Lagrangian approach for solution of time-harmonic acoustic coefficient inverse problem

1 Introduction

The goal of the project is solution of the coefficient inverse problem (CIP) for determination of wave speed function in the time-harmonic acoustic wave equation from scattered wave field at the boundary of the computational domain. To solve this problem we will study the Lagrangian method and formulate conjugate gradient algorithm for iterative update of the wave speed function. The Lagrangian method is similar to the one applied in [5, 8, 9, 11] for the solution of different hyperbolic CIPs. Practical applications of a such CIP are huge, and we will mention some of them such as subsurface imaging, nondestructive testing of materials and detection of landmines [8, 9, 20], archaeology, construction of photonic crystals [18] and cloaking materials [19], remote sensing and medical imaging [24].

It is typical that in industrial applications computational domains often are very large with constant values of material parameters. Usually, only some part of these domains where materials change presents interest. In such cases the problem is described by the model equations with the constant material parameters in a boundary neighborhood of the computational domain. In this project we consider a particular case when the wave speed function has a constant value in a boundary neighborhood.

The description of the course project is organized as follows. In section 2 are formulated the forward and inverse problems. In section 3 is presented the Tikhonov functional to be minimized and formulated Lagrangian approach to solve the inverse problem. In section 4 is formulated the conjugate gradient algorithm to solve our CIP. In section 6 are described data and how to use them to solve time-harmonic CIP of this project.

Notes

- You can work in groups by 2-5 persons.
- To pass this course you should write the report for the project and present results of the project to examiner which will be your oral examination. Remote presentation of the results of this project is also possible.

- Sent final report (tex and pdf files) with description of your work together with Matlab or C++/PETSc programs for testing to my e-mail

  `larisa@chalmers.se`

  before **19 June 2020**. Report should have description of used techniques, tables and figures confirming your investigations. Analysis of obtained results is necessary to present in section "Numerical examples" and summarized results - in section "Conclusions".

- Matlab and C++/PETSc programs for solution of the Poisson equation (see example in section 8.1.3 of the book [2]) can be used as templates for the problem of this project. These programs are available for free download: go to the link of the book [2] and click *"GitHub Page with MATLAB® Source Codes"* on the bottom of this page.

- Data for solution of CIP of this project is available for download from the links

  http://www.math.chalmers.se/ larisa/www/IPcourse2019/OBSref5noise3h03125gaus1.zip

  http://www.math.chalmers.se/ larisa/www/IPcourse2019/OBSref5noise3h03125gaus2.zip

  http://www.math.chalmers.se/ larisa/www/IPcourse2019/OBSref5noise10h03125gaus1.zip

  Description of data is given in section 6.

2 The mathematical model

Our basic model is given in terms of the function $u(x,s), x \in \mathbb{R}^d, d = 2,3$ which depends on the pseudo-frequency $s > const. > 0$, see [6] for details:

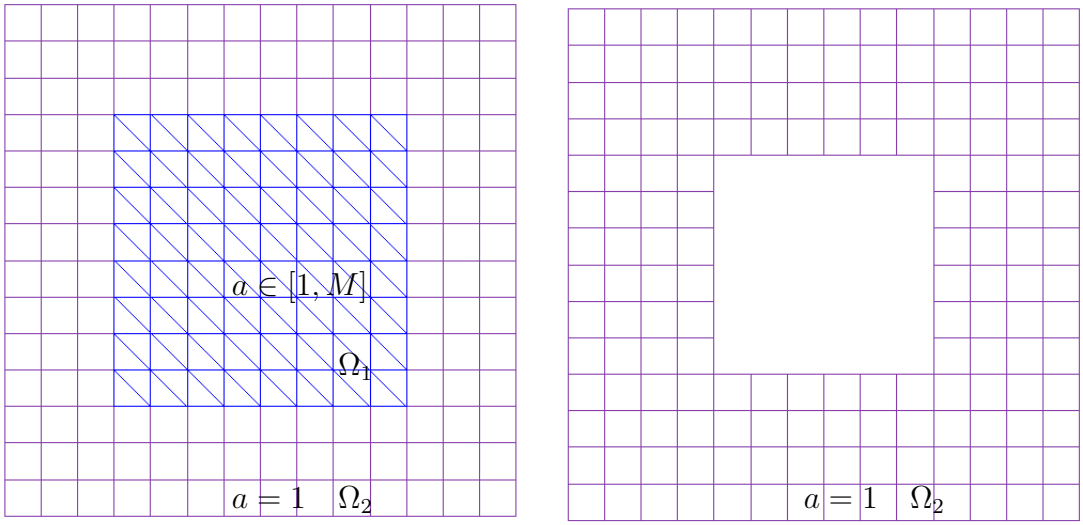
$$\triangle u(x) - s^2 a(x) u(x) = -s a(x) f_0(x), \quad x \in \mathbb{R}^d, d = 2,3, \tag{2.1}$$

Here, $a(x) = 1/\sqrt{c(x)}$, where $c(x)$ is the sound speed.

To solve the problem (2.1) numerically in $\mathbb{R}^d, d = 2,3$ we will use the domain decomposition finite element/finite difference method of [5]. We introduce a convex bounded domain $\Omega \subset \mathbb{R}^2$ with boundary Γ such that $\Omega_2 := \Omega \setminus \Omega_1$, where $\Omega_1 \subset \Omega$, $\partial\Omega \cap \partial\Omega_1 = \emptyset$ with $\partial\Omega_2 = \partial\Omega \cup \partial\Omega_1$, $\Omega = \Omega_1 \cup \Omega_2$, $\Omega_1 = \Omega \setminus \Omega_2$ and $\bar{\Omega}_1 \cap \bar{\Omega}_2 = \partial\Omega_1$, where $\partial\Omega, \partial\Omega_1, \partial\Omega_2$ are boundaries of the domains $\Omega, \Omega_1, \Omega_2$, respectively. To introduce boundary conditions on $\Gamma := \partial\Omega$ we denote $\Gamma = \Gamma_1 \cup \Gamma_2 \cup \Gamma_3$ such that Γ_1 and Γ_2 are the top and bottom sides of the domain Ω, respectively, and Γ_3 denotes the rest of the boundary, see Figure 1.

The model equation (2.1) can be obtained by applying the Laplace transform in time,

$$u(x,s) := \int_0^{+\infty} u(x,t) e^{-st} dt, \qquad s = const. > 0 \tag{2.2}$$



a) $\Omega = \Omega_1 \cup \Omega_2$ b) Ω_2

Figure 1: *Domain decomposition in* Ω.

to the function $U(x,t)$ satisfying the time-dependent acoustic wave equation

$$\begin{aligned} a(x)\frac{\partial^2 U(x,t)}{\partial t^2} - \triangle U(x,t) = 0, \;\; x \in \mathbb{R}^d, d = 2,3, \;\; t \in (0,T]. \\ U(x,0) = f_0(x), \quad \frac{\partial U}{\partial t}(x,0) = 0, \;\; x \in \mathbb{R}^d, \quad d = 2,3. \end{aligned} \tag{2.3}$$

After applying the Laplace transform for the problem (2.3) and to the absorbing and Neumann boundary conditions

$$\begin{aligned} \partial_\nu u + \partial_t u = 0, \qquad (x,t) \in (\Gamma_1 \cup \Gamma_2) \times (0,T], \\ \partial_\nu u = 0, \qquad (x,t) \in \Gamma_3 \times (0,T]. \end{aligned} \tag{2.4}$$

we get the following model problem

$$\begin{aligned} \triangle u(x) - s^2 a(x) u(x) = -s a(x) f_0(x), \;\; x \in \mathbb{R}^d, d = 2,3, \\ \partial_\nu u(x) = 0, \;\; x \in \Gamma_3, \\ \partial_\nu u(x) = f_0(x) - s u(x,s), \;\; x \in \Gamma_1 \cup \Gamma_2. \end{aligned} \tag{2.5}$$

Here, $\partial_\nu(\cdot)$ denotes the normal derivative on Γ, where ν is the outward unit normal vector on the boundary Γ.

