Partial Differential Equations in MATLAB 7.0

P. Howard

Spring 2005

Contents

1 PDE in One Space Dimension

For initial–boundary value partial differential equations with time t and a single spatial variable x, MATLAB has a built-in solver *pdepe*.

1.1 Single equations

Example 1.1. Suppose, for example, that we would like to solve the heat equation

$$\begin{aligned} u_t &= u_{xx} \\ u(t,0) = 0&, \quad u(t,1) = 1 \\ u(0,x) &= \frac{2x}{1+x^2}. \end{aligned} \tag{1.1}$$

MATLAB specifies such parabolic PDE in the form

$$c(x,t,u,u_x)u_t = x^{-m}\frac{\partial}{\partial x}\Big(x^m b(x,t,u,u_x)\Big) + s(x,t,u,u_x),$$

with boundary conditions

$$\begin{aligned} p(x_l,t,u) + q(x_l,t)\cdot b(x_l,t,u,u_x) &= 0 \\ p(x_r,t,u) + q(x_r,t)\cdot b(x_r,t,u,u_x) &= 0, \end{aligned}$$

where x_l represents the left endpoint of the boundary and x_r represents the right endpoint of the boundary, and initial condition

$$u(0,x) = f(x).$$

(Observe that the same function b appears in both the equation and the boundary conditions.) Typically, for clarity, each set of functions will be specified in a separate M-file. That is, the functions c, b, and s associated with the equation should be specified in one M-file, the functions p and q associated with the boundary conditions in a second M-file (again, keep in mind that b is the same and only needs to be specified once), and finally the initial function $f(x)$ in a third. The command *pdepe* will combine these M-files and return a solution to the problem. In our example, we have

$$\begin{aligned} c(x,t,u,u_x) =& 1 \\ b(x,t,u,u_x) =& u_x \\ s(x,t,u,u_x) =& 0, \end{aligned}$$

which we specify in the function M-file *eqn1.m*. (The specification $m = 0$ will be made later.)

```
function [c,b,s] = eqn1(x,t,u,DuDx)
%EQN1: MATLAB function M-file that specifies
%a PDE in time and one space dimension.
c = 1;
b = DuDx;
s = 0;
```

For our boundary conditions, we have

$$\begin{aligned} p(0,t,u) = u; &\quad q(0,t) = 0 \\ p(1,t,u) = u-1; &\quad q(1,t) = 0, \end{aligned}$$

which we specify in the function M-file *bc1.m*.

```
function [pl,ql,pr,qr] = bc1(xl,ul,xr,ur,t)
%BC1: MATLAB function M-file that specifies boundary conditions
%for a PDE in time and one space dimension.
pl = ul;
ql = 0;
pr = ur-1;
qr = 0;
```

For our initial condition, we have

$$f(x) = \frac{2x}{1+x^2},$$

which we specify in the function M-file *initial1.m*.

```
function value = initial1(x)
%INITIAL1: MATLAB function M-file that specifies the initial condition
%for a PDE in time and one space dimension.
value = 2*x/(1+x^2);
```

We are finally ready to solve the PDE with *pdepe*. In the following script M-file, we choose a grid of x and t values, solve the PDE and create a surface plot of its solution (given in Figure 1.1).

```
%PDE1: MATLAB script M-file that solves and plots
%solutions to the PDE stored in eqn1.m
m = 0;
%NOTE: m=0 specifies no symmetry in the problem. Taking
%m=1 specifies cylindrical symmetry, while m=2 specifies
%spherical symmetry.
%
%Define the solution mesh
x = linspace(0,1,20);
t = linspace(0,2,10);
%Solve the PDE
u = pdepe(m,@eqn1,@initial1,@bc1,x,t);
%Plot solution
surf(x,t,u);
title('Surface plot of solution.');
xlabel('Distance x');
ylabel('Time t');
```

Often, we find it useful to plot solution *profiles*, for which t is fixed, and u is plotted against x. The solution $u(t, x)$ is stored as a matrix indexed by the vector indices of t and x. For example, $u(1, 5)$ returns the value of u at the point $(t(1), x(5))$. We can plot u initially (at $t = 0$) with the command *plot(x,u(1,:))* (see Figure 1.2).

Finally, a quick way to create a movie of the profile's evolution in time is with the following MATLAB sequence.

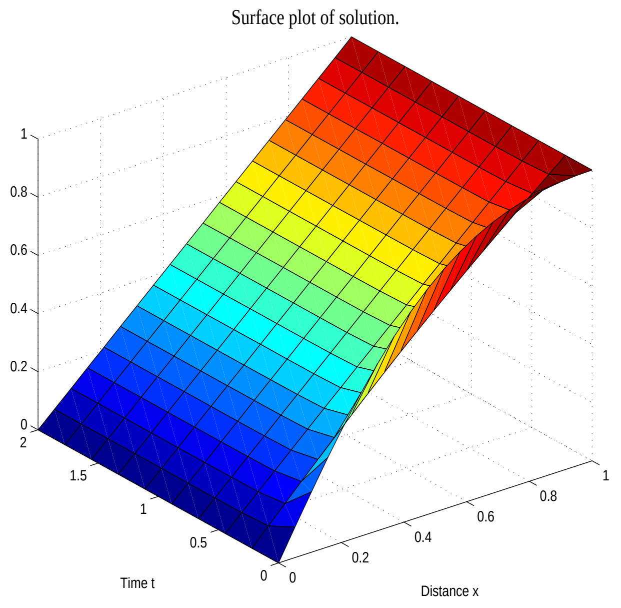

Figure 1.1: Mesh plot for solution to Equation (1.1)

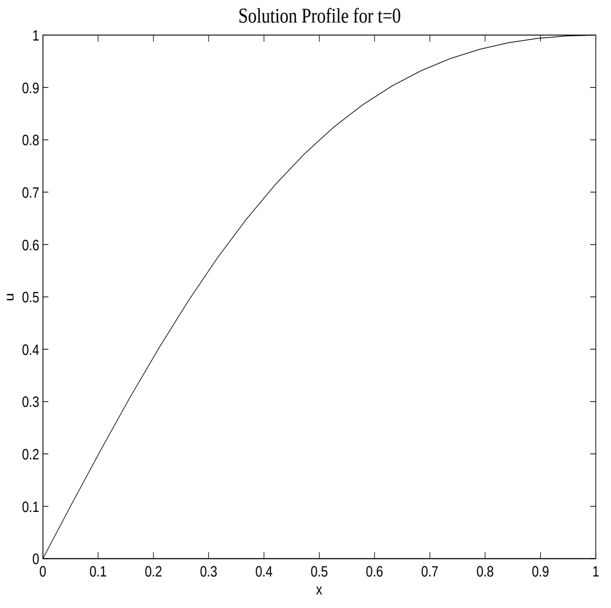

Figure 1.2: Solution Profile at $t = 0$.

```
fig = plot(x,u(1,:),'erase','xor')
for k=2:length(t)
set(fig,'xdata',x,'ydata',u(k,:))
pause(.5)
end
```

If you try this out, observe how quickly solutions to the heat equation approach their equilibrium configuration. (The equilibrium configuration is the one that ceases to change in time.) △

1.2 Single Equations with Variable Coefficients

The following example arises in a roundabout way from the theory of detonation waves.

