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

The mathematical model

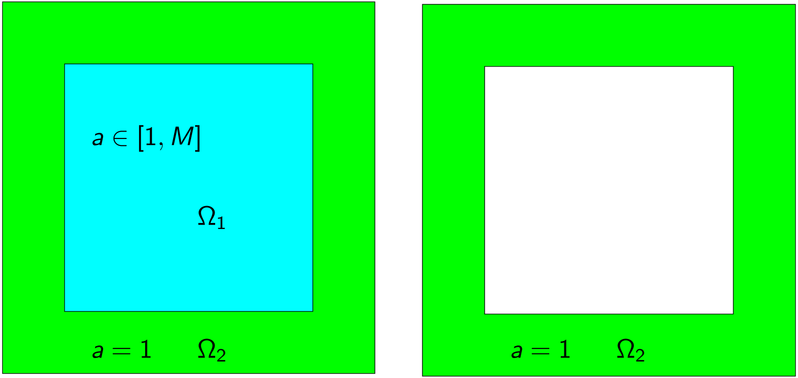
Our basic model is given in terms of the function $u(x, s), x \in \mathbb{R}^d, d = 2, 3$ which depends on the pseudo-frequency $s > const. > 0$:

$$\triangle u(x) - s^2 a(x) u(x) = -s a(x) f_0(x), \quad x \in \mathbb{R}^d, d = 2, 3, \qquad (1)$$

Here, $a(x) = 1/\sqrt{c(x)}$, where $c(x)$ is the sound speed.
To solve the problem (1) numerically in $\mathbb{R}^d, d = 2, 3$ we will use the domain decomposition finite element/finite difference method of [BeilinaHyb]. 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.

[BeilinaHyb] 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

Domain decomposition in Ω



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

b) Ω_2

Figure: *Domain decomposition in Ω.*

The model equation (1) can be obtained by applying the Laplace transform in time,

$$u(x,s) := \int_0^{+\infty} u(x,t) e^{-st} dt, \qquad s = const. > 0 \tag{2}$$

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

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

After applying the Laplace transform for the problem (3) and to the absorbing and Neumann boundary conditions

$$\begin{aligned} \partial_\nu u + \partial_t u = 0, &\qquad (x,t) \in (\Gamma_1 \cup \Gamma_2) \times (0, T], \\ \partial_\nu u = 0, &\qquad (x,t) \in \Gamma_3 \times (0, T]. \end{aligned} \tag{4}$$

we get the following model problem

$$\begin{aligned} \triangle u(x) - s^2 a(x) u(x) = -sa(x) f_0(x), &\quad x \in \mathbb{R}^d, d = 2, 3, \\ \partial_\nu u(x) = 0, &\quad x \in \Gamma_3, \\ \partial_\nu u(x) = f_0(x) - su(x,s), &\ \ x \in \Gamma_1 \cup \Gamma_2. \end{aligned} \tag{5}$$

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

$$\begin{aligned} &a(x) \in [1, M], \quad \text{for } x \in \Omega_1, \\ &a(x) = 1, \ \text{for } x \in \Omega \setminus \Omega_1. \end{aligned} \tag{6}$$

and assume that

$$f_0 \in H^1(\Omega), \quad c(x) \in C^2(\mathbb{R}^d), d = 2, 3. \tag{7}$$

Let us introduce the following space of real valued functions

$$U^1 = H^1(\Omega) \times H^1(\Omega) \times C\left(\overline{\Omega}\right). \tag{8}$$

We consider the following inverse problem

Inverse Problem (IP)

Let the coefficient $a(x)$ in the problem (5) satisfies conditions (6) and assume that $a(x)$ is unknown in the domain Ω_1. Determine the function $a(x)$ in (5) for $x \in \Omega_1$ assuming that the following function $\widetilde{u}(x)$ is known

$$u(x) = \widetilde{u}(x) \ \forall x \in \Gamma. \tag{9}$$

Lagrangian approach for solution IP

The reconstruction method to solve inverse problem **IP** is based on the finding of the stationary point of the following Tikhonov functional

$$F(u,c) = \frac{1}{2}\int_{\Gamma}(u - \widetilde{u})^2 z_{\delta}(x) dS + \frac{1}{2}\gamma \int_{\Omega}(a - a_0)^2 \quad dx, \qquad (10)$$

where u satisfies the equations (5), a_0 is the initial guess for a, $\widetilde{u}$ is the observed field at Γ, $\gamma > 0$ is the regularization parameter and $z_{\delta}(x)$ is the compatibility function.

