Solving PDEs using the finite element method with the Matlab PDE Toolbox

Jing-Rebecca Li[a]

[a]*INRIA Saclay, Equipe DEFI, CMAP, Ecole Polytechnique, Route de Saclay, 91128 Palaiseau Cedex, France*

---

1. Project

1.1. Solving PDE with Neumann boundary conditions

Please write Matlab code to solve for the unknown function ω that is the solution the PDE below that has zero forcing term, zero initial condition, and nonhomogeneous (non-zero) Neumann boundary conditions

$$\begin{aligned}
\frac{\partial}{\partial t}\omega(\mathbf{x},t) - \nabla\left(\mathcal{D}_0 \nabla \omega(\mathbf{x},t)\right) &= 0, & \mathbf{x} \in \Omega, \\
\mathcal{D}_0 \nabla \omega(\mathbf{x},t) \cdot \nu(\mathbf{x}) &= \mathcal{D}_0 F(t) \mathbf{u_g} \cdot \nu(\mathbf{x}), & \mathbf{x} \in \partial\Omega, \\
\omega(\mathbf{x},0) &= 0, & \mathbf{x} \in \Omega,
\end{aligned} \tag{1}$$

ν being the outward normal to the domain Ω. The vector $\mathbf{u_g}$ is a vector in two dimensions and has unit norm. You need to solve this in the time interval

$$t \in [0, TE], \quad TE = \delta + \Delta,$$

where δ and Δ are given constants. The time-dependent function $F(t)$ in the Neumann boundary condition is defined in the following way:

$$F(t) \equiv \int_0^t f(s)ds. \tag{2}$$

Email address: `jingrebecca.li@inria.fr` (Jing-Rebecca Li)

where $f(t)$ is related to δ and Δ by:

$$f(t) = \begin{cases} 1 & 0 < t \leq \delta, \\ 0 & \delta < t \leq \Delta, \\ -1 & \Delta < t \leq \Delta + \delta, \\ 0 & \text{elsewhere}, \end{cases} \tag{3}$$

In particular,

$$F(t) = \begin{cases} t & 0 < t \leq \delta, \\ \delta & 0 + \delta < t \leq \Delta, \\ \Delta + \delta - t & 0 + \Delta < t \leq \Delta + \delta, \\ 0 & \text{elsewhere}. \end{cases} \tag{4}$$

Remark 1. *When you construct the Neumann boundary conditions, please generate the FE matrices without time dependence by writing the function*

```
NeumannBC_notime.m
```

In this function, declare

```
global UG
```

Then add the time dependence to the ODE solver by writing the function

```
odefun_NeumannBC.m
```

Remark 2. *The format in Matlab to define functions for the PDE Toolbox is the following*

```
function [value] = NeumannBC_notime(region,state)
```

where x position is accessed as $region.x$, y position is accessed as $region.y$ and the normal at the position (x, y) is accessed by $(region.nx, region.ny)$. So if you want the function to return xy you would write

```
value = region.x*region.y.
```

Remark 3. *Below is from the documentation of the Matlab Toolbox.*

Partial Differential Equation Toolbox™ solvers pass the region and state data to your function.

- *"region" is a structure containing the following fields. If you pass a name-value pair to applyBoundaryCondition with Vectorized set to 'on', then region can contain several evaluation points. If you do not set Vectorized or use Vectorized,'off', then solvers pass just one evaluation point in each call.*

  region.x — The x-coordinate of the point or points
  region.y — The y-coordinate of the point or points
  region.z — For 3-D geometry, the z-coordinate of the point or points

  Furthermore, if there are Neumann conditions, then solvers pass the following data in the region structure.

  region.nx — x-component of the normal vector at the evaluation point or points
  region.ny — y-component of the normal vector at the evaluation point or points
  region.nz — For 3-D geometry, z-component of the normal vector at the evaluation point or points

- *"state" is for transient or nonlinear problems.*

  state.u contains the solution vector at evaluation points. state.u is an 1-by-M matrix, where each column corresponds to one evaluation point, and M is the number of evaluation points.
  state.time contains the time at evaluation points. state.time is a scalar.

