A Study of Advanced Computational Methods

Project in Advanced Scientific Computing

Tselepidis Nikolaos

February 2017

Contents

1 Convection-Diffusion PDE - FDM - Geometric Multigrid

Solve the following partial differential equation:

$$-\epsilon\left(\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}\right)+\alpha\frac{\partial u}{\partial x}=1,\ \ \epsilon,\alpha\in\mathbb{R},\ \ (x,y)\in\Omega\equiv[0,1]^2,$$

$$u(x,y)=0,\ \ (x,y)\in\partial\Omega,$$

using the finite difference method with mesh size $h=1/64$. The respective linear system should be solved using the geometric multigrid method both with the V-cycle and the W-cycle strategy. The Damped Jacobi and Gauss-Seidel smoothers should be utilized and comparative results concerning the convergence behavior of the multigrid method in each case should be given. The termination criterion for the linear system should be the following: $||r_i||<10^{-10}||r_0||$. Solve the PDE for various cases of ϵ and α.

Solution

Let us consider a uniform grid with $h_x=h_y=h$.

In order to achieve error compatibility, all partial derivatives are discretized using central differences.

Hence, it follows that:

$$\left(\frac{\partial^2 u}{\partial x^2}\right)_{i,j}=\frac{u_{i+1,j}-2u_{i,j}+u_{i-1,j}}{h^2}+O(h^2), \tag{1}$$

$$\left(\frac{\partial^2 u}{\partial y^2}\right)_{i,j}=\frac{u_{i,j+1}-2u_{i,j}+u_{i,j-1}}{h^2}+O(h^2), \tag{2}$$

$$\left(\frac{\partial u}{\partial x}\right)_{i,j}=\frac{u_{i+1,j}-u_{i-1,j}}{2h}+O(h^2). \tag{3}$$

Substituting the equations (1), (2) and (3) in the given PDE, yields the following:

$$-\epsilon\left(\frac{u_{i+1,j}-2u_{i,j}+u_{i-1,j}}{h^2}+\frac{u_{i,j+1}-2u_{i,j}+u_{i,j-1}}{h^2}\right)+\alpha\left(\frac{u_{i+1,j}-u_{i-1,j}}{2h}\right)=1\Rightarrow$$

$$-\epsilon\left(\frac{u_{i,j-1}+u_{i-1,j}-4u_{i,j}+u_{i+1,j}+u_{i,j+1}}{h^2}\right)+\alpha\left(\frac{-u_{i-1,j}+u_{i+1,j}}{2h}\right)=1, \tag{4}$$

$$i,j\in[1,m-1].$$

Applying the equation (4) in each one of the internal grid points, considering a natural ordering, yields a linear system of the form $Au=s$, with N equations and N unknowns.

The equation (4) can be rewritten as:

$$-\frac{\epsilon}{h^2}u_{i,j-1}-\left(\frac{\epsilon}{h^2}+\frac{\alpha}{2h}\right)u_{i-1,j}+\frac{4\epsilon}{h^2}u_{i,j}-\left(\frac{\epsilon}{h^2}-\frac{\alpha}{2h}\right)u_{i+1,j}-\frac{\epsilon}{h^2}u_{i,j+1}=1\Rightarrow$$

$$-2\epsilon u_{i,j-1}-(2\epsilon+h\alpha)\,u_{i-1,j}+8\epsilon u_{i,j}-(2\epsilon-h\alpha)\,u_{i+1,j}-2\epsilon u_{i,j+1}=2h^2 \tag{5}$$

The numerical entries in the coefficient matrix A depend on the values of the parameters ϵ and α.

Moreover, it can be observed that:

- Increasing the value of α, leads to a decrease in the numerical symmetry of the coefficient matrix A.
- When α is much greater than ϵ, the coefficient matrix A is not diagonally dominant.

Thus, in order to be able to simulate cases where $\alpha >> \epsilon$, the first derivative in the given PDE should be approximated using backward differences.

Therefore, in such cases, instead of using the equation (3), the following equation should be utilized:

$$\left(\frac{\partial u}{\partial x}\right)_{i,j} = \frac{u_{i,j} - u_{i-1,j}}{h} + O(h) \tag{6}$$

Substituting the equations (1), (2) and (6) in the given PDE, yields the following:

$$-\epsilon\left(\frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{h^2} + \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{h^2}\right) + \alpha\left(\frac{u_{i,j} - u_{i-1,j}}{h}\right) = 1 \Rightarrow$$

$$-\epsilon\left(\frac{u_{i,j-1} + u_{i-1,j} - 4u_{i,j} + u_{i+1,j} + u_{i,j+1}}{h^2}\right) + \alpha\left(\frac{-u_{i-1,j} + u_{i,j}}{h}\right) = 1 \tag{7}$$

$$i, j \in [1, m-1]$$

The equation (7) can be rewritten as:

$$-\frac{\epsilon}{h^2}u_{i,j-1} - \left(\frac{\epsilon}{h^2} + \frac{\alpha}{h}\right)u_{i-1,j} + \left(\frac{4\epsilon}{h^2} + \frac{\alpha}{h}\right)u_{i,j} - \frac{\epsilon}{h^2}u_{i+1,j} - \frac{\epsilon}{h^2}u_{i,j+1} = 1 \Rightarrow$$

$$-\epsilon u_{i,j-1} - (\epsilon + h\alpha)\,u_{i-1,j} + (4\epsilon + h\alpha)\,u_{i,j} - \epsilon u_{i+1,j} - \epsilon u_{i,j+1} = h^2 \tag{8}$$

It can be observed, that decreasing the order of the discretization scheme by one, i.e. from $O(h^2)$ to $O(h)$, we have achieved to move the parameter α into the main diagonal of the coefficient matrix A, and thus, we have constructed a solution method which is robust with respect to any increase in the value of the parameter α. This due to the fact that the matrix A, described by the equation (8), is always diagonally dominant, and no matter how much the value of α is increased, the smoothers Damped Jacobi and Gauss-Seidel, will always be effective.

The MATLAB scripts that were implemented in order to solve the 2D convection-diffusion equation using the geometric multigrid method in conjunction with either the Damped Jacobi or the Gauss-Seidel smoother, and based on either the V-cycle or the W-cycle strategy, are given below. It should be mentioned, that both the V-cycle and the W-cycle strategies are special cases of the μ-cycle multigrid procedure, which is implemented in the mgmu3.m function. The overall solution of the PDE is performed using the script multigrid2d_convection_diffusion.m.

Listing 1: mgmu3.m

```
function x=mgmu3(A,b,x,P,R,M,n1,n2,lev,levmax,omega,mu)
    if (lev==levmax)
        x{lev}=A{lev}\b{lev};
        return;
    end
    for i=1:n1
        x{lev}=x{lev}+omega*(M{lev}\(b{lev}-A{lev}*x{lev}));
    end
    b{lev+1}=R{lev}*(b{lev}-A{lev}*x{lev});
    x{lev+1}=zeros(length(A{lev+1}),1);
    for i=1:mu
        x=mgmu3(A,b,x,P,R,M,n1,n2,lev+1,levmax,omega,mu);
    end
    x{lev}=x{lev}+P{lev}*x{lev+1};
    for i=1:n2
        x{lev}=x{lev}+omega*(M{lev}\(b{lev}-A{lev}*x{lev}));
    end
end
```

Listing 2: multigrid2d_convection_diffusion.m

```
clear;clc;close all;
lev=6; % h=1/(2^lev)
mu=1; % choose 1 for V-cycle or 2 for W-cycle
str='VW';
omega=4/5;
% omega=1;
tol=1e-10;
Nmax=50;
n1=2; % Number of pre-smoothing iterations
n2=2; % Number of post-smoothing iterations
epsilon=1; % Diffusion Coefficient
alpha=25; % Convection Coefficient
A=cell(lev,1); % Coefficient Matrix
T=cell(lev,1); % Diffusion Matrix
K=cell(lev,1); % Convection Matrix
M=cell(lev,1); % Smoother
b=cell(lev,1); % Rhs vector (source)
x=cell(lev,1); % Solution vector
P=cell(lev-1,1); % Prolongation Matrix
R=cell(lev-1,1); % Restriction Matrix
for i=1:lev
    n=2^(lev-i+1)-1;
    T{i}=epsilon*gallery('tridiag',n)*(n+1)*(n+1);
%     K{i}=alpha*spdiags([-ones(n,1) ones(n,1)],[-1 1],n,n)*((n+1)/2);
    K{i}=alpha*spdiags([-ones(n,1) ones(n,1)],[-1 0],n,n)*(n+1);
    A{i}=kron(T{i},speye(n))+kron(speye(n),T{i}+K{i});
    M{i}=diag(diag(A{i})); % Jacobi
%     M{i}=tril(A{i}); % Gauss-Seidel
    x{i}=zeros(n*n,1);
    b{i}=zeros(n*n,1);
end
for i=1:lev-1
    r=2^(lev-i+1)-1;
    c=2^(lev-i)-1;
    P{i}=spalloc(r,c,3*c);
    for j=1:c
        P{i}((1:3)+2*(j-1),j)=[0.5 1 0.5]';
    end
    R{i}=0.5*P{i}';
    P{i}=kron(P{i},P{i});
    R{i}=kron(R{i},R{i});
```

```
end
b{1}=ones(length(A{1}),1);
disp(['2D Convection-Diffusion Equation : ' num2str(length(A{1}))]);
disp('Relative residual');disp(' ');
for i=1:Nmax
    x=mgmu3(A,b,x,P,R,M,n1,n2,1,lev,omega,mu); % Implicit Smoothing
    normr=norm(b{1}-A{1}*x{1})/norm(b{1});
    disp(normr);
    if(normr<tol)
        disp([str(mu) '-cycle multigrid converged in ' ...
            num2str(i) ' iterations.']);
        break;
    end
end
nx=sqrt(length(x{1}));
row=A{1}(floor((nx^2)/2),:);
row=full(row(row~=0)); disp(' '); disp(row);
ddom=abs(row(3))-sum(abs(row([1 2 4 5]))) % diagonal dominance
condest(A{1})
figure, mesh(reshape(x{1},nx,nx)),
title(['Multigrid solution for epsilon = ' ...
    num2str(epsilon) ' and alpha = ' num2str(alpha)]);
figure, mesh(reshape(A{1}\b{1},nx,nx)),
title(['Direct solution for epsilon = ' ...
    num2str(epsilon) ' and alpha = ' num2str(alpha)]);
```

In Tables 1 and 2, the convergence behavior of the geometric multigrid method is presented, for increasing values of the parameter α, and for both the case of *central* (Table 1) and *backward* (Table 2) difference discretization of the first derivative term, respectively. It should be stated, that the mesh size was chosen to be $h = 1/64$, and the value of the parameter ϵ was set to 1. Furthermore, when the Damped Jacobi smoother was utilized, the relaxation parameter ω was set to $4/5$, whereas when the Gauss-Seidel smoother was utilized, the value of ω was set to 1.

Central	$\alpha = 0$	$\alpha = 10$	$\alpha = 15$	$\alpha = 20$	$\alpha = 25$	$\alpha = 30$	$\alpha = 35$
Jacobi (V-cycle)	14	13	13	17	29	33	F
G-S (V-cycle)	10	9	9	8	8	11	F
Jacobi (W-cycle)	11	12	12	12	11	F	F
G-S (W-cycle)	8	8	8	8	8	F	F

Table 1: Convergence behavior of the geometric multigrid method for various values of the parameter α, for the case of *central* difference discretization of the first derivative.

Backward	$\alpha = 0$	$\alpha = 10$	$\alpha = 15$	$\alpha = 20$	$\alpha = 25$	$\alpha = 30$	$\alpha = 35$
Jacobi (V-cycle)	14	22	23	23	23	23	23
G-S (V-cycle)	10	15	15	14	14	14	13
Jacobi (W-cycle)	11	12	12	13	13	13	14
G-S (W-cycle)	8	10	10	10	11	11	11

Table 2: Convergence behavior of the geometric multigrid method for various values of the parameter α, for the case of *backward* difference discretization of the first derivative.

It can be observed, that the Gauss-Seidel smoother is more effective than the Damped Jacobi smoother, as it leads to smaller number of iterations of the geometric multigrid method. Furthermore, the W-cycle strategy has as a result improved convergence behavior of the multigrid method compared to the V-cycle strategy. Finally, as expected, in the cases when α is much greater than ϵ, only the scheme that utilizes backward difference discretization of the first derivative converges to a solution in the prescribed tolerance.

In Fig. 1, the solution of the 2D convection-diffusion equation for various cases of ϵ and α is presented.

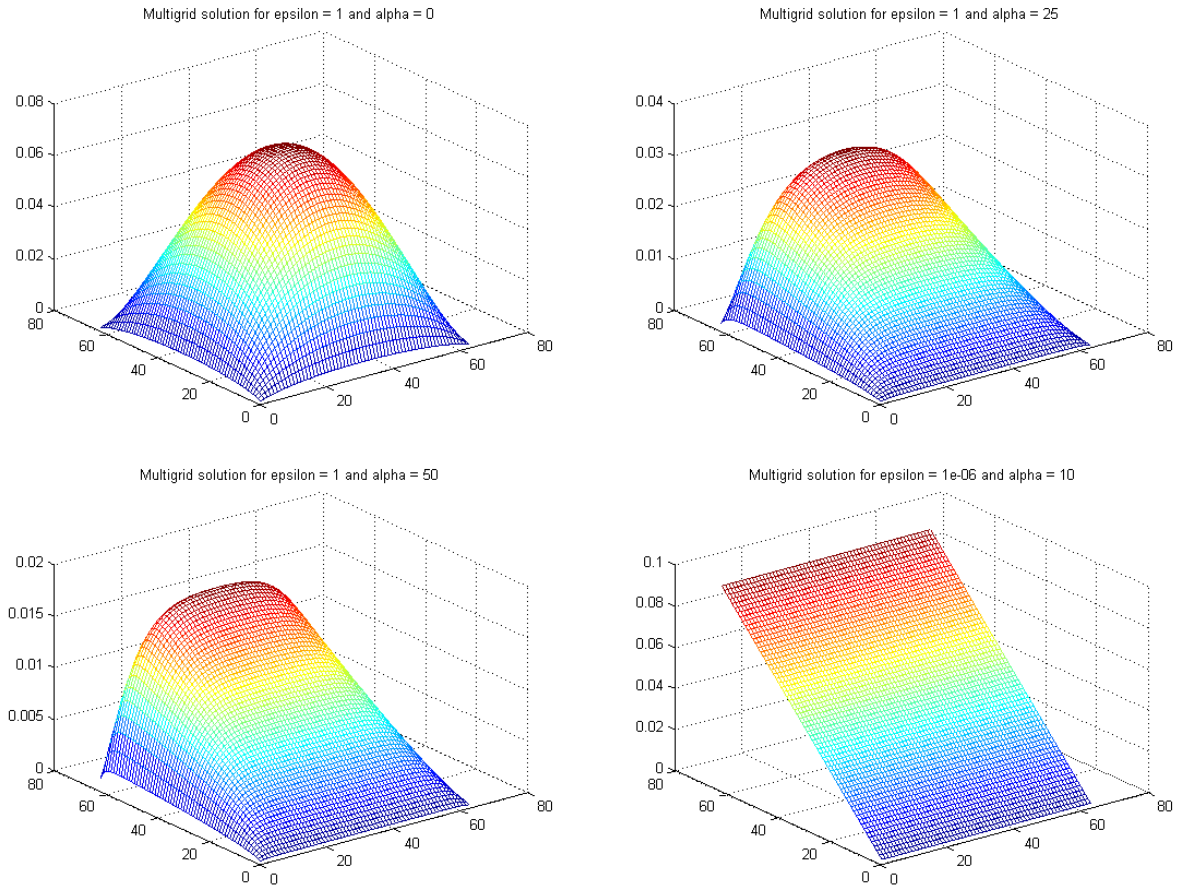


Figure 1: Solution of the 2D convection-diffusion equation for various cases of ϵ and α.

2 Helmholtz PDE - FEM - Krylov Solvers - Preconditioning

Let us consider the Helmholtz equation in two space variables:

$$\Delta u + k^2 u = 1, \quad (x, y) \in \Omega \equiv [-1, 1]^2,$$

where $k = 2\pi f/c$ is the wave number, and which is subject to the following boundary conditions:

$$u(x, y) = 0, \quad \{x = -1, \quad y \in [-1, 1]\} \bigcup \{y = -1, \quad x \in [-1, 1]\},$$

$$\frac{\partial u(x, y)}{\partial \eta} = 0, \quad \{x = 1, \quad y \in [-1, 1]\} \bigcup \{y = 1, \quad x \in [-1, 1]\}.$$

The Helmholtz equation should be discretized using the finite element method, and specifically utilizing bilinear elements, i.e. $\phi(x, y) = a + bx + cy + dxy$. Furthermore, the PDE should be solved for different values of the frequency f. The respective linear systems should be solved using the Conjugate Gradient, Bi-CGSTAB, as well as GMRES(m) method with m = 10, 20, 40, 80. The termination criterion for the Krylov subspace iterative methods should be the following: $||r_i|| < 10^{-10}||r_0||$. The convergence behavior of the Krylov methods should be examined with and without preconditioners. The preconditioners that should be tested are Jacobi, Gauss-Seidel as well as Symmetric Gauss-Seidel.

