Solving PDEs using the finite element method with the Matlab PDE Toolbox
September 30 - October 11, 2019

Jing-Rebecca Li[a]

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

---

1. Classification of second order partial differential equations

Consider the following partial differential equation (PDE)

$$\sum_{i=1}^{n}\sum_{j=1}^{n} k_{ij}\frac{\partial^2 u}{\partial x_i \partial x_j} + \sum_{i}^{n} h_i \frac{\partial u}{\partial x_i} + lu + r = 0, \tag{1}$$

where the coefficients k_{ij}, h_i, l, r are functions of $\mathbf{x} = \{x_1, x_2, \cdots, x_n\}$, $u = u(x_1, x_2, \cdots, x_n)$ and n is the number of independent variables.

Put the coefficients $\{k_{ij}\}$ into a matrix:

$$K \equiv \begin{bmatrix} k_{11} \cdots k_{1n} \\ \vdots \ddots \vdots \\ k_{n1} \cdots k_{nn} \end{bmatrix}.$$

Assume the matrix K is symmetric. If K is not symmetric, then the PDE in Eq. 1 has to be transformed first to get a symmetric coefficient matrix before classifying the PDE.

The classification of second-order equations in n variables is the following:

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

- the PDE is *elliptic* if all eigenvalues $\lambda_1, \cdots, \lambda_n$ of K are non-zero and have the same sign.
- the PDE is *hyperbolic* if all eigenvalues of K are non-zero and have the same sign except for one of the eigenvalues.
- the PDE is *parabolic* if any of the eigenvalues of K is zero.

1.1. Important examples

- the Laplace PDE :

$$u_{xx}(x,y,z) + u_{yy}(x,y,z) + u_{zz}(x,y,z) - f(x,y,z) = 0.$$

- the heat PDE :

$$u_t(x,y,z,t) - c\left(u_{xx}(x,y,z) + u_{yy}(x,y,z) + u_{zz}(x,y,z)\right) - f(x,y,z) = 0.$$

- heat wave PDE :

$$u_{tt}(x,y,z,t) - c\left(u_{xx}(x,y,z) + u_{yy}(x,y,z) + u_{zz}(x,y,z)\right) - f(x,y,z) = 0.$$

2. Notations for partial differential equations

The Matlab PDE Toolbox can solve a partial differential equation of the form

$$m\frac{\partial^2 u}{\partial t^2} + d\frac{\partial u}{\partial t} - \nabla \cdot (c\nabla u) + au = f. \tag{2}$$

The coefficients m, d, c, a, and f can be functions of location (x,y, and in 3 dimensions, z) and they can be functions of the solution u or its gradient.

2.1. Gradient and Laplacian operators

Definition 1. *The gradient operator acting on a function $f(x,y,z)$ produces a vector of functions*

$$\nabla f \equiv \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}\right] \in \mathbb{R}^3.$$

The above quantity is called the gradient of f, pronounced "grad f".

Definition 2. *The Laplacian operator acting on $f(x, y, z)$ produces the function*

$$\nabla \cdot \nabla f \equiv \left[\frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} + \frac{\partial^2 f}{\partial z^2} \right] \in \mathbb{R}.$$

More generally:

$$\nabla \cdot c \nabla f \equiv \left[\frac{\partial \left(c \frac{\partial f}{\partial x} \right)}{\partial x} + \frac{\partial \left(c \frac{\partial f}{\partial y} \right)}{\partial y} + \frac{\partial \left(c \frac{\partial f}{\partial z} \right)}{\partial z} + \right] \in \mathbb{R}.$$

The above quantities are both called the Laplacian of f and c is called the diffusion coefficient.

2.2. Domain where the PDE is defined

Typically, the solution $u(x, y, z, t)$ is not sought everywhere in space, rather, we look for the values of u only on a subset of $\mathbb{R}^{dim}$, in other words, only for $(x, y, z) \in \Omega$, where Ω is a domain in $\mathbb{R}^{dim}$. For Ω with boundary $\Gamma = \partial\Omega$, the PDE in Eq. 2 needs to be supplemented by boundary conditions on Γ.

2.3. Boundary conditions of the PDE

A very useful set of boundary conditions that the Matlab PDE Toolbox can treat are Neumann boundary conditions of the form:

$$(c \nabla u) \cdot \mathbf{n} + qu = g, \quad (x, y, z) \in \Gamma, \tag{3}$$

where $\mathbf{n}$ is the unit outward-pointing normal to Ω. This means $\mathbf{n}$ is a vector in $\mathbb{R}^{dim}$ and it has norm 1.

Remark 1. *The term* **Neumann boundary condition** *means the condition involves the value of the gradient of the solution on the boundary. The term* **Dirichlet boundary condition** *means that the condition involves the value of the solution itself on the boundary.*

Remark 2. *A* **boundary condition** *is given on the boundary for all time that the PDE is defined.*

2.4. Initial conditions of the PDE

If $m \neq 0$, then Eq. 2 is the wave equation (you should check first that the coefficients of the PDE give a hyperbolic equation). In this case, there need to be initial conditions of the form:

$$\begin{aligned} u(x, y, z, 0) &= w(x, y, z), \\ u_t(x, y, z, 0) &= v(x, y, z). \end{aligned} \tag{4}$$

If $m = 0$ and $d \neq 0$ then Eq. 2 is the heat equation, also called the diffusion equation (after you check the coefficients of the PDE gave a parabolic equation). In this case, there need to be initial conditions of the form:

$$u(x, y, z, 0) = w(x, y, z), \tag{5}$$

to supplement to PDE and the boundary conditions.

Remark 3. *An* **initial condition** *is given on the domain for one point in time (at the initial time).*

In summary, when the PDE is defined on a domain Ω with the boundary Γ, boundary conditions on Γ need to be imposed, in addition to the PDE. When the PDE has time derivatives, then initial conditions need to be imposed. When there are only first order time derivatives, initial conditions on the value of the solution need to be imposed. When there are second order time derivatives, initial conditions on the value of the solution and the value of the time derivative of the solution need to be imposed.

3. Solving PDEs numerically

- The Matlab PDE Toolbox uses the finite element method (FEM) to discretize in space.
- For time-dependent problems, the PDE is first discretized in space to get a semi-discretized system of equations that has one or more time derivatives.
- The semi-discretized system of equations is solved using one of the ODE solvers available in Matlab.

To explain the FEM, it is easier to look at the PDE without the time derivative terms. So we take Eq. 2 and remove the terms that have time derivatives to obtain:

$$-\nabla \cdot (c\nabla u) + au = f. \tag{6}$$

Without time derivatives, the solution of the above equation does not have time dependence, so we call the above PDE a steady-state problem.