Example 1.2. Consider the linear *convection–diffusion* equation

$$\begin{aligned} u_t + (a(x)u)_x &= u_{xx} \\ u(t,-\infty) &= u(t,+\infty) = 0 \\ u(0,x) &= \frac{1}{1+(x-5)^2}, \end{aligned}$$

where $a(x)$ is defined by

$$a(x) = 3\bar{u}(x)^2 - 2\bar{u}(x),$$

with $\bar{u}(x)$ defined implicitly through the relation

$$\frac{1}{\bar{u}} + \log|\frac{1-\bar{u}}{\bar{u}}| = x.$$

(The function $\bar{u}(x)$ is an equilibrium solution to the conservation law

$$u_t + (u^3 - u^2)_x = u_{xx},$$

with $\bar{u}(-\infty) = 1$ and $\bar{u}(+\infty) = 0$. In particular, $\bar{u}(x)$ is a solution typically referred to as a *degenerate viscous shock wave.*)

Since the equilibrium solution $\bar{u}(x)$ is defined implicitly in this case, we first write a MATLAB M-file that takes values of x and returns values $\bar{u}(x)$. Observe in this M-file that the guess for *fzero()* depends on the value of x.

```
function value = degwave(x)
%DEGWAVE: MATLAB function M-file that takes a value x
%and returns values for a standing wave solution to
%u_t + (u^3 - u^2)_x = u_xx
guess = .5;
if x < -35
value = 1;
else
```

```
if x > 2
guess = 1/x;
elseif x>-2.5
guess = .6;
else
guess = 1-exp(-2)*exp(x);
end
value = fzero(@f,guess,[],x);
end
function value1 = f(u,x)
value1 = (1/u)+log((1-u)/u)-x;
```

The equation is now stored in *deglin.m*.

```
function [c,b,s] = deglin(x,t,u,DuDx)
%EQN1: MATLAB function M-file that specifies
%a PDE in time and one space dimension.
c = 1;
b = DuDx - (3*degwave(x)^2 - 2*degwave(x))*u;
s = 0;
```

In this case, the boundary conditions are at $\pm\infty$. Since MATLAB only understands finite domains, we will approximate these conditions by setting $u(t,-50) = u(t,50) = 0$. Observe that at least initially this is a good approximation since $u_0(-50) = 3.2e-4$ and $u_0(+50) = 4.7e-4$. The boundary conditions are stored in the MATLAB M-file *degbc.m*.

```
function [pl,ql,pr,qr] = degbc(xl,ul,xr,ur,t)
%BC1: MATLAB function M-file that specifies boundary conditions
%for a PDE in time and one space dimension.
pl = ul;
ql = 0;
pr = ur;
qr = 0;
```

The initial condition is specified in *deginit.m*.

```
function value = deginit(x)
%DEGINIT: MATLAB function M-file that specifies the initial condition
%for a PDE in time and one space dimension.
value = 1/(1+(x-5)^2);
```

Finally, we solve and plot this equation with *degsolve.m*.

```
%DEGSOLVE: MATLAB script M-file that solves and plots
%solutions to the PDE stored in deglin.m
%Suppress a superfluous warning:
clear h;
warning off MATLAB:fzero:UndeterminedSyntax
m = 0;
%
%Define the solution mesh
x = linspace(-50,50,200);
t = linspace(0,10,100);
%
u = pdepe(m,@deglin,@deginit,@degbc,x,t);
%Create profile movie
flag = 1;
while flag==1
answer = input('Finished iteration. View plot (y/n)','s')
if isequal(answer,'y')
figure(2)
fig = plot(x,u(1,:),'erase','xor')
for k=2:length(t)
set(fig,'xdata',x,'ydata',u(k,:))
pause(.4)
end
else
flag = 0;
end
end
```

The line *warning off MATLAB:fzero:UndeterminedSyntax* simply turns off an error message MATLAB issued every time it called *fzero()*. Observe that the option to view a movie of the solution's time evolution is given inside a for-loop so that it can be watched repeatedly without re-running the file. The initial and final configurations of the solution to this example are given in Figures 1.3 and 1.4.

1.3 Systems

We next consider a system of two partial differential equations, though still in time and one space dimension.

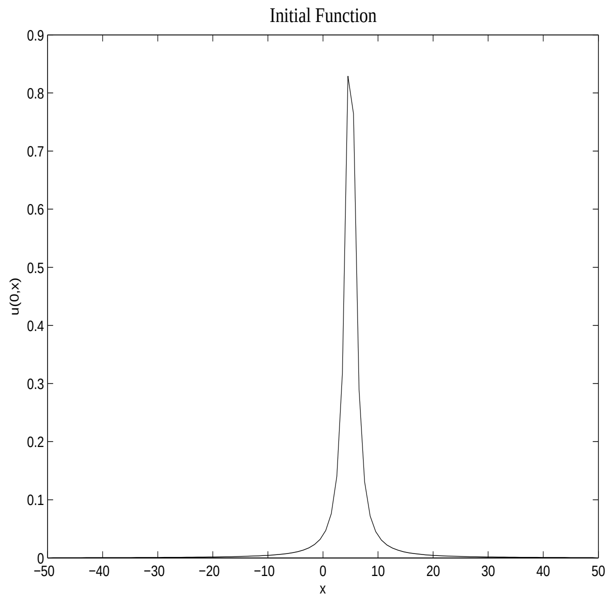

Figure 1.3: Initial Condition for Example 1.2.

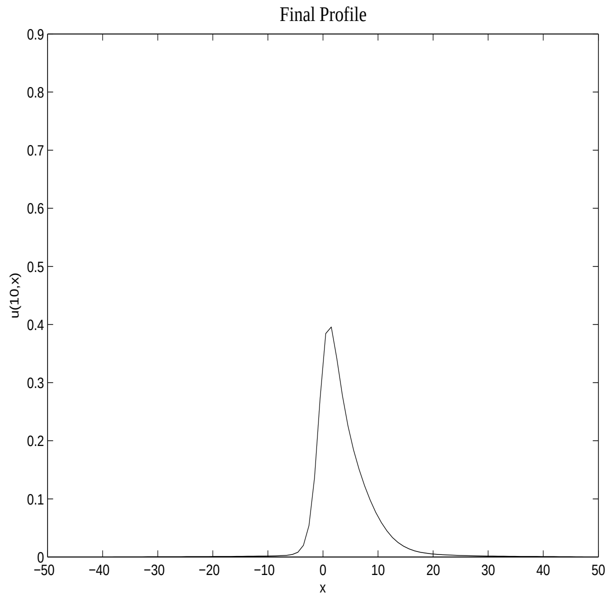

Figure 1.4: Final profile for Example 1.2 solution.

Example 1.3. Consider the nonlinear system of partial differential equations

$$\begin{aligned}
u_{1_t} =& u_{1_{xx}} + u_1(1 - u_1 - u_2)\\
u_{2_t} =& u_{2_{xx}} + u_2(1 - u_1 - u_2),\\
u_{1_x}(t,0) =& 0; \quad u_1(t,1) = 1\\
u_2(t,0) =& 0; \quad u_{2_x}(t,1) = 0,\\
u_1(0,x) =& x^2\\
u_2(0,x) =& x(x-2). \qquad (1.2)
\end{aligned}$$

(This is a non-dimensionalized form of a PDE model for two competing populations.) As with solving ODE in MATLAB, the basic syntax for solving systems is the same as for solving single equations, where each scalar is simply replaced by an analogous vector. In particular, MATLAB specifies a system of n PDE as

$$\begin{aligned}
c_1(x,t,u,u_x)u_{1_t} =& x^{-m}\frac{\partial}{\partial x}\Big(x^m b_1(x,t,u,u_x)\Big) + s_1(x,t,u,u_x)\\
c_2(x,t,u,u_x)u_{2_t} =& x^{-m}\frac{\partial}{\partial x}\Big(x^m b_2(x,t,u,u_x)\Big) + s_2(x,t,u,u_x)\\
&\vdots\\
c_n(x,t,u,u_x)u_{n_t} =& x^{-m}\frac{\partial}{\partial x}\Big(x^m b_n(x,t,u,u_x)\Big) + s_n(x,t,u,u_x),
\end{aligned}$$