Solution

In order to solve the Helmholtz equation using the finite element method, we have to impose the weighted average of the equation on the domain Ω.

Thus, we can consider that:

$$\int_\Omega (\Delta u + k^2 u)v = \int_\Omega 1v \Rightarrow$$

$$\int_\Omega (\Delta u)v + k^2 \int_\Omega uv = \int_\Omega v. \tag{1}$$

Moreover, it is true that:

$$\nabla(v\nabla u) = \nabla v \nabla u + v\Delta u \Rightarrow$$

$$v\Delta u = \nabla(v\nabla u) - \nabla v \nabla u \Rightarrow$$

$$\int_\Omega (\Delta u)v = \int_\Omega \nabla(v\nabla u) - \int_\Omega \nabla u \nabla v. \tag{2}$$

Using the Gauss's theorem, the equation (2) can be rewritten as:

$$\int_\Omega (\Delta u)v = \int_{\partial\Omega} (v\nabla u)\cdot \hat{n} - \int_\Omega \nabla u \nabla v \Rightarrow$$

$$\int_\Omega (\Delta u)v = \int_{\partial\Omega} (\partial_n u)v - \int_\Omega \nabla u \nabla v. \tag{3}$$

Substituting the equation (3) in equation (1), it follows that:

$$-\int_\Omega \nabla u \nabla v + k^2 \int_\Omega uv = \int_\Omega v - \int_{\partial\Omega} (\partial_n u)v \Rightarrow$$

$$-\int_\Omega \nabla u \nabla v + k^2 \int_\Omega uv = \int_\Omega v - \int_{\Gamma_D} (\partial_n u)v - \int_{\Gamma_N} (\partial_n u)v. \tag{4}$$

In the problem statement, it is given that $\partial_n u = 0$ on Γ_N.

Hence, the equation (4) can be written equivalently as follows:

$$-\int_\Omega \nabla u \nabla v + k^2 \int_\Omega uv = \int_\Omega v - \int_{\Gamma_D} (\partial_n u)v. \tag{5}$$

Moreover, assuming that $v = 0$ on Γ_D, the equation (5) can be rewritten as:

$$-\int_\Omega \nabla u \nabla v + k^2 \int_\Omega uv = \int_\Omega v. \tag{6}$$

The *weak form* of the initial problem can now be stated as follows:

Find the function $u \in H^1(\Omega)$ such that:

$$\begin{cases} u = 0, \;\; u \in \Gamma_D \\ -\int_\Omega \nabla u \nabla v + k^2 \int_\Omega uv = \int_\Omega v, \;\; \forall v \in H^1_{\Gamma_D}(\Omega) \end{cases}.$$

At this point, in order to be precise, the following function spaces should be defined:

- $L^2(\Omega) = \left\{ f : \Omega \to \mathbb{R} \;\mid\; \int_\Omega |f|^2 < \infty \right\}$
- $H^1(\Omega) = \left\{ u \in L^2(\Omega) \;\mid\; \frac{\partial u}{\partial x}, \frac{\partial u}{\partial y} \in L^2(\Omega) \right\}, \;\; ||u||_{1,\Omega} = \left(\int_\Omega |\nabla u|^2 + \int_\Omega |u|^2 \right)^{1/2}$
- $H^1_{\Gamma_D}(\Omega) = \left\{ v \in H^1(\Omega) \;\mid\; v = 0, \;\; \forall v \in \Gamma_D \right\}$
- $C^1(\overline{\Omega}) = \left\{ u \in C(\overline{\Omega}) \;\mid\; \frac{\partial u}{\partial x}, \frac{\partial u}{\partial y} \in C(\overline{\Omega}) \right\}$
- $V_h = \left\{ u_h \in C(\overline{\Omega}) \;\mid\; u_h|_K \in \mathbb{P}_1, \;\; \forall K \in T_h \right\}$
- $V_h^{\Gamma_D} = V_h \cap H^1_{\Gamma_D}(\Omega) = \{ v_h \in V_h \;\mid\; v_h = 0, \;\; \forall v_h \in \Gamma_D \}$

Now, let us consider that we have discretized the domain Ω using non-overlapping finite elements (in this case linear square elements), the set of which is denoted T_h.

Moreover, let us define the basis functions of V_h as follows:

$$\phi_i \in V_h, \;\; \phi_i(p_j) = \delta_{ij} = \begin{cases} 1, & j = i \\ 0, & j \neq i \end{cases},$$

where p_j denotes a specific vertex of a finite element in T_h.

The approximate solution of the PDE is given by $u_h \in V_h$, which can be written as:

$$u_h(\cdot) = \sum_{j=1}^{N} u_h(p_j)\phi_j(\cdot) \Rightarrow$$

$$u_h = \sum_{j=1}^{N} u_j\phi_j \Rightarrow$$

$$u_h = \sum_{j\in Ind} u_j\phi_j + \sum_{j\in Dir} u_j\phi_j \Rightarrow$$

$$u_h = \sum_{j\in Ind} u_j\phi_j + \sum_{j\in Dir} 0\cdot\phi_j \Rightarrow$$

$$u_h = \sum_{j\in Ind} u_j\phi_j, \tag{7}$$

where N denotes the number of vertices of the finite elements, while Dir and Ind denote the lists containing the Dirichlet and non-Dirichlet (independent or free) vertices, respectively.

Furthermore, the test function $v_h \in V_h^{\Gamma_D}$ is given from the equation:

$$v_h = \sum_{j=1}^{N} v_j\phi_j \Rightarrow$$

$$v_h = \sum_{j\in Ind} v_j\phi_j.$$

The discrete form of equation (6) is the following:

$$-\int_\Omega \nabla u_h\cdot\nabla v_h + k^2\int_\Omega u_h\cdot v_h = \int_\Omega v_h,\ \ \forall v_h \in V_h^{\Gamma_D}. \tag{8}$$

Substituting, in equation (8), the u_h from equation (7) as well as $v_h = \phi_i \in V_h^{\Gamma_D},\ \ \forall i \in Ind$, yields:

$$\sum_{j\in Ind}\left(-\int_\Omega \nabla\phi_j\cdot\nabla\phi_i + k^2\int_\Omega \phi_j\cdot\phi_i\right)u_j = \int_\Omega \phi_i,\ \ i\in Ind.$$

The integrals over Ω can be decomposed as the sum of integrals over the different elements K, i.e.:

$$w_{ij} = \int_\Omega \nabla\phi_j\cdot\nabla\phi_i = \sum_K\int_K \nabla\phi_j\cdot\nabla\phi_i = \sum_K w_{ij}^K, \tag{9}$$

$$m_{ij} = \int_\Omega \phi_j\cdot\phi_i = \sum_K\int_K \phi_j\cdot\phi_i = \sum_K m_{ij}^K, \tag{10}$$

$$b_i = \int_\Omega \phi_i = \sum_K\int_K \phi_i = \sum_K b_i^K. \tag{11}$$

It should be mentioned, that in this case, we chose to discretize the domain Ω using bilinear finite elements. The function describing such elements is the following:

$$\phi(x, y) = a + bx + cy + dxy.$$

Now, let us consider four local basis functions on each square element. In order to do this, first, we need to assign a number to every vertex of each square K, i.e. p_1^K, p_2^K, p_3^K, p_4^K, following the lexicographic ordering, and then, we can define the functions N_1^K, N_2^K, N_3^K, $N_4^K \in \mathbb{Q}1$, which satisfy the equation:

$$N_a^K(p_b^K) = \delta_{a,b}, \quad a, b = 1, 2, 3, 4.$$

Furthermore, let us define the reference element $\hat{K}$ with the following vertices:

$$\hat{p}_1 = (-1, -1), \quad \hat{p}_2 = (1, -1), \quad \hat{p}_3 = (1, 1), \quad \hat{p}_4 = (-1, 1),$$

along with the respective local basis functions:

$$\hat{N}_1(\xi, \eta) = \frac{1}{4}(1 - \xi)(1 - \eta),$$

$$\hat{N}_2(\xi, \eta) = \frac{1}{4}(1 + \xi)(1 - \eta),$$

$$\hat{N}_3(\xi, \eta) = \frac{1}{4}(1 + \xi)(1 + \eta),$$

$$\hat{N}_4(\xi, \eta) = \frac{1}{4}(1 - \xi)(1 + \eta).$$

Next, in order to compute the integrals (9), (10), and (11), we will first compute the integrals on the reference element $\hat{K}$, and use a transformation between each physical element K and the reference element $\hat{K}$ so as to compute the required integrals.

Locally, on each physical element K, the integrals w_{ij}^K, m_{ij}^K and b_i^K can be rewritten as follows:

$$w_{n_a n_b}^K = \kappa_{a,b}^K = \int_K \nabla N_b^K \cdot \nabla N_a^K,$$

$$m_{n_a n_b}^K = \lambda_{a,b}^K = \int_K N_b^K \cdot N_a^K,$$

$$b_{n_a}^K = \mu_a^K = \int_K N_a^K,$$

$$a, b = 1, 2, 3, 4,$$

where n_x denotes the mapping from the local $x \in \{1, 2, 3, 4\}$ to the global n_x numbering of the vertices.

Now, we need to find a linear transformation based on which the reference element $\hat{K}$ could be transformed into any physical element K.

Supposing that we had chosen the triangle with vertices $\hat{p}_1$, $\hat{p}_2$ and $\hat{p}_4$, as a reference element, then the local basis functions would be the following:

$$\hat{N}_1'(\xi,\eta) = -\frac{1}{2}(\xi+\eta),$$

$$\hat{N}_2'(\xi,\eta) = \frac{1}{2}(1+\xi),$$

$$\hat{N}_4'(\xi,\eta) = \frac{1}{2}(1+\eta).$$

It should be noted, that a linear transformation that maps the aforementioned reference triangle onto the general triangle with vertices p_1^K, p_2^K and p_4^K, also maps the reference square with vertices $\hat{p}_1$, $\hat{p}_2$, $\hat{p}_3$, $\hat{p}_4$, onto the general square with vertices p_1^K, p_2^K, p_3^K, p_4^K. This happens due to the fact that in a linear transformation parallelism is preserved, and thus the vertex $\hat{p}_3$ is mapped implicitly onto the vertex p_3^K.

Such a linear transformation is the following:

$$\begin{bmatrix} x \\ y \end{bmatrix} = -\frac{1}{2}(\xi+\eta)\begin{bmatrix} x_1 \\ y_1 \end{bmatrix} + \frac{1}{2}(1+\xi)\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} + \frac{1}{2}(1+\eta)\begin{bmatrix} x_4 \\ y_4 \end{bmatrix} \Rightarrow$$

$$\begin{bmatrix} x \\ y \end{bmatrix} = \frac{1}{2}\begin{bmatrix} x_2-x_1 & x_4-x_1 \\ y_2-y_1 & y_4-y_1 \end{bmatrix}\begin{bmatrix} \xi \\ \eta \end{bmatrix} + \frac{1}{2}\begin{bmatrix} x_2+x_4 \\ y_2+y_4 \end{bmatrix}.$$

Based on the chain rule of differentiation, we have:

$$\frac{\partial \hat{N}}{\partial \xi} = \frac{\partial \hat{N}}{\partial x}\frac{\partial x}{\partial \xi} + \frac{\partial \hat{N}}{\partial y}\frac{\partial y}{\partial \xi},$$

$$\frac{\partial \hat{N}}{\partial \eta} = \frac{\partial \hat{N}}{\partial x}\frac{\partial x}{\partial \eta} + \frac{\partial \hat{N}}{\partial y}\frac{\partial y}{\partial \eta},$$

which can be equivalently rewritten in matrix form as follows:

$$\begin{bmatrix} \partial\hat{N}/\partial\xi \\ \partial\hat{N}/\partial\eta \end{bmatrix} = \begin{bmatrix} \partial x/\partial\xi & \partial y/\partial\xi \\ \partial x/\partial\eta & \partial y/\partial\eta \end{bmatrix}\begin{bmatrix} \partial\hat{N}/\partial x \\ \partial\hat{N}/\partial y \end{bmatrix} \Rightarrow$$

$$\begin{bmatrix} \partial\hat{N}/\partial x \\ \partial\hat{N}/\partial y \end{bmatrix} = \begin{bmatrix} \partial x/\partial\xi & \partial y/\partial\xi \\ \partial x/\partial\eta & \partial y/\partial\eta \end{bmatrix}^{-1}\begin{bmatrix} \partial\hat{N}/\partial\xi \\ \partial\hat{N}/\partial\eta \end{bmatrix} \Rightarrow$$

$$\begin{bmatrix} \partial\hat{N}/\partial x \\ \partial\hat{N}/\partial y \end{bmatrix} = \begin{bmatrix} (x_2-x_1)/2 & (y_2-y_1)/2 \\ (x_4-x_1)/2 & (y_4-y_1)/2 \end{bmatrix}^{-1}\begin{bmatrix} \partial\hat{N}/\partial\xi \\ \partial\hat{N}/\partial\eta \end{bmatrix}. \tag{12}$$

Let F be the following matrix:

$$F = \begin{bmatrix} (x_2-x_1)/2 & (y_2-y_1)/2 \\ (x_4-x_1)/2 & (y_4-y_1)/2 \end{bmatrix},$$

the determinant $|F|$ of which is given by:

$$|F| = \frac{1}{4}\left((x_2 - x_1)(y_4 - y_1) - (x_4 - x_1)(y_2 - y_1)\right).$$

Hence, the equation (12) can be rewritten as:

$$\begin{bmatrix} \partial\hat{N}/\partial x \\ \partial\hat{N}/\partial y \end{bmatrix} = \frac{1}{|F|}\begin{bmatrix} (y_4 - y_1)/2 & (y_1 - y_2)/2 \\ (x_1 - x_4)/2 & (x_2 - x_1)/2 \end{bmatrix}\begin{bmatrix} \partial\hat{N}/\partial\xi \\ \partial\hat{N}/\partial\eta \end{bmatrix}. \tag{13}$$

Utilizing the equation (13), as well as the local basis functions $\hat{N}_1$, $\hat{N}_2$, $\hat{N}_3$, $\hat{N}_4$, we can compute the values of the partial derivatives $\partial\hat{N}/\partial x$ and $\partial\hat{N}/\partial y$, which are required for the computation of the integrals $\kappa^K_{a,b}$.

$$\begin{bmatrix} \partial\hat{N}_1/\partial\xi \\ \partial\hat{N}_1/\partial\eta \end{bmatrix} = \frac{1}{4}\begin{bmatrix} \eta - 1 \\ \xi - 1 \end{bmatrix}$$

$$\begin{bmatrix} \partial\hat{N}_2/\partial\xi \\ \partial\hat{N}_2/\partial\eta \end{bmatrix} = \frac{1}{4}\begin{bmatrix} -\eta + 1 \\ -\xi - 1 \end{bmatrix}$$

$$\begin{bmatrix} \partial\hat{N}_3/\partial\xi \\ \partial\hat{N}_3/\partial\eta \end{bmatrix} = \frac{1}{4}\begin{bmatrix} \eta + 1 \\ \xi + 1 \end{bmatrix}$$

$$\begin{bmatrix} \partial\hat{N}_4/\partial\xi \\ \partial\hat{N}_4/\partial\eta \end{bmatrix} = \frac{1}{4}\begin{bmatrix} -\eta - 1 \\ -\xi + 1 \end{bmatrix}$$

Now, the integrals $\kappa^K_{a,b}$, $\lambda^K_{a,b}$ and μ^K_a, can be computed using the following transformation:

$$\iint_K f(x,y)dxdy = \iint_{\hat{K}} f(x(\xi,\eta), y(\xi,\eta))|J|d\xi d\eta,$$

$$|J| = |F|.$$

Thus, we have that:

$$\kappa^K_{a,b} = \int_K \nabla N^K_b \cdot \nabla N^K_a = |J| \int_{-1}^{1}\int_{-1}^{1} \nabla\hat{N}_b \cdot \nabla\hat{N}_a \;\; d\xi d\eta,$$

$$\lambda^K_{a,b} = \int_K N^K_b \cdot N^K_a = |J| \int_{-1}^{1}\int_{-1}^{1} \hat{N}_b \cdot \hat{N}_a \;\; d\xi d\eta,$$

$$\mu^K_a = \int_K N^K_a = |J| \int_{-1}^{1}\int_{-1}^{1} \hat{N}_a \;\; d\xi d\eta,$$

$$a, b = 1, 2, 3, 4.$$

The computation of the aforementioned integrals is performed using the MATLAB script integrals_qfem.m which is given in Appendix A1. Furthermore, the discretization of the domain Ω using bilinear (square)

finite elements is performed using the function qfem.m, while the assembly of the linear system is done using the function qfem_assemble.m. The overall solution of the Helmholtz equation as well as the comparative experiments between various solution techniques for the associated linear system are presented in the script fem_helmholtz.m.