- *Your function returns the boundary condition values. The output has the following form: 1-by-M matrix, where each column corresponds to one evaluation point, and M is the number of evaluation points.*
  If boundary conditions depend on state.u or state.time, ensure that your function returns a matrix of NaN of the correct size when state.u or state.time are NaN. Solvers check whether a problem is nonlinear or time-dependent by passing NaN state values, and looking for returned NaN values.

After you solved for $\omega(\mathbf{x}, t)$, the quantity $h(t)$

$$h(t) = \frac{1}{|\Omega|} \int_\Gamma \omega(\mathbf{y}, t) \left(\mathbf{u_g} \cdot \nu(\mathbf{y})\right) ds_{\mathbf{y}} \tag{5}$$

is computed in the code

```
driver_project.m
```

1.2. Solving PDE with forcing term

Writing ω, which solves the problem (1), as the sum

$$\omega(\mathbf{x},t) = \widetilde{\omega}(\mathbf{x},t) + F(t)\,\mathbf{x}\cdot\mathbf{u_g}, \mathbf{x}\in\Omega, \quad t\in[0,TE] \tag{6}$$

where $\widetilde{\omega}(\mathbf{x},t)$ satisfied the following diffusion equation with a forcing term and homogeneous boundary condition:

$$\frac{\partial}{\partial t}\widetilde{\omega}(\mathbf{x},t) - \nabla\left(\mathcal{D}_0\nabla\widetilde{\omega}(\mathbf{x},t)\right) = -f(t)\mathbf{x}\cdot\mathbf{u_g}, \qquad \mathbf{x}\in\Omega,\ t\in[0,TE], \tag{7}$$

$$\mathcal{D}_0\nabla\widetilde{\omega}(\mathbf{x},t)\cdot\nu(\mathbf{x}) = 0, \qquad \mathbf{x}\in\Gamma,\ t\in[0,TE], \tag{8}$$

$$\widetilde{\omega}(\mathbf{x},t) = 0, \qquad \mathbf{x}\in\Omega, \tag{9}$$

Please compute ω from $\widetilde{\omega}$ after solving the above PDE with a forcing term.

1.3. Solving PDE with eigenfunctions

The function $\widetilde{\omega}(\mathbf{x},t)$ also can be expanded in the basis of Laplace eigenfunctions. Let $\phi_n(\mathbf{x})$ and λ_n be the L^2-normalized eigenfunctions and eigenvalues associated to the Laplace operator with homogeneous Neumann boundary conditions:

$$-\nabla\mathcal{D}_0\left(\nabla\phi_n(\mathbf{x})\right) = \lambda_n\phi_n(\mathbf{x}), \quad \mathbf{x}\in\Omega, \tag{10}$$

$$\mathcal{D}_0\nabla\phi_n(\mathbf{x})\cdot\nu(\mathbf{x}) = 0, \qquad \mathbf{x}\in\Gamma. \tag{11}$$

We can write $\widetilde{\omega}(\mathbf{x},t)$ in the basis of the eigenfunctions as

$$\widetilde{\omega}(\mathbf{x},t) = \sum_{n=1}^{\infty}(-a_n)\phi_n(\mathbf{x})\int_0^t e^{-\lambda_n(t-s)}f(s)ds, \tag{12}$$

where the coefficients are

$$a_n = \int_\Omega \mathbf{x}\cdot\mathbf{u_g}\phi_n(\mathbf{x})d\mathbf{x}. \tag{13}$$

Please solve Eqs 10-11 and compute ω using the formula in Eq. 12.