For further analysi we will assume that for some known constant $M > 1$ the wave speed function $a(x)$ is such that

$$\begin{aligned} a(x) &\in [1, M], \text{ for } x \in \Omega_1, \\ a(x) &= 1, \text{ for } x \in \Omega \setminus \Omega_1. \end{aligned} \tag{2.6}$$

and assume that

$$f_0 \in H^1(\Omega), \ \ c(x) \in C^2(\mathbb{R}^d), d = 2, 3. \tag{2.7}$$

Let us introduce the following space of real valued functions

$$U^1 = H^1(\Omega) \times H^1(\Omega) \times C\left(\overline{\Omega}\right). \tag{2.8}$$

We consider the following inverse problem

Inverse Problem (IP)

Let the coefficient $a(x)$ in the problem (2.5) satisfies conditions (2.6) and assume that $a(x)$ is unknown in the domain Ω_1. Determine the function $a(x)$ in (2.5) for $x \in \Omega_1$ assuming that the following function $\widetilde{u}(x)$ is known

$$u(x) = \widetilde{u}(x) \ \forall x \in \Gamma. \tag{2.9}$$

3 Lagrangian approach for solution IP

In this section we present the reconstruction method to solve inverse problem **IP**. This method is based on the finding of the stationary point of the following Tikhonov functional

$$F(u, c) = \frac{1}{2} \int_{\Gamma} (u - \widetilde{u})^2 z_\delta(x) dS + \frac{1}{2} \gamma \int_{\Omega} (a - a_0)^2 \ dx, \tag{3.1}$$

where u satisfies the equations (2.5), a_0 is the initial guess for a (we refer to [6, 8, 9] about details how to choose an initial guess for problem (2.5)), $\widetilde{u}$ is the observed field at Γ, $\gamma > 0$ is the regularization parameter and $z_\delta(x)$ is the compatibility function which can be defined similarly with [11].

To find minimum of (3.1) we apply the Lagrangian approach (see details in [5, 3, 8, 9, 11]) and define the following Lagrangian in the week form using definition of the forward model problem (2.5)

$$L(v) = F(u, a) + (\lambda, f_0 - su)_{\Gamma_1 \cup \Gamma_2} - (\nabla u, \nabla \lambda)_\Omega - (\lambda, s^2 a u)_\Omega + (\lambda, s a f_0)_\Omega, \tag{3.2}$$

where $(\cdot, \cdot)$ denote the standard scalar product in $L^2(\Omega)^d, d = 2, 3$, and $v = (u, \lambda, a) \in U^1$. We now search for a stationary point of the Lagrangian with respect to v satisfying for all $\bar{v} = (\bar{u}, \bar{\lambda}, \bar{\bar{a}}) \in U^1$

$$L'(v; \bar{v}) = 0, \tag{3.3}$$

where $L'(v;\cdot)$ is the Jacobian of L at v.

In order to find the Fréchet derivative (3.3) of the Lagrangian (3.2) we consider $L(v+\bar{v})-L(v)\ \forall \bar{v}\in U^1$ and single out the linear part of the obtained expression with respect to $\bar{v}$ ignoring all nonlinear terms. In the derivation of the Fréchet derivative we assume that in the Lagrangian (3.2) functions in $v=(u,\lambda,a)\in U^1$ can be varied independent on each others and thus the Fréchet derivative of the Lagrangian (3.2) will be the same as by assuming that functions u and λ are dependent on the function a, see details in Chapter 4 of [6]. The optimality condition (3.3) for the Lagrangian (3.2) for all $\bar{v}\in U^1$ is

$$L'(v;\bar{v})=\frac{\partial L}{\partial \lambda}(v)(\bar{\lambda})+\frac{\partial L}{\partial u}(v)(\bar{u})+\frac{\partial L}{\partial a}(v)(\bar{a})=0. \tag{3.4}$$

Thus, to satisfy optimum condition $L'(v;\bar{v})=0$ every component of (3.4) should be zero out. Using integration by parts together with boundary conditions in (2.5) we get

$$\begin{aligned} 0=\frac{\partial L}{\partial \lambda}(v)(\bar{\lambda})&=\int_\Omega(\triangle u-s^2au+saf_0)\bar{\lambda}\,dx\\ &+\int_{\Gamma_1\cup\Gamma_2}(f_0-su)\bar{\lambda}\,dS-\int_\Gamma \partial_n u\bar{\lambda}\,dS,\ \forall\bar{\lambda}\in H^1(\Omega), \end{aligned} \tag{3.5}$$

$$\begin{aligned} 0=\frac{\partial L}{\partial u}(v)(\bar{u})&=\int_\Gamma(u-\widetilde{u})\,\bar{u}\,z_\delta\,dS+\int_\Omega(\triangle\lambda-s^2a\lambda)\bar{u}\,dx\\ &-\int_{\Gamma_1\cup\Gamma_2}s\lambda\bar{u}\,dS-\int_\Gamma\partial_n\lambda\bar{u}\,dS,\ \forall\bar{u}\in H^1(\Omega), \end{aligned} \tag{3.6}$$

$$0=\frac{\partial L}{\partial a}(v)(\bar{a})=\int_\Omega(-s^2\lambda u+s\lambda f_0)\,\bar{a}\,dx+\gamma\int_{\Omega_1}(a-a_0)\,\bar{a}\,dx,\ \forall\bar{a}\in C\left(\overline{\Omega}\right). \tag{3.7}$$

It is clear that (3.5) corresponds to the state equation (2.5) and (3.6) corresponds to the following adjoint problem

$$\begin{aligned} \triangle\lambda-s^2a\lambda&=-(u-\widetilde{u})z_\delta,\ \ x\in\Omega,\\ \partial_n\lambda&=0,\ \ x \text{ on } \Gamma_3,\\ \partial_n\lambda&=-\lambda s,\ \ x \text{ on } \Gamma_1\cup\Gamma_2. \end{aligned} \tag{3.8}$$

Let us define by $u(a),\lambda(a)$ exact solutions of the forward and adjoint problems, respectively, for the known wave speed function a. Then using the fact that exact solutions $u(a),\lambda(a)$ are sufficiently stable (see Chapter 5 of book [21] for details), we get from (3.2)

$$F(u(a),a)=L(v(a)), \tag{3.9}$$

and the Fréchet derivative of the Tikhonov functional can be written as

$$F'(a):=F'(u(a),a)=\frac{\partial F}{\partial a}(u(a),a)=\frac{\partial L}{\partial a}(v(a)). \tag{3.10}$$

Inserting (3.7) into (3.10), we get the expression for the gradient with respect to the wave speed function which we will use for updating this function in the conjugate gradient method

$$F'(a)(x) := F'(u(a), a)(x) = -s^2\lambda u + s\lambda f_0 + \gamma(a - a_0). \tag{3.11}$$

4 Conjugate gradient algorithm

Recall that we denote the standard inner product in $[L^2(\Omega)]^d$ as $(\cdot, \cdot)$, $d \in \{1, 2, 3\}$, and the corresponding norm by $\| \cdot \|$. To compute minimum of the functional (3.1) we use the conjugate gradient method (CGM). Let us define the gradient with respect to the wave speed function at the iteration m in CGM as

$$g^m(x) = -s^2\lambda_h^m u_h^m + s\lambda^m f_0 + \gamma^m(a_h^m - a_0), \tag{4.1}$$

where a_h^m is approximation of the function a_h on the iteration step m in GCM, $u_h(x, a_h^m), \lambda_h(x, a_h^m)$ are computed by solving the state problem (2.5) and the adjoint problem (3.8), respectively, with $a := a_h^m$, γ^m is iteratively computed regularization parameter via rules of [1].

