with the computationally more economic method: CG.

For all the numerical results demonstrated in this section, GMRES restarts every 30 iteration steps and stops whenever one of the following conditions is satisfied:

1) the ℓ_2 norm of the relative residual is lower than the tolerance 1e-7;

2) maximum number of iteration steps 5×30 has been reached;

3) the iterative process has stagnated.

In all the following tables, the number of iteration steps is denoted as $A(B)$ where A stands for the number of restarts while B stands for the number of GMRES iterations after the latest restart. Therefore, the total number of iterations can be calculated as $(A-1)\times 30+B$. The superscript ^{s}, which appears occasionally, means the iterative process has been stopped due to stagnation; The dash '—' means the iterative process has reached the maximum number of iteration steps.

In Table 4.8, 4.10 and 4.12, N_{outer} columns show the number of outer iteration steps, R_{res} columns show the relative residual in the ℓ_2 norm while R_{err} columns show the relative error in the energy norm. N_{inner} stands for the number of inner iteration steps.

Example 4.1. *As shown in Figure 4.4, in the center of the whole computational domain, we have a smaller rectangular domain where the coefficient κ is very high*

(10^2 in case A, 10^4 in case B and 10^6 in case C), in contrast with the rest of the domain, called the background, where κ is 1. The same example can be found in [2], where it is referred to as the 'one island' problem.

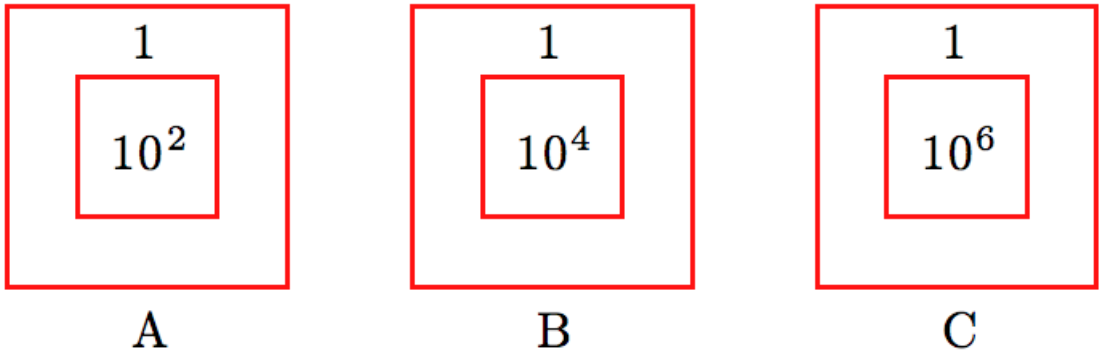


Figure 4.4: 'Island'.

		$\left(K^X + K^Y\right)^{-1}$			$\left(D_\kappa\right)^{-1}$			Hybrid		
	N_{1D}	N_{outer}	R_{res}	R_{err}	N_{outer}	R_{res}	R_{err}	N_{outer}	R_{res}	R_{err}
A	128	1(13)	2.3E-08	2.0E-09	—	8.2E-02	8.8E-01	1(10)	4.2E-09	8.9E-10
	256	1(14)	5.1E-08	4.8E-09	—	5.1E-02	9.2E-01	1(10)	6.5E-08	1.5E-08
	512	1(15)	2.9E-08	6.7E-10	—	1.7E-01	9.5E-01	1(11)	9.1E-08	1.7E-09
B	128	1(17)	6.1E-08	5.9E-09	—	8.3E-02	8.9E-01	1(11)	9.0E-09	2.3E-09
	256	1(18)	5.0E-08	6.5E-09	—	5.1E-02	9.2E-01	1(11)	7.5E-08	1.4E-08
	512	1(20)	6.3E-08	1.1E-09	—	1.7E-01	9.5E-01	$1(11)^s$	8.3E-06	1.9E-07
C	128	1(21)	9.0E-08	9.8E-08	—	8.3E-02	8.9E-01	1(12)	6.7E-08	9.4E-08
	256	$1(23)^s$	1.5E-07	7.5E-08	—	5.1E-02	9.2E-01	$1(12)^s$	2.9E-06	3.1E-07
	512	$1(24)^s$	7.2E-06	1.4E-06	—	1.7E-01	9.5E-01	$1(13)^s$	8.5E-04	1.5E-05

Table 4.8: Performances: $p = 1,\ c = 0,\ N_{inner} = 16$.

	N_{1D}	2	4	8	16	32	64
A	128	1(25)	1(13)	1(10)	1(10)	1(10)	1(10)
	256	2(1)	$1(16)^s$	1(11)	1(10)	1(10)	1(10)
	512	$2(18)^s$	$1(16)^s$	$1(11)^s$	1(11)	1(11)	1(11)
B	128	$1(28)^s$	$1(14)^s$	1(11)	1(11)	1(11)	1(11)
	256	$2(18)^s$	$1(16)^s$	$1(11)^s$	1(11)	1(11)	1(11)
	512	$3(26)^s$	$1(18)^s$	$1(12)^s$	$1(11)^s$	1(12)	1(12)
C	128	$2(21)^s$	$1(16)^s$	1(11)	1(12)	1(12)	1(12)
	256	$3(11)^s$	$1(18)^s$	$1(12)^s$	$1(12)^s$	$1(12)^s$	$1(12)^s$
	512	—	$1(21)^s$	$1(13)^s$	$1(13)^s$	$1(13)^s$	$1(13)^s$

Table 4.9: Effect of inner iterations on the hybrid preconditioner: $p = 1,\ c = 0$.

Example 4.2. *In this example, the whole computational domain, as shown in Figure 4.5, is partitioned uniformly into small pieces of rectangles. In each small rectangle,*

coefficient κ is assigned with a specific value that differs from the other rectangles. These numbers are printed on each of the rectangles. For instance, 3e3 *means* 3×10^3. *The coefficient κ varies from* 1e1 *to* 4e3 *in In case A,* 1e1 *to* 4e6 *in case B and* 1e1 *to* 4e9 *in case C.*

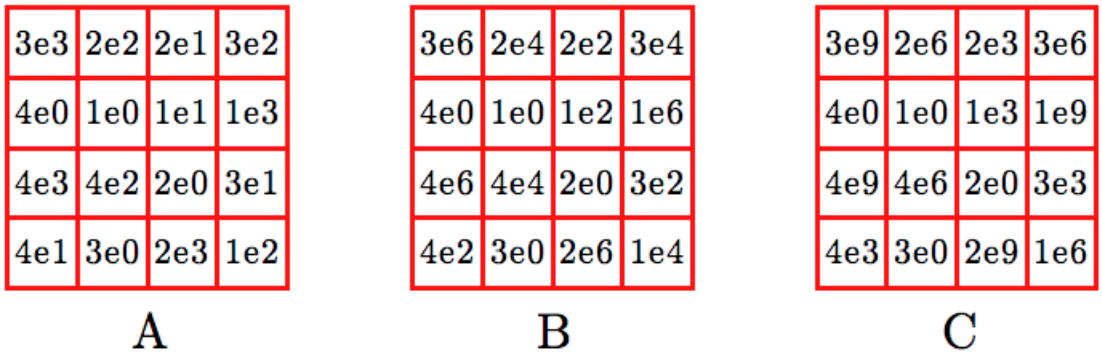


Figure 4.5: 'Block'.

		$(K^X + K^Y)^{-1}$			$(D_\kappa)^{-1}$			Hybrid		
	N_{1D}	N_{outer}	R_{res}	R_{err}	N_{outer}	R_{res}	R_{err}	N_{outer}	R_{res}	R_{err}
A	128	—	2.5E-02	5.0E-02	—	4.6E-03	1.1E-02	1(20)	1.8E-08	1.7E-08
	256	—	2.9E-02	5.4E-02	—	2.5E-02	1.2E-01	1(21)	2.6E-08	2.8E-08
	512	—	5.0E-02	5.8E-02	—	1.8E-01	5.9E-01	1(22)	7.6E-08	1.5E-08
B	128	—	5.5E-01	9.8E-01	—	4.0E-03	9.9E-03	1(21)	2.1E-08	2.1E-08
	256	—	5.6E-01	9.8E-01	—	3.8E-02	9.5E-02	1(22)	3.8E-08	4.2E-08
	512	—	8.1E-01	1.0E+00	—	1.8E-01	5.6E-01	1(24)	5.4E-08	1.2E-08
C	128	—	7.2E-01	1.0E+00	—	4.0E-03	9.9E-03	1(21)	2.9E-08	2.8E-08
	256	—	7.2E-01	1.0E+00	—	4.1E-02	1.1E-01	1(22)	5.7E-08	6.2E-08
	512	—	8.4E-01	1.0E+00	—	1.8E-01	5.5E-01	1(24)	6.8E-08	1.5E-08

Table 4.10: Performances: $p = 1,\ c = 0,\ N_{inner} = 16$.

	N_{1D}	2	4	8	16	32	64
A	128	1(27)	1(20)	1(20)	1(20)	1(20)	1(20)
	256	2(5)	1(21)	1(21)	1(21)	1(21)	1(21)
	512	2(16)	$1(24)^s$	1(23)	1(22)	1(22)	1(22)
B	128	1(29)	1(21)	1(21)	1(21)	1(21)	1(21)
	256	2(7)	1(23)	1(22)	1(22)	1(22)	1(22)
	512	2(20)	$1(30)^s$	1(24)	1(24)	1(24)	1(24)
C	128	1(29)	1(22)	1(21)	1(21)	1(21)	1(21)
	256	2(7)	1(23)	1(22)	1(22)	1(22)	1(22)
	512	2(20)	$1(28)^s$	1(24)	1(24)	1(24)	1(24)

Table 4.11: Effect of inner iterations on the hybrid preconditioner: $p = 1,\ c = 0$.

Example 4.3. *In this example, the whole computational domain is combined by four horizontal layers with a thin vertical layer in the middle, which resembles the transversal surface of a borehole. The numbers showed in Figure 4.6 are the reciprocals of κ, called the resistivity.*

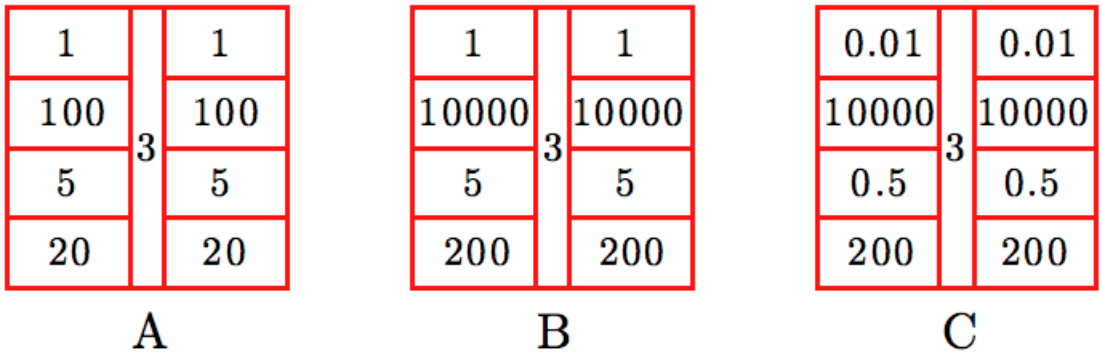


Figure 4.6: 'Borehole'.

		$\left(K^X + K^Y\right)^{-1}$			$(D_\kappa)^{-1}$			Hybrid		
	N_{1D}	N_{outer}	R_{res}	R_{err}	N_{outer}	R_{res}	R_{err}	N_{outer}	R_{res}	R_{err}
A	128	2(6)	8.9E-08	6.7E-08	—	7.4E-03	4.1E-02	1(15)	5.0E-08	3.3E-08
	256	2(7)	8.9E-08	7.0E-08	—	2.6E-02	3.7E-01	1(16)	5.1E-08	3.2E-08
	512	2(12)	6.6E-08	1.8E-08	—	1.5E-01	7.0E-01	1(17)	7.4E-08	7.5E-09
B	128	3(4)	9.6E-08	9.0E-08	—	9.0E-03	5.8E-02	1(16)	3.3E-08	3.1E-08
	256	3(27)	9.8E-08	2.1E-08	—	2.8E-02	1.3E-01	1(16)	8.4E-08	8.4E-08
	512	4(2)	7.6E-08	3.2E-08	—	1.6E-01	1.5E-01	1(18)	7.6E-08	1.5E-08
C	128	—	4.5E-01	9.4E-01	—	1.0E-02	1.9E-02	1(20)	4.7E-08	3.6E-08
	256	—	4.5E-01	9.3E-01	—	3.2E-02	4.1E-02	1(21)	6.4E-08	4.8E-08
	512	—	6.7E-01	9.9E-01	—	1.7E-01	4.0E-01	1(23)	3.5E-08	4.9E-09

Table 4.12: Performances: $p = 1,\ c = 0,\ N_{inner} = 16$.

	N_{1D}	2	4	8	16	32	64
A	128	1(23)	1(16)	1(15)	1(15)	1(15)	1(15)
	256	1(27)	1(17)	1(16)	1(16)	1(16)	1(16)
	512	$2(11)^s$	$1(19)^s$	$1(17)^s$	1(17)	1(17)	1(17)
B	128	1(24)	1(17)	1(16)	1(16)	1(16)	1(16)
	256	1(28)	1(17)	1(16)	1(16)	1(16)	1(16)
	512	$2(8)^s$	$1(20)^s$	$1(22)^s$	1(18)	1(18)	1(18)
C	128	1(27)	1(21)	1(20)	1(20)	1(20)	1(20)
	256	1(30)	1(23)	1(21)	1(21)	1(21)	1(21)
	512	$2(10)^s$	$1(25)^s$	$1(24)^s$	1(23)	1(23)	1(23)

Table 4.13: Effect of inner iterations on the hybrid preconditioner: $p = 1,\ c = 0$.

From Tables 4.8, 4.10 and 4.12, we see that the hybrid preconditioner brings significant improvement on the convergence speed, comparing with applying preconditioner $(K^X + K^Y)^{-1}$ or D_κ^{-1} individually. Meanwhile, when increasing the contrast in the coefficient, or refining the computational mesh, the number of iteration steps corresponding to the hybrid preconditioner is growing, but only very mildly. Moreover, despite of the ill-conditioned linear system, reduction on energy norm of the solution is satisfactory.

From Tables 4.9, 4.11 and 4.13, once again, we observe that the effect of the number of inner iteration steps is diminishing as it increases and eventually, it plays no role on the number of outer iteration steps required for convergence.

4.4 Complicated geometry

An important motivation for our investigation on tensor product basis functions is Isogeometric analysis (IGA). It is natural to consider the preconditioning techniques developed above for solving stiffness matrices arising from IGA.

For instance, we consider the Poisson equation with isotropic homogeneous coefficients defined on the two testing domains shown in Figure 3.2. Numerical results concerning the performance of preconditioner $\left(K^X + K^Y\right)^{-1}$ are demonstrated in Table 4.14, where CG is used for the outer iteration while ADI is used for the inner iteration. The number of inner iteration steps is fixed at 64. The whole iteration process is stopped whenever the relative residual (in ℓ_2 norm) is lower than $1e$-7 or the number of outer iterations reaches 200. Row 'Stretched' and row 'Perturbed' record the number of outer iterations needed for convergence on the stretched rectangle and the perturbed rectangle shown in Figure 3.2, respectively.

N_{1D}	32	64	128	256	512
Strected	57	63	70	78	85
Perturbed	28	38	44	50	55

Table 4.14: Performance of $\left(K^X + K^Y\right)^{-1}$ on complicated geometries.

Preconditioner $\left(K^X + K^Y\right)^{-1}$ does not work well in this case as we see a growing number of outer iterations required for convergence when refining the mesh. This is due to the numerical anisotropy introduced by the complicated geometries. In general, direction splitting algorithms do not work well for anisotropic cases. This is easier to understand from a physical perspective: anisotropy simply means the two directions are coupled together. The application of direction splitting preconditioner on IGA stiffness matrix is therefore, not successful.

Chapter 5

The Helmholtz equation

5.1 The ideal case

In this section, we consider the following linear system:

$$\left(K^X + K^Y - k^2 M\right) b = \mathcal{F}, \quad k^2 > 0 \tag{5.1}$$

arising from finite element discretizations of the Helmholtz equation (Model Problem 3) with tensor product basis functions.

(5.1) has a similar looking with the following linear system:

$$\left(K^X + K^Y + k^2 M\right) b = \mathcal{F}, \quad k^2 > 0 \tag{5.2}$$

which might come from numerical discretization of a reaction-diffusion problem, or a time-dependent heat equation. However, these two equations present totally different numerical difficulties. We have discussed numerical techniques for (5.2) in Chapter 4 as a byproduct of the discussion for stiffness matrices. The issue back there is the growing condition number as we refine the mesh.

On the other hand, the numerical difficulty of (5.1) lies in its indefiniteness brought by the negative sign in front of the mass matrix term. A detailed discussion on why

most classical iterative methods do not work well for Helmholtz problem can be found in review paper [39].

From a numerical point of view, due to its indefiniteness, (5.1) is very sensitive in the sense that given the same right hand side $\mathcal{F}$, a slight perturbation of the matrix can lead to a totally different solution. This sensitivity makes it very hard to design a good preconditioner that is purely algebraic-based. Instead, preconditioners based on simulating the physical effects are typically better choices. For instance, it is mentioned in [39] that analytic incomplete LU (AILU) works much better than incomplete LU (ILU) as a preconditioner for Helmholtz problem.