Listing 3: qfem.m

```
function [h,ne,n,coo,con,bounds]=qfem(a,b,m)
h=(b-a)/m;        % mesh size
ne=m^2;           % number of elements (squares)
n=(m+1)^2;        % number of vertices
coo=zeros(n,2);   % coordinate matrix
con=zeros(ne,4); % connectivity matrix
t=(a:h:b)';
for i=1:(m+1)
    idx=(i-1)*(m+1)+1:i*(m+1);
    coo(idx,1)=t;
    coo(idx,2)=a+(i-1)*h;
end
idx=1;
for i=1:m
    for j=1:m
        con(idx,:)=[j j+1 j+m+2 j+m+1]+(i-1)*(m+1);
        idx=idx+1;
    end
end
south=1:m+1;
west=(m+2):(m+1):((m+1)^2-2*m-1);
north=south+(m+1)*m;
east=west+m;
bounds=[south north west east];
end
```

Listing 4: qfem_assemble.m

```
function [A,b]=qfem_assemble(coo,con,f)
n=size(coo,1);  % number of vertices
ne=size(con,1); % number of elements (squares)
c=3e8;
k=(2*pi*f)/c;
A=spalloc(n,n,9*n); % approximate number of nonzeros is used here
b=zeros(n,1);
W=zeros(4);
M=zeros(4);
f=zeros(4,1);
for e=1:ne
    x1=coo(con(e,1),1);
    x2=coo(con(e,2),1);
    x3=coo(con(e,3),1);
    x4=coo(con(e,4),1);
    y1=coo(con(e,1),2);
    y2=coo(con(e,2),2);
    y3=coo(con(e,3),2);
    y4=coo(con(e,4),2);
    y41=y4-y1;
    y12=y1-y2;
    x14=x1-x4;
    x21=x2-x1;
    J=0.25*((x2-x1)*(y4-y1)-(x4-x1)*(y2-y1));
    W(1,1)=(2*x14^2+3*x14*x21+2*x21^2+2*y12^2+3*y12*y41+2*y41^2)/(24*J);
    W(2,2)=(2*x14^2-3*x14*x21+2*x21^2+2*y12^2-3*y12*y41+2*y41^2)/(24*J);
    W(3,3)=(2*x14^2+3*x14*x21+2*x21^2+2*y12^2+3*y12*y41+2*y41^2)/(24*J);
```

```
    W(4,4)=(2*x14^2-3*x14*x21+2*x21^2+2*y12^2-3*y12*y41+2*y41^2)/(24*J);
    W(1,2)=-(2*x14^2-x21^2-y12^2+2*y41^2)/(24*J);
    W(1,3)=-(x14^2+3*x14*x21+x21^2+y12^2+3*y12*y41+y41^2)/(24*J);
    W(1,4)=(x14^2-2*x21^2-2*y12^2+y41^2)/(24*J);
    W(2,3)=(x14^2-2*x21^2-2*y12^2+y41^2)/(24*J);
    W(2,4)=-(x14^2-3*x14*x21+x21^2+y12^2-3*y12*y41+y41^2)/(24*J);
    W(3,4)=-(2*x14^2-x21^2-y12^2+2*y41^2)/(24*J);
    W(2,1)=W(1,2); W(3,1)=W(1,3); W(4,1)=W(1,4);
    W(3,2)=W(2,3); W(4,2)=W(2,4);
    W(4,3)=W(3,4);
    M(1,1)=(4*J)/9;
    M(2,2)=(4*J)/9;
    M(3,3)=(4*J)/9;
    M(4,4)=(4*J)/9;
    M(1,2)=(2*J)/9;
    M(1,3)=J/9;
    M(1,4)=(2*J)/9;
    M(2,3)=(2*J)/9;
    M(2,4)=J/9;
    M(3,4)=(2*J)/9;
    M(2,1)=M(1,2); M(3,1)=M(1,3); M(4,1)=M(1,4);
    M(3,2)=M(2,3); M(4,2)=M(2,4);
    M(4,3)=M(3,4);
    f(1)=J;
    f(2)=J;
    f(3)=J;
    f(4)=J;
    for i=1:4
        for j=1:4
            A(con(e,i),con(e,j))=A(con(e,i),con(e,j))-W(i,j)+(k^2)*M(i,j);
        end
        b(con(e,i))=b(con(e,i))+f(i);
    end
end
end
```

Listing 5: fem_helmholtz.m

```
clear;clc;close all;
a=-1;
b=1;
m=32; % number of subintervals per dimension
f=5e8; % if f=0 then Helmholtz -> Poisson
prec=1; % enable/disable preconditioning
tol=1e-10;
Nmax=1000;
restart=[10 20 40 80];
fem_type=1; % choose 0 for triangles or 1 for squares
if (fem_type)
    [h,ne,n,coo,con,bounds]=qfem(a,b,m);
    [A,b]=qfem_assemble(coo,con,f);
else
    [h,ne,n,coo,con,bounds]=tfem(a,b,m);
    [A,b]=tfem_assemble(coo,con,f);
end
bounds=[1:(m+1) (m+2):(m+1):((m+1)^2-m)]; % Dirichlet(0) on NE
in=setdiff(1:n,bounds);
xx=zeros(n,1);
A(bounds,:)=[];
A(:,bounds)=[];
b(bounds)=[];
ex=A\b; % Direct solution
xx(in)=ex;
```

```
xx(bounds)=0;
figure, trimesh(con,coo(:,1),coo(:,2),xx), % plot solution
title(sprintf('Solution of Helmholtz PDE for f = %e',f));

fprintf(' ---------- Matrix statistics ---------- \n');
fprintf('length(A) = %d\n',length(A));
fprintf('condest(A) = %e\n',condest(A));
eigvals=eig(A); mineig=min(eigvals); maxeig=max(eigvals);
% figure, stem(1:length(A),eigvals),
% title(['Eigenvalues of A are in D=[' ...
%     num2str(mineig) ',' num2str(maxeig) ']']);

% Testing Krylov Solvers (cg, bicgstab, gmres) and
% Preconditioners (Jacobi, Gauss-Seidel, Symmetric Gauss-Seidel)
D=diag(diag(A));
L=tril(A);

if ((maxeig*mineig)>0) % check for same sign
    fprintf('\n ---------- cg convergence ---------- \n');
    x=pcg(-A,-b,tol,Nmax); % A is negative definite
    if (prec)
        x=pcg(-A,-b,tol,Nmax,@(y) -D\y);
        x=pcg(-A,-b,tol,Nmax,@(y) -L\y);
        x=pcg(-A,-b,tol,Nmax,@(y) -L\(D*(L'\y)));
    end
end

fprintf('\n ---------- bicgstab convergence ---------- \n');
Nmax2=ceil(Nmax/2);
x=bicgstab(A,b,tol,Nmax2);
if (prec)
    x=bicgstab(A,b,tol,Nmax2,@(y) D\y);
    x=bicgstab(A,b,tol,Nmax2,@(y) L\y);
    x=bicgstab(A,b,tol,Nmax2,@(y) L\(D*(L'\y)));
end

for i=1:length(restart)
    fprintf('\n ---------- gmres(%d) convergence ---------- \n',restart(i));
    Nmax3=ceil(Nmax/restart(i));
    x=gmres(A,b,restart(i),tol,Nmax3);
    if (prec)
        x=gmres(A,b,restart(i),tol,Nmax3,@(y) D\y);
        x=gmres(A,b,restart(i),tol,Nmax3,@(y) L\y);
        x=gmres(A,b,restart(i),tol,Nmax3,@(y) L\(D*(L'\y)));
    end
end
```

In Tables 3 - 6, the convergence behavior of certain Krylov subspace iterative methods, i.e. the Conjugate Gradient, Bi-CGSTAB and GMRES(m) methods, with and without preconditioners, for the 2D Helmholtz problem and for various values of the frequency f, is presented. It should be noted, that for the discretization of the domain Ω, the number of subintervals per dimension (and hence the number of square elements per dimension) was set to $m = 32$, while the prescribed tolerance for the solution of the linear system was chosen to be 10^{-10}. In cases when a Krylov method failed to converge to an accurate solution within the maximum number of iterations, the achieved residual is given in the respective Table, colored in red and following the letter "N" meaning "No Convergence".

It can be observed, that the coefficient matrix A is always symmetric, and when $f = 0$, it is also negative definite. Therefore, in this case the matrix $-A$ is positive definite. So when $f = 0$, it is possible to solve the linear system $Ax = b$ using the Conjugate Gradient method, by first transforming it into the equivalent linear system $-Ax = -b$, the coefficient matrix of which is symmetric positive definite (SPD). In any other case, i.e. when $f \neq 0$, the coefficient matrix is simply symmetric, and thus only Bi-CGSTAB and GMRES(m)

methods can be used.

Furthermore, it can be seen, that increasing the value of the frequency f, leads to a decrease in the magnitude of the diagonal elements of the matrix A, as opposed to an increase in the magnitude of the off-diagonal elements, making the chosen preconditioners less effective.

Overall, as expected, in all cases the most effective preconditioner, among the ones chosen, is the symmetric Gauss-Seidel. Concerning the GMRES(m) method, increasing the restart parameter, and as a result the dimension of the Krylov subspace, leads to improved convergence behavior of the method.

$f = 0$	No prec.	Jacobi	G-S	Sym. G-S
CG	96	93	N1	50
Bi-CGSTAB	64	62.5	61	35.5
GMRES(10)	N4.0e-10	68(9)	40(7)	20(6)
GMRES(20)	31(7)	26(14)	14(14)	5(7)
GMRES(40)	7(26)	6(29)	4(13)	2(15)
GMRES(80)	2(21)	2(41)	1(80)	1(49)

Table 3: Convergence behavior of the Conjugate Gradient, Bi-CGSTAB and GMRES(m) methods, with and without preconditioners, for the Helmholtz problem with $f = 0$.

$f = 10^8$	No prec.	Jacobi	G-S	Sym. G-S
Bi-CGSTAB	71.5	68.5	77.5	42.5
GMRES(10)	N1.3e-01	N9.2e-02	N2.8e-09	28(4)
GMRES(20)	N1.1e-10	N3.5e-10	31(2)	8(1)
GMRES(40)	11(15)	9(38)	6(8)	2(36)
GMRES(80)	2(33)	2(66)	2(39)	1(56)

Table 4: Convergence behavior of the Bi-CGSTAB and GMRES(m) methods, with and without preconditioners, for the Helmholtz problem with $f = 10^8$.

$f = 3 \cdot 10^8$	No prec.	Jacobi	G-S	Sym. G-S
Bi-CGSTAB	275.5	237.5	251.5	113.5
GMRES(10)	N8.8e-03	N6.2e-04	N2.3e-07	87(3)
GMRES(20)	N7.9e-04	N4.5e-07	N1.8e-09	34(9)
GMRES(40)	N5.4e-07	N2.6e-08	20(12)	8(25)
GMRES(80)	7(9)	7(8)	3(74)	2(2)

Table 5: Convergence behavior of the Bi-CGSTAB and GMRES(m) methods, with and without preconditioners, for the Helmholtz problem with $f = 3 \cdot 10^8$.

$f = 5 \cdot 10^8$	No prec.	Jacobi	G-S	Sym. G-S
Bi-CGSTAB	387.5	329.5	497.5	282.5
GMRES(10)	N7.4e-02	N6.5e-02	N5.0e-02	N1.9e-02
GMRES(20)	N6.1e-02	N5.3e-02	N5.6e-02	N6.2e-03
GMRES(40)	N3.7e-02	N2.8e-02	N7.8e-03	N1.1e-03
GMRES(80)	N7.6e-03	N4.3e-03	N1.6e-05	N2.7e-09

Table 6: Convergence behavior of the Bi-CGSTAB and GMRES(m) methods, with and without preconditioners, for the Helmholtz problem with $f = 5 \cdot 10^8$.

In Fig. 2, the solution of the 2D Helmholtz equation for various values of the frequency f is presented.

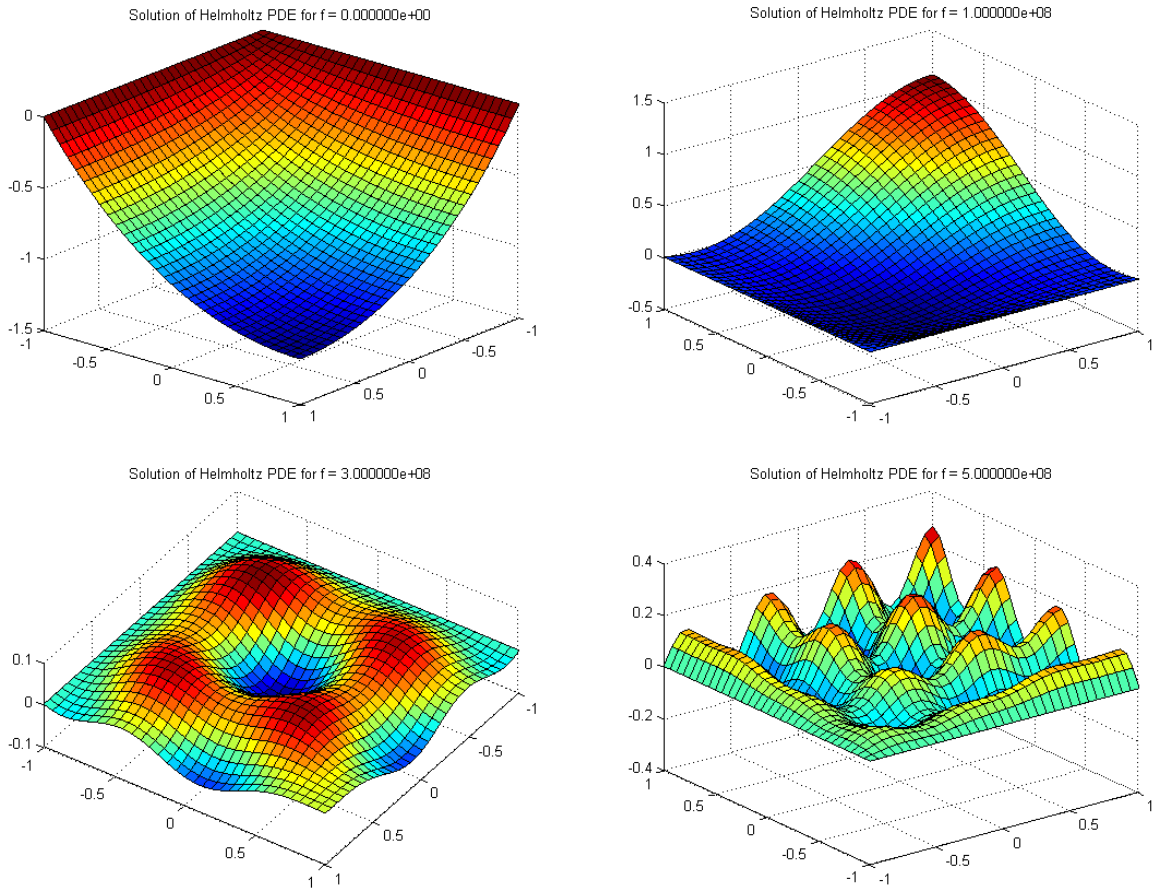


Figure 2: Solution of the 2D Helmholtz equation for various values of the frequency f.

3 Laplace PDE - Meshless Method

Let us consider the Laplace equation in two space variables:

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0, \quad 0 < x, y < 1,$$

which is subject to the following boundary conditions:

$$u(x, 1) = e^x cos(1), \quad 0 \leq x \leq 1,$$

$$u(0, y) = cos(y), \quad 0 \leq y \leq 1,$$

$$u(1, y) = ecos(y), \quad 0 \leq y \leq 1,$$

$$\frac{\partial u}{\partial y}(x, 0) = 0, \quad 0 \leq x \leq 1.$$

Solve the above PDE using meshless method and compute the associated approximation error.

The analytical solution is given by $u(x, y) = e^x cos(y)$.

Solution

A function can be written as a weighted sum (linear combination) of the basis functions that span the function space in which it exists. Therefore, assuming that we have chosen to express the solution u of the given PDE, in terms of radial basis functions, it follows that:

$$u = \sum_{j=1}^{N} w_j \phi_{ij}, \tag{1}$$

where $\phi_{ij}, \quad j = 1, 2, ..., N$, denote the radial basis functions spanning the function space in which u lies, and $w_j, \quad j = 1, 2, ..., N$, denote the associated weighting terms.

