ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Partition of unity method Based Wave-Ray Multigrid

Master Thesis

Liaowang Huang

April 15, 2021

Advisors: Prof. Dr. Ralf Hiptmair

Department of Mathematics, ETH Zürich

Abstract

The standard multigrid methods are unable to solve the Helmholtz problem efficiently, because they can't eliminate some problematic error components, for these components are almost invisible for local relaxation on fine grids and too oscillatory to be approximated on coarse grids. The wave-ray methods represent the problematic components as the product of a high-frequency Fourier component and a smooth ray function, then another ray cycle is employed to deal with the ray function. This project aims at approximating the problematic error components in plane wave PUM spaces, which have good local approximation properties regarding the problematic error components. This project also investigate the effect of using GMRES and local smoothing with impedance boundary condition on coarse grids.

Acknowledgements

First and foremost, I want to thank my supervisor Prof. Dr. Ralf Hiptmair for the valuable discussions and constructive feedback on my work. I sincerely appreciate all his contributions of time, ideas and encouragement in this period. I also would like to thank my parents and family for their unconditioned support and patience during my study in Switzerland. Last, I thank my flatmates and all my friends for their support and company, life would be less fun without them.

Contents

Chapter 1

Introduction

This master thesis project aims at solving the Helmholtz boundary value problem posed on a two-dimensional domain $\Omega \subset \mathbb{R}^2$:

$$\Delta u + k^2 u = 0 \quad \text{in } \Omega$$

with suitable boundary conditions. Discretization of the Helmholtz equation by finite elements or finite differences leads to a linear system of equations

$$A\vec{\mu} = \vec{\varphi}$$

However, when the wave number k is large, the matrix A becomes highly indefinite, which prevents us from solving the problem efficiently. Fast relaxation methods such as Jacobi and Gauss-Seidel relaxation fail to eliminate or even diverge some error components, while the stable methods such as Kaczmarz relaxation need large amount sweeps. The above motivates us to use the multigrid methods [3][7][13], however, standard multigrid methods fail to solve the highly indefinite Helmholtz equations. This is because the standard multigrid methods are unable to reduce the error components $e^{ik(d_1 x + d_2 y)}$ with $d_1^2 + d_2^2 \approx 1$, these components are called *characteristic components*. On fine grids, the characteristic components are near the null space of the Helmholtz operator $\mathcal{L} := \Delta + k^2$, which make them invisible to local relaxation since their errors can have very small residuals. On the other hand, characteristic components are too oscillatory to be treated on coarse grids.

As for modern solvers, I. Livshits and A. Brandt [2][11] has proposed the wave-ray method. Lee et al.[9][10] combine the wave ray method and the first-order system least squares for Helmholtz problems. Elman et al.[4] modify the standard multigrid by adding GMRES iterations at coarse grids and use the modified multigrid as outer iteration for GMRES. Other methods such as domain decomposition preconditioning and complex shift Laplace preconditioning can be found at [5][8].

Among these methods, we are most interested in the wave ray method. The standard multigrid cycle is called wave cycle, and another *ray cycle* is employed to deal with the oscillatory characteristic components, and these components are represented as the product of an oscillatory Fourier mode and a smooth amplitude (or ray function), the ray function is smooth so that it can be treated on the coarse grids. The wave-ray method motivates us the use of PUM[1], the PUM space use functions which have better local approximation properties for oscillatory functions than the polynomials, and we are going to treat the characteristic components in PUM space since the coarse grids cannot eliminate them, the difference with the wave-ray method is that we will employ PUM based cycle instead of ray cycle after wave cycle, and we call this method the PUM wave-ray method, and we will implement it in C++ with LehrFEM++ library[1].

An outline of the paper is as follows. In chapter 2, we will give the derivation of Helmholtz equation, the boundary condition and variation form for our model problem. In chapter 3, we will give a brief introduction of the multigrid method and analysis why it fails for Helmholtz problem. Chapter 4 presents the basic idea of wave-ray method. Chapter 5 talks about the PUM and its implementation in LehrFEM++. We presents our PUM wave-ray method in chapter 6 and the numerical experiments are shown in chapter 7. In chapter 8, we use some stable relaxation methods and use the multigrid as the preconditioner for GMRES to solve the Helmholtz problems. Finally, we make some discussion in chapter 9. And the codes generating the results of numerical experiments can be found at my Github repository https://github.com/liaowangh/PUM_WaveRay.

[1] https://craffael.github.io/lehrfempp/

Chapter 2

Helmholtz equation

In this chapter we will introduce the acoustic waves, Helmholtz equation and boundary conditions. We will also give the variational form of our model Helmholtz problem.

2.1 Derivation

In this section we will give a short introduction about the acoustic wave propagation and its governing equations, which are obtained from the fundamental laws for compressible fluids[6]:

- *Conservation of Mass*:

  Consider the flow of fluid material with pressure $P(\boldsymbol{x}, t)$, density $\rho(\boldsymbol{x}, t)$ and velocity $\boldsymbol{V}(\boldsymbol{x}, t)$. The conservation of mass in a unit time interval is expressed as:

  $$-\frac{\partial}{\partial t}\int_V \rho dV = \int_{\partial V} \rho \boldsymbol{V}\cdot \boldsymbol{n} dS = \int_V \mathrm{div}(\rho \boldsymbol{V}) dV$$

  where V is a volume element with boundary ∂V and $\boldsymbol{n}$ is a outward normal vector. The second equality comes from the Gauss theorem, thus we have the continuity equation

  $$\frac{\partial \rho}{\partial t} + \mathrm{div}(\rho \boldsymbol{V}) = 0. \tag{2.1}$$

- *Equation of Motion:*

  The total force along ∂V is $\boldsymbol{F} = -\int_{\partial V} P\boldsymbol{n} dS$, according to the second Newtonian law $\boldsymbol{F} = m\boldsymbol{a}$, we have

  $$-\int_{\partial V} P\boldsymbol{n} dS = \int_V \rho \frac{d\boldsymbol{V}}{dt} dV$$

and $\int_{\partial V} P\boldsymbol{n}dS = \int_V \nabla P dV$ comes from the Gauss theorem, thus we arrive at the equation of motion

$$\rho \frac{\partial \boldsymbol{V}}{\partial t} = -\nabla P \tag{2.2}$$

where we make a linearization: $d\boldsymbol{V}/dt \approx \partial \boldsymbol{V}/\partial t$

From linear material law, we write:

$$P = c^2 \rho$$

where constant c is called the speed of sound. Then we can write

$$\begin{aligned}
\frac{\partial^2 P}{\partial t^2} &= c^2 \frac{\partial^2 \rho}{\partial t^2} \\
&\overset{i}{=} -c^2 \frac{\partial}{\partial t} \operatorname{div}(\rho \boldsymbol{V}) \\
&= -c^2 (\operatorname{div}(\frac{\partial \rho}{\partial t} \boldsymbol{V}) + \operatorname{div}(\rho \frac{\partial \boldsymbol{V}}{\partial t})) \\
&\overset{ii}{=} -c^2 \operatorname{div}(\frac{\partial \rho}{\partial t} \boldsymbol{V}) + c^2 \Delta P
\end{aligned}$$

where i and ii are derived based on the equation (2.1) and (2.2) respectively. If we further assume that ρ is constant and assume time harmonic waves, i.e. the scalar field P can be separated as

$$P(\boldsymbol{x}, t) = p(\boldsymbol{x}) \exp(-iwt)$$

where $w > 0$ is the angular frequency, we can finally obtain the Helmholtz equation

$$\Delta p + k^2 p = 0$$

with the wave number defined by

$$k := \frac{w}{c}.$$

2.2 Boundary condition

If we want the problem to be well-posed, we need to formulate suitable boundary conditions, typical the Helmholtz problem is considered in an unbounded exterior domain with Sommerfeld radiation condition at infinity, one way to make it friendly to numerical computational is to simply cuts off the infinite domain to form a domain Ω with limited size, and add an artificial absorbing boundary condition (ABC) at the new boundary, which models the effect of the rest of the domain. One kind of ABC is

$$\frac{\partial u}{\partial \boldsymbol{n}} - iku = g \quad \text{on } \partial\Omega$$

In practice we will consider the combination of ABS and Dirichlet boundary condition.

2.3 Model problem and Variational formulation

Throughout the paper, our experimental will base on the following Helmholtz boundary value problem posed on a two-dimensional domain $\Omega \subset \mathbb{R}^2$:

$$\begin{aligned} \Delta u + k^2 u &= 0 \quad \text{in } \Omega \\ \frac{\partial u}{\partial \boldsymbol{n}} - iku &= g \quad \text{on } \Gamma_R \\ u &= f \quad \text{on } \Gamma_D \end{aligned} \tag{2.3}$$

Here $k > 0$ is the wave number, $g \in L^2(\Gamma_R), f \in L^2(\Gamma_D)$, Γ_D and Γ_R are two well separated parts of the boundary, $\Gamma_D \cup \Gamma_R = \partial\Omega$. In practice, we will use different type of domains,

The weak formulation of Helmholtz problem (2.3) is given by

Find $u \in V := H^1_{\Gamma_D}(\Omega)$ such that

$$a(u, v) = F(v) \qquad \forall v \in V \tag{2.4}$$

where

$$a(u, v) = \int_\Omega (\operatorname{grad} u \cdot \operatorname{grad} \bar{v} - k^2 u\bar{v})dx - ik \int_{\Gamma_R} u\bar{v}dS \quad \forall u, v \in V \tag{2.5a}$$

$$F(v) = \int_{\Gamma_R} g\bar{v}dS \quad \forall v \in V \tag{2.5b}$$

Let V_h be a finite dimensional subspace of V. The Galerkin finite element discretizatin of (2.4) is as follows:

Find $u_h \in V_h$ such that,

$$a(u_h, v_h) = F(v_h) \quad \forall v_h \in V_h \tag{2.6}$$

Suppose $\{c_1(\mathrm{x}), \cdots, c_M(\mathrm{x})\}$ is a basis of V_h, and for $u_h \in V_h$, there are unique coefficients $\mu_j \in \mathbb{C}, j \in \{1, 2, \cdots, M\}$, such that $u_h = \sum_{j=1}^M \mu_j c_j$, then (2.6) is equivalent to the following linear system of equations:

$$A\vec{\boldsymbol{\mu}} = \vec{\boldsymbol{\varphi}} \tag{2.7}$$

Where

$$\begin{aligned} A &= [a(c_j, c_m)]_{m,j=1}^M \in \mathbb{C}^{M,M} \\ \vec{\boldsymbol{\mu}} &= (\mu_1, \cdots, \mu_M)^\top \in \mathbb{C}^M \\ \vec{\boldsymbol{\varphi}} &= [F(c_j)]_{j=1}^M \end{aligned}$$

Chapter 3

Multigrid method

3.1 Introduction

When we try to solve a boundary value problem with numerical method, we will probably end up solving a sparse system of linear equations. There are two kinds of methods to solve them: direct methods and iterative (or relaxation) methods. Direct method is generally expensive, and we are interested in the iterative methods. Jacobi and Gauss-Seidel iterations are two typical relaxation methods. The relaxation methods start with a initial guess at a solution, they generate new approximation from the previous one through some simple updating steps or iterations. Relaxation methods also have their limitations, if we apply Fourier mode analysis (assume the error consists of Fourier modes of different frequencies), we can find that high-frequency (oscillatory) modulation on the long wave in the error can be eliminated efficiently while the low-frequency (smooth) components persists [3]. This smoothing property inspires the multigrid method, the smooth modes on a fine grid look less smooth on a coarse grid, so when relaxation on fine grid begins to stall, we can move the smooth error modes to the coarser grid, where them appear more oscillatory and relaxation will be more effective.

Algorithm 1 shows the procedure of a standard correction scheme, the coarse grid provides the correction to the fine grid by solving the residual equation by multigrid recursively.

To implement the V-cycle, we need figure out how to move to coarser grid and what is system of equations in coarser grid. In algorithm 1, we move the fine grid residual to coarser grid and transfer the error correction back to fine grid, the I_H^h and I_h^H denote the prolongation and restriction operator respectively, and A_H is the coarse grid operator, their implementation in finite element background is introduced in next section.

Algorithm 1: $\vec{v} = MG(A, \vec{v}, \vec{\varphi})$ V-cycle scheme

if *On coarsest grid* **then**
  $\vec{v} = A^{-1}\vec{\varphi}$
else
  Relax ν_1 times on $A\vec{\mu} = \vec{\varphi}$ with initial guess $\vec{v}$.
  $\vec{v} \leftarrow \vec{v} + I_H^h MG(A_H, 0, I_h^H(\vec{\varphi} - A\vec{v}))$.
  Relax ν_2 times on $A\vec{\mu} = \vec{\varphi}$ with initial guess $\vec{v}$.
end
Here $\vec{\varphi}$ denotes the initial guess, A_H is the coarse-grid representation of A and I_h^H, I_H^h are restriction and prolongation operator.

3.2 Elements of Multigrid in Finite element method

In this section, we will illustrate the multigrid method in a two-mesh background. Suppose V_h and V_H are two finite element spaces associated with a fine and a coarse mesh respectively, and their basis functions are denoted as $\{b_1^h, \cdots, b_{n_h}^h\}$ and $\{b_1^H, \cdots, b_{n_H}^H\}$.

In practice, we create the meshes by uniform regular refinement, and choose the finite element space to be Lagrangian finite element space with the basis functions to be the nodal basis functions, thus we have $V_H \subset V_h$. For any $v \in V_h$, we can write it as a linear combination of $\{b_i^h\}$:

$$v = \sum_{i=1}^{n_h} v_i b_i^h, \quad v_i \in \mathbb{F}$$

where $\mathbb{F}$ is the scalar field ($\mathbb{R}$ or $\mathbb{C}$), so we can represent the element of V_h by its coefficient vector regarding nodal basis $\{b_i\}$, and we use $\mathbb{F}^{n_h}$ ($\mathbb{F}^{n_H}$) to denote the coefficient vector space with respect to V_h (V_H). In the sequel, we will use $v^h(v^H)$ to denote a function from $V_h(V_H)$ and use $\vec{v}^h = [v_i^h]$ ($\vec{v}^H = [v_i^H]$) to denote its coefficient vector. And the prolongation and restriction operator I_h^H and I_H^h will be used to denote the mapping between finite element spaces and coefficient vector spaces simultaneously.

Prolongation operator

As for the prolongation operator $I_H^h : V_H \to V_h$, since $V_H \subset V_h$, the prolongation operator is exact and is the embedding, implementation detail can be found in section 6.1, we assume

$$b_i^H = \sum_{j=1}^{n_l} q_{ji} b_j^h.$$

then we can write the I_H^h in the matrix form $I_H^h = Q = [q_{ij}] \in \mathbb{F}^{n_h \times n_H}$.

Restriction operator

Restriction operator I_h^H transfer the residual from fine mesh to coarse mesh. Suppose the fine grid problem $A_h\vec{\boldsymbol{\mu}}^h = \vec{\boldsymbol{\varphi}}^h$ is generated from the discrete variational problem

$$\text{find } u_h \in V_h, \text{ such that} \quad a(u_h, v_h) = F(v_h) \quad \forall v_h \in V_h$$

Suppose we have an approximation solution $v_h \in V_h$ and its coefficient vector is $\vec{\boldsymbol{v}}^h$, then the residual is $\vec{\boldsymbol{r}}^h = \vec{\boldsymbol{\varphi}}^h - A_h\vec{\boldsymbol{v}}^h$, we can observe that in terms of component of $\vec{\boldsymbol{r}}^h = [r_i^h]$

$$\begin{aligned} r_i^h &= \varphi_i^h - \sum_{j=1}^{n_h} a_{ij}^h v_i^h = F(b_i^h) - \sum_{j=1}^{n_h} a(b_j^h, b_i^h) v_i^h \\ &= F(b_i^h) - a(\boldsymbol{v}^h, b_i^h) := \hat{F}_h(b_i^h) \end{aligned}$$

we can say that residual is a functional $\hat{F}_h : V_h \mapsto \mathbb{F}$, and on the coarser grid, its natural to define $\hat{F}_H(w) := \hat{F}_h(w),\ \forall w \in V_H$ [13], thus

$$\begin{aligned} r_i^H &= \hat{F}_h(b_i^H) = \hat{F}_h(\sum_{j=1}^{n_h} q_{ji} b_j^h) \\ &= \sum_{j=1}^{n_h} q_{ji} \hat{F}_h(b_j^h) = \sum_{j=1}^{n_h} q_{ji} r_j^h \end{aligned}$$

so

$$[r_1^H, \cdots, r_{n_H}^H]^\top = Q^\top [r_1^h, \cdots, r_{n_h}^h]^\top$$

and we define $I_h^H = (I_H^h)^\top$.

The coarse mesh problem

A straightforward approach is to let the coarse mesh matrix A_H to be the "V_H version of the fine mesh matrix A_h". Another method is to construct the matrix A_H by Galerkin projection[7]. We start from the residual equation

$$A_h\vec{\boldsymbol{e}}^h = \vec{\boldsymbol{r}}^h.$$

For the moment assume $\vec{\boldsymbol{e}}^h$ lies in the range of the prolongation operator I_H^h. Then, there is a vector $\vec{\boldsymbol{e}}^H \in \mathbb{F}^{n_H}$ such that

$$\vec{\boldsymbol{e}}^h = I_H^h(\vec{\boldsymbol{e}}^H).$$

Substituting this equation into the former equation gives

$$A_h I_H^h(\vec{\boldsymbol{e}}^H) = \vec{\boldsymbol{r}}^h.$$

Now apply the restriction operator on both sides of this equation

$$I_h^H A_h I_H^h(\vec{e}^H) = I_h^H \vec{r}^h.$$

which leads to the definition

$$A_H := I_h^H A_h I_H^h.$$

This definition of the coarse grid matrix is called Galerkin projection. Although the assumption that $\vec{e}^h$ lies in the range of prolongation is in general not given, this derivation gives a motivation for defining the coarse grid operator.