To find minimum of (10) we apply the Lagrangian approach and define the following Lagrangian in the week form using definition of the forward model problem (5)

$$L(v) = F(u,a) + (\lambda, f_0 - su)_{\Gamma_1 \cup \Gamma_2} - (\nabla u, \nabla \lambda)_{\Omega} - (\lambda, s^2 au)_{\Omega} + (\lambda, saf_0)_{\Omega}, \qquad (11)$$

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 $\bar{v} = (\bar{u}, \bar{\lambda}, \bar{\bar{a}}) \in U^1$

$$L'(v; \bar{v}) = 0, \qquad (12)$$

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

Fréchet derivative of the Lagrangian

To find the Fréchet derivative (12) of the Lagrangian (11) 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. The optimality condition (12) for the Lagrangian (11) for all $\bar{v} \in U^1$ is

$$L'(v;\bar{v}) = \frac{\partial L}{\partial \lambda}(v)(\bar{\lambda}) + \frac{\partial L}{\partial u}(v)(\bar{u}) + \frac{\partial L}{\partial a}(v)(\bar{a}) = 0. \tag{13}$$

Thus, to satisfy optimum condition $L'(v;\bar{v}) = 0$ every component of (13) should be zero out.

Fréchet derivative of the Lagrangian

Using integration by parts together with boundary conditions in (5) we get

$$\begin{aligned} 0 = \frac{\partial L}{\partial \lambda}(v)(\bar{\lambda}) &= \int_{\Omega} (\triangle u - s^2 au + saf_0)\bar{\lambda}\ dx \\ &+ \int_{\Gamma_1 \cup \Gamma_2} (f_0 - su)\bar{\lambda}\ dS - \int_{\Gamma} \partial_n u \bar{\lambda}\ dS,\ \forall \bar{\lambda} \in H^1(\Omega), \end{aligned} \tag{14}$$

$$\begin{aligned} 0 = \frac{\partial L}{\partial u}(v)(\bar{u}) &= \int_{\Gamma} (u - \widetilde{u})\ \bar{u}\ z_\delta\ dS + \int_{\Omega} (\triangle \lambda - s^2 a\lambda)\bar{u}\ dx \\ &- \int_{\Gamma_1 \cup \Gamma_2} s\lambda\bar{u}\ dS - \int_{\Gamma} \partial_n \lambda \bar{u}\ dS,\ \forall \bar{u} \in H^1(\Omega), \end{aligned} \tag{15}$$

$$0 = \frac{\partial L}{\partial a}(v)(\bar{a}) = \int_{\Omega} (-s^2 \lambda u + s\lambda f_0)\ \bar{a}\ dx + \gamma \int_{\Omega_1} (a - a_0)\ \bar{a}\ dx,\ \forall \bar{a} \in C\left(\overline{\Omega}\right). \tag{16}$$

Fréchet derivative of the Tikhonov functional

It is clear that (14) corresponds to the state equation (5) and (15) corresponds to the following adjoint problem

$$\begin{aligned} \triangle \lambda - s^2 a \lambda &= -(u - \widetilde{u}) z_\delta, \quad x \in \Omega, \\ \partial_n \lambda &= 0, \quad x \text{ on } \Gamma_3, \\ \partial_n \lambda &= -\lambda s, \quad x \text{ on } \Gamma_1 \cup \Gamma_2. \end{aligned} \tag{17}$$

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 we get from (11)

$$F(u(a), a) = L(v(a)), \tag{18}$$

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

$$F'(a) := F'(u(a), a) = \frac{\partial F}{\partial a}(u(a), a) = \frac{\partial L}{\partial a}(v(a)). \tag{19}$$

Inserting (16) into (19), 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

$$F'(a)(x) := F'(u(a), a)(x) = -s^2 \lambda u + s \lambda f_0 + \gamma (a - a_0). \tag{20}$$

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 (10) 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

$$g^m(x) = -s^2 \lambda_h^m u_h^m + s\lambda^m f_0 + \gamma^m (a_h^m - a_0), \tag{21}$$

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 (5) and the adjoint problem (17), respectively, with $a := a_h^m$, γ^m is iteratively computed regularization parameter via rules of [BKS] as

$$\gamma^m = \frac{\gamma_0}{(m+1)^p}, \quad p \in (0, 1). \tag{22}$$

[BKS] Bakushinsky A., Kokurin M.Y., Smirnova A.,
Iterative Methods for Ill-posed Problems,

Inverse and Ill-Posed Problems Series 54, De Gruyter, 2011.