For more information and advances on numerical methods for Helmholtz problem, one can consult, for instance, [14, 32, 33, 34, 39] and the references therein.

5.1.1 The difficulty of indefiniteness for ADI

Due to the similar lookings between (5.1) and (5.2), one might think that ADI is readily applicable for (5.1) as well. However, this is far from true.

According to (4.16):

$$\mathcal{P}^{(k)}v_i = \left(\frac{r^{(k)} - \lambda_i^X}{r^{(k)} + \lambda_i^X}\right) \cdot \left(\frac{r^{(k)} - \lambda_i^Y}{r^{(k)} + \lambda_i^Y}\right) v_i,$$

when both λ_i^X and λ_i^Y are negative, setting $r^{(k)}$ to a positive number will increase the error associated with v_i. Figuratively, the errors associated with negative and positive generalized eigenvalues are playing a teeterboard when ADI iterations are applied, that is, when one side of the error is pushed down, the other side rises up. The usually leads to a blowing up of the error.

As an attempt to fix this, recall from Chapter 4 that if all the generalized eigenvalues $\{\lambda_i^X\}_{i=1}^N$ or $\{\lambda_i^Y\}_{i=1}^N$ are known *a priori*, a direct method for (5.2) at computational cost $\mathcal{O}(N^{\frac{3}{2}})$ can be derived based on ADI iterations. This is still valid for (5.1).

Aside from Theorem 4.3, here we provide a different explanation based on transformations of the equations, which may offer some extra insight on the effect of ADI iterations for linear systems like (5.1) and (5.2). Recall the relationship between errors at different ADI iteration steps:

$$\begin{aligned}(r^{(k)}M + K^X)e^{(k+\frac{1}{2})} &= (r^{(k)}M - K^Y)e^{(k)},\\ (r^{(k)}M + K^Y)e^{(k+1)} &= (r^{(k)}M - K^X)e^{(k+\frac{1}{2})}.\end{aligned}$$

If we express the errors in terms of columns of the generalized eigen-matrix V:

$$e^j = V\epsilon^j, \quad j = k, k+\frac{1}{2}, k+1,$$

these relationships can be rephrased as:

$$\begin{aligned}(r^{(k)}M + K^X)V\epsilon^{(k+\frac{1}{2})} &= (r^{(k)}M - K^Y)V\epsilon^{(k)},\\ (r^{(k)}M + K^Y)V\epsilon^{(k+1)} &= (r^{(k)}M - K^X)V\epsilon^{(k+\frac{1}{2})}.\end{aligned}$$

Left multiplying V^T on both sides of these equations, we end up with:

$$(r^{(k)} + I^y \otimes D^x)\epsilon^{(k+\frac{1}{2})} = (r^{(k)} - D^y \otimes I^x)\epsilon^{(k)}, \tag{5.3a}$$

$$(r^{(k)} + D^y \otimes I^x)\epsilon^{(k+1)} = (r^{(k)} - I^y \otimes D^x)\epsilon^{(k+\frac{1}{2})}. \tag{5.3b}$$

If we set $r^{(k)} = \lambda_i^Y$, then (5.3a) simultaneously eliminates errors associated with all the generalized eigenvectors corresponding to λ_i^Y. Similar for (5.3b) if we set $r^{(k)} = \lambda_i^X$.

Moreover, due to the diagonal structure of the matrices in (5.3a) and (5.3b), once these errors (in the sense of ϵ) are eliminated, they shall stay at zero during the subsequent iterations if exact arithmetic were performed.

Furthermore, for (5.1), positive generalized eigenvalues often outnumber their opponents quite a lot in practice, particularly for low wavenumber cases.

All the above observations are motivations of a potential solution for the teeter-board dilemma: Eliminate all the errors corresponding to the negative generalized eigenvalues by setting $r^{(k)}$ to these values in ADI iterations, then apply ADI iterations with optimal acceleration parameters to eliminate errors corresponding to the rest positive generalized eigenvalues.

Unfortunately, this idea fails as well due to two reasons: 1) The presence of round off error perturbs the eliminated errors and eventually leads to a catastrophic blowing up; 2) By the time all the errors corresponding to negative generalized eigenvalues are eliminated, errors corresponding to positive generalized eigenvalues have been amplified quite a lot, which leaves a big space for round off error to play a role.

Calling columns of V the eigen-modes, the above understanding makes us wonder whether or not it is possible to pick certain eigen-modes and solve them, but without messing up the rest. The answer is yes, as explained in the following section.

5.1.2 A direct method approach

We start from transforming (5.1) as shown in the following:

$$\begin{aligned} & \left(K^X + K^Y - k^2 M\right) b &&= \mathcal{F} \\ \Rightarrow \quad & V^T \left(K^X + K^Y - k^2 M\right) V V^{-1} b &&= V^T \mathcal{F} \\ \Rightarrow \quad & \left(I^y \otimes D^x + D^y \otimes I^x - k^2 (I^y \otimes I^x)\right) V^{-1} b &&= V^T \mathcal{F}. \end{aligned}$$

This further leads us to the explicit formula for solution of linear system (5.1):

$$b = V \left(I^y \otimes D^x + D^y \otimes I^x - k^2 (I^y \otimes I^x)\right)^{-1} V^T \mathcal{F}. \tag{5.4}$$

Recall that $V = V^y \otimes V^x$. The multiplications involved in (5.4) can be dealt with in a Kronecker product manner at computational cost of $\mathcal{O}(N^{\frac{3}{2}})$. For 3D, the computational cost sounds even better: $\mathcal{O}(N^{\frac{4}{3}})$. Therefore, (5.4) itself serves as a good direct

method for solving (5.1).

Calculations of the generalized eigen-pairs, albeit in 1D, can be costly if high accuracy is demanded. However, in lots of real life applications, problem (5.1) is often presented with multiple sources (right hand sides) or multiple wavenumbers (k) where these generalized eigen-pairs can be reused.

The direct method (5.4) solves all the eigen-modes simultaneously. The following questions arise natural: do we really need that many eigen-modes to represent the solution? What if we only want a decent approximation?

To simplify notation, we define

$$\mathcal{D} = \left(I^y \otimes D^x + D^y \otimes I^x - k^2(I^y \otimes I^x)\right).$$

(5.4) can then be written as:

$$b = V\mathcal{D}^{-1}V^T\mathcal{F}. \tag{5.5}$$

Symbolically, (5.5) can also be applied for the 1D case, after adapting the definition of $\mathcal{D}$ to $\mathcal{D} = D^x - k^2 I^x$. For simplicity, we first discuss this 1D case as a motivation. In the following, for diagonal matrices like $\mathcal{D}$, we use 'entry' as a short term for 'diagonal entry' if there is no ambiguity.

To illustrate the varying importance of different eigen-modes on representing the solution, we present a simple 1D numerical experiment here. For wavenumber $k = 50$, setting mesh size $h = \dfrac{1}{80}$ such that $kh \approx \dfrac{2\pi}{10}$ (roughly 10 elements per wavelength), we plot the entries of $\mathcal{D}$ and $\mathcal{D}^{-1}$ in Figure 5.1, where zero abscissa corresponds to the first entry in D^x that is bigger than k^2. (Linear basis functions are used in the finite element discretization.)

From Figure 5.1, we have the following observations: for entries in D^x that are around k^2, cancellation happens and the corresponding entries in $\mathcal{D}^{-1}$ are large; on the contrary, for entries in D^x that are much larger than k^2, the corresponding entries

in $\mathcal{D}^{-1}$ are close to zero.

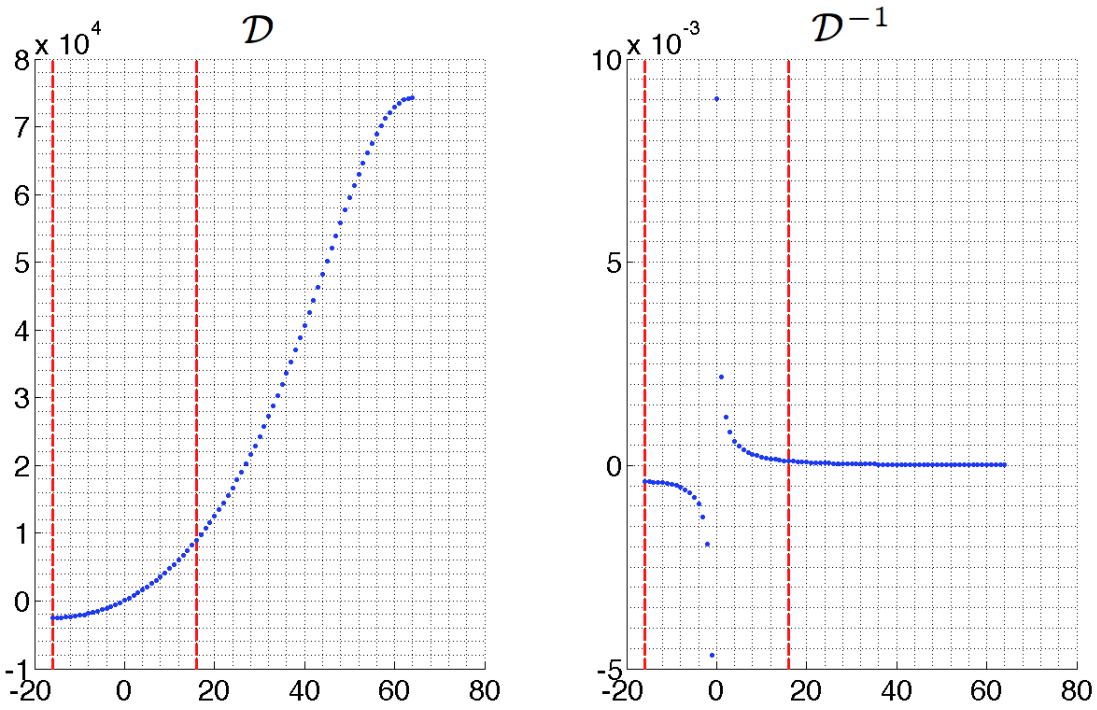


Figure 5.1: Eigenvalue distribution: 1D case.

Thus, eigen-modes corresponding to entries around k^2 in D^x are likely to be more important on representing the solution. Just picking these eigen-modes while ignorantly throwing the rest away may already provide a decent approximation for the solution.

To test this idea, we pick eigen-modes with the following procedure: For the 1D case, we pick the eigen-modes corresponding to all the negative entries and the next consecutive positive entries in $\mathcal{D} = D^x - k^2 I^x$ at the same amount; For the 2D case, we first pick the 1D eigen-modes in the same manner according to entries in $D^x - \frac{k^2}{2} I^x$ and $D^y - \frac{k^2}{2} I^y$. Their tensor products are picked as the 2D eigen-modes; The 3D case is dealt with in a similar manner, except that we consider entries in $D^x - \frac{k^2}{3} I^x$, $D^y - \frac{k^2}{3} I^y$ and $D^z - \frac{k^2}{3} I^z$ instead.

It is interesting to check how much does this cost. The number of negative entries in $\mathcal{D}$ has major dependence on the wavenumber k (almost proportional) while minor

dependence on mesh size h, as illustrated in Tables 5.1 and 5.2 for the 1D case. In these tables, N_{neg} stands for the number of negative entries in $\mathcal{D} = D^x - k^2 I^x$. We set $kh = 0.625$ in Table 5.1 and $k = 40$ in Table 5.2.

$k(1/h)$	10(16)	20(32)	40(64)	80(128)	160(256)	320(512)
N_{neg}	4	7	13	26	51	101
ratio	23.5%	21.2%	20.0%	20.2%	19.8%	19.7%

Table 5.1: Number and ratio of negative entries in $\mathcal{D} = D^x - k^2 I^x$.

$k(1/h)$	40(16)	40(32)	40(64)	40(128)	40(256)	40(512)
N_{neg}	11	13	13	13	13	13
ratio	64.7%	39.4%	20.0%	10.1%	5.1%	2.5%

Table 5.2: Number and ratio of negative entries in $\mathcal{D} = D^x - k^2 I^x$.

If we keep $kh = 0.625$, for the 1D case, the ratio of the negative entries in $\mathcal{D}$ is roughly 20% as shown in Table 5.1. Therefore, roughly 40% of the total eigen-modes are picked. For the 2D case, the ratio is roughly 8% while for the 3D case, roughly 1.3% only.

From 1D to 3D, the ratio gets smaller and smaller mainly because the eigen-modes are picked in a tensor product manner. Far more importantly, due to the same reason,

$$b_p = V_p \mathcal{D}_p^{-1} V_p^T \mathcal{F} \tag{5.6}$$

can be performed in a tensor product manner by exploiting the Kronecker product structures in V_p and V_p^T, where V_p is constructed by retaining only the picked eigen-modes in V while replacing the rest with zero vectors and $\mathcal{D}_p^{-1}$ is constructed similarly from $\mathcal{D}^{-1}$; b_p is the approximate solution obtained.

Therefore, the cost for performing (5.6) is still at $\mathcal{O}(N^{\frac{3}{2}})$ in 2D and $\mathcal{O}(N^{\frac{4}{3}})$ in 3D, same as performing (5.5), but with much smaller constants.

Table 5.3 shows some numerical results regarding the quality of this approximate solution b_p, in terms of the relative error e_p (in ℓ_2 norm). Linear system (5.1) dis-

cretized from the 2D Model Problem 3 with tensor product linear basis functions is considered; The right hand sides are generated randomly. We set $kh = 0.625$ for Table 5.3.

$k(1/h)$	10(16)	20(32)	40(64)	80(128)	160(256)	320(512)
e_p	4.27E-02	5.40E-02	6.22E-02	4.80E-03	7.83E-03	4.59E-03

Table 5.3: Relative error for b_p.

If higher accuracy is demanded, ADI iterations can now be applied to (5.1) using b_p as the starting point. Since the error associated with the selected eigen-modes are already eliminated by b_p, we only need to choose the acceleration parameters according to the remaining generalized eigenvalues, which are all positive. Table 5.4 shows some numerical results with 16 ADI iterations following the starting point b_p, where relative errors (in ℓ_2 norm) before and after the ADI iterations are recorded. We set $kh = 0.625$ for Table 5.4.

$k(1/h)$	10(16)	20(32)	40(64)	80(128)	160(256)	320(512)
e_p	1.03E-01	8.35E-02	7.17E-02	7.08E-03	1.19E-02	3.89E-03
e	1.85E-10	4.79E-09	2.73E-08	2.20E-09	3.14E-09	9.50E-10

Table 5.4: Relative error: 16 ADI iterations.

From Table 5.4, we see that 16 ADI iterations work very well this time. However, large number of ADI iterations is still not recommended due to the presence of round off error. This is because of the unstable nature of the ADI iterations for indefinite problems. We show some numerical results in Table 5.5 with varying number of ADI iterations to demonstrate this. We set $k = 160$, $h = \frac{1}{256}$ for Table 5.5.

ADI iterations	2	4	8	16	32	64
e_p	1.02E-02	8.53E-03	7.84E-03	8.58E-03	7.11E-03	8.84E-03
e	1.36E-03	1.70E-04	3.65E-06	2.23E-09	3.80E-09	9.48E-03

Table 5.5: Relative error: varying number of ADI iterations.

5.1.3 A stable alternative: GMRES

Due to the results shown in Table 5.5, a robust alternative to ADI iterations is preferred. Our choice is the Generalized minimal residual method (GMRES), [70].

Simply applying GMRES to linear system (5.1) with starting point b_p, but with no preconditioner, the convergence speed is slow, as shown in Table 5.6, where GMRES(10), which restarts after every 10 iterations, is applied with maximum number of restarts set as 10 and tolerance of relative residual set as $1e$-6.

In Table 5.6, row e_p and row e record the relative errors (in ℓ_2 norm) before and after the GMRES iterations, respectively while row N_{iter} records the number of iteration steps that has been performed.

$k(h)$	10(16)	20(32)	40(64)	80(128)	160(256)	320(512)
e_p	2.07E-02	9.70E-02	5.97E-02	7.71E-03	1.60E-02	4.20E-03
e	7.71E-04	2.44E-03	1.66E-03	1.50E-04	2.18E-04	4.32E-05
N_{iter}	10(10)	10(10)	10(10)	10(10)	10(10)	10(10)

Table 5.6: Performance of GMRES(10) with starting point b_p but no preconditioner.

The results in Table 5.6 justify the need of a good preconditioner. We propose the inverse of mass matrix $M = M^y \otimes M^x$ as the preconditioner, originally motivated by its cheap application cost. Applying GMRES(10) to linear system (5.1) again, but with starting point b_p and preconditioner M^{-1}, the convergence speed is significantly improved, as shown in Table 5.7.

$k(h)$	10(16)	20(32)	40(64)	80(128)	160(256)	320(512)
e_p	4.27E-02	5.40E-02	6.22E-02	4.80E-03	7.83E-03	4.59E-03
e	1.75E-07	4.33E-07	7.11E-07	5.50E-08	8.54E-08	4.65E-08
N_{iter}	3(10)	5(1)	6(2)	6(2)	6(1)	6(1)

Table 5.7: Performance of GMRES(10) with starting point b_p and preconditioner M^{-1}.