3.3 Power iteration to obtain convergence factor

For an iterative method, the biggest eigenvalue of the iteration matrix is the convergence factor, which tells us roughly the worst factor by which the error is reduced with each relaxation sweep. We can use the power iteration to obtain the convergence factor of the multigrid method. We take the two grid correction scheme for example.

After building the linear system of equations

$$A_h \vec{\mu}^h = \vec{\varphi}^h$$

in fine mesh, the two-mesh correction scheme goes like:

- Relax ν_1 times on V_h with initial guess $\vec{\mu}^h$.
- Compute the residual $\vec{r}^h = \vec{\varphi}^h - A_h \vec{\mu}^h$, and transfer it to $\mathbb{F}^{n_H}$ by $I_h^H \vec{r}^h$.
- Solve $A_H \vec{e}^H = \vec{r}^H$.
- Transfer the coarse-mesh error to the fine-mesh by $\vec{e}^h = I_H^h \vec{e}^H$ and correct the fine-mesh approximation by $\vec{\mu}^h \leftarrow \vec{\mu}^h + \vec{e}^h$.
- Relax ν_2 times with initial guess $\vec{\mu}^h$.

Suppose the relaxation operator is R and it updates $\vec{\mu}^h$ in the following way:

$$\vec{\mu}^h \leftarrow R\vec{\mu}^h + C\vec{\varphi}^h$$

for constant matrix C. Then relaxation ν times can be written as

$$\vec{\mu}^h \leftarrow R^\nu \vec{\mu}^h + \underbrace{(\sum_{t=0}^{\nu-1} R^t) C\vec{\varphi}^h}_{:=f\left(\nu, \vec{\varphi}^h\right)}$$

Take the whole correction scheme process one step a time, we can write

$$\begin{aligned}\vec{\boldsymbol{\mu}}^h \leftarrow & R^{\nu_2}\left(I_d - I_H^h\ (A_H)^{-1}\, I_h^H A_h\right) R^{\nu_1}\vec{\boldsymbol{\mu}}^h \\ & + R^{\nu_2}(I_H^h A_H^{-1} I_h^H(\vec{\boldsymbol{\varphi}}^h - A_h f(\nu_1, \vec{\boldsymbol{\varphi}}^h)) + f(\nu_1, \vec{\boldsymbol{\varphi}}^h)) + f(\nu_2, \vec{\boldsymbol{\varphi}}^h) \quad (3.1)\end{aligned}$$

And

$$M = R^{\nu_2}\left(I_d - I_H^h\,(A_H)^{-1}\, I_h^H A_h\right) R^{\nu_1}$$

is the correction scheme operator. Note that in equation (3.1) if we let $\vec{\boldsymbol{\varphi}}^h = 0$, then the update goes like

$$\vec{\boldsymbol{\mu}}^h \leftarrow M\vec{\boldsymbol{\mu}}^h$$

If we normalize the $\vec{\boldsymbol{\mu}}^h$ after update and then keep updating and normalization, we are actually doing the power iteration regarding the operator M, which provides us the slowest convergence mode about this two grid correction scheme. And the corresponding eigenvalue (convergence factor) can be obtained by Rayleigh quotient.

3.4 Difficulties in solving Helmholtz equation with standard multigrid method

Most of the difficulties of standard multigrid method applied to Helmholtz problem can be seen from a one-dimensional model problem. We consider the following one-dimensional Helmholtz problem posed on the unit interval $(0,1)$ with homogeneous Dirichlet boundary conditions

$$-u'' - k^2 u = f, \quad u(0) = u(1) = 0 \quad (3.2)$$

Finite difference discretization of (3.2) on a uniform grid containing N interior points leads to a linear system of equations $A\vec{\boldsymbol{\mu}} = \vec{\boldsymbol{\varphi}}$ with the coefficient matrix

$$A = \left(1/h^2\right)\text{tridiag}(-1, 2, -1) - k^2\boldsymbol{I} \in \mathbb{R}^{N\times N}$$

where $h = 1/(N+1)$ denotes the mesh width and $\boldsymbol{I}$ denotes the identity matrix.

The eigenvalues of A are

$$\lambda_j = \frac{2(1-\cos j\pi h)}{h^2} - k^2 = \frac{4}{h^2}\sin^2\frac{j\pi h}{2} - k^2, \quad j = 1, \ldots, N$$

and the eigenvectors are

$$\vec{\boldsymbol{v}}_j = [\sin ij\pi h]_{i=1}^N, \quad j = 1, \ldots, N \quad (3.3)$$

The choice of Dirichlet boundary conditions in (3.2) allows us to perform Fourier analysis using these analytic expressions for eigenvalues and eigenvectors.

Smoothing

We consider the Gauss-Seidel(GS) relaxation

$$\vec{u}_{m+1} = (D - L)^{-1}(\vec{\varphi} + U\vec{u}_m)$$

as the smoothing operator, where we split A as $A = D - L - U$, D is the matrix consisting of the diagonal of A, $-L$ and $-U$ are strict lower and upper triangle matrix of A respectively. Then the error propagation matrix is $R_{GS} = (D - L)^{-1}U$, and its eigenvalue is

$$\lambda_j^{GS} = \frac{4}{(2 - h^2k^2)^2}\cos^2(j\pi h), \quad j = 1, \cdots, N$$

the derivation can be found in (A.2). When $hk > 2$, $\lambda_j^{GS} < 1$ for all j, the GS smoothing will damp every modes. When hk is small, some modes may be amplified, but GS is still stable if the the amplification factor is within the acceptable range. However, when kh is in the intermediate range, say $hk \approx \sqrt{2}$, then some eigenvalues will be very large and GS is very unstable, so we can perform GS in fine and coarse grids, but we need to avoid doing smoothing in intermediate grids. (Note that the modes mentioned above refer to the eigenvectors of R_{GS}, not the eigenvectors (3.3) of A.)

Figure (3.1) shows the effect of GS operator R_{GS} applied to eigenvectors (3.3) of A, it shows the norm reduction of each eigenvector after 5 G-S iterations. We can see that the oscillatory part are reduced efficiently, while small number of smooth modes are amplified.

Coarse grid correction

Here we will focus on an analysis of the two-grid correction scheme, since V-cycle is just nested application of the two-grid correction scheme, an understanding of it is important. We will show why standard multigrid will not always work for Helmholtz problem.

Assume the number of interior grid points N is odd, and let the next coarser grid consists of $n = (N - 1)/2$ interior grid points. Let $H = 2h$ denotes the coarse mesh size, let $\vec{e}^h = \vec{\mu} - \vec{v}^h$ denotes the fine-grid error, let $\vec{r}^h = \vec{\varphi}^h - A_h\vec{v}^h$ denotes the residual.

Let the coarse-to-fine prolongation be the interpolation $I_H^h : \mathbb{F}^n \to \mathbb{F}^N$

$$\left[I_H^h\vec{w}^H\right]_i := \begin{cases} [\vec{w}^H]_{i/2}, & i \text{ even}, \\ \frac{1}{2}[\vec{w}^H]_{(i-1)/2} + \frac{1}{2}[\vec{w}^H]_{(i+1)/2}, & i \text{ odd}, \end{cases} \quad i = 1, \ldots, N$$

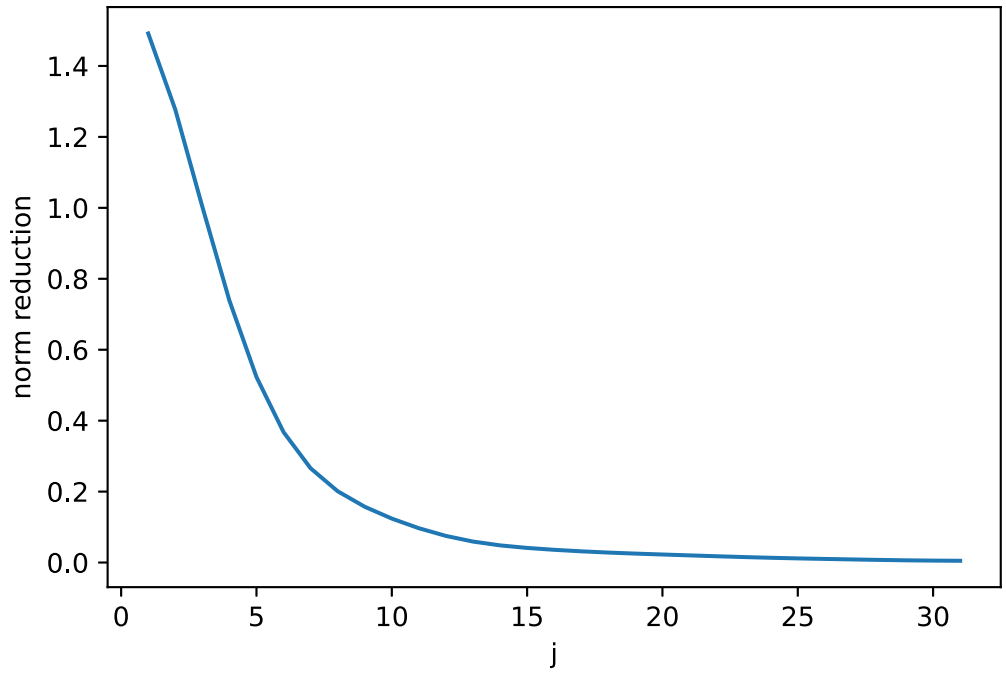


Figure 3.1: Gauss-Seidel iteration matrix applied to the model problem with $N = 31$, wave number $k = 3\pi$. The initial guesses consist of the eigenvectors $\{v_j\}_{j=1}^{N}$ of A. The figure shows the norm reduction $\|R_{GS}^{m}\vec{v}_j\|/\|\vec{v}_j\|$ with $m = 5$.

Let the fine-to-coarse operator be the full weighting $I_h^H : \mathbb{F}^N \to \mathbb{F}^n$, we have componentwise

$$\left[I_h^H \vec{u}^h\right]_i := \frac{1}{4}\left(\left[\vec{u}^h\right]_{2i-1} + 2\left[\vec{u}^h\right]_{2i} + \left[\vec{u}^h\right]_{2i+1}\right), \quad i = 1, \ldots, n$$

and the relation $I_H^h = 2\left(I_h^H\right)^\top$.

Following analysis was given in [4]. Consider a fine-grid error $\vec{e}^h = \vec{v}^h$ which is the exactly the smoothest eigenvector $\vec{v}^h$ of A_h with associated eigenvalue λ^h. The fine-grid residual is thus given by $\vec{r}^h = A_h\vec{e}^h = \lambda^h\vec{v}^h$, and, since we are assuming that $\vec{v}^h$ is smooth, its restriction $\vec{r}^H := I_h^H\vec{r}^h = \lambda^h I_h^H\vec{v}^h$ to the coarse grid will also be close to an eigenvector of the coarse grid operator A_H but with respect to a slightly different eigenvalue λ^H. The coarse grid version of correction is obtained by solving the residual equation in coarse

grid exactly

$$\vec{e}^H = (A_H)^{-1}\vec{r}^H = \lambda^h (A_H)^{-1} I_h^H \boldsymbol{v}^h \approx \frac{\lambda^h}{\lambda^H} I_h^H \boldsymbol{v}^h$$

The error on the fine grid after the correction is

$$\vec{e}^h - I_H^h \vec{e}^H \approx \vec{v}^h - \frac{\lambda^h}{\lambda^H} I_H^h I_h^H \vec{v}^h = \left(1 - \frac{\lambda^h}{\lambda^H}\right) \vec{v}^h, \tag{3.4}$$

since the mode $\vec{v}^h$ is very smooth, we can assume that it is invariant under restriction followed by prolongation. We can see from (3.4) that the effectiveness of the coarse grid correction depends on the ratio λ^h/λ^H under the assumption that the restrictions of smooth eigenvectors are again eigenvectors of A_H. If the ratio is 1, then the correction is exact, but if it's large, the correction can be arbitrarily bad. This occurs whenever one of λ^h, λ^H is close 0 and the other is not. Moreover, if λ^h and λ^H have opposite signs, then the correction is in the wrong direction.

Next let's see which eigenvalues are problematic. The fine grid eigenvectors are related by $\left[\vec{v}^h_{N+1-j}\right]_i = (-1)^{i+1}\left[\vec{v}^h_j\right]_i$. And the coarse grid eigenvectors are

$$\vec{v}_j^H = [\sin ij\pi H]_{i=1}^n, \quad j = 1, \cdots, n$$

Proposition 3.1 *The coarse-grid eigenvectors are mapped by the prolongation operator I_H^h according to*

$$I_H^h \vec{v}_j^H = c_j^2 \vec{v}_j^h - s_j^2 \vec{v}^h_{N+1-j}, \quad j = 1, \ldots, n \tag{3.5}$$

with

$$c_j := \cos\frac{j\pi h}{2}, \quad s_j := \sin\frac{j\pi h}{2}, \quad j = 1, \ldots, N$$

Proof For $i = 1, \cdots, n$

$$\begin{aligned}
[I_H^h \vec{v}_j^H]_{2i} &= [\vec{v}_j^H]_{2i} \\
&= \sin(ij\pi H)(c_j^2 + s_j^2) \\
&= c_j^2 \sin(2ij\pi h) - s_j^2(-\sin(2ij\pi h)) \\
&= c_j^2 [v_j^h]_{2i} - s_j^2 [v^h_{N+1-j}]_{2i}
\end{aligned}$$

For $i = 0, \cdots, n$

$$\begin{aligned}
[I_H^h \vec{\boldsymbol{v}}_j^H]_{2i+1} &= \frac{1}{2}[\sin(ij\pi H) + \sin((i+1)j\pi H)] \\
&= \frac{1}{2}[\sin((2i+1)j\pi h - j\pi h) + \sin((2i+1)j\pi h + j\pi h)] \\
&= \sin((2i+1)j\pi h)\cos(j\pi h) \\
&= [\vec{\boldsymbol{v}}_j^h]_{2i+1}(c_j^2 - s_j^2) \\
&= c_j^2[\vec{\boldsymbol{v}}_j^h]_{2i+1} - s_j^2[\vec{\boldsymbol{v}}_{N+1-j}^h]_{2i+1}
\end{aligned}$$ □

Proposition 3.2 *The fine-grid eigenvectors are mapped by the restriction operator I_h^H according to*

$$\boldsymbol{I}_h^H \vec{\boldsymbol{v}}_j^h = \begin{cases} c_j^2 \vec{\boldsymbol{v}}_j^H, & j = 1, \ldots, n \\ 0, & j = n+1 \\ -c_j^2 \vec{\boldsymbol{v}}_{N+1-j}^H, & j = n+2, \ldots, N \end{cases} \tag{3.6}$$

with c_j and s_j as defined as in Proposition 3.1

Proof For $j = 1, \cdots, n$

$$\begin{aligned}
[I_h^H \vec{\boldsymbol{v}}_j^h]_i &= \frac{1}{4}(\sin((2i-1)j\pi h) + 2\sin(2ij\pi h) + \sin((2i+1)j\pi h)) \\
&= \frac{1}{4}(2\sin(2ij\pi h)\cos(j\pi h) + 2\sin(2ij\pi h)) \\
&= \frac{1}{2}[\vec{\boldsymbol{v}}_j^H]_i(\cos(j\pi h) + 1) \\
&= c_j^2[\vec{\boldsymbol{v}}_j^H]_i
\end{aligned}$$

The derivation for $j = n+1, \cdots, N$ are similar. □

If A_H denotes the coarse grid discretization matrix, then one iterate of two-grid correction scheme updates the initial value as

$$\vec{\boldsymbol{u}}^h := \vec{\boldsymbol{u}}^h + I_H^h (A_H)^{-1} I_h^H (\vec{\boldsymbol{\varphi}}^h - A_h \vec{\boldsymbol{u}}^h)$$

from which we obtain the error propagation operator $\boldsymbol{C} := \boldsymbol{I} - I_H^h (A_H)^{-1} I_h^H A_h$. Denoting the eigenvalues of A_h and A_H by $\left\{\lambda_j^h\right\}_{j=1}^N$ and $\left\{\lambda_j^H\right\}_{j=1}^n$ respectively, we may summarize the action of $\boldsymbol{C}$ on the eigenvectors using (3.5) and (3.6) as follows.