The usual gradient method (GM) is the special case of the conjugate gradient method (CGM) such that functions a^m are computed as

$$a^{m+1}(x) = a^m(x) + \alpha^m d^m(x), \tag{4.2}$$

where α^m are iteratively updated parameter and d^m is the direction which is computed for usual gradient method as

$$d^m = -g^m \tag{4.3}$$

and for the conjugate gradient method as

$$d^m = -g^m + \beta_m d_{m-1}, \tag{4.4}$$

at the iteration m where parameters β_m are computed as in (4.10).

Step-size in the gradient update α^m is computed such that it gives minimum to the Lagrangian $L(u_h^m, \lambda_h^m, a_h^m + \alpha^m d^m)$, or such that $L'_{\alpha^m}(u_h^m, \lambda_h^m, a_h^m + \alpha^m d^m) = 0$. More precisely, using definition of the Lagrangian (3.2) we have

$$\begin{aligned} L(u_h^m, \lambda_h^m, a_h^m + \alpha^m d^m) = F(u_h^m, a_h^m + \alpha^m d^m) \; + \\ \int_\Omega \lambda_h^m[-s^2(a_h^m + \alpha^m d^m)u_h^m + s(a_h^m + \alpha^m d^m)f_0]\, dx \\ - \int_\Omega \nabla\lambda_h^m \nabla u_h^m \, dx + \int_{\Gamma_1 \cup \Gamma_2} (f_0 - s u_h^m)\lambda_h^m \, dS, \end{aligned} \tag{4.5}$$

and

$$L'_{\alpha^m}(u_h^m, \lambda_h^m, a_h^m + \alpha^m d^m) = \gamma^m \int_\Omega (a_h^m + \alpha^m d^m - a_0) d^m \, dx + \int_\Omega \lambda_h^m (-s^2 d^m u_h^m + s d^m f_0) \, dx = 0. \tag{4.6}$$

We can obtain directly from (4.6)

$$\alpha^m = \frac{(\lambda_h^m (s^2 u_h^m - s f_0) - \gamma^m a_h^m + \gamma^m a_0, d^m)}{\gamma^m (d^m, d^m)} \tag{4.7}$$

or using definition (4.1) for g^m we obtain following expression for computation of the iterative parameter α^m:

$$\alpha^m = -\frac{(g^m, d^m)}{\gamma^m (d^m, d^m)}. \tag{4.8}$$

Algorithm (CGM)

- Step 0. Choose the computational space mesh K_h in Ω. Start with the initial approximation $a_h^0 = a_0$ at K_h and compute the sequences of a_h^m via the following steps:
- Step 1. Compute solutions $u_h(x, a_h^m)$ and $\lambda_h(x, a_h^m)$ of state (2.5) and adjoint (3.8) problems, respectively, on K_h.
- Step 2. Update the coefficient $a_h := a_h^{m+1}$ on K_h (only inside the discretized domain Ω_1) using

$$a_h^{m+1} = a_h^m + \alpha^m d^m(x), \tag{4.9}$$

where

$$d^m(x) = -g^m(x) + \beta^m d^{m-1}(x),$$

with

$$\beta^m = \frac{\|g^m(x)\|^2}{\|g^{m-1}(x)\|^2}, \tag{4.10}$$

where $d^0(x) = -g^0(x)$. In (4.9) the step size α^m in the gradient update is computed as

$$\alpha^m = -\frac{(g^m, d^m)}{\gamma^m \|d^m\|^2}, \tag{4.11}$$

and the regularization parameter γ^m at iteration m is computed iteratively accordingly to [1] as

$$\gamma^m = \frac{\gamma_0}{(m+1)^p}, \quad p \in (0,1). \tag{4.12}$$

- Step 3. Stop computing a_h^m and obtain the function a_h at $M = m$ if either $\|g^m\|_{L_2(\Omega)} \leq \theta$ or norms $\|g^m\|_{L_2(\Omega)}$ are stabilized. Here θ is the tolerance in updates m of gradient method. Otherwise set $m := m + 1$ and go to step 1.

5 Instructions for solution of CIP

In the project report you should present numerical simulations for reconstruction of the function $a(x)$ in (2.5) via CGM algorithm of section 4 in the domain $\Omega = [0,1] \times [0,1]$ using data generated at the boundary Γ of Ω described in section 6. Alternatively, you can generate your own data as it is described below, and solve **IP** via CGM.

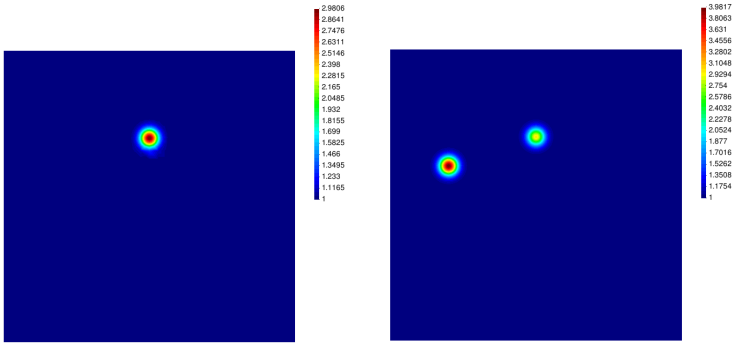


Figure 2: Functions $a(x)$ in Ω_1 used in Test 1 (left figure) and Test 2 (right figure).

1. Discretize the computational domain $\Omega = [0,1] \times [0,1]$ and construct mesh K_{data_h}. To avoid variational crime the mesh K_{data_h} should not coincide with the mesh K_h used for solution **IP** in the algorithm CGM.

2. Compute solution $u_h(x,a)$ of state (2.5) problem with $f_0 = 0$ and known function $a(x)$ for Tests 1 and 2, see section 5.2 for definition of this function. Save discrete values of the observed solution $u_h(x) = \tilde{u_h}$ at Γ.

3. You can also generate data for the case when $f_0 \neq 0$ in (2.5). For example, take $f_0 = e^{-}x_1^2 + x_2^2$ at the backscattered boundary of the domain Ω.

4. When solving **IP** it is assumed that the exact values of the function $a(x)$ are not known and algorithm CGM should be runned with some initial guess for $a = a_0$. Usually, we take $a_0 = 1$ in all points of the computational domain Ω since previous computational works [5, 8, 9] as well as experimental works of [22, 23] have shown that a such choice gives good results of reconstruction.

5. Add additive or random noise δ to data $\tilde{u}$ using the formula

$$\tilde{u}_\sigma = \tilde{u}(1+\delta),$$

for additive noise and

$$\tilde{u}_\sigma = \tilde{u}(1+\delta\alpha),$$

for random noise. Here, $\alpha \in (-1,1)$ is randomly distributed number, $\delta \in [0,1]$ is the noise level. For example, if noise in data is 5%, then $\delta = 0.05$.

