ECN
MSc Computational Mechanics
Sem 3

Model Reduction

Lab 4b

Prasad ADHAV

Atul AGRAWAL

1 Introduction

In previous labs, we have developed different PGD solvers depending on the equation we were asked to solve. However, it is possible to create a general function allowing to apply PGD method to any problem with appropriate tensor structure, i.e

$$\mathbf{A}\mathbf{u} = \mathbf{b}$$

with

$$\mathbf{A} = \sum_{k=1}^{N_a} \mathbf{A}_1^k \otimes \cdots \otimes \mathbf{A}_D^k \qquad and \qquad \mathbf{b} = \sum_{k=1}^{N_b} \mathbf{b}_1^k \otimes \cdots \otimes \mathbf{b}_D^k$$

where D is the number of separated variables, usually also known referred as *dimensions*. The formulation is very general and thus allows us creating a PGD function able to solve almost any problem that fits into this structure. Recall that the PGD method allows computing

$$\mathbf{u} = \sum_{k=1}^{N_m} \mathbf{u}_1^k \otimes \cdots \otimes \mathbf{u}_D^k$$

by adding one mode at a time(greedy algorithm). Each mode is computed by solving a non-linear optimization problem, using the alternating directions algorithm. These non-linear optimization problems can be written as follows for one iteration from the lecture 4 notes

$$(\alpha_1 \mathbf{K}_x + \beta_1 \mathbf{M}_x)\phi_x = \gamma_1 \mathbf{f}_x$$

$$(\alpha_2 \mathbf{M}_y + \beta_2 \mathbf{K}_y)\phi_x = \gamma_2 \mathbf{f}_y$$

These equations are used to fill the **AA** cell and **BB** cell.

2 Exercise 1

In exercise 1, a script is to be made to give suitable input arguments to run the function `easy_PGD` in order to solve the following equation:

$$\nabla^2 u = -1 \qquad in]0,2[\times]0,1[$$

$$u = 0 \qquad on \;\; boundaries$$

A mesh is created by dividing x and y in 100 elements, this can be changed by changing the variable N. N is also the input used for number of points on with exact solution is computed. Since, this is the same problem as lab $4a$, the exact solution provided in lab $4a$ is used to compute point wise error discussed in later sections.
Corresponding FE matrices are assembled(mass, conductivity), so as to fill **AA**. These can be found as $\mathbf{K}_x$, $\mathbf{M}_x$, $\mathbf{K}_y$ and $\mathbf{M}_y$. As specified in the problem statement, $\mathbf{M}$ is obtained by setting $op1 = op2 = 0$, and $\mathbf{K}$ is obtained by setting $op1 = op2 = 1$. According to the equation discussed in introduction section, and the specifications in the problem statement, **AA** is filled as follows

$$\mathbf{AA} = \begin{bmatrix} \mathbf{K}_x & \mathbf{M}_x \\ \mathbf{M}_y & \mathbf{K}_y \end{bmatrix}$$

The cell **BB** contains the terms **f**. Accordingly after multiplying `ones` vector, with the discreitized vector **dx** and **dy**. **GG** is a cell similar to **BB**, but it is just empty, in `easy_PGD` this cell **GG** will store previous modes. **Bords** is a cell which contains indices where boundary conditions are to be applied, the are basically the line $x = 0$, $x = 2$, $y = 0$ and $y = 1$. Respective indices are fed to **Bords**. Since now all the inputs are ready, the function `easy_PGD` can be called and we can solve the problem. The solution obtained from this general PGD can be seen in fig 1. The main script for solving the problem can be found in appendix named as `Main_Lab4b_ADHAV_AGARWAL.m`