Theorem 3.3 *The image of the fine-grid eigenfunctions* $\left\{\vec{v}_h^h\right\}_{j=1}^N$ *under the error propagation operator* $\boldsymbol{C}$ *of the exact coarse grid correction is given by*

$$\boldsymbol{C}\vec{\boldsymbol{v}}_j^h = \begin{cases} (1-c_j^4\frac{\lambda_j^h}{\lambda_j^H})\vec{\boldsymbol{v}}_j^h + s_j^2c_j^2\frac{\lambda_j^h}{\lambda_j^H}\vec{\boldsymbol{v}}_{N+1-j}^h, & j=1,\ldots,n, \\ \vec{\boldsymbol{v}}_{n+1}^h, & j=n+1, \\ (1-c_j^4\frac{\lambda_j^h}{\lambda_{N+1-j}^H})\vec{\boldsymbol{v}}_j^h + s_j^2c_j^2\frac{\lambda_j^h}{\lambda_{N+1-j}^H}\vec{\boldsymbol{v}}_{N+1-j}^h, & j=n+2,\ldots,N. \end{cases}$$

Proof We give the derivation for $j = 1, \cdots, n$,

$$\begin{aligned} \boldsymbol{C}\vec{\boldsymbol{v}}_j^h &= (\boldsymbol{I} - I_H^h\left(A_H\right)^{-1}I_h^HA^h)\vec{\boldsymbol{v}}_j^h \\ &= \vec{\boldsymbol{v}}_j^h - \lambda_j^hI_H^h\left(A_H\right)^{-1}I_h^H\vec{\boldsymbol{v}}_j^h \\ &= \vec{\boldsymbol{v}}_j^h - c_j^2\lambda_j^hI_H^h\left(A_H\right)^{-1}\vec{\boldsymbol{v}}_j^h \\ &= \vec{\boldsymbol{v}}_j^h - c_j^2\frac{\lambda_j^h}{\lambda_j^H}I_H^h\vec{\boldsymbol{v}}_j^h \\ &= \vec{\boldsymbol{v}}_j^h - c_j^2\frac{\lambda_j^h}{\lambda_j^H}(c_j^2\vec{\boldsymbol{v}}_j^h - s_j^2\vec{\boldsymbol{v}}_{N+1-j}^h) \\ &= (1-c_j^4\frac{\lambda_j^h}{\lambda_j^H})\vec{\boldsymbol{v}}_j^h + c_j^2s_j^2\frac{\lambda_j^h}{\lambda_j^H}\vec{\boldsymbol{v}}_{N+1-j}^h \end{aligned}$$

□

As a consequence, the two-dimensional spaces spanned by a smooth mode and its complementary mode are invariant under $\boldsymbol{C}$: $\boldsymbol{C}\left[\vec{\boldsymbol{v}}_j^h, \vec{\boldsymbol{v}}_{N+1-j}^h\right] = \left[\vec{\boldsymbol{v}}_j^h, \vec{\boldsymbol{v}}_{N+1-j}^h\right]\boldsymbol{C}_j$ with

$$\boldsymbol{C}_j := \begin{bmatrix} 1-c_j^4\frac{\lambda_j^h}{\lambda_j^H} & c_j^2s_j^2\frac{\lambda_{N+1-j}^h}{\lambda_j^H} \\ s_j^2c_j^2\frac{\lambda_j^h}{\lambda_j^H} & 1-s_j^4\frac{\lambda_{N+1-j}^h}{\lambda_j^H} \end{bmatrix}, j=1,\ldots,n$$

The following result shows the dependence of the matrices $\boldsymbol{C}_j$ on kh.

Theorem 3.4 *Using the notation defined above, there holds*

$$\boldsymbol{C}_j = \begin{bmatrix} s_j^2\left(1-\frac{k^2c_j^2}{\lambda_j^H}\right) & c_j^2\left(1+\frac{k^2c_j^2}{\lambda_j^H}\right) \\ s_j^2\left(1+\frac{k^2s_j^2}{\lambda_j^H}\right) & c_j^2\left(1-\frac{k^2s_j^2}{\lambda_j^H}\right) \end{bmatrix}, \quad j=1,\ldots,n \tag{3.7}$$

Moreover,

$$\lim_{kh\to 0}\boldsymbol{C}_j = \begin{bmatrix} s_j^2 & c_j^2 \\ s_j^2 & c_j^2 \end{bmatrix}, \quad \lim_{kh\to\infty}\boldsymbol{C}_j = \begin{bmatrix} s_j^2\left(1+c_j^2\right) & s_j^2c_j^2 \\ s_j^2c_j^2 & c_j^2\left(1+s_j^2\right) \end{bmatrix}, \quad j=1,\ldots,n$$

Proof Recall that $c_j = \cos(j\pi h/2)$, $s_j = \sin(j\pi h/2)$ and

$$\lambda_j^h = \frac{4}{h^2}\sin^2\frac{j\pi h}{2} - k^2 = \frac{4}{h^2}s_j^2 - k^2, \quad j = 1, \cdots, N$$

$$\begin{aligned}\lambda_j^H &= \frac{4}{H^2}\sin^2\frac{j\pi H}{2} - k^2 = \frac{4}{h^2}\sin^2\frac{j\pi h}{2}\cos^2\frac{j\pi h}{2} - k^2 \\ &= \frac{4}{h^2}c_j^2 s_j^2 - k^2 \quad j = 1, \cdots, n\end{aligned}$$

We can rewrite the $(1,1)-$entry of C_j, $1 - c_j^4\lambda_j^h/\lambda_j^H = (\lambda_j^H - c_j^4\lambda_j^h)/\lambda_j^H$, substitude the expression of λ_j^h and λ_j^H into the numerator:

$$\begin{aligned}\lambda_j^H - c_j^4\lambda_j^h &= \frac{4}{h^2}c_j^2 s_j^2 - k^2 - c_j^4(\frac{4}{h^2}s_j^2 - k^2) \\ &= \frac{4}{h^2}c_j^2 s_j^2(1 - c_j^2) - k^2(1 - c_j^4) \\ &= \frac{4}{h^2}c_j^2 s_j^4 - k^2 s_j^2(1 + c_j^2)) \\ &= s_j^2(\frac{4}{h^2}c_j^2 s_j^2 - k^2 - k^2 c_j^2) \\ &= s_j^2(\lambda_j^H - k^2 c_j^2)\end{aligned}$$

so we have

$$1 - c_j^4\frac{\lambda_j^h}{\lambda_j^H} = s_j^2(1 - \frac{k^2 c_j^2}{\lambda_j^H})$$

For the $(2,1)-$ entry

$$\begin{aligned}s_j^2 c_j^2\lambda_j^h &= s_j^2 c_j^2(\frac{4}{h^2}s_j^2 - k^2) \\ &= s_j^2(\frac{4}{h^2}s_j^2 c_j^2 - k^2 + k^2 - k^2 c_j^2) \\ &= s_j^2(\lambda_j^H + k^2 s_j^2)\end{aligned}$$

we have

$$s_j^2 c_j^2\frac{\lambda_j^h}{\lambda_j^H} = s_j^2(1 + \frac{k^2 s_j^2}{\lambda_j^H}).$$

The derivation for the remaining two items are similar, we need to use the fact that $s_{N+1-j} = c_j$.

For the second part of the theorem, we note that

$$\begin{aligned}\frac{k^2c_j^2}{\lambda_j^H} &= \frac{k^2c_j^2}{\frac{4}{h^2}s_j^2c_j^2 - k^2} \\ &= \frac{k^2h^2c_j^2}{4s_j^2c_j^2 - k^2h^2} \\ &= \frac{c_j^2}{\frac{4s_j^2c_j^2}{k^2h^2} - 1}\end{aligned}$$

we can conclude that

$$\lim_{kh\to 0}\frac{k^2c_j^2}{\lambda_j^H} = 0 \quad \lim_{kh\to\infty}\frac{k^2c_j^2}{\lambda_j^H} = -c_j^2$$

similarly,

$$\lim_{kh\to 0}\frac{k^2s_j^2}{\lambda_j^H} = 0 \quad \lim_{kh\to\infty}\frac{k^2s_j^2}{\lambda_j^H} = -s_j^2$$ □

If we apply the error propagation operator to a smooth mode $\vec{v}_j^h$, theorem 3.4 tells us that

$$\boldsymbol{C}\vec{v}_j^h = \boldsymbol{C}\left[\vec{v}_j^h, \vec{v}_{N+1-j}^h\right]\begin{pmatrix}1\\0\end{pmatrix} = \left[\vec{v}_j^h, \vec{v}_{N+1-j}^h\right]\boldsymbol{C}_j\begin{pmatrix}1\\0\end{pmatrix}$$

Denote $\boldsymbol{C}_j = [c_{ij}]_{i,j=1}^2$, then

$$\boldsymbol{C}\vec{v}_j = c_{11}\vec{v}_j + c_{12}\vec{v}_{N+1-j}$$

we can see that if both c_{11} and c_{12} are small, then this mode is damped by the coarse grid correction. However if c_{11} is large, this mode is amplified, and if c_{12} is large, the complementary mode is introduced by the coarse grid, from (3.7), we can know that these situations happen when λ_j^H is small. Look back to equation (3.4),

$$\frac{\lambda_j^h}{\lambda_j^H} = \frac{4s_j^2/h^2 - k^2}{4s_j^2c_j^2/h^2 - k^2} = 1 + \frac{4s_j^2}{4s_j^2c_j^2 - h^2k^2}$$

For very small or very large value of kh, the ratio approaches 1 and thus the coarse grid correction strongly damps the smooth error modes, but this does not hold when kh is in the intermediate range.

Chapter 4

Wave-ray multigrid method

4.1 Characteristic Component and Ray Equations

From the analysis of the section 3.4, we know that some components are hard to eliminate when using standard multigrid method (wave cycle) to solve Helmholtz equation. Fourier component of form $\exp(i(k_1 x + k_2 y)$ with $k_1^2 + k_2^2 = k^2$ are such problematic errors, these components will be called the principal components. The principal components satisfy the homogeneous Helmholtz equation $(\Delta + k^2)u = 0$, so they are in the null space of the operator, hence in discretization form, they are close to the eigenvectors with associated eigenvalue to 0, what's more, they may have very small residual, which make them nearly invisible to standard local relaxation. On the other hand, on coarser grids such components cannot be approximated, because the grid does not resolve their oscillations. (to avoid large phase error, mesh width h should be very small.)

Brandt and Livshits[2][11] proposed a variant of multigrid especially tailored to the Helmholtz equation by exploiting the structure of the characteristic error components.

Let's consider a two-dimensional Helmholtz equation:

$$\Delta u(x,y) + k^2 u(x,y) = f(x,y), \quad (x,y) \in \mathbb{R}^2 \tag{4.1}$$

the problem is descretized with five point scheme:

$$\frac{u^h_{i-1,j} - 2u^h_{i,j} + u^h_{i+1,j}}{h^2} + \frac{u^h_{i,j-1} - 2u^h_{i,j} + u^h_{i,j+1}}{h^2} + k^2 u^h_{i,j} = f_{i,j}$$

where $u_{i,j} \approx u(ih, jh)$ and $f_{i,j} = f(ih, jh)$.

Let $v(x,y)$ be the error left after the standard multigrid cycle, and it has the form

$$v(x,y) = \sum_{l=1}^{L} \hat{v}_l(x,y) \exp(i(k_1^l x + k_2^l y))$$

where $\hat{v}_l$ are called the *ray functions*, if L is chosen to be sufficiently large, ray functions are smooth enough to be approximated on coarse levels. The residual corresponding to the error can be represnented as

$$r(x,y) = \sum_{l=1}^{L} \hat{r}_l(x,y) \exp(i(k_1^l x + k_2^l y))$$

Substitute v and r into (4.1), we get the equations for ray functions $\hat{v}_l$:

$$\Delta \hat{v}_l + 2ik_1 \frac{\partial \hat{v}_l}{\partial x} + 2ik_1 \frac{\partial \hat{v}_l}{\partial y} = \hat{r}_l \quad , l = 1, \cdots, L$$

The ray equation describe the envelope function of a ray function, and the oscillatory component $\exp(i(k_1^l x + k_2^l y))$ disappears from the equation.

4.2 Separation

What we can get after the standard multigrid cycle is the residual $r(x,y)$, to obtain the ray equations we need to obtain the residual $\hat{r}_l$, this procedure is called *separation*. In practice, the residual is obtained on a sufficiently fine grid, then transfer to the separation level.

A basic tool for the separation is a simple one-dimensional three-point weighting

$$W = (w_0, w_1, w_0)\,.$$

Let $g(\alpha)$ be a one-dimensional function defined on a gird with mesh size h, and it has the form

$$g(\alpha) = a_1(\alpha)e^{-ip\alpha} + a_2(\alpha) + a_3(\alpha)e^{ip\alpha}$$

where the $a_j(\alpha)$ are smooth, compared to $e^{\pm ip\alpha}$. We want to remove the characteristic waves and only smooth a_2 should be left. The result of applying the weighting operator to g is

$$\begin{aligned}
(Wg)(\alpha) &= w_0 g(\alpha - h) + w_1 g(\alpha) + w_0 g(\alpha + h) \\
&= e^{-ip\alpha}[w_0 a_1(\alpha - h)e^{iph} + w_1 a_1(\alpha) + w_0 a_1(\alpha - h)e^{-iph}] \\
&\quad + w_0 a_2(\alpha - h) + w_1 a_2(\alpha) + w_0 a_2(\alpha + h) \\
&\quad + e^{ip\alpha}[w_0 a_3(\alpha - h)e^{-iph} + w_1 a_3(\alpha) + w_0 a_3(\alpha + h)e^{iph}] \\
&\approx e^{-ip\alpha} a_1(\alpha)(w_0 e^{iph} + w_1 + w_0 e^{-iph}) + a_2(\alpha)(2w_0 + w_1) \\
&\quad + e^{-ip\alpha} a_3(\alpha)(w_0 e^{-iph} + w_1 + w_0 e^{iph})
\end{aligned}$$

Where we use the fact that $a_j(\alpha)$ are smooth and assume $a_j(\alpha - h) \approx a_j(\alpha) \approx a_j(\alpha + h)$. To let the result after weighting be an approximation to the function $a_2(\alpha)$, we can set

$$\begin{aligned}
2w_0 + w_1 &= 1 \\
w_0(e^{-iph} + e^{iph}) + w_1 &= 0
\end{aligned}$$

which gives us

$$w_0 = \frac{1}{2(1-\cos(ph))} \quad w_1 = -\frac{\cos(ph)}{1-\cos(ph)}$$

A *two-dimensional* weighting operator can be constructed as a tensor product of two one-dimensional operators.

After the separation, ray equations can be solved by another multigrid cycle, which is called the ray cycle. Then the ray solution have to be interpolated back to the separation level to begin the merge procedure, the solution of the ray equations is multiplied with the corresponding characteristic component and added to the wave-cycle solution.

Chapter 5

Plane-wave PUM

5.1 The partition of unity method

The partition of unity method (PUM) was proposed by I. Babuška and J. Melenk[1]. The PUM method allows the construction of conforming ansatz spaces with local properties determined by the user. The main features of the PUM are the following.

- The construction of ansatz spaces can make use of *a priori* knowledge about the differential equation. The classical FEM relies on the local approximation properties of polynomials, for some problems (problems with highly oscillatory solution), polynomials have poor approximation properties. If we know the analytic knowledge about the local behaviour of the exact solution, we may can approximate the exact solution locally with functions better than polynomials.

- The PUM enable us to construct ansatz spaces of any desired regularity.

To see the second feature clearly, we need to present the definition of PUM first.

Definition 5.1 Let $\Omega \subset \mathbb{R}^n$ be an open set, $\{\Omega_i\}$ be an open cover of Ω satisfying a pointwise overlap condition

$$\exists M \in \mathbb{N} \quad \forall x \in \Omega \quad \operatorname{card}\{i \mid x \in \Omega_i\} \leqslant M$$

Let $\{\varphi_i\}$ be a Lipschitz partition of unity subordinate to the cover $\{\Omega_i\}$

satisfying

$$\begin{aligned} \operatorname{supp} \varphi_i &\subset \operatorname{closure}(\Omega_i) \quad \forall i \\ \sum_i \varphi_i &\equiv 1 \quad \text{on} \Omega \\ \|\varphi_i\|_{L^\infty(\mathbb{R}^n)} &\leqslant C_\infty \\ \|\nabla \varphi_i\|_{L^\infty(\mathbb{R}^n)} &\leqslant \frac{C_G}{\operatorname{diam} \Omega_i} \end{aligned}$$

where C_∞, C_G are two constants. Then $\{\varphi_i\}$ is called a (M, C_∞, C_G) partition of unity subordinate to the cover $\{\Omega_i\}$. The partition of unity $\{\varphi_i\}$ is said to be of degree $m \in \mathbb{N}_0$ if $\{\varphi_i\} \subset C^m(\mathbb{R}^n)$. The covering sets $\{\Omega_i\}$ are called patches.

Definition 5.2 Let $\{\Omega_i\}$ be an open cover of $\Omega \subset \mathbb{R}^n$ and let $\{\varphi_i\}$ be a (M, C_∞, C_G) partition of unity subordinate to $\{\Omega_i\}$. Let $V_i \subset H^1(\Omega_i \cap \Omega)$ be given. Then the space

$$V := \sum_i \varphi_i V_i = \left\{ \sum_i \varphi_i v_i \mid v_i \in V_i \right\} \subset H^1(\Omega)$$

is called the *PUM* space. The PUM space V is said to be of degree $m \in \mathbb{N}$ if $V \subset C^m(\Omega)$. The spaces V_i are referred to as the local approximation spaces.

From the definition of the PUM, we can see that local approximation in the spaces V_i can be either achieved by the smallness of the patches (an 'h' version) or by good properties of V_i (a 'p' version).