Remark 4. *To calculate $\phi_n(\mathbf{x})$ and λ_n in Matlab, you need to use the Matlab function pdeeig, with the command:*

```
[V,L]=pdeeig(heatmodel,C_COEFF,0,1,[-inf,3*C_COEFF]);
```

V contains the ϕ_n *and L contains* λ_n*, for* $n = 1, \cdots n_{eig}$*. How many eigenfunctions get computed depends on the interval on which you are looking for eigenvalues. The range I gave is* $\lambda_n \in [-\infty, 3C_{COEFF}]$.

Remark 5. *Before you use Eq. 12, please first (numerically) normalize the eigenfunctions* $\{\phi_n\}$ *to have* L^2*-norm equal to 1, meaning:*

$$\int_\Omega |\phi_n(\mathbf{x})|^2 d\mathbf{x} = 1. \tag{14}$$

The inner product of the functions f and g (for projection and norm operation) is defined by

$$< f, g > \equiv \int_\Omega f(\mathbf{x}) g(\mathbf{x}) d\mathbf{x}. \tag{15}$$

If you have the values of f and g at the nodes, how do you do the above integral? Remember $f(\mathbf{x}) \approx \sum_1^{N_p} f_j \psi_j(\mathbf{x})$*, where* $\psi_j(\mathbf{x})$ *is the* $\mathbb{P}_1$ *finite elements basis function on the node j and* f_j *is the value of f at node j. And* $g(\mathbf{x}) \approx \sum_1^{N_p} g_i \psi_i(\mathbf{x})$*. You want to compute*

$$\int_\Omega \left(\sum_1^{N_p} g_i \psi_i(\mathbf{x}) \right) \left(\sum_1^{N_p} f_j \psi_j(\mathbf{x}) \right) d\mathbf{x}.$$

Remark 6. *Write a function*

```
seqprofile_expintegral.m
```

that integrates

$$\int_0^t e^{-\lambda_n (t-s)} f(s) ds$$

analytically. Break into different cases for $t \in [0, \delta], [\delta, \Delta], [\Delta, \Delta + \delta]$*. The function should take as inputs t and* λ *:*

```
function value = seqprofile_expintegral(time,lambda)
  global BDELTA SDELTA
```

Be sure to treat the case $\lambda = 0$.

1.4. Computing a quantity related to the solution ω

Please compute the quantity $h(t)$ using the first definition given below:

$$h(t) = \frac{1}{|\Omega|} \int_\Omega \mathbf{u_g} \cdot \nabla \omega(\mathbf{x}, t) d\mathbf{x} = \frac{1}{|\Omega|} \int_\Gamma \omega(\mathbf{y}, t) \left(\mathbf{u_g} \cdot \nu(\mathbf{y}) \right) ds_{\mathbf{y}}. \tag{16}$$

and compare with the computed result from using Eq. 5.