Conjugate gradient algorithm

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

$$a^{m+1}(x) = a^m(x) + \alpha^m d^m(x), \tag{23}$$

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

$$d^m = -g^m \tag{24}$$

and for the conjugate gradient method as

$$d^m = -g^m + \beta_m d_{m-1}, \tag{25}$$

at the iteration m where parameters β_m are computed as in (31).

Step-size in the gradient update

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 (11) we have

$$\begin{aligned} L(u_h^m, \lambda_h^m, a_h^m + \alpha^m d^m) = F(u_h^m, a_h^m + \alpha^m d^m) \; + \\ \int_\Omega \lambda_h^m [-s^2 (a_h^m + \alpha^m d^m) u_h^m + s(a_h^m + \alpha^m d^m) f_0] \; dx \\ - \int_\Omega \nabla \lambda_h^m \nabla u_h^m \; dx + \int_{\Gamma_1 \cup \Gamma_2} (f_0 - s u_h^m) \lambda_h^m \; dS, \end{aligned} \tag{26}$$

$$\begin{aligned} L'_{\alpha^m}(u_h^m, \lambda_h^m, a_h^m + \alpha^m d^m) = \gamma^m \int_\Omega (a_h^m + \alpha^m d^m - a_0) d^m \; dx \\ + \int_\Omega \lambda_h^m (-s^2 d^m u_h^m + s d^m f_0) \; dx = 0. \end{aligned} \tag{27}$$

Step-size in the gradient update

We can obtain directly from (27)

$$\alpha^m = \frac{(\lambda_h^m(s^2 u_h^m - sf_0) - \gamma^m a_h^m + \gamma^m a_0, d^m)}{\gamma^m(d^m, d^m)} \tag{28}$$

or using definition (21) for g^m we obtain following expression for computation of the iterative parameter α^m:

$$\alpha^m = -\frac{(g^m, d^m)}{\gamma^m(d^m, d^m)}. \tag{29}$$

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 (5) and adjoint (17) 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

$$a_h^{m+1} = a_h^m + \alpha^m d^m(x), \tag{30}$$

where

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

with

$$\beta^m = \frac{\|g^m(x)\|^2}{\|g^{m-1}(x)\|^2}, \tag{31}$$

where $d^0(x) = -g^0(x)$ and

$$\alpha^m = -\frac{(g^m, d^m)}{\gamma^m \|d^m\|^2}. \tag{32}$$

Goal of the project

In the project report should be presented numerical simulations for reconstruction of the function $a(x)$ in (5)

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

via CGM algorithm in the domain $\Omega = [0,1] \times [0,1]$ using data generated at the boundary Γ of Ω described in the course project at

`www.waves24.com/download`

Alternatively, you can generate your own data as it is described in the course project at

`www.waves24.com/download`

and solve **IP** via CGM.

Description of the process of data generation via FEM/FDM domain decomposition method

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. 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 (3) with first order absorbing boundary conditions in time $T = [0, 2.0]$ with the time step $\tau = 0.002$ which satisfies to the CFL condition. More precisely, the model problem for time-dependent wave equation is

$$\begin{aligned} a(x)\partial_t^2 U(x,t) - \triangle U(x,t) &= 0 \text{ in } \ \Omega \times (0,T), \\ U(x,0) = 0, \quad U_t(x,0) &= 0 \text{ in } \ \Omega, \\ \partial_n U &= f(t) \text{ on } \Gamma_{1,1}, \\ \partial_n U &= -\partial_t U \text{ on } \Gamma_{1,2} \cup \Gamma_2, \\ \partial_n U &= 0 \text{ on } \Gamma_3. \end{aligned} \tag{33}$$