(observe that the functions c_k, b_k, and s_k can depend on all components of u and u_x) with boundary conditions

$$\begin{aligned}
p_1(x_l,t,u) + q_1(x_l,t)\cdot b_1(x_l,t,u,u_x) =& 0\\
p_1(x_r,t,u) + q_1(x_r,t)\cdot b_1(x_r,t,u,u_x) =& 0\\
p_2(x_l,t,u) + q_2(x_l,t)\cdot b_2(x_l,t,u,u_x) =& 0\\
p_2(x_r,t,u) + q_2(x_r,t)\cdot b_2(x_r,t,u,u_x) =& 0\\
&\vdots\\
p_n(x_l,t,u) + q_n(x_l,t)\cdot b_n(x_l,t,u,u_x) =& 0\\
p_n(x_r,t,u) + q_n(x_r,t)\cdot b_n(x_r,t,u,u_x) =& 0,
\end{aligned}$$

and initial conditions

$$\begin{aligned}
u_1(0,x) =& f_1(x)\\
u_2(0,x) =& f_2(x)\\
&\vdots\\
u_n(0,x) =& f_n(x).
\end{aligned}$$

In our example equation, we have

$$c = \begin{pmatrix} c_1\\ c_2\end{pmatrix} = \begin{pmatrix} 1\\ 1\end{pmatrix}; \quad b = \begin{pmatrix} b_1\\ b_2\end{pmatrix} = \begin{pmatrix} u_{1_x}\\ u_{2_x}\end{pmatrix}; \quad s = \begin{pmatrix} s_1\\ s_2\end{pmatrix} = \begin{pmatrix} u_1(1-u_1-u_2)\\ u_2(1-u_1-u_2)\end{pmatrix},$$

which we specify with the MATLAB M-file *eqn2.m*.

```
function [c,b,s] = eqn2(x,t,u,DuDx)
%EQN2: MATLAB M-file that contains the coefficents for
%a system of two PDE in time and one space dimension.
c = [1; 1];
b = [1; 1] .* DuDx;
s = [u(1)*(1-u(1)-u(2)); u(2)*(1-u(1)-u(2))];
```

For our boundary conditions, we have

$$p(0,t,u) = \begin{pmatrix} p_1 \\ p_2 \end{pmatrix} = \begin{pmatrix} 0 \\ u_2 \end{pmatrix}; \quad q(0,t) = \begin{pmatrix} q_1 \\ q_2 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$
$$p(1,t,u) = \begin{pmatrix} p_1 \\ p_2 \end{pmatrix} = \begin{pmatrix} u_1 - 1 \\ 0 \end{pmatrix}; \quad q(1,t) = \begin{pmatrix} q_1 \\ q_2 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

which we specify in the function M-file *bc2.m*.

```
function [pl,ql,pr,qr] = bc2(xl,ul,xr,ur,t)
%BC2: MATLAB function M-file that defines boundary conditions
%for a system of two PDE in time and one space dimension.
pl = [0; ul(2)];
ql = [1; 0];
pr = [ur(1)-1; 0];
qr = [0; 1];
```

For our initial conditions, we have

$$\begin{aligned} u_1(0,x) =& x^2 \\ u_2(0,x) =& x(x-2), \end{aligned}$$

which we specify in the function M-file *initial2.m*.

```
function value = initial2(x);
%INITIAL2: MATLAB function M-file that defines initial conditions
%for a system of two PDE in time and one space variable.
value = [x^2; x*(x-2)];
```

We solve equation (1.2) and plot its solutions with *pde2.m* (see Figure 1.5).

```
%PDE2: MATLAB script M-file that solves the PDE
%stored in eqn2.m, bc2.m, and initial2.m
m = 0;
x = linspace(0,1,10);
t = linspace(0,1,10);
sol = pdepe(m,@eqn2,@initial2,@bc2,x,t);
u1 = sol(:,:,1);
u2 = sol(:,:,2);
```

```
subplot(2,1,1)
surf(x,t,u1);
title('u1(x,t)');
xlabel('Distance x');
ylabel('Time t');
subplot(2,1,2)
surf(x,t,u2);
title('u2(x,t)');
xlabel('Distance x');
ylabel('Time t');
```

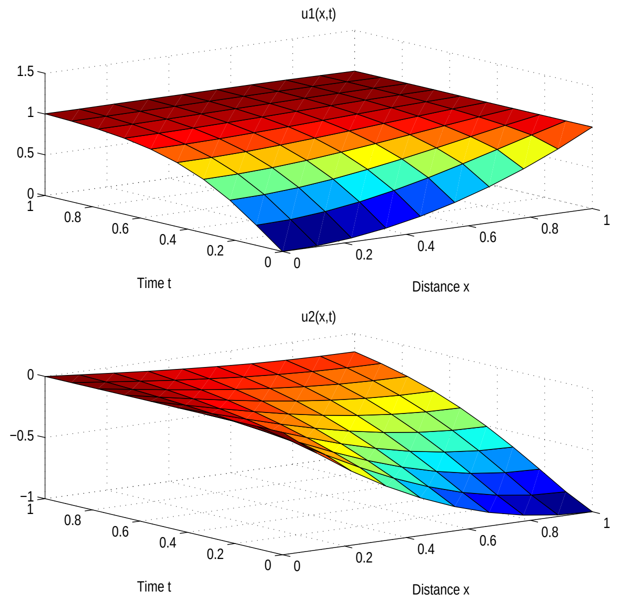

Figure 1.5: Mesh plot of solutions for Example 1.3.

1.4 Systems of Equations with Variable Coefficients

We next consider a system analogue to Example 1.2.

Example 1.4. Consider the system of convection–diffusion equations

$$\begin{aligned} u_{1_t} - 2u_{1_x} - u_{2_x} &= u_{1_{xx}} \\ u_{2_t} - u_{1_x} - 2u_{2_x} - (3\bar{u}_1(x)^2 u_1) &= u_{2_{xx}} \\ u_1(t,-\infty) = u_1(t,+\infty) &= 0 \\ u_2(t,-\infty) = u_2(t,+\infty) &= 0 \\ u_1(0,x) &= e^{-(x-5)^2} \\ u_2(0,x) &= e^{-(x+5)^2}, \end{aligned}$$

where $\bar{u}_1(x)$ is the first component in the solution of the boundary value ODE system

$$\begin{aligned}
\bar{u}_{1_x} &= -\,2(\bar{u}_1+2)-\bar{u}_2\\
\bar{u}_{2_x} &= -\,(\bar{u}_1+2)-2\bar{u}_2-(\bar{u}_1^3+8)\\
\bar{u}_1(-\infty)=-2;&\quad \bar{u}_1(+\infty)=1\\
\bar{u}_2(-\infty)=0;&\quad \bar{u}_2(+\infty)=-6.
\end{aligned}$$

In this case, the vector function $\bar{u}(x)=(\bar{u}_1(x),\bar{u}_2(x))^{\mathrm{tr}}$ is a degenerate viscous shock solution to the conservation law

$$\begin{aligned}
u_{1_t}-2u_{1_x}-u_{2_x} &= u_{1_{xx}}\\
u_{2_t}-u_{1_x}-2u_{2_x}-(u_1^3)_x &= u_{2_{xx}}.
\end{aligned}$$