5.2 Approximate the solution of Helmholtz equation

Here we consider solving the Helmholtz problem (2.3) with the PUM. When the wave number k is high, the solution of Helmholtz problem is highly oscillatory, approximation with polynomials performs poorly and Melenk[12] has demonstrated that the approximation with plane waves which displaying the same oscillatory behaviour as the solution can be very efficient. Therefore we choose the following type of local approximation spaces

$$\tilde{V} = \operatorname{span}\{e_t(\boldsymbol{x}) | t = 1, \cdots, N\}$$

where N is the number of plane waves and

$$\begin{aligned} \boldsymbol{d}_t &= (\cos \tfrac{2\pi(t-1)}{N}, \sin \tfrac{2\pi(t-1)}{N}) \quad t = 1, \cdots, N \\ e_t(\boldsymbol{x}) &= \exp(ik\boldsymbol{d}_t^l \cdot \boldsymbol{x}) \end{aligned}$$

Let $\mathcal{T}$ denote the mesh, $\mathcal{V}$ denote the set of vertices of $\mathcal{T}$, assume $n = \sharp|\mathcal{V}|$. The nodal basis functions $\{b_i\}$ associated with $\mathcal{T}$ is chosen as the partition

of unity, which leads to the following PUM space

$$W = \text{span}\{b_i(\boldsymbol{x})e_t(\boldsymbol{x})|\boldsymbol{p}_i \in \mathcal{V}, t = 1, \cdots, N\} \tag{5.1}$$

In order to keep using the notation from §2.3, we write the basis functions of W in the following order:

$$b_1e_1, \cdots, b_1e_N, \cdots, b_ne_1, \cdots, b_ne_N$$

and denote them as $c_1, \cdots, c_M, M = nN$, the relation between b_je_t and c_p is that if $b_je_t = c_p$, then

$$\begin{aligned} &p = (j-1)N + t \\ &j = \lfloor (p-1)/N \rfloor + 1, \quad t = (p-1) \bmod N + 1 \end{aligned} \tag{5.2}$$

We can have the following linear system of equations

$$A\vec{\boldsymbol{\mu}} = \vec{\boldsymbol{\varphi}} \tag{5.3}$$

Note that in plane wave PUM space (5.1), the nodal basis function b_i is not included in the space, we can define the extended PUM space:

$$EW = \text{span}\{b_i(\boldsymbol{x})e_t(\boldsymbol{x})|\boldsymbol{p}_i \in \mathcal{V}, t = 0, \cdots, N\} \tag{5.4}$$

where $e_0 \equiv 1$, so that the Lagrangian finite element space $\mathcal{S}_1^0(\mathcal{T}) \subset EW$. The intuition is that adding the nodal basis function to PUM space is helpful to dealing with the non-homogeneous Helmholtz equation, since the plane wave function e_t satisfies $(\Delta + k^2)e_t = 0$, the nodal basis function can help to resolve the non-zero right hand side.

5.3 Implementation in LehrFEM++

Assembly Algorithms in LehrFEM++

We make use of the assembly algorithms implemented in LehrFEM++ to build the stiffness matrix and right hand side vector in (5.3). Let's first introduce the idea of the assembly. We need to compute the matrix $A = [a(c_j, c_i)]_{i,j=1}^M$ and $\vec{\boldsymbol{\varphi}} = [F(c_i)]_{i=1}^M$. Computational of every entry in A and $\vec{\boldsymbol{\varphi}}$ can be written in terms of local cell contributions,

$$\begin{aligned} a(u,v) &= \int_\Omega (\text{grad}\, u \cdot \text{grad}\, \bar{v} - k^2 u\bar{v})dx - ik\int_{\Gamma_R} u\bar{v}dS \\ &= \sum_{K\in\mathcal{T}} \int_K (\text{grad}\, u_{|K} \cdot \text{grad}\, \bar{v}_{|K} - k^2 u_{|K}\bar{v}_{|K})dx \\ &\quad - ik\sum_{e\in\Gamma_R} \int_e u_{|e}\bar{v}_{|e}dS \quad \forall u, v \in W \\ F(v) &= \int_{\Gamma_R} g\bar{v}dS = \sum_{e\in\Gamma_R} \int_e g\bar{v}_{|e} \quad \forall v \in W \end{aligned}$$

where K is the element of the mesh, e is the the element of the boundary. $\cdot|_K$ restrict the function to K, $\cdot|_e$ restrict the function to e, and they are not defined outside the local element. Let

$$a_1(u,v) = \int_\Omega (\operatorname{grad} u \cdot \operatorname{grad} \bar{v} - k^2 u\bar{v})dx$$

and

$$a_{1|K}(u,v) = \int_K (\operatorname{grad} u_{|K} \cdot \operatorname{grad} \bar{v}_{|K} - k^2 u_{|K}\bar{v}_{|K})dx$$

We will illustrate the algorithm by assembling the matrix $A_1 = [a_1(c_j, c_i)]_{i,j=1}^M$. Suppose the local basis functions in cell K is $\{c_K^1, \cdots, c_K^Q\}, Q = Q(K) \in \mathbb{N}$, and define the element matrix

$$A_{1|K} := [a_{1|K}(c_K^j, c_K^i)]_{i,j=1}^Q$$

Since the local basis functions are restriction of global shape functions to the element K, there is a connection between them, let G be the index mapping such that

$$c_{j|K} = c_K^i, \quad \text{if } G(K,i) = j$$

Algorithm 2: $B \leftarrow$Assemble($\mathcal{T}$), Assembly routine for element matrix

```
Initial B of size M × M
for K ∈ T do
    get index mapping G
    get local element matrix B_K
    for i = 1 to Q_K do
        for j = 1 to Q_K do
            B(G(K,i), G(K,j)) += B_K(i,j)
        end
    end
end
```

Algorithm 2 is the assemble procedure, in LehrFEM++ routine

$$lf :: assemble :: AssembleMatrixLocally$$

implements this assembly process, what we need to do is to give the routine to compute the local element matrix and local to global index mapping. Things are tricky when the underlying finite element space is the plane wave PUM spaces and the basis function is the nodal basis function times the plane wave, we need to design our own class to compute the local matrix.

Local computation for plane wave PUM space

```
Mat_t Eval(const lf::mesh::Entity& cell){
        const lf::base::RefEl ref_el{cell.RefEl()};
    LF_ASSERT_MSG(ref_el == lf::base::RefEl::kTria(),
                "Cell must be of triangle type");
    Mat_t elem_mat(3 * N_, 3 * N_);
    const lf::geometry::Geometry *geo_ptr = cell.Geometry();

    auto vertices = geo_ptr->Global(ref_el.NodeCoords());

    Eigen::Matrix3d X, tmp;
    tmp.block<3,1>(0,0) = Eigen::Vector3d::Ones();
    tmp.block<3,2>(0,1) = vertices.transpose();
    X = tmp.inverse();

    for(int i = 0; i < 3*N_; ++i){
        int i1 = i / N_; int t1 = i % N_;
        for(int j = 0; j < 3*N_; ++j) {
            int j2 = j / N_; int t2 = j % N_;
            auto f = [this,&X,&i1,&j2,&t1,&t2](const
             ↪  Eigen::Vector2d& x)->Scalar {
                Eigen::Vector2d di, dj, betai, betaj;
                double pi = std::acos(-1);
                di << std::cos(2*pi*t1/N_),
                 ↪  std::sin(2*pi*t1/N_);
                dj << std::cos(2*pi*t2/N_),
                 ↪  std::sin(2*pi*t2/N_);
                betai << X(1, i1), X(2, i1);
                betaj << X(1, j2), X(2, j2);
                double lambdai = X(0,i1) + betai.dot(x);
                double lambdaj = X(0,j2) + betaj.dot(x);

                auto gradci = std::exp(1i*k_*di.dot(x)) * (betai
                 ↪  + 1i*k_*lambdai*di);
                auto gradcj = std::exp(1i*k_*dj.dot(x)) * (betaj
                 ↪  + 1i*k_*lambdaj*dj);
                auto val_ci = lambdai *
                 ↪  std::exp(1i*k_*di.dot(x));
                auto val_cj = lambdaj *
                 ↪  std::exp(1i*k_*dj.dot(x));

```

```
34                  return alpha_ * gradci.dot(gradcj) + gamma_ *
                    ↪  val_cj * std::conj(val_ci);
35              };
36              elem_mat(i, j) = LocalIntegral(cell, degree_, f);
37          }
38      }
39      return elem_mat;
40  }
```

The C++ code above is the key routine to compute the local element matrix $A_{1|K}$. Here we made a small change about $a_1(\cdot,\cdot)$:

$$a_{1|K}(c_p, c_q) = \int_K (\alpha \operatorname{grad} c_p \cdot \operatorname{grad} \overline{c_q} + \gamma c_p \overline{c_q}) dx \tag{5.5}$$

where K is a local triangle cell, which corresponds to the input argument

```
const lf::mesh::Entity& cell
```

$\alpha, \gamma \in \mathbb{C}$ are two free parameters, for Helmholtz equation, we can set $\alpha = 1, \gamma = -k^2$. Let $c_p(\boldsymbol{x}) = b_{i(p)}(\boldsymbol{x}) \exp(ik\boldsymbol{d}_{t(p)} \cdot \boldsymbol{x})$, where $t(\cdot), i(\cdot)$ is defined as in (5.2), the gradient is

$$\operatorname{grad} c_p(\boldsymbol{x}) = \exp(ik\boldsymbol{d}_{t(p)} \cdot \boldsymbol{x}) \operatorname{grad} b_{i(p)}(\boldsymbol{x}) + ikb_{i(p)}(\boldsymbol{x}) \exp(ik\boldsymbol{d}_{t(p)} \cdot \boldsymbol{x}) \boldsymbol{d}_{t(p)}$$

Line 19-35 in the code creates the lambda expression for

$$\alpha \operatorname{grad} c_p \cdot \operatorname{grad} \overline{c_q} + \gamma c_p \overline{c_q}$$

then in line 36, we use the numerical quadrature to compute the value $a_{1|K}(c_p, c_q)$.

Now let's go into the detail of the code. In line 5, we create the matrix object to represent $A_{1|K}$, where $N\_$ is the number of plane waves (as the N_l in (5.1)), since the cell is triangle, the number of local basis functions is $3N\_$. The nodal basis functions restricted in this cell is the barycentric coordinate functions, their expressions can be obtained if we know the coordinates of the nodes of the triangle cell, and line 11-13 does this job, $X(0,i) + X(1,i)x + X(2,i)y$ represents the barycentric coordinate function associates with node i. Line 15-18 are two iterations over the local basis functions and compute the index for barycentric coordinate function and the plane wave function, line 24-27 compute the value of barycentric coordinate function at point $\boldsymbol{x}$, line 29-32 compute the value of basis function and its gradient at point $\boldsymbol{x}$.

As for edge matrix $A_{2|e} = [a_{2|e}(c_K^j, c_K^i)]_{i,j=1}^{Q(e)}$ As for the local computation concerning

$$a_{2|e}(c_p, c_q) = \gamma \int_e c_p \overline{c_q} dS$$

where e is an edge of the mesh, $\gamma \in \mathbb{C}$ is a parameter, for Helmholtz problem, we need to set $\gamma = -ik$. we can make use of the LefrFEM++ module

`lf::uscalfe::MassEdgeMatrixProvider`

which computes the edge matrix corresponding to

$$(u, v) \mapsto \int_e \gamma(\boldsymbol{x}) u(\boldsymbol{x}) \overline{v(\boldsymbol{x})} dS,$$

where u, v are nodal basis functions restricted at edge e. Note that

$$c_p \overline{c_q} = \exp(ik(\boldsymbol{d}_{t(p)} - \boldsymbol{d}_{t(q)}) \cdot \boldsymbol{x}) b_{i(p)} b_{i(q)}$$

if we set $\gamma(\boldsymbol{x}) = \exp(ik(\boldsymbol{d}_{t(p)} - \boldsymbol{d}_{t(q)}) \cdot \boldsymbol{x})$, we can build matrix $A_{2|e}$ with the help of the LehrFEM++ routine. The C++ code are as follows:

```
1  Mat_t Eval(const lf::mesh::Entity &edge){
2      const lf::base::RefEl ref_el{edge.RefEl()};
3      LF_ASSERT_MSG(ref_el == lf::base::RefEl::kSegment(),"Edge
       ↪  must be of segment type");
4      Mat_t edge_mat(2 * N_, 2 * N_);
5
6      double pi = std::acos(-1.);
7      Eigen::Matrix<Scalar, 2, 1> d1, d2;
8      for(int t1 = 0; t1 < N_; ++t1) {
9          d1 << std::cos(2*pi*t1/N_), std::sin(2*pi*t1/N_);
10         for(int t2 = 0; t2 < N_; ++t2) {
11             d2 << std::cos(2*pi*t2/N_), std::sin(2*pi*t2/N_);
12             auto new_gamma = [this, &d1, &d2](const
               ↪  Eigen::Vector2d& x)->Scalar{
13                 return gamma_ * std::exp(1i * k_ *
                   ↪  (d2-d1).dot(x));
14             };
15             lf::mesh::utils::MeshFunctionGlobal
               ↪  mf_gamma{new_gamma};
16             lf::uscalfe::MassEdgeMatrixProvider<double,
               ↪  decltype(mf_gamma), decltype(edge_selector_)>
17                 edgeMat_builder(fe_space_, mf_gamma,
                   ↪  lf::quad::make_QuadRule(ref_el, degree_),
                   ↪  edge_selector_);
18             const auto edge_mat_tmp =
               ↪  edgeMat_builder.Eval(edge);
19             edge_mat(t1, t2) = edge_mat_tmp(0, 0);
20             edge_mat(t1, t2 + N_) = edge_mat_tmp(0, 1);
21             edge_mat(t1 + N_, t2) = edge_mat_tmp(1, 0);
```

```
22                edge_mat(t1 + N_, t2 + N_) = edge_mat_tmp(1, 1);
23            }
24        }
25        return edge_mat;
26    };
```

Here the number of local basis functions in edge e is $2N_{\_}$. Line 8 and 9 are two iterations over local basis functions. Line 12-14 define the lambda object representing the function $\tilde{\gamma}(\boldsymbol{x}) = \gamma \exp(ik(\boldsymbol{d}_{t(p)} - \boldsymbol{d}_{t(q)}) \cdot \boldsymbol{x})$. Line 16-18 computes the local edge matrix corresponds to

$$(b_i, b_j) \to \int_e \tilde{\gamma}(\boldsymbol{x}) b_i(\boldsymbol{x}) \overline{b_j(\boldsymbol{x})} dS$$

Line 19-22 assign the value to $A_{2|e}$.

Local computation for right hand side vector is similar, we can make use of the routine

`lf::uscalfe::ScalarLoadEdgeVectorProvider`

which computes the local edge vector corresponding to

$$v \to \int_e \psi(\boldsymbol{x}) \overline{v(\boldsymbol{x})} dS$$

where g is a locally continuous source function. As for

$$F_e(c_p) = \int_e g(\boldsymbol{x}) \exp(-ik\boldsymbol{d}_{t(p)} \cdot \boldsymbol{x}) b_{i(p)}(\boldsymbol{x})$$

we can set $\psi(\boldsymbol{x}) = g(\boldsymbol{x}) \exp(-ik\boldsymbol{d}_{t(p)} \cdot \boldsymbol{x})$ to make use of the LehrFEM++ routine.

Treatment of non-zero Dirichlet condition

There is a boundary condition $u = f$ on Γ_D in problem (2.3), treatment in LehrFEM++ is that first ignore the essential boundary conditions and assemble the linear system of equations $A\vec{\mu} = \vec{\varphi}$, since some component of $\vec{\mu}$ is known (Dirichlet data), next step is to modify the linear system according to the known value. The Dirichlet data is easily obtained in Lagrangian finite element space, which is nodal projection, however, in plane wave PUM space, things are not straightforward since more than one basis functions don't vanish in boundary node, the strategy taken here is the L^2 projection.

For any function $u \in L^2(\Omega)$, L^2 projection into a finite element space V_h seeks the element $u_h \in V_h$ that is the best approximation:

$$u_h = \operatorname{argmin}_{v_h} \|v_h - u\|^2$$

equivalently,

$$(u_h, v_h)_\Omega = (u, v_h)_\Omega \quad \forall v_h \in V_h$$

where we use the L_2 norm, $\|u\| = \int_\Omega |u|^2 dx$ and the inner product $(u, v)_\Omega = \int_\Omega u\bar{v}dx$. we can also build the linear system of equations

$$B\vec{\boldsymbol{x}} = \vec{\boldsymbol{b}}$$

with $\{\varphi_i\}$ a basis of V_h, $B = [(\varphi_j, \varphi_i)]_{ij}$, $u_h = \sum_i x_i \varphi_i$ and $\vec{\boldsymbol{b}} = [(f, v_i)]_i$

For Dirichlet condition, let $V_h(\Gamma_D)$ the finite space spanning by basis functions of V_h which associate with nodes on Γ_D, then $f_h \in V_h(\Gamma_D)$ we seek satisfies

$$(f_h, v)_{\Gamma_D} = (f, v)_{\Gamma_D} \quad \forall v \in V_h(\Gamma_D)$$

In implementation, we can first build the linear equations $Bx = b$ for

$$(f_h, v)_{\Gamma_D} = (f, v)_{\Gamma_D} \quad \forall v \in V_h$$

and then modify the equations such that $x_i = 0$ if i-th basis function associates a node not in Γ_D. and equation $A\vec{\boldsymbol{\mu}} = \vec{\boldsymbol{\varphi}}$ should be modified such that $\mu_i = x_i$ if i-th basis function associates a node in Γ_D.