Substituting the equation (1) into the given PDE, we have that:

$$\Delta u = 0 \Rightarrow$$

$$\Delta \left(\sum_{j=1}^{N} w_j \phi_{ij} \right) = 0 \Rightarrow$$

$$\sum_{j=1}^{N} \left(\Delta \phi_{ij} \right) w_j = 0, \quad i = 1, 2, ..., N. \tag{2}$$

The application of the equation (2) in N uniformly or non-uniformly distributed points in the domain Ω, yields a linear system with N equations and N unknowns. Hence, the weights $w_j, \quad j = 1, 2, .., N$, can be

computed by solving the linear system, and after that it will be possible to construct an approximate solution of the given PDE using the equation (1).

Assuming that we have chosen a radial basis function of the form:

$$\phi(r_{ij}) = \phi_{ij} = \sqrt{r_{ij}^2 + c^2}, \tag{3}$$

where:

$$r_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2},$$

and c an arbitrary constant, known as the *shape parameter*, the following partial derivatives can be computed:

$$\frac{\partial \phi_{ij}}{\partial x_i} = \frac{x_i - x_j}{\sqrt{r_{ij}^2 + c^2}}, \tag{4}$$

$$\frac{\partial \phi_{ij}}{\partial y_i} = \frac{y_i - y_j}{\sqrt{r_{ij}^2 + c^2}}, \tag{5}$$

$$\frac{\partial^2 \phi_{ij}}{\partial x_i^2} = \frac{(y_i - y_j)^2 + c^2}{(r_{ij}^2 + c^2)^{3/2}}, \tag{6}$$

$$\frac{\partial^2 \phi_{ij}}{\partial y_i^2} = \frac{(x_i - x_j)^2 + c^2}{(r_{ij}^2 + c^2)^{3/2}}. \tag{7}$$

Utilizing the equation (2) as well as the equations (4)-(7), and imposing the boundary conditions on the problem statement, leads to the following equations that define the linear system that needs to be solved in order to compute the weights $w_j, \;\; j = 1, 2, ..., N$:

$$\sum_{j=1}^{N} \left(\frac{\partial^2 \phi_{ij}}{\partial x_i^2} + \frac{\partial^2 \phi_{ij}}{\partial y_i^2} \right) w_j = 0, \;\; \forall i \in Inner \; Nodes, \tag{8}$$

$$\sum_{j=1}^{N} \left(\phi_{ij} \right) w_j = e^{x_i} cos(y_i), \;\; \forall i \in Dirichlet \; Nodes, \tag{9}$$

$$\sum_{j=1}^{N} \left(\frac{\partial \phi_{ij}}{\partial y_i} \right) w_j = 0, \;\; \forall i \in Neumann \; Nodes. \tag{10}$$

The MATLAB script that was implemented in order to solve the 2D Laplace equation subject to the boundary conditions described in the problem statement, using the meshless method is given below. It should be noted, that the shape parameter c was set to 1.

This specific parametrization of the meshless method led to the following error: $2.8918 \cdot 10^{-4}$.

In Fig. 3, the grid point distribution (top) as well as the exact and approximate solutions of the 2D Laplace equation (bottom left and right) are presented.

Listing 6: meshless2.m

```
% Lu=0 (2D) on a square domain [0,1]x[0,1]
% neumann on south boundary
% dirichlet on the other boundaries
clear;clc;close all;
rng(0); % used for reproducibility
u=@(x,y) exp(x).*cos(y);
phi=@(r,c) sqrt(r.^2+c^2);
c=1; % shape parameter
a=0;
b=1;
nx=25;
h=1/(nx-1);
xo=[a:h:b a:h:b repmat(a,1,nx-2) repmat(b,1,nx-2)]';
yo=[repmat(a,1,nx) repmat(b,1,nx) a+h:h:b-h a+h:h:b-h]';
% [xi,yi]=meshgrid((a+h):h:(b-h)); % inner points
% xi=xi(:);
% yi=yi(:);
xiyi=rand((nx-4)*nx,2);
xiyi(xiyi(:,1)==a,:)=[];
xiyi(xiyi(:,1)==b,:)=[];
xiyi(xiyi(:,2)==a,:)=[];
xiyi(xiyi(:,2)==b,:)=[];
xi=xiyi(:,1); yi=xiyi(:,2);
figure, plot(xo,yo,'r*',xi,yi,'*'), grid
x=[xo;xi];
y=[yo;yi];
n=length(x);
no=length(xo);
r=zeros(n);
dphix=zeros(n);
dphiy=zeros(n);
dphi2x=zeros(n);
dphi2y=zeros(n);
for i=1:n
    j=setdiff(1:n,i);
    r(i,j)=sqrt( (x(i)-x(j)).^2 + (y(i)-y(j)).^2 )';
end
j=1:n;
for i=1:n
    dphix(i,j)=(x(i)-x(j))./(sqrt(r(i,j).^2+c^2))';
    dphiy(i,j)=(y(i)-y(j))./(sqrt(r(i,j).^2+c^2))';
    dphi2x(i,j)=((y(i)-y(j)).^2+c^2)./((r(i,j).^2+c^2).^(3/2))';
    dphi2y(i,j)=((x(i)-x(j)).^2+c^2)./((r(i,j).^2+c^2).^(3/2))';
end
A=dphi2x+dphi2y;
b=zeros(n,1);
% Dirichlet Boundaries
A(1:no,:)=phi(r(1:no,:),c);
b(1:no)=u(x(1:no),y(1:no));
% Neumann Boundaries (overwrite south boundary)
A(1:nx,:)=dphiy(1:nx,:);
b(1:nx)=zeros(nx,1);
% Solve
w=A\b;
xx=phi(r,c)*w;
exact=u(x,y);
error=norm(xx-exact)/norm(exact)
% figure, imagesc(A);
figure, plot3(x,y,exact,'*'), grid, title('Exact solution');
figure, plot3(x,y,xx,'*'), grid, title('Meshless solution');
% figure, plot3(x(1:no),y(1:no),xx(1:no),'r*', ...
%               x(no+1:end),y(no+1:end),xx(no+1:end),'b*')
```

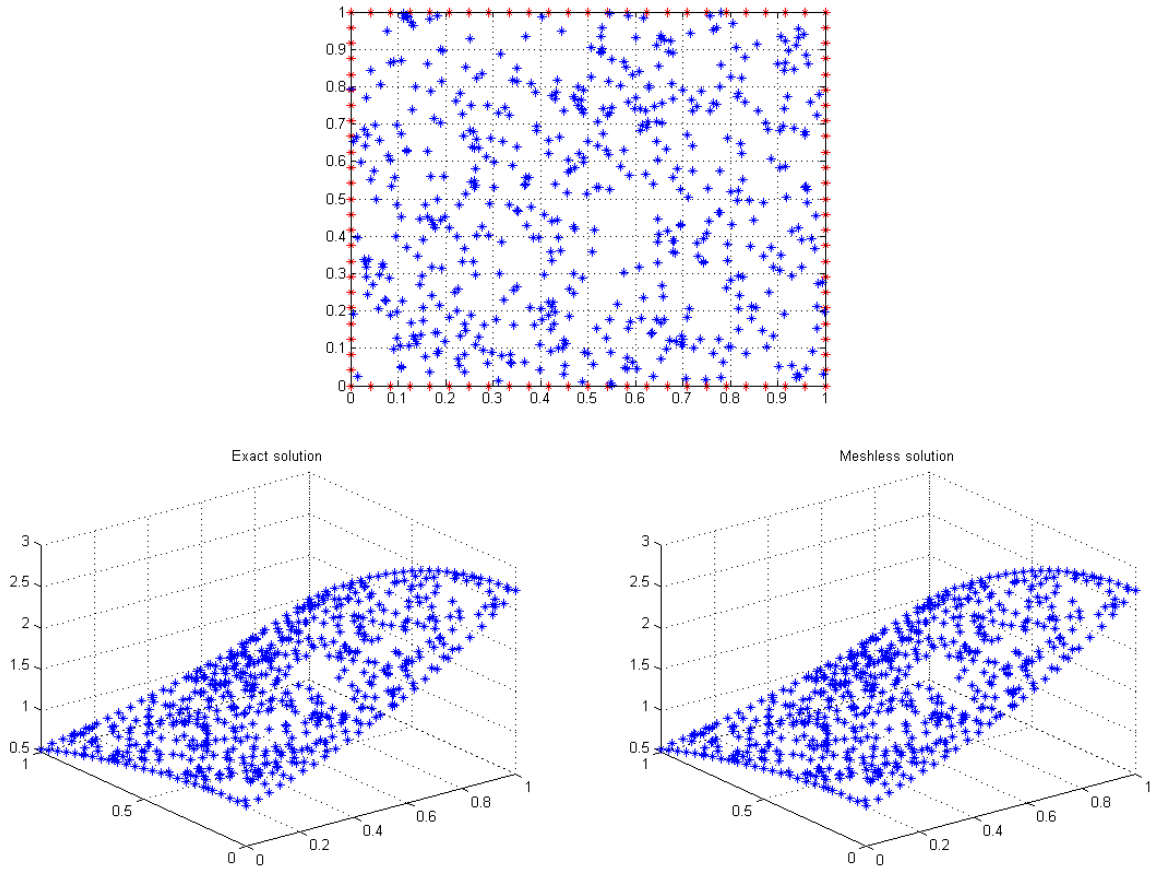


Figure 3: Grid point distribution (top) as well as the exact and approximate solutions of the 2D Laplace equation (bottom left and right).

4 Poisson PDE - FEM - Domain Decomposition - Preconditioning

Let us consider the following partial differential equation:

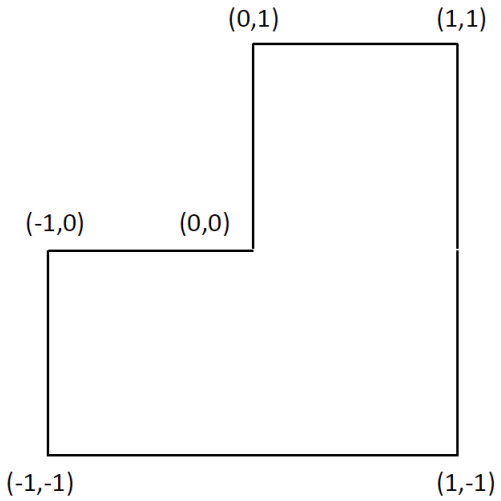
$$\Delta u = 1, \quad (x, y) \in \Omega,$$

subject to the Dirchlet boundary conditions:

$$u(x, y) = 0, \quad (x, y) \in \partial\Omega.$$

Solve the PDE using the finite element method, and specifically using linear elements (triangles) and bilinear elements (squares).

The domain Ω is depicted below:



The linear system should be solved using the Schur complement method. The Shur complement system should be solved iteratively, using a preconditioned Krylov subspace method. The preconditioners that should be tested should be based on incomplete LU as well as Sparse Approximate Inverses.

Solution

The discretization of the PDE using the finite element method is performed as in Section 2. It can be observed, that setting $k = 0$ in the Helmholtz equation, yields the Poisson equation. Hence, everything that is needed concerning bilinear elements (squares) has already been analyzed and computed. We only need to compute the values of the integrals that are associated with the linear finite elements (triangles).

Supposing that we had chosen the triangle with vertices:

$$\hat{p}_1 = (-1, -1), \quad \hat{p}_2 = (1, -1), \quad \hat{p}_4 = (-1, 1),$$

as a reference element $\hat{K}$, along with the following local basis functions:

$$\hat{N}_1(\xi,\eta) = -\frac{1}{2}(\xi+\eta),$$

$$\hat{N}_2(\xi,\eta) = \frac{1}{2}(1+\xi),$$

$$\hat{N}_4(\xi,\eta) = \frac{1}{2}(1+\eta),$$

then, the linear transformation mapping the reference triangle $\hat{K}$ on a general triangle K, would be exactly the same with the one utilized in Section 2.

Therefore, we have that:

$$\begin{bmatrix} \partial\hat{N}/\partial x \\ \partial\hat{N}/\partial y \end{bmatrix} = \frac{1}{|F|}\begin{bmatrix} (y_4-y_1)/2 & (y_1-y_2)/2 \\ (x_1-x_4)/2 & (x_2-x_1)/2 \end{bmatrix}\begin{bmatrix} \partial\hat{N}/\partial\xi \\ \partial\hat{N}/\partial\eta \end{bmatrix} \tag{1}$$

Utilizing the equation (1), as well as the local basis functions $\hat{N}_1$, $\hat{N}_2$, $\hat{N}_4$, it is possible to compute the values of the partial derivatives $\partial\hat{N}/\partial x$ and $\partial\hat{N}/\partial y$, which are required for computing the integrals $\kappa^K_{a,b}$.

$$\begin{bmatrix} \partial\hat{N}_1/\partial\xi \\ \partial\hat{N}_1/\partial\eta \end{bmatrix} = \frac{1}{2}\begin{bmatrix} -1 \\ -1 \end{bmatrix}, \tag{2}$$

$$\begin{bmatrix} \partial\hat{N}_2/\partial\xi \\ \partial\hat{N}_2/\partial\eta \end{bmatrix} = \frac{1}{2}\begin{bmatrix} 1 \\ 0 \end{bmatrix}, \tag{3}$$

$$\begin{bmatrix} \partial\hat{N}_4/\partial\xi \\ \partial\hat{N}_4/\partial\eta \end{bmatrix} = \frac{1}{2}\begin{bmatrix} 0 \\ 1 \end{bmatrix}. \tag{4}$$

Now, the integrals $\kappa^K_{a,b}$, $\lambda^K_{a,b}$ and μ^K_a, can be computed using the following transformation:

$$\iint_K f(x,y)dxdy = \iint_{\hat{K}} f(x(\xi,\eta),y(\xi,\eta))|J|d\xi d\eta,$$

$$|J| = |F|.$$

Thus, we have that:

$$\kappa^K_{a,b} = \int_K \nabla N^K_b \cdot \nabla N^K_a = |J|\int_{-1}^{1}\int_{-1}^{1} \nabla\hat{N}_b \cdot \nabla\hat{N}_a \;\; d\xi d\eta,$$

$$\lambda^K_{a,b} = \int_K N^K_b \cdot N^K_a = |J|\int_{-1}^{1}\int_{-1}^{1} \hat{N}_b \cdot \hat{N}_a \;\; d\xi d\eta,$$

$$\mu^K_a = \int_K N^K_a = |J|\int_{-1}^{1}\int_{-1}^{1} \hat{N}_a \;\; d\xi d\eta,$$

$$a,b = 1,2,4.$$

The computation of the aforementioned integrals is performed using the MATLAB script integrals_tfem.m which is given in Appendix A2. Furthermore, similarly to Section 2, the discretization of the domain Ω using linear (triangular) finite elements is performed using the function tfem.m, while the assembly of the linear system is done using the function tfem_assemble.m. It should be mentioned, that all implementations are generic so that the solution of the Helmholtz equation is also supported. We simply need to set $k = 0$ when solving the Poisson equation.