In (33) the function $f(t)$ is defined as

$$f(t) = \begin{cases} \sin(\omega_s t), & \text{if } t \in \left(0, \frac{2\pi}{\omega_s}\right), \\ 0, & \text{if } t > \frac{2\pi}{\omega_s}. \end{cases} \tag{34}$$

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

Functions $a(x)$ in Test 1 and Test 2

We assume that $a(x) = 1$ is known inside Ω_2. The wave speed function was chosen as (35) in Test 1 and as (36) in Test 2. Then the Laplace transform (2) was applied to the computed time-dependent solution $u(x,t)$ of the problem (33) at the pseudo-frequency interval for $s \in [2.0, 7.0]$, then additive noise σ was added to the obtained solution.

- Test 1

  In this test should be reconstructed one smooth function given by

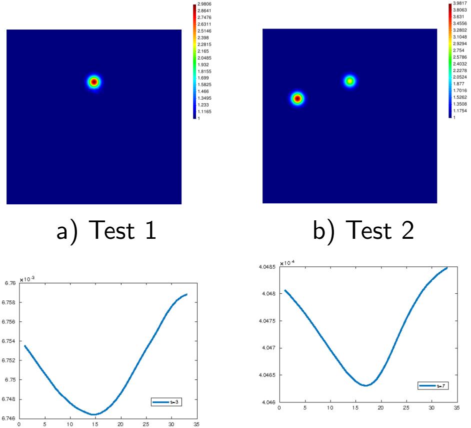
$$a(x) = 1.0 + 2.0 \cdot \mathrm{e}^{-((x_1-0.5)^2+(x_2-0.7)^2)/0.001}. \tag{35}$$

- Test 2

  In this test should be reconstructed two smooth functions given by

$$\begin{aligned} a(x) &= 1.0 \\ &+ 2.0 \cdot \mathrm{e}^{-((x_1-0.5)^2+(x_2-0.7)^2)/0.001} + 3.0 \cdot \mathrm{e}^{-((x_1-0.2)^2+(x_2-0.6)^2)/0.001}. \end{aligned} \tag{36}$$

Functions $a(x)$ in Test 1 and Test 2



a) Test 1

b) Test 2

c) $s = 3$

d) $s = 7$

Figure: a), b) Functions $a(x)$ in Ω_1 used in Test 1 (left figure) and Test 2 (right figure). s), d) Laplace transform of data $\tilde{U}(x, t)$ with 3% noise at the backscattered boundary Γ_1.

Hints for solution IP

:

1. Study Examples 8.2, 8.4.4 or 12.5 of [BKK] where is presented solution of the Dirichlet problem for the Poisson's equation using PETSc. PETSc programs for solution of this problem are available download from the course homepage: go to the link of the book [BKK] and click to "GitHub Page with MATLAB® Source Codes" on the bottom of this page.
2. Modify PETSc code of the Example 12.5 of [BKK], or PETSc code in this note (this code is also available for download from waves24.com/download) such that it can be applied for the solution of **IP**.

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

Example: solution of Poisson's equation [BKK]

The model problem is the following Dirichlet problem for Poisson's equation:

$$\begin{aligned} -\triangle u(x) &= f(x) \ \text{ in } \ \Omega, \\ u &= 0 \text{ on } \partial\Omega. \end{aligned} \tag{37}$$

Here $f(x)$ is a given function, $u(x)$ is the unknown function, and the domain Ω is the unit square $\Omega = \{(x_1, x_2) \in (0,1) \times (0,1)\}$. To solve numerically (37) 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. 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 $0 < i, j \leq n$ 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 as:

$$n_{glob} = i + n_j \cdot (j - 1). \tag{38}$$

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

$$\Delta u_{i,j} = \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{h^2} + \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{h^2}, \qquad (39)$$

where $u_{i,j}$ is the solution at the discrete point (i,j). Using (39), we obtain the following scheme for solving problem (37):

$$-\left(\frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{h^2} + \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{h^2} \right) = f_{i,j}, \qquad (40)$$

where $f_{i,j}$ are the value of the function f at the discrete point (i,j). Then (40) can be rewritten as

$$-\left(u_{i+1,j} - 2u_{i,j} + u_{i-1,j} + u_{i,j+1} - 2u_{i,j} + u_{i,j-1}\right) = h^2 f_{i,j}, \qquad (41)$$

or in the more convenient form as

$$-u_{i+1,j} + 4u_{i,j} - u_{i-1,j} - u_{i,j+1} - u_{i,j-1} = h^2 f_{i,j}. \qquad (42)$$