We recall the definitions of the gradient of a function u

$$\nabla u \equiv \left(\frac{\partial}{\partial x}, \frac{\partial}{\partial y}, \frac{\partial}{\partial z}\right) u \equiv \left(\frac{\partial u}{\partial x}, \frac{\partial u}{\partial y}, \frac{\partial u}{\partial z}\right)$$

and the Laplacian operator

$$\begin{aligned} -\nabla \cdot (c\nabla u) &\equiv -\left(\frac{\partial}{\partial x}, \frac{\partial}{\partial y}, \frac{\partial}{\partial z}\right) \cdot c \left(\frac{\partial u}{\partial x}, \frac{\partial u}{\partial y}, \frac{\partial u}{\partial z}\right) \\ &\equiv -\left(\frac{\partial}{\partial x}\left(c\frac{\partial u}{\partial x}\right) + \frac{\partial}{\partial y}\left(c\frac{\partial u}{\partial y}\right) + \frac{\partial}{\partial z}\left(c\frac{\partial u}{\partial z}\right)\right). \end{aligned}$$

The PDE in Eq. 6 has second order derivatives in space and it is called the strong formulation.

3.1. Weak formulation of PDE

The FEM does not solve the strong formulation in Eq. 6, rather, it solves a weak formulation of the PDE, where the solution u only needs to have one spatial derivative $(\frac{\partial}{\partial x}, \frac{\partial}{\partial y}, \frac{\partial}{\partial z})$ instead of two $(\frac{\partial^2}{\partial x^2}, \frac{\partial^2}{\partial y^2}, \frac{\partial^2}{\partial z^2})$. This is done by moving one of the spatial derivatives of u onto test functions v using Green's identity.

The weak formulation is obtained in the following way. We take the strong formulation and multiple it by a test function v and integrate over Ω.

$$\int_\Omega -\nabla \cdot (c\nabla u)\, v \; d\mathbf{x} + \int_\Omega a\, u\, v \; d\mathbf{x} = \int_\Omega f\, v \; d\mathbf{x}. \tag{7}$$

The notation $d\mathbf{x}$ indicates volume integration in the domain Ω. In three dimensions, we would have $d\mathbf{x} = dx\, dy\, dz$.

Theorem 1. *Green's identity relates the following quantities:*

$$\int_\Omega -\nabla \cdot (c\nabla u)\, v \; d\mathbf{x} = \int_\Omega (c\nabla u) \cdot \nabla v \; d\mathbf{x} - \int_{\partial\Omega} (c\nabla u) \cdot \mathbf{n}\, v \; d\mathbf{s}, \tag{8}$$

where $d\mathbf{s}$ indicates surface integration, over the boundary of the domain: $\Gamma = \partial\Omega$. This is in contrast to the volume integration indicated by $d\mathbf{x}$, over the domain Ω.

We replace the first term of Eq. 7 by the right hand side of Eq. 8 to obtain :

$$\int_\Omega (c\nabla u)\cdot\nabla v\ d\mathbf{x} - \int_{\partial\Omega}(c\nabla u)\cdot\mathbf{n}\ v\ d\mathbf{s}, + \int_\Omega a\,u\,v\ d\mathbf{x} = \int_\Omega f\,v\ d\mathbf{x}. \tag{9}$$

Since the solution u must satisfy the boundary conditions in Eq. 3 we obtain

$$\int_\Omega (c\nabla u)\cdot\nabla v\ d\mathbf{x} - \int_{\partial\Omega}(g-qu)\ v\ d\mathbf{s} + \int_\Omega a\,u\,v\ d\mathbf{x} = \int_\Omega f\,v\ d\mathbf{x}. \tag{10}$$

Putting all terms containing u on the left hand side and the other terms on the right hand side :

$$\int_\Omega (c\nabla u)\cdot\nabla v\ d\mathbf{x} + \int_{\partial\Omega} q\ u\ v\ d\mathbf{s} + \int_\Omega a\,u\,v\ d\mathbf{x} = \int_\Omega f\,v\ d\mathbf{x} + \int_{\partial\Omega} g\ v\ d\mathbf{s}. \tag{11}$$

3.2. Finite elements

Now it remains to choose functional spaces for u and v. The functional spaces are closely related to the discretization of Ω into the union of little geometrical pieces called finite elements. The most common finite elements used in practice are triangles in $\mathbb{R}^2$ and tetrahedra in $\mathbb{R}^3$.

The domain Ω will be approximated by the union of N_T elements:

$$\Omega \approx \mathcal{T}^h \equiv \bigcup_{i=1}^{N_T} T_i,$$

where T_i is the ith element (each T_i is triangle in 2 dimensions or tetrahedron in 3 dimensions). The union of the elements is called $\mathcal{T}^h$, and it is the finite element mesh for the domain Ω. The number h indicates the size of the elements. So it's possible to have several meshes of different sizes, where h and N_T are different, for Ω. The smaller the size of the elements, the more accurate the approximate solution u.

The nodes or points in the finite element mesh $\mathcal{T}^h$ is the union of all the vertices in the elements. In 2 dimensions, there are 3 vertices in each element, in 3 dimensions, there are 4 vertices in each element. However, it should be clear that the number of nodes is a lot fewer than $3N_T$ or $4N_T$ because the elements touch each other, so the same node can belong to several elements. Let v_i^1, v_i^2, v_i^3 (in 2 dimensions) or $v_i^1, v_i^2, v_i^3, v_i^4$ (in 3 dimensions) be the vertices of the element T_i, then the set of nodes is

$$\{P_1, P_2, \cdots, P_{N_p}\} = \bigcup\{v_i^1, \cdots, v_i^k\}, \quad i = 1\cdots N_T,\ k = 3 \text{ or } 4.$$

The number N_p is the total number of nodes in $\mathcal{T}^h$.

3.3. More about $\mathbb{P}_1$ elements

The simplest function space for u and v that we will use is the space $\mathbb{P}_1$, which is the space of globally continuous piecewise polynomials of degree 1. This space has a set of basis functions

$$\phi_j(x, y, z), j = 1 \cdots N_P,$$

where N_p is the number of nodes, as explained above.