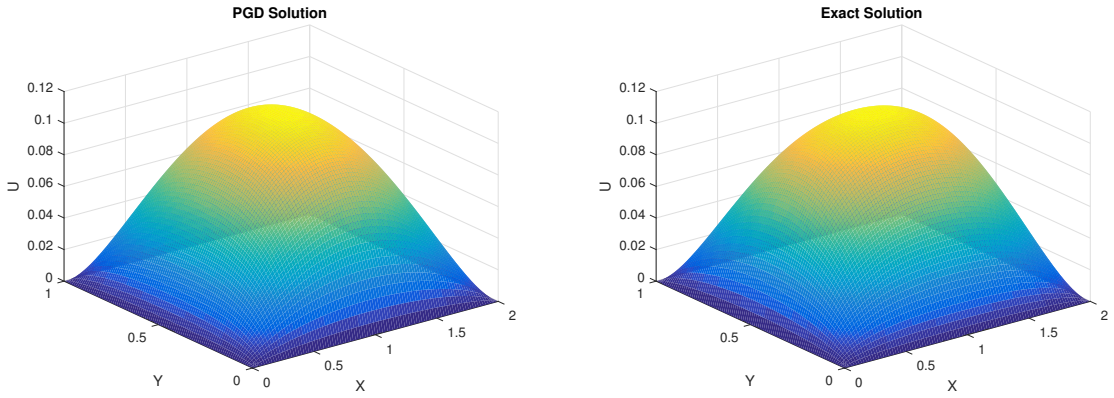

Figure 1: PGD and Exact solution

Since now we have both PGD and Exact solutions, we can compute error. This point-wise error in the domain can be seen in fig 2.

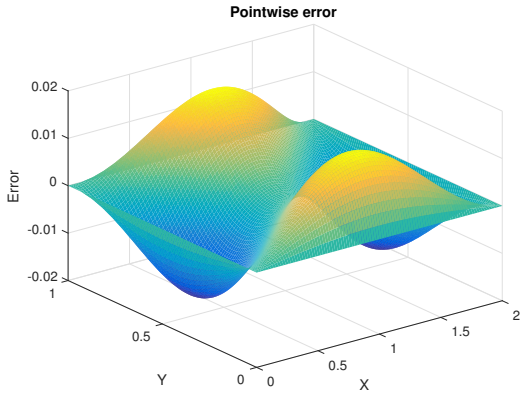

Figure 2: Point-wise error

It can be clearly observed that the solution obtained from the general PGD is same as one obtained in case specific PGD of lab 4*a*. Even, the point-wise error is same for both. Hence, we can conclude that the generalized PGD is working just as good as case specific PGD derived for particular case.

3 Exercise 2

In this exercise `easy_PGD` is to be modified in order to implement the following alternating directions criterion:

$$\|(\mathbf{u}_1 \otimes \cdots \otimes \mathbf{u}_D)_{current} - (\mathbf{u}_1 \otimes \cdots \otimes \mathbf{u}_D)_{previous}\| < \varepsilon_1$$

ε_1 is given as an input argument to `easy_PGD` under the variable name `epsilon1`. In the code, the loop on fixed point iterations the stagnation criterion is implemented. Since we want to built a general PGD formulation and not a case specific PGD, the modes are used. From the lecture notes this stagnation criterion is defined as:

$$\|\phi_x\phi_y - \phi_x^o\phi_y^o\|^2 = (\phi_x, \phi_x)(\phi_y, \phi_y) + (\phi_x^o, \phi_x^o)(\phi_y^o, \phi_y^o) - 2(\phi_x, \phi_x^o)(\phi_y, \phi_y^o)$$

where upper script o denotes old values of ϕ. Here, ϕ are the modes in respective directions. We know from problem statement that $\mathbf{u}$ are modes, and are used to built the solution later on. These ϕ terms could be replaced by $\mathbf{u}$, but in terms of code provided, these terms are stored in cell `RS`, where `RS{1}` has modes for x

dimension and `RS{2}` has modes for y direction. As a standard notation, we know that (ϕ_x, ϕ_x) is integral. Similarly other terms in the stagnation terms are as integrals as well. After rearranging and applying suitable integrals, the MATLab code is given as shown in fig 3. The stagnation condition can be applied in multiple ways. Here two of them are shown.

```
%% Stagnation Criteria
%Trapezoidal

stag1 = (trapz(x,RS{1}))*(trapz(x,RS{1}))*(trapz(y,RS{2}))*(trapz(y,RS{2}))...
    + (trapz(x,RS_old{1}))*(trapz(x,RS_old{1}))*(trapz(y,RS_old{2}))*(trapz(y,RS_old{2})) ...
     - 2*(trapz(x,RS{1}))*(trapz(y,RS{2}))*(trapz(x,RS_old{1}))*(trapz(y,RS_old{2}));
stag1 = sqrt(stag1);

%Rectangular integration
stag1 = sqrt((RS{1}'*RS{1})*(RS{2}'*RS{2}) + (RS_old{1}'*RS_old{1})*...
    (RS_old{2}'*RS_old{2})-2*(RS{1}'*RS_old{1})*(RS{2}'*RS_old{2}));
stag(mode,iter) = stag1;
%if (stag1 < epsilon1), disp('Stagnation'), break; end;
if (stag1 < epsilon1), break; end;
```