System (42) can be written in the form $Au = b$. The vector b has the components $b_{i,j} = h^2 f_{i,j}$. The explicit elements of the matrix A are given by the following block matrix

$$A = \left(\begin{array}{c|cc|c} A_N & -I_N & & \\ \hline -I_N & \ddots & \ddots & \\ & \ddots & \ddots & -I_N \\ \hline & & -I_N & A_N \end{array} \right)$$

with blocks A_N of order N given by

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

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. The matrix A is symmetric and positive definite and we can use the LU factorization algorithm without pivoting.

Suppose, that we have discretized the two-dimensional domain Ω as described above with $N = n_i = n_j = 3$. We present the schematic discretization via the global nodes numbering

$$n_{glob} = i + n_j \cdot (j - 1).$$

in the following scheme:

$$\begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,1} & a_{3,2} & a_{3,3} \end{pmatrix} \Longrightarrow \begin{pmatrix} n_1 & n_2 & n_3 \\ n_4 & n_5 & n_6 \\ n_7 & n_8 & n_9 \end{pmatrix} \Longrightarrow \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}. \tag{43}$$

Then the explicit form of the block matrix A will be:

$$A = \left(\begin{array}{rrr|rrr|rrr} 4 & -1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ -1 & 4 & -1 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 4 & 0 & 0 & -1 & 0 & 0 & 0 \\ \hline -1 & 0 & 0 & 4 & -1 & 0 & -1 & 0 & 0 \\ 0 & -1 & 0 & -1 & 4 & -1 & 0 & -1 & 0 \\ 0 & 0 & -1 & 0 & -1 & 4 & 0 & 0 & -1 \\ \hline 0 & 0 & 0 & -1 & 0 & 0 & 4 & -1 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & -1 & 4 & -1 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & -1 & 4 \end{array}\right).$$

Example 8.2: Gaussian elimination for solution of Poisson's equation

We illustrate the numerical solution of problem (37). We define the right hand side $f(x)$ of (37) as

$$f(x_1, x_2) = A_f \exp\left(-\frac{(x_1 - c_1)^2}{2s_1^2} - \frac{(x_2 - c_2)^2}{2s_2^2}\right)\frac{1}{a(x_1, x_2)}, \tag{44}$$

The coefficient $a(x_1, x_2)$ in (44) is given by the following Gaussian function:

$$a(x_1, x_2) = 1 + A\exp\left(-\frac{(x_1 - c_1)^2}{2s_1^2} - \frac{(x_2 - c_2)^2}{2s_2^2}\right), \tag{45}$$

Here A, A_f are the amplitudes of these functions, c_1, c_2 are constants which show the location of the center of the Gaussian functions, and s_1, s_2 are constants which show spreading of the functions in x_1 and x_2 directions.

We produce the mesh with the points (x_{1i}, x_{2j}) such that $x_{1i} = ih, x_{2j} = jh$ with $h = 1/(N+1)$, where N is the number of the inner points in x_1 and x_2 directions. The linear system of equations $Au = f$ is solved then via the LU factorization of the matrix A without pivoting.

Example 8.2: solution of Poisson's equation using LU factorization

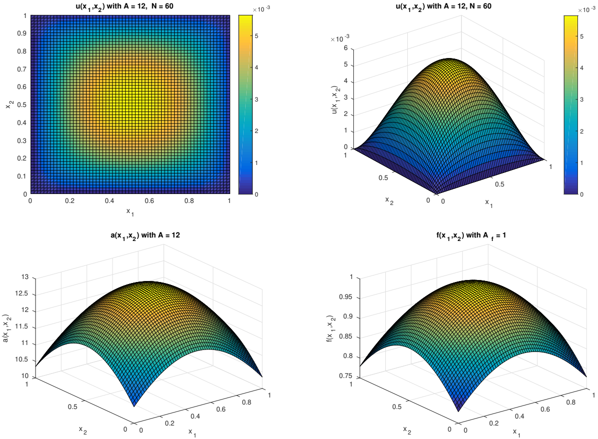


Figure: Solution of Poisson's equation (37) with $f(x_1, x_2)$ as in (44) and $a(x_1, x_2)$ as in (45).