The basis function $\phi_j(x, y, z)$ is the linear function in x,y,z and has the following properties:

$$\phi_j(x, y, z) = \begin{cases} 0 & \text{on } T_i \text{ if } P_j \notin \{v_i^1, \cdots, v_i^k\}, \\ a_j^i x + b_j^i y + c_j^i z + d_j^i & \text{on } T_i \text{ if } P_j \in \{v_i^1, \cdots, v_i^k\}. \end{cases} \tag{12}$$

To obtain the coefficients of the polynomial, a_j^i, b_j^i, c_j^i, d_j^i on T_i, the following 4 constraints are imposes on the vertices of T_i:

$$\phi_j(x, y, z) = \begin{cases} 1 & \text{if } (x, y, z) = P_j \\ 0 & \text{if } (x, y, z) \neq P_j \text{ and } (x, y, z) \in \{v_i^1, \cdots, v_i^k\}. \end{cases} \tag{13}$$

In short, to obtain the basis function $\phi_j(x, y, z)$ we have to find all the triangles T_i for which P_j is a vertex and find the coefficients a_j^i, b_j^i, c_j^i, d_j^i that define ϕ_j on the element T_i by imposing the constraints in Eq. 13. The support of ϕ_j is small, as can be seen in Eq. 12, ϕ_j is only non-zero on elements for which P_j is a vertex. On the vast majority of elements, ϕ_j is identically zero.

The function space with which we will work is the space spanned by the basis functions ϕ_j:

$$\mathcal{U} = \left\{ f(x, y, z) = \sum_{j=1}^{N_p} f_j \, \phi_j(x, y, z), \quad f_j \in \mathbb{R} \right\}.$$

It is an easy exercise to show that for the choice of ϕ_j described above, the coefficient f_j is just the value of f on the node P_j:

$$f(P_j) = f_j.$$

This is a very useful and convenient property of the choice of the finite element function space we have chosen.

4. Discretization in space

Having described the function space $\mathcal{U}$, then we suppose that we seek an approximate solution to the PDE that belongs to this function space. So the approximate solution will have the form

$$u^h(x,y,z) = \sum_{j=1}^{N_p} U_j \; \phi_j(x,y,z), \quad U_j \in \mathbb{R}. \tag{14}$$

The superscript h reminds us of the underlying finite element mesh $\mathcal{T}^h$ on which the basis functions on defined. To find the approximate solution u^h we just need to find the coefficients U_j. And since U_j coincides with the value of the function at the node P_j we obtain at the same time the values of the approximate solution at the finite element mesh nodes.

Now we take Eq. 11 and plug in the approximate solution u^h:

$$\int_\Omega \left(c\nabla u^h\right)\cdot\nabla v \; d\mathbf{x} + \int_{\partial\Omega} q \; u^h \; v \; d\mathbf{s} + \int_\Omega a \, u^h \, v \; d\mathbf{x} = \int_\Omega f \, v \; d\mathbf{x} + \int_{\partial\Omega} g \; v \; d\mathbf{s} \tag{15}$$

to get

$$\begin{aligned} \sum_{j=1}^{N_p} U_j \int_\Omega (c\nabla \; \phi_j)\cdot\nabla v \; d\mathbf{x} + \sum_{j=1}^{N_p} U_j \; \int_{\partial\Omega} q \; \phi_j \; v \; d\mathbf{s} \\ + \sum_{j=1}^{N_p} U_j \; \int_\Omega a \, \phi_j \, v \; d\mathbf{x} = \int_\Omega f \, v \; d\mathbf{x} + \int_{\partial\Omega} g \; v \; d\mathbf{s}. \end{aligned} \tag{16}$$

The above is a constraint the approximate solution must satisfy. Since there are N_p unknown coefficients, we need N_p constraint equations. These equations come from choosing v to be each of the basis functions $\phi_i, \; i = 1 \cdots, N_p$. The N_p constraint equations are:

$$\begin{aligned} \sum_{j=1}^{N_p} U_j \int_\Omega (c\nabla \; \phi_j)\cdot\nabla\phi_i \; d\mathbf{x} + \sum_{j=1}^{N_p} U_j \; \int_{\partial\Omega} q \; \phi_j \; \phi_i \; d\mathbf{s} \\ + \sum_{j=1}^{N_p} U_j \; \int_\Omega a \, \phi_j \, \phi_i \; d\mathbf{x} = \int_\Omega f \, \phi_i \; d\mathbf{x} + \int_{\partial\Omega} g \; \phi_i \; d\mathbf{s}. \end{aligned} \quad , \quad i = 1, \cdots, N_p. \tag{17}$$

Hence, we have N_p unknowns and N_p equations above, which will give a unique solution $U_1, \cdots, U_{N_p}$.

We now proceed to write Eq. 17 in matrix form by defining the following finite element matrices and vectors:

$$K_{ij} \equiv \int_\Omega (c\nabla\ \phi_j) \cdot \nabla\phi_i\ d\mathbf{x}, \quad i = 1, \cdots, N_p,\ j = 1, \cdots, N_p, \tag{18}$$

$$Q_{ij} \equiv \int_{\partial\Omega} q\ \phi_j\ \phi_i\ d\mathbf{s}, \qquad i = 1, \cdots, N_p,\ j = 1, \cdots, N_p, \tag{19}$$

$$A_{ij} \equiv \int_\Omega a\ \phi_j\ \phi_i\ d\mathbf{x}, \qquad i = 1, \cdots, N_p,\ j = 1, \cdots, N_p, \tag{20}$$

$$F_i \equiv \int_\Omega f\ \phi_i\ d\mathbf{x}, \qquad i = 1, \cdots, N_p, \tag{21}$$

$$G_i \equiv \int_{\partial\Omega} g\ \phi_i\ d\mathbf{s}, \qquad i = 1, \cdots, N_p. \tag{22}$$

The matrix K is called the stiffness matrix, A and Q are matrices, F and G are colume vectors of length N_p. The matrix form of Eq. 17 is then :

$$KU + AU + QU = F + G, \quad U = \begin{bmatrix} U_1 \\ \vdots \\ U_{N_p} \end{bmatrix}. \tag{23}$$

The function u^h is an approximate solution to the steady-state PDE that we started with in this section, Eq. 6.

5. Time stepping of FEM matrix equations using ODE solvers

Now we go back to the PDE in Eq. 2 that has has time derivative terms. We make a slight change to the form of the approximate solution u^h, instead of U_j being constants (numbers), we make U_j functions of time:

$$u^h(x, y, z, t) = \sum_{j=1}^{N_p} U_j(t)\ \phi_j(x, y, z). \tag{24}$$

We will assume that $U_j(t)$ has two continuous time derivatives if $m \neq 0$ and $d = 0$ (the wave equation) and it has one continuous derivative in time if $m = 0$ and $d \neq 0$. This just means that the time derivatives make sense in the formulation in Eq. 2. Since $\phi_j(x, y, z)$ does not have any dependence on time, then much of the derivation of Eq. 23 can be reused.