6. Solve **IP** via CGM using data $\tilde{u}$ with different functions a to be reconstructed defined in Test 1 and Test 2. Analyze obtained results by computing the relative error e in

the obtained reconstructions of a depending on the different noise level $\delta \in [0,1]$ in data $\tilde{u}$.

The relative error e compute as

$$e = \frac{\|a_h - a^*\|}{\|a^*\|}, \tag{5.1}$$

Here, a^* is the exact value given by (5.5) for Test 1 and by (5.6) for Test 2, correspondingly, and a_h - computed one. Present results how relative errors (5.1) depend on the random noise $\delta \in [0,1]$ in graphical form and in the corresponding table.

7. Optional: discretize the computational domain Ω into elements with different sizes $h_l = 2^{-l}, l = 1, ..., 6$. and present results of computations in graphical form and in the corresponding table. More precisely, present how relative error (5.1) depends on the number of discretization points N and mesh size $h_l = 2^{-l}, l = 1, ..., 6$ in Ω. Compute convergence rates as

$$q = \frac{\log\left(\frac{e_h}{e_{2h}}\right)}{\log(0.5)}, \tag{5.2}$$

where e_h, e_{2h} are computed relative norms e on the mesh $\mathcal{K}_h$ with the mesh size h and $2h$, respectively.

Results can be presented as in the following example:

l	nel	nno	e	q
1	8	9	$2.71 \cdot 10^{-2}$	
2	32	25	$6.66 \cdot 10^{-3}$	2.02
3	128	81	$1.78 \cdot 10^{-3}$	1.90
4	512	289	$4.13 \cdot 10^{-4}$	2.11
5	2048	1089	$1.05 \cdot 10^{-4}$	1.97
6	8192	4225	$2.65 \cdot 10^{-5}$	1.99

Table 1: Example of computing relative errors in the L_2-norm for mesh sizes $h_l = 2^{-l}, l = 1, ..., 6$.

5.1 Description of the process of data generation of section 6

We set the dimensionless computational domain $\Omega = [-0.5, 1.5] \times [-0.5, 1.5]$ such that it is divided into two subdomains Ω_2 and $\Omega_1 = [0,1] \times [0,1]$ where $\Omega = \Omega_1 \cup \Omega_2$ with two layers of structured overlapping nodes between these domains, see Figure 1 and Figure 2

of [5] for details about communication between two domains. The mesh size h in $\Omega = \Omega_1 \cup \Omega_2$ is $h = 0.03125$. In computations time-dependent observations are collected at $(\Gamma_1 \cup \Gamma_2) \times (0, T)$ at the backscattering Γ_1 and transmitted Γ_2 sides of Ω, respectively. We define $\Gamma_{1,1} := \Gamma_1 \times (0, t_1]$, $\Gamma_{1,2} := \Gamma_1 \times (t_1, T)$.

For generation of data at the boundary Γ of the domain $\Omega = [0, 1] \times [0, 1]$ was solved the time-dependent problem (2.3) with absorbing boundary conditions (2.4) in time $T = [0, 2.0]$ with the time step $\tau = 0.002$ which satisfies to the CFL condition [7] using the domain decomposition method of [5]. More precisely, the model problem for time-dependent wave equation is

$$\begin{aligned} a(x)\partial_t^2 U(x,t) - \triangle U(x,t) &= 0 \text{ in } \Omega \times (0,T), \\ U(x,0) = 0, \quad U_t(x,0) &= 0 \text{ in } \Omega, \\ \partial_n U &= f(t) \text{ on } \Gamma_{1,1}, \\ \partial_n U &= -\partial_t U \text{ on } \Gamma_{1,2} \cup \Gamma_2, \\ \partial_n U &= 0 \text{ on } \Gamma_3. \end{aligned} \tag{5.3}$$

In (5.3) the function $f(t)$ is defined as

$$f(t) = \begin{cases} \sin(\omega_s t), & \text{if } t \in \left(0, \frac{2\pi}{\omega_s}\right), \\ 0, & \text{if } t > \frac{2\pi}{\omega_s}. \end{cases} \tag{5.4}$$

and represents the single direction of a plane wave initialized at Γ_1 in time $t = [0, 2.0]$. In all computations we take $\omega_s = 80$.

We assume that $a(x) = 1$ is known inside Ω_2 and satisfy conditions (2.6). The wave speed function was chosen as (5.5) in Test 1 and as (5.6) in Test 2. Then the Laplace transform (2.2) was applied to the computed time-dependent solution $u(x, t)$ of the problem (5.3) at the pseudo-frequency interval for $s \in [2.0, 7.0]$, then additive noise σ was added to the obtained solution.

5.2 Test 1

In this test should be reconstructed one smooth function given by

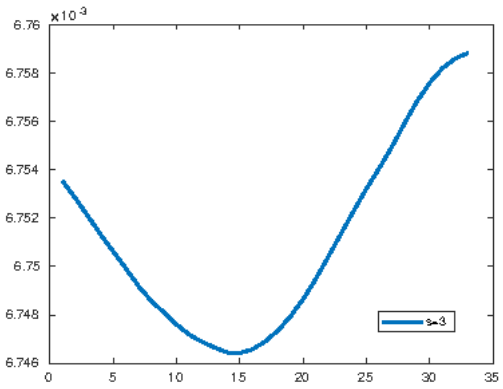
$$a(x) = 1.0 + 2.0 \cdot \mathrm{e}^{-((x_1-0.5)^2+(x_2-0.7)^2)/0.001} \tag{5.5}$$

which is presented in Figure 2- a).

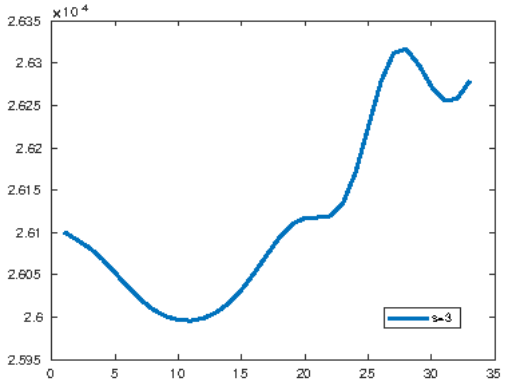
5.3 Test 2

In this test should be reconstructed two smooth functions given by

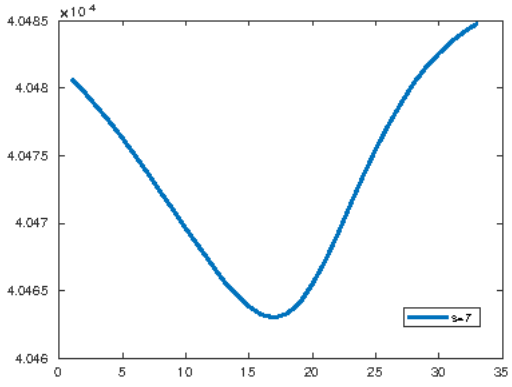
$$a(x) = 1.0 + 2.0 \cdot \mathrm{e}^{-((x_1-0.5)^2+(x_2-0.7)^2)/0.001} + 3.0 \cdot \mathrm{e}^{-((x_1-0.2)^2+(x_2-0.6)^2)/0.001}. \tag{5.6}$$



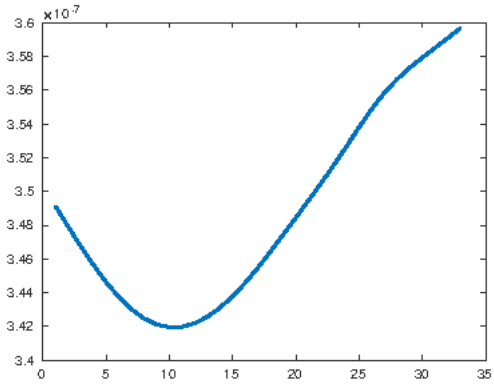
$\tilde{u}(x,3)$ at Γ_1



$\tilde{u}(x,3)$ at Γ_2



$\tilde{u}(x,7)$ at Γ_1



$\tilde{u}(x,7)$ at Γ_2

Figure 3: Test 1. Laplace transform of data $\tilde{U}(x,t)$ with 3% noise at the backscattered boundary Γ_1 and transmitted boundary Γ_2 for different values of parameter s in (2.2).

see Figure 2- b) for this function.