5.4 Resolution test

In this section we will do a convergence studies about the PUM discretization, we will choose the function g and f properly such that the true solution for Helmholtz equation (2.3) is the plane wave $u(\boldsymbol{x}) = \exp(ik\boldsymbol{d} \cdot \boldsymbol{x})$ with the frequency $\boldsymbol{d} = (0.8, 0.6)$. We will solve the problem on a sequence of regular refinement meshes (indexed by l, $l = 0$ is the coarsest mesh, and mesh width satisfies $h_l = 2^{-l}$), and with different number of plane waves in (Extend) PUM spaces, and for different wave number in Helmholtz equation, and the computational domain is the unit square $\Omega = (0, 1) \times (0, 1)$. We will use N to indicate the number of plane waves, ($N = 0$ for extend PUM space means the Lagrangian finite element space). Suppose the finite element solution is u_h, we will show the L_2 norm $\|u - u_h\|_2$ and H_1 semi-norm $|u - u_h|_{H^1}$.

Table 5.1, 5.2 and 5.3 shows the results obtained from PUM space, while Table 5.4, 5.5 and 5.6 shows the results obtained from Extend PUM space. And we can observe that

1. using the extend PUM space, using the finer mesh and increasing the number of plane waves can resolve the problem better, since they all enlarge the finite element space.

2. For high wave number, coarse mesh and small number of plane waves can hardly resolve the problem. When $k = 60$, we need mesh on level 5 to resolve the problem if 3 plane waves are provided in the (extend) PUM space, and level 4 if 5 plane waves are provided.

l \ N	3	5	7	9	11	13
0	4.6e-01	8.4e-02	2.2e-03	1.7e-05	1.5e-06	1.6e-07
1	1.1e-01	6.5e-03	2.1e-04	7.4e-07	7.4e-08	7.4e-09
2	1.7e-02	8.2e-04	1.2e-05	2.7e-08	1.1e-09	1.8e-09
3	3.2e-03	1.1e-04	7.3e-07	8.0e-10	9.0e-09	5.3e-09
4	7.2e-04	1.4e-05	4.6e-08	5.1e-10	9.3e-09	1.2e-07
5	1.8e-04	1.8e-06	2.9e-09	3.6e-09	1.8e-06	1.2e-06

(a) L2 error

l \ N	3	5	7	9	11	13
0	3.7e+00	1.0e+00	5.1e-02	4.1e-04	3.8e-05	4.1e-06
1	1.4e+00	2.0e-01	8.0e-03	3.6e-05	3.3e-06	4.0e-07
2	4.8e-01	4.5e-02	8.8e-04	2.1e-06	9.8e-08	1.6e-07
3	1.9e-01	1.1e-02	1.1e-04	1.2e-07	1.4e-06	8.2e-07
4	8.9e-02	2.8e-03	1.4e-05	1.5e-07	2.8e-06	3.3e-05
5	4.3e-02	7.1e-04	1.7e-06	1.8e-06	9.1e-04	5.9e-04

(b) H1 semi error

Table 5.1: Resolution test of PUM space, $k = 6$, solution:plane wave

Solve non-homogeneous Helmholtz equation on extend PUM space

To show that our extend PUM space can resolve non-homogeneous Helmholtz equation, we consider the following problem:

$$\begin{aligned} \Delta u + k^2 u &= 1 \quad \text{in } \Omega \\ \frac{\partial u}{\partial \boldsymbol{n}} - iku &= g \quad \text{on } \partial\Omega \end{aligned} \tag{5.6}$$

where the domain Ω is the unit square $(0,1) \times (0,1)$, and we choose the function g carefully such that the true solution is $u(\boldsymbol{x}) = \exp(ik\boldsymbol{d} \cdot \boldsymbol{x}) + 1/k^2$, again we will solve the problem on a sequence of meshes and with different number of plane waves and mesh width. We can see from Table 5.7 and 5.8 that the extend PUM space resolve the non-homogeneous Helmholtz problem well.

l \ N	3	5	7	9	11	13
0	1.1e+00	1.2e+00	7.0e-01	5.5e-02	6.2e-02	1.1e-01
1	1.2e+00	1.9e+00	9.1e-01	5.7e-03	3.5e-03	8.6e-02
2	1.2e+00	4.1e-01	1.1e-02	1.4e-04	7.2e-05	5.5e-05
3	2.7e-01	3.6e-02	5.4e-04	4.8e-06	1.4e-06	5.5e-07
4	5.6e-02	3.4e-03	3.4e-05	1.4e-07	1.8e-08	4.7e-09
5	1.3e-02	3.3e-04	2.1e-06	4.5e-09	9.0e-10	3.2e-08

(a) L2 error

l \ N	3	5	7	9	11	13
0	2.2e+01	2.4e+01	1.4e+01	1.1e+00	1.3e+00	2.4e+00
1	2.4e+01	3.9e+01	1.8e+01	1.4e-01	1.1e-01	1.7e+00
2	2.5e+01	9.6e+00	6.1e-01	1.1e-02	5.7e-03	4.7e-03
3	6.7e+00	1.6e+00	8.3e-02	6.6e-04	1.9e-04	8.4e-05
4	2.1e+00	4.3e-01	1.0e-02	3.8e-05	5.3e-06	1.4e-06
5	9.0e-01	1.1e-01	1.3e-03	2.3e-06	5.6e-07	2.1e-05

(b) H1 semi error

Table 5.2: Resolution test of PUM space, $k = 20$, solution:plane wave

l \ N	3	5	7	9	11	13
0	1.0e+00	1.2e+00	1.0e+00	3.1e-01	5.4e-01	1.2e+00
1	1.0e+00	1.3e+00	1.1e+00	8.7e-02	1.7e-01	1.5e+00
2	1.2e+00	1.5e+00	1.1e+00	3.9e-02	1.2e-01	6.4e-01
3	1.4e+00	2.0e+00	1.6e-01	1.8e-03	3.7e-03	2.8e-02
4	1.3e+00	7.7e-01	6.0e-03	4.7e-05	2.0e-05	2.1e-05
5	4.4e-01	1.0e-01	2.1e-04	1.2e-06	2.8e-07	1.6e-07

(a) L2 error

l \ N	3	5	7	9	11	13
0	6.1e+01	7.2e+01	6.2e+01	1.8e+01	3.3e+01	7.3e+01
1	6.1e+01	7.6e+01	6.4e+01	5.3e+00	1.1e+01	8.7e+01
2	7.0e+01	8.8e+01	6.9e+01	2.4e+00	7.4e+00	3.9e+01
3	8.7e+01	1.2e+02	1.0e+01	1.7e-01	2.5e-01	1.7e+00
4	8.0e+01	4.8e+01	1.0e+00	1.2e-02	5.3e-03	6.7e-03
5	2.8e+01	7.7e+00	1.3e-01	6.2e-04	1.5e-04	1.0e-04

(b) H1 semi error

Table 5.3: Resolution test of PUM space, $k = 60$, solution:plane wave

l \ N	0	3	5	7	9	11	13
0	6.9e-01	2.7e-01	6.2e-02	1.9e-03	1.2e-05	1.5e-06	1.6e-07
1	3.7e-01	5.9e-02	5.5e-03	1.8e-04	6.5e-07	6.9e-08	6.9e-09
2	1.4e-01	7.7e-03	5.8e-04	1.0e-05	2.1e-08	1.0e-09	2.7e-09
3	3.9e-02	9.9e-04	7.0e-05	6.6e-07	5.3e-10	1.3e-08	2.2e-09
4	1.0e-02	1.3e-04	7.9e-06	4.3e-08	3.6e-10	1.3e-08	5.3e-08
5	2.5e-03	1.6e-05	7.7e-07	2.8e-09	8.5e-09	7.0e-07	5.5e-06

(a) L2 error

l \ N	0	3	5	7	9	11	13
0	5.0e+00	2.5e+00	8.8e-01	4.6e-02	3.1e-04	3.8e-05	4.2e-06
1	3.6e+00	9.9e-01	1.8e-01	7.4e-03	3.3e-05	3.2e-06	3.8e-07
2	1.9e+00	2.9e-01	3.5e-02	7.9e-04	1.8e-06	9.3e-08	2.6e-07
3	9.5e-01	7.9e-02	7.4e-03	1.0e-04	9.6e-08	2.3e-06	3.7e-07
4	4.7e-01	2.1e-02	1.6e-03	1.3e-05	1.3e-07	4.2e-06	1.7e-05
5	2.3e-01	5.2e-03	3.2e-04	1.6e-06	5.4e-06	4.1e-04	3.2e-03

(b) H1 semi error

Table 5.4: Resolution test of Extend PUM space, $k = 6$, solution:plane wave

l \ N	0	3	5	7	9	11	13
0	1.0e+00	1.1e+00	1.2e+00	7.5e-01	5.6e-02	1.1e-01	3.2e-01
1	1.0e+00	1.2e+00	1.5e+00	1.1e-01	1.9e-03	2.1e-03	3.1e-03
2	1.0e+00	6.5e-01	2.8e-01	1.0e-02	1.2e-04	5.8e-05	4.5e-05
3	8.9e-01	1.2e-01	2.6e-02	5.2e-04	3.4e-06	1.2e-06	4.8e-07
4	3.4e-01	1.3e-02	2.5e-03	3.3e-05	8.6e-08	1.6e-08	4.4e-09
5	9.4e-02	1.2e-03	2.5e-04	2.1e-06	2.7e-09	8.7e-10	1.1e-07

(a) L2 error

l \ N	0	3	5	7	9	11	13
0	2.0e+01	2.2e+01	2.4e+01	1.5e+01	1.1e+00	2.3e+00	6.6e+00
1	2.0e+01	2.4e+01	2.7e+01	2.8e+00	7.9e-02	9.1e-02	1.6e-01
2	2.1e+01	1.4e+01	7.0e+00	5.8e-01	9.7e-03	5.2e-03	4.3e-03
3	1.9e+01	3.5e+00	1.4e+00	7.8e-02	5.6e-04	1.8e-04	8.1e-05
4	8.3e+00	9.0e-01	3.4e-01	1.0e-02	3.2e-05	5.0e-06	1.4e-06
5	3.2e+00	2.4e-01	8.8e-02	1.3e-03	1.9e-06	5.5e-07	7.7e-05

(b) H1 semi error

Table 5.5: Resolution test of Extend PUM space, $k = 20$, solution:plane wave

l \ N	0	3	5	7	9	11	13
0	1.0e+00	1.0e+00	1.1e+00	1.1e+00	1.1e+00	1.1e+00	1.1e+00
1	1.0e+00	1.0e+00	1.2e+00	1.1e+00	9.0e-01	1.1e+00	1.2e+00
2	1.0e+00	1.2e+00	1.5e+00	1.1e+00	4.4e-02	8.8e-02	2.1e-01
3	1.0e+00	1.3e+00	1.5e+00	8.4e-02	7.9e-04	9.3e-04	2.5e-03
4	1.2e+00	9.9e-01	5.3e-01	5.6e-03	4.3e-05	1.9e-05	2.0e-05
5	1.3e+00	1.5e-01	5.8e-02	2.1e-04	7.6e-07	2.7e-07	1.6e-07

(a) L2 error

l \ N	0	3	5	7	9	11	13
0	6.0e+01	6.2e+01	6.6e+01	6.3e+01	6.4e+01	6.4e+01	6.7e+01
1	6.0e+01	6.2e+01	7.3e+01	6.6e+01	5.4e+01	6.4e+01	7.0e+01
2	6.0e+01	7.1e+01	8.8e+01	6.6e+01	2.7e+00	5.3e+00	1.3e+01
3	6.0e+01	7.9e+01	8.8e+01	6.3e+00	1.3e-01	1.4e-01	4.0e-01
4	7.1e+01	6.0e+01	3.3e+01	9.8e-01	1.1e-02	5.1e-03	6.5e-03
5	8.0e+01	1.0e+01	4.8e+00	1.3e-01	5.3e-04	1.5e-04	1.0e-04

(b) H1 semi error

Table 5.6: Resolution test of Extend PUM space, $k = 60$, solution:plane wave

l \ N	5	7	9	11
0	6.2e-02	1.9e-03	1.2e-05	1.5e-06
1	5.5e-03	1.8e-04	6.5e-07	6.9e-08
2	5.8e-04	1.0e-05	2.1e-08	1.0e-09
3	7.0e-05	6.6e-07	5.2e-10	5.0e-09
4	7.9e-06	4.3e-08	8.7e-10	7.3e-09

(a) L2 error

l \ N	5	7	9	11
0	8.7e-01	4.6e-02	3.1e-04	3.8e-05
1	1.8e-01	7.4e-03	3.3e-05	3.2e-06
2	3.5e-02	7.9e-04	1.8e-06	9.3e-08
3	7.4e-03	9.9e-05	9.6e-08	8.8e-07
4	1.6e-03	1.3e-05	3.2e-07	2.3e-06

(b) H1 semi error

Table 5.7: Resolution test for non-homogeneous Helmholtz problem with $k = 6$

l \ N	5	7	9	11
0	1.2e+00	7.5e-01	5.6e-02	1.1e-01
1	1.4e+00	1.1e-01	1.9e-03	2.1e-03
2	2.8e-01	1.0e-02	1.1e-04	5.8e-05
3	2.6e-02	5.2e-04	3.4e-06	1.2e-06
4	2.5e-03	3.3e-05	8.6e-08	1.6e-08

(a) L2 error

l \ N	5	7	9	11
0	2.4e+01	1.5e+01	1.1e+00	2.3e+00
1	2.7e+01	2.8e+00	7.9e-02	9.1e-02
2	7.0e+00	5.8e-01	9.7e-03	5.2e-03
3	1.4e+00	7.8e-02	5.6e-04	1.8e-04
4	3.4e-01	1.0e-02	3.2e-05	5.0e-06

(b) H1 semi error

Table 5.8: Resolution test for non-homogeneous Helmholtz problem with $k = 20$

Chapter 6

PUM wave-ray method

We know that standard multigrid method can't remove the error of the form $v(\boldsymbol{x})\exp(ik\boldsymbol{d}\cdot\boldsymbol{x})$ with $\|\boldsymbol{d}\|\approx 1$, where $v(\boldsymbol{x})$ is the smooth ray function, and the wave-ray method introduces the ray cycle to deal with these problematic errors. Here we move the residual after the wave cycle to the PUM spaces to obtain the characteristic error correction.

First let's introduce the ingredients to perform the PUM ray cycle, the algorithm detail and numerical experiments will be presented in section 7.3. (the explanation will be based on PUM space, the extension to the extend PUM space is trivial). We assume a hierarchy of nested meshes $\mathcal{T}_0 \prec \mathcal{T}_1 \prec \cdots \prec \mathcal{T}_L$ created by uniform, regular refinement. Let $\mathcal{S}_l$ denote the Lagrangian finite element space $\mathcal{S}_1^0(\mathcal{T}_l)$. Let $\mathcal{V}_l$ denote the set of vertices of $\mathcal{T}_l$, and assume $\sharp|\mathcal{V}_l| = n_l$ and $\mathcal{V}_l = \{\boldsymbol{p}_1^l, \cdots, \boldsymbol{p}_{n_l}^l\}$. Write b_i^l for the piecewise linear nodal basis function ("tent function") associated with vertex $\boldsymbol{p}_i^l \in \mathcal{V}_l$. Let N_l be the number of plane waves on mesh level l and $N_l = 2^{L+1-l}$, $l = 0, \cdots, L-1$ and again write

$$\begin{aligned} \boldsymbol{d}_t^l &= (\cos\tfrac{2\pi(t-1)}{N_l}, \sin\tfrac{2\pi(t-1)}{N_l}) \quad t = 1, \cdots, N_l \\ e_t^l(\boldsymbol{x}) &= \exp(ik\boldsymbol{d}_t^l\cdot\boldsymbol{x}) \end{aligned}$$

and define the wave modulated partition of unity space according to

$$\begin{aligned} W_L &:= \mathcal{S}_1^0(\mathcal{T}_L) \\ W_l &:= \operatorname{span}\{b_i^l(\boldsymbol{x})e_t^l(\boldsymbol{x}) | \boldsymbol{p}_i^l \in \mathcal{V}_l, t = 1, \cdots, N_l\}, l = 0, \cdots, L-1 \end{aligned} \tag{6.1}$$

Here the space W_L is still the Lagrangian finite element space, and the residual obtained at this mesh will be transferred to the PUM spaces to start the ray cycle, we need the transfer operator between these spaces, and coarse grid operator. If we have defined $I_l^{l+1} : W_l \mapsto W_{l+1}$, then operator $I_{l+1}^l : W_{l+1} \mapsto W_l$ can be defined as the transpose of I_l^{l+1}, and the coarse grid

operator is built by Galerkin projection

$$A_l = I_{l+1}^l A_{l+1} I_l^{l+1}$$

Following we will focus on the construction of the operator I_l^{l+1} and its implementation in LehrFEM++.