5.1. Wave PDE: second order ODE in time

For the wave equation ($m \neq 0, d = 0$) we have the following time-dependent matrix equations :

$$M\frac{\partial^2 U}{\partial t^2} + KU + AU + QU = F + G, \quad U = \begin{bmatrix} U_1 \\ \vdots \\ U_{N_p} \end{bmatrix}, \tag{25}$$

where

$$M_{ij} \equiv \int_\Omega m\,\phi_j\,\phi_i\,d\mathbf{x}, \quad i = 1, \cdots, N_p,\ j = 1, \cdots, N_p. \tag{26}$$

5.2. Heat equation: first order ODE in time

For the heat or diffusion equation ($m = 0, d \neq 0$) we have the following time-dependent matrix equations :

$$M\frac{\partial U}{\partial t} + KU + AU + QU = F + G, \quad U = \begin{bmatrix} U_1 \\ \vdots \\ U_{N_p} \end{bmatrix}, \tag{27}$$

where

$$M_{ij} \equiv \int_\Omega d\,\phi_j\,\phi_i\,d\mathbf{x}, \quad i = 1, \cdots, N_p,\ j = 1, \cdots, N_p. \tag{28}$$

The matrix M is called the mass matrix.

5.3. Calling a Matlab ODE solver with initial conditions

The Matlab ODE solver routines can be used to solve Eq. 27 or Eq. 28 to obtain approximations to $U_j(t),\ j = 1, \cdots N_p$. The initial conditions to be passed into the ODE solvers will come from Eq. 4 or Eq. 5. For the wave equation, the initial conditions are :

$$U_j(0) = w(P_j),\ \frac{\partial U_j}{\partial t}(0) = v(P_j), \quad j = 1, \cdots, N_p.$$

For the heat equation, the initial conditions are :

$$U_j(0) = w(P_j), \quad j = 1, \cdots, N_p.$$

6. Analytical solutions of PDEs

6.1. Fundamental solutions

Fundamental solutions solve the PDE in free space ($\mathbb{R}^{dim}$) with the δ function initial condition. They can be used to generate solutions for arbitray initial conditions and forcing terms in forms of convolutions.

Definition 3. *A convolution of two function f and g is the function (of $\mathbf{x}$):*

$$f \star g(\mathbf{x}) \equiv \int_{\mathbb{R}^{dim}} f(\mathbf{y}) g(\mathbf{x}-\mathbf{y}) d\mathbf{y}.$$

Convolution can be thought of as an averaging process, in which $f(\mathbf{x})$ is replaced by the "averaged value" of $f(\mathbf{x})$ relative to the "profile" function $g(\mathbf{x})$.

Theorem 2. *The convolution operator is commutative:*

$$(f \star g)(\mathbf{x}) = (g \star f)(\mathbf{x}),$$

and associative:

$$f \star (g \star h)(\mathbf{x}) = (f \star g) \star h.$$

6.1.1. Heat equation

The heat equation in free space with the forcing term $F(\mathbf{x}, t)$:

$$\frac{\partial u}{\partial t} - \sigma \Delta u = F(\mathbf{x}, t), \quad \mathbf{x} \in \mathbb{R}^{dim}, \tag{29}$$

where the coefficient σ is a positive constant and $\Delta = \nabla \cdot \nabla$ is the Laplacian operator, subject to the initial condition,

$$u(\mathbf{x}, 0) = IC(\mathbf{x}), \quad \mathbf{x} \in \mathbb{R}^{dim} \tag{30}$$

has solution

$$u(\mathbf{x}, t) = \int_{\mathbb{R}^{dim}} IC(\mathbf{y}) G(\mathbf{x}-\mathbf{y}, t) d\mathbf{y} + \int_0^t \int_{\mathbb{R}^{dim}} F(\mathbf{y}, \tau) G(\mathbf{x}-\mathbf{y}, t-\tau) d\mathbf{y} d\tau, \tag{31}$$

where the fundamental solution for the heat equation is:

$$G(\mathbf{x}, t) = \frac{1}{(4\pi\sigma t)^{dim/2}} e^{-\|\mathbf{x}\|^2/(4\sigma t)} \tag{32}$$

The *dim* can be 1,2 or 3.

6.1.2. Wave equation

The wave equation in free space with the forcing term $F(\mathbf{x}, t)$:

$$\frac{\partial u}{\partial t} - c^2 \Delta u = F(\mathbf{x}, t), \quad \mathbf{x} \in \mathbb{R}^{dim}, \tag{33}$$

where the wave speed c is a positive constant and $\Delta = \nabla \cdot \nabla$ is the Laplacian operator, subject to the initial conditions,

$$\begin{aligned} u(\mathbf{x}, 0) &= IC(\mathbf{x}), \quad \mathbf{x} \in \mathbb{R}^{dim}, \\ u_t(\mathbf{x}, 0) &= IT(\mathbf{x}), \quad \mathbf{x} \in \mathbb{R}^{dim}, \end{aligned} \tag{34}$$

has solution

$$\begin{aligned} &\frac{\partial}{\partial t} \int_{\mathbb{R}^{dim}} G(\mathbf{x} - \mathbf{y}, t) IC(\mathbf{y}) d\mathbf{y} + \int_{\mathbb{R}^{dim}} G(\mathbf{x} - \mathbf{y}, t) IT(\mathbf{y}) d\mathbf{y} \\ &+ \int_{\mathbb{R}^{dim}} \int_0^t G(\mathbf{x} - \mathbf{y}, t - \tau) F(\mathbf{y}, \tau) d\tau d\mathbf{y} \end{aligned} \tag{35}$$

where in 3 dimensions:

$$G(\mathbf{x}, t) = \frac{1}{4\pi c \|\mathbf{x}\|} \delta\left((ct) - \|\mathbf{x}\|\right), \tag{36}$$

in 2 dimensions:

$$G(\mathbf{x}, t) = \begin{cases} \frac{1}{2\pi c} \frac{1}{\sqrt{(ct)^2 - \|\mathbf{x}\|^2}}, & \|\mathbf{x}\| < (ct) \\ 0 & \text{otherwise} \end{cases} \tag{37}$$

in 1 dimension :

$$G(\mathbf{x}, t) = \begin{cases} \frac{1}{2c}, & \|\mathbf{x}\| < (ct) \\ 0 & \text{otherwise} \end{cases} \tag{38}$$

We show the solution that matches IC is related to the solution that matches IT.