However, if we apply GMRES(10) with preconditioner M^{-1}, but with zero starting point, the convergence speed is disastrously slow, as shown in Table 5.8.

$k(h)$	10(16)	20(32)	40(64)	80(128)	160(256)	320(512)
e_p	1.58E-02	6.25E-02	6.31E-02	1.05E-02	1.02E-02	5.29E-03
e	9.93E-01	9.72E-01	9.83E-01	9.99E-01	9.99E-01	1.00E+00
N_{iter}	10(10)	10(10)	10(10)	10(10)	10(10)	10(10)

Table 5.8: Performance of GMRES(10) with zero starting point and preconditioner M^{-1}.

Results shown in Tables 5.6 - 5.8 reveal that the combination of starting point b_p and preconditioner M^{-1} can speed up the convergence rate for GMRES significantly. We explain this in the following by theory of Krylov subspace methods.

For brevity, we denote the matrix $K^X + K^Y - k^2 M$ in (5.1) as A. According to Lemma 4.1, we have:

$$AV = MV\mathcal{D},$$

where $V = V^y \otimes V^x$ and $\mathcal{D} = I^y \otimes D^x + D^y \otimes I^x - k^2(I^y \otimes I^x)$.

After applying m iteration steps of any Krylov subspace method to linear system $Ab = \mathcal{F}$, the relation between current error and initial error can be written in the following polynomial form:

$$e^{(m)} = p^{(m)}(A)e^{(0)}, \tag{5.7}$$

where $p^{(m)}(z)$ is a polynomial of degree m satisfying $p^{(m)}(0) = 1$. For more details, one can consult [30, 41, 47, 70]. Different Krylov subspace methods lead to different forms of polynomial $p^{(m)}(z)$. For example, CG leads to the polynomial that minimizes A-norm of the error while GMRES leads to the polynomial that minimizes ℓ_2 norm of the residual. Nevertheless, specific form of the polynomial is not the concern here.

If we precondition the linear system $Ab = \mathcal{F}$, matrix A in relation (5.7) shall be replaced by the preconditioned matrix. For instance, if we left precondition (5.1) with M^{-1}, the error relationship becomes:

$$e^{(m)} = p^{(m)}(M^{-1}A)e^{(0)}. \tag{5.8}$$

Next, we express the error at each iteration step in terms of the columns of the eigen-matrix V as the following form:

$$e^{(i)} = \sum_{j=1}^{N} \epsilon_j^{(i)} V_j = V\epsilon^{(i)}, \quad i = 0, 1, \ldots, m. \tag{5.9}$$

Substituting (5.9) into (5.8), we obtain:

$$e^{(m)} = p^{(m)}(M^{-1}A)V\epsilon^{(0)}. \tag{5.10}$$

Recall that $AV = MV\mathcal{D}$. We have $M^{-1}A = V\mathcal{D}V^{-1}$. Therefore, (5.10) leads us to:

$$e^{(m)} = V\left(p^{(m)}(\mathcal{D})\epsilon^{(0)}\right), \tag{5.11}$$

i.e.,

$$\epsilon^{(m)} = p^{(m)}(\mathcal{D})\epsilon^{(0)}, \tag{5.12}$$

where $p^{(m)}(\mathcal{D})$ is diagonal since $\mathcal{D}$ is diagonal.

According to (5.11) and (5.12), if in the initial error $e^{(0)} = V\epsilon^{(0)}$, some components of $\epsilon^{(0)}$ are zeros, they will remain at zeros in the subsequent iterations if exact arithmetic were used. In other words, those eigen-modes that are eliminated by b_p never come back and play a role in the subsequent iterations. Therefore, if b_p eliminates all the negative eigen-modes, the indefiniteness of linear system (5.1) is effectively 'hidden' to the Krylov subspace methods. Similarly, if b_p also eliminates eigen-modes corresponding to small positive entries in $\mathcal{D}$, the performances of Krylov subspace methods will be further enhanced due to smaller effective condition number.

Remark 5.1. *Due to the inaccuracy in the eigen-pairs used to construct b_p, their corresponding components can be small numbers initially, instead of exact zeros. These small numbers can have influences in the subsequent iterations.*

Remark 5.2. *Right preconditioning $Ab = \mathcal{F}$ with M^{-1} can be analyzed in a similar manner by using the following algebraic relation: $M^{-1}p^{(k)}(AM^{-1}) = p^{(k)}(M^{-1}A)M^{-1}$.*

Remark 5.3. *Since the indefiniteness is 'hidden' by the starting point b_p, CG may be a better alternative than GMRES for symmetric linear systems. For instance, we apply CG to linear system* (5.1) *with tolerance of relative residual set as* 1e-6 *and record the numerical results in Table 5.9.*

$k(h)$	10(16)	20(32)	40(64)	80(128)	160(256)	320(512)
e_p	4.88E-02	6.54E-02	6.27E-02	6.33E-03	1.05E-02	1.05E-02
e	5.97E-08	9.52E-08	1.19E-07	1.18E-08	1.89E-08	2.32E-08
N_{iter}	30	41	49	49	49	48

Table 5.9: CG is applicable.

Comparing the results in Tables 5.7 and 5.9, CG takes less iteration steps with less computational cost per iteration, therefore seems to be a better choice comparing with GMRES. However, in general, linear systems arising from Helmholtz problems are typically non-symmetric due to complicated boundary conditions. Therefore, in the following content, we still show numerical results with GMRES.

Remark 5.4. *Recall the relation $V^TMV = I$. We can express M^{-1} as: $M^{-1} = VI^{-1}V^T$. It is possible to have a preconditioner better than M^{-1} by replacing I^{-1} with a different diagonal matrix that approximates $\mathcal{D}^{-1}$ better. More detail can be found in C.1.*

5.1.4 An obvious extension

The techniques developed in Sections 5.1.2 and 5.1.3 for Model Problem 3 can be easily extended to the heterogeneous case where the heterogeneity depends only on one direction. In this section, we consider the following PDE:

$$-\nabla\cdot\left(\frac{1}{\rho}\nabla u(x,y)\right)-\frac{k^2}{\mathcal{B}}u(x,y)=f(x,y), \tag{5.13}$$

defined on a rectangular domain. In (5.13), ρ and $\mathcal{B}$ are scalar functions depending on x-direction only, possibly with discontinuities.

After finite element discretization with tensor product basis functions, (5.13) leads us to linear algebraic system:

$$\left(K_\rho^X + K_\rho^Y - k^2 M_\mathcal{B}\right) b = \mathcal{F}, \quad k^2 > 0, \tag{5.14}$$

where $K_\rho^X = M^y \otimes K_\rho^x$, $K_\rho^Y = K^y \otimes M_\rho^x$ and $M = M^y \otimes M_\mathcal{B}^x$. Matrices M^y and K^y have the same definitions as before while M_ρ^x, K_ρ^x and $M_\mathcal{B}^x$ are defined as:

$$M_\rho^x = \int_x \frac{1}{\rho} B^x (B^x)^T d_x, \; K_\rho^x = \int_x \frac{1}{\rho} B_{,x}^x (B_{,x}^x)^T d_x \text{ and } M_\mathcal{B}^x = \int_x \frac{1}{\mathcal{B}} B^x (B^x)^T d_x.$$

With the above notations, (5.14) can be rewritten as:

$$\left(M^y \otimes \left(K_\rho^x - k^2 M_\mathcal{B}^x\right) + K^y \otimes M_\rho^x\right) b = \mathcal{F}, \quad k^2 > 0. \tag{5.15}$$

Meanwhile, we assume the following relations among these 1D matrices:

$$\begin{aligned} \left(K_\rho^x - k^2 M_\mathcal{B}^x\right) V^x &= M_\rho^x V^x D^x; \\ K^y V^y &= M^y V^y D^y, \end{aligned}$$

where D^x and D^y are diagonal matrices while relations $\left(V^x\right)^T M_\rho^x V^x = I^x$ and $\left(V^y\right)^T M^y V^y = I^y$ hold. Define $V = V^y \otimes V^x$. The matrix in linear system (5.15) can still be diagonalized by left multiplying with V^T and right multiplying with V while the solution b can still be expressed as (5.4). Therefore, the techniques developed in Sections 5.1.2 and 5.1.3 can still be applied to solve linear system (5.15), with the adapted definitions of V^x and D^x.

We show a simple example in the following where piecewise constant coefficients on x-direction is considered.

Example 5.1. *In this example, we apply GMRES(20), which restarts after every 20 iterations, to linear system* (5.14) *built from* (5.13) *with coefficients:*

$$\mathcal{B}(x) = \begin{cases} 5 & 0 \le x \le 0.25; \\ 1 & 0.25 < x \le 0.5; \\ 10 & 0.5 < x \le 0.75; \\ 2 & 0.75 < x \le 1, \end{cases} \text{ and } \rho(x) = \begin{cases} 0.5 & 0 \le x \le 0.25; \\ 1 & 0.25 < x \le 0.5; \\ 2 & 0.5 < x \le 0.75; \\ 1 & 0.75 < x \le 1. \end{cases} \tag{5.16}$$

The maximum number of restarts for GMRES(20) is set as 10 while tolerance of the relative residual is set as 1e-6. $M^{-1} = (M^y)^{-1} \otimes \left(M_\rho^x\right)^{-1}$ *is used as the preconditioner while* b_p *constructed as in* (5.6) *is used as the starting point. Numerical results are shown in Table 5.10 where row* N_{picked} *records the number of picked eigen-pairs in order to construct* b_p.

$k(h)$	10(16)	20(32)	40(64)	80(128)	160(256)	320(512)
e_p	6.18E-01	3.74E-01	1.97E-01	1.21E-01	1.20E-01	9.29E-02
e	8.47E-06	1.70E-06	2.80E-06	2.29E-06	2.69E-06	1.57E-06
N_{iter}	3(1)	2(15)	3(2)	3(10)	3(10)	3(10)
N_{picked}	4	10	18	32	62	126
ratio	5.5%	9.2%	7.7%	6.2%	5.8%	6.0%

Table 5.10: Performance of GMRES(20): Example 5.1.

From Table 5.10, we can see that both the number of iterations and the ratio of selected eigen-modes are approaching constants for the coefficients shown in (5.10).

5.2 More general cases

For most real life problems, ρ and c in (5.13) typically depend on more than one spatial direction, which leads us to the more realistic linear system:

$$\left(K - k^2 M\right) b = \mathcal{F}, \quad k^2 > 0, \tag{5.17}$$

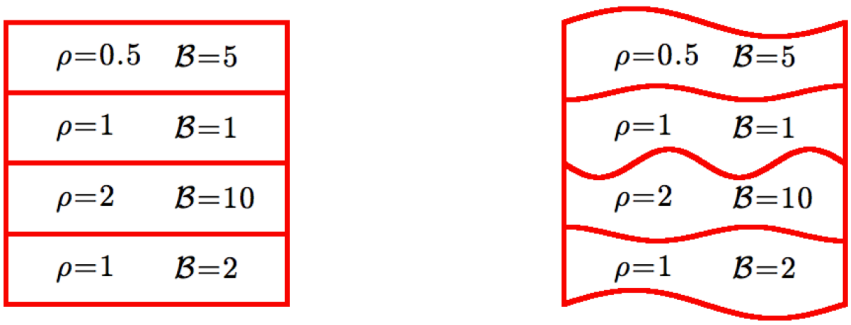
where both K and M are entangled due to either ρ or c. As a result, the calculation of 2D generalized eigen-pairs cannot be decomposed into 1D problems anymore.

However, for certain problems that are close enough to those ideal cases examined in Section 5.1, we can resort to the idea of preconditioning. One example about the Helmholtz equation on layered media with curved interfaces is shown in Section 5.2.1.

On the other hand, when wavenumber k in (5.17) is small enough such that calculation of the 2D/3D eigen-pairs is affordable, we can still tackle the problem by removing the indefiniteness with these 2D/3D eigen-pairs. Further discussion on this approach can be found in Section 5.2.2.

5.2.1 Approach 1: preconditioning

In this section, the Helmholtz equation with layered media coefficients, shown as the physical domain in Figure 5.2, is considered. Detailed information about this physical domain can be found in Appendix C.2, including formulae of the curved interfaces and boundaries, as well as the mapping from the parametric domain to the physical domain. Images on the physical domain of the finite elements on the parametrical domain under this mapping are conforming with these interfaces and boundaries.



Parametrical domain Physical domain

Figure 5.2: Layered media.

If these curved interfaces are replaced with straight lines, as shown as the parametrical domain in Figure 5.2, we know how to solve (5.17) efficiently, already demon-

strated in Example 5.1. Here, we exploit this ability for preconditioning, i.e., we use the solution for Helmholtz problem defined on the parametrical domain as the approximate solution for Helmholtz problem defined on the physical domain.

To solve the Helmholtz problem on parametrical domain, or in another word, to apply the preconditioner, we use the direct method with formula shown in (5.4), albeit we can improve the computational efficiency by using a starting point b_p constructed as (5.6), followed with a fixed number of ADI iterations, validated by the results shown in Table 5.4. It can be shown that in either way, the resulting preconditioner has a fixed form, as long as we pick the same eigen-pairs to construct b_p at each iteration. We choose the first one such that the numerical tests reflect more accurate information about the approximation we make on the layered media coefficients.

Numerical results are demonstrated in Table 5.11 for different wavenumber k and mesh size $h(= 1/N_{1D})$. GMRES(50), which restarts every 50 iterations, is applied with maximum number of restarts set as 10 and tolerance of relative residual set as $1e$-7.

	N_{1D}	16	32	64	128	256	512
k=10	N_{iter}	1(15)	1(17)	1(17)	1(17)	1(18)	1(18)
	R_{err}	6.3E-07	1.5E-07	1.4E-07	1.3E-07	9.9E-08	1.2E-07
	R_{res}	2.0E-07	2.6E-08	2.9E-08	3.3E-08	1.0E-08	2.1E-08
k=20	N_{iter}		1(26)	1(28)	1(29)	1(28)	1(28)
	R_{err}		2.4E-07	3.5E-06	6.1E-07	9.6E-07	9.9E-07
	R_{res}		2.9E-08	2.1E-08	1.9E-08	3.3E-08	3.0E-08
k=40	N_{iter}			1(49)	1(48)	1(46)	1(45)
	R_{err}			1.1E-06	1.8E-06	2.7E-06	3.6E-06
	R_{res}			2.9E-08	3.2E-08	7.1E-08	1.7E-07

Table 5.11: Performance of GMRES(50).

Due to the indefiniteness of the discrete Helmholtz equation, this preconditioning approach only works for relatively low wavenumber, even just for this special case.

5.2.2 Approach 2: remove the indefiniteness

This approach is rather straightforward: calculate the 2D/3D generalized eigen-pairs and construct the starting point b_p accordingly to remove the indefiniteness of (5.17).

Therefore, we are presented with the following generalized eigenvalue problem:

$$Kv_i = \lambda_i M v_i \text{ for } i = 1, 2, \ldots. \tag{5.18}$$

We are only interested in those generalized eigenvalues that are smaller than k^2 in order to remove the indefiniteness and possibly several extra generalized eigenvalues that are slightly bigger than k^2 in order to improve the condition number.

Moreover, we can obtain good starting points for generalized eigenvalue problem (5.18) cheaply, that is, the generalized eigenvectors corresponding to the homogeneous media case (5.1). It is also worth mentioning that the generalized eigen-pairs of (5.18) can be reused for different right hand side $\mathcal{F}$ and different wavenumber k. Besides, these generalized eigen-pairs do not have to be stored simultaneously and therefore do not impose a burden on the memory.

However, recall the results shown in Table 5.1 where for the 1D case, the number of negative generalized eigenvalues for (5.1) is almost proportional to wavenumber k. For the 3D case, it is almost proportional to k^3, which makes the situation even worse. Thereby, for high wavenumber, this approach is not effective either based on current technology.

Remark 5.5. *The numerical challenge of solving discrete Helmholtz equation with iterative methods mainly comes from its indefiniteness. Even the currently most powerful family of iterative methods, Krylov subspace methods, cannot handle indefinite linear system efficiently. A strategy to resolve the indefiniteness of the linear system is therefore crucial for a successful iterative solving. In a lot of occasions, the difficulty of indefiniteness is first transformed into another form; then a solution technique is*

proposed for the new difficulty.

Normal equation. *An obvious approach following this process is: instead of the original equation $Ab = \mathcal{F}$, solve the normal equation $A^T Ab = \mathcal{A}^T F$. Indeed, the normal equation delivers the same solution while we only have to deal with a positive definite matrix. However, the price is also dramatic: condition number of $A^T A$ is squared comparing with A. Therefore, the difficulty of indefiniteness is transformed into ill-condition. Of course, one would rarely apply iterative methods without a preconditioner. If it is easier to precondition the normal equation, then this transformation is worthwhile.*

Shifted Laplace. *Another example is the class of shifted Laplace preconditioners, which eases the indefiniteness of the discrete Helmholtz equation by driving the generalized eigenvalues towards the right half of the complex plane with a Laplace-like preconditioner. For detail about shifted Laplace preconditioners, one can consult [37, 80]. The difficulty of indefiniteness is transformed into repeatedly applying the preconditioner, which can be expensive. Extensive research has been done on efficiently applying these preconditioners, see for instance, [35, 36, 74].*

Our approach. *In comparison, our approach also shares some similarity. By constructing a starting point that hides all the negative eigen-modes from the Krylov subspace methods, the difficulty of indefiniteness is transformed into a series of generalized eigenvalue problems. For general cases, whether or not this transformation is worthwhile depends on the techniques for the generalized eigenvalue problem, which is left for future work.*

Chapter 6

Miscellaneous

6.1 On boundary conditions

In this section, we discuss the effect of boundary conditions on matrix structure and how to adapt those matrix structure based algorithms accordingly. Two most commonly used boundary conditions, Dirichlet and Neumann boundary conditions, are considered.