Listing 7: tfem.m

```
function [h,ne,n,coo,con,bounds]=tfem(a,b,m)
h=(b-a)/m;        % mesh size
ne=2*m^2;         % number of elements (triangles)
n=(m+1)^2;        % number of vertices
coo=zeros(n,2);   % coordinate matrix
con=zeros(ne,3); % connectivity matrix
t=(a:h:b)';
for i=1:(m+1)
    idx=(i-1)*(m+1)+1:i*(m+1);
    coo(idx,1)=t;
    coo(idx,2)=a+(i-1)*h;
end
idx=1;
for i=1:m
    for j=1:m
        con(idx,:)=[j j+1 j+m+1]+(i-1)*(m+1);
        con(idx+1,:)=[j+m+2 j+m+1 j+1]+(i-1)*(m+1);
        idx=idx+2;
    end
end
south=1:m+1;
west=(m+2):(m+1):((m+1)^2-2*m-1);
north=south+(m+1)*m;
east=west+m;
bounds=[south north west east];
end
```

Listing 8: tfem_assemble.m

```
function [A,b]=tfem_assemble(coo,con,f)
n=size(coo,1); % number of vertices
ne=size(con,1); % number of elements (triangles)
c=3e8;
k=(2*pi*f)/c;
A=spalloc(n,n,7*n); % approximate number of nonzeros is used here
b=zeros(n,1);
W=zeros(3);
M=zeros(3);
f=zeros(3,1);
for e=1:ne
    x1=coo(con(e,1),1);
    x2=coo(con(e,2),1);
    x4=coo(con(e,3),1);
    y1=coo(con(e,1),2);
    y2=coo(con(e,2),2);
    y4=coo(con(e,3),2);
    y41=y4-y1;
    y12=y1-y2;
    x14=x1-x4;
    x21=x2-x1;
    J=0.25*((x2-x1)*(y4-y1)-(x4-x1)*(y2-y1));
    W(1,1)=(x14^2 + 2*x14*x21 + x21^2 + y12^2 + 2*y12*y41 + y41^2)/(8*J);
```

```
    W(2,2)=(x14^2 + y41^2)/(8*J);
    W(3,3)=(x21^2 + y12^2)/(8*J);
    W(1,2)=-(x14^2 + x21*x14 + y41^2 + y12*y41)/(8*J);
    W(1,3)=-(x21^2 + x14*x21 + y12^2 + y41*y12)/(8*J);
    W(2,3)=(x14*x21 + y12*y41)/(8*J);
    W(2,1)=W(1,2); W(3,1)=W(1,3);
    W(3,2)=W(2,3);
    M(1,1)=J/3;
    M(2,2)=J/3;
    M(3,3)=J/3;
    M(1,2)=J/6;
    M(1,3)=J/6;
    M(2,3)=J/6;
    M(2,1)=M(1,2); M(3,1)=M(1,3);
    M(3,2)=M(2,3);
    f(1)=(2*J)/3;
    f(2)=(2*J)/3;
    f(3)=(2*J)/3;
    for i=1:3
        for j=1:3
            A(con(e,i),con(e,j))=A(con(e,i),con(e,j))-W(i,j)+(k^2)*M(i,j);
        end
        b(con(e,i))=b(con(e,i))+f(i);
    end
end
end
```

In order to solve the 2D Poisson equation on the L-shaped domain Ω, we follow the steps listed below:

- First, we construct a square domain discretized with either square or triangular finite elements, using either the function qfem.m or the function tfem.m, respectively.
- Next, using the function lshape_fem.m (which is given in Appendix B), we compose the L-shaped domain Ω, by utilizing the coordinate and connectivity matrices defining the discretized square domain, which was constructed in the previous step.
- Then, we assemble the respective coefficient matrix A as well as the right-hand-side vector b, using either the function qfem_assemble.m or the function tfem_assemble.m.
- Finally, we decompose the domain Ω into subdomains based on the coordinates of the elements' vertices, using the k-means algorithm, and then we solve the linear system using the Schur complement method.

It should be stated, that the Schur complement system is solved iteratively, using either preconditioned Bi-CGSTAB or preconditioned GMRES(20) method. The preconditioners that are tested are the iLU(0) (MATLAB implementation), as well as the SPAI method, which is implemented as follows:

Listing 9: make_spai.m

```
function M=make_spai(A,lfill)
n=length(A);
I=speye(n);
G=A^lfill;
M=G;
for i=1:n
    ind=find(G(i,:));
    [k,~]=find(A(:,ind));
    k=unique(k);
    AA=A(k,ind);
    M(ind,i)=(AA'*AA)\(AA'*I(k,i));
end
end
```

The MATLAB script that was implemented in order to solve the 2D Poisson equation using the aforementioned methodology, is presented below.

Listing 10: lshape_ddm2.m

```
clear;clc;close all;
rng(0); % used for reproducibility
a=-1;
b=0;
m=64; % number of subintervals per dimension (per square domain)
fem_type=0; % 0 for triangle and 1 for square
if (fem_type)
    [h,ne,n,coo1,con1,bounds1]=qfem(a,b,m);
else
    [h,ne,n,coo1,con1,bounds1]=tfem(a,b,m);
end
[coo,con,bounds,~,~]=lshape_fem(a,b,m,coo1,con1);

f=0; % Helmholtz -> Poisson
if (fem_type)
    [A,b]=qfem_assemble(coo,con,f);
else
    [A,b]=tfem_assemble(coo,con,f);
end
disp('Matrix Assembly - Done');

% Domain Decomposition
ndoms=10;
map=kmeans(coo,ndoms);
disp('kmeans - Done');
clr=['b' 'r' 'g' 'm' 'c'];
while(length(clr)<ndoms)
    clr=[clr clr];
end
In=cell(ndoms,1);
Out=cell(ndoms,1);
G=abs(A)+abs(A)';
figure, hold on,
for i=1:ndoms
    dpnts=find(map==i);
    plot(coo(dpnts,1),coo(dpnts,2),strcat(clr(i),'*'));
    for jj=1:length(dpnts)
        j=dpnts(jj);
        nbr=setdiff(find(G(:,j)),j);
        if (any(map(nbr)~=i))
            Out{i}=[Out{i} j];
        else
            In{i}=[In{i} j];
        end
    end
    In{i}=setdiff(In{i},bounds);
    Out{i}=setdiff(Out{i},bounds);
end
hold off;

in=horzcat(In{:});
out=horzcat(Out{:});
p=[in out];
disp('Data Partitioning - Done');

% Solve by Schur Complement Method
xx=zeros(n,1);
% S=A(out,out)-A(out,in)*(A(in,in)\A(in,out));
% g=b(out)-A(out,in)*(A(in,in)\b(in));
```

```
S=A(out,out);
g=b(out);
for i=1:ndoms
    S=S-A(out,In{i})*(A(In{i},In{i})\A(In{i},out));
    g=g-A(out,In{i})*(A(In{i},In{i})\b(In{i}));
end
% xout=S\g;
[L,U]=ilu(S);
lfill=1;
M=make_spai(S,lfill);
xout=bicgstab(S,g,1e-10,500);
xout=bicgstab(S,g,1e-10,500,@(y) U\(L\y));
xout=bicgstab(S,g,1e-10,500,@(y) M*y);
xout=gmres(S,g,20,1e-10,100);
xout=gmres(S,g,20,1e-10,100,@(y) U\(L\y));
xout=gmres(S,g,20,1e-10,100,@(y) M*y);
% xin=A(in,in)\(b(in)-A(in,out)*xout);
for i=1:ndoms
    xx(In{i})=A(In{i},In{i})\(b(In{i})-A(In{i},out)*xout);
end
% xx(in)=xin;
xx(out)=xout;
xx(bounds)=0;
figure, trimesh(con,coo(:,1),coo(:,2),xx);
```

In Tables 7 and 8, the convergence behavior of the Bi-CGSTAB and GMRES(20) methods in conjunction with the iLU(0) and SPAI preconditioners is presented, for the solution of the Schur complement system arising from the discretization of the 2D Poisson equation using triangular (tfem) and square (qfem) finite elements, respectively. It should be mentioned, that in both cases (linear and bilinear finite element discretization), the order of the coefficient matrix of the global linear system was equal to $12,033$, while the L-shaped domain Ω was decomposed geometrically into 10 subdomains. Moreover, the prescribed tolerance for the Krylov method was set to 10^{-10}.

tfem	No prec.	iLU(0)	SPAI
Bi-CGSTAB	99	27.5	52
GMRES(20)	14(5)	3(12)	6(5)

Table 7: Convergence behavior of the Bi-CGSTAB and GMRES(20) methods in conjunction with the iLU(0) and SPAI preconditioners for the solution of the Schur complement system arising from the discretization of the 2D Poisson equation using *triangular* finite elements.

qfem	No prec.	iLU(0)	SPAI
Bi-CGSTAB	78	23.5	46.5
GMRES(20)	11(4)	3(3)	5(10)

Table 8: Convergence behavior of the Bi-CGSTAB and GMRES(20) methods in conjunction with the iLU(0) and SPAI preconditioners for the solution of the Schur complement system arising from the discretization of the 2D Poisson equation using *square* finite elements.

It can be observed, that the iLU(0) preconditioner leads to improved convergence behavior of the Bi-CGSTAB and GMRES(20) methods, compared to the SPAI preconditioner. However, it should be mentioned, that the SPAI preconditioner can be parallelized efficiently, as opposed to the iLU(0) method which is highly sequential in nature.

Fig. 4, depicts the decomposition of the L-shaped domain (top) as well as the associated solution of the 2D Poisson equation (bottom).

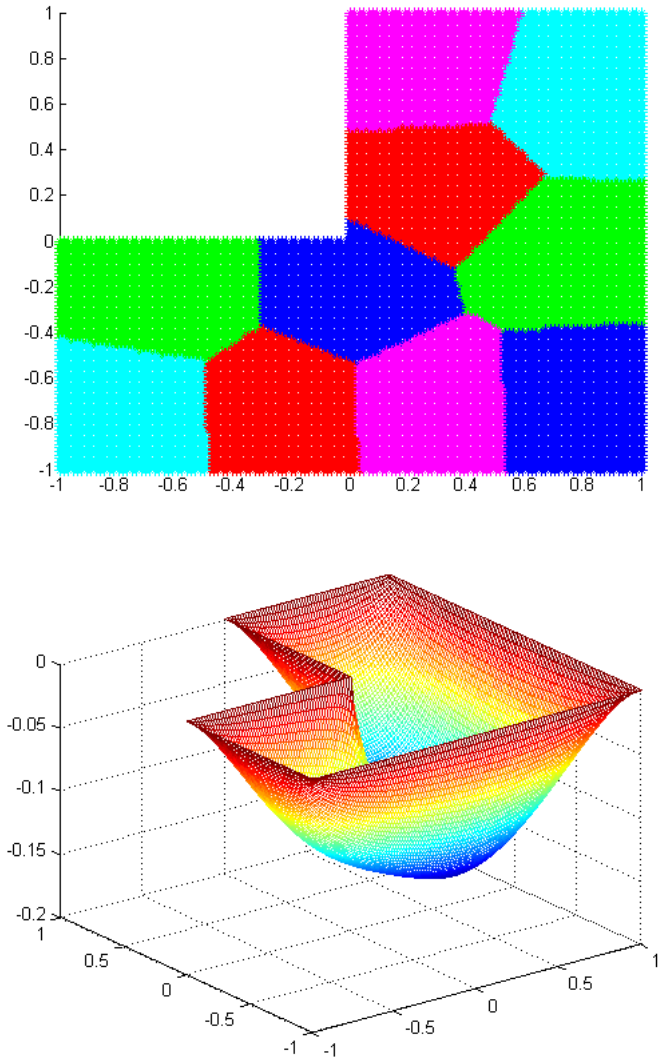


Figure 4: Decomposition of the L-shaped domain (top) as well as the associated solution of the 2D Poisson equation (bottom).

5 Fredholm IE - Nystrom Method

Solve the following integral equation:

$$u(x) + \int_0^1 x^2 t u(t) dt = e^x + x^2, \;\; 0 \leq x, t \leq 1,$$

using the Nystrom method. The computation of the integral should be performed using the Trapezoid rule as well as the Simpson 1/3 method. The linear system should be solved iteratively, using a Krylov subspace iterative method. The analytical solution of the integral equation is $u(x) = e^x$.

Solution

The given integral equation can be written more generally as follows:

$$u(x) + \int_0^1 K(x,t) u(t) dt = f(x), \tag{1}$$

where $K(x,t) = x^2 t$ is the kernel function and $f(x) = e^x + x^2$.

Therefore, it can be observed that the given equation is an instance of the Fredholm integral equation of the second kind.

Substituting the integral of the equation (1) with a weighted sum, yields the following:

$$u(x) + \sum_{j=1}^{N} w_j K(x, t_j) u(t_j) = f(x), \tag{2}$$

where the values of the weights $w_j, \;\; j = 1, 2, .., N$, depend on the chosen numerical integration method.

Applying the equation (2) in N points, leads to a linear system of N equations and N unknowns, i.e.:

$$u(x_i) + \sum_{j=1}^{N} w_j K(x_i, t_j) u(t_j) = f(x_i), \;\; i = 1, 2, .., N, \tag{3}$$

which can be equivalently rewritten, in compact matrix form, as follows:

$$(I + KD)u = f, \tag{4}$$

where

$$K = [k_{ij}] = [K(x_i, t_j)],$$

and

$$D = diag([w_1 \; w_2 \; ... \; w_N]).$$

As stated above, the entries of the matrix D, i.e. the weights $w_j, \;\; j = 1, 2, .., N$, depend on the chosen numerical integration method.

The trapezoid rule is given by:

$$\int_{x_1}^{x_N} f(x)dx = \frac{h}{2}[f(x_1) + 2\sum_{j=2}^{N-1} f(x_j) + f(x_N)] + O(h^2). \tag{5}$$

Furthermore, assuming that N is odd, the Simposon 1/3 rule is given by:

$$\int_{x_1}^{x_N} f(x)dx = \frac{h}{3}[f(x_1) + 4\sum_{j=1}^{(N-1)/2} f(x_{2j}) + 2\sum_{j=2}^{(N-1)/2} f(x_{2j-1}) + f(x_N)] + O(h^4) \tag{6}$$

The MATLAB script that was implemented in order to solve the given integral equation using the Nystrom method based on both the Trapezoid and the Simpson 1/3 rule is given below.

Listing 11: nystrom.m

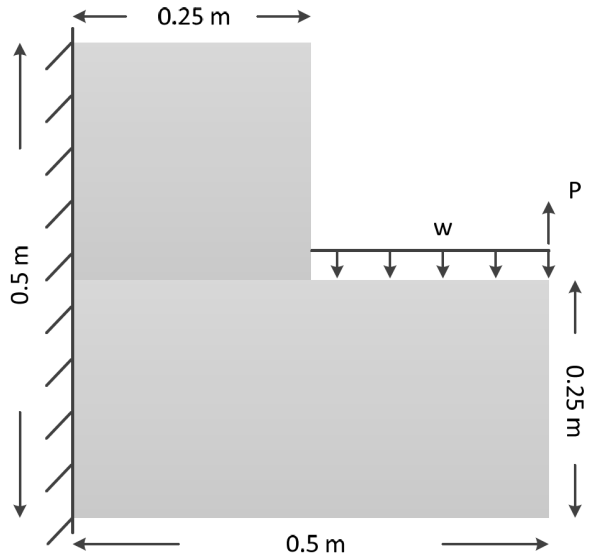
```
clear;clc;close all;
n=999; % use odd number of points (simpson 1/3)
% n=n+(1-mod(n,2));
f=@(x) (exp(x)+x.^2);
a=0;
b=1;
h=(b-a)/(n-1);
x=a:h:b;
t=0:h:(b-a);
Dt=(h/2)*spdiags([1; 2*ones(n-2,1); 1],0,n,n);
d=ones(n,1);
d(2:2:n-1)=4;
d(3:2:n-1)=2;
Ds=(h/3)*spdiags(d,0,n,n);
K=(x.^2)'*t;
I=speye(n);
At=I+K*Dt;
As=I+K*Ds;
b=f(x)';
% ut=At\b;
% us=As\b;
tol=1e-10;
ut=bicgstab(At,b,tol,100);
us=bicgstab(As,b,tol,100);
% ut=gmres(At,b,20,tol,5);
% us=gmres(As,b,20,tol,5);
plot(1:n,ut,'bo',1:n,us,'r*');
exact=exp(x)';
err_t=norm(exact-ut)%/norm(ut)
err_s=norm(exact-us)%/norm(us)
```

Choosing $N = 999$, led to an approximation error equal to $4.2007 \cdot 10^{-6}$, when the Trapezoid rule was utilized, and to an approximation error equal to $5.0178 \cdot 10^{-13}$, when the Simpson 1/3 rule was used.

It should be mentioned, that the coefficient matrices A_t and A_s are unsymmetric and have a clustered spectrum near unity. Thus, the unpreconditoned bicgstab was utilized, and converged to a precise solution within 1.5 iterations.

6 Elasticity - FEM - RAS - Coarse Spaces

Let us consider a thin plate which is subject to both a uniformly distributed load and a concentrated load as shown below:



The characteristics of the plate are the following:

- E = 210 GPA (modulus of elasticity)
- v = 0.3 (Young's modulus)
- t = 0.025 m (thickness)
- w = 100 kN/m (distributed load)
- P = 12.5 kN (concentrated load)

Discretize the plate using linear triangular finite elements, and compute the horizontal and vertical displacements on the plate, as well as the reactions at the supports. The linear system should be solved using Bi-CGSTAB method in conjunction with one-level and two-level Restricted Additive Schwarz (RAS) preconditioners.

Solution

The MATLAB program that was implemented in order to solve the given elasticity problem, is presented below.