Example 8.4.4: solution of Poisson's equation using Cholesky factorization

$$f(x_1, x_2) = 1 + 10e^{\left(-\frac{(x_1-0.25)^2}{0.02} - \frac{(x_2-0.25)^2}{0.02}\right)} + 10e^{\left(-\frac{(x_1-0.75)^2}{0.02} - \frac{(x_2-0.75)^2}{0.02}\right)} \quad (46)$$

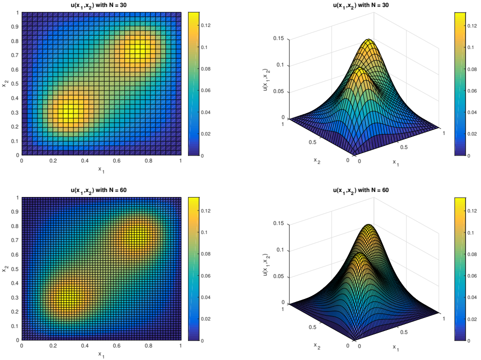


Figure: Solution of Poisson's equation (37) with $f(x_1, x_2)$ as in (46).

Solution of the test problem

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

$$\begin{aligned} \triangle u(x) + \omega^2 \varepsilon(x) u = f(x) \ \text{ in } \ \Omega, \\ u = 0 \text{ on } \partial\Omega. \end{aligned} \tag{47}$$

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 (47) with the right hand side

$$\begin{aligned} f(x_1, x_2) = -(8\pi^2) \sin(2\pi x_1) \sin(2\pi x_2) - 2ix_1(1 - x_1) - 2ix_2(1 - x_2) \\ + \omega^2 \varepsilon(x)(\sin(2\pi x_1) \sin(2\pi x_2) + ix_1(1 - x_1)x_2(1 - x_2)) \end{aligned} \tag{48}$$

is the function

$$u(x_1, x_2) = \sin(2\pi x_1) \sin(2\pi x_2) + ix_1(1 - x_1)x_2(1 - x_2). \tag{49}$$

Description of C++/PETSc solver

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 main program

```
cplxmaxwell.cpp
```

is compiled using version of PETSc

```
petsc-3.10.4c
```

on 64 bits Red Hat Linux Workstation as

```
make runmaxwell
```

Makefile

An example of Makefile used for compilation of PETSc program cplxmaxwell.cpp which we present below is:

```
  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 $Ax = b$ 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 [PETSc]. 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
```

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

Preconditioning for Linear Systems

Preconditioning technique is used for the reduction of the condition number of the considered problem. For the solution of linear system of equations $Ax = b$ the preconditioner matrix P of a matrix A is a matrix $P^{-1}A$ such that $P^{-1}A$ has a smaller condition number then the original matrix A. This means that instead of the solution of a system $Ax = b$ we will consider solution of the system

$$P^{-1}Ax = P^{-1}b. \tag{50}$$

The matrix P should have the following properties:

- P is s.p.d. matrix;
- $P^{-1}A$ is well conditioned;
- The system $Px = b$ should be easy solvable.

The preconditioned conjugate gradient method is derived as follows. First we multiply both sides of (50) by $P^{1/2}$ to get

$$(P^{-1/2}AP^{-1/2})(P^{1/2}x) = P^{-1/2}b. \tag{51}$$

Preconditioning for Linear Systems

We note that the system (51) is s.p.d. since we have chosen the matrix P such that $P = QQ^T$ which is the eigendecomposition of P. Then the matrix $P^{1/2}$ will be s.p.d. if it is defined as

$$P^{1/2} = Q^{1/2}Q^T.$$

Defining

$$\tilde{A} := P^{-1/2}AP^{-1/2}, \tilde{x} := P^{1/2}x, \tilde{b} = P^{-1/2}b$$

we can rewrite (51) as the system $\tilde{A}\tilde{x} = \tilde{b}$. Matrices $\tilde{A}$ and $P^{-1}A$ are similar since $P^{-1}A = P^{-1/2}\tilde{A}P^{1/2}$. Thus, $\tilde{A}$ and $P^{-1}A$ have the same eigenvalues. Thus, instead of the solution of $P^{-1}Ax = P^{-1}b$ we will present preconditioned conjugate gradient (PCG) algorithm for the solution of $\tilde{A}\tilde{x} = \tilde{b}$.