Matrix structure is unchanged under explicit imposition of Neumann boundary conditions since they only affect the right hand side. However, explicit imposition of Dirichlet boundary conditions delete those involved degrees of freedom (dofs) from the linear system, therefore may change the matrix structure.

Take the model problems presented in section 1.1 as examples, it is clear that as long as each edge of the domain is imposed with only 1 type of boundary condition, either Dirichlet or Neumann, matrices arising from these model problems still possess the Kronecker product structure.

The tricky case comes when different types of boundary conditions are imposed on the same edge, for instance, when only half of an edge is imposed with Dirichlet boundary condition while all the rest is imposed with Neumann boundary condition, as illustrated Figure 6.1.

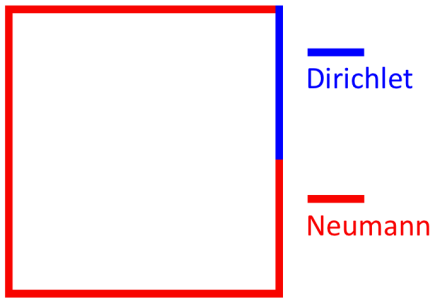


Figure 6.1: Boundary conditions and matrix structure.

The corresponding matrices do not possess the Kronecker product structure for missing some pieces due to the Dirichlet boundary condition. In order to restore the Kronecker product structure, we need to put these missing pieces back, which leads to 'wrong' matrices. In the following, we present a tactic to adjust the iterative methods accordingly such that these 'wrong' matrices can be used to calculate the 'right' solution.

Consider the following abstract PDE:

$$\begin{aligned} \mathcal{L}(u) &= f && \text{on } \Omega, && (6.1a)\\ \nabla u \cdot \overrightarrow{n} &= g_N && \text{on } \partial\Omega_N, && (6.1b)\\ u &= g_D && \text{on } \partial\Omega_D, && (6.1c) \end{aligned}$$

where $\mathcal{L}$ is a second order elliptic operator, $\overrightarrow{n}$ is the outward normal vector and $\partial\Omega_N \cup \partial\Omega_D = \partial\Omega$.

Suppose (6.1a) and (6.1b) lead us to the following linear system after finite element discretization with basis functions $\{B_i\}_{i=1}^n$:

$$Ax = b, \tag{6.2}$$

in which A is an $n \times n$ matrix while x and b are $n \times 1$ vectors.

Yet we still need to impose the Dirichlet boundary condition (6.1c). Without loss of generality, we assume that the first n_N basis functions in $\{B_i\}_{i=1}^n$ correspond to Dirichlet boundary condition (6.1c), therefore need to be deleted from linear system (6.2). Denote the first n_D components of x and b as x_D and b_D, respectively; Denote the rest components of x and b as x_R and b_R, respectively, at size $n_R = n - n_N$. We also decompose the matrix A accordingly as follows:

$$A = \begin{bmatrix} A_{DD} & A_{DR} \\ A_{RD} & A_{RR} \end{bmatrix}. \tag{6.3}$$

Explicit imposition of Dirichlet boundary condition (6.1c) leads to prescription of values for x_D. Denote these values as G_D, an $n_D \times 1$ vector originated from g_D in (6.1c). With the above notations, the linear system that corresponds to (6.1a), (6.1b) and (6.1c) can be written as:

$$\begin{bmatrix} A_{DD} & O_{DR} \\ A_{RD} & A_{RR} \end{bmatrix} \begin{bmatrix} x_D \\ x_R \end{bmatrix} = \begin{bmatrix} A_{DD}G_D \\ b_R \end{bmatrix}, \tag{6.4}$$

where O_{DR} stands for a zero matrix with the same size as A_{DR}. Linear system (6.4) is equivalent to

$$A_{RR}x_R = b_R - A_{RD}G_D \tag{6.5}$$

in the sense that they deliver the same solution for x_R.

To distinguish, we refer to (6.2) as the *original* linear system; refer to (6.4) as the *modified* linear system; refer to (6.5) as the *reduced* linear system. Analogically, we refer to the matrices in (6.2), (6.4) and (6.5) the *original* matrix, the *modified* matrix and the *reduced* matrix, respectively.

At the first stage, we want to have an iterative method to solve the *desired* linear system by performing operations with the *original* matrix at each iteration. As an

example, we present a modified conjugate gradient method in Algorithm 6.

Algorithm 6.

1: *Initial guess:* $x^{(0)} = [G_D^T, (x_R^{(0)})^T]^T$
2: *Initial residual:* $r^{(0)} = [O_D^T, b_R - A_{RD}G_D - A_{RR}x_R^{(0)}]^T$ *
3: *Initial direction:* $p^{(0)} = r^{(0)}$
4: *Initial measure of the residual:* $tol = norm(r^{(0)})$
5: $k = 0$
6: ***while*** *(tol is not small enough)* ***do***
7: $\quad \alpha_{num} = (r^{(k)})^T r^{(k)}$
8: $\quad \alpha_{den} = (p^{(k)})^T A p^{(k)}$
9: $\quad \alpha^{(k)} = \dfrac{\alpha_{num}}{\alpha_{den}}$
10: $\quad x^{(k+1)} = x^{(k)} + \alpha^{(k)} p^{(k)}$
11: $\quad r^{(k+1)} = r^{(k)} - \alpha^{(k)} A p^{(k)}$
12: $\quad$ *Set the first* n_D *components of* $r^{(k+1)}$ *to* 0*:* $r^{(k+1)}(1 : n_D) = O_D$ *
13: $\quad tol = norm(r^{(k+1)})$
14: $\quad \beta_{num} = (r^{(k+1)})^T r^{(k+1)}$
15: $\quad \beta^{(k)} = \dfrac{\beta_{num}}{\alpha_{num}}$
16: $\quad p^{(k+1)} = r^{(k+1)} + \beta^{(k)} p^{(k)}$
17: $\quad k = k + 1$
18: ***end while***

---

* O_D stands for a zero vector with the same size as G_D.

In Algorithm 6, we manually change the searching direction at each iteration (line 12) such that x_D, i.e., dofs corresponding to the Dirichlet boundary condition, never get updated. Therefore, if the initial guess satisfies the Dirichlet boundary condition, all the approximate solutions at subsequent iterations will do so.

Moreover, with some simple algebraic derivation following the lines, it is easy to show that Algorithm 6 is just a special approach to apply the usual conjugate gradient method on the *reduced* linear system (6.5) in the sense that, parameters $\alpha^{(k)}$, $\beta^{(k)}$ and the last n_R components of vectors $r^{(k)}$, $p^{(k)}$, $x^{(k)}$ produced in these two approaches coincide with each other at every iteration.

Next, we bring preconditioning into the picture. Instead of $Ax = b$, we consider the preconditioned linear system $PAx = Pb$. Consistent with the decomposition of

A as in (6.3), P can also be decomposed as

$$P = \begin{bmatrix} P_{DD} & P_{DR} \\ P_{RD} & P_{RR} \end{bmatrix},$$

where every block matrix has the same size as their correspondence in (6.3). We present a modified preconditioned conjugate gradient method in Algorithm 7 to solve the *modified* linear system (6.4), which only requires performing operations with A and P at each iteration.

Algorithm 7.

Initial guess: $x^{(0)} = [g^T, (x_R^{(0)})^T]^T$
Initial residual: $r^{(0)} = [O_D^T, b_R - A_{RD}G_D - A_{RR}x_R^{(0)}]^T$ *
Initial assist vector: $z^{(0)} = Pr^{(0)}$
Set the first n_D *components of* $z^{(0)}$ *to* 0*:* $z^{(0)}(1:n_D) = O_D$ *
Initial direction: $p^{(0)} = z^{(0)}$
Initial measure of the residual: $tol = norm(r^{(0)})$
$k = 0$
while *(tol is not small enough)* ***do***
    $\alpha_{num} = (r^{(k)})^T z^{(k)}$
    $\alpha_{den} = (p^{(k)})^T A p^{(k)}$
    $\alpha^{(k)} = \dfrac{\alpha_{num}}{\alpha_{den}}$
    $x^{(k+1)} = x^{(k)} + \alpha^{(k)} p^{(k)}$
    $r^{(k+1)} = r^{(k)} - \alpha^{(k)} A p^{(k)}$
    Set the first n_D *components of* $r^{(k+1)}$ *to* 0*:* $r^{(k+1)}(1:n_D) = O_D$ *
    $tol = norm(r^{(k+1)})$
    $z^{(k+1)} = Pr^{(k+1)}$
    Set the first n_D *components of* $z^{(k+1)}$ *to* O*:* $z^{(k+1)}(1:n_D) = O_D$ *
    $\beta_{num} = (r^{(k+1)})^T z^{(k+1)}$
    $\beta^{(k)} = \dfrac{\beta_{num}}{\alpha_{num}}$
    $p^{(k+1)} = z^{(k+1)} + \beta^{(k)} p^{(k)}$
    $k = k + 1$
end while

---

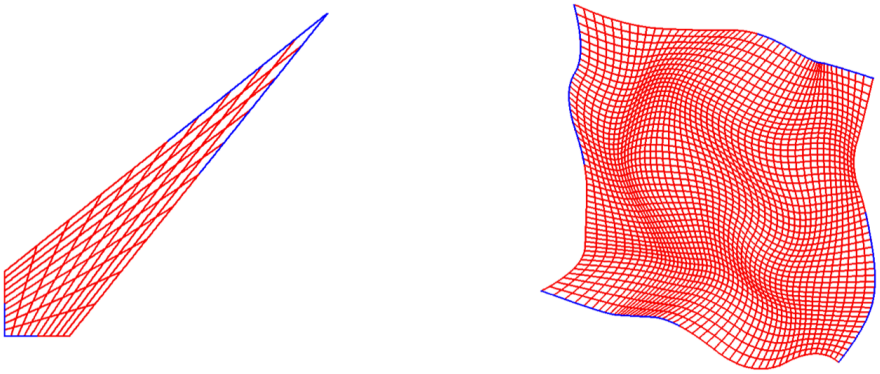
* O_D stands for a zero vector with the same size as G_D.

Similar to Algorithm 6, one can show that Algorithm 7 is just a special approach to apply the usual preconditioned conjugate gradient method on the *reduced* linear

system (6.5) with preconditioner P_{RR}.

Remark 6.1. *Algorithm 6 and 7 can still be applied when the matrix or the preconditioner is not explicitly formed since no surgery on the linear system is needed for imposing Dirichlet boundary condition. Instead, these algorithms themselves guarantee that the approximate solution is always compatible with the Dirichlet boundary condition.*

In the following, we show some numerical results to demonstrate the effect of boundary conditions on the convergence speed. Inversion of mass matrices built on complicated geometries is considered. The concerned geometries, shown in Figure 6.2, are exactly the same ones in Figure 3.2. The blue lines indicate the part of boundary that is imposed with Dirichlet boundary condition while the rest are imposed with Neumann boundary condition. Each blue line is mapped from half of the corresponding edge in the parametrical domain.



Stretched rectangle Perturbed rectangle

Figure 6.2: Testing domains with boundary conditions.

Table 6.1 - 6.3 record the number of iteration steps required for convergence with preconditioner M^{-1}, preconditioner $M^{\xi S}$ and the hybrid preconditioner, correspondingly. The iteration process is stopped when relative residual (in ℓ_2 norm) is less than 1e-12. In all these tables, bilinear basis functions ($p = 1, c = 0$) are used while

the mesh size is varying. Row 'boundary' corresponds to Figure 6.2 while row '—' corresponds to the full Neumann boundary condition case, shown here for comparison.

M^{-1}	N_{1D}	2^3	2^4	2^5	2^6	2^7	2^8	2^9	2^{10}
Stretched	Boundary	19	26	31	34	35	35	36	36
	—	26	31	32	33	33	33	33	33
Perturbed	Boundary	22	31	36	40	42	43	43	44
	—	25	31	36	39	42	44	44	45

Table 6.1: h-scaling: $p = 1, c = 0$.

$M^{\xi S}$	N_{1D}	2^3	2^4	2^5	2^6	2^7	2^8	2^9	2^{10}
Stretched	Boundary	10	10	9	8	8	8	7	6
	—	8	6	5	5	4	4	3	3
Perturbed	Boundary	13	12	10	10	8	8	7	7
	—	12	10	9	8	7	6	5	4

Table 6.2: h-scaling: $p = 1, c = 0$.

Hybrid	N_{1D}	2^3	2^4	2^5	2^6	2^7	2^8	2^9	2^{10}
Stretched	Boundary	10	9	8	7	7	7	7	6
	—	8	7	5	5	4	4	3	3
Perturbed	Boundary	14	13	11	10	9	9	8	7
	—	14	13	10	9	7	6	5	5

Table 6.3: h-scaling: $p = 1, c = 0$.

From Table 6.1 - 6.3, we see that for bilinear basis functions, the proposed preconditioners can handle the mixed boundary conditions shown in Figure 6.2 fairly well. The convergence speed is not heavily affected by these mixed boundary conditions.

Remark 6.2. *When highly continuous basis functions are used, the proposed preconditioners do not work as nicely as with bilinear basis functions for the boundary conditions shown in Figure 6.2. This is because, due to the broad interactions between high continuous basis functions, the Kronecker product structure in the matrix is damaged more severely for losing those dofs corresponding to the Dirichlet part of the boundary.*

6.2 On isogeometric spectral element method

Transformation. Kronecker product properties can also be used on transforming geometry from one representation to another.

For instance, suppose originally we have a 2D B-spline represented function: $f = b^T B$, where $B = B^y \otimes B^x$, and we want to represent it with Lagrangian basis functions $L = L^y \otimes L^x$. If L^x and B^x are built with the same polynomial order on the same mesh, we have $B^x \subseteq L^x$. Under the same condition, $B^y \subseteq L^y$ and therefore, $B \subseteq L$, i.e., f can be represented by L without losing any accuracy.

To find the coefficients of f corresponding to L, we recall the properties of Kronecker product in section 2.1, particularly the mixed product property. Suppose we have 1D relations $B^x = A^x L^x$ and $B^y = A^y L^y$, then,

$$B = (A^y L^y) \otimes (A^x L^x) = (A^y \otimes A^x)(L^y \otimes L^x).$$

Define $A = A^y \otimes A^x$, we have $B = AL$ and subsequently, $f = \left(A^T b\right)^T L$. Multiplying b with A^T is cheap due to its Kronecker product structure. A fast algorithm similar to Algorithm 1 can be designed for this job. Thus, transforming a B-spline representation to a Lagrangian representation is conceptually easy and computationally cheap.

This simple transformation process can be generalized to NURBS represented function as well. Suppose we have NURBS basis functions $R_B = \dfrac{B}{W_N}$, where $W_N = w^T B$, and function f is represented as $f = b^T R_B$. Since $B = AL$, W_N can also be written as $W_N = \left(A^T w\right)^T L$. Moreover, define $R_L = \dfrac{L}{W_N}$, we have $R_B \subseteq R_L$ since $B \subseteq L$. Therefore, f can be represented by R_L without losing any accuracy. Specifically, $f = \left(A^T b\right)^T R_L$. We refer to R_L as rational Lagrangian basis functions.

Spectral element method. The reason that one might want to go through this transformation, rather than using B-spline basis functions directly as FEM basis functions, is that there are existing efficient techniques developed for Lagrangian basis

functions, which may overpay the transformation cost.

As an example, spectral element method [18, 52, 54] is based on Lagrangian basis functions. By using Gauss-Lobatto-Legendre (GLL) quadrature rule [25, 55, 56] and collocating the interpolating points of Lagrangian basis functions with the GLL quadrature points, the resulting mass matrix is diagonal, based on which a fast explicit dynamic solver can be developed. This result also extends to the induced rational Lagrangian basis functions. We refer to these (rational) Lagrangian basis functions, with their interpolating points collocating with the GLL quadrature points, as (rational) GLL basis functions.

With the ability of transforming a B-spline/NURBS represented geometry to a GLL/rational GLL represented geometry efficiently, we are able to have a smooth interface between CAD program and spectral element simulation. Moreover, since this transformation does not lose any accuracy of the geometry, the resulting spectral element method also inherits the isogeometric feature.

Chapter 7

Concluding Remarks

7.1 Summary

In this work, we present efficient iterative algorithms to invert matrices arising from finite element discretization of various partial differential equations with tensor product basis functions by exploiting the Kronecker product structure in these matrices, inherited from the basis functions.

We start from efficient algorithms for ideal model problems where the matrices therein possess the Kronecker product structures. Then we move to more general cases and presents various preconditioning techniques. Those algorithms developed for the ideal cases, or their variants, are exploited on applying the preconditioners efficiently.