Listing 12: lshape_elasticity.m

```
clear; clc; close all; rng(0);

ndoms  = 256;
nbasis = 4;
gptype = 0; % 0 for kmeans | 1 for metis

tol    = 1e-8;
maxit  = 500;
```

```
x0 = 0;
x1 = 0.25;
m = 128;        % number of subintervals per dimension

E  = 210*1e6; % Elastic modulus (Young's modulus)
NU = 0.3;     % Poisson's ratio
t  = 0.025;   % thickness
w  = 100;     % distributed load
P  = 12.5;    % concentrated load
p  = 1;       % plane stress

[h,ne,n,coo,con,bounds] = tfem(x0,x1,m);
[coo,con,bounds,sep,dpnts] = lshape_fem_2(x0,x1,m,coo,con);
ne = size(con,1);
n  = size(coo,1);

% Vertex numbering per square subdomain
n1 = dpnts{1};
n2 = dpnts{2};
n3 = dpnts{3};

% Dirichlet Boundaries (0)
west1 = n1(1:(m+1):(m+1)^2-m);
west3 = n3(1+(m+1):(m+1):(m+1)^2-m);
bounds = [west1 west3];
bounds = [2*bounds-1 2*bounds]; % both ux and uy components are zero

% Form forces (rhs vector)
b = zeros(2*n,1);
north2 = n2( (1:(m+1)) + (m+1)*m );
b(2*north2) = -(w*x1)/(m+1);      % distributed load
b(2*n2(end)) = b(2*n2(end)) + P;  % concentrated load

% A = spalloc(2*n,2*n,12*n);
% Coordinate format of global stiffness matrix
indi = zeros(12*n,1);
indj = zeros(12*n,1);
vval = zeros(12*n,1);
cnt  = 1;

% assemble global stiffness matrix
for e=1:ne
    x1 = coo(con(e,1),1);
    y1 = coo(con(e,1),2);
    x2 = coo(con(e,2),1);
    y2 = coo(con(e,2),2);
    x4 = coo(con(e,3),1);
    y4 = coo(con(e,3),2);
    M = LinearTriangleElementStiffness(E,NU,t,x1,y1,x2,y2,x4,y4,p);
    for i=1:3
        for j=1:3
            ii=con(e,i);
            jj=con(e,j);
            indi(cnt) = 2*ii-1;
            indj(cnt) = 2*jj-1;
            vval(cnt) = M(2*i-1,2*j-1);
            cnt = cnt + 1;
            indi(cnt) = 2*ii-1;
            indj(cnt) = 2*jj;
            vval(cnt) = M(2*i-1,2*j);
            cnt = cnt + 1;
            indi(cnt) = 2*ii;
            indj(cnt) = 2*jj-1;
            vval(cnt) = M(2*i,2*j-1);
            cnt = cnt + 1;
```

```
            indi(cnt) = 2*ii;
            indj(cnt) = 2*jj;
            vval(cnt) = M(2*i,2*j);
            cnt = cnt + 1;
%             A(2*ii-1,2*jj-1) = A(2*ii-1,2*jj-1) + M(2*i-1,2*j-1);
%             A(2*ii-1,2*jj  ) = A(2*ii-1,2*jj  ) + M(2*i-1,2*j  );
%             A(2*ii  ,2*jj-1) = A(2*ii  ,2*jj-1) + M(2*i  ,2*j-1);
%             A(2*ii  ,2*jj  ) = A(2*ii  ,2*jj  ) + M(2*i  ,2*j  );
        end
    end
end

% A = sparse(indi,indj,vval,2*n,2*n);
A = sparse(indi(1:cnt-1),indj(1:cnt-1),vval(1:cnt-1),2*n,2*n);

disp('Matrix Assembly - Done');

ind = setdiff(1:length(A),bounds); % all dofs appart from boundaries

bnd=[west1 west3];    % bound points
inn=setdiff(1:n,bnd); % inner points

u = zeros(length(A),1); % allocate solution vector and fill with zeros

% ------------------------------------------------------------------------
% ------------------------- Solve using RAS-BiCGSTAB ---------------------
% ------------------------------------------------------------------------
AA = A(ind,ind);
bb = b(ind);

G = 0.5*( abs(AA) + abs(AA)' );

if ( gptype )
    % Metis Partitioning
    map = PartSparseMat( G, ndoms );
else
    % Geometric Partitioning
    part = kmeans(coo(inn,:),ndoms,'MaxIter',200);
    map = zeros(1,length(ind));
    for i=1:ndoms
        idx = find( part==i );
        map(2*idx-1) = i;
        map(2*idx)   = i;
    end
end
ndoms = max(map);

disp('Partitioning - Done');

[In, Out, RAS.All, blk]      = inner_outer( G, map );
[RAS.All2, ~]                = form_overlap( G, map, RAS.All, Out );
[RAS.L, RAS.U, RAS.P, RAS.Q] = factorize( AA, RAS.All2 );

% SHEM Coarse Space for RAS
[R, basis] = shem( AA, In, Out, RAS.All, nbasis );
RAR = R*AA*R';
[cL,cU,cP,cQ] = lu(RAR, 'vector');

disp('RAS Preprocessing - Done');

u(ind) = bicgstab( AA, bb, tol, maxit, @(y) rasprec(y,RAS) );
u(ind) = bicgstab( AA, bb, tol, maxit, ...
    @(y) rasprec(y,RAS) + R'*sp_solve(cL,cU,cP,cQ,R*y) );

% ------------------------------------------------------------------------
```

```
ux=u( ind(1:2:end) ); % x component of u (active dofs)
uy=u( ind(2:2:end) ); % y component of u (active dofs)

% bnd=[west1 west3];    % bound points
% inn=setdiff(1:n,bnd); % inner points

f=A*u;          % compute forces
fx=f(2*bnd-1); % x component of f on bounds
fy=f(2*bnd);   % y component of f on bounds

% plot solution
inc = 0.05;
figure, hold on,
quiver(coo(inn,1),coo(inn,2),ux,uy,'b');
quiver(coo(bnd,1),coo(bnd,2),fx,fy,'r');
axis([x0-inc 2*x1+inc x0-inc 2*x1+inc]);
axis square; grid; xlabel('x'); ylabel('y');
title('displacements and bound reactions');
legend('displacements','bound reactions');
```

The discretization of the plate was performed using linear triangular finite elements, and implemented utilizing the tfem.m and lshape_fem_2.m functions. The function tfem.m is presented in Section 4, while the lshape_fem_2.m function is given in Appendix B.

In this case, the dimension of the local stiffness matrices is 6×6, since each vertex of a triangular finite element has two degrees of freedom (dofs), i.e. the horizontal and the vertical component of the displacement field. The following MATLAB function describes the computation of the local stiffness matrices.

Listing 13: LinearTriangleElementStiffness.m

```
function k = LinearTriangleElementStiffness(E,NU,t,x1,y1,x2,y2,x4,y4,p)

A = (x1*(y2-y4) + x2*(y4-y1) + x4*(y1-y2))/2;

y24 = y2-y4;
y41 = y4-y1;
y12 = y1-y2;
x42 = x4-x2;
x14 = x1-x4;
x21 = x2-x1;

% six dofs per triangle - two dofs at each node
% B is of size 3x6
B = [y24  0  y41  0  y12  0;
       0  x42  0  x14  0  x21;
     x42 y24 x14 y41 x21 y12 ] / (2*A);

if ( p == 1 ) % plane stress

    D = (E/(1-NU*NU))*[1 NU 0; NU 1 0; 0 0 (1-NU)/2];

elseif ( p == 2 ) % plane strain

    D = (E/(1+NU)/(1-2*NU))*[1-NU NU 0; NU 1-NU 0; 0 0 (1-2*NU)/2];

end

k = t*A*B'*D*B;

end
```

In order to decompose the L-shaped domain into multiple subdomains, two cases were considered, i.e. (i) geometric techniques; and (ii) algebraic techniques. In the case of the geometric partitioning of the domain, the k-means algorithm was utilized, while in the case of the algebraic partitioning of the domain, the k-way algorithm of the METIS software library was used.

The MATLAB functions inner_outer.m, form_overlap.m and factorize.m constitute the building blocks of the preprocessing step of the RAS preconditioner, and are given in Appendix C1. The function ras_prec.m implements the preconditioning step of the RAS method, and is presented in Appendix C3. Concerning the two-level RAS preconditioner, the Nicolaides as well as the SHEM coarse space was utilized. The Spectral Harmonically Enriched Multiscale (SHEM) coarse space was computed using the shem.m function, which is given in Appendix C6. It should be noted, that setting nbasis = 1 to the shem.m function computes the Nicolaides coarse space.

In Tables 9 and 10, the convergence behavior of the Bi-CGSTAB method in conjunction with the RAS, RAS+Nico (RAS with Nicolaides coarse space), and RAS+SHEM$_4$ (RAS with SHEM coarse space) preconditioners, for increasing number of subdomains, is presented, for the cases of geometric and algebraic decomposition of the domain, respectively. It should be noted, that for the SHEM coarse space, four basis functions were utilized (1 piecewise constant + 3 eigenfunctions). Furthermore, the parameter m, i.e. the number of subintervals per dimension of the square domain which is used to construct the L-shaped domain, was set to 128, leading to a linear system with $98,816$ degrees of freedom (dofs) and $1,179,638$ non-zero elements. The prescribed tolerance for the solution of the linear system using the preconditioned Bi-CGSTAB method, was chosen to be 10^{-8}.

	128	256	512
RAS	195.5	241.5	303.5
RAS+Nico	165.5	202.5	241.5
RAS+SHEM$_4$	160.5	179.5	209.5

Table 9: Convergence behavior of the Bi-CGSTAB method in conjunction with the preconditioners RAS, RAS+Nico and RAS+SHEM$_4$, for increasing number of subdomains, and for the case of geometric decomposition of the domain using k-means algorithm.

	128	256	512
RAS	226.5	271.5	404.5
RAS+Nico	206	214.5	240.5
RAS+SHEM$_4$	166	194.5	221.5

Table 10: Convergence behavior of the Bi-CGSTAB method in conjunction with the preconditioners RAS, RAS+Nico and RAS+SHEM$_4$, for increasing number of subdomains, and for the case of algebraic decomposition of the domain using METIS algorithm.

It can be observed, that the geometric decomposition of the domain, has as a result a partitioning of improved quality compared to the algebraic counterpart, and hence, leads to improved convergence behavior of the one-level and two-level RAS preconditioned Bi-CGSTAB methods. Furthermore, the convergence behavior of the Bi-CGSTAB method in conjunction with the RAS+SHEM$_4$ preconditioner, leads to improved convergence behavior compared to the Bi-CGSTAB method in conjunction with the RAS and RAS+Nico preconditioners. However, the RAS+Nico preconditioner leads to improved convergence behavior compared to the one-level RAS method, and requires substantially less computational work compared the RAS+SHEM$_4$ preconditioner in order to construct the associated coarse space.

Fig. 5 depicts the displacements and reactions at the supports of the plate with $t = 0.025m$, which is subject to a uniformly distributed load $w = 100kN/m$, and a concentrated load $P = 12.5kN$.

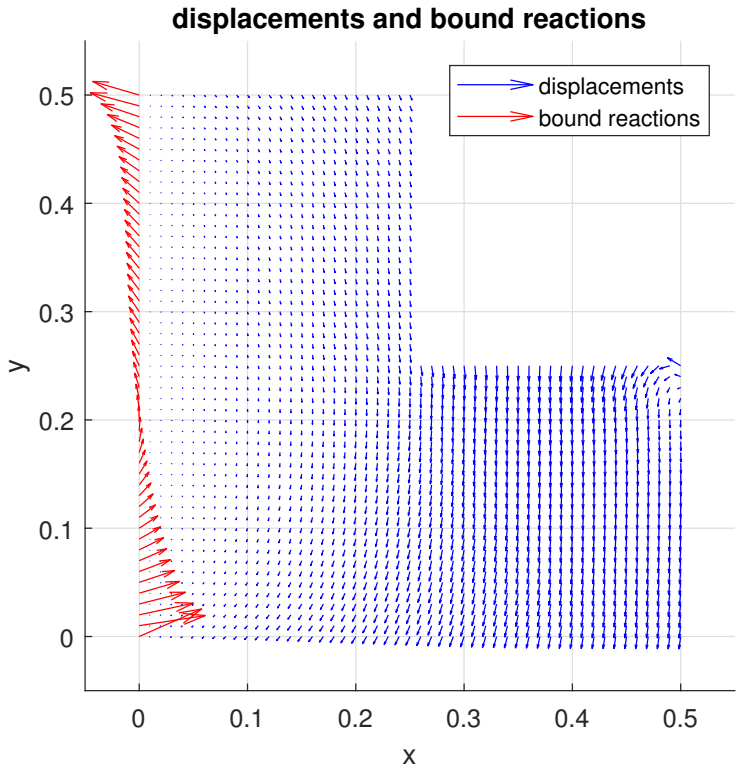


Figure 5: Displacements and reactions at the supports of the plate with $t = 0.025m$, which is subject to a uniformly distributed load $w = 100kN/m$, and a concentrated load $P = 12.5kN$.

7 Parabolic PDE - Time Domain Decomposition - Schur Complement Method

Let us consider the following parabolic partial differential equation:

$$\frac{\partial u}{\partial t} = \alpha \Delta u + f(x,y), \quad (x,y) \in \Omega,$$

which is subject to boundary and initial conditions of your choice. Discretize the PDE using finite differences and solve it utilizing the time domain decomposition method. The augmented linear system associated with the grouped time-steps, should be solved using the Schur complement method. The Schur complement system should be solved iteratively, utilizing the pipelined Bi-CGSTAB method combined with one-level and two-level domain decomposition preconditioners of your choice.

Solution

The MATLAB program that was implemented in order to solve the given parabolic PDE using the time domain decomposition method is presented below.

Listing 14: tdd.m

```
clear; clc; close all; rng(0);

alpha = 0.5;  % diffusion coefficient
epsil = 0;    % convection coefficient
nx    = 32;   % number of unknowns per dimension
nt    = 101;  % number of timesteps (total)
m     = 10;   % number of steps in each "grouped" timestep
ndoms = 8;    % number of subdomains
tol   = 1e-8; % KSM tolerance
maxit = 100;  % KSM max number of iterations
prob  = 50;   % probing parameter (set to -1 to turn probing OFF)

% space parameters
x0 = 0;
x1 = 1;
h  = (x1-x0)/(nx+1); % mesh size
n  = nx^2;

[y,x] = meshgrid( (x0+h):h:(x1-h) );
x = x(:);
y = y(:);

% time parameters
t0 = 0;
t1 = 0.10;
dt = (t1-t0)/(nt-1); % timestep size
nT = floor( nt / m ); % number of "grouped" timesteps

% single timestep matrix assembly
I = speye(n);
C = spdiags(ones(nx,1)*[-1 1],[-1 1],nx,nx);
A = I + ((alpha*dt)/(h^2))*gallery('poisson',nx) ...
        - ((epsil*dt)/(2*h))*kron(speye(nx),C);

% multiple timestep matrix assembly
K = matrix_unroll(A, m);

% initial condition setup
f = zeros( length(K), 1 );
```

```
f(1:n) = sin(pi*x).*sin(pi*y);

% Graph Partitioning
G = 0.5*(abs(K)+abs(K)'); % undirected graph
map = PartSparseMat(G,ndoms);
ndoms = max(map);

% Classify subdomain components into inner and outer points
[In, Out, All, blk] = inner_outer(G, map);

% Schur Complement Method Preprocessing
SC = sc_prep(K,In,Out);

tic
if (prob == -1)
    % Schur Complement Block Jacobi Setup
    SBJ = scbj_prep(K,SC);
    disp('Probing OFF');
else
    % Schur Complement Block Probing Setup
    SBP = scbp_prep(K,SC,prob);
    SBJ = SBP;
    disp('Probing ON');
end
toc

% Extract data from SC data structure
[In,out,L,U,P,Q,Kio] = deal(SC.In,SC.out,SC.L,SC.U,SC.P,SC.Q,SC.Aio);

% % Coarse Space Construction
% V = spalloc( ndoms, length(out), length(out) );
% mask = zeros(length(K),1);
% mask(out) = 1:length(out);
% for i=1:ndoms
%     V(i,mask(Out{i})) = 1;
% end
% VKV = V*SC.Aoo*V';
% VSV = VKV - V*blkdiag( SBJ.T{:} )*V';

% Start Simulation
umin = min(f);
umax = max(f);
x = reshape(x,nx,nx);
y = reshape(y,nx,nx);
mesh( x, y, reshape(f(1:n),nx,nx) ); % plot initial condition
axis( [x0+h x1-h x0+h x1-h umin umax] );

u = zeros( length(K), 1 );
for i=1:nT

    % Solve interface problem
    u(out) = pbicgstab2(@(x) Smultx(x,SC), Srhs(f,SC), tol, maxit, ...
                                    @(x) blkjac(x,SBJ));
%     u(out) = pbicgstab2(@(x) Smultx(x,SC), Srhs(f,SC), tol, maxit, ...
%                                     @(x) blkjac(x,SBJ) + V'*(VKV\(V*x)) );
%     u(out) = pbicgstab2(@(x) Smultx(x,SC), Srhs(f,SC), tol, maxit, ...
%                                     @(x) blkjac(x,SBJ) + V'*(VSV\(V*x)) );

    % Back substitute to inner dofs
    for j=1:ndoms
        u(In{j}) = sp_solve( L{j}, U{j}, P{j}, Q{j}, ...
                                    f(In{j}) - Kio{j} * u(out) );
    end

    f(1:n) = u(end-n+1:end);
```

```
    % Plot batch solution
    for j=1:m
        II = (j*n+1):(j+1)*n;
        mesh( x, y, reshape(u(II),nx,nx) );
        axis( [x0+h x1-h x0+h x1-h umin umax] );
        getframe;
    end

end
```

The function matrix_unroll.m assembles the augmented coefficient matrix in which m time-steps are grouped together, and is given in Appendix C7. The functions sc_prep.m, scbj_prep.m, scbp_prep.m, Smultx.m and Srhs.m implement the Schur complement method combined with various preconditioners based on the block Jacobi method as well as probing techniques, and are presented in Appendix C4. The MATLAB implementation of the pipelined Bi-CGSTAB method is given in Appendix D.