Theorem 3. *Let v_{IT} denote the solution to the problem*

$$\begin{aligned} u_{tt} &= c^2 u, \quad \mathbf{x} \in \mathbb{R}^{dim}, \\ u(\mathbf{x}, 0) &= 0, \\ u_t(\mathbf{x}, 0) &= IT(\mathbf{x}). \end{aligned}$$

Then the function $w \equiv \frac{\partial}{\partial t} v_{IC}$ solves

$$\begin{aligned} u_{tt} &= c^2 u, \quad \mathbf{x} \in \mathbb{R}^{dim}, \\ u(\mathbf{x}, 0) &= IC(\mathbf{x}), \\ u_t(\mathbf{x}, 0) &= 0. \end{aligned}$$

More concretely, in one dimension

$$\begin{aligned} u(x,t) &= \frac{1}{2}\left(IC(x+ct)+IC(x-ct)\right) \\ &+ \frac{1}{2c}\int_{x-ct}^{x+ct} IT(y)dy \\ &+ \frac{1}{2c}\int_{\tau=0}^{t}\int_{x-c(t-\tau)}^{x+c(t-\tau)} F(y,\tau)dyd\tau \end{aligned}$$

In two dimensions

$$\begin{aligned} u(\mathbf{x},t) &= \frac{\partial}{\partial t}\left(\frac{1}{2\pi c}\int_{\|\mathbf{x}-\mathbf{y}\|\le ct}\frac{IC(\mathbf{y})}{\sqrt{c^2t^2-\|\mathbf{y}-\mathbf{x}\|^2}}d\mathbf{y}\right) \\ &+ \frac{1}{2\pi c}\int_{\|\mathbf{x}-\mathbf{y}\|\le ct}\frac{IT(\mathbf{y})}{\sqrt{c^2t^2-\|\mathbf{y}-\mathbf{x}\|^2}}d\mathbf{y} \\ &+ \frac{1}{2\pi c}\int_{\tau=0}^{t}\int_{\|\mathbf{x}-\mathbf{y}\|\le c(t-\tau)}\frac{F(\mathbf{y},\tau)}{\sqrt{c^2(t-\tau)^2-\|\mathbf{y}-\mathbf{x}\|^2}}d\mathbf{y}d\tau \end{aligned}$$

6.2. Green's functions

Green's functions are like fundamental solutions, with the added boundary conditions.

6.2.1. Heat equation

In one dimension, $\Omega = [0,l]$, given the homogeneous Neumann boundary condition

$$\frac{\partial u}{\partial x} = 0, \quad x = \{0,l\},$$

then

$$u(x,t) = \int_0^l IC(y)G(x,y,t)dy + \int_0^t\int_0^l F(y,\tau)G(x,y,t-\tau)dyd\tau,$$

where the Green's function has two representations:

$$\begin{aligned} G(x,y,t) &= \frac{1}{l} + \frac{2}{l}\sum_{n=1}^{\infty}\cos\frac{n\pi x}{l}\cos\frac{n\pi y}{l}e^{-\frac{\sigma n^2\pi^2 t}{l^2}} \\ &= \frac{1}{2\sqrt{\pi\sigma t}}\sum_{n=-\infty}^{\infty} e^{-\frac{(x-y+2nl)^2}{4\sigma t}} + e^{-\frac{(x+y+2nl)^2}{4\sigma t}} \end{aligned}$$

The first series converges fast for large t, the second series converges fast for small t.

6.2.2. Wave equation

In one dimension, $\Omega = [0, l]$, given the homogeneous Neumann boundary condition

$$\frac{\partial u}{\partial x} = 0, \quad x = \{0, l\},$$

then

$$u(x,t) = \frac{\partial}{\partial t}\int_0^l IC(y)G(x,y,t)dy + \int_0^l IT(y)G(x,y,t)dy + \int_0^t \int_0^l F(y,\tau)G(x,y,t-\tau)dyd\tau,$$

where the Green's function is:

$$G(x,y,t) = \frac{t}{l} + \frac{2}{c\pi}\sum_{n=1}^{\infty}\frac{1}{n}\cos\frac{n\pi x}{l}\cos\frac{n\pi y}{l}\sin -\frac{cn\pi t}{l}$$

7. Eigenfunction expansions for a separable problem

Let $u(\mathbf{x}, t)$ satisfy the following diffusion equation with a separable forcing term and homogeneous Neumann boundary condition and initial condition:

$$\frac{\partial}{\partial t}u(\mathbf{x},t) - \nabla\left(\mathcal{D}_0\nabla u(\mathbf{x},t)\right) = k(\mathbf{x})f(t), \qquad \mathbf{x}\in\Omega, \tag{39}$$

$$\mathcal{D}_0\nabla u(\mathbf{x},t)\cdot\nu(\mathbf{x}) = 0, \qquad \mathbf{x}\in\Gamma, \tag{40}$$

$$u(\mathbf{x},t) = 0, \qquad \mathbf{x}\in\Omega, \tag{41}$$

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:

$$-\nabla\mathcal{D}_0\left(\nabla\phi_n(\mathbf{x})\right) = \lambda_n\phi_n(\mathbf{x}), \quad \mathbf{x}\in\Omega, \tag{42}$$

$$\mathcal{D}_0\nabla\phi_n(\mathbf{x})\cdot\nu(\mathbf{x}) = 0, \qquad \mathbf{x}\in\Gamma \tag{43}$$

such that

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

We claim that the solution $u(\mathbf{x}, t)$ is

$$u(\mathbf{x},t) = \sum_{n=1}^{\infty}(a_n)\phi_n(\mathbf{x})\int_0^t e^{-\lambda_n(t-s)}f(s)ds, \tag{44}$$

where the coefficients are

$$a_n = \int_\Omega k(\mathbf{x})\phi_n(\mathbf{x})d\mathbf{x}. \tag{45}$$

To prove the above claim, we need to show

1. $u(\mathbf{x}, 0) = 0$;
2. $\mathcal{D}_0 \nabla u(\mathbf{x}, t) \cdot \nu(\mathbf{x}) = 0$;
3. $\frac{\partial}{\partial t} u(\mathbf{x}, t) - \nabla \left(\mathcal{D}_0 \nabla u(\mathbf{x}, t) \right) = k(\mathbf{x}) f(t)$;

Remark 4. *We just show below that $u(\mathbf{x}, t)$ satisfies the third item.*

We use the properties of an ortho-normal basis to write $k(\mathbf{x})$ in the eigenfunction basis:

$$k(\mathbf{x}) = \sum_{n=1}^{\infty} (a_n)\phi_n(\mathbf{x}),$$

where a_n is the projection of $k(\mathbf{x})$ on the elements of the basis:

$$a_n = \int_\Omega k(\mathbf{x})\phi_n(\mathbf{x})d\mathbf{x}.$$

Then we show

$$\frac{\partial}{\partial t}\left(\phi_n(\mathbf{x}) \int_0^t e^{-\lambda_n(t-s)} f(s)ds\right) - \nabla\left(\mathcal{D}_0 \nabla\left(\phi_n(\mathbf{x}) \int_0^t e^{-\lambda_n(t-s)} f(s)ds\right)\right) = \phi_n(\mathbf{x}) f(t),$$

by computing

$$\frac{\partial}{\partial t}\left(\phi_n(\mathbf{x}) \int_0^t e^{-\lambda_n(t-s)} f(s)ds\right) = \phi_n(\mathbf{x})\left(\int_0^t (-\lambda_n) e^{-\lambda_n(t-s)} f(s)ds + f(t)\right)$$

and

$$-\nabla\left(\mathcal{D}_0 \nabla\left(\phi_n(\mathbf{x}) \int_0^t e^{-\lambda_n(t-s)} f(s)ds\right)\right) = \lambda_n \phi_n(\mathbf{x}) \int_0^t e^{-\lambda_n(t-s)} f(s)ds.$$