Specifically, we deal with three different types of matrices that present different difficulties, namely, the mass matrices, the stiffness matrices and matrices arising from the Helmholtz equation.

For the mass matrices, the computational complexity inherent with the basis functions used in isogeometric analysis raises concern on their numerical efficiency. Higher continuous basis functions are shown to be harder to deal with using direct methods. We thus turn to iterative methods and focus on designing effective preconditioners.

First, we propose a 'plain' preconditioner that addresses the issue of the basis func-

tions. Numerical performance of this preconditioner is shown to be independent of mesh size, polynomial order and continuity order when combined with the conjugate gradient method. However, this 'plain' preconditioner does not provide correction for complicated geometries, which arise in virtually all applications of isogeometric analysis. We then propose preconditioners that provide partial correction for complicated geometries, achieving better performances than the 'plain' preconditioner. Meanwhile, we also present a hybrid preconditioner that is numerically competitive with better adaptivity on nonlinear or inverse problems.

For the stiffness matrices, the increasing condition number as we refine the mesh is the major numerical concern. We propose a compound iteration preconditioner to mitigate the mesh dependence for iterative methods. For inhomogeneous orthotropic media, this preconditioner works very well when combined with the conjugate gradient method. For fully anisotropic media, it does not provide satisfactory results due to the strong physical coupling of different directions or distortion of the mesh. We also consider the isotropic media with high contrast, for which a hybrid preconditioner is proposed, whose performance only depends on the contrast mildly.

For matrices arising from the Helmholtz equation, the numerical challenge mainly comes from the indefiniteness. Krylov subspace methods, or in general, polynomial based iterative methods, do not work efficiently for indefinite problems. We provide a strategy to 'hide' the indefiniteness from the Krylov subspace methods by constructing a starting point that eliminates the error corresponding to all the negative generalized eigen-modes. This strategy can also be used to improve the condition number.

However, the cost is also sizable since the number of negative generalized eigen-pairs is proportional to k^d where k is the wave number and d is the spatial dimension of the problem. For 3D heterogenous problem with high wave number, removing the indefiniteness in this approach can be very expensive. Thus this strategy is now only effective for low wave number case.

7.2 Future Research Work

Mass matrix

On the direction of preconditioning mass matrix, the author would like to focus on the applications of the algorithms developed in Chapter 3. For instance, crash testing, turbulence modeling and wave propagation are all potential application areas.

Stiffness matrix

On the direction of preconditioning stiffness matrix, the author would like to continue the investigation on the methodologies in order to tackle more complicated physical equations.

The first target would be the fully anisotropy with application on isogeometric analysis. Multigrid algorithms seem to be good candidates due to their success on elliptic problems. On the other hand, the tensor grid enabled by isogeometric analysis makes it very simple for multigrid algorithms to apply.

Moreover, the extension towards advection-diffusion equation is also very interesting. The extra advection operator leads to asymmetric linear system, boundary layer and instability, which makes it very challenging for preconditioning. A further extension will be the Navier-Stokes equations.

Helmholtz equation

Helmholtz equation, particularly when it comes to high wave numbers, remains invincible for decades. Its indefiniteness makes it a subtle object when it comes to preconditioning. Designing a good preconditioner for Helmholtz equation requires a radical understanding on virtually everything related: physics of wave propagation, mathematical analysis, numerical techniques and so on. The author hopes to further

investigate on this problem in the future due to its intriguing difficulty, as well as its broad application areas. However, no clear path has been realized so far.

Parallel computing

All the algorithms presented in this work are designed in a serial computing environment. Research on adapting these algorithms to parallel computing environment is necessary.

REFERENCES

[1] N. I. Achieser. *Theory of approximation*. Courier Dover Publications, 2011.

[2] B. Aksoylu, I. G. Graham, H. Klie, and R. Scheichl. Towards a rigorously justified algebraic preconditioner for high-contrast diffusion problems. *Computing and Visualization in Science*, 11(4-6):319–331, 2008.

[3] F. Auricchio, L. B. Da Veiga, A. Buffa, C. Lovadina, A. Reali, and G. Sangalli. A fully locking-free isogeometric approach for plane linear elasticity problems: a stream function formulation. *Computer Methods in Applied Mechanics and Engineering*, 197(1):160–172, 2007.

[4] O. Axelsson. A survey of preconditioned iterative methods for linear systems of algebraic equations. *BIT Numerical Mathematics*, 25(1):165–187, 1985.

[5] O. Axelsson. *Iterative solution methods*. Cambridge University Press, 1996.

[6] Z. Bai, J. Demmel, J. Dongarra, A. Ruhe, and H. Van Der Vorst. *Templates for the solution of algebraic eigenvalue problems: a practical guide*, volume 11. Society for Industrial Mathematics, 1987.

[7] A. Banegas. Fast Poisson solvers for problems with sparsity. *Mathematics of Computation*, pages 441–446, 1978.

[8] R. Bank. Efficient algorithms for solving tensor product finite element equations. *Numerische Mathematik*, 31(1):49–61, 1978.

[9] Y. Bazilevs, L. B. Da Veiga, J. Cottrell, T. Hughes, and G. Sangalli. Isogeometric analysis: approximation, stability and error estimates for h-refined meshes. *Mathematical Models and Methods in Applied Sciences*, 16(07):1031–1090, 2006.

[10] M. Benzi. Preconditioning techniques for large linear systems: a survey. *Journal of Computational Physics*, 182(2):418–477, 2002.

[11] G. Birkhoff, R. S. Varga, and D. Young. Alternating direction implicit methods. *Advances in Computers*, 3:189–273, 1962.

[12] J. M. Borwein and P. B. Borwein. The arithmetic-geometric mean and fast computation of elementary functions. *SIAM review*, 26(3):351–366, 1984.

[13] J. M. Borwein and P. B. Borwein. *Pi and the AGM: a study in the analytic number theory and computational complexity*. Wiley-Interscience, 1987.

[14] Y. Boubendir, X. Antoine, and C. Geuzaine. A quasi-optimal non-overlapping domain decomposition algorithm for the Helmholtz equation. *Journal of Computational Physics*, 231(2):262–280, 2012.

[15] D. Braess. *Finite elements: Theory, fast solvers, and applications in solid mechanics*. Cambridge University Press, 2001.

[16] A. Buffa, G. Sangalli, and R. Vázquez. Isogeometric analysis in electromagnetics: B-splines approximation. *Computer Methods in Applied Mechanics and Engineering*, 199(17):1143–1152, 2010.

[17] B. L. Buzbee. A fast Poisson solver amenable to parallel computation. *IEEE Transactions on Computers*, 22(8):793–796, 1973.

[18] C. Canuto, Y. Hussaini, and A. Quarteroni. *Spectral Methods: Evolution to Complex Geometries and Applications to Fluid Dynamics*. Scientific Computation. Springer-Verlag Berlin Heidelberg, 2007.

[19] E. W. Cheney. *Introduction to approximation theory*, volume 3. McGraw-Hill New York, 1966.

[20] P. Ciarlet. *The finite element method for elliptic problems*, volume 4. North Holland, 1978.

[21] J. Cottrell, T. Hughes, and Y. Bazilevs. *Isogeometric analysis: toward integration of CAD and FEA*. John Wiley & Sons Inc, 2009.

[22] J. Cottrell, T. Hughes, and A. Reali. Studies of refinement and continuity in isogeometric structural analysis. *Computer Methods in Applied Mechanics and Engineering*, 196(41):4160–4183, 2007.

[23] J. Cottrell, A. Reali, Y. Bazilevs, and T. Hughes. Isogeometric analysis of structural vibrations. *Computer Methods in Applied Mechanics and Engineering*, 195(41):5257–5296, 2006.

[24] L. B. da Veiga, A. Buffa, J. Rivas, and G. Sangalli. Some estimates for h–p–k-refinement in isogeometric analysis. *Numerische Mathematik*, 118(2):271–305, 2011.

[25] P. J. Davis and P. Rabinowitz. *Methods of Numerical Integration*. Courier Dover Publications, 2007.

[26] C. de Boor. Efficient computer manipulation of tensor products. *ACM Transactions on Mathematical Software (TOMS)*, 5(2):173–182, 1979.

[27] C. de Boor. *A Practical Guide to Splines*. Number v. 27 in Applied Mathematical Sciences. Springer, 2001.

[28] W. R. Dyksen. Tensor product generalized ADI methods for separable elliptic problems. *SIAM Journal on Numerical Analysis*, 24(1):59–76, 1987.

[29] Y. Efendiev, J. Galvis, and X.-H. Wu. Multiscale finite element methods for high-contrast problems using local spectral basis functions. *Journal of Computational Physics*, 230(4):937–955, 2011.

[30] H. C. Elman, D. J. Silvester, and A. J. Wathen. *Finite Elements and Fast Iterative Solvers: with Applications in Incompressible Fluid Dynamics: with Applications in Incompressible Fluid Dynamics*. OUP Oxford, 2005.

[31] B. Engquist and O. Runborg. Computational high frequency wave propagation. *Acta numerica*, 12(1):181–266, 2003.

[32] B. Engquist and L. Ying. Sweeping preconditioner for the Helmholtz equation: hierarchical matrix representation. *Communications on Pure and Applied Mathematics*, 64(5):697–735, 2011.

[33] B. Engquist and L. Ying. Sweeping preconditioner for the Helmholtz equation: moving perfectly matched layers. *Multiscale Modeling & Simulation*, 9(2):686–710, 2011.

[34] Y. A. Erlangga. Advances in iterative methods and preconditioners for the Helmholtz equation. *Archives of Computational Methods in Engineering*, 15(1):37–66, 2008.

[35] Y. A. Erlangga and R. Nabben. On a multilevel krylov method for the helmholtz equation preconditioned by shifted laplacian. *Electronic Transactions on Numerical Analysis*, 31(403-424):3, 2008.

[36] Y. A. Erlangga, C. Vuik, and C. Oosterlee. Comparison of multigrid and incomplete LU shifted-Laplace preconditioners for the inhomogeneous Helmholtz equation. *Applied Numerical Mathematics*, 56(5):648–666, 2006.

[37] Y. A. Erlangga, C. Vuik, and C. W. Oosterlee. On a class of preconditioners for solving the Helmholtz equation. *Applied Numerical Mathematics*, 50(3):409–425, 2004.

[38] A. Ern and J.-L. Guermond. Evaluation of the condition number in linear systems arising in finite element approximations. *ESAIM-Mathematical Modelling and Numerical Analysis*, 40(1):29–48, 2006.

[39] O. G. Ernst and M. J. Gander. Why it is difficult to solve Helmholtz problems with classical iterative methods. *Numerical Analysis of Multiscale Problems*, pages 325–363, 2012.

[40] G. E. Farin. *NURBS: from projective geometry to practical use*. AK Peters, Ltd., 1999.

[41] B. Fischer. *Polynomial Based Iteration Methods for Symmetric Linear Systems*, volume 68. Society for Industrial and Applied Mathematics, 2011.

[42] I. Fried and D. S. Malkus. Finite element mass matrix lumping by numerical integration with no convergence rate loss. *International Journal of Solids and Structures*, 11(4):461–466, 1975.

[43] J. Galvis and Y. Efendiev. Domain decomposition preconditioners for multiscale flows in high-contrast media. *Multiscale Modeling & Simulation*, 8(4):1461–1483, 2010.

[44] J. Galvis and Y. Efendiev. Domain decomposition preconditioners for multiscale flows in high contrast media: reduced dimension coarse spaces. *Multiscale Modeling & Simulation*, 8(5):1621–1644, 2010.

[45] G. H. Golub and D. P. O'Leary. Some history of the conjugate gradient and Lanczos algorithms: 1948-1976. *SIAM review*, 31(1):50–102, 1989.

[46] G. H. Golub and C. F. Van Loan. *Matrix Computations*, volume 3. Johns Hopkins University Press, 1996.

[47] A. Greenbaum. *Iterative Methods for Solving Linear Systems*, volume 17. Society for Industrial and Applied mathematics, 1987.

[48] L. Greengard and J.-Y. Lee. A direct adaptive poisson solver of arbitrary order accuracy. *Journal of Computational Physics*, 125(2):415–424, 1996.

[49] E. Hinton, T. Rock, and O. Zienkiewicz. A note on mass lumping and related processes in the finite element method. *Earthquake Engineering & Structural Dynamics*, 4(3):245–249, 1976.

[50] T. Hughes. *The Finite Element Method: linear static and dynamic finite element analysis*. Dover Publications, 2000.

[51] T. J. Hughes, J. A. Cottrell, and Y. Bazilevs. Isogeometric analysis: CAD, finite elements, NURBS, exact geometry and mesh refinement. *Computer Methods in Applied Mechanics and Engineering*, 194(39):4135–4195, 2005.

[52] G. Karniadakis and S. Sherwin. *Spectral/hp Element Methods for Computational Fluid Dynamics: Second Edition*. Numerical Mathematics and Scientific Computation. OUP Oxford, 2005.

[53] D. E. Keyes and W. D. Gropp. A comparison of domain decomposition techniques for elliptic partial differential equations and their parallel implementation. *SIAM Journal on Scientific and Statistical Computing*, 8(2):166–202, 1987.

[54] D. Kopriva. *Implementing Spectral Methods for Partial Differential Equations: Algorithms for Scientists and Engineers*. Scientific Computation. Springer Science+Business Media B.V., 2009.

[55] V. I. Krylov and A. H. Stroud. *Approximate Calculation of Integrals*. Courier Dover Publications, 1962.

[56] P. Kythe and M. Schäferkotter. *Handbook of Computational Methods for Integration*, volume 1. CRC Press, 2005.

[57] S. Lipton, J. A. Evans, Y. Bazilevs, T. Elguedj, and T. J. Hughes. Robustness of isogeometric structural discretizations under severe mesh distortion. *Computer Methods in Applied Mechanics and Engineering*, 199(5):357–373, 2010.

[58] C. D. Martin. *Higher-order Kronecker Products and Tensor Decompositions*. PhD thesis, Cornell University, 2005.

[59] MATLAB. Version 7.10.0 (R2010a). *The MathWorks Inc. Natick, Massachusetts*, 2010.

[60] A. Mayo. The fast solution of poisson's and the biharmonic equations on irregular regions. *SIAM Journal on Numerical Analysis*, 21(2):285–299, 1984.

[61] B. Parlett. *The Symmetric Eigenvalue Problem*. Classics in Applied Mathematics. Society for Industrial and Applied Mathematics, 1980.

[62] D. W. Peaceman and H. H. Rachford, Jr. The numerical solution of parabolic and elliptic differential equations. *Journal of the Society for Industrial & Applied Mathematics*, 3(1):28–41, 1955.

[63] V. Pereyra and G. Scherer. Efficient computer manipulation of tensor products with applications to multidimensional approximation. *Mathematics of Computation*, 27(123):595–605, 1973.

[64] L. Piegl. On NURBS: a survey. *Computer Graphics and Applications, IEEE*, 11(1):55–71, 1991.

[65] L. A. Piegl and W. Tiller. *The NURBS Book*. Springer Verlag, 1997.

[66] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes 3rd Edition: the art of scientific computing*. Cambridge University Press, 2007.

[67] A. Quarteroni, R. Sacco, and F. Saleri. *Numerical Mathematics*, volume 37. Springer, 2006.

[68] P. A. Regalia and M. K. Sanjit. Kronecker products, unitary matrices and signal processing applications. *SIAM review*, 31(4):586–613, 1989.

[69] D. F. Rogers. *An Introduction to NURBS: with historical perspective*. Morgan Kaufmann, 2000.

[70] Y. Saad. *Iterative Methods for Sparse Linear Systems*. Society for Industrial and Applied Mathematics, 2003.

[71] Y. Saad. *Numerical Methods for Large Eigenvalue Problems*. Classics in Applied Mathematics. Society for Industrial and Applied Mathematics, 2011.

[72] Y. Saad and H. A. Van Der Vorst. Iterative solution of linear systems in the 20th century. *Journal of Computational and Applied Mathematics*, 123(1):1–33, 2000.

[73] J. Schröder, U. Trottenberg, and K. Witsch. On fast Poisson solvers and applications. *Numerical Treatment of Differential Equations*, pages 153–187, 1978.

[74] A. Sheikh, D. Lahaye, and C. Vuik. On the convergence of shifted laplace preconditioner combined with multilevel deflation. *Numerical Linear Algebra with Applications*, 2013.

[75] W. Steeb and T. Shi. *Matrix Calculus and Kronecker Product with Applications and C++ Programs*. World Scientific, 1997.

[76] W. Strang and G. Fix. *An Analysis of the Finite Element Method*. Prentice-Hall, 1973.

[77] L. Trefethen and D. Bau. *Numerical Linear Algebra*. Miscellaneous Bks. Society for Industrial and Applied Mathematics, 1997.

[78] U. Trottenberg, C. W. Oosterlee, and A. Schüller. *Multigrid*. Academic Pr, 2001.

[79] D. Turkington. *Generalized Vectorization, Cross-Products, and Matrix Calculus*. Generalized Vectorization, Cross-products, and Matrix Calculus. Cambridge University Press, 2013.

[80] M. Van Gijzen, Y. Erlangga, and C. Vuik. Spectral analysis of the discrete Helmholtz operator preconditioned with a shifted Laplacian. *SIAM Journal on Scientific Computing*, 29(5):1942–1958, 2007.