Preconditioned conjugate gradient algorithm

```
Initialization: $r = 0$;   $x_0 = 0$;   $R_0 = b$;   $p_1 = P^{-1}b; y_0 = P^{-1}R_0$
repeat
$r = r + 1$
$z = A \cdot p_r$
$\nu_r = (y_{r-1}^T R_{r-1})/(p_r^T z)$
$x_r = x_{r-1} + \nu_r p_r$
$R_r = R_{r-1} - \nu_r z$
$y_r = P^{-1} R_r$
$\mu_{r+1} = (y_r^T R_r)/(y_{r-1}^T R_{r-1})$
$p_{r+1} = y_r + \mu_{r+1} p_r$
until $||R_r||_2$ is small enough
```

Common preconditioners

Common preconditioner matrices P are:

- Jacobi preconditioner $P = (a_{11}, ..., a_{nn})$. Such choice of the preconditioner reduces the condition number of $P^{-1}A$ around factor n of its minimal value.
- block Jacobi preconditioner

$$P = \begin{pmatrix} P_{1,1} & ... & 0 \\ ... & ... & ... \\ 0 & ... & P_{r,r} \end{pmatrix} \tag{52}$$

with $P_{i,i} = A_{i,i}, i = 1, ..., r$, for the block matrix A given by

$$A = \begin{pmatrix} A_{1,1} & ... & A_{1,r} \\ ... & ... & ... \\ A_{r,1} & ... & A_{r,r} \end{pmatrix} \tag{53}$$

with square blocks $A_{i,i}, i = 1, ..., r$. Such choice of preconditioner P minimizes the condition number of $P^{-1/2}AP^{-1/2}$ within a factor of r.

- Method of SSOR can be used as a block preconditioner as well. If the original matrix A can be split into diagonal, lower and upper triangular as $A = D + L + L^T$ then the SSOR preconditioner matrix is defined as
$$P = (D + L)D^{-1}(D + L)^T$$
It can also be parametrised by ω as follows:
$$P(\omega) = \frac{\omega}{2-\omega}\left(\frac{1}{\omega}D + L\right)D^{-1}\left(\frac{1}{\omega}D + L\right)^T$$

- Incomplete Cholesky factorization with $A = LL^T$ is often used for PCG algorithm. In this case a sparse lower triangular matrix $\tilde{L}$ is chosen to be close to L. Then the preconditioner is defined as $P = \tilde{L}\tilde{L}^T$.

- Incomplete LU preconditioner.

Solution of the problem (47) using the C++/PETSc program cplmaxwell.cpp via SOR with $n_x = n_y = 21$.

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

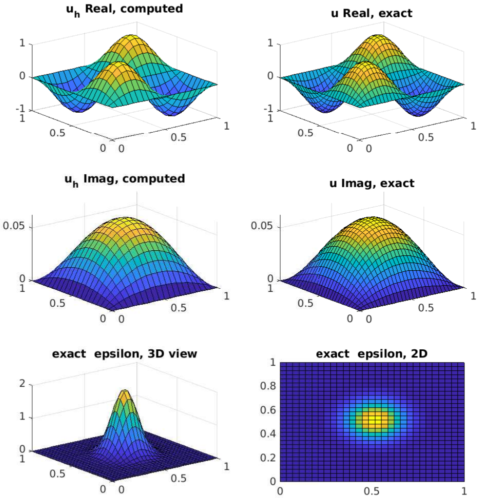
```
nodes.m
values.m
```

and can be visualized in Matlab using the file

```
viewer.m
```

which is available for download on the course homepage, see also below.

Solution of the problem (47) using the C++/PETSc program cplmaxwell.cpp via SOR with $n_x = n_y = 21$.



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;
}
```

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
```