1.4.1. Sample output

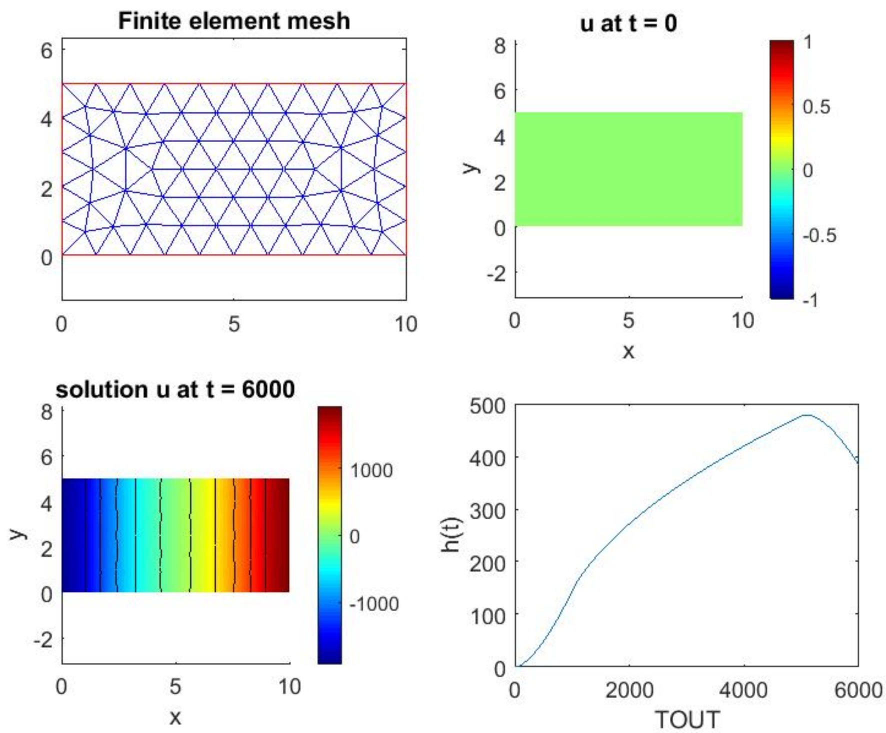

Figure 1: This is an example of the output.

1.5. Questions

1. What should you use as the limits of the eigenvalues in the command

   `[V,L]=pdeeig(heatmodel,C_COEFF,0,1,[-inf,EigLim]);`

   I used

   `[V,L]=pdeeig(heatmodel,C_COEFF,0,1,[-inf,3*C_COEFF]);`

   What can you use if you want to get about 10 eigenvalues? For the rectangle? For the disk? For the ellipse? Hint, where are the eigenvalues for the rectangle,...?

2. Plot the eigenfunctions $\phi_n(\mathbf{x})$ corresponding to the 4 largest $a_n = \int_\Omega \mathbf{x}\cdot\mathbf{u_g}\phi_n(\mathbf{x})d\mathbf{x}$ using the command

```
pdeplot
```

How do the most important eigenfunctions vary according to $\mathbf{u_g}$? Try $\mathbf{u_g} = [0, 1]$ and $\mathbf{u_g} = [1, 0]$.

3. Fix a geometry and δ (SDELTA), increase Δ (BDELTA). How does $h(t)$ change as Δ increases?

1.6. Matlab files

1.6.1. Matlab functions that you should have in your project folder

```
seqprofile.m (given, this is f(t))
seqintprofile.m  (given, this is F(t))
project_inputs.in (given)
read_simulation_parameters.m  (given)
Build_FE_Mesh.m (given)
Plot_Figures.m (given)

Solve_NeumannBC.m (given)

odefun_ForcingTerm.m   (need to write)
odefun_NeumannBC.m  (need to write)
ForcingTerm_notime.m (need to write)
NeumannBC_notime.m  (need to write)
Solve_EigenFunctions.m (need to write)
Solve_ForcingTerm.m (need to write)
```

1.6.2. Files provided to you

There is an input file

```
project_inputs.in
```

containing the necessary input parameters for the code.

```
0               % M_COEFF
1               % D_COEFF
1e-3            % C_COEFF
0               % A_COEFF
```

```
0 1          % UG
2500         % SDELTA_INPUT
3000         % BDELTA_INPUT
1            % SHAPE_INPUT (1=rectangle,2=ellipse)
10 5         % WIDTH, HEIGHT (of rectangle or ellipse)
0.5          % hmax of FE mesh
1e-6         % ODESOLVER_TOL
```

There is a file that reads the input parameters

`read_simulation_parameters.m`

```
function [M_COEFF,D_COEFF,C_COEFF,A_COEFF,UG,SDELTA_input,
   BDELTA_input,...
    shape_input,shape_parameters,hmax,odesolve_tol]...
    = read_simulation_parameters(fname)

fid=fopen(fname);

tline = fgetl(fid);
M_COEFF = sscanf(tline,'%f',1);

tline = fgetl(fid);
D_COEFF = sscanf(tline,'%f',1);

tline = fgetl(fid);
C_COEFF = sscanf(tline,'%f',1);

tline = fgetl(fid);
A_COEFF = sscanf(tline,'%f',1);

tline = fgetl(fid);
UG = sscanf(tline,'%f',2);
UG = UG/norm(UG);

tline = fgetl(fid);
SDELTA_input = sscanf(tline,'%f',1);