6.1 Transfer operator

Prolongation operator $Q_l^{l+1} : \mathcal{S}_l \mapsto \mathcal{S}_{l+1}$

Suppose

$$Q_l^{l+1} b_j^l(\boldsymbol{x}) = \sum_{k=1}^{n_{l+1}} q_{kj} b_k^{l+1}(\boldsymbol{x})$$

then,

$$Q_l^{l+1}[b_1^l, \cdots, b_{n_l}^l] = [b_1^{l+1}, \cdots, b_{n_{l+1}}^{l+1}][q_{kj}]_{k,j=1}^{n_{l+1},n_l}$$

we can write the prolongation operator in matrix form $Q_l^{l+1} = [q_{kj}]_{k,j=1}^{n_{l+1},n_l}$, Since the mesh $\{\mathcal{T}_l\}$ are created by uniform refinement, we have that for $l < L$, $\mathcal{S}_l \subset \mathcal{S}_{l+1}$, since b_k^{l+1} is tent function associated with vertex $\boldsymbol{p}_k^{l+1}$, we can conclude that

$$b_k^{l+1}(\boldsymbol{p}_s^{l+1}) = \delta_{ks}, \quad q_{kj} = b_j^l(\boldsymbol{p}_k^{l+1})$$

and

$$b_j^l(\boldsymbol{p}_k^{l+1}) = \begin{cases} 1, & \text{if } \boldsymbol{p}_j^l = \boldsymbol{p}_k^{l+1} \\ \frac{1}{2}, & \text{if segment } \boldsymbol{p}_j^l \boldsymbol{p}_k^{l+1} \text{ is an edge in } \mathcal{T}_{l+1} \\ 0, & \text{otherwise} \end{cases} \tag{6.2}$$

In LehrFEM++, if the mesh hierarchy is generated by uniform refinement, for a vertex $\boldsymbol{p}_k^{l+1}$ in mesh $\mathcal{V}_{l+1}$, we can get its parent entity in mesh $\mathcal{V}_l$, its parent entity is either a vertex or an edge (in this case, $\boldsymbol{p}_k^{l+1}$ is generated in the refinement process), so we can know the value of $b_j^l(\boldsymbol{p}_k^{l+1})$ easily, and thus the operator Q_l^{l+1}.

```
1  HE_FEM::SpMat_t HE_FEM::prolongation_lagrange(size_type l) {
2      LF_ASSERT_MSG(l >= 0 && l < L_, "l in prolongation should be
       ↪  smaller to L_");
3      auto coarse_mesh = getmesh(l);
4      auto fine_mesh   = getmesh(l+1);
5
6      auto coarse_dofh =
       ↪  lf::assemble::UniformFEDofHandler(coarse_mesh,
7                           {{lf::base::RefEl::kPoint(), 1}});
```

```
8       auto fine_dof    =
        ↪  lf::assemble::UniformFEDofHandler(fine_mesh,
9                          {{lf::base::RefEl::kPoint(), 1}});
10
11      size_type n_c = coarse_dofh.NumDofs();
12      size_type n_f = fine_dof.NumDofs();
13
14      Mat_t M = Mat_t::Zero(n_c, n_f);
15      for(const lf::mesh::Entity* edge: fine_mesh->Entities(1)) {
16          nonstd::span<const lf::mesh::Entity* const> points =
            ↪  edge->SubEntities(1);
17          size_type num_points =
            ↪  (*edge).RefEl().NumSubEntities(1);
18          LF_ASSERT_MSG((num_points == 2),
19              "Every EDGE should have 2 kPoint subentities");
20          for(int j = 0; j < num_points; ++j) {
21              auto parent_p = mesh_hierarchy_->ParentEntity(l+1,
                ↪  *points[j]); // parent entity of current point
22              if(parent_p->RefEl() == lf::base::RefEl::kPoint()) {
23                  // it's parent is also a NODE. If the point in
                    ↪  finer mesh does not show in coarser mesh,
24                  // then it's parent is an EDGE
25                  M(coarse_mesh->Index(*parent_p),
                    ↪  fine_mesh->Index(*points[j])) = 1.0;
26                  M(coarse_mesh->Index(*parent_p),
                    ↪  fine_mesh->Index(*points[1-j])) = 0.5;
27              }
28          }
29      }
30      return (M.transpose()).sparseView();
31  }
```

Above is the C++ implementation of the prolongation matrix Q_l^{l+1}. Line 14 creates the object to store the transpose of the prolongation operator (in code, $n_f = n_{l+1}, n_c = n_l$). Line 15 iterates through edges to find the connecting edge pairs in $\mathcal{T}_{l+1}$, line 22 determines whether the current node also appear in the mesh $\mathcal{T}_l$, line 25 and 26 deal with the first two cases in (6.2).

Operator $I_{L-1}^L : W_{L-1} \mapsto W_L$

W_L is the Lagrangian finite element space $\mathcal{S}_1^0(\mathcal{T}_L)$ while W_{L-1} is the plane wave PUM space, we can define the operator I_{L-1}^L as the nodal projection,

that is for $u \in W_{L-1}$, $I_{L-1}^{L} u$ satisfies

$$I_{L-1}^{L} u(\boldsymbol{p}) = u(\boldsymbol{p}), \quad \text{for } \boldsymbol{p} \in \mathcal{V}_L$$

Assume $b_j^{L-1}(\boldsymbol{x}) = \sum_{k=1}^{n_L} q_{kj} b_k^L(\boldsymbol{x})$, and let matrix $\mathcal{A} = [\alpha_{ij}] = [e_j^{L-1}(\boldsymbol{p}_i^L)]_{i,j=1}^{n_L, N_{L-1}}$ then for $b_j^{L-1} e_t^{L-1} \in W_{L-1}$,

$$b_j^{L-1}(\boldsymbol{p}_i^L) e_t^{L-1}(\boldsymbol{p}_i^L) = q_{ij}\alpha_{it}$$

which means

$$I_{L-1}^{L}(b_j^{L-1} e_t^{L-1}) = \sum_{i=1}^{n_L} q_{ij}\alpha_{it} b_i^L$$

and we can write the operator I_{L-1}^{L} in the matrix form:

$$I_{L-1}^{L} = \begin{pmatrix} q_{11}\mathcal{A}_{1\cdot} & q_{12}\mathcal{A}_{1\cdot} & \cdots & q_{1,n_{L-1}}\mathcal{A}_{1\cdot} \\ q_{21}\mathcal{A}_{2\cdot} & q_{22}\mathcal{A}_{2\cdot} & \cdots & q_{2,n_{L-1}}\mathcal{A}_{2\cdot} \\ & & \vdots & \\ q_{n_L,1}\mathcal{A}_{n_L\cdot} & q_{n_L,2}\mathcal{A}_{n_L,\cdot} & \cdots & q_{n_L,n_{L-1}}\mathcal{A}_{n_L,\cdots} \end{pmatrix}$$

where $\mathcal{A}_{i\cdot}$ indicates the i-th row of matrix $\mathcal{A}$.

Operator $I_l^{l+1} : W_l \mapsto W_{l+1}, l < L-1$

Recall that $N_l = 2^{L+1-l}$, so $N_l = 2N_{l+1}$, note that for $t = 1, 2, \cdots, N_{l+1}$

$$\begin{aligned} \boldsymbol{d}_{2t-1}^l &= (\cos\frac{2\pi(2t-2)}{N_l}, \sin\frac{2\pi(2t-2)}{N_l}) \\ &= (\cos\frac{2\pi(t-1)}{N_{l+1}}, \sin\frac{2\pi(t-1)}{N_{l+1}}) \\ &= \boldsymbol{d}_t^{l+1} \end{aligned}$$

so $e_{2t-1}^l = e_t^{l+1}$, suppose $b_i^l = \sum_{j=1}^{n_{l+1}} q_{ji} b_j^{l+1}$ then we can write

$$b_i^l e_{2t-1}^l = \sum_{j=1}^{n_{l+1}} q_{ji} b_j^{l+1} e_t^{l+1} \quad i = 1, \cdots, n_l \quad t = 1, \cdots, N_{l+1} \tag{6.3}$$

Inspired by [10], we employ an "exponential interpolation" for e_{2t}^l, which we write in a different form:

$$\begin{aligned} e_{2t}^l &= \exp(ik\boldsymbol{d}_{2t}^l \cdot \boldsymbol{x}) \\ &= \exp(ik\boldsymbol{d}_t^{l+1} \cdot \boldsymbol{x}) \exp(ik(\boldsymbol{d}_{2t}^l - \boldsymbol{d}_t^{l+1}) \cdot \boldsymbol{x}) \\ &:= e_t^{l+1}\delta_t \end{aligned}$$

where $\delta_t(\boldsymbol{x}) = \exp(ik(\boldsymbol{d}_{2t}^l - \boldsymbol{d}_t^{l+1}) \cdot \boldsymbol{x})$, which is less oscillatory compared to e_{2t}^l and we can do a nodal projection into $\mathcal{S}_{l+1}$, so we write

$$I_l^{l+1}(b_i^l e_{2t}^l) = \sum_{j=1}^{n_{l+1}} \alpha_j^{it} b_j^{l+1} e_t^{l+1} \quad i = 1, \cdots, n_l \quad t = 1, \cdots, N_{l+1} \tag{6.4}$$

where

$$\alpha_j^{ik} = q_{ji} \delta_t(\boldsymbol{p}_j^{l+1})$$

(6.3) and (6.4) together give us the transfer operator I_l^{l+1}.

```
HE_PUM::SpMat_t HE_PUM::prolongation(size_type l) {
    LF_ASSERT_MSG((l < L_),
        "in prolongation, level should smaller than" << L_);
    double pi = std::acos(-1.);
    auto Q = prolongation_lagrange(l);
    int n1 = Q.cols(), n2 = Q.rows(); // n1: n_l, n2: n_{l+1}
    int N1 = num_planewaves_[l], N2 = num_planewaves_[l+1]; //
    ↪  N1: N_l, N2: N_{l+1}

    auto mesh = getmesh(l+1);  // fine mesh
    auto dofh = lf::assemble::UniformFEDofHandler(mesh,
    ↪  {{lf::base::RefEl::kPoint(), 1}});

    SpMat_t res(n2 * N2, n1 * N1); // transfer operator
    std::vector<triplet_t> triplets;

    for(int outer_idx = 0; outer_idx < Q.outerSize();
    ↪  ++outer_idx) {
        for(SpMat_t::InnerIterator it(Q, outer_idx); it; ++it) {
            int i = it.row();
            int j = it.col();
            Scalar qij = it.value();
            for(int t = 1; t <= N2; ++t) {
                triplets.push_back(triplet_t(i*N2+t-1,
                ↪  j*N1+2*t-2, qij));
            }
            const lf::mesh::Entity& p_i = dofh.Entity(i);
            coordinate_t pi_coordinate =
            ↪  lf::geometry::Corners(*p_i.Geometry()).col(0);
            for(int t = 1; t <= N2; ++t) {
                Eigen::Vector2d d1, d2;
                d1 << std::cos(2*pi*(2*t-1)/N1),
                ↪  std::sin(2*pi*(2*t-1)/N1); // d_{2t}^l
```

```
28                  d2 << std::cos(2*pi*(  t-1)/N2), std::sin(2*pi*(
                    ↪  t-1)/N2); // d_{t}^{l+1}
29                  Scalar tmp = qij *
                    ↪  std::exp(1i*k_*(d1-d2).dot(pi_coordinate));
30                  triplets.push_back(triplet_t(i*N2+t-1,
                    ↪  j*N1+2*t-1, tmp));
31              }
32          }
33      }
34      res.setFromTriplets(triplets.begin(), triplets.end());
35      return res;
36  }
```

Above is the C++ implementation of the operator $I_l^{l+1}, l < L-1$, note that here the index for b_i^l starts from 0 and index for e_t^l starts from 1, so $b_i^l e_t^l$ has global index $i * N_l + t - 1$. Line 15 and 16 loop from the entry of Q (prolongation operator betweenS_l and S_{l+1}). Line 20-21 implement (6.3). Line 25-33 implement (6.4).

Chapter 7

Numerical Experiments

In this chapter, we will examine our PUM Wave-Ray methods for solving Helmholtz problem (2.3) with some manufacture solutions. We can choose g and f properly in (2.3) such that the exact solution is

1. Plane wave of form $u(\boldsymbol{x}) = \exp(ik\boldsymbol{d}\cdot\boldsymbol{x})$, where $\boldsymbol{d} = (0.8, 0.6)$.
2. Fundamental solution: $u(\boldsymbol{x}) = \frac{i}{4}H_0^{(1)}(ik\|\boldsymbol{x}-\boldsymbol{c}\|)$, where $H_0^{(1)}$ is the Hankel function of first kind and $\vec{\boldsymbol{c}}$ is a point outside the domain Ω.
3. Spherical wave(in polar coordiante): $u(r,\theta) = J_{|l|}(kr)\exp(il\theta)$, $l \in \mathbf{Z}$

To ensure the correctness of our code, we will first do a code validation, which includes the validation of the prolongation(transfer) operator and to solve the problem on the single mesh with Lagrangian finite element space and plane wave (extend) PUM spaces to verify that finest and coarser mesh can resolve the problem correctly. The resolution test for (extend) PUM spaces can be found in §5.4.

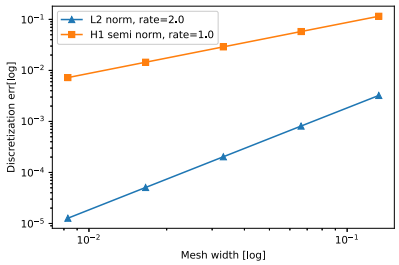
7.1 Code Validation

Resolution test in standard Lagrangian finite element space

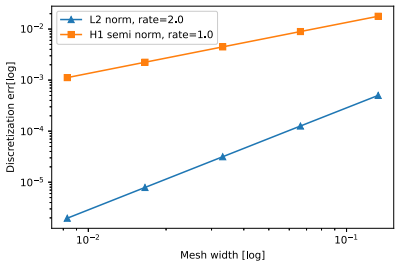
We solve the Helmholtz problem on a sequence of meshes $\mathcal{T}_l$ and the finite element space is the standard Lagrangian finite element space $\mathcal{S}_l$, the true solutions are manufacture solutions introduced above, and we choose the wave number $k = 2$. Suppose u_h is the finite element solution and u is the exact solution, we can observe from Figure 7.1 that

$$\|u_h - u\|_\Omega = \mathcal{O}(h^2), \quad |u_h - u|_{H_1(\Omega)} = \mathcal{O}(h)$$

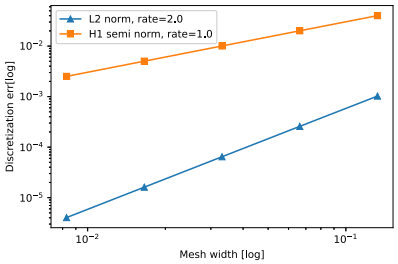
satisfies in all three cases, where h is the mesh width, which means that we solved the problem correctly.



(a) solution: plane wave



(b) solution: fundamental solution



(c) solution: spherical wave

Figure 7.1: Solved on Lagrangian finite element space.

Test transfer operator

Here we test the prolongation operator $Q_l^{l+1} : \mathcal{S}_l \mapsto \mathcal{S}_{l+1}$ and $I_l^{l+1} : EW_l \mapsto EW_{l+1}$, where $\mathcal{S}_l$ is the standard Lagrangian finite element space $\mathcal{S}_1^0(\mathcal{T}_l)$ and EW_l is the extend PUM space. The prolongation(transfer) operator is very important in the multi-grid method, and it should preserve the function being transferred as much as possible, i.e. $\|I_l^{l+1} v_l\|_{L^2(\Omega)}$ should be very close to $\|v_l\|_{L^2(\Omega)}$. Our test shows that $\|Q_l^{l+1} v_l\|_{L^2(\Omega)} / \|v_l\|_{L^2(\Omega)} = 1$, which is exactly what we want since the standard Lagrangian finite element space is nested ($\mathcal{S}_l \subset \mathcal{S}_{l+1}$), Q_l^{l+1} is actually an embedding operator. As for the operator I_l^{l+1}, Table 7.1 gives the result of $\|v_{l+1}\|_{L^2(\Omega)} / \|v_l\|_{L^2(\Omega)}$, where $v_0 \in EW_l$ is a randomly generated function and $v_{l+1} = I_l^{l+1} v_l$, as we can see, all the ratios are very close to 1, we can conclude that the operator I_l^{l+1} is implemented correctly and its a preserves the function being transferred very well in terms of L^2 norm.

$\|v_{l+1}\|_{L^2(\Omega)} / \|v_l\|_{L^2(\Omega)}$	0	1	2	3	4	5
l	0.995	1.000	0.999	0.998	0.998	1.000

Table 7.1: Test transfer operator between extend PUM spaces

Multi-grid method for Laplace problem

Here we consider solving the following Dirichlet problem with the standard multi-grid method

$$\begin{aligned} \Delta u &= 0 \quad \text{in } \Omega \\ u &= g \quad \text{on } \partial\Omega \end{aligned} \tag{7.1}$$

where $\Omega = (0,1) \times (0,1)$, and we choose the exact solution to be $u(x,y) = \exp(x+iy)$. The purpose is to validate the transfer operator Q_l^{l+1} again and the procedure of the multi-grid method. The mesh width of the finest mesh is $h_L = \frac{1}{32}$, and we use the three coarse meshes to perform the standard v-cycle. Let $v^{(0)}$ be the randomly chosen initial value, $v^{(i)}$ be the value in the ith iterate, u_h be the finite element solution and $e^{(i)} = u_h - v^{(i)}$ be the error function in the ith iterate. From Table 7.2, we can conclude that the convergence rate of this 4-mesh v-cycle is about 0.103, which coincides the result obtained doing the power iteration to this v-cycle, so the implementation of the v-cycle and the prolongation operator is correct.

i	1	2	3	4	5
$\|e^{(i+1)}\|_{L^2(\Omega)} / \|e^{(i)}\|_{L^2(\Omega)}$	0.112	0.100	0.103	0.103	0.103

Table 7.2: L^2 norm of error function in standard multi-grid method

7.2 Apply Standard Multigrid to Helmholtz problem

We know that standard multigrid can't resolve Helmholtz problem when the wave number is large, we will compute the convergence factor by power iteration to show this. Table 7.3 shows some results with different wave number and number of meshes (levels), the domain is chosen to be unit square and the finest mesh has mesh width 1/32. We perform 2 sweeps of Gauss-Seidel relaxation on each level. We can see that the convergence degrades as increasing of wave number and levels, since we know that G-S relaxation is unstable on levels with k times mesh width falls in an intermediate range.