[81] C. F. Van Loan. The ubiquitous Kronecker product. *Journal of Computational and Applied Mathematics*, 123(1):85–100, 2000.

[82] C. F. Van Loan and N. Pitsianis. Approximation with Kronecker products. Technical report, Cornell University, 1992.

[83] R. S. Varga. *Matrix Iterative Analysis*, volume 27. Springer, 2009.

[84] C. Vuik, A. Segal, and J. Meijerink. An efficient preconditioned CG method for the solution of a class of layered problems with extreme contrasts in the coefficients. *Journal of Computational Physics*, 152(1):385–403, 1999.

[85] E. L. Wachspress. Optimum alternating-direction-implicit iteration parameters for a model problem. *Journal of the Society for Industrial & Applied Mathematics*, 10(2):339–350, 1962.

[86] E. L. Wachspress. Extended application of alternating direction implicit iteration model problem theory. *Journal of the Society for Industrial & Applied Mathematics*, 11(4):994–1016, 1963.

[87] E. L. Wachspress. *Iterative Solution of Elliptic Systems*. Prentice-Hall, 1966.

[88] E. L. Wachspress and G. Habetler. An alternating-direction-implicit iteration technique. *Journal of the Society for Industrial & Applied Mathematics*, 8(2):403–423, 1960.

[89] A. Wathen. Realistic eigenvalue bounds for the Galerkin mass matrix. *IMA Journal of Numerical Analysis*, 7(4):449–457, 1987.

[90] A. Wathen. On relaxation of Jacobi iteration for consistent and generalized mass matrices. *Communications in Applied Numerical Methods*, 7(2):93–102, 1991.

[91] A. Wathen and T. Rees. Chebyshev semi-iteration in preconditioning for problems including the mass matrix. *Electronic Transactions on Numerical Analysis*, 34:125–135, 2009.

[92] D. Young. *Iterative Solution of Large Linear Systems*. Dover Books on Mathematics Series. Dover Publ., 2003.

APPENDICES

Appendix A

Mass matrix

A.1 About the two testing domains

The stretched rectangle and the perturbed rectangle shown in Figure 3.2 are mapped from square parametrical domains. The mappings can be represented as linear combinations of B-spline basis functions as follows:

$$\begin{aligned} x &= \sum_{i=1}^{N_\xi}\sum_{j=1}^{N_\eta} C_{ij}^x B_{ij}, \\ y &= \sum_{i=1}^{N_\xi}\sum_{j=1}^{N_\eta} C_{ij}^y B_{ij}. \end{aligned} \tag{A.1}$$

In (A.1), $B_{ij} = B_i^\xi B_j^\eta$ where $\{B^\xi\}_{i=1}^{N_\xi}$ and $\{B^\eta\}_{i=1}^{N_\eta}$ are the sets of 1D basis functions, respectively, on the horizontal direction and vertical direction of the parametrical domain. C_{ij}^x and C_{ij}^y are called the control points.

For the stretched rectangle, $\{B^\xi\}_{i=1}^{N_\xi}$ and $\{B^\eta\}_{i=1}^{N_\eta}$ are both defined by knot vector [0,0,1,1], which yields 2 linear basis functions. The corresponding control points are listed in Table A.1.

C_{ij}^x	B_1^ξ	B_2^ξ
B_1^η	0	1
B_2^η	0	5

C_{ij}^y	B_1^ξ	B_2^ξ
B_1^η	0	0
B_2^η	1	5

Table A.1: Control points for the stretched rectangle.

For the perturbed rectangle, $\{B^\xi\}_{i=1}^{N_\xi}$ and $\{B^\eta\}_{i=1}^{N_\eta}$ are defined by knot vector [0,0,0,1,2,3,4,5,5,5], which yields 7 quadratic basis functions with continuity 1 across the knots. The corresponding control points are listed in Table A.2 and Table A.3.

C_{ij}^x	B_1^ξ	B_2^ξ	B_3^ξ	B_4^ξ	B_5^ξ	B_6^ξ	B_7^ξ
B_1^η	-0.3142	0.7899	1.2358	2.0241	3.3905	4.5477	4.8925
B_2^η	0.4367	1.4064	2.2922	2.9479	3.9269	4.4844	5.3394
B_3^η	0.4665	1.5038	1.9556	2.7903	3.3913	4.6600	5.5869
B_4^η	0.4961	0.9212	1.2594	2.8849	4.1303	4.8637	5.3697
B_5^η	0.0596	0.7995	1.6432	2.3276	3.4461	4.9565	5.0942
B_6^η	0.4818	1.2737	1.8799	2.6568	3.9682	4.9108	5.3195
B_7^η	0.2539	1.1095	2.5355	3.4629	4.4784	4.6660	5.4954

Table A.2: Control points for the perturbed rectangle: x direction.

C_{ij}^y	B_1^ξ	B_2^ξ	B_3^ξ	B_4^ξ	B_5^ξ	B_6^ξ	B_7^ξ
B_1^η	0.6342	0.2239	0.1948	0.1525	-0.9304	-0.3156	-0.6165
B_2^η	1.0204	1.0852	0.7924	0.6019	0.1027	-0.0380	-0.0487
B_3^η	1.8142	1.5583	1.6155	1.7423	0.7937	0.7314	0.7311
B_4^η	2.8017	2.6391	2.7418	2.6091	1.6442	1.7947	2.0765
B_5^η	4.1169	3.2819	4.0476	3.7363	2.9908	3.1530	2.8399
B_6^η	4.7886	4.4857	4.7162	4.4456	3.8017	3.9477	3.6096
B_7^η	5.6561	5.2585	5.1921	5.2091	4.6306	4.6299	4.3596

Table A.3: Control points for the perturbed rectangle: y direction.

A.2 Additional numerical results for preconditioner M^{-1}

In this section, we present some additional numerical results to demonstrate the performance of preconditioner M^{-1}, in comparison with lumped-mass preconditioner. Conjugate gradient method with both preconditioners are applied on the stretched rectangle and perturbed rectangle shown in Figure 3.2. The iteration process is stopped when relative residual (in ℓ_2 norm) is less than $1e$-12.

Numbers of iteration steps required for convergence are presented in Table A.4

and Table A.5. In Table A.4, mesh size, h, and continuity order, c, are fixed while polynomial order, p, is varying; In Table A.5, mesh size and polynomial order are fixed while continuity order is varying.

	p	1	2	3	4	5	6	7	8
Stretched	M^{-1}	33	33	33	33	33	33	33	33
	Lumped-mass	31	86	200	431	766	798	932	828
Perturbed	M^{-1}	42	43	44	44	45	45	45	45
	Lumped-mass	31	86	196	406	748	742	930	1334

Table A.4: p-scaling: $N_{1D} = 2^7$, $c = 0$.

	c	7	6	5	4	3	2	1	0
Stretched	M^{-1}	33	33	33	33	33	33	33	33
	Lumped-mass	4040	7664	2983	3990	4757	3449	6059	828
Perturbed	M^{-1}	42	44	44	44	44	45	45	45
	Lumped-mass	4246	7357	2312	4050	4327	3103	4884	1334

Table A.5: c-scaling: $N_{1D} = 2^7$, $p = 8$.

A.3 Additional numerical results for preconditioner $(M^{\xi S})^{-1}$ and $(M^{\xi M})^{-1}$

In this section, we present some additional numerical results to demonstrate the performance of preconditioner $(M^{\xi S})^{-1}$ and $(M^{\xi M})^{-1}$, in comparison with lumped-mass preconditioner. Conjugate gradient method with these preconditioners are applied on the stretched rectangle and perturbed rectangle shown in Figure 3.2. The iteration process is stopped when relative residual (in ℓ_2 norm) is less than $1e$-12. Numbers of iteration steps required for convergence are presented in Table A.6 - A.11.

Table A.6 and Table A.7 are related with preconditioner $(M^{\xi S})^{-1}$. In Table A.6, mesh size, h, and continuity order, c, are fixed while polynomial order, p, is varying; In Table A.7, mesh size and polynomial order are fixed while continuity order is varying.

	p	1	2	3	4	5	6	7	8
Stretched	$(M^{\xi S})^{-1}$	4	4	5	5	5	5	5	5
	Lumped-mass	31	86	200	431	766	798	932	828
Perturbed	$(M^{\xi S})^{-1}$	7	7	8	8	8	9	9	9
	Lumped-mass	31	86	196	406	748	742	930	1334

Table A.6: p-scaling: $N_{1D} = 2^7$, $c = 0$.

	c	7	6	5	4	3	2	1	0
Stretched	$(M^{\xi S})^{-1}$	6	7	6	6	6	6	5	5
	Lumped-mass	4040	7664	2983	3990	4757	3449	6059	828
Perturbed	$(M^{\xi S})^{-1}$	10	16	15	14	14	14	12	9
	Lumped-mass	4246	7357	2312	4050	4327	3103	4884	1334

Table A.7: c-scaling: $N_{1D} = 2^7$, $p = 8$.

Table A.8 - A.11 are related with preconditioner $(M^{\xi M})^{-1}$. In Table A.8, polynomial order, p, and continuity order, c, are fixed while mesh size, h, is varying; In Table A.9, mesh size is fixed while polynomial order and continuity order are varying with the relation $c = p - 1$.

	N_{1D}	2^3	2^4	2^5	2^6	2^7	2^8	2^9	2^{10}
Stretched	$(M^{\xi M})^{-1}$	12	9	7	6	5	4	4	4
	Lumped-mass	168	418	516	413	320	254	206	151
Perturbed	$(M^{\xi M})^{-1}$	21	20	15	12	10	8	7	6
	Lumped-mass	289	512	463	380	306	245	200	152

Table A.8: h-scaling: $p = 4, c = 3$.

	p	1	2	3	4	5	6	7	8
Stretched	$(M^{\xi M})^{-1}$	4	4	4	4	4	4	4	4
	Lumped-mass	25	54	107	206	373	650	682	728
Perturbed	$(M^{\xi M})^{-1}$	6	6	7	7	7	8	8	8
	Lumped-mass	26	55	105	200	346	459	505	599

Table A.9: k-scaling: $N_{1D} = 2^9$, $c = p - 1$.

In Table A.10, mesh size, h, and continuity order, c, are fixed while polynomial order, p, is varying; In Table A.11, mesh size and polynomial order are fixed while continuity order is varying.

	p	1	2	3	4	5	6	7	8
Stretched	$(M^{\xi M})^{-1}$	4	5	5	5	5	5	5	5
	Lumped-mass	31	86	200	431	766	798	932	828
Perturbed	$(M^{\xi M})^{-1}$	8	8	8	8	8	8	9	9
	Lumped-mass	31	86	196	406	748	742	930	1334

Table A.10: p-scaling: $N_{1D} = 2^7$, $c = 0$.

	c	7	6	5	4	3	2	1	0
Stretched	$(M^{\xi M})^{-1}$	6	5	5	5	5	5	5	5
	Lumped-mass	4040	7664	2983	3990	4757	3449	6059	828
Perturbed	$(M^{\xi M})^{-1}$	13	11	10	9	9	9	9	9
	Lumped-mass	4246	7357	2312	4050	4327	3103	4884	1334

Table A.11: c-scaling: $N_{1D} = 2^7$, $p = 8$.

A.4 Complexity analysis

In this section, we continue our discussion on the computational cost at each iteration step for preconditioner $\left(M^{\xi S}\right)^{-1}$ or $\left(M^{\xi M}\right)^{-1}$, as a following-up of section 3.3.3. Here, we deal with more general cases where c is not necessarily equivalent to $p-1$.

Multiplication cost is heavily affected by the varying continuity, c, due to the change of sparsity in the matrix. Roughly, it requires $2(p+c+2)^2 \cdot N$ operation counts. Factorization cost and substitution cost are also affected. However, for simplicity, we still use the estimates obtained for the case where $c = p-1$, which are overestimates when $c < p-1$.

Again, since $(3p+1)(p+1) < 4(p+c+2)^2$, the factorization cost for $\left(M^{\xi S}\right)^{-1}$ or $\left(M^{\xi M}\right)^{-1}$ is still marginal, roughly amounts to 2 additional iteration cost.

Denote $r(p) = \dfrac{4(2p+1)\cdot N}{2(p+c+2)^2 \cdot N}$, $r(p)$ is an upper bound for the ratio of the substitution cost versus the multiplication cost. After some simple calculus, it can be shown that $r'(p) \leq 0$ for $p \geq 1$, where $r'(p)$ denotes the derivative of $r(p)$. Moreover, since $r(1) \leq \dfrac{2}{3}$, we have $r(p) \leq \dfrac{2}{3}$ for any $p \geq 1$ and $0 \leq c \leq p-1$, i.e., per iteration,

the additional computational cost does not exceed two thirds of the multiplication cost. Actually, when p or c is high, this ratio can be much smaller.

Recall the numerical results shown in Table A.6 - A.7 and Table A.10 - A.11, these additional costs are worthwhile considering the significant reduction of the number of iteration steps.

A.5 Additional numerical results for hybrid preconditioning

In this section, we present some additional numerical results to demonstrate the performance of the hybrid preconditioner presented in section 3.3.4, in comparison with preconditioner $(M^{\xi S})^{-1}$. Conjugate gradient method with both preconditioners are applied on the stretched rectangle and perturbed rectangle shown in Figure 3.2. The iteration process is stopped when relative residual (in ℓ_2 norm) is less than $1e$-12.

Numbers of iteration steps required for convergence are presented in Table A.12 and A.13. In Table A.12, mesh size, h, and continuity order, c, are fixed while polynomial order, p, is varying; In Table A.13, mesh size and polynomial order are fixed while continuity order is varying.

	p	1	2	3	4	5	6	7	8
Stretched	Hybrid	4	4	5	5	5	5	5	5
	$(M^{\xi S})^{-1}$	4	4	5	5	5	5	5	5
Perturbed	Hybrid	7	8	9	9	10	10	10	10
	$(M^{\xi S})^{-1}$	7	7	8	8	8	9	9	9

Table A.12: p-scaling: $N_{1D} = 2^7$, $c = 0$.

	c	7	6	5	4	3	2	1	0
Stretched	Hybrid	6	7	6	6	6	6	6	5
	$(M^{\xi S})^{-1}$	6	7	6	6	6	6	5	5
Perturbed	Hybrid	13	17	18	16	17	17	14	10
	$(M^{\xi S})^{-1}$	10	16	15	14	14	14	12	9

Table A.13: c-scaling: $N_{1D} = 2^7$, $p = 8$.

A.6 Quarter annulus

In this section, we show that the 2D mass matrix built on a quarter annulus with NURBS basis functions can still possess the Kronecker product property.

A quarter annulus, as shown in Figure A.1, can be mapped from a unit square.

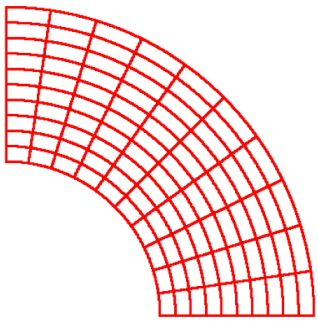


Figure A.1: Quarter annulus.

The mapping can be represented as a linear combination of NURBS basis functions as follows:

$$\begin{aligned} x &= \sum_{i=1}^{N_\xi}\sum_{j=1}^{N_\eta} C_{ij}^x \frac{B_{ij}}{W_N}, \\ y &= \sum_{i=1}^{N_\xi}\sum_{j=1}^{N_\eta} C_{ij}^y \frac{B_{ij}}{W_N} \end{aligned} \tag{A.2}$$

with $W_N = \sum_{i=1}^{N_\xi}\sum_{j=1}^{N_\eta} w_{ij}B_{ij}$. In (A.2), $B_{ij} = B_i^\xi B_j^\eta$ where $\{B^\xi\}_{i=1}^{N_\xi}$ and $\{B^\eta\}_{i=1}^{N_\eta}$ are the sets of 1D B-spline basis functions, respectively, on the horizontal direction and vertical direction of the parametrical domain. C_{ij}^x and C_{ij}^y are called the control points while w_{ij} are the weights for NURBS basis functions.

To construct the mapping for the quarter annulus, $\{B^\xi\}_{i=1}^{N_\xi}$ is defined by knot vector [0,0,0,1,1,1] while $\{B^\eta\}_{i=1}^{N_\eta}$ is defined by knot vector [0,0,1,1]. Corresponding control points and weights are listed in Table A.14.

C^x_{ij}	B^ξ_1	B^ξ_2	B^ξ_3
B^η_1	0	1	1
B^η_2	0	2	2

C^y_{ij}	B^ξ_1	B^ξ_2	B^ξ_3
B^η_1	1	1	0
B^η_2	2	2	0

w_{ij}	B^ξ_1	B^ξ_2	B^ξ_3
B^η_1	1	$\sqrt{2}/2$	1
B^η_2	1	$\sqrt{2}/2$	1

Table A.14: Control points and weights for quarter annulus.