6 Description of data

Testing data are available for download from the following links:

- Observation data for one Gaussian function given by (5.5) and with 3% additive noise:

  http://www.math.chalmers.se/ larisa/www/IPcourse2019/OBSref5noise3h03125gaus1.zip

- Observation data for one Gaussian function given by (5.6) and with 3% additive noise:

  http://www.math.chalmers.se/ larisa/www/IPcourse2019/OBSref5noise3h03125gaus2.zip

- Observation data for one Gaussian function given by (5.5) and with 10% additive noise:

  http://www.math.chalmers.se/ larisa/www/IPcourse2019/OBSref5noise10h03125gaus1.zip

- Observation data for one Gaussian function given by (5.6) and with 10% additive noise:

  http://www.math.chalmers.se/ larisa/www/IPcourse2019/OBSref5noise10h03125gaus2.zip

In every directory

```
OBSref5noise*h03125gaus*
```

will be following files:

- Time-dependent solution of the problem (5.3) is computed in time $[0, 2]$ and can be visualized in GID using files

  ```
  ex_plane2Dcommon.res
  ex_plane2DFDM.res
  ex_plane2DFEM.res
  ```

  or in PARAVIEW using files

  ```
  outInn*.inp
  ```

(only FEM solution).

- Laplace transform of the time-dependent solution on the pseudo-frequency interval for $s \in [2.0, 7.0]$ can be visualized in GID using the file

  ```
  Laplace.res
  ```

- Fourier transform (real, imaginary and absolute values) can be visualized on the frequency interval for $\omega \in [30, 80]$ in GID using the file

  ```
  Fourier.res
  ```

  Files

  ```
  AbsFourierTr*.inp
  ```

  contain only absolute value of the Fourier transform and can be visualized in PARAVIEW.

- Files

  ```
  L_top*.m
  L_bot*.m
  L_left*.m
  L_right*.m
  ```

  contain data $\tilde{u}(x, s)$ after Laplace transform (2.2) of data $\tilde{U}(x, t)$ at the pseudo-frequency interval for $s \in [2.0, 7.0]$ in time $[0, 2]$ at the top, bottom, left and right boundaries, respectively, of Ω_1. One example of visualization of this data in Matlab is given in Figure 3 for top Γ_1 and bottom Γ_2 boundaries of Ω. To visualize data in Matlab,

  ```
  >load L_top*.m
  >plot(L_top*)
  ```

- Files

  ```
  Freal_top*.m
  Freal_bot*.m
  Freal_left*.m
  Freal_right*.m
  ```

contain real part of data $\tilde{u}(x,\omega)$ after Fourier transform of data $\tilde{U}(x,t)$ at the top, bottom, left and right boundaries, respectively, of Ω_1 on the frequency interval for $\omega \in [30,80]$ in time $[0,2]$.

- Files

```
Fimag_top*.m
Fimag_bot*.m,
Fimag_left*.m
Fimag_right*.m
```

contain imaginary part of data $\tilde{u}(x,\omega)$ after Fourier transform of data $\tilde{U}(x,t)$ at the top, bottom, left and right boundaries, respectively, of Ω_1 on the frequency interval for $\omega \in [30,80]$ in time $[0,2]$.

- Files

```
FourierTrReal*.m
FourierTrImag*.m
```

contain Fourier transform of the time-dependent noisy solution $U(x,t)$ on the frequency interval for $\omega \in [30,80]$ in the whole domain Ω_1.

7 Solution of the test problem using C++/PETSc

In this section we illustrate how C++/PETSc solver can be used for solution of the following Dirichlet problem for Helmholtz equation in two dimensions:

$$\begin{aligned} \triangle u(x) + \omega^2 \varepsilon(x) u = f(x) \ \text{ in } \ \Omega, \\ u = 0 \text{ on } \partial\Omega. \end{aligned} \tag{7.1}$$

Here $f(x)$ is a given function, $u(x)$ is the unknown function to be computed, and the domain Ω is the unit square $\Omega = \{(x_1,x_2) \in (0,1)\times(0,1)\}$.

The exact solution of (7.1) with the right hand side

$$\begin{aligned} f(x_1,x_2) = -(8\pi^2)\sin(2\pi x_1)\sin(2\pi x_2) - 2ix_1(1-x_1) - 2ix_2(1-x_2) \\ + \omega^2\varepsilon(x)(\sin(2\pi x_1)\sin(2\pi x_2) + ix_1(1-x_1)x_2(1-x_2)) \end{aligned} \tag{7.2}$$

is the function

$$u(x_1,x_2) = \sin(2\pi x_1)\sin(2\pi x_2) + ix_1(1-x_1)x_2(1-x_2). \tag{7.3}$$

To solve numerically (7.1) we first discretize the domain Ω with $x_{1i} = ih_1$ and $x_{2j} = jh_2$, where $h_1 = 1/(n_i - 1)$ and $h_2 = 1/(n_j - 1)$ are the mesh sizes in the directions x_1, x_2, respectively, n_i and n_j are the numbers of discretization points in the directions x_1, x_2, respectively. Usually, in computations we have the same mesh size $h = h_1 = h_2$. In this example we choose $n_i = n_j = n$ with $n = N + 2$, where N is the number of inner nodes in the directions x_1, x_2, respectively.

Indices (i, j) are such that $0 \leq i, j < n$ and are associated with every global node n_{glob} of the finite difference mesh. Global nodes numbers n_{glob} in two-dimensional case can be computed using the following formula:

$$n_{glob} = j + n_i i. \tag{7.4}$$

We use the standard finite difference discretization of the Laplace operator Δu in two dimensions and obtain discrete laplacian $\Delta u_{i,j}$:

$$\Delta u_{i,j} = \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{h^2} + \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{h^2}, \tag{7.5}$$

where $u_{i,j}$ is the solution at the discrete point (i, j). Using (7.5), we obtain the following scheme for solving problem (7.1):

$$\left(\frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{h^2} + \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{h^2} \right) + \omega^2 \cdot \varepsilon_{i,j} \cdot u_{i,j} = f_{i,j}, \tag{7.6}$$

where $\varepsilon_{i,j}, f_{i,j}$ are the values of the functions ε, f at the discrete point (i, j), correspondingly. Next, (7.6) can be rewritten as

$$u_{i+1,j} - 4u_{i,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1} + h^2 \cdot \omega^2 \cdot \varepsilon_{i,j} \cdot u_{i,j} = h^2 f_{i,j}. \tag{7.7}$$

which forms the system of linear equations

$$Au = b, \tag{7.8}$$

where the vector b has the components $b_{i,j} = h^2 f_{i,j}$, the elements of the matrix A are given by

$$A = \left(\begin{array}{c|c|c} A_N & I_N & \\ \hline I_N & \ddots & \ddots \\ \hline & \ddots & \ddots \\ \hline & & I_N \\ \hline & I_N & A_N \end{array} \right) + h^2 \cdot \omega^2 \cdot \varepsilon_{i,j} \cdot I$$

with identity matrix I of order N^2 and blocks A_N of order N such that

$$A_N = \begin{pmatrix} -4 & 1 & 0 & 0 & \cdots & 0 \\ 1 & -4 & 1 & 0 & \cdots & 0 \\ 0 & 1 & -4 & 0 & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & \cdots & \cdots & 0 & 1 & -4 \end{pmatrix},$$

which are located on the diagonal of the matrix A, and blocks with the identity matrices I_N of order N on its off-diagonals.