Figure 3: Stagnation computation

After applying the stagnation condition, the values of stagnation terms are saved for different modes. Although only first two modes are asked in the assignment, here first 5 modes are presented in the fig 4. This is because since for the computation of first mode the ϕ terms are initialized as random terms, and due to this the algorithm is not able to compute the first mode fast. It can be clearly observed that for first mode after 3 to 4 iterations, the alternating directions algorithm has converged. In the left figure trapezoidal rule is applied and he right figure rectangular rule is applied for integration. It is clear that trapezoidal is faster than rectangular, since the integrals are more accurate compared to rectangular method.

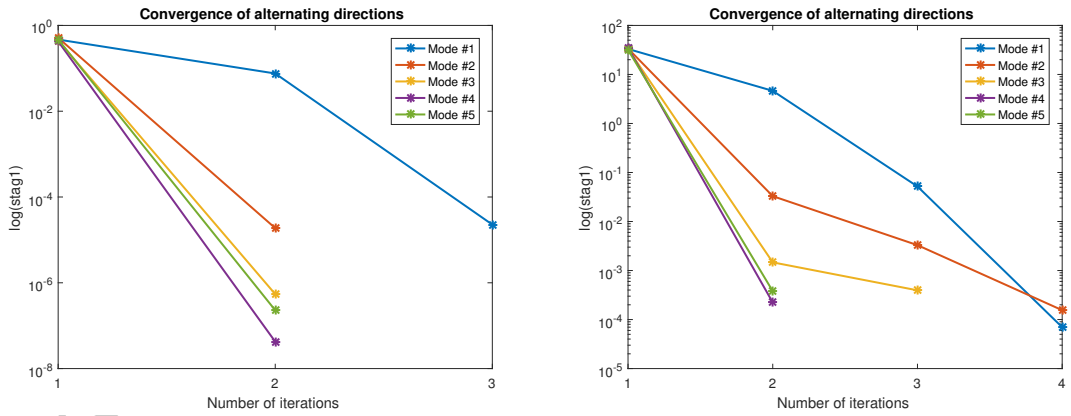

Figure 4: Convergence for alternating directions algorithm

But, since for mode 2 and other higher modes, previous modes are available, they are computed more precisely and satisfy the stagnation condition within 3 to 5 iterations for the given ε_1 value.

4 Exercise 3

Modify `easy_PGD` in order to implement the following global convergence criterion based on the residual norm:

$$\frac{\|\mathbf{b} - \mathbf{Au}\|_{L^2}}{\|\mathbf{b}\|_{L^2}} < \varepsilon_2$$

The variable ε_2 is given as an input argument to `easy_PGD` function. Once we expand the the above $L2$ norms, we will get an expansion as follows for the numerator term (coded as `convgN`)

$$\|\mathbf{b} - \mathbf{Au}\|_{L^2}^2 = \mathbf{b}^2 + (\mathbf{Au})^2 - 2\mathbf{bAu}$$

and the denominator term will be just $\mathbf{b}^2$. The MATLab code implementation for the global convergence is shown in fig 5.

```
%% Global convergence criterion

convgN = ((trapz(x,b)*trapz(x,b)*trapz(y,b)*trapz(y,b))...
        + (trapz(x,K*RS{1})*trapz(x,K*RS{1}))*...
           (trapz(y,K*RS{2})*trapz(y,K*RS{2}))...
      - 2*trapz(x,b)*(trapz(x,K*RS{1})*trapz(y,b)*(trapz(y,K*RS{2}))));
convgD = sqrt(trapz(x,b)*trapz(x,b)*trapz(y,b)*trapz(y,b));
convg1 = convgN/convgD;
convg(mode,iter) = convg1;
if (convg < epsilon2),disp('Global Converged'), break; end;
```

Figure 5: Implementation of $L2$ norm of residual in MATLab

The convergence plot for 10 modes is shown in the fig 6. It can be clearly seen that the convergence is very rapid over the modes. Only for first mode it took, multiple iteration to achieve convergence. The term `convg` is basically the residual provided in the problem statement.