One of the main obstacles of this example is that it is prohibitively difficult to develop even an implicit representation for $\bar{u}(x)$. We will proceed by solving the ODE for $\bar{u}(x)$ at each step in our PDE solution process. First, the ODE for $\bar{u}(x)$ is stored in *degode.m*.

```
function xprime = degode(t,x);
%DEGODE: Stores an ode for a standing wave
%solution to the p-system.
xprime=[-2*(x(1)+2)-x(2); -(x(1)+2)-2*x(2)-(x(1)^3+8)];
```

We next compute $\bar{u}_1(x)$ in *pdegwave.m* by solving this ODE with appropriate approximate boundary conditions.

```
function u1bar=pdegwave(x)
%PDEGWAVE: Function M-file that takes input x and returns
%the vector value of a degenerate wave.
%in degode.m
small = .000001;
if x <= -20
u1bar = -2;
u2bar = 0;
else
tspan = [-20 x];
%Introduce small perturbation from initial point
x0 = [-2+small,-small];
[t,x]=ode45('degode',tspan,x0);
u1bar = x(end,1);
u2bar = x(end,2);
end
```

In this case, we solve the ODE by giving it boundary conditions extremely close to the asymptotic boundary conditions. The critical issue here is that the asymptotic boundary conditions are equilibrium points for the ODE, so if we started right at the boundary condition we would never leave it. We define our boundary conditions and initial conditions in *psysbc.m* and *degsysinit.m* respectively.

```
function [pl,ql,pr,qr]=psysbc(xl,ul,xr,ur,t)
%PSYSBC: Boundary conditions for the linearized
%p-system.
pl=[ul(1);ul(2)];
ql=[0;0];
pr=[ur(1);ur(2)];
qr=[0;0];
```

and

```
function value = degsysinit(x);
%DEGSYSINIT: Contains initial condition for linearized
%p-system.
value = [exp(-(x-5)^2);exp(-(x+5)^2)];
```

The PDE is stored in *deglinsys.m*.

```
function [c,b,s] = deglinsys(x,t,u,DuDx)
%DEGLINSYS: MATLAB M-file that contains the coefficents for
%a system of two PDE in time and one space dimension.
c = [1; 1];
b = [1; 1] .* DuDx + [2*u(1)+u(2);u(1)+2*u(2)+3*pdegwave(x)^2*u(1)];
s = [0;0];
```

Finally, we solve the PDE and plot its solutions with *degsolve.m*.

```
%DEGSOLVE: MATLAB script M-file that solves the PDE
%stored in deglinsys.m, psysbc.m, and degsysinit.m
clf;
m = 0;
x = linspace(-25,25,100);
t = linspace(0,2,20);
sol = pdepe(m,@deglinsys,@degsysinit,@psysbc,x,t);
u1 = sol(:,:,1);
u2 = sol(:,:,2);
flag = 1;
while flag==1
answer = input('Finished iteration. View plot (y/n)','s')
if isequal(answer,'y')
figure;
hold on;
fig1=plot(x,u1(1,:),'erase','xor')
axis([min(x) max(x) -1 1]);
fig2=plot(x,u2(2,:),'r','erase','xor')
for k=2:length(t)
set(fig1,'ydata',u1(k,:));
set(fig2,'ydata',u2(k,:));
pause(.5)
end
else
flag=0
end
end
```

The initial condition for this problem is given in Figure 1.6, while the final configuration is given in Figure 1.7. Ideally, this would have been run for a longer time period, but since an ODE was solved at each step of the PDE solution process, the compuation was extremely time-consuming. Evolving the system for two seconds took roughly ten minutes.

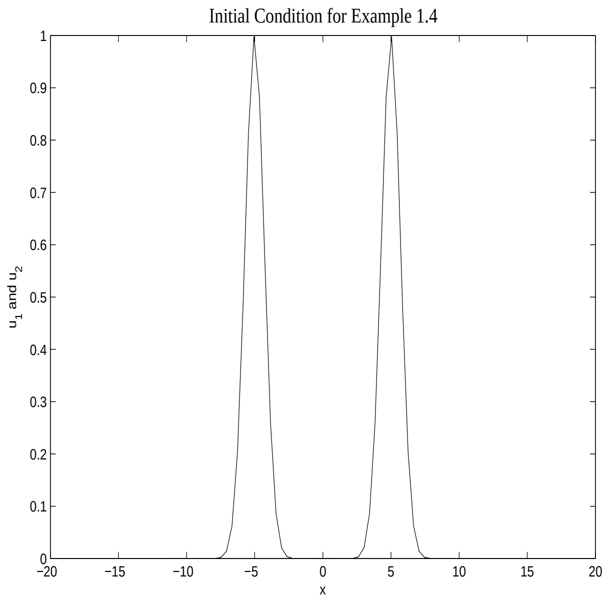

Figure 1.6: Initial configuration for Example 1.4.

2 Single PDE in Two Space Dimensions

For partial differential equations in two space dimensions, MATLAB has a GUI (graphical user interface) called PDE Toolbox, which allows four types of equations (the d in this equations is a parameter, not a differential):

1. Elliptic

$$-\nabla \cdot (c\nabla u) + au = f.$$

2. Parabolic

$$du_t - \nabla \cdot (c\nabla u) + au = f.$$

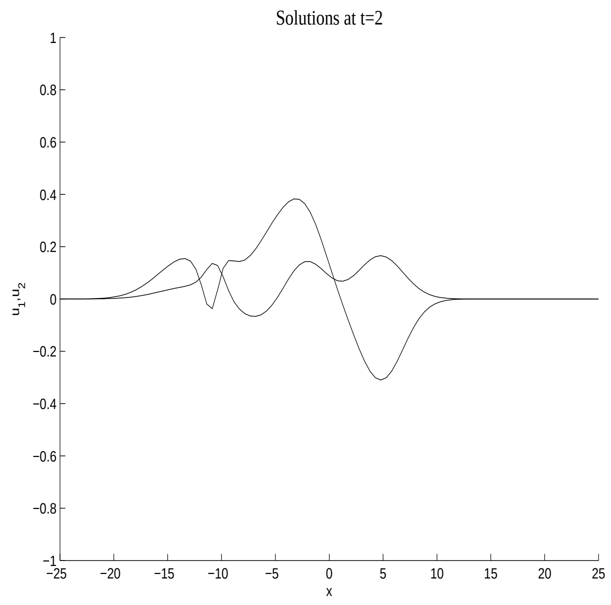

Figure 1.7: Final configuration for Example 1.4.

3. Hyperbolic

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

4. Eigenvalue

$$-\nabla \cdot (c\nabla u) + au = \lambda du$$

In order to get an idea of how this works, we will consider a number of examples.

2.1 Elliptic PDE

Example 2.1. Consider Poisson's equation on a rectangle $(x, y) \in [0, 2] \times [0, 1]$,

$$\begin{aligned} u_{xx} + u_{yy} &= x^2 + y^2 \\ u(x, 0) = x; &\quad u(x, 1) = 1 \\ u(0, y) = y; &\quad u(2, y) = 1. \end{aligned}$$

In order to solve this equation in MATLAB, we start the MATLAB PDE Toolbox by typing *pdetool* at the MATLAB Command Window prompt. A GUI screen should appear, with a window in which we can draw our domain, in this case a rectangle. We turn the grid on by selecting **Options, Grid**. MATLAB's default domain window is $(x, y) \in [-1.5, 1.5] \times [-1, 1]$, which does not contain the rectangle we would like to define. We can increase the size of MATLAB's domain window by choosing **Options, Axes Limits**. In this case, change the limits on the x axis from the default $[-1.5, 1.5]$ to $[-.5, 2.5]$ and the limits on the y axis from the default $[-1, 1]$ to $[-.5, 1.5]$. **Apply** and **Close**. Next, in order to specify that we would like to draw a rectangle, we click on the rectangle icon at the top left of our menu options. Now, we left-click on the point $(0, 0)$, and keeping the left mouse button pressed, drag the rectangle up to the point $(2,1)$, where we release it. Observe that the exact coordinates of our cursor appear in the top right of the window, and they may not be exactly $(0,0)$ and $(2,1)$. We can correct this by double-clicking on our rectangle and entering exact coordinates for the lower left corner, the rectangle width and the rectangle height. (Notice that MATLAB automatically gives our domain a name, in this case *R1* for rectangle 1.)