The coarse space that is utilized is a modified version of the Nicolaides coarse space for the Schur complement matrix. The coefficient matrix S_0 according to the Nicolaides coarse space is as follows:

$$S_0 = VSV^T = V(A_{\Gamma\Gamma} - A_{\Gamma I}A_{II}^{-1}A_{I\Gamma})V^T = VA_{\Gamma\Gamma}V^T - VA_{\Gamma I}A_{II}^{-1}A_{I\Gamma}V^T, \tag{7}$$

where the subscript Γ denotes the interface dofs, and the subscript I denotes the interior dofs. It should be noted, that the rows of the restriction matrix $V \in \{0,1\}^{n_{doms}\times n_\Gamma}$ define piecewise constant basis functions, each one associated with a different subdomain. The modified coarse space that is utilized, which is referred as the inexact Nicolaides coarse space, approximates the coefficient matrix S_0 using only the leading term $VA_{\Gamma\Gamma}V^T$, ignoring the term $VA_{\Gamma I}A_{II}^{-1}A_{I\Gamma}V^T$, and hence, reducing the floating-point operations required for the computation of the coarse space.

In Table 11, the convergence behavior of the pipelined Bi-CGSTAB method in conjunction with the preconditioners:

- SCBJ, i.e. the Schur Complement Block Jacobi method,
- SCBJ+iNico, i.e. the SCBJ method combined with the inexact Nicolaides coarse space,
- SCBP(60), i.e. the SCBJ method that utilizes the probing technique,
- SCBP(60)+iNico, i.e. the SCBP(60) method combined with the inexact Nicolaides coarse space,

is presented, for increasing number of time-steps in each grouped time-step, and for constant number of dofs per subdomain. It should be mentioned, that the k-way graph partitioning algorithm from METIS software library, is utilized in order to subdivide the augmented space-time domain into subdomains.

Time-Steps in Grouped Time-Step	10	20	40	80
Subdomains	8	16	32	64
size(K)	11,264	21,504	41,984	82,944
size(S)	2,271	6,521	14,159	29,828
SCBJ	11	15	40	52
SCBJ+iNico	15	20	24	29
SCBP(60)	12	15	22	55
SCBP(60)+iNico	15	21	26	34

Table 11: Convergence behavior of the pipelined Bi-CGSTAB method in conjunction with the SCBJ, SCBJ+iNico, SCBP(60) and SCBP(60)+iNico preconditioners, for increasing number of time-steps in each grouped time-step, and for constant number of dofs per subdomain.

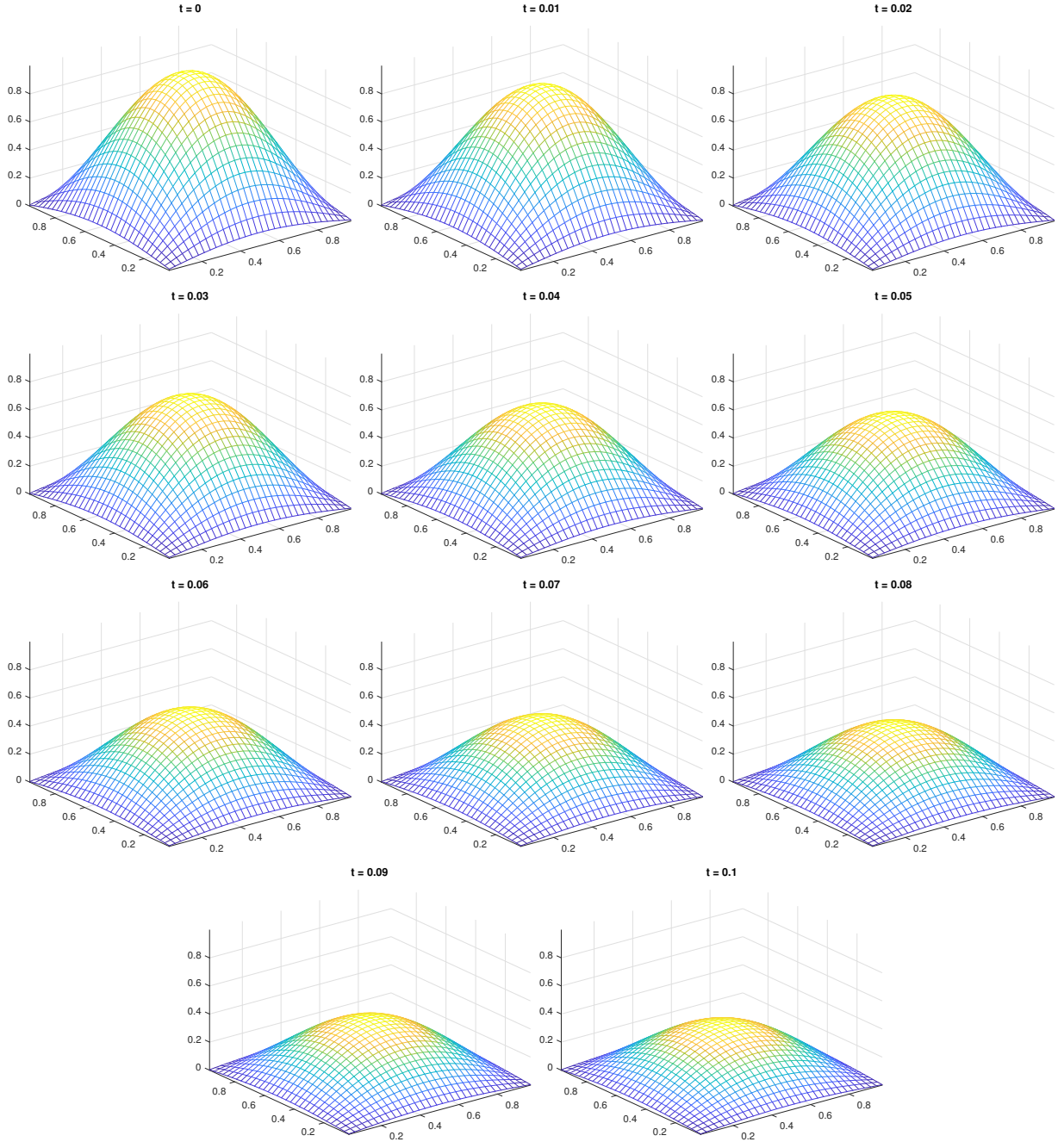


Figure 6: Solution of the parabolic PDE in various time-steps $t \in [0, 0.1]$.

A Computing the Integrals for the Finite Element Method

A.1 Bilinear Finite Elements

Listing 15: integrals_qfem.m

```
clear; clc;
syms xi eta J y41 y12 x14 x21

N1=(1/4)*(1-xi)*(1-eta);
N2=(1/4)*(1+xi)*(1-eta);
N3=(1/4)*(1+xi)*(1+eta);
N4=(1/4)*(1-xi)*(1+eta);

N1x=(1/(8*J))*(y41*(eta-1)+y12*(xi-1));
N1y=(1/(8*J))*(x14*(eta-1)+x21*(xi-1));

N2x=(1/(8*J))*(y41*(-eta+1)+y12*(-xi-1));
N2y=(1/(8*J))*(x14*(-eta+1)+x21*(-xi-1));

N3x=(1/(8*J))*(y41*(eta+1)+y12*(xi+1));
N3y=(1/(8*J))*(x14*(eta+1)+x21*(xi+1));

N4x=(1/(8*J))*(y41*(-eta-1)+y12*(-xi+1));
N4y=(1/(8*J))*(x14*(-eta-1)+x21*(-xi+1));

% Diffusion Terms

W11=J*int(int(N1x*N1x+N1y*N1y,xi,-1,1),eta,-1,1);
W22=J*int(int(N2x*N2x+N2y*N2y,xi,-1,1),eta,-1,1);
W33=J*int(int(N3x*N3x+N3y*N3y,xi,-1,1),eta,-1,1);
W44=J*int(int(N4x*N4x+N4y*N4y,xi,-1,1),eta,-1,1);

W12=J*int(int(N1x*N2x+N1y*N2y,xi,-1,1),eta,-1,1);
W13=J*int(int(N1x*N3x+N1y*N3y,xi,-1,1),eta,-1,1);
W14=J*int(int(N1x*N4x+N1y*N4y,xi,-1,1),eta,-1,1);

W23=J*int(int(N2x*N3x+N2y*N3y,xi,-1,1),eta,-1,1);
W24=J*int(int(N2x*N4x+N2y*N4y,xi,-1,1),eta,-1,1);

W34=J*int(int(N3x*N4x+N3y*N4y,xi,-1,1),eta,-1,1);

% Reaction Terms

M11=J*int(int(N1*N1,xi,-1,1),eta,-1,1);
M22=J*int(int(N2*N2,xi,-1,1),eta,-1,1);
M33=J*int(int(N3*N3,xi,-1,1),eta,-1,1);
M44=J*int(int(N4*N4,xi,-1,1),eta,-1,1);

M12=J*int(int(N1*N2,xi,-1,1),eta,-1,1);
M13=J*int(int(N1*N3,xi,-1,1),eta,-1,1);
M14=J*int(int(N1*N4,xi,-1,1),eta,-1,1);

M23=J*int(int(N2*N3,xi,-1,1),eta,-1,1);
M24=J*int(int(N2*N4,xi,-1,1),eta,-1,1);

M34=J*int(int(N3*N4,xi,-1,1),eta,-1,1);

% Source Terms

f1=J*int(int(N1,xi,-1,1),eta,-1,1);
f2=J*int(int(N2,xi,-1,1),eta,-1,1);
```

```
f3=J*int(int(N3,xi,-1,1),eta,-1,1);
f4=J*int(int(N4,xi,-1,1),eta,-1,1);
```

A.2 Linear Finite Elements

Listing 16: integrals_tfem.m

```
clear; clc;
syms xi eta J y41 y12 x14 x21

N1=-(1/2)*(xi+eta);
N2=(1/2)*(1+xi);
N4=(1/2)*(1+eta);

N1x=(1/(4*J))*(y41*(-1)+y12*(-1));
N1y=(1/(4*J))*(x14*(-1)+x21*(-1));

N2x=(1/(4*J))*(y41*(1)+y12*(0));
N2y=(1/(4*J))*(x14*(1)+x21*(0));

N4x=(1/(4*J))*(y41*(0)+y12*(1));
N4y=(1/(4*J))*(x14*(0)+x21*(1));

% Diffusion Terms

W11=J*int(int(N1x*N1x+N1y*N1y,xi,-1,-eta),eta,-1,1);
W22=J*int(int(N2x*N2x+N2y*N2y,xi,-1,-eta),eta,-1,1);
W44=J*int(int(N4x*N4x+N4y*N4y,xi,-1,-eta),eta,-1,1);

W12=J*int(int(N1x*N2x+N1y*N2y,xi,-1,-eta),eta,-1,1);
W14=J*int(int(N1x*N4x+N1y*N4y,xi,-1,-eta),eta,-1,1);

W24=J*int(int(N2x*N4x+N2y*N4y,xi,-1,-eta),eta,-1,1);

% Reaction Terms

M11=J*int(int(N1*N1,xi,-1,-eta),eta,-1,1);
M22=J*int(int(N2*N2,xi,-1,-eta),eta,-1,1);
M44=J*int(int(N4*N4,xi,-1,-eta),eta,-1,1);

M12=J*int(int(N1*N2,xi,-1,-eta),eta,-1,1);
M14=J*int(int(N1*N4,xi,-1,-eta),eta,-1,1);

M24=J*int(int(N2*N4,xi,-1,-eta),eta,-1,1);

% Source Terms

f1=J*int(int(N1,xi,-1,-eta),eta,-1,1);
f2=J*int(int(N2,xi,-1,-eta),eta,-1,1);
f4=J*int(int(N4,xi,-1,-eta),eta,-1,1);
```

B Discretizing L-Shaped Domains using Finite Elements

Listing 17: lshape_fem.m

```
function [coo,con,bounds,sep,dpnts]=lshape_fem(a,b,m,coo1,con1)

n=(m+1)^2;
coo2=zeros(size(coo1)); con2=zeros(size(con1)); nodes2=zeros(1,n);
coo3=zeros(size(coo1)); con3=zeros(size(con1)); nodes3=zeros(1,n);

west=1:(m+1):(n-m);
nodes2(west)=west+m; % west2 same as east1
for i=1:m+1
    idx=(i-1)*(m+1)+2:i*(m+1); % attention to +2
    nodes2(idx)=idx-1+n-(i-1); % continue numbering
end

south=1:(m+1);
nodes3(south)=nodes2(south+(m+1)*m); % south3 same as north2
nodes3(m+2:n)=(2*n-m):(3*n-2*m-2); % continue numbering

idx=1;
if (size(con1,2)==4)
    for i=1:m
        for j=1:m
            con2(idx,:)=nodes2(con1(idx,:));
            con3(idx,:)=nodes3(con1(idx,:));
            idx=idx+1;
        end
    end
else % if (size(con1,2)==3)
    for i=1:m
        for j=1:m
            con2(idx,:)=nodes2(con1(idx,:));
            con2(idx+1,:)=nodes2(con1(idx+1,:));
            con3(idx,:)=nodes3(con1(idx,:));
            con3(idx+1,:)=nodes3(con1(idx+1,:));
            idx=idx+2;
        end
    end
end
coo2(:,1)=coo1(:,1)+(b-a);
coo2(:,2)=coo1(:,2);
coo2(west,:)=[];

coo3(:,1)=coo1(:,1)+(b-a);
coo3(:,2)=coo1(:,2)+(b-a);
coo3(south,:)=[];

coo=[coo1;coo2;coo3];
con=[con1;con2;con3];

lsouth=[1:m+1 nodes2(2:m+1)];
lnorth=[(1:m+1)+(m+1)*m nodes3((1:m+1)+(m+1)*m)];
lwest=[(m+2):(m+1):(n-2*m-1) nodes3((m+2):(m+1):(n-2*m-1))];
least=[nodes2(((m+2):(m+1):(n-2*m-1))+m) nodes3((1:(m+1):(n-2*m-1))+m)];
bounds=[lsouth lnorth lwest least];

sep=[nodes2(west) nodes3(south(2:end))];

dpnts=cell(3,1);
dpnts{1}=1:n;
dpnts{2}=nodes2;
dpnts{3}=nodes3;

end
```

Listing 18: lshape_fem_2.m

```
function [coo,con,bounds,sep,dpnts]=lshape_fem_2(a,b,m,coo1,con1)

n=(m+1)^2;
coo2=zeros(size(coo1)); con2=zeros(size(con1)); nodes2=zeros(1,n);
coo3=zeros(size(coo1)); con3=zeros(size(con1)); nodes3=zeros(1,n);

west=1:(m+1):(n-m);
nodes2(west)=west+m; % west2 same as east1
for i=1:m+1
    idx=(i-1)*(m+1)+2:i*(m+1); % attention to +2
    nodes2(idx)=idx-1+n-(i-1); % continue numbering
end

south=1:(m+1);
nodes3(south)=south+(m+1)*m; % south3 same as north1
nodes3(m+2:n)=(2*n-m):(3*n-2*m-2); % continue numbering

idx=1;
if (size(con1,2)==4)
    for i=1:m
        for j=1:m
            con2(idx,:)=nodes2(con1(idx,:));
            con3(idx,:)=nodes3(con1(idx,:));
            idx=idx+1;
        end
    end
else % if (size(con1,2)==3)
    for i=1:m
        for j=1:m
            con2(idx,:)=nodes2(con1(idx,:));
            con2(idx+1,:)=nodes2(con1(idx+1,:));
            con3(idx,:)=nodes3(con1(idx,:));
            con3(idx+1,:)=nodes3(con1(idx+1,:));
            idx=idx+2;
        end
    end
end
coo2(:,1)=coo1(:,1)+(b-a);
coo2(:,2)=coo1(:,2);
coo2(west,:)=[];

coo3(:,1)=coo1(:,1);
coo3(:,2)=coo1(:,2)+(b-a);
coo3(south,:)=[];

coo=[coo1;coo2;coo3];
con=[con1;con2;con3];

lsouth=[1:m+1 nodes2(2:m+1)];
lnorth=[nodes3((1:m+1)+(m+1)*m) nodes2((2:m+1)+(m+1)*m)];
lwest=[(m+2):(m+1):(n-2*m-1) nodes3(1:(m+1):(n-2*m-1))];
least=[nodes2(((m+2):(m+1):(n-2*m-1))+m) nodes3((1:(m+1):(n-2*m-1))+m)];
bounds=[lsouth lnorth lwest least];

sep=[nodes2(west) nodes3(south(1:end-1))];

dpnts=cell(3,1);
dpnts{1}=1:n;
dpnts{2}=nodes2;
dpnts{3}=nodes3;

end
```