After substituting the basis functions and coefficients into (A.2), x, y and W_N can be expressed explicitly as follows:

$$\begin{aligned} x &= \frac{(1+\eta)\left((\sqrt{2}-1)\xi-\sqrt{2}\right)\xi}{-1-(-2+\sqrt{2})\xi+(-2+\sqrt{2})\xi^2}, \\ y &= \frac{(1+\eta)(\xi-1)\left((-1+\sqrt{2})\xi+1\right)}{-1-(-2+\sqrt{2})\xi+(-2+\sqrt{2})\xi^2}, \\ W_N &= 1+(-2+\sqrt{2})\xi+(2-\sqrt{2})\xi^2. \end{aligned}$$

Moreover, determinant of Jacobian with respect to this mapping can be written as:

$$J = \frac{\sqrt{2}(1+\eta)}{(1+(-2+\sqrt{2})\xi+(2-\sqrt{2})\xi^2)}.$$

It is clear that W_N only depends on ξ while J can be separated as product of two functions, each of which depends on either only ξ or η. Therefore, the resulting 2D mass matrix still possesses the Kronecker product property.

Appendix B

Stiffness matrix

B.1 Upper bound of the error reduction rate

In this section, we briefly describe the upper bound of error reduction rate for ADI iterations with optimal acceleration parameters, as mentioned in section 4.1.2.

Suppose the set of optimal acceleration parameters $\{r^{(j)}\}_{j=1}^{k}$ has been chosen as the minimizer of optimization problem (4.10). Denote D_k^O as the observed error reduction rate, then we have, according to (4.20) and (4.21) in Theorem 4.3,

$$D_k^O \leq \max_{1\leq i\leq N} \left\{ \prod_{j=0}^{k} \left| \frac{r^{(j)} - \lambda_i^X}{r^{(j)} + \lambda_i^X} \right| \cdot \left| \frac{r^{(j)} - \lambda_i^Y}{r^{(j)} + \lambda_i^Y} \right| \right\} \leq \left\{ \max_{0<\alpha\leq x\leq \beta} \left| \prod_{j=0}^{k} \frac{r^{(j)} - x}{r^{(j)} + x} \right| \right\}^2 . \quad \text{(B.1)}$$

Define $d_k^P[\alpha, \beta]$ as:

$$d_k^P[\alpha, \beta] = \max_{0<\alpha\leq x\leq \beta} \left| \prod_{j=0}^{k} \frac{r^{(j)} - x}{r^{(j)} + x} \right|,$$

then $D_k^P = \left(d_k^P[\alpha, \beta]\right)^2$ is an upper bound of the observed error reduction rate D_k^O.

Denote $g_k\left(x; \{r^{(j)}\}_{j=1}^{k}\right)$ as:

$$g_k\left(x; \{r^{(j)}\}_{j=1}^{k}\right) = \left| \prod_{j=0}^{k} \frac{r^{(j)} - x}{r^{(j)} + x} \right|,$$

we have the following corollary based on Theorem 4.2:

Corollary B.1. *For any* $r^{(j)} \in \{r^{(j)}\}_{j=1}^{k}$, $\dfrac{\alpha\beta}{r^{(j)}}$ *also belongs to* $\{r^{(j)}\}_{j=1}^{k}$. *Moreover,*

$$g_k\left(x; \{r^{(j)}\}_{j=1}^{k}\right) = g_k\left(\frac{\alpha\beta}{x}; \{r^{(j)}\}_{j=1}^{k}\right).$$

Based on Corollary B.1, we also have the following corollary:

Corollary B.2.

$$d_{2k}^{P}[\alpha, \beta] = d_{k}^{P}\left[\sqrt{\alpha\beta}, \frac{\alpha+\beta}{2}\right]. \tag{B.2}$$

We omit the proof of Corollary B.1 and Corollary B.2. For detail, please refer to [83] or [85]. We also borrow the following Lemma from [83]:

Lemma B.1. *If* $k = 1$, *then* $r_1 = \sqrt{\alpha\beta}$ *is the single optimal parameter and correspondingly,*

$$d_{1}^{P}[\alpha, \beta] = \frac{1-\sqrt{\alpha\beta}}{1+\sqrt{\alpha\beta}}. \tag{B.3}$$

Corollary B.2 and Lemma B.1 together lead us to the following theorem:

Theorem B.1.

$$d_{2^m}[\alpha, \beta] = \frac{1-\sqrt{\alpha_m/\beta_m}}{1+\sqrt{\alpha_m/\beta_m}}, \tag{B.4}$$

where $\alpha_m = \sqrt{\alpha_{m-1}\beta_{m-1}}$ *and* $\beta_m = \dfrac{\alpha_{m-1}+\beta_{m-1}}{2}$ *with* $\alpha_0 = \alpha$ *and* $\beta_0 = \beta$.

Proof of Theorem B.1 is straightforward based on (B.2) and (B.3).

According to Theorem B.1, given α and β, $d_{2^m}[\alpha, \beta]$ only depends on α_m and β_m. Both sequences, $\{\alpha_m\}_{m=0}^{\infty}$ and $\{\beta_m\}_{m=0}^{\infty}$, monotonically converge to the Gauss arithmetic-geometric mean of α and β very quickly. See Table B.1 for an example where $\alpha = 1$ and $\beta = 10^7$. For more detail on Gauss arithmetic-geometric mean, one can consult [12, 13].

In Table B.1, we observe a quadratically decreasing behavior of $d_{2^m}[\alpha, \beta]$ and $D_{2^m}^{P} = \left(d_{2^m}[\alpha, \beta]\right)^2$, the upper bound of error reduction rate $D_{2^m}^{O}$, when α_m and β_m are close to each other. This is explained in the following.

m	α_m	β_m	$d_{2^m}[\alpha,\beta]$	$D_{2^m}^P$
0	1.0000000E+00	1.0000000E+07	9.99E-01	9.99E-01
1	3.1622777E+03	5.0000005E+06	9.51E-01	9.04E-01
2	1.2574335E+05	2.5015814E+06	6.34E-01	4.02E-01
3	5.6085401E+05	1.3136624E+06	2.10E-01	4.39E-02
4	8.5835471E+05	9.3725819E+05	2.20E-02	4.83E-04
5	8.9693923E+05	8.9780645E+05	2.42E-04	5.84E-08
6	8.9737274E+05	8.9737284E+05	2.92E-08	8.52E-16

Table B.1: Quadratic convergence behavior.

For $\alpha_m = \mu > 0$ and $\beta_m = \mu(1+\delta)$ where $\delta > 0$ is a small number comparing with 1, it can be shown that on one hand:

$$d_{2^m}[\alpha,\beta] \;\leq\; \frac{1-\sqrt{1/(1+\delta)}}{1+\sqrt{1/(1+\delta)}}\left(1+\sqrt{\frac{1}{1+\delta}}\right)^2 \;=\; 1-\frac{1}{1+\delta} \;\leq\; \delta;$$

on the other hand:

$$d_{2^m}[\alpha,\beta] \;=\; \frac{\sqrt{1+\delta}-1}{\sqrt{1+\delta}+1} \;=\; \frac{\delta}{\left(\sqrt{1+\delta}+1\right)^2} \;\geq\; C_1\delta.$$

Therefore, $d_{2^m}[\alpha,\beta] = C_m\delta$ for some constant C_m.

Similar results exist for $d_{2^{m+1}}[\alpha,\beta]$ where α_{m+1} and β_{m+1} can be expressed as:

$$\alpha_{m+1} = \sqrt{\alpha_m\beta_m} = \mu(1+\delta)^{\frac{1}{2}} \;\text{ and }\; \beta_{m+1} = \frac{\alpha_m+\beta_m}{2} = \mu(1+\frac{\delta}{2}).$$

On one hand, we have:

$$d_{2^{m+1}}[\alpha,\beta] \;\leq\; 1-\frac{\alpha_{m+1}}{\beta_{m+1}} \;\leq\; 1-\left(\frac{\alpha_{m+1}}{\beta_{m+1}}\right)^2 \;=\; 1-\frac{(1+\delta)}{\left(1+\frac{\delta}{2}\right)^2} \;\leq\; C_2\delta^2;$$

On the other hand, we have:

$$\begin{aligned} d_{2^{m+1}}[\alpha,\beta] &= \frac{1-\sqrt{\alpha_{m+1}/\beta_{m+1}}}{1+\sqrt{\alpha_{m+1}/\beta_{m+1}}}\cdot\frac{1+\sqrt{\alpha_{m+1}/\beta_{m+1}}}{1+\sqrt{\alpha_{m+1}/\beta_{m+1}}} \;\geq\; \frac{1}{4}\left(1-\frac{\alpha_{m+1}}{\beta_{m+1}}\right) \\ &= \frac{1}{4}\cdot\frac{\beta_{m+1}-\alpha_{m+1}}{\beta_{m+1}}\cdot\frac{\beta_{m+1}+\alpha_{m+1}}{\beta_{m+1}+\alpha_{m+1}} \;\geq\; C_3\delta^2. \end{aligned}$$

Therefore, $d_{2^{m+1}}[\alpha,\beta] = C_{m+1}\delta^2$ for some constant C_{m+1}.

Now, it is easy to understand the quadratically decreasing behavior of $d_{2^m}[\alpha,\beta]$ and $D^P_{2^m}$ in Table B.1 since $d_{2^m}[\alpha,\beta] = C_m\delta$ and $d_{2^{m+1}}[\alpha,\beta] = C_{m+1}\delta^2$.

B.2 Symmetry and positive definiteness of preconditioner $\left(\bar{K}^X + \bar{K}^Y\right)^{-1}$

To clarify, by preconditioner $\left(\bar{K}^X + \bar{K}^Y\right)^{-1}$, we actually mean the preconditioner induced by the ADI iteration process applied on matrix $\left(\bar{K}^X + \bar{K}^Y\right)$ with a zero starting point. In this section, we call it preconditioner $\left(\bar{K}^X + \bar{K}^Y\right)^{-1}$ for simplicity, despite of the potential ambiguity.

We use $\bar{K}^X$ to approximate $\tilde{K}^X$ and $\bar{K}^Y$ to approximate $\tilde{K}^Y$, where $\bar{K}^X$ and $\bar{K}^Y$ are symmetric. With a set of positive acceleration parameters $\{r^{(j)}\}_{j=0}^k$, we apply preconditioner $\left(\bar{K}^X + \bar{K}^Y\right)^{-1}$ in the following manner.

First a forward cycle:

$$\begin{aligned}
(r^{(0)}M + \bar{K}^X)b^{(0+\frac{1}{2})} &= (r^{(0)}M - \bar{K}^Y)b^{(0)} + \mathcal{F},\\
(r^{(0)}M + \bar{K}^Y)b^{(0+1)} &= (r^{(0)}M - \bar{K}^X)b^{(0+\frac{1}{2})} + \mathcal{F},\\
&\vdots\\
(r^{(k)}M + \bar{K}^X)b^{(k+\frac{1}{2})} &= (r^{(k)}M - \bar{K}^Y)b^{(k)} + \mathcal{F},\\
(r^{(k)}M + \bar{K}^Y)b^{(k+1)} &= (r^{(k)}M - \bar{K}^X)b^{(k+\frac{1}{2})} + \mathcal{F}.
\end{aligned} \tag{B.5}$$

Then a backward cycle:

$$\begin{aligned}
(r^{(k)}M + \bar{K}^Y)\tilde{b}^{(0+\frac{1}{2})} &= (r^{(k)}M - \bar{K}^X)b^{(k+1)} + \mathcal{F},\\
(r^{(k)}M + \bar{K}^X)\tilde{b}^{(0+1)} &= (r^{(k)}M - \bar{K}^Y)\tilde{b}^{(0+\frac{1}{2})} + \mathcal{F},\\
&\vdots\\
(r^{(0)}M + \bar{K}^Y)\tilde{b}^{(k+\frac{1}{2})} &= (r^{(0)}M - \bar{K}^X)\tilde{b}^{(k)} + \mathcal{F},\\
(r^{(0)}M + \bar{K}^X)\tilde{b}^{(k+1)} &= (r^{(0)}M - \bar{K}^Y)\tilde{b}^{(k+\frac{1}{2})} + \mathcal{F}.
\end{aligned} \tag{B.6}$$

$b^{(0)}$ and $\tilde{b}^{(k+1)}$ are the initial guess and final result of the above iteration process, respectively.

Regarding symmetry of the preconditioner, we present a proof for a more general case in the context of matrix splitting algorithms.

Consider linear system $Ab = \mathcal{F}$ where symmetric positive definite matrix A can be split as $A = D - E - F$ such that $D - E$ and $D - F$ are invertible while D, E and F are all symmetric. We start from the following simple iterative process:

Forward cycle:

$$\begin{aligned} (D-E)b^{(\frac{1}{2})} &= Fb^{(0)} + \mathcal{F}, \\ (D-F)b^{(1)} &= Eb^{(\frac{1}{2})} + \mathcal{F}. \end{aligned} \tag{B.7}$$

Backward cycle:

$$\begin{aligned} (D-F)\tilde{b}^{(\frac{1}{2})} &= Eb^{(1)} + \mathcal{F}, \\ (D-E)\tilde{b}^{(1)} &= F\tilde{b}^{(\frac{1}{2})} + \mathcal{F}. \end{aligned} \tag{B.8}$$

The forward cycle gives us:

$$b^{(1)} = (D-F)^{-1}E(D-E)^{-1}Fb^{(0)} + \left[(D-F)^{-1}E(D-E)^{-1} + (D-F)^{-1}\right]\mathcal{F}. \tag{B.9}$$

Define $P_1 = (D-F)^{-1}E(D-E)^{-1}F$ and $Q_1 = (D-F)^{-1}E(D-E)^{-1} + (D-F)^{-1}$, the following relation between P_1 and Q_1 can be easily verified:

$$I - P_1 = Q_1 A, \tag{B.10}$$

where I stands for the identity matrix at compatible size. Therefore, (B.9) can be simplified as:

$$b^{(1)} = P_1 b^{(0)} + Q_1 \mathcal{F}. \tag{B.11}$$

Moreover, after some simple linear algebra, Q_1 can be written in the following form:

$$Q_1 = (D-F)^{-1}D(D-E)^{-1}. \tag{B.12}$$

Analogically, the backward cycle gives us:

$$\tilde{b}^{(1)} = P_2 b^{(1)} + Q_2 \mathcal{F}, \tag{B.13}$$

where $P_2 = (D-E)^{-1}F(D-F)^{-1}E$ and $Q_2 = (D-E)^{-1}F(D-F)^{-1} + (D-E)^{-1}$. Moreover, P_2 and Q_2 satisfy a similar relation to (B.10):

$$I - P_2 = Q_2 A \tag{B.14}$$

while Q_2 can be written in the following form:

$$Q_2 = (D-E)^{-1}D(D-F)^{-1}. \tag{B.15}$$

Recalling that D, E and F are all symmetric, we have $Q_1^T = Q_2$.

Combining (B.11) and (B.13) together, we have

$$\tilde{b}^{(1)} = P_2 P_1 b^{(0)} + (P_2 Q_1 + Q_2)\mathcal{F}. \tag{B.16}$$

Denote $P = P_2 P_1$ and $Q = P_2 Q_1 + Q_2$, it is easy to derive the following relation between P and Q from (B.10) and (B.14):

$$I - P = QA. \tag{B.17}$$

Moreover, according to (B.12), (B.15) and the fact that D, E and F are all symmetric, we have:

$$Q = Q^T$$

since $Q = P_2 Q_1 + Q_2 = Q_1 + Q_2 - Q_2 A Q_1$.

To sum up, for iterative process defined by forward cycle (B.7) and backward cycle

(B.8), we have the following relation between the inputs $b^{(0)}$, $\mathcal{F}$ and the output $\tilde{b}^{(1)}$:

$$\tilde{b}^{(1)} = Pb^{(0)} + Q\mathcal{F}, \tag{B.18}$$

where $I - P = QA$ and $Q = Q^T$. Relation (B.18) will serve as the basis of the induction proof for the following more general iterative process.

Forward cycle:

$$\begin{aligned}
(D^{(0)} - E^{(0)})b^{(\frac{1}{2})} &= F^{(0)}b^{(0)} + \mathcal{F}, \\
(D^{(0)} - F^{(0)})b^{(1)} &= E^{(0)}b^{(\frac{1}{2})} + \mathcal{F}, \\
&\vdots \\
(D^{(k)} - E^{(k)})b^{(k+\frac{1}{2})} &= F^{(k)}b^{(k)} + \mathcal{F}, \\
(D^{(k)} - F^{(k)})b^{(k+1)} &= E^{(k)}b^{(k+\frac{1}{2})} + \mathcal{F}.
\end{aligned} \tag{B.19}$$

Backward cycle:

$$\begin{aligned}
(D^{(k)} - F^{(k)})\tilde{b}^{(\frac{1}{2})} &= E^{(k)}b^{(k+1)} + \mathcal{F}, \\
(D^{(k)} - E^{(k)})\tilde{b}^{(1)} &= F^{(k)}\tilde{b}^{(\frac{1}{2})} + \mathcal{F}, \\
&\vdots \\
(D^{(0)} - F^{(0)})\tilde{b}^{(k+\frac{1}{2})} &= E^{(0)}\tilde{b}^{(k)} + \mathcal{F}, \\
(D^{(0)} - E^{(0)})\tilde{b}^{(k+1)} &= F^{(0)}\tilde{b}^{(k+\frac{1}{2})} + \mathcal{F}.
\end{aligned} \tag{B.20}$$

Theorem B.2. *For the iterative process defined by* (B.19) *and* (B.20)*, the following relation between the inputs $b^{(0)}$, $\mathcal{F}$ and the output $\tilde{b}^{(k+1)}$ holds:*

$$\tilde{b}^{(k+1)} = Pb^{(0)} + Q\mathcal{F}, \tag{B.21}$$

for some P and Q that satisfy $I - P = QA$ and $Q = Q^T$.