Next, we will specify our boundary conditions. We enter MATLAB's *boundary mode* by clicking on the symbol $\partial\Omega$ (or by selecting **Boundary, Boundary mode**, or by typing **Control-b**.) Once boundary mode has been selected, each boundary will appear as a red arrow pointing in the direction of orientation. The color of the boundary indicates the type of condition imposed: red for *Dirichlet* (MATLAB's default), blue for *Neumann*, and green for mixed. We set the boundary condition for $u(x,0)$ by double-clicking on the bottom horizontal line. A pop-up menu will appear, in which we see that MATLAB specifies Dirichlet conditions through the formula $hu = r$. Leaving h specified as 1, we choose r to be x by typing x into the box next to r. Next, we continue around the boundary similarly specifying the remaining conditions.

Before going on to specifying the PDE we want to solve, this is probably a good time to save our work. In order to save these domain specifications for later use, choose **Save As** (or type **Control-s**) and save them as a MATLAB M-file, say *domain1.m.*

Once the geometry and boundary conditions for the problem have been specified, we select the type of PDE we would like to solve by clicking on the PDE icon (or by choosing **PDE, PDE Specification**). MATLAB divides the PDE it can solve into four categories, *elliptic*, *parabolic*, *hyperbolic*, and *eigenmodes.* Poisson's equation is classified as an elliptic PDE. MATLAB's general elliptic PDE specification takes the form

$$-\nabla \cdot (c\nabla u) + au = f,$$

where the operator ∇, typically referred to as *nabla*, is defined through

$$\begin{aligned}\nabla \cdot v =& \frac{\partial v_1}{\partial x} + \frac{\partial v_2}{\partial y}; \quad \text{(divergence)}\\ \nabla f =& (\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}); \quad \text{(gradient)}.\end{aligned}$$

In our case, $c = -1$, $a = 0$, and $f(x,y) = x^2 + y^2$, which we can type directly into the appropriate boxes, with f taking the form *x.ˆ2+y.ˆ2* (that is, array operations are required).

Next, we create a triangular finite element mesh by selecting the $\triangle$ icon (or alternatively choosing **Mesh, Initialize Mesh**). Observe that the initial mesh is fairly coarse (i.e., the grid points are not close together). If this fails to return a suitably accurate solutions, we can refine it by choosing the icon of a triangle inside a triangle. Finally, we are prepared to solve the PDE. For this, we select the = icon (or alternatively choose **Solve, Solve PDE**). MATLAB's default solution mode is a color-scale contour plot. For further options, choose **Plot, Parameters**. For example, by selecting the two options **Height (3-D plot)** and **Plot in x-y grid** (and un-selecting the option **Color**) we get a surface mesh plot (see Figure 2.1). The appearance of the mesh in this plot can be edited in the MATLAB graphics window by first selecting the mesh (by choosing the pointer arrow and clicking on it) and then selecting **Edit, current object properties**. In order to create Figure 2.1, I chose the color tab associated with **Edges** and selected black.

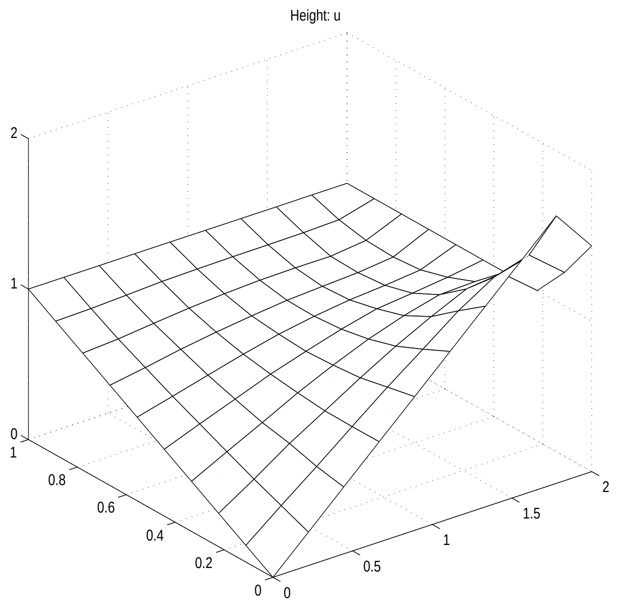

Figure 2.1: Mesh plot for solution to Poission's equation from Example 2.1.

MATLAB's graphics editor has too many options for us to consider exhaustively here, but one more that is certainly worth oberving is the choice **Tools, Rotate 3D**, which allows the plot to be rotated so that it can be viewed from different perspectives.

2.2 Parabolic PDE

We next consider an example from the class of PDE referred to as *parabolic.*

Example 2.2. Consider the heat equation

$$\begin{aligned} u_t =& u_{xx} + u_{yy} + \sin t \\ u(t,0,y) = 0;& \quad u_x(t,\pi,y) = 1 \\ u_y(t,x,0) = 0;& \quad u(t,x,2\pi) = x \\ u(0,x,y) =& 0. \end{aligned}$$

Again, our domain is a simple rectangle, in this case with $(x, y) \in [0, \pi] \times [0, 2\pi]$. We select this in the domain window by setting the lower left corner point to be $(0, 0)$ and specifying the width and height of the domain as *pi* and *2*pi* respectively. We have already seen in the previous example how we specify the Dirichlet boundary conditions $u(t, 0, y) = 0$ and $u(t, x, 2\pi) = x$, so we will focus here on the *Neumann* conditions $u_x(t, \pi, y) = 1$ and $u_y(t, x, 0) = 0$. In order to specify the latter of these, begin by selecting boundary mode and then double-clicking on the bottom horizontal boundary. MATLAB specifies Neumann conditions in the form

$$n * c * \mathrm{grad}(u) + qu = g,$$

which we view as

$$\vec{n} \cdot \Big(c(x,y)\nabla u\Big) + q(x,y)u = g(x,y); \quad (x,y) \in \partial\Omega,$$

where $\vec{n}$ represents a unit vector normal to the domain. (That is, $\vec{n}$ points in the normal direction from the domain (directly outward, perpendicular to the tangent vector) and has unit length.) For the case $u_y(t,x,0) = 0$, we have

$$\begin{aligned}
\vec{n} =& (0,-1) \quad \text{(keep in mind: } \vec{n}\cdot\nabla u = (n_1,n_2)\cdot(u_x,u_y) = n_1 u_x + n_2 u_y) \\
c(x,0) =& 1 \\
q(x,0) =& 0 \\
g(x,0) =& 0,
\end{aligned}$$

of which we specify q and g. (The value of c must correspond with the value of c that arises when we specify our PDE, so we will define it there, keeping in mind that it must be 1.) Similarly, we specify the boundary condition $u_x(t,\pi,y) = 1$ by setting $q = 0$ and $g = 1$. (Observe that our segments of boundary with Dirichlet conditions are shaded red while our segments of boundary with Neumann conditions are shaded blue.)