```

```
tline = fgetl(fid);
BDELTA_input = sscanf(tline,'%f',1);

tline = fgetl(fid);
shape_input = sscanf(tline,'%f',1);

tline = fgetl(fid);
shape_parameters = sscanf(tline,'%f',2);

tline = fgetl(fid);
hmax = sscanf(tline,'%f',1);

tline = fgetl(fid);
odesolve_tol = sscanf(tline,'%f',1);

fclose(fid);
```

There is a file that constructs the FE mesh

`Build_FE_Mesh.m`

```
function [heatmodel] = Build_FE_Mesh(shape,shape_parameters,hmax)

width = shape_parameters(1);
height = shape_parameters(2);

if (shape == 1)
    % gdm is a matrix to describe the geometry.
    gdm = [3 4 0,width,width,0,0,0,height,height]';
    %gdm = [3 4 -width/2,width/2,width/2,-width/2,-height/2,-height/2,height/2,height/2]';
elseif (shape == 2)
    gdm = [4 0 0 width,height,0]';
end

g = decsg(gdm);

```

```
% Creates PDE model object
heatmodel = createpde();
% Creates PDE model geometry
heatmodel_geom = geometryFromEdges(heatmodel,g);

% generates finite elements mesh
msh = generateMesh(heatmodel,'GeometricOrder','linear','hmax',hmax);
```

There is a file that plots figures

`Plot_Figures.m`

```
function Plot_Figures(heatmodel,TOUT,YOUT,hvec)
figure;
subplot(2,2,1);
pdeplot(heatmodel);
title('Finite element mesh');
axis equal;
subplot(2,2,2);
pdeplot(heatmodel,'XYData',YOUT(1,:)','Contour','on','ColorMap','jet');
title(['u at t = ',num2str(TOUT(1))]);
xlabel('x');
ylabel('y');
axis equal;
subplot(2,2,3);
pdeplot(heatmodel,'XYData',YOUT(end,:)','Contour','on','ColorMap','jet');
title(['solution u at t = ',num2str(TOUT(end))]);
xlabel('x');
ylabel('y');
axis equal;
subplot(2,2,4);
plot(TOUT,hvec);
xlabel('TOUT');
ylabel('h(t)');
```