7.1. Eigenfunctions and eigenvalues for rectangle and disk

For Neumann boundary condition the eigenvalues of the Laplacian operator

$$-\nabla \cdot \nabla \phi = \lambda \phi \tag{46}$$

for a rectangle $[0, l_x] \times [0, l_y]$ are

$$\phi_{mn}(x, y) = cos\frac{\pi m x}{l_x} cos\frac{\pi n y}{l_y}, \quad \lambda_{mn} = \frac{\pi^2 m^2}{l_x^2} + \frac{\pi^2 n^2}{l_y^2}, \quad n, m = 0, 1, 2, \cdots \tag{47}$$

For a disk of radius R, the eigenfunctions are:

$$\phi_{nk}(r, \theta) = J_n\left(\frac{\alpha_{nk} r}{R}\right)(A_n cos n\theta + B_n \sin n\theta), \quad \lambda_{nk} = \frac{\alpha_{nk}^2}{R^2}. \tag{48}$$

The function $J_n(z)$ is the nth Bessel function of the first kind, $n = 0, 1, 2, \cdots$. The number α_{nk} is the k-th root, $k = 1, 2, \cdots$, of $J_n'(z)$, the derivative of $J_n(z)$. The coefficients A_n and B_n are arbitary constants, meaning that for each nk combination there are two eigenfunctions, except when $n = 0$, where $\sin n\theta \equiv 0$, so there is only one eigenfunction. In summary, when $n = 0$, λ_{nk} is a simple root (counted only once), when $n > 0$, λ_{nk} is a double root (counted twice). Zero is also an eigenvalue.

Roots of Derivatives of Bessel functions

The n-th roots of Jm'(x)=0.

m\n	n=1	n=2	n=3	n=4	n=5
m=0	3.83170597020751	7.01558666981561	10.1734681350627	13.3236919363142	16.4706300508776
m=1	1.84118378134065	5.33144277352503	8.53631636634628	11.7060049025920	14.8635886339090
m=2	3.05423692822714	6.70613319415845	9.96946782308759	13.1703708560161	16.3475223183217
m=3	4.20118894121052	8.01523659837595	11.3459243107430	14.5858482861670	17.7887478660664
m=4	5.31755312608399	9.28239628524161	12.6819084426388	15.9641070377315	19.1960288000489
m=5	6.41561637570024	10.5198608737723	13.9871886301403	17.3128424878846	20.5755145213868
m=6	7.50126614468414	11.7349359530427	15.2681814610978	18.6374430096662	21.9317150178022
m=7	8.57783648971407	12.9323862370895	16.5293658843669	19.9418533665273	23.2680529264575
m=8	9.64742165199721	14.1155189078946	17.7740123669152	21.2290626228531	24.5871974863176
m=9	10.7114339706999	15.2867376673329	19.0045935379460	22.5013987267772	25.8912772768391
m=10	11.7708766749555	16.4478527484865	20.2230314126817	23.7607158603274	27.1820215271905

Figure 1: The zeros of the derivatives of Bessel functions of the first kind.

8. Matlab programming

8.1. Visualizing basis functions on finite element mesh

```
% This clears workspace variables
clear;
% This closes all figure windows
close all;

width = 5;
height = 5;

% gdm is a matrix to describe the geometry.
%       For a polygon solid, row one contains 2, and the second row
%       contains the number, N, of line segments in the boundary.
%       The following N rows contain the x-coordinates of the starting points
%       of the edges, and the following N rows contain the y-coordinates of the
%       starting points of the edges.

%       For an ellipse solid, row one contains 4, the second and
%       third row contain the center x- and y-coordinates respectively. Row
%       four and five contain the major and minor axes of the ellipse.
%      The rotational angle of the ellipse is stored in row six.

% This is for an rectangle
gdm = [2 4 -width/2,width/2,width/2,-width/2,...
-height/2,-height/2,height/2,height/2]';

% This is for an ellipse
%gdm = [4,0.0,0.0,width/2,width/2,0]';

% Creates geometry
g = decsg(gdm, 'S1', ('S1')');
```

```
30
31 % hmax is related to the size of the finite elements
      requested by the user.
32 % hmax is the maximum edge length requested.
33 hmax = 1;
34
35 % Creates the FE elements mesh for geometry in g that you
      made above.
36 [P,E,T]=initmesh(g,'hmax',hmax);
37
38 % P: nodes.
39 % E: the edges.
40 % T: triangles (elements in 2D).
41 % T is size 4 x N_elem
42 % T(1:3,ii) = the node indices of the 3 vertices of the ii
       element
43 % T(4,ii) = 1 for all ii if we are in dim 2.
44 % Otherwise, T(1:4,ii) are the 4 vertices of the
45 % 3D tetrahedral element.
46 % P = ndim x numofnodes.
47 % P(1,ii) = x coord of node ii
48 % P(2,ii) = y coord of node ii
49 % P(3,ii) = z coord of node ii
50
51 N_edge = size(E,2);
52 N_elem = size(T,2);
53 N_node = size(P,2);
54
55 %  The following code plots out the basis function phi_i(x
      ,y) for each node.
56 %  The node is indicated by the red star.
57 %  The basis function phi_i(x,y) is plotted in blue as a "
      tent".
58 %  This function is non-zeros at SEVERAL triangles
      touching the red star.
59 %  phi_i(x,y) is 0 on the triangles which do not touch the
       red star.
60
```

```
figure;
% this loop goes through all N_nodes nodes
for ii = 1:N_node
        clf; pdeplot(P,E,T); hold on;
    % this finds the index of the elements that contain
       the node ii
        [index_on_element] = find(T(1,:)==ii | T(2,:) ==
           ii | T(3,:) == ii);
        N_on_element = length(index_on_element);
    % this loop goes through all elements that contain the
        node ii
        for jj = 1:N_on_element
        % this is the node index of vertex1 of the element
            jj
                vertex1_index = T(1,index_on_element(jj));
        % this is the node index of vertex2 of the element
            jj
                vertex2_index = T(2,index_on_element(jj));
                vertex3_index = T(3,index_on_element(jj));
        % this is the (x,y) coordinates of vertex 1
                P_vertex1 = P(:,vertex1_index);
        % this is the (x,y) coordinates of vertex 2
                P_vertex2 = P(:,vertex2_index);
                P_vertex3 = P(:,vertex3_index);
        % this is the x-coordinates of (vertex1, vertex2,
           vertex3)
                Xcoords = [P_vertex1(1),P_vertex2(1),
                   P_vertex3(1)]';
        % this is the y-coordinates of (vertex1,vertex2,
           vertex3)
                Ycoords = [P_vertex1(2),P_vertex2(2),
                   P_vertex3(2)]';
        % this is saying if the vertex is the node ii
        %
                if (vertex1_index == ii)
                        Zcoords = [1,0,0]';
                elseif (vertex2_index == ii)
                        Zcoords = [0,1,0]';
```

```
                elseif (vertex3_index == ii)
                        Zcoords = [0,0,1]';
        end

                C = linspace(0,1,length(Zcoords));
        % this plots the basis shape on the triangle jj.
                h=patch(Xcoords,Ycoords,Zcoords,C);
                set(h,'FaceAlpha',0.5);
                plot3(P(1,ii),P(2,ii),0,'r*','markersize'
                   ,10,'linewidth',10);
        end
        view(3);
        axis equal;
        title(['Basis function \phi_{',num2str(ii),'}(x,y)
            for node ',...
        num2str(ii), ' (red star) defined on ',num2str(
           N_on_element),' triangles']);
        pause(1);
end
```