We now specify our PDE as in Example 2.1, except this time we choose the option **Parabolic**. MATLAB specifies parabolic PDE in the form

$$d * u' - \text{div}(c * \text{grad}(u)) + a * u = f,$$

which we view as

$$d(t,x,y)u_t - \nabla\cdot(c(t,x,y)\nabla u) + a(t,x,y)u = f(t,x,y).$$

In our case $d(t,x,y) = 1$, $c(t,x,y) = 1$, and $f(t,x,y) = \sin t$, which we can type directly into the pop-up menu. (Write $\sin t$ as *sin(t)*.)

Next, we set our initial condition and our times for solution by choosing **Solve, Parameters**. A vector of times for which we want solution values is specified under **Time:** and the initial condition is specified under **u(t0)**. A good way to specify time is with the linspace command, so we type *linspace(0,5,10)*, which runs time from 0 to five seconds, with ten points. In this case, we can leave the initial condition specified as 0.

Finally, we introduce a mesh by clicking the $\triangle$ icon and then solve with $=$. (If we select $=$ prior to specifying a mesh, MATLAB will automatically specify a mesh for us.) As before, MATLAB returns a color scale solution, this time at the final time of solution, $t = 5$. In order to get an idea of the time evolution of the solutions, we select **Plot, Parameters** and choose **Animation**. For example, we can set **Height (3-D plot), Animation,** and **Plot in x-y grid**, which will produce an evolving mesh plot.

Final remark. In this example, we have specified c as a constant scalar. It can also be specified as a non-constant scalar or, more generally, as a 2×2 non-constant matrix.

3 Linear systems in two space dimensions

We next consider the case of solving linear systems of PDE in time and two space variables.

3.1 Two Equations

Linear systems consisting of two equations can still be solved directly in PDE Toolbox.

Example 3.1. Consider the following system of two linear parabolic equations defined on a circle centered at the origin with radius 1 (denoted Ω).

$$\begin{aligned}
u_{1_t} + (1/(1+x^2))u_1 + u_2 &= u_{1_{xx}} + u_{1_{yy}} \\
u_{2_t} + u_1 + u_2 &= u_{1_{xx}} + u_{2_{yy}} \\
u_1(t,x,y) &= e^{-(x^2+y^2)}, \quad x<0, (x,y) \in \partial\Omega \\
u_2(t,x,y) &= \sin(x+y), \quad x<0, (x,y) \in \partial\Omega \\
\vec{n} \cdot \nabla \vec{u} &= 0, \quad x>0 \\
u_1(0,x,y) &= e^{-(x^2+y^2)} \\
u_2(0,x,y) &= \sin(x+y).
\end{aligned}$$

We begin solving these equations by typing *pdecirc(0,0,1)* in the MATLAB command window. (The usage of *pdecirc* is *pdecirc(*x_{center}*,*y_{center}*,radius,label)*, where the label can be omitted.) This will open the MATLAB Toolbox GUI and create a circle centered at the orgin with radius 1. The first thing we need to choose in this case is **Options, Application, Generic System**. Next, enter boundary mode and observe that MATLAB expects boundary conditions for a system of two equations. (That's what it considers a generic system. For systems of higher order, we will have to work a little harder.) MATLAB specifies Dirichlet boundary conditions in such systems in the form

$$\begin{pmatrix} h_{11} & h_{12} \\ h_{21} & h_{22} \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \end{pmatrix} = \begin{pmatrix} r_1 \\ r_2 \end{pmatrix}.$$

Define the two Dirichlet boundary conditions by choosing $h_{11} = h_{22} = 1$ and $h_{12} = h_{21} = 0$ (which should be MATLAB's default values), and by choosing r_1 to be *exp(-x.^2-y.^2)* (*don't omit the array operations*) and r_2 to be *sin(x+y)*. MATLAB specifies Neumann boundary conditions in such systems in the form

$$\vec{n} \cdot (c \otimes \nabla \vec{u}) + q\vec{u} = g,$$

where

$$\vec{n} = \begin{pmatrix} n_1 \\ n_2 \end{pmatrix}, \quad c = \begin{pmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{pmatrix}, \quad q = \begin{pmatrix} q_{11} & q_{12} \\ q_{21} & q_{22} \end{pmatrix}, \quad \text{and } g = \begin{pmatrix} g_1 \\ g_2 \end{pmatrix},$$

and the k^{th} component of $c \otimes \nabla \vec{u}$ is defined by

$$\{c \otimes \nabla \vec{u}\}_k = \begin{pmatrix} c_{11}u_{k_x} + c_{12}u_{k_y} \\ c_{21}u_{k_x} + c_{22}u_{k_y} \end{pmatrix},$$

so that

$$\vec{n} \cdot (c \otimes \nabla \vec{u}) = \begin{pmatrix} n_1c_{11}u_{1_x} + n_1c_{12}u_{1_y} + n_2c_{21}u_{1_x} + n_2c_{22}u_{1_y} \\ n_1c_{11}u_{2_x} + n_1c_{12}u_{2_y} + n_2c_{21}u_{2_x} + n_2c_{22}u_{2_y} \end{pmatrix}.$$

(More generally, if the diffusion isn't the same for each variable, c can be defined as a tensor, see below.) In this case, taking q and g both zero suffices. (The matrix c will be defined in the problem as constant, identity.)

Next, specify the PDE as **parabolic**. For parabolic systems, MATLAB's specification takes the form

$$du_t - \nabla \cdot (c \otimes \nabla u) + au = f,$$

where

$$u = \begin{pmatrix} u_1 \\ u_2 \end{pmatrix}, \quad f = \begin{pmatrix} f_1 \\ f_2 \end{pmatrix}, \quad a = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}, \quad d = \begin{pmatrix} d_{11} & d_{12} \\ d_{21} & d_{22} \end{pmatrix},$$

and

$$\nabla \cdot (c \otimes \nabla u) = \begin{pmatrix} c_{11}u_{1_{xx}} + c_{12}u_{1_{yx}} + c_{21}u_{1_{xy}} + c_{22}u_{1_{yy}} \\ c_{11}u_{2_{xx}} + c_{12}u_{1_{yx}} + c_{21}u_{1_{xy}} + c_{22}u_{1_{yy}} \end{pmatrix}.$$

In this case, we take $c_{11} = c_{22} = 1$ and $c_{12} = c_{21} = 0$. Also, we have

$$\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} = \begin{pmatrix} \frac{1}{1+x^2} & 1 \\ 1 & 1 \end{pmatrix}, \quad \begin{pmatrix} d_{11} & d_{12} \\ d_{21} & d_{22} \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \quad \text{and} \quad \begin{pmatrix} f_1 \\ f_2 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix},$$

which can all be specified by typing valid MATLAB expressions into the appropriate text boxes. (For the expression $\frac{1}{1+x^2}$, we must use array operations, *1./(1+x.^2)*.)

We specify the initial conditions by selecting **Solve, Parameters**. In this case, we set the time increments to be *linspace(0,10,25)*, and we specify the vector initial values in *u(t0)* as *[exp(-x.^2-y.^2);sin(x+y)]*. Finally, solve the problem by selecting the = icon. (MATLAB will create a mesh automatically.)

The first solution MATLAB will plot is a color plot of $u_1(x, y)$, which MATLAB refers to as u. In order to view a similar plot of u_2, choose **Plot, Parameters** and select the **Property v**.

4 Nonlinear elliptic PDE in two space dimensions

Though PDE Toolbox is not generally equipped for solving nonlinear problems directly, in the case of elliptic equations certain nonlinearities can be accomodated.