There is a driver file that calls all the Matlab functions driver_project.m

```
%close all; clear;
global UG;
global BDELTA SDELTA

fname = 'project_inputs.in';

[M_COEFF,D_COEFF,C_COEFF,A_COEFF,UG_input,SDELTA_input,
   BDELTA_input,...
    shape_input,shape_parameters,hmax,odesolve_tol]...
    = read_simulation_parameters(fname);

SDELTA = SDELTA_input;
BDELTA = BDELTA_input;

[heatmodel] = Build_FE_Mesh(shape_input,shape_parameters,
   hmax);

[TOUT,YOUT,G_save,M_save] = Solve_NeumannBC(M_COEFF,
   D_COEFF,C_COEFF,A_COEFF,...
    UG_input,SDELTA_input,BDELTA_input,heatmodel,
       odesolve_tol);

VOL = sum(sum(M_save));

hvec = G_save.'*YOUT'/VOL;

Plot_Figures(heatmodel,TOUT,YOUT,hvec);

[TOUT,YOUT] = Solve_ForcingTerm(M_COEFF,D_COEFF,C_COEFF,
   A_COEFF,...
    UG_input,SDELTA_input,BDELTA_input,heatmodel,
       odesolve_tol);

hvec = G_save.'*YOUT'/VOL;

Plot_Figures(heatmodel,TOUT,YOUT,hvec);
```

```
if (shape_input==2)
    BesselJPrimeRoots;
    Leig_mat = [besseljprimeroots;besseljprimeroots(2:end,:)].^2/shape_parameters(1)/shape_parameters(2);
elseif (shape_input == 1)
    Leig_mat = pi^2*([0:1:10]').^2/shape_parameters(1)^2*ones(1,11)...
        +ones(11,1)*pi^2*([0:1:10]).^2/shape_parameters(2)^2;
end
numroots = prod(size(Leig_mat));
L_exact = reshape((Leig_mat*C_COEFF)',[numroots,1]);
[L_sort,L_index]=sort(L_exact,'ascend');
if (shape_input == 2)
    L1 = [0;L_sort];
else
    L1 = L_sort;
end
neig = min(40,length(L1)-1);
EigLim = (L1(neig)+L1(neig+1))/2;

[TOUT,YOUT,V,L,proju0] = Solve_EigenFunctions(M_COEFF,D_COEFF,C_COEFF,A_COEFF,...
    UG_input,SDELTA_input,BDELTA_input,heatmodel,EigLim);

hvec = G_save.'*YOUT'/VOL;

Plot_Figures(heatmodel,TOUT,YOUT,hvec);

L2 = L;
Stmp = min(length(L1),length(L2));

figure; hold on; plot(L1(1:Stmp),'x'); plot(L2(1:Stmp),'o');
figure; plot(proju0,'o');

PU0_max = max(abs(proju0));
```

```
PU0_ind = find(abs(proju0)>=0.001*PU0_max);

figure;
for ieig = 1:min(12,length(PU0_ind))
    figure;
    %subplot(1,1,ieig);
    pdeplot(heatmodel,'XYData',V(:,PU0_ind(ieig)),'Contour','on','ColorMap','jet');
    title(['ef',mynum2str(PU0_ind(ieig),1),',ev=',mynum2str(L(PU0_ind(ieig)),1),...
        ',an=',mynum2str(abs(proju0(PU0_ind(ieig))),2)]);
    xlabel('x');
    ylabel('y');
    axis equal;
end
```

There is the function that solves the PDE in Eq. 1

`Solve_NeumannBC.m`

```
function [TOUT,YOUT,G_save,M_save] = Solve_NeumannBC(M_COEFF,D_COEFF,C_COEFF,A_COEFF,...
    UG_input,SDELTA_input,BDELTA_input,heatmodel,odesolve_tol)

global FEM_M FEM_K FEM_A FEM_Q FEM_G FEM_F
global UG;
global BDELTA SDELTA

SDELTA = SDELTA_input;
BDELTA = BDELTA_input;
UG = UG_input;

% set PDE coefficients
specifyCoefficients(heatmodel,'m',M_COEFF,'d',D_COEFF,'c',C_COEFF,'a',A_COEFF,'f',0);
NumEdges = heatmodel.Geometry.NumEdges;
% set boundary conditions
```

```
for ie = 1:NumEdges
    applyBoundaryCondition(heatmodel,'neumann','Edge',ie, ...
        'g',@NeumannBC_notime,'q',0,'Vectorized','on');
end
% assemble the 6 finite elements matrices
model_FEM_matrices = assembleFEMatrices(heatmodel);
FEM_M = model_FEM_matrices.M;
FEM_K = model_FEM_matrices.K;
FEM_A = model_FEM_matrices.A;
FEM_Q = model_FEM_matrices.Q;
FEM_G = C_COEFF*model_FEM_matrices.G;
FEM_F = model_FEM_matrices.F;

G_save = model_FEM_matrices.G;

M_save = model_FEM_matrices.M/D_COEFF;


% set time of simulation
startTime = 0;
endTime = BDELTA+SDELTA;

tlist = [startTime,endTime];

% This evaluates Initial Condition
u0 = zeros(1,size(FEM_M,1));

options = odeset('Mass',FEM_M,'AbsTol',odesolve_tol,'RelTol',odesolve_tol,'Stats','on');

disp('ode23t');

tic
  [TOUT,YOUT] = ode23t(@odefun_NeumannBC,tlist,u0',options);
toc
```