Proof. The assertion is certainly true for the case $k = 0$ according to (B.18). We

prove the general case by induction.

Suppose the assertion is true for iterative process composing $k-1$ steps in both forward and backward cycles, i.e., we have

$$\tilde{b}^{(k)} = P_0 b^{(1)} + Q_0 \mathcal{F} \tag{B.22}$$

for some P_0 and Q_0 that satisfy $I - P_0 = Q_0 A$ and $Q_0 = Q_0^T$.

Analogical to (B.10), with the following definitions: $P_1 = (D^{(0)} - F^{(0)})^{-1} E^{(0)} (D^{(0)} - E^{(0)})^{-1} F^{(0)}$ and $Q_1 = (D^{(0)} - F^{(0)})^{-1} D^{(0)} (D^{(0)} - E^{(0)})^{-1}$, we have

$$b^{(1)} = P_1 b^{(0)} + Q_1 \mathcal{F} \tag{B.23}$$

from the first pair of equations in the forward cycle (B.19). Moreover, we have $I - P_1 = Q_1 A$.

Similarly, define $P_2 = (D^{(0)} - E^{(0)})^{-1} F^{(0)} (D^{(0)} - F^{(0)})^{-1} E^{(0)}$ and $Q_2 = (D^{(0)} - E^{(0)})^{-1} D^{(0)} (D^{(0)} - F^{(0)})^{-1}$, we have

$$\tilde{b}^{(k+1)} = P_2 \tilde{b}^{(k)} + Q_2 \mathcal{F} \tag{B.24}$$

from the last pair of equations in the backward cycle (B.20). Moreover, we have $I - P_2 = Q_2 A$ and $Q_1 = Q_2^T$.

Combining (B.22), (B.23) and (B.24) together, we have the following relation between $b^{(0)}$, $\mathcal{F}$ and $\tilde{b}^{(k+1)}$:

$$\tilde{b}^{(k+1)} = P_2 P_0 P_1 b^{(0)} + (P_2 P_0 Q_1 + P_2 Q_0 + Q_2) \mathcal{F}. \tag{B.25}$$

Define $P = P_2 P_0 P_1$ and $Q = P_2 P_0 Q_1 + P_2 Q_0 + Q_2$, it is easy to verify that

$$Q = Q_0 + Q_1 + Q_2 - Q_2 A Q_1 - Q_0 A Q_1 - Q_2 A Q_0 + Q_2 A Q_0 A Q_1. \tag{B.26}$$

Since we have $Q_1^T = Q_2$ and $Q_0 = Q_0^T$, it is easy to see that $Q = Q^T$. Moreover, it is also easy to verify that $I - P = QA$ by substituting relations $I - P_0 = Q_0 A$, $I - P_1 = Q_1 A$ and $I - P_2 = Q_2 A$ into the definition of P.

Thus, by induction, the assertion in the theorem is true for any integer $k \geq 0$. □

From Theorem B.2, we naturally have the following corollary:

Corollary B.3. *Applying the iterative process defined by forward cycle* (B.5) *and backward cycle* (B.6) *with a zero starting point is equivalent to applying a symmetric matrix.*

Proof. With the following definitions

$$\begin{aligned} D^{(j)} &= 2r^{(j)}M, \\ E^{(j)} &= r^{(j)}M - \bar{K}^X, \\ F^{(j)} &= r^{(j)}M - \bar{K}^Y \end{aligned} \tag{B.27}$$

for $j = 0, \ldots, k$, matrix $\left(\bar{K}^X + \bar{K}^Y\right)^{-1}$ can be split as $D^{(j)} - E^{(j)} - F^{(j)}$. Since $D^{(j)}$, $E^{(j)}$ and $F^{(j)}$ are all symmetric, the iterative process defined by (B.5) and (B.6) can be viewed as a special case of the iterative process defined by (B.19) and (B.20).

Applying Theorem B.2, we have the following relation between the inputs and output of the iterative process defined by (B.5) and (B.6):

$$\tilde{b}^{(k+1)} = Pb^{(0)} + Q\mathcal{F}, \tag{B.28}$$

where Q is a symmetric matrix. If the starting point $b^{(0)}$ is zero, (B.28) leads us to $\tilde{b}^{(k+1)} = Q\mathcal{F}$. Therefore, applying the iterative process defined by (B.5) and (B.6) with $b^{(0)} = 0$ is equivalent to applying matrix Q, which is symmetric. □

So far now, we have proven the symmetry of preconditioner $\left(\bar{K}^X + \bar{K}^Y\right)^{-1}$. Next, we want to show the positive definiteness of this preconditioner.

Theorem B.3. *For iterative process defined by* (B.5) *and* (B.6)*, matrix* Q *in* (B.28) *is always positive definite for any integer* $k \geq 0$.

Proof. For the case $k = 0$, with definitions $Q_1 = (D^{(0)} - F^{(0)})^{-1} D^{(0)} (D^{(0)} - E^{(0)})^{-1}$ and $Q_2 = (D^{(0)} - E^{(0)})^{-1} D^{(0)} (D^{(0)} - F^{(0)})^{-1}$, we have

$$\begin{aligned} Q &= Q_1 + Q_2 - Q_2 A Q_1 \\ &= Q_2 Q_2^{-1} Q_1 + Q_2 Q_1^{-1} Q_1 - Q_2 A Q_1 \\ &= Q_2 \left(Q_2^{-1} + Q_1^{-1} - A \right) Q_1. \end{aligned} \tag{B.29}$$

The invertibility of Q_1 and Q_2 are obvious once we substitute (B.27) into the definition of Q_1 and Q_2:

$$\begin{aligned} Q_1 &= \left(r^{(0)} M + \bar{K}^Y \right)^{-1} \left(2 r^{(0)} M \right) \left(r^{(0)} M + \bar{K}^X \right)^{-1}, \\ Q_2 &= \left(r^{(0)} M + \bar{K}^X \right)^{-1} \left(2 r^{(0)} M \right) \left(r^{(0)} M + \bar{K}^Y \right)^{-1}. \end{aligned}$$

Recall that $A = \bar{K}^X + \bar{K}^Y$, for $\left(Q_2^{-1} + Q_1^{-1} - A \right)$, we have

$$\begin{aligned} & \left(Q_2^{-1} + Q_1^{-1} - A \right) \\ = \quad & \left(r^{(0)} M + \bar{K}^Y \right) \left(2 r^{(0)} M \right)^{-1} \left(r^{(0)} M + \bar{K}^X \right) - \left(r^{(0)} M + \bar{K}^Y \right) \\ + \quad & \left(r^{(0)} M + \bar{K}^X \right) \left(2 r^{(0)} M \right)^{-1} \left(r^{(0)} M + \bar{K}^Y \right) - \left(r^{(0)} M + \bar{K}^X \right) \\ + \quad & 2 r^{(0)} M \\ = \quad & \left(r^{(0)} M + \bar{K}^Y \right) \left[\left(2 r^{(0)} M \right)^{-1} \left(r^{(0)} M + \bar{K}^X \right) - I \right] \\ + \quad & \left(r^{(0)} M + \bar{K}^X \right) \left[\left(2 r^{(0)} M \right)^{-1} \left(r^{(0)} M + \bar{K}^Y \right) - I \right] \\ + \quad & 2 r^{(0)} M \\ = \quad & \left(r^{(0)} M + \bar{K}^Y \right) \left[\frac{1}{2 r^{(0)}} M^{-1} \bar{K}^X - \frac{1}{2} I \right] \\ + \quad & \left(r^{(0)} M + \bar{K}^X \right) \left[\frac{1}{2 r^{(0)}} M^{-1} \bar{K}^Y - \frac{1}{2} I \right] \\ + \quad & 2 r^{(0)} M \\ = \quad & r^{(0)} M + \frac{1}{2 r^{(0)}} \bar{K}^X M^{-1} \bar{K}^Y + \frac{1}{2 r^{(0)}} \bar{K}^X M^{-1} \bar{K}^Y \end{aligned}$$

It is obvious now that as long as M is positive definite while $\bar{K}^X$ and $\bar{K}^Y$ are semi

positive definite, $\left(Q_2^{-1}+Q_1^{-1}-A\right)$ is positive definite. Moreover, Q is congruent with $\left(Q_2^{-1}+Q_1^{-1}-A\right)$ since $Q_1=Q_2^T$. Thus, Q is also positive definite, i.e., the assertion is true for $k=0$.

Suppose the assertion is also true for iterative process composing $k-1$ steps in both forward and backward cycles, i.e., Q_0 in (B.22) is positive definite. Then, for the iterative process composing k steps, according to (B.26), we have

$$Q \;=\; (Q_1+Q_2-Q_2AQ_1)+(Q_0-Q_0AQ_1-Q_2AQ_0+Q_2AQ_0AQ_1)\,.$$

From the proof for the $k=0$ case, we know that $(Q_1+Q_2-Q_2AQ_1)$ is positive definite. Moreover, we have

$$(Q_0-Q_0AQ_1-Q_2AQ_0+Q_2AQ_0AQ_1)=(I-Q_2A)Q_0(I-AQ_1).$$

Since $(I-Q_2A)^T=(I-AQ_1)$, based on the assumption that Q_0 is positive definite, $(I-Q_2A)Q_0(I-AQ_1)$ is at least semi positive definite. Thereby, Q is positive definite.

Thus, by induction, the assertion in the theorem is true for any integer $k\geq 0$. □

B.3 Formulae for coefficients of the numerical examples in section 4.2.3

Figure 4.1 (Orthotropic):

$$\begin{aligned}
\kappa_{11} &= 10^0\Big\{1+0.1\Big[1+0.99\cos\big(50(x-y)\big)\Big]+1+0.1\Big[1+0.99\cos\big(50(x+y)\big)\Big]\\
&\quad+\Big[1.01+\frac{e^{100(x+y-1)-1}}{e^{100(x+y-1)+1}}\Big]\Big\};\\
\kappa_{22} &= 10^5\Big\{1+0.1\Big[1+0.99\cos\big(50(x-y)\big)\Big]+1+0.1\Big[1+0.99\cos\big(50(x+y)\big)\Big]\\
&\quad+\Big[1.01+\frac{e^{100(1-x-y)-1}}{e^{100(1-x-y)+1}}\Big]\Big\}.
\end{aligned}$$

Figure 4.2 (Low frequency oscillation):

$$\kappa_{11} = \Big[1 + 0.99\cos\big(5(x-y)\big)\Big] + \Big[1 + 0.99\sin\big(5(x+y)\big)\Big];$$
$$\kappa_{22} = \Big[1 + 0.99\sin\big(5(x-y)\big)\Big] + \Big[1 + 0.99\cos\big(5(x+y)\big)\Big].$$

Figure 4.3 (Gaussian spikes):

$$\kappa_{11} = \sum_{i=1}^{5} Ae^{-\left(a(x-x_i)^2 + c(y-y_i)^2\right)}$$

with $A = 100, a = 75, c = 75$ and $(x_1, y_1) = (0.25, 0.25)$; $(x_2, y_2) = (0.25, 0.75)$; $(x_3, y_3) = (0.5, 0.5)$; $(x_4, y_4) = (0.75, 0.25)$; $(x_5, y_5) = (0.75, 0.75)$.

$$\kappa_{22} = \sum_{i=1}^{5} Ae^{-\left(a(x-x_i)^2 + c(y-y_i)^2\right)}$$

with $A = 100, a = 150, c = 150$ and $(x_1, y_1) = (0.5, 0.25)$; $(x_2, y_2) = (0.5, 0.75)$; $(x_3, y_3) = (0.5, 0.5)$; $(x_4, y_4) = (0.75, 0.5)$; $(x_5, y_5) = (0.25, 0.5)$.

Appendix C

Helmholtz equation

C.1 A better preconditioner

As mentioned in Remark 5.4, M^{-1} can be expressed as $M^{-1} = VI^{-1}V^T$ while it is possible to have a better preconditioner than M^{-1} by replacing I^{-1} with a different diagonal matrix. We explore this idea in this section.

Denote this diagonal matrix as $\mathcal{D}_{pre}^{-1}$. Its corresponding preconditioner can be written as $VD_{pre}^{-1}V^T$. Similar to relation (5.12), with some simple algebra, one can find the following relation for the case of left preconditioning with $VD_{pre}^{-1}V^T$:

$$\epsilon^{(m)} = p^{(m)}(\mathcal{D}_{pre}^{-1}\mathcal{D})\epsilon^{(0)}. \tag{C.1}$$

According to (C.1), if $\mathcal{D}_{pre}^{-1}$ can approximate $\mathcal{D}^{-1}$ better, the performance of GMRES is very likely to be improved. However, it is also important that the Kronecker product structure in the preconditioner is still maintained.

For instance, we can build $\mathcal{D}_{pre}$ in the following manner. First, find the smallest and largest unpicked entries in D^x and D^y, denotes as λ_{min}^x, λ_{max}^x, λ_{min}^y and λ_{max}^y, respectively; Then, build diagonal matrices D_C^x and D_C^y such that their entries are roots of Chebyshev polynomials on interval $\left[\sqrt{\lambda_{min}^x\lambda_{min}^y}, \sqrt{\lambda_{max}^x\lambda_{max}^y}\right]$ with order N_x and N_y, respectively; Finally, $\mathcal{D}_{pre}$ is built as $\mathcal{D}_{pre} = D_C^x \otimes D_C^y$.

Denote $M_C^{-1} = (M^y)_C^{-1} \otimes (M_C^x)^{-1}$, where $(M_C^x)^{-1} = V^x \left(D_C^x\right)^{-1} (V^x)^T$ and $(M_C^y)^{-1} = V^y \left(D_C^y\right)^{-1} (V^y)^T$, applying $V\mathcal{D}_{pre}^{-1}V^T$ is equivalent to applying M_C^{-1}, which is cheap due to its Kronecker product structure. Numerical results comparing the performances of GMRES(20) with M^{-1} and M_C^{-1} are shown in Table C.1. Maximum number of restarts is set as 10 while tolerance of relative residual is set as $1e$-6.

	$k(h)$	10(16)	20(32)	40(64)	80(128)	160(256)	320(512)
	e_p	5.26E-02	9.49E-02	2.79E-01	1.58E-02	3.21E-02	1.32E-02
M^{-1}	e	7.36E-07	2.55E-06	3.66E-04	2.02E-05	9.25E-04	2.33E-03
	N_{iter}	3(19)	6(5)	10(20)	10(20)	10(20)	10(20)
M_C^{-1}	e	1.45E-07	3.41E-07	1.86E-06	1.46E-07	2.12E-05	3.48E-04
	N_{iter}	2(5)	3(5)	9(4)	9(4)	10(20)	10(20)

Table C.1: Possibility of a better preconditioner.

From Table C.1, we can see that GMRES with M_C^{-1} indeed performs slightly better than with M^{-1}.

C.2 About the physical domain in Figure 5.2

Formulae of functions representing the five curved boundaries and interfaces in Figure 5.2 , from bottom to top, are listed in the following:

$$\begin{aligned}
f_1 &= 0.00 + 0.050\sin(2\pi\xi);\\
f_2 &= 0.25 + 0.025\sin(3\pi\xi + 0.5\pi);\\
f_3 &= 0.50 + 0.050\sin(4\pi\xi + \pi);\\
f_4 &= 0.75 + 0.025\sin(3\pi\xi + 1.5\pi);\\
f_5 &= 1.00 + 0.050\sin(2\pi\xi).
\end{aligned}$$

The mapping from the parametrical domain to the physical domain is constructed as:

$$x = \xi,$$

$$y = \begin{cases} 4\big((0.25-\eta)f_1+\eta f_2\big) & \text{if} \quad 0 \le \eta \le 0.25; \\ 4\big((0.5-\eta)f_2+(\eta-0.25)f_3\big) & \text{if} \quad 0.25 < \eta \le 0.5; \\ 4\big((0.75-\eta)f_3+(\eta-0.5)f_4\big) & \text{if} \quad 0.5 < \eta \le 0.75; \\ 4\big((1-\eta)f_4+(\eta-0.75)f_5\big) & \text{if} \quad 0.75 < \eta \le 1. \end{cases}$$

where ξ and η denote the horizontal and vertical variables in the parametrical domain while x and y denote their counterparts in the physical domain. Constructed in this way, the five horizontal straight lines in the parametrical domain are mapped into the five curved ones in the physical domain, shown in Figure 5.2.

Papers submitted and under preparation

• L. Gao, V. M. Calo, "Fast Isogeometric solvers for explicit dynamics", *Submitted to Computer Methods in Applied Mechanics and Engineering*, Oct. 2013.

• L. Gao, V. M. Calo, "Preconditioning the 2D diffusion equation with orthotropic heterogeneous coefficients based on the alternating direction implicit algorithm", *Under preparation for Journal of Computational and Applied Mathematics.*

• L. Gao, V. M. Calo, "Isogeometric spectral element method. Methodology", *Under preparation for Computer Methods in Applied Mechanics and Engineering.*