4.1 Single nonlinear elliptic equations

Example 4.1. Consider the nonlinear elliptic PDE in two space dimensions, defined on the ball of radius 1,

$$\begin{aligned} \triangle u + u(1 - u_x - u_y) &= 2u^2 \\ u(x, y) = 1; &\forall (x, y) \in \partial B(0, 1). \end{aligned}$$

We begin solving this equation in MATLAB by typing *pdecirc(0,0,1)* at the MATLAB prompt. Proceeding as in the previous examples, we set the boundary condition to be Dirichlet and identically 1, and then choose **PDE Specification** and specify the PDE as

Elliptic. In this case, we must specify c as *1.0*, a as *-(1-ux-uy)* and f as *-2u.^2*. The key point to observe here is that nonlinear terms can be expressed in terms of u, u_x, and u_y, for which MATLAB uses respectively *u, ux,* and *uy*. Also, we observe that array operations must be used in the expressions. Next, in order to solve the nonlinear problem, we must choose **Solve, Parameters** and specify that we want to use MATLAB's nonlinear solver. In this case, the default nonlinear tolerance of *1e-4* and the designation of **Jacobian** as **Fixed** are sufficient, and we are ready to solve the PDE by selecting the icon =.

5 General nonlinear systems in two space dimensions

5.1 Parabolic Problems

While MATLAB's PDE Toolbox does not have an option for solving nonlinear parabolic PDE, we can make use of its tools to develop short M-files that will solve such equations.

Example 5.1. Consider the Lotka–Volterra predator–prey model in two space dimensions,

$$\begin{aligned}u_{1_t} =& c_{11}u_{1_{xx}} + c_{12}u_{1_{yy}} + a_1u_1 - r_1u_1u_2\\ u_{2_t} =& c_{21}u_{2_{xx}} + c_{22}u_{2_{xx}} - a_2u_2 + r_2u_1u_2,\end{aligned}$$

where $u_1(t,x,y)$ represents prey population density at time t and position (x,y) and $u_2(t,x,y)$ represents predator population density at time t and position (x,y). For a_1, b_1, a_2, and b_2, we will take values obtained from an ODE model for the Hudson Bay Company Hare–Lynx example: $a_1 = .47$, $r_1 = .024$, $a_2 = .76$, and $r_2 = .023$. For the values c_{kj}, we take $c_{11} = c_{12} = .1$ and $c_{21} = c_{22} = .01$, which signifies that the prey diffuse through the domain faster than the predators. MATLAB's PDE Toolbox does not have an option for solving an equation of this type, so we will proceed through an iteration of the form

$$\begin{aligned}u_{1_t}^{n+1} - c_{11}u_{1_{xx}}^{n+1} - c_{12}u_{1_{yy}}^{n+1} - a_1u_1^{n+1} =& - r_1u_1^nu_2^n\\ u_{2_t}^{n+1} - c_{21}u_{2_{xx}}^{n+1} - c_{22}u_{2_{yy}}^{n+1} + a_2u_2^{n+1} =& r_2u_1^nu_2^n.\end{aligned} \tag{5.1}$$

That is, given u_1 and u_2 at some time t_0 (beginning with the initial conditions), we solve the linear parabolic equation over a short period of time to determine values of u_1 and u_2 at time t_1.

In general, initial and boundary conditions can be difficult to pin down for problems like this, but for this example we will assume that the domain is square of length 1 (denoted S), that neither predator nor prey enters or exits the domain, and that initially the predator density is concentrated at the edges of the domain and the prey density is concentrated at the center. In particular, we will assume the following:

$$\begin{aligned}\vec{n}\cdot\nabla u_1 =& 0, \forall x \in \partial S\\ \vec{n}\cdot\nabla u_2 =& 0, \forall x \in \partial S\\ u_1(0,x,y) =& \begin{cases}1, & (x-\frac{1}{2})^2 + (y-\frac{1}{2})^2 \le \frac{1}{16}\\ 0, & \text{otherwise}\end{cases}\\ u_2(0,x,y) =& \begin{cases}1, & (x-\frac{1}{2})^2 + (y-\frac{1}{2})^2 \ge \frac{1}{4}\\ 0, & \text{otherwise.}\end{cases}\end{aligned}$$

Though we will have to carry out the actual calculation with an M-file, we will first create the domain and define our boundary conditions using PDE Toolbox. To begin, at the MATLAB command line prompt, type *pderect([0 1 0 1])*, which will initiate a session with PDE Toolbox and define a square of length one with lower left corner at the origin. (The exact usage of *pderect* is *pderect([xmin xmax ymin ymax])*). Since the upper edge of this square is on the edge of our window, choose **Options, Axes Equal**, which will expand the y axis to the interval $[-1.5, 1.5]$. Next, choose boundary mode, and then hold the **Shift** key down while clicking one after the other on each of the borders. When they are all selected, click on any one of them and set the boundary condition to be Neumann with g and q both 0. Once the boundary conditions are set, export them by selecting **Boundary, Export Decomposed Boundary**. The default boundary value assignments are q and g. For clarity, rename these $q1$ and $g1$ to indicate that these are the boundary conditions for u_1. (Though for this problem the boundary conditions for u_1 and u_2 are the same, for generality's sake, we will treat them as if they were different.) For u_2, export the boundary again and this time label as $q2$ and $g2$. The last thing we can do in the GUI window is create and export our triangulation, so select the icon △ to create a mesh and select **Mesh, Export Mesh** to export it. The three variables associated with the mesh are p, e, and t, vectors containing respectively the points if the triangulation, the edges of the triangulation, and an index of the triangulation.

At this point it's a good idea to save these variables as a MATLAB workspace (.mat file). To do this, choose **File**, **Save Workspace As**. Finally, we store the initial conditions $u_1(0, x, y)$ and $u_2(0, x, y)$ in the function M-file *lvinitial.m*.

```
function [u1initial,u2initial] = lvinitial(x,y)
%LVINITIAL: MATLAB function M-file that contains the
%initial population distributions for the Lotka-Volterra model.
if (x-1/2)^2+(y-1/2)^2<=1/16
u1initial=100;
else
u1initial=0;
end
if (x-1/2)^2+(y-1/2)^2<=1/4
u2initial=0;
else
u2initial=10;
end
```

Now, we can solve the PDE with the MATLAB M-file *lvpde.m*. While this file might look prohibitively lengthy, it's actually fairly simple. For example, the long sections in bold type simply plot the solution and can be ignored with regard to understanding how the M-file works.

```
%LVPDE: MATLAB script M-file for solving the PDE
%Lotka-Volterra system.
%
%Parameter definitions
```

```
a1=.47; r1=.024; a2=.76; r2=.023;
m=size(p,2); %Number of endpoints
n=size(t,2); %Number of triangles
t_final=1.0; %Stop time
M=30;
dt=t_final/M; %Time-stepping increment (M-file time-stepping)
tlist=linspace(0,dt,2); %Time vector for MATLAB's time-stepping
%Rectangular coordinates for plotting
x=linspace(0,1,25);
y=linspace(0,1,25);
%Set diffusion
c1=.1; %Prey diffusion
c2=.01; %Predator diffusion
%Initial conditions
for i=1:m %For each point of the triangular grid
[u1old(i),u2old(i)]=lvinitial(p(1,i),p(2,i));
end
%
for k=1:M
%Nonlinear interaction
for i=1:m
f1(i)=-r1*u1old(i)*u2old(i);
f2(i)=r2*u1old(i)*u2old(i);
end
%NOTE: The nonlinear interaction terms must be defined at the centerpoints
%of the triangles. We can accomplish this with the function
%pdeintrp (pde interpolate).
f1center=pdeintrp(p,t,f1');
f2center=pdeintrp(p,t,f2');
%Solve the PDE
u1new=parabolic(u1old,tlist,b1,p,e,t,c1,-a1,f1center,1);
u2new=parabolic(u2old,tlist,b2,p,e,t,c2,a2,f2center,1);
%Update u1old, u2old
u1old=u1new(:,2);
u2old=u2new(:,2);
%Plot each iteration
u1=tri2grid(p,t,u1old,x,y);
u2=tri2grid(p,t,u2old,x,y);
subplot(2,1,1)
%imagesc(x,y,u1,[0 10])
%colorbar
mesh(x,y,u1)
axis([0 1 0 1 0 10])
subplot(2,1,2)
%imagesc(x,y,u2, [0 10])
```

```
%colorbar
mesh(x,y,u2)
axis([0 1 0 1 0 10])
pause(.1)
%
end
```

In general, the function

```
parabolic(u0,tlist,b,p,e,t,c,a,f,d)
```

solves the the single PDE

$$du_t - \nabla \cdot (c\nabla u) + au = f.$$

or the system of PDEs

$$du_t - \nabla \cdot (c \otimes \nabla u) + au = f.$$

In this case, according to (5.1), we take the nonlinearity as a driving term from the previous time step, and the remaining linear equations are decoupled, so that we solve two single equations rather than a system.