8.2. Assembling finite element matrices

```
clear; close all;

hmax = 1; % requested finite elements size
width = 5; height = 5;

% gdm is a matrix to describe the geometry.
gdm = [3 4 -width/2,width/2,width/2,-width/2,...
-height/2,-height/2,height/2,height/2]';

g = decsg(gdm, 'S1', ('S1')');

% Number PDE system, just 1 PDE.
numberOfPDE = 1;

% 2nd derivative in time
M_COEFF = 0;
```

```
% 1st derivative in time
D_COEFF = 1;
% diffusion tensor/coefficient
C_COEFF = 1;
% coefficient in front of u
A_COEFF = 1;
% source term
F_COEFF = 0;

% Creates PDE model object
heatmodel = createpde(numberOfPDE);

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

% Plots the geometry
figure;
pdegplot(heatmodel_geom,'EdgeLabels','on');
title('Geometry With Edge Labels Displayed');

% Generates a finite elements mesh using P1 elements.
msh = generateMesh(heatmodel,'GeometricOrder','linear','
   hmax',hmax);

% Outputs the mesh into the Nodes, Edges, and Elements.
[P,E,T] = meshToPet(msh);

% Plots the FE mesh.
figure;
pdeplot(P,E,T);
title('Finite element mesh');

% Set PDE coefficients to be the heat equation
specifyCoefficients(heatmodel,'m',M_COEFF,'d',D_COEFF,...
'c',C_COEFF,'a',A_COEFF,'f',F_COEFF);

% Find the number of pieces of the boundary of the
   geometry.
```

```
NumEdges = heatmodel_geom.NumEdges;

% Set zero Neumann boundary conditions on all the pieces
   of the boundary.
for ie = 1:NumEdges
        applyBoundaryCondition(heatmodel,'edge',ie,'g',0,'
          q',0);
end
% Call the assembly routines in Matlab PDE Toolbox to get
   6 FE matrices.
model_FEM_matrices = assembleFEMatrices(heatmodel);

% The 6 FE matrices can be obtained in the following way.
FEM_M = model_FEM_matrices.M;

% matrix, integral in domain
FEM_K = model_FEM_matrices.K;


% matrix, integral in domain
FEM_A = model_FEM_matrices.A;
% matrix, integral in boundary
FEM_Q = model_FEM_matrices.Q;

% vector, integral in domain
FEM_F = model_FEM_matrices.F;
% vector, integral in boundary
FEM_G = model_FEM_matrices.G;
```

8.3. Solving the heat equation using the Matlab PDE Toolbox

```
clear; close all;

hmax = 1; % requested finite elements size

width = 5; height = 5;
% gdm is a matrix to describe the geometry.
```

```
gdm = [3 4 -width/2,width/2,width/2,-width/2,-height/2,-height/2,height/2,height/2]';
g = decsg(gdm, 'S1', ('S1')');

% No PDE system, just 1 PDE.
numberOfPDE = 1;
% 2nd derivative in time
M_COEFF = 0;
% 1st derivative in time
D_COEFF = 1;
% diffusion tensor/coefficient
C_COEFF = 1;
% coefficient in front of u
A_COEFF = 0;
% source term
F_COEFF = 0;

% Creates PDE model object
heatmodel = createpde(numberOfPDE);

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

% generates finite elements mesh
msh = generateMesh(heatmodel,'GeometricOrder','linear');
[P,E,T] = meshToPet(msh);

% set PDE coefficients
specifyCoefficients(heatmodel,'m',M_COEFF,'d',D_COEFF,'c',C_COEFF,'a',A_COEFF,'f',F_COEFF);

NumEdges = heatmodel_geom.NumEdges;

% set zero Neumann boundary conditions
for ie = 1:NumEdges
        applyBoundaryCondition(heatmodel,'edge',ie,'g',0,'q',0);
end
```

```
42
43
44 % set initial conditions.
45 % this is for the PDE toolbox.
46 setInitialConditions(heatmodel,@IC_pdetoolbox);
47
48 % set time of simulation
49 startTime = 0;
50 endTime = 0.05;
51 ntime = 101;
52 tlist = linspace(startTime,endTime,ntime);
53
54 % solves PDE using the Matlab PDE toolbox
55 R = solvepde(heatmodel,tlist);
56 u = R.NodalSolution;
57
58 figure;
59 subplot(2,2,1);
60 pdegplot(heatmodel_geom,'EdgeLabels','on');
61 title('Geometry With Edge Labels Displayed');
62 subplot(2,2,2);
63 pdeplot(P,E,T);
64 title('Finite element mesh');
65 subplot(2,2,3);
66 pdeplot(heatmodel,'XYData',u(:,1),'Contour','on','ColorMap
      ','jet');
67 title(sprintf('solution u at t = %d \n',tlist(1,1)));
68 xlabel('X-coordinate');
69 ylabel('Y-coordinate');
70 axis equal;
71 subplot(2,2,4);
72 pdeplot(heatmodel,'XYData',u(:,end),'Contour','on','
      ColorMap','jet');
73 title(sprintf('solution u at t = %d \n',tlist(1,end)));
74 xlabel('X-coordinate');
75 ylabel('Y-coordinate');
76 axis equal;
77
```

```
function f = IC_pdetoolbox(region,state)
  nr = length(region.x);
  f = zeros(1,nr);
  f(1,:) = IC_general(region.x,region.y);
end

function f = IC_general(x,y)
        aa = 0.01;
        nr = length(x);
        f = zeros(1,nr);
        f(1,:) = exp(-(((x+0.25)).^2+((y-0.25).^2))/aa);

end
```