$L\backslash k$	1	2	3	4	5	6	7
4	0.10	0.10	0.11	0.11	0.12	0.15	0.19
5	0.10	0.10	0.11	0.14	0.28	0.59	-
6	0.11	0.12	0.38	-	-	-	-

Table 7.3: Convergence factor of using standard multigrid to solve Helmholtz equation, a dash denotes divergence, k denotes the wave number, L denotes the number of meshed.

7.3 Extend PUM wave-ray multigrid

Before implementing the multigrid method, we need to decide the relaxation method performed on each levels. From Table 7.4, we can observe that for operator constructed based on Lagrangian finite element space, if kh(k is the wave number while h denotes the mesh width) is between 2 and 8, the Gauss-Seidel(G-S) relaxation has a bad divergence, it will amplify the errors, so G-S relaxation should not be performed, while on extend PUM space, G-S relaxation should only be used on fine meshes. These observation also agrees with the conclusion by the Brandt and Livshits (§7 in [2]), in practice, we only use G-S relaxation when $kh < 1.5$ or $kh > 8.0$ in wave cycle, and use block G-S relaxation when $kh < 1.0$ in ray cycle, otherwise no relaxation is performed.

kh	16	8	4	2	1	0.5
μ_1	0.37	0.77	div	2.67	1.26	1.06
μ_2	div	div	div	div	1.45	-

Table 7.4: μ_1 and μ_2 are the convergence factor for Gauss-Seidel on different meshes for operator constructed based on Lagrangian finite element space and Extend PUM space respectively.

Algorithm 3 describes the solver used for the following experiments. Index the levels used in the wave cycle from coarsest to finest as 0 to L_w, let L_r denotes the "wave-to-ray switching" level, that is, on the L_r-th level of second leg of the wave cycle, the residual is transferred to the (extend) PUM space to do another v-cycle (ray cycle), and let n_r to denote the number of extra (extend) PUM spaces. The space $\mathcal{S}_1^0(\mathcal{T}_{L_r})$ act as the space W_L (here L should be $n_r + 1$) in (6.1).

Figure 7.2 to 7.5 shows the convergence factor study of the PUM wave-ray algorithm. The experiments are carried out in different domains (unit square, unit square with a square hole, unit square with a triangle hole), the coarsest mesh width is $h_0 = 1$, and mesh with index l has mesh width $h_l = 2^{-l}$, the exact solution is chosen to be plane wave, and the convergence factor is obtained from the power iteration (run the multigrid with zero right hand side and normalize solution after each iteration). The experiments produce the results in Figure 7.2, 7.3 and 7.5 use only 1 extra extend PUM space, that is, the ray-cycle is a 2-mesh correction scheme, while 7.4 uses 2 extra extend PUM spaces. We have the following observations:

- the convergence factor increase along with the increase of wave number, when the "wave-to-ray switch" level L_r is set to 3, the wave-ray method diverges when $k \gtrsim 22$, while the algorithm stay convergence when $L_r = 4$ if $n_r = 1$ in both cases, so we should choose a fine mesh

Algorithm 3: PUM Wave-Ray algorithm

L_w: index for the finest mesh, L_r: "wave-to-ray switch" level, n_r: number of extra extend PUM spaces;
u_{L_w} denotes the approximate solution we want, and it is randomly initialized, $u_l (l < L_w)$ is initialized to zero
for $l = L_w; l \geq 1; l = l - 1$ **do**
    Relaxation* $A_l \vec{\boldsymbol{u}}_l = \vec{\boldsymbol{\varphi}}_l$;
    Residual calculation $\varphi_{l-1} = I_l^{l-1}(\vec{\boldsymbol{\varphi}}_l - A_l \vec{\boldsymbol{u}}_l)$;
end
Coarsest wave grid($l = 0$), direct solve $A_0 \vec{\boldsymbol{u}}_0 = \vec{\boldsymbol{\varphi}}_0$
for $l = 1; l \leq L_w; l = l + 1$ **do**
    $\vec{\boldsymbol{u}}_l = \vec{\boldsymbol{u}}_l + I_{l-1}^l \vec{\boldsymbol{u}}_{l-1}$;
    Relaxation* $A_l \vec{\boldsymbol{u}}_l = \varphi_l$;
    if $l == L_r$, transfer residual $\vec{\boldsymbol{\varphi}}_l - A_l \vec{\boldsymbol{u}}_l$ to the extend PUM space to do another v-cycle(ray cycle) with n_r extra extend PUM spaces, then transfer correction back to wave level l;
end
* relaxation strategy: if $kh < 1.5$ or $kh > 8$, perform m_1 steps of Gauss-Seidel relaxation, otherwise no relaxation, as for relaxation in extend PUM spaces, block G-S (group the d.o.f with the same node together) is performed only when $kh < 1$.

as "wave-to-ray switch" mesh ($kh_{L_r} < 2$ at least).

- Figure 7.3 and Figure 7.5 have a similar trend and they all set $L_r = 4$, so the choice of L_r really matters, it determines whether the problematic error components can be eliminated.
- The domain also influences the convergence.
- When $L_w = 5, L_r = 4$, adding the number of extend PUM spaces seems have no improvement, when $k \gtrsim 24$, it even worse the convergence compared to only one extend PUM space.

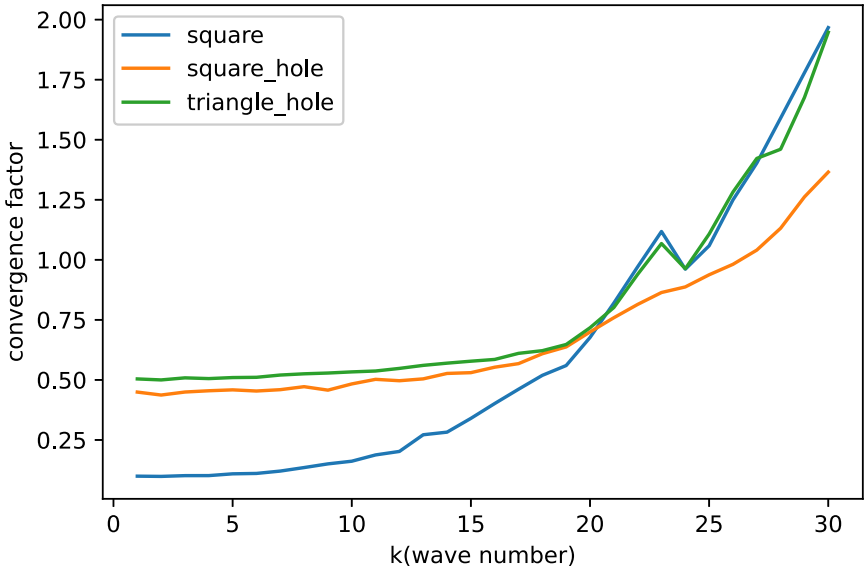


Figure 7.2: Convergence factor, here $L_w = 5$ means there are 6 meshes in wave cycle, and "wave-to-ray switching" level is $L_r = 3$, and $n_r = 1$ extend PUM space is used (2 mesh ray cycle) , and $m_1 = 2$ iteration of relaxation steps. Three curves corresponding to the computational domain to be unit square, unit square with a square hole, unit square with a triangle hole.

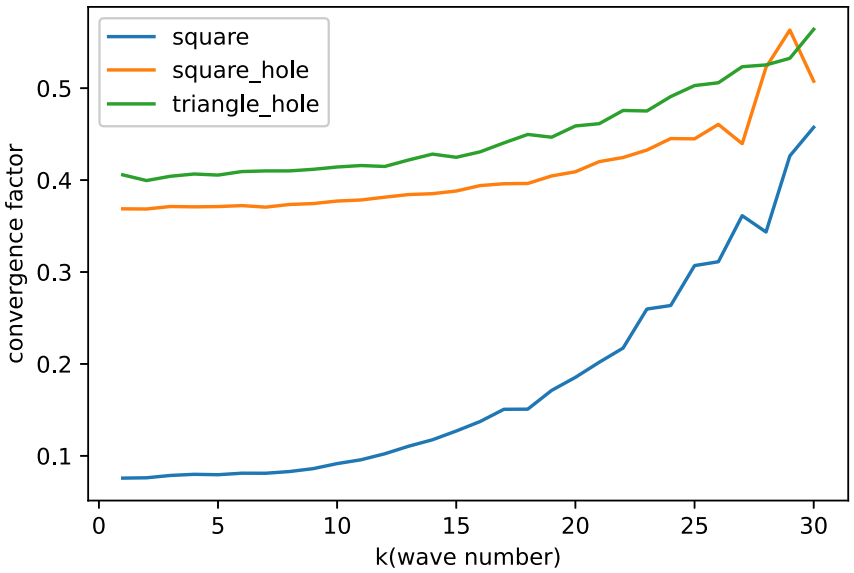


Figure 7.3: Convergence factor, $L_w = 5, L_r = 4, n_r = 1, m_1 = 2$.

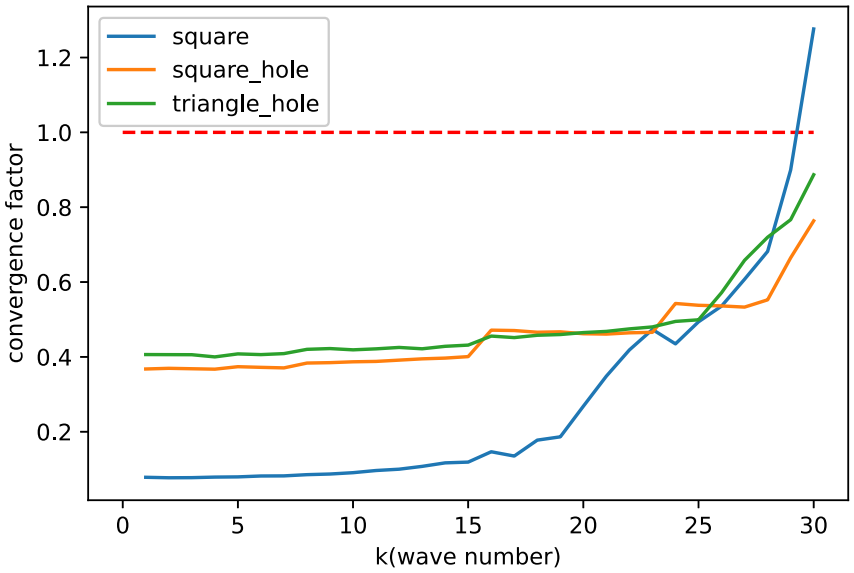


Figure 7.4: Convergence factor, $L_w = 5, L_r = 4, n_r = 2, m_1 = 2$, that is the ray cycle consists 3 meshes.

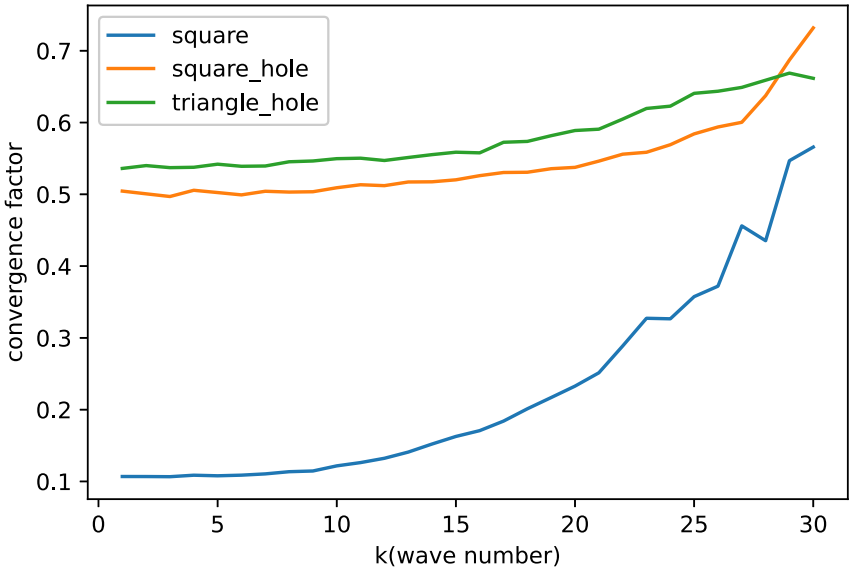


Figure 7.5: Convergence factor, $L_w = 6, L_r = 4, n_r = 1, m_1 = 2$.

Chapter 8

Stable smoothing method

From the discussion in 3.4, we know that we should not implement Gauss-Seidel relaxation on intermediate meshes. In this chapter we will use two stable smoothing methods, GMRES and local smoothing with impedance boundary condition, on intermediate meshes, and use the multigrid method with new smoothing strategy as the preconditioner for outer GMRES iteration [4].

8.1 GMRES

For a linear system of equations $A\vec{\mu} = \vec{\varphi}$, the Krylov subspace methods seek an approximate solution in the Krylov space

$$K_n(A, \vec{r}) = \text{span}\{\vec{r}, A\vec{r}, \cdots, A^{n-1}\vec{r}\}$$

where $r = \vec{\varphi} - A\vec{\mu_0}$ is the initial residual with initial guess $\vec{\mu_0}$. Generalized minimal residual method (GMRES) seeks $\vec{\mu}_n \in K_n$ that minimizes the Euclidean norm of the residual

$$\vec{\mu}_n = \text{argmin}_{\vec{x} \in K_n} \|\vec{\varphi} - A\vec{x}\|$$

Arnoldi iteration is used to find the orthonormal basis of K_n.

Let M be a preconditioner and change the linear equations into

$$AM^{-1}\vec{y} = \vec{\varphi}, \quad \vec{\mu} = M^{-1}\vec{y}$$

if the preconditioner is chosen properly, we may improve the convergence behavior. And when we use the GMRES to solve the preconditioned system, in every iteration we need to solve an equation of form $M\vec{z} = \vec{v}$, and this equation may also be solved by an iterative method, if we use a nonlinear iterative method, then the preconditioner M actually differs in every iteration, in this situation we need to use Flexible GMRES (FGMRES), the detail can be found in [14]. The C++ implementation of GMRES for original system is presented at appendix A.1.

8.2 Local smoothing with impedance boundary condition

Domain decomposition

Here we present again the Helmholtz boundary value problem posed on a two-dimensional domain $\Omega \subset \mathbb{R}^2$:

$$\begin{aligned} -\Delta u - k^2 u &= 0 \quad \text{in } \Omega \\ \frac{\partial u}{\partial \boldsymbol{n}} - iku &= g \quad \text{on } \Gamma_R \\ u &= f \quad \text{on } \Gamma_D \end{aligned} \tag{8.1}$$

The variational formulation for (8.1) is: Find $u \in V := H^1_{\Gamma_D}(\Omega)$ such that

$$a(u,v) = F(v) \qquad \forall v \in V \tag{8.2}$$

where

$$a(u,v) = \int_\Omega (\operatorname{grad} u \cdot \operatorname{grad} \bar{v} - k^2 u\bar{v})dx - ik\int_{\Gamma_R} u\bar{v}dS \quad \forall u,v \in V \tag{8.3a}$$

$$F(v) = \int_{\Gamma_R} g\bar{v}dS \quad \forall v \in V \tag{8.3b}$$

We approximate (8.2) using the Galerkin method in a finite-element space $V^h \subset V$ on a triangular mesh $\mathcal{T}^h$ with mesh width h. The linear system is

$$A\vec{\boldsymbol{\mu}} = \vec{\boldsymbol{\varphi}} \tag{8.4}$$

Let $\{b_1, \cdots, b_n\}$ be the nodal basis of V^h, b_i is the tent function associated with vertex $\boldsymbol{x_i}$. Suppose there is a set of open subdomains $\{\Omega_l\}_{l=1}^N$, forming an overlapping cover of Ω, and each $\overline{\Omega_l}$ is a union of elements of the mesh $\mathcal{T}^h$. The discrete local version of (2.3) with impedance boundary condition is[5]:

$$\begin{aligned} -\Delta u - k^2 u &= 0 \quad \text{in } \Omega_l \\ \frac{\partial u}{\partial \boldsymbol{n}} - iku &= 0 \quad \text{on } \partial\Omega_l \backslash \Gamma_D \\ u &= f \quad \text{on } \partial\Omega_l \cap \Gamma_D \end{aligned} \tag{8.5}$$

The local impedance sesquilinear form on Ω_l is

$$a_l(u,v) = \int_{\Omega_l} (\operatorname{grad} u \cdot \operatorname{grad} \bar{v} - k^2 u\bar{v})dx - ik\int_{\partial\Omega_l \backslash \Gamma_D} u\bar{v}dS \tag{8.6}$$

For $u, v \in V_l^h = \{v_h|_{\overline{\Omega_l}} : v_h \in V^h\}$. And suppose $V_l^h = \operatorname{span}\{c_1^l, \cdots, c_{n_l}^l\}$, define a local to global index mapping G such that

$$c_i^l = b_j|_{\Omega_l}, \quad j = G(i,l)$$

Define the matrix $A_l = [a_l(c_j^l, c_i^l)]_{i,j=1}^{n_l}$, which is a local version of A in (8.4).

Local Smoothing

Let $\vec{v}$ be an approximation solution of systems,

$$A\vec{\mu} = \vec{\varphi}$$

the residual equation is

$$A\vec{e} = \vec{r}$$

where $\vec{r} = \vec{\varphi} - A\vec{v}$.