8 Description of C++/PETSc solver

In this section is demonstrated how PETSc [25] can be used for the solution of the problem (7.1). We set the computational domain to be the unit square $\Omega = \{(x_1, x_2) \in (0,1) \times (0,1)\}$ and discretize it as it described in the previous section.

The PETSc program of Appendix A are available for running of this example. We have executed the main program

```
cplxmaxwell.cpp
```

using version of PETSc

```
petsc-3.10.4c
```

on 64 bits Red Hat Linux Workstation. An example of Makefile used for compilation of PETSc program in Appendix A is presented below:

```
PETSC_ARCH=/chalmers/sw/sup64/petsc-3.10.4c

include ${PETSC_ARCH}/lib/petsc/conf/variables
include ${PETSC_ARCH}/lib/petsc/conf/rules

MPI_INCLUDE = ${PETSC_ARCH}/include/mpiuni

CXX = g++
CXXFLAGS = -Wall -Wextra -g -O0 -c -Iinclude -I${PETSC_ARCH}/include
 -I${MPI_INCLUDE}
LD = g++
LFLAGS =
OBJECTS = cplxmaxwell.o
RUNMAXWELL = runmaxwell

all: $(RUNMAXWELL)
%.o: %.cpp
$(CXX) $(CXXFLAGS) -o $@ $<
$(RUNMAXWELL): $(OBJECTS)
$(LD) $(LFLAGS) $(OBJECTS) $(PETSC_LIB) -o $@
```

For solution of system of linear equations (7.8) was used inbuilt PETSc function with the scalable linear equations solvers (KSP) component. This component provides interface to the combination of a Krylov subspace iterative method and a preconditioner which can be chosen by user [25]. It is possible choose between three different preconditioners which are encoded by numbers:

```
1-  Jacobi's method
2 - Gauss-Seidel method
3 - Successive Overrelaxation method (SOR)
```

To run the main program `cplxmaxwell.cpp` one need to write:

```
>runmaxwellv2 argv[1] argv[2]
```

Here, arguments are defined as follows:

```
argv[1] - preconditioner (should be 1,2 or 3)
argv[2] - number of discretization points in x and y directions
```

For example, to execute the main program `cplxmaxwell.cpp` using SOR method and 21 discretization points in x and y directions, one should run this program, as follows:

```
>runmaxwell 3 21
```

The results will be printed in the files

```
nodes.m
values.m
```

and can be visualized in Matlab using the file

```
viewer.m
```

which is available for download in Appendix A. Figure 4 shows these results.

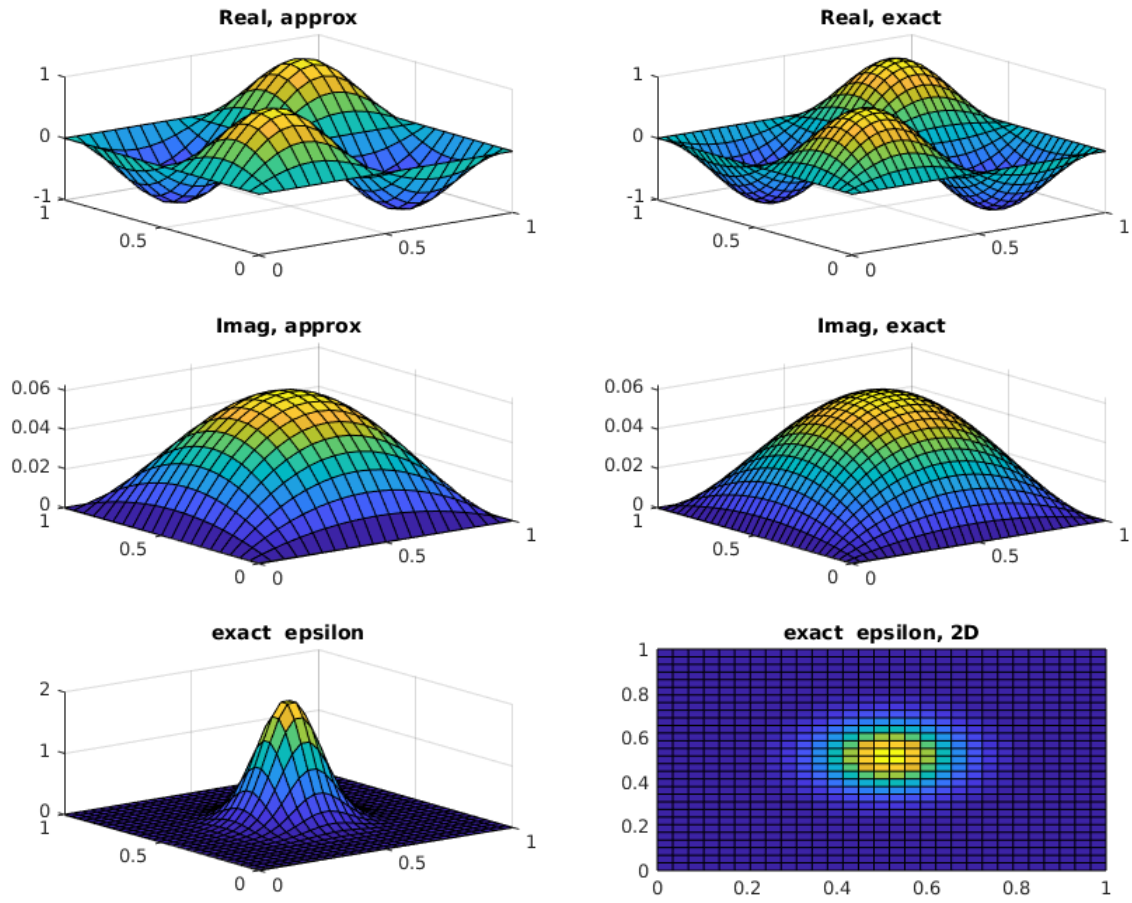


Figure 4: Solution of the problem (7.1) using the C++/PETSc program of Appendix A via SOR with $n_x = n_y = 21$.