8.4. Solving the heat equation using the FE matrices and ODE routines

```
clear; close all;

global FEM_M FEM_K FEM_A FEM_Q FEM_G FEM_F
global t0;
global sigma0;
global x0;
global y0;

hmax = 0.3; % requested finite elements size

x0 = 0;
y0 = 0;
t0 = 0.5;

width = 10; height = 8;
% gdm is a matrix to describe the geometry.
gdm = [3 4 -width/2,width/2,width/2,-width/2,-height/2,-
   height/2,height/2,height/2]';
g = decsg(gdm, 'S1', ('S1')');

% No PDE system, just 1 PDE.
numberOfPDE = 1;
```

```
% 2nd derivative in time
M_COEFF = 0;
% 1st derivative in time
D_COEFF = 1;
% diffusion tensor/coefficient
C_COEFF = 1;
% coefficient in front of u
A_COEFF = 0;
% source term
F_COEFF = 0;

sigma0 = C_COEFF;

% Creates PDE model object
heatmodel = createpde(numberOfPDE);

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

% generates finite elements mesh
msh = generateMesh(heatmodel,'GeometricOrder','linear','
   hmax',hmax);
[P,E,T] = meshToPet(msh);
pdeplot(P,E,T);

% set PDE coefficients
specifyCoefficients(heatmodel,'m',M_COEFF,'d',D_COEFF,'c',
   C_COEFF,'a',A_COEFF,'f',F_COEFF);

NumEdges = heatmodel_geom.NumEdges;
% set zero Neumann boundary conditions
for ie = 1:NumEdges
        applyBoundaryCondition(heatmodel,'edge',ie,'g',0,'
           q',0);
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 = model_FEM_matrices.G;
FEM_F = model_FEM_matrices.F;


% set time of simulation
startTime = 0;
endTime = 0.4;
ntime = 101;
tlist = linspace(startTime,endTime,ntime);

odesolve_tol = 1e-6;

% This evaluates Initial Condition
u0 = IC_general(P(1,:),P(2,:));

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

disp('ode23t');

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


figure;
subplot(2,2,1);
pdegplot(heatmodel_geom,'EdgeLabels','on');
title('Geometry With Edge Labels Displayed');
subplot(2,2,2);
pdeplot(P,E,T);
```

```
title('Finite element mesh');
subplot(2,2,3);
pdeplot(heatmodel,'XYData',YOUT(1,:)','Contour','on','
   ColorMap','jet');
title(sprintf('solution u at t = %d \n',tlist(1,1)));
xlabel('X-coordinate');
ylabel('Y-coordinate');
caxis([0,0.5]);
axis equal;
subplot(2,2,4);
pdeplot(heatmodel,'XYData',YOUT(end,:)','Contour','on','
   ColorMap','jet');
title(sprintf('solution u at t = %d \n',tlist(1,end)));
xlabel('X-coordinate');
ylabel('Y-coordinate');
caxis([0,0.5]);
axis equal;

function f = IC_general(x,y)

global t0;
global sigma0;
global x0;
global y0;

        nr = length(x);
        f = zeros(1,nr);
        f(1,:) = exp(-(((x-x0)).^2)/(4*sigma0*t0))/sqrt(4*
           pi*sigma0*t0);

end

function Yout= odefun_semidiscretize_pde(t,Y)

        global FEM_M FEM_K FEM_A FEM_Q FEM_G FEM_F
        Yout = -(FEM_K*Y+FEM_A*Y+FEM_Q*Y)+FEM_G+FEM_F;

end
```

9. Homework problems

1. Compute the integral of the function $g(x,y) = (ax + by + c)(dx + ey + f)$, where a, b, c, d, e, f are constants, over the segment with endpoints $\{P_1 = (x_1, y_1), P_2 = (x_2, y_2)\}$.

2. Given the basis functions $\phi_i(x,y)$ and $\phi_j(x,y)$, associated with nodes P_i and P_j, for which edges $\{E_k\}$ is $\int_{E_k} \phi_i(x,y)\phi_j(x,y)ds = 0$? For which edges $\{E_k\}$ is $\int_{E_k} \phi_i(x,y)\phi_j(x,y)ds \neq 0$?

3. Compute the finite element matrix Q_{ij} for $q \equiv 1$ in Eq. 19.

4. Use the following change of variables $(x,y) \to (\xi,\eta)$:

$$\xi = \frac{\det \begin{vmatrix} 1 & x & y \\ 1 & x_1 & y_1 \\ 1 & x_3 & y_3 \end{vmatrix}}{\det \begin{vmatrix} 1 & x_2 & y_2 \\ 1 & x_1 & y_1 \\ 1 & x_3 & y_3 \end{vmatrix}}, \quad \eta = \frac{\det \begin{vmatrix} 1 & x & y \\ 1 & x_1 & y_1 \\ 1 & x_2 & y_2 \end{vmatrix}}{\det \begin{vmatrix} 1 & x_3 & y_3 \\ 1 & x_1 & y_1 \\ 1 & x_2 & y_2 \end{vmatrix}}$$

   to map a triangle with vertices $\{P_1 = (x_1, y_1), P_2 = (x_2, y_2), P_3 = (x_3, y_3)\}$ into a canonical element with vertices $\{(\xi_1, \eta_1) = (0,0), (\xi_2, \eta_2) = (1,0), (\xi_3, \eta_3) = (0,1)\}$. Compute the integral of the function $g(x,y) = (ax+by+c)(dx+ey+f)$, where a, b, c, d, e, f are constants, over the triangle with vertices $\{P_1 = (x_1, y_1), P_2 = (x_2, y_2), P_3 = (x_3, y_3)\}$.

5. Given the basis functions $\phi_i(x,y)$ and $\phi_j(x,y)$, associated with nodes P_i and P_j, for which triangles $\{T_k\}$ is $\int_{T_k} \phi_i(x,y)\phi_j(x,y)dxdy = 0$? For which triangles $\{T_k\}$ is $\int_{T_k} \phi_i(x,y)\phi_j(x,y)dxdy \neq 0$?

6. Compute the finite element matrix A_{ij} for $a \equiv 1$ in Eq. 20.

7. Explain why the basis functions $\mathbb{P}_1$ are continuous.