Let the subdomain Ω_l be the vertex patch associated with vertex $\boldsymbol{x_l}$, here the vertex patch is the two layers of triangles around $\boldsymbol{x_l}$. The local linear system for "vertex-patch subproblems":

$$A_l\vec{e}_l = \vec{r}_l \tag{8.7}$$

where $\vec{r}_l \in \mathbb{R}^{n_l}$ and

$$\vec{r}_l[i] = \begin{cases} \vec{r}[l] & \text{if } l = G(i,l) \\ 0 & \text{otherwise} \end{cases}$$

where $[i]$ denotes the ith component of the vector. Solve the problem (8.7) directly and then update:

$$\vec{v}[l] = \vec{v}[l] + \vec{e}_l[i], \text{where } l = G(i,l)$$

Implementation in LehrFEM++

Algorithm 4 gives the procedure of how to obtaining index of cells and vertices in a vertex patch, with these information, we can assemble local matrix A_l and construct the local to global mapping G.

Suppose $\tilde{A}_l = [a_l(b_j, b_i)]_{i,j=1}^n$, and $Q_l \in \mathbb{R}^{n_l,n}$ such that $Q_l[i,j] = 1$ if $G(i,l) = j$, then

$$A_l = Q_l\tilde{A}_lQ_l^\top$$

Algorithm 4: Get the index of vertices and cells in any vertex patch

Input: **mesh**;

Output: **adjacent_vertex**, **adjacent_cell** (**adjacent_vertex**$[i]$ should contain the index of vertices in vertex patch i, while **adjacent_cell**$[i]$ should contain the index of cells in vertex patch i);

for *cell in mesh* **do**
    i = mesh→Index(**cell**) ;
    for *vertex in* ***cell*** **do**
        p=mesh→Index(vertex);
        adjacent_cell$[p]$.insert(i);
        insert the index of three vertices of **cell** to **adjacent_vertex**$[p]$;
    end
end

Add elements of **adjacent_vertex**$[j]$ to **adjacent_vertex**$[i]$ if $j \in$**adjacent_vertex**$[i]$;

Add elements of **adjacent_cell**$[j]$ to **adjacent_cell**$[i]$ if $j \in$**adjacent_cell**$[i]$;

8.3 Numerical experiments

Algorithm 5 shows the new smoothing strategy, GMRES or local smoothing with impedance boundary condition is used on intermediate meshes, in practice we choose to perform G-S twice ($\mu_1 = \mu_2 = 2$). We use algorithm 5 as the action of the inverse of a preconditioner operator for the finest problem

$$A_L \vec{\boldsymbol{\mu}} = \vec{\boldsymbol{\varphi}}$$

We will run a series of numerical experiments to show the effectiveness of this method, in all tests the outer FGMRES iteration is run until

$$\|\vec{\boldsymbol{r}}_m\| / \|\vec{\boldsymbol{r}}_0\| < 10^{-6}$$

is satisfied, where $\vec{\boldsymbol{r}}_m = \vec{\boldsymbol{\varphi}} - A_L \vec{\boldsymbol{v}}_m$ is the residual of the mth outer FGMRES iterate and the norm is the vector Euclidean norm. In practice we use three different domains (unit square, unit square with square hole and unit square with triangle hole), g and f are chosen such that the true solution is plane wave $u(\boldsymbol{x}) = \exp(ik\boldsymbol{d} \cdot \boldsymbol{x})$ with frequency $\boldsymbol{d} = (0.8, 0.6)$, and table (8.1) and (8.2) shows the iterations counts with increasing wave number for the intermediate smoothing being GMRES and local smoothing with impedance boundary condition respectively.

Algorithm 5: $\vec{v}_l = MG(\vec{v}_l, \vec{\varphi}_l)$ (Multi-grid V-cycle with stable smoothing on intermediate meshes)

$l = 0$ indicates the coarsest mesh, while $l = L$ is the finest mesh.
if *l==0* **then**
  $\vec{v}_l = A_l^{-1}\vec{\varphi}_l$
else
  if $kh_l < 1.5$ *or* $kh_l > 8.0$ **then**
    perform ν_1 steps of G-S smoothing.
  else
    perform local smoothing with impedance boundary condition once or perform GMRES with 20 iterations.
  end
  $\vec{v}_l \leftarrow \vec{v}_l + I_{l-1}^{l} MG(0, I_l^{l-1}(\vec{\varphi}_l - A_l\vec{v}_l))$
  if $kh_l < 1.5$ *or* $kh_l > 8.0$ **then**
    perform ν_2 steps of G-S smoothing.
  else
    perform local smoothing with impedance boundary condition once or perform GMRES with 20 iterations.
  end
end

$k\backslash$domain	unit square	square with square hole	square with triangle hole
4	6	8	8
6	7	8	8
8	11	6	9
10	23	9	18
12	44	21	47
14	67	51	76
16	88	82	99
18	103	131	108
20	121	114	114

Table 8.1: Iteration counts for FGMRES until residual is reduced by a factor of 10^{-6} with different wave number and underlying computation domain, here the multi-grid preconditioner contains 5 meshes, the finest mesh has mesh width h_L=1/16, and GMRES is used on intermediate meshes.

$k\backslash$domain	unit square	square with square hole	square with triangle hole
4	7	8	8
6	10	9	10
8	14	10	13
10	18	14	19
12	24	25	28
14	32	32	33
16	40	37	39
18	51	43	49
20	60	49	59

Table 8.2: Iteration counts for FGMRES until residual is reduced by a factor of 10^{-6}. The mesh setting is same as table 8.1. Local smoothing with impedance boundary condition is used on intermediate meshes.

Chapter 9

Discussion

In this project we implement the PUM wave-ray method to solve the Helmholtz problem, we also implement the multigrid with GMRES and local impedance smoothing on coarse mesh and use it as preconditioner for outer GMRES iteration. Results in chapter 8 tell us that the characteristic components are quiet stubborn, use the GMRES and local impedance smoothing can only guarantee not to amplify the problematic errors, the ability of outer GMRES to handle problematic components degrades as the increase of the wave number. We can see from results in section 5.4 that (Extend) PUM space can approximate the Helmholtz problem very well, although the results for PUM wave-ray method are not satisfactory, the idea of approximating the characteristic components in PUM spaces worth exploring. Problems may occur when transferring the residual after wave cycle to PUM space and transferring the correction back to continue the wave cycle. Characteristic components are quite oscillatory, we may lose the precision when moving these components, while in wave-ray method, the smooth ray functions are separated and treated by ray cycle. It is worth noting that all mesh operator A_l excepting the one on finest wave mesh are constructed by Galerkin projection, using the operator assembling in PUM spaces cause divergence. For PUM spaces, if the mesh width is small and the number of plane waves is high, then the basis functions $\{b_i^l e_t^l\}$ are 'nearly' linear dependent, which make the stiffness matrix to be badly conditioned. In further research, accuracy properties regarding transfer operator between Lagrangian finite element space and PUM space, the choice of coarse mesh operator need to be studied.

Appendix A

Appendix

A.1 Implementation of GMRES

The implementation of GMRES in C++, the main routine *gmres* has five parameters, first two is the matrix and right hand side, the third is the initial guess, the forth denotes the maximum number of iterations, last one is aimed at terminate the procedure early if the residual is reduced at a factor of *threshold*. If predonditioned GMRES is required, then the main change would be at line 29, to change the way to create the new vector for the Arnoldi iteration.

```
1  using Scalar = std::complex<double>;
2  using Vec_t = Eigen::Matrix<Scalar, Eigen::Dynamic, 1>;
3  using Mat_t = Eigen::Matrix<Scalar, Eigen::Dynamic,
   ↪  Eigen::Dynamic>;
4
5  template <typename Mat_type>
6  void gmres(Mat_type& A, Vec_t& b, Vec_t& x, int max_iterations,
   ↪  double threshold) {
7      int n = A.rows();
8      int m = max_iterations;
9      Vec_t r = b - A * x; // initial residual
10     double r_norm = r.norm();
11     double error = r_norm / r_norm;
12     std::vector<double> e;
13     e.push_back(error);
14
15     // For givens roatation
16     Vec_t sn = Vec_t::Zero(m), cs = Vec_t::Zero(m);
17
```

```
18      // beta is initialized to r_norm*e1
19      Vec_t beta = Vec_t::Zero(m+1);
20      beta(0) = r_norm;
21
22      Mat_t V(n, m+1);  // to store the orthonormal bases
23      Mat_t H = Mat_t::Zero(m+1, m);
24      V.col(0) = r / r_norm;
25
26      int j;
27      for(j = 1; j <= m; ++j) {
28          // arnoldi process
29          Vec_t wj = A * V.col(j-1);
30          for(int i = 0; i < j; ++i) {
31              H(i, j-1) = wj.dot(V.col(i));
32              wj = wj - H(i, j-1) * V.col(i);
33          }
34          H(j, j-1) = wj.norm();
35          V.col(j) = wj / H(j, j-1);
36
37          // eliminate the last element in H jth row and update
            ↪  the rotation matrix
38          applay_givens_roataion(H, cs, sn, j-1);
39
40          // update the residual vector
41          beta(j)  = -sn(j-1) * beta(j-1);
42          beta(j-1) = cs(j-1) * beta(j-1);
43
44          error = std::abs(beta(j)) / r_norm;
45          e.push_back(error);
46
47          if(error <= threshold) {
48              break;
49          }
50      }
51      // calculate the result
52      if(j == m + 1) {
53          --j;
54      }
55      Vec_t y = solve_upper_triangle(H.topLeftCorner(j, j),
        ↪  beta.head(j));
56      x = x + V.leftCols(j) * y;
57  }
58
59  Vec_t solve_upper_triangle(const Mat_t& U, const Vec_t& b) {
```

```
60      int n = b.size();
61      Vec_t x = Vec_t(n);
62      for(int i = n - 1; i >= 0; --i) {
63          Scalar tmp = b(i);
64          for(int j = i + 1; j < n; ++j) {
65              tmp -= U(i,j) * x(j);
66          }
67          x(i) = tmp / U(i, i);
68      }
69      return x;
70  }
71
72  std::pair<Scalar, Scalar> givens_rotation(Scalar rho, Scalar
    ↪  sigma) {
73      if(rho == 0.0) {
74          return {0.0, 1.0};
75      } else {
76          double tmp = std::abs(rho * rho) + std::abs(sigma *
            ↪  sigma);
77          Scalar c = std::abs(rho) / (std::sqrt(tmp));
78          Scalar s = c * sigma / rho;
79          return {c, s};
80      }
81  }
82
83  void applay_givens_roataion(Mat_t& H, Vec_t& cs, Vec_t& sn, int
    ↪  j) {
84      for(int i = 0; i < j; ++i) {
85          Scalar tmp = cs(i) * H(i, j) + std::conj(sn(i)) * H(i+1,
            ↪  j);
86          H(i+1, j) = -sn(i) * H(i, j) + cs(i) * H(i+1, j);
87          H(i, j) = tmp;
88      }
89      auto cs_pair = givens_rotation(H(j, j), H(j+1, j));
90      cs(j) = cs_pair.first;
91      sn(j) = cs_pair.second;
92      H(j, j) = cs(j) * H(j,j) + std::conj(sn(j)) * H(j+1, j);
93      H(j+1, j) = 0.0;
94  }
```

A.2 Eigenvalues of Gauss-Seidel relaxation

For matrix $A = (1/h^2)\,\text{tridiag}(-1, 2, -1) - k^2\boldsymbol{I} \in \mathbb{R}^{N\times N}$, $h = 1/(N+1)$, we split A into $A = D - L - U$, where D is the matrix consisting of the diagonal of A, $-L$ and $-U$ are strict lower and upper triangle matrix of A. The Gauss-Seidel iteration matrix is $R_{GS} = (D-L)^{-1}U$, and its eigenvalues are

$$\lambda_j^{GS} = \frac{4}{(2-h^2k^2)^2}\cos^2(j\pi h), \quad j = 1, \cdots, N$$

Proof Suppose λ is an eigenvalue of R_{GS} and $\vec{\boldsymbol{w}} = [w_i]_{i=1}^N$ is the associated eigenvector, then we have

$$R_{GS}\vec{\boldsymbol{w}} = \lambda\vec{\boldsymbol{w}} \iff U\vec{\boldsymbol{w}} = \lambda(D-L)\vec{\boldsymbol{w}}$$

we have componentwise

$$\begin{cases} w_{s+1} + \lambda(2-h^2k^2)w_s + \lambda w_{s-1} = 0, & s = 1, \cdots, N \\ w_0 = w_{N+1} = 0 \end{cases} \tag{A.1}$$

To solve this difference equation, suppose w_s satisfies

$$w_s = t^s, \quad , s = 0, \cdots, N+1$$

for some $t \in \mathbb{C}$, substitute it into the first equation in (A.1), we get

$$t^2 + \lambda(2-h^2k^2)t + \lambda = 0$$

suppose this quadratic equation has two solution t_1 and t_2, according to Vieta's formulas

$$\begin{cases} t_1 + t_2 = -\lambda(2-h^2k^2) \\ t_1t_2 = \lambda \end{cases} \tag{A.2}$$

Due to the linearity, we can assume the solution of (A.1) is

$$w_k = p_1t_1^k + p_2t_2^k, \quad , i = 0, \cdots, N+1$$

where $p_1, p_2 \in \mathbb{C}$, for the boundary condition

$$\begin{cases} w_0 = p_1 + p_2 = 0 \\ w_{N+1} = p_1t_1^{N+1} + p_2t_2^{N+1} = 0 \end{cases} \Rightarrow \begin{cases} p := p1 = -p2 \\ (\frac{t_1}{t_2})^{N+1} = 1 \end{cases}$$

we can know that t_1/t_2 is the $(N+1)-$th root of 1, and we are not interested in the case $t_1 = t_2$, so we can write

$$\frac{t_{1,j}}{t_{2,j}} = e^{i\frac{2j\pi}{N+1}}, \quad j = 1, \cdots, N$$

together with the fact that $t_{1,j}t_{2,j} = \lambda_j$, we can get that

$$t_{1,j}^2 = \lambda_j e^{i\frac{2j\pi}{N+1}}, \quad t_{2,j}^2 = \lambda_j e^{-i\frac{2j\pi}{N+1}}$$

and

$$\begin{aligned}\lambda_j^2(2-h^2k^2)^2 &= t_{1,j}^2 + t_{2,j}^2 + 2t_{1,j}t_{2,j} \\ &= \lambda_j(e^{i\frac{2j\pi}{N+1}} + e^{-i\frac{2j\pi}{N+1}} + 2) \\ &= \lambda_j(2 + 2\cos(2j\pi h)) \\ &= 4\lambda_j \cos^2(j\pi h)\end{aligned}$$

Finally we get

$$\lambda_j = \frac{4}{(2-h^2k^2)^2}\cos^2(j\pi h).$$

Bibliography

[1] I. Babuška and J. M. Melenk. The partition of unity method. *International Journal for Numerical Methods in Engineering*, 40(4):727–758, 1997.

[2] A. Brandt and I. Livshits. Wave-ray multigrid method for standing wave equations. *Electronic Transactions on Numerical Analysis*, 6, 08 1998.

[3] William Briggs, Van Henson, and Steve McCormick. *A Multigrid Tutorial, 2nd Edition*. SIAM, 01 2000.

[4] Howard C. Elman, Oliver G. Ernst, and Dianne P. O'Leary. A multigrid method enhanced by krylov subspace iteration for discrete helmholtz equations. *SIAM Journal on Scientific Computing*, 23(4):1291–1315, 2001.

[5] Ivan G. Graham, Euan A. Spence, and Jun Zou. Domain decomposition with local impedance conditions for the helmholtz equation with absorption. *SIAM Journal on Numerical Analysis*, 58(5):2515–2543, 2020.

[6] Frank Ihlenburg. *Finite Element Analysis of Acoustic Scattering*. Springer, New York, NY, 1998.

[7] Volker John. Multigrid methods. https://www.wias-berlin.de/people/john/LEHRE/MULTIGRID/multigrid.pdf, 2013. [Online].

[8] D. Lahaye, J.M. Tang, and C. Vuik. *Modern Solvers for Helmholtz Problems*. Birkhäuser, 01 2017.

[9] B. Lee, T. A. Manteuffel, S. F. McCormick, and J. Ruge. Multilevel first-order system least squares(fosls) for helmholtz equations. *SIAM Journal on Scientific Computing*, 1999.

[10] B. Lee, T. A. Manteuffel, S. F. McCormick, and J. Ruge. First-order system least-squares for the helmholtz equation. *SIAM Journal on Scientific Computing*, 21(5):1927–1949, 2000.

[11] Irene Livshits and Achi Brandt. Accuracy properties of the wave-ray multigrid algorithm for helmholtz equations. *SIAM Journal on Scientific Computing*, 28(4):1228–1251, 2006.

[12] J. M. Melenk. On generalized finite element methods. *Ph.D. Thesis, University of Maryland*, 1995.

[13] Christoph Pflaum. Multigrid methods. https://www.cs10.tf.fau.de/files/2018/06/pflaum-script-mg.pdf, 2005. [Online].

[14] Yousef Saad. *Iterative Methods for Sparse Linear Systems*. Society for Industrial and Applied Mathematics, second edition, 2003.

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of originality

The signed declaration of originality is a component of every semester paper, Bachelor's thesis, Master's thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

Title of work (in block letters):

Authored by (in block letters):
For papers written by groups the names of all authors are required.

Name(s):	**First name(s):**

With my signature I confirm that

- I have committed none of the forms of plagiarism described in the 'Citation etiquette' information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

Place, date	**Signature(s)**
Zürich, 15. 04. 2021	Liaowang Huang

For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.