C Domain Decomposition Methods

C.1 Utility Functions

Listing 19: sp_solve.m

```
function x=sp_solve(L,U,p,q,b)
x=U\(L\b(p,:));
x(q,:)=x;
end
```

Listing 20: ksm.m

```
function x = ksm(A,b,tol,maxit,M,krylov_sel)

if ( krylov_sel == 0 )
    x = bicgstab(A,b,tol,maxit,M);
elseif ( krylov_sel == 1 )
    x = gmres(A,b,20,tol,maxit,M);
else
    x = gmres(A,b,[],tol,maxit,M);
end

end
```

Listing 21: inner_outer.m

```
function [In,Out,All,blk]=inner_outer(G,map)

    ndoms = max(map);
    In =cell(ndoms,1);
    Out=cell(ndoms,1);
    All=cell(ndoms,1);
    blk=zeros(ndoms+1,1); blk(1)=1;
    for i=1:ndoms
        dpnts=find(map==i);
        for j=1:length(dpnts)
            pnt=dpnts(j);
            nbr=find(G(:,pnt));
            if any(map(nbr)~=i)
                Out{i}=[Out{i} pnt];
            else
                In{i}=[In{i} pnt];
            end
        end
        All{i}=[In{i} Out{i}];
        blk(i+1)=blk(i)+length(All{i});
    end

end
```

Listing 22: form_overlap.m

```
function [All2,overlap]=form_overlap(G,map,All,Out)

    n       = length(G);
    ndoms   = length(All);
    overlap = cell(ndoms,1);
    All2    = cell(ndoms,1); % All including overlap

    % one level overlap (delta = 1)
    for i=1:ndoms
        pnts=zeros(n,1);
        for j=Out{i}
            nbr=find(G(:,j));
            map_neq_i=(map(nbr)~=i);
            pnts( nbr(map_neq_i) )=1;
        end
        overlap{i}=find(pnts)';
        All2{i}=[All{i} overlap{i}];
    end

end
```

Listing 23: factorize.m

```
function [L,U,P,Q]=factorize(A,All)

    ndoms=length(All);
    L=cell(ndoms,1);
    U=cell(ndoms,1);
    P=cell(ndoms,1);
    Q=cell(ndoms,1);
    for i=1:ndoms
        [L{i},U{i},P{i},Q{i}]=lu( A(All{i},All{i}), 'vector' );
    end

end
```

C.2 Block Jacobi Method

Listing 24: blkjac.m

```
function x=blkjac(b,params)

    blk   = params.blk;
    L     = params.L;
    U     = params.U;
    P     = params.P;
    Q     = params.Q;
    ndoms = length(L);
    bc    = cell(ndoms,1);
    xc    = cell(ndoms,1);
    for i=1:ndoms
        bc{i} = b( blk(i):blk(i+1)-1 );
        xc{i} = sp_solve(L{i},U{i},P{i},Q{i},bc{i});
    end
    x = vertcat(xc{:});

end
```

C.3 Restricted Additive Schwarz Method

Listing 25: rasprec.m

```
function x=rasprec(b,params)

    L     = params.L;
    U     = params.U;
    P     = params.P;
    Q     = params.Q;
    All   = params.All;
    All2  = params.All2;
    ndoms = length(L);

    x = zeros(length(b),1);
    for i=1:ndoms
        bb = b(All2{i});
        xx = sp_solve(L{i},U{i},P{i},Q{i},bb);
        x( All{i} ) = xx(1:length(All{i}));
    end

end
```

C.4 Schur Complement Method

Listing 26: Smultx.m

```
function y = Smultx(x, SC)

    [Aoo, Aio, Aoi] = deal(SC.Aoo, SC.Aio, SC.Aoi);
    [L, U, P, Q] = deal( SC.L, SC.U, SC.P, SC.Q );
    ndoms = length(L);

    y = Aoo*x;
    for i=1:ndoms
        y = y - Aoi{i} * sp_solve( L{i}, U{i}, P{i}, Q{i}, Aio{i} * x );
    end

end
```

Listing 27: Srhs.m

```
function g = Srhs(b, SC)

    [In, out, Aoi] = deal( SC.In, SC.out, SC.Aoi );
    [L, U, P, Q] = deal( SC.L, SC.U, SC.P, SC.Q );
    ndoms = length(L);

    g = b(out);
    for i=1:ndoms
        g = g - Aoi{i} * sp_solve( L{i}, U{i}, P{i}, Q{i}, b(In{i}) );
    end

end
```

Listing 28: sc_prep.m

```
function SC = sc_prep(A,In,Out)
ndoms = length(In);
out = horzcat(Out{:});
Aoo = A(out,out);
[Aio, Aoi, L, U, P, Q] = deal( cell(ndoms,1) );
for i=1:ndoms
    inn = In{i};
    Aio{i} = A(inn,out);
    Aoi{i} = A(out,inn);
    [L{i}, U{i}, P{i}, Q{i}] = lu( A(inn,inn), 'vector' );
end
[SC.L, SC.U, SC.P, SC.Q] = deal(L, U, P, Q);
[SC.Aoo, SC.Aio, SC.Aoi] = deal(Aoo, Aio, Aoi);
[SC.In, SC.Out, SC.out] = deal(In, Out, out);
end
```

Listing 29: scbj_prep.m

```
function SBJ = scbj_prep(A,SC)
[In,Out,L,U,P,Q] = deal(SC.In,SC.Out,SC.L,SC.U,SC.P,SC.Q);
ndoms = length(In);
[S, Ls, Us, Ps, Qs, T] = deal( cell(ndoms,1) );
blk2 = zeros(ndoms+1,1);
blk2(1) = 1;
for i=1:ndoms
%     S{i} = A(Out{i},Out{i}) - A(Out{i},In{i}) * ...
%         sp_solve( L{i},U{i},P{i},Q{i}, A(In{i},Out{i}) );
    T{i} = A(Out{i},In{i})*sp_solve( L{i},U{i},P{i},Q{i}, A(In{i},Out{i}) );
    S{i} = A(Out{i},Out{i}) - T{i};
    [Ls{i}, Us{i}, Ps{i}, Qs{i}] = lu( S{i}, 'vector' );
    blk2(i+1) = length(Out{i});
end
blk2 = cumsum(blk2);
[SBJ.L, SBJ.U, SBJ.P, SBJ.Q, SBJ.blk] = deal( Ls, Us, Ps, Qs, blk2 );
SBJ.T = T;
end
```

Listing 30: scbp_prep.m

```
function SBP = scbp_prep(A,SC,prob)
[In,Out,L,U,P,Q] = deal(SC.In,SC.Out,SC.L,SC.U,SC.P,SC.Q);
ndoms = length(In);
[Sp, Lp, Up, Pp, Qp] = deal( cell(ndoms,1) );
blk2 = zeros(ndoms+1,1);
blk2(1) = 1;
for i=1:ndoms
    [AOO,AOI,AIO] = deal(A(Out{i},Out{i}),A(Out{i},In{i}),A(In{i},Out{i}));
    [LL,UU,PP,QQ] = deal( L{i}, U{i}, P{i}, Q{i} );
    Sx = @(x) AOO*x-AOI*sp_solve(LL,UU,PP,QQ,AIO*x);
    Sp{i} = probe2(Sx, prob, length(Out{i}));
    [Lp{i}, Up{i}, Pp{i}, Qp{i}] = lu( Sp{i}, 'vector' );
    blk2(i+1) = length(Out{i});
end
blk2 = cumsum(blk2);
[SBP.L, SBP.U, SBP.P, SBP.Q, SBP.blk] = deal( Lp, Up, Pp, Qp, blk2 );
end
```

C.5 Probing Technique

Listing 31: probe2.m

```
function M = probe2( A, d, n )
% n = length(A);
k = min( [2*d+1,n] );
if k == 1
    M = spdiags( feval(A,ones(n,1)), 0, n, n );
else
    v = rem( [1:n]'*ones(1,k), k ) == ones(n,1)*[1:(k-1),0];
    av = feval(A,v);
    M = spalloc(n,n,n);
    for c=1:k
        for i=c:k:n
            M( max([i-d 1]):min([i+d n]), i ) = ...
                                av( max([i-d 1]):min([i+d n]), c );
        end
    end
end
end
```

C.6 SHEM Coarse Space

Listing 32: shem.m

```
function [R,basis] = shem(A,In,Out,All,nbasis)

    n      = length(A); ndoms  = length(In);
    basis  = cell(ndoms,1); source = cell(ndoms,1);
    Sc     = cell(ndoms,1);
    for i=1:ndoms

        ninn = length(In{i} );
        nout = length(Out{i});
        basis{i} = zeros(ninn+nout, nbasis);
        basis{i}(:,1) = 1;

        if ( nbasis > 1 )

            Sc{i} = A(Out{i},Out{i})-A(Out{i},In{i})* ...
                        (A(In{i},In{i})\A(In{i},Out{i}));
            [source{i},~] = eigs( Sc{i}, nbasis-1, 'SM' );

            for j = 2:nbasis
                basis{i}(ninn+1:end,j) = source{i}(:,j-1);
                basis{i}(1:ninn,    j) = A(In{i},In{i}) \ ...
                    ( - A(In{i},Out{i}) * basis{i}(ninn+1:end,j) );
            end

        end

    end

    R = spalloc(nbasis*ndoms,n,nbasis*n);
    for i=1:ndoms
        for j=1:nbasis
            R(nbasis*(i-1)+j, All{i}) = basis{i}(:,j)';
        end
```

```
    end

end
```

C.7 Unrolling Time-Steps in Time Domain Decomposition Method

Listing 33: matrix_unroll.m

```
function K = matrix_unroll(A,m)

n = length(A);
[Ai,Aj,Av] = find(A);

Kr  = (m+1)*n;
Knz = m*nnz(A)+(m+1)*n;
Ki  = zeros(Knz,1);
Kj  = zeros(Knz,1);
Kv  = zeros(Knz,1);

Ki(1:n) = 1:n;
Kj(1:n) = 1:n;
Kv(1:n) = 1;
% si = n;
ei = n;
for i=1:m
    si = ei + 1;
    ei = ei + n;
    Ki(si:ei) = ((1:n) + i*n);
    Kj(si:ei) = ((1:n) + i*n - n);
    Kv(si:ei) = -1;
end
for i=1:m
    si = ei + 1;
    ei = ei + nnz(A);
    Ki(si:ei) = (Ai + i*n);
    Kj(si:ei) = (Aj + i*n);
    Kv(si:ei) = Av;
end

K = sparse( Ki, Kj, Kv, Kr, Kr );

end
```

D Pipelined Bi-CGSTAB Method

Listing 34: pbicgstab2.m

```
function x=pbicgstab2(A,b,tol,Nmax,M)

n=length(b);
phat = zeros(n,1);
shat = zeros(n,1);
s = zeros(n,1);
z = zeros(n,1);
zhat = zeros(n,1);
v = zeros(n,1);
omega = 0;
tolb = tol*norm(b);
x = zeros(n,1);
r0=b-feval(A,x); r=r0; rhat=feval(M,r); w=feval(A,rhat); what=feval(M,w);
t=feval(A,what); alpha=(r'*r)/(r'*w); beta=0;
for i=1:Nmax
    r_old = r;
    phat = rhat + beta*(phat - omega*shat);
    s    = w    + beta*(s    - omega*z);
    shat = what + beta*(shat - omega*zhat);
    z    = t    + beta*(z    - omega*v);
    q    = r    - alpha*s;
    qhat = rhat - alpha*shat;
    y    = w    - alpha*z;
    qdoty = q'*y;
    ydoty = y'*y;
    zhat = feval(M,z);
    v = feval(A,zhat);
    omega = qdoty/ydoty;
    x = x + alpha*phat + omega*qhat;
    r = q    - omega*y;

%     if (norm(r) < tolb)
%         disp(['pipelined bicgstab needed ' ...
%             num2str(i-1) '.5 iterations'])
%         break;
%     end

    rhat = qhat - omega*(what - alpha*zhat);
    w    = y    - omega*(t    - alpha*v);
    r0dotr_old = r0'*r_old;
    r0dotr = r0'*r;
    r0dotw = r0'*w;
    r0dots = r0'*s;
    r0dotz = r0'*z;
    what = feval(M,w);
    t = feval(A,what);

    if (norm(r) < tolb)
        disp(['pipelined bicgstab converged at iteration ' num2str(i) ...
              ' to a solution with relative residual ' ...
            num2str( norm(r)/norm(b) ) '.'])
        break;
    end

    beta = (alpha/omega)*(r0dotr/r0dotr_old);
    alpha = r0dotr/(r0dotw + beta*r0dots - beta*omega*r0dotz);
end

end
```

References

[1] X.-C. Cai and M. Sarkis. A restricted additive Schwarz preconditioner for general sparse linear systems. *SIAM Journal on Scientific Computing*, 21(2):792–797, 1999.

[2] T. F. Chan and T. P. Mathew. The interface probing technique in domain decomposition. *SIAM Journal on Matrix Analysis and Applications*, 13(1):212–238, 1992.

[3] S. Cools and W. Vanroose. The communication-hiding pipelined BiCGstab method for the parallel solution of large unsymmetric linear systems. *Parallel Computing*, 65:1–20, 2017.

[4] T. A. Davis. *Direct Methods for Sparse Linear Systems*. SIAM, 2006.

[5] R. D. Falgout, S. Friedhoff, T. V. Kolev, S. P. MacLachlan, and J. B. Schroder. Parallel time integration with multigrid. *SIAM Journal on Scientific Computing*, 36(6):C635–C661, 2014.

[6] M. J. Gander and A. Loneland. SHEM: An optimal coarse space for RAS and its multiscale approximation. In *Domain Decomposition Methods in Science and Engineering XXIII*, pages 313–321. Springer, 2017.

[7] M. J. Gander, A. Loneland, and T. Rahman. Analysis of a new harmonically enriched multiscale coarse space for domain decomposition methods. *arXiv preprint arXiv:1512.05285*, 2015.

[8] G. H. Golub and C. F. V. Loan. *Matrix Computations, 4th Edition*. The Johns Hopkins University Press, 2013.

[9] M. J. Grote and T. Huckle. Parallel preconditioning with sparse approximate inverses. *SIAM Journal on Scientific Computing*, 18(3):838–853, 1997.

[10] W. Hackbusch. *Multi-Grid Methods and Applications*. Springer, 2003.

[11] P. I. Kattan. *MATLAB Guide to Finite Elements: An Interactive Approach*. Springer Science & Business Media, 2010.

[12] T. Mathew. *Domain Decomposition Methods for the Numerical Solution of Partial Differential Equations*, volume 61. Springer Science & Business Media, 2008.

[13] F. Nataf, H. Xiang, V. Dolean, and N. Spillane. A coarse space construction based on local Dirichlet-to-Neumann maps. *SIAM Journal on Scientific Computing*, 33(4):1623–1642, 2011.

[14] R. A. Nicolaides. Deflation of conjugate gradients with applications to boundary value problems. *SIAM Journal on Numerical Analysis*, 24(2):355–365, 1987.

[15] Y. Saad. *Iterative Methods for Sparse Linear Systems, 2nd Edition*. SIAM, 2003.

[16] F.-J. Sayas. A gentle introduction to the finite element method. Lecture Notes, 2008.

[17] J. R. Shewchuk. An introduction to the conjugate gradient method without the agonizing pain. Technical report, Carnegie Mellon University, 1994.

[18] G. Strang. *Linear Algebra and its Applications, 3rd Edition*. Brooks/Cole Thomson Learning, 1988.

[19] E. Suli. Lecture notes on finite element methods for partial differential equations. University of Oxford, 2012.

[20] A. Toselli and O. Widlund. *Domain Decomposition Methods - Algorithms and Theory*, volume 34. Springer Science & Business Media, 2006.

[21] L. N. Trefethen and D. Bau. *Numerical Linear Algebra*. SIAM, 1997.

[22] H. van der Vorst. *Iterative Krylov Methods for Large Linear Systems*. Cambridge University Press, 2003.