A critical parameter in the development above is M, which determines how refined our time-stepping will be (the larger M is, the more refined our analysis is). We can heuristically check our numerical solution by increasing the value of M and checking if the solution remains constant.

The plotting code creates a window in which mesh plots of both the predator and prey population densities are plotted. These are updated at each iteration, so running this code, we see a slow movie of the progression. Example plots of the initial and final population densities are given in Figures 5.1 and 5.2. Another good way to view the solution is through a color pixel plot, created by the command *imagesc*. If we comment out the *mesh* and *axis* commands above and add the *imagesc* and *colorbar* commands instead, we can take a bird's eye view of a color-coded depiction of the dynamics.

6 Defining more complicated geometries

One of the biggest advantages in using the GUI interface of MATLAB's PDE toolbox is the ease with which fairly complicated geometries can be defined and triangularized.

7 FEMLAB

7.1 About FEMLAB

FEMLAB is a program developed by COMSOL Ltd. for solving PDE numerically based on the finite element method. COMSOL Ltd. is the same group who developed MATLAB's PDE Toolbox, and consequently the GUI for FEMLAB is conveniently similar to the GUI for PDE Toolbox. The difference between the two programs is that FEMLAB is considerably more general. Some fundamental features that FEMLAB offers and PDE Toolbox does not are:

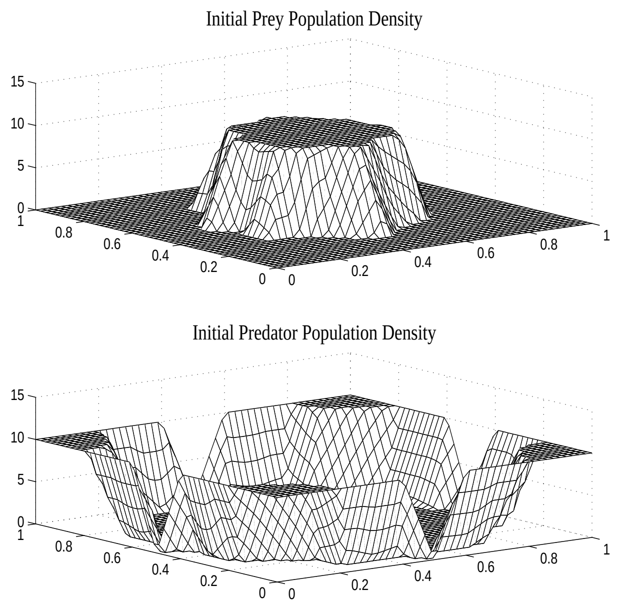

Figure 5.1: Initial population densities for predator–prey example.

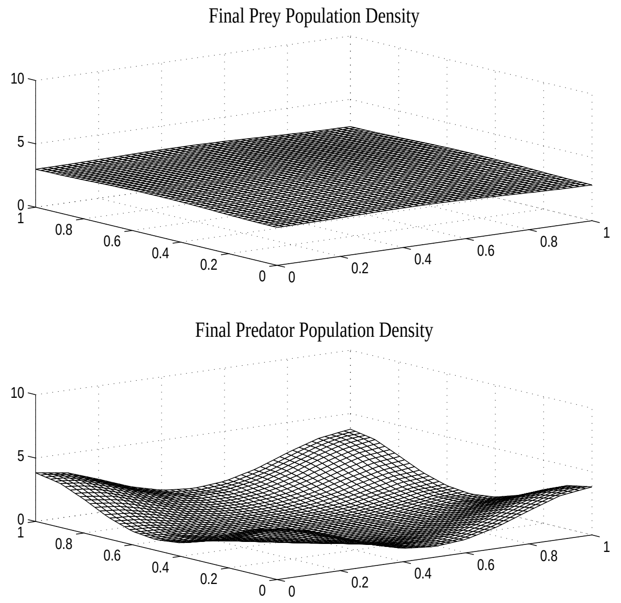

Figure 5.2: Final population densities for predator–prey example.

1. The ability to solve PDE in three space dimensions
2. The ability to solve several additional predefined equations, including
   (a) Navier–Stokes
   (b) Reaction–convection–diffusion equations
   (c) Maxwell's equations for electrodynamics
3. The ability to solve nonlinear systems of equations directly from the GUI interface.

7.2 Getting Started with FEMLAB

FEMLAB is such a broad program that it's easy on first glance to get lost in the options. Though our eventual goal in using FEMLAB is to solve fairly complicated equations that PDE Toolbox is not equipped for, we will begin, as we did with PDE Toolbox, with a simple example.

Example 7.1. Consider Poisson's equation on the ball of radius 1,

$$\begin{aligned} \triangle u =& u(1-u), \quad (x,y) \in B(0,1) \\ u(x,y) =& x^3 + y^3, \quad (x,y) \in \partial B(0,1). \end{aligned}$$

We open FEMLAB at the MATLAB Command Window prompt by typing *femlab*. A geometry window should open with a pop-up menu labeled **Model Navigator**. We observe immediately that FEMLAB offers the choice of one, two, or three dimensions. We choose **2D** (which should be the default) and then double-click on **Classical PDEs**. FEMLAB's options under **Classical PDEs** are:

- Laplace's equation
- Poisson's equation
- Helmholtz's equation
- Heat equation
- Wave equation
- Schrodinger equation
- Convection–diffusion equation

We can select the option **Poisson's equation** by double-clicking on it, after which we observe that *Poisson's Equation* appears in the upper left corner of the FEMLAB geometry window. In this case, the default grid in the geometry window is too small, so we increase it by selecting **Options, Axes/Grid Settings** and specifying a y range between -1.5 and 1.5. We can now draw a circle of radius 1 by selecting the ellipse icon from the left panel of the window, clicking on the point $(0,0)$ and dragging the radius to 1. By default, FEMLAB labels this region *E1* for ellipse 1. As in PDE Toolbox, we can double-click anywhere in the region to alter or refine its definition. Next, we choose **Boundary, Boundary mode** and specify our boundary condition by selecting each part of the curve and choosing the Dirichlet boundary conditions with h as 1 and r as *x.ˆ3+y.ˆ*3. (For the moment, we will do well to ignore the options for more complicated boundary value selections.) Next, we need to specify our governing equation. FEMLAB is set up so that different equations can be specified in different regions of the domain, so equation specification is made under the option **Subdomain, Subdomain Settings**. FEMLAB specifies Poisson's equation in the form

$$-\nabla \cdot (c\nabla u) = f,$$

so in this case we choose c to be 1 and f to be *-u.*(1-u)*. Finally, we solve the PDE by selecting **Solve, Solve Problem**. We observe that FEMLAB offers a number of options for viewing the solutions, listed as icons on the left corner of the geometry window.

Index