A C++/PETSc Program

A.1 Program cplxmaxwell.cpp

```
// to run
// runmaxwell argv[1] argv[2]
// Arguments:
// argv[1] - preconditioner (should be 1,2 or 3)
// argv[2] - number of discretization points in x and y directions

static char help[] ="";
#include<iostream>
#include<fstream>
#include<petsc.h>
#include<petscvec.h>
#include<petscmat.h>
#include<petscksp.h>
#include<complex>

using namespace std;

char METHOD_NAMES[8][70] = {
    "invalid method",
    "Jacobi's method",
    "Gauss-Seidel method",
    "Successive Overrelaxation method (SOR)"};

char *GetMethodName(PetscInt method) {
    if (method < 0 || method > 3)
        return METHOD_NAMES[0];
    else
        return METHOD_NAMES[method];
}


PetscScalar   epsilon(const PetscReal x, const PetscReal y)
{
  PetscReal rpart, ipart;

PetscReal x_0=0.5;
PetscReal y_0=0.5;
PetscReal c_x=1;
PetscReal c_y=1;
```

```
 rpart=2*exp(-((x-x_0)*(x-x_0)/(2*c_x*c_x) +(y-y_0)*(y-y_0)/(2*c_y*c_y)));
  ipart = 0;
  PetscScalar scalareps(rpart, ipart);
  return scalareps;

}


PetscScalar right_hand_side(const PetscReal x, const PetscReal y,
    const PetscReal omega)
{
  PetscReal rpart, ipart, pi = 3.14159265359;


  PetscReal x_0=0.5;
  PetscReal y_0=0.5;
  PetscReal c_x=1;
  PetscReal c_y=1;
  PetscReal epsilon_real =
 2*exp(-((x-x_0)*(x-x_0)/(2*c_x*c_x) +(y-y_0)*(y-y_0)/(2*c_y*c_y)));


  rpart =  -(8*pi*pi)*sin(2*pi*x)*sin(2*pi*y)
+ omega*omega*epsilon_real*(sin(2*pi*x)*sin(2*pi*y));

  ipart = -2*(x - x*x + y - y*y)
 + omega*omega*epsilon_real*x*(1-x)*y*(1-y);

  PetscScalar f(rpart, ipart);
  return f;
}


PetscScalar wave_number(const PetscReal kreal, const PetscReal kimag)
{
  //PetscReal rpart, ipart;
    //rpart = 1;
    //ipart = 1;
    PetscScalar k(kreal, kimag);
    return k;
}

int main(int argc, char **argv)
```

```
{
  PetscErrorCode ierr;

  cout << "Initializing ..." << endl;
  // PetscInitialize(&argc, &argv, NULL, NULL);

 ierr = PetscInitialize(&argc, &argv,(char *)0, help);CHKERRQ(ierr);

  PetscInt method =  atoi(argv[1]);
  PetscBool methodSet = PETSC_FALSE;

ierr = PetscOptionsGetInt(NULL, NULL, "-m", &method, &methodSet);
    if (method < 1 || method > 7) {
        cout << "Invalid number of the selected method: "
     << method << ".\nExiting..." << endl;
        exit(-1);
    }

        PetscPrintf(PETSC_COMM_WORLD, "Using %s\n", GetMethodName(method));

  cout << "Setting parameters..." << endl;
   Vec b, u;
  Mat A;
  KSP ksp;
  PC preconditioner;
  PetscInt Nx = atoi(argv[2]), Ny = Nx, Nsys, node_idx = 0, col[5], nadj;

  Nsys = Nx*Ny; // dimension of linear system = number of nodes
  PetscReal x[Nx], y[Ny], nodes[Nsys][2];
  PetscScalar value, value_epsilon, diffpoints[5], h;


  // Set up vectors
  cout << "Setting up vectors..." << endl;
  ierr = VecCreate(PETSC_COMM_WORLD, &b); CHKERRQ(ierr);
  ierr = VecSetSizes(b, PETSC_DECIDE, Nsys); CHKERRQ(ierr);
  ierr = VecSetType(b, VECSTANDARD); CHKERRQ(ierr);
  ierr = VecDuplicate(b, &u);



  // Set up matrix
  cout << "Setting up matrix..." << endl;
```

```
  ierr = MatCreate(PETSC_COMM_WORLD, &A); CHKERRQ(ierr);
  ierr = MatSetSizes(A,PETSC_DECIDE, PETSC_DECIDE, Nsys, Nsys);
   CHKERRQ(ierr);
  ierr = MatSetFromOptions(A); CHKERRQ(ierr);
  ierr = MatSetUp(A); CHKERRQ(ierr);

  // Create grid
  cout << "Constructing grid..." << endl;
  h = 1.0/(Nx - 1);
  for (int i = 0; i < Nx; i++)
    x[i] = 1.0*i/(Nx - 1);
  for (int j = 0; j < Ny; j++)
    y[j] = 1.0*j/(Ny - 1);

  // Assemble linear system ...
  cout << "Assembling system..." << endl;

  PetscScalar k;
  double omegareal=40;
  for (int i = 0; i < Nx; i++)
    {
      for (int j = 0; j < Ny; j++)
{
  nodes[node_idx][0] = x[i];
  nodes[node_idx][1] = y[j];

   k = omegareal*omegareal*epsilon(x[i], y[j]);
   value_epsilon =  h*h*k;

diffpoints[0] =  -4.0 + h*h*k;
        diffpoints[1] = 1.0;
        diffpoints[2] = 1.0;
        diffpoints[3] = 1.0;
        diffpoints[4] = 1.0;

  if (i > 0 && i < Nx - 1 && j > 0 && j < Ny - 1) // interior
    {
            col[0] = node_idx;
            col[1] = node_idx - 1;
            col[2] = node_idx + 1;
            col[3] = node_idx - Ny;
            col[4] = node_idx + Ny;
```

```
            nadj = 5;
            value = h*h*right_hand_side(x[i], y[j],omegareal);


    } else // on boundary
    {
      col[0] = node_idx;
      nadj   = 1;
      value  = 0.0;
    }
ierr = MatSetValues(A, 1, &node_idx, nadj, col, diffpoints, INSERT_VALUES);
CHKERRQ(ierr);
ierr = VecSetValues(b, 1, &node_idx, &value, INSERT_VALUES);
CHKERRQ(ierr);

  node_idx++;
}
    }
  ierr = MatAssemblyBegin(A, MAT_FINAL_ASSEMBLY); CHKERRQ(ierr);
  ierr = MatAssemblyEnd(A, MAT_FINAL_ASSEMBLY); CHKERRQ(ierr);

  // Solve linear system
  cout << "Solving linear system ..." << endl;
  ierr = KSPCreate(PETSC_COMM_WORLD, &ksp); CHKERRQ(ierr);
  ierr = KSPSetOperators(ksp, A, A); CHKERRQ(ierr);

  // set preconditioner
   ierr = KSPGetPC(ksp, &preconditioner); CHKERRQ(ierr);

    if (method == 1)
      {
 ierr = PCSetType(preconditioner,PCJACOBI); CHKERRQ(ierr);
      }
    else if (method == 2)
      {

 ierr = PCSetType(preconditioner, PCSOR);
  CHKERRQ(ierr);
      }
else if (method == 3)
  {
    const PetscReal omega = 1.5;
    ierr = PCSetType(preconditioner, PCSOR); CHKERRQ(ierr);
```

```
    ierr = PCSORSetOmega(preconditioner, omega); CHKERRQ(ierr);
  }


  ierr = KSPSetFromOptions(ksp); CHKERRQ(ierr);
  ierr = KSPSolve(ksp, b, u); CHKERRQ(ierr);

  // Print to files
  cout << "Writing to files..." << endl;
  FILE* nodefile = fopen("nodes.m", "w");
  for (int idx = 0; idx < Nsys; idx++)
    fprintf(nodefile, "%f \t %f \n", nodes[idx][0], nodes[idx][1]);
  fclose(nodefile);
  FILE* solfile = fopen("values.m", "w");
  for (int idx = 0; idx < Nsys; idx++)
    {
      ierr = VecGetValues(u, 1, &idx, &value);
      fprintf(solfile, "%f \t %f \n", real(value), imag(value));
    }
  fclose(solfile);

  // Clean up
  ierr = VecDestroy(&b); CHKERRQ(ierr);
  ierr = VecDestroy(&u); CHKERRQ(ierr);
  ierr = MatDestroy(&A); CHKERRQ(ierr);
  ierr = KSPDestroy(&ksp); CHKERRQ(ierr);

  // Finalize and finish
  ierr = PetscFinalize();
  return 0;
}
```