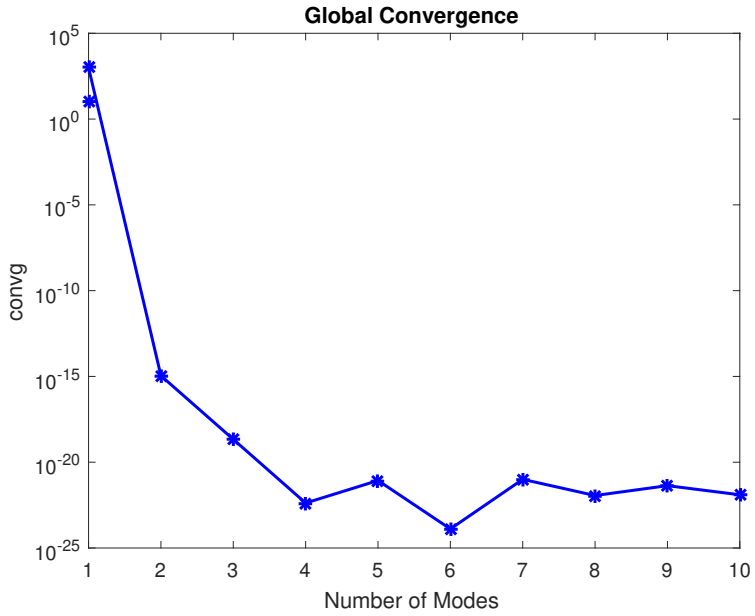

Figure 6: Global convergence

A Main_Lab4b_ADHAV_AGARWAL.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                 ECN                                      %
%                             MoReD Lab 4b                                 %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clc
clear all;
close all;
%% Initialisations & Inputs
X = 2; Y = 1;
N = 100;

dx = X/N;  %Defines mesh size in x directions
dy = Y/N;  %Defines mesh size in y directions

x = linspace(0,X,X/dx+1); % ]0,2[
y = linspace(0,Y,Y/dy+1); % ]0,1[

Nx = X/dx+1;
Ny = Y/dy+1;

Nmodes = 10;
Niter = 1000;

epsilon1 = 1e-03;
epsilon2 = 1e-04;

%% FEM

Mx = FEM_mat_1D(x,0,0); %Mass Matrix
Kx = FEM_mat_1D(x,1,1); %Conductivity Matrix
My = FEM_mat_1D(y,0,0); %Mass Matrix
Ky = FEM_mat_1D(y,1,1); %Conductivity Matrix

%% Creating inputs for easy_PGD
AA{1,1} = Kx;
AA{1,2} = Mx;
AA{2,1} = My;
AA{2,2} = Ky;
BB{1,1} = ones(Nx,1)*dx;
BB{2,1} = ones(Ny,1)*dy;
GG{1} = zeros(Nx,0);
GG{2} = zeros(Ny,0);
Bords{1,1} = [1 Nx];
Bords{2,1} = [1 Ny];

%% Calling Functions to solve

[FF,stag,convg] = easy_PGD(x,y,AA,BB,GG,Nmodes,Niter,Bords,epsilon1,
   epsilon2);
```

```
Uexact = ExactSolution(x,y,N);

%% Post Process

u = FF{1,1}*FF{2,1}'; %Computing solution

% PGD Solution
figure,
surf(x,y,u,'edgecolor','none');
title('PGD Solution');
xlabel('X'); ylabel('Y'); zlabel('U');

% Pointwise error
err = u - Uexact;
figure,
surf(x,y,err,'edgecolor','none');
title('Pointwise error');
xlabel('X'); ylabel('Y'); zlabel('Error');

%Exact solution
figure,
surf(x,y,Uexact,'edgecolor','none');
title('Exact Solution');
xlabel('X'); ylabel('Y'); zlabel('U');

%Convergence for alternating directions algorithm
figure,
for i = 1:5
    loglog(stag(i,:),'Linewidth',1.5)
    hold on
end
title('Convergence of alternating directions')
xlabel('Number of iterations'); ylabel('stag1');
legend('Mode #1', 'Mode #2','Mode #3','Mode #4','Mode #5')

%Global convergence
figure,
for i = 1:5
    semilogy(convg(:,i),'b-*','Linewidth',1.5)
    hold on
end
title('Global Convergence')
xlabel('Number of Modes'); ylabel('convg');
```