A.2 Matlab program viewer.m for visualization of results

```
load nodes.m
load values.m
u = @(x, y) sin(2*pi*x).*sin(2*pi*y) + 1i*x.*(1 - x).*y.*(1 - y);
x_0=0.5;
y_0=0.5;
c_x= 0.1;
c_y=0.1;
epsilon = @(x, y) 2*exp(-((x-x_0).*(x-x_0)/(2*c_x.*c_x) ...
```

```
+(y-y_0).*(y-y_0)/(2*c_y.*c_y)));
% for test 2
%epsilon = @(x, y) 1+2*exp(-((x-0.5).*(x-0.5) +(y-0.7).*(y-0.7))/0.001) ...
 +  3*exp(-((x-0.2).*(x-0.2) +(y-0.6).*(y-0.6))/0.001);
n = sqrt(size(nodes, 1));
X = reshape(nodes(:, 1), n, n);
Y = reshape(nodes(:, 2), n, n);
Ur = reshape(values(:, 1), n, n);
Ui = reshape(values(:, 2), n, n);
[Xe, Ye] = meshgrid(linspace(0, 1, 30), linspace(0, 1, 30));
ur = real(u(Xe, Ye));
ui = imag(u(Xe, Ye));
Eps = epsilon(Xe, Ye)
subplot(3, 2, 1)
surf(X', Y', Ur)
title('u_h Real, computed')
  view(2)
subplot(3, 2, 2)
surf(Xe, Ye, ur)
title('u Real, exact')
  view(2)
subplot(3, 2, 3)
surf(X', Y', Ui)
title('u_h Imag, computed')
  view(2)
subplot(3, 2, 4)
surf(Xe, Ye, ui)
title('u Imag, exact')
  view(2)
subplot(3, 2, 5)
surf(Xe, Ye, Eps)
  title('exact  epsilon, 3D view')
subplot(3, 2, 6)
surf(Xe, Ye, Eps)
view(2)
title('exact  epsilon, 2D')
shg
```

References

[1] Bakushinsky A., Kokurin M.Y., Smirnova A., *Iterative Methods for Ill-posed Problems*, Inverse and Ill-Posed Problems Series 54, De Gruyter, 2011.

[2] L. Beilina, E. Karchevskii, M. Karchevskii, Numerical Linear Algebra: Theory and Applications, Springer, 2017.

[3] L. Beilina and C. Johnson, A posteriori error estimation in computational inverse scattering, *Mathematical Models in Applied Sciences*, 1, 23-35, 2005.

[4] L. Beilina, Application of the finite element method in a quantitative imaging technique, *J. Comput. Methods Sci. Eng.*, IOS Press, 16(4), 755-771, 2016. DOI 10.3233/JCM-160689.

[5] Beilina, Domain decomposition finite element/finite difference method for the conductivity reconstruction in a hyperbolic equation, *Communications in Nonlinear Science and Numerical Simulation*, Elsevier, 2016, doi:10.1016/j.cnsns.2016.01.016

[6] L. Beilina and M. V. Klibanov, *Approximate global convergence and adaptivity for Coefficient Inverse Problems*, Springer, New York, 2012.

[7] L. Beilina, V. Ruas, An explicit P1 finite element scheme for Maxwell's equations with constant permittivity in a boundary neighborhood, arXiv:1808.10720.

[8] L. Beilina, N. T. Th'anh, M. Klibanov, and J. B. Malmberg, Reconstruction of shapes and refractive indices from blind backscattering experimental data using the adaptivity, *Inverse Problems*, 30, 105007, 2014.

[9] L. Beilina, N. T. Th'anh, M.V. Klibanov and J. B. Malmberg, Globally convergent and adaptive finite element methods in imaging of buried objects from experimental backscattering radar measurements, *Journal of Computational and Applied Mathematics, Elsevier*, DOI: 10.1016/j.cam.2014.11.055, 2015.

[10] S. C. Brenner and L. R. Scott, *The Mathematical Theory of Finite Element Methods*, Springer-Verlag, Berlin, 1994.

[11] J. Bondestam Malmberg, L. Beilina, An Adaptive Finite Element Method in Quantitative Reconstruction of Small Inclusions from Limited Observations, *Appl. Math. Inf. Sci.*, 12(1), 1-19, 2018.

[12] Y. Chen, Inverse scattering via Heisenberg uncertainty principle, *Inverse Problems*, 13, 253-282, 1997.

[13] G. C. Cohen, *Higher Order Numerical Methods for Transient Wave Equations*, Springer-Verlag, Berlin, 2002.

[14] A. E. Bulyshev, A. E. Souvorov, S. Y. Semenov, V. G. Posukh and Y. E. Sizov, Three-dimensional vector microwave tomography: theory and computational experiments, *Inverse Problems*, 20(4), pp.1239-1259, 2004.

[15] A. Elmkies and P. Joly, Finite elements and mass lumping for Maxwell's equations: the 2D case. *Numerical Analysis*, C. R. Acad.Sci.Paris, 324, pp. 1287–1293, 1997.

[16] H. W. Engl, M. Hanke, and A. Neubauer, *Regularization of Inverse Problems*, Kluwer Academic Publishers, Dordrecht, The Netherlands, 1996.

[17] B. Engquist and A. Majda, Absorbing boundary conditions for the numerical simulation of waves, *Math. Comp.*, 31, 629-651, 1977.

[18] Joannopoulos, Johnson, Winn and Meade, *Photonic Crystals: Molding the Flow of Light*, Second edition, Princeton Univ. Press, 2008.

[19] A. Greenleaf, Y. Kurylev, M. Lassas and G. Uhlmann, Cloaking devices, electromagnetic wormholes, and transformation optics, SIAM Rev. 51(1) (2009) 3-33.

[20] Kuzhuget, A.V., Beilina, L., Klibanov, M.V., Sullivan, A., Nguyen, L., Fiddy, M.A., Blind experimental data collected in the field and an approximately globally convergent inverse algorithm, *Inverse Problems*, V.28, N.9, 2012, DOI:10.1088/0266-5611/28/9/095007

[21] O. A. Ladyzhenskaya, *Boundary Value Problems of Mathematical Physics*, Springer-Verlag, Berlin, 1985.

[22] C. Eyraud, J.-M. Geffrin, and A. Litman, 3-D imaging of a microwave absorber sample from microwave scattered field measurements, *IEEE Microwave and Wireless Components Letters*, **25(7)**(2015), 472–474.

[23] T. M. Grzegorczyk, P. M. Meaney, P. A. Kaufman, R. M. diFlorio Alexander, and K.D. Paulsen, Fast 3-d tomographic microwave imaging for breast cancer detection, *IEEE Trans Med Imaging*, **31** (2012), 1584–1592.

[24] M. Pastorino, *Microwave Imaging*, John Wiley Sons, Hoboken, NJ, 2010.

[25] Portable, Extensible Toolkit for Scientific Computation PETSc at http://www.mcs.anl.gov/petsc/

[26] Software package WavES at http://www.waves24.com/