3. Generate the shape functions

$$N_j^{(i)} = a_j^{(i)} + b_j^{(i)} x + c_j^{(i)} y \tag{11.2.17}$$

   where $i = 1, 2, 3, ..., m$ and $j = 1, 2, 3, ..., m$.

4. Compute the integrals for matrix elements α_{ij} and vector elements β_j in the interior and boundary.

5. Construct the matrix $\mathbf{A}$ and vector $\mathbf{b}$.

6. Solve $\mathbf{Ax} = \mathbf{b}$.

7. Plot the solution $u(x, y) = \sum_{i=1}^{m} \gamma_i \phi_i(x, y)$.

For a wide variety of problems, the above procedures can simply be automated. That is exactly what commercial packages do. Thus once the coefficients and boundary conditions associated with (11.2.1), (11.2.2), and (11.2.3) are known, the solution procedure is straightforward.

11.3 MATLAB for Partial Differential Equations

Given the ubiquity of partial differential equations, it is not surprising that MATLAB has a built in PDE solver: **pdepe**. Thus the time and space discretization, as well as time-stepping within the CFL tolerances, are handled directly as a subroutine call to MATLAB. This is similar to using a differential equation solver such as **ode45**. The following specific PDE can be solved with **pdepe**:

$$c\left(x, t, u, \frac{\partial u}{\partial x}\right) \frac{\partial u}{\partial t} = x^{-m} \frac{\partial}{\partial x}\left[x^m f\left(x, t, u, \frac{\partial u}{\partial x}\right)\right] + s\left(x, t, u, \frac{\partial u}{\partial x}\right) \tag{11.3.1}$$

where $u(x, t)$ generically is a vector field, $t \in [t_-, t_+]$ is the finite time range to be solved for, $x \in [a, b]$ is the spatial domain of interest, and $m = 0, 1$ or 2. The integer m arises from considering the Laplacian operator in cylindrical and spherical coordinates for which $m = 1$ and $m = 2$ respectively. Note that $c(x, t, u, u_x)$ is a diagonal matrix with identically zero or positive coefficients. The functions $f(x, t, u, u_x)$ and $s(x, t, u, u_x)$ correspond to a flux and source term respectively. All three functions allow for time and spatially dependent behavior that can be nonlinear.

In addition to the PDE, boundary conditions must also be specific. The specific form required for **pdepe** can be nonlinear and time dependent so that:

$$p(x, t, u) + q(x, t) g(x, t, u, u_x) = 0 \text{ at } x = a, b\,. \tag{11.3.2}$$

As before $p(x, t, u)$ and $g(x, t, u, u_x)$ is time and spatially dependent behavior that can be nonlinear. The function $q(x, t)$ is simply time and space dependent.

Heat Equation

To start, consider the simplest PDE: the heat equation:

$$\pi^2 \frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} \tag{11.3.3}$$

where the solution is defined on the domain $x \in [0, 1]$ with the boundary conditions

$$u(0, t) = 0 \tag{11.3.4a}$$

$$\pi \exp(-t) + \frac{\partial u(1, t)}{\partial t} = 0 \tag{11.3.4b}$$

and initial conditions

$$u(x, 0) = \sin(\pi x)\,. \tag{11.3.5}$$

This specifies the PDE completely and allows for the construction of a unique solution. In MATLAB, the **pdepe** function call relies on three subroutines that specify the PDE, initial conditions and boundary conditions. Before calling these, the time and space grid must be defined. The following code defines the time domain, spatial discretization and plots the solution

```
m = 0;
x = linspace(0,1,20);
t = linspace(0,2,5);
u = pdepe(m,'pdex1pde','pdex1ic','pdex1bc',x,t);
surf(x,t,u)
```

It only remains to specify the three subroutines. The PDE subroutine is constructed as follows.

pdex1pde.m

```
function [c,f,s] = pdex1pde(x,t,u,DuDx)
c = pi^2;
f = DuDx;
s = 0;
```

In this case, the implementation is fairly straightfoward since $s(x, t, u, u_x) = 0$, $m = 0$, $c(x, t, u, u_x) = \pi^2$ and $f(x, t, u, u_x) = u_x$. The initial condition is quite easy and can be done in one line.

pdex1ic.m

```
function u0 = pdex1ic(x)
u0 = sin(pi*x);
```

Finally, the left and right boundaries can be implemented by specifying the function q, q and g at the right and left.

pdex1bc.m

```
function [pl,ql,pr,qr] = pdex1bc(xl,ul,xr,ur,t)
pl = ul;
ql = 0;
pr = pi * exp(-t);
qr = 1;
```

Combined, the subroutines quickly and efficiently solve the heat equation with a time-dependent boundary condition.

Fitzhugh-Nagumo Equation

Overall, the combination of (11.3.1) with boundary conditions (11.3.2) allows for a fairly broad range of problems to solve. As a more sophisticated example, the Fitzhugh-Nagumo equation is considered which models the voltage dynamics in neurons. The Fitzhugh-Nagumo model supports the propagation of voltage spikes that are held together in a coherent structure through the interaction of diffusion, nonlinearity and coupling to a recovery field. Thus we will consider the following system of PDEs:

$$\frac{\partial V}{\partial t} = D\frac{\partial^2 V}{\partial x^2} + V(a - V)(V - 1) - W \tag{11.3.6a}$$

$$\frac{\partial W}{\partial t} = bV - cW\,. \tag{11.3.6b}$$

where $V(x,t)$ measures the voltage in the neuron. The following initial conditions will be applied that generate a voltage spike moving from left to right in time.

$$V(x,0) = \exp(-x^2) \tag{11.3.7a}$$

$$W(x,0) = 0.2\exp(-(x+2)^2)\,. \tag{11.3.7b}$$

Finally, boundary conditions must be imposed on the PDE system. For convenience, no flux boundary conditions will be applied at both ends of the computational domain so that

$$\frac{\partial V}{\partial x} = 0 \text{ and } \frac{\partial W}{\partial x} = 0 \text{ at } x = a, b \tag{11.3.8}$$

The partial differential equation along with the boundary conditions and initial conditions completely specify the system.

The Fitzhugh-Nagumo PDE system is a vector sytem for the field variables $V(x,t)$ and $W(x,t)$. Before calling on **pdepe**, the time and space domain discretization is defined along with the parameter m:

```
m = 0;
x = linspace(-10,30,400);  % space
t = linspace(0,400,20);    % time
```

In this example, the time and space domains are discretized with equally spaced Δt and Δx. However, an arbitrary spacing can be specified. This is especially important in the spatial domain as there can be, for instance, boundary layer effects near the boundary where a higher density of grid points should be imposed.

The next step is to call upon **pdepe** in order to generate the solution. The following code picks the parameters a, b and c in the Fithugh-Nagumo and passes them into the PDE solver. The final result is the generation of a matrix **sol** that contains the solution matrices for both $V(x,t)$ and $W(x,t)$.

```
A=0.1;  B=0.01; C=0.01;
sol = pdepe(m,'pde_fn_pde','pde_fn_ic','pde_fn_bc',x,t,[],A,B,C);

u1 = sol(:,:,1);
u2 = sol(:,:,2);

waterfall(x,t,u1), map=[0 0 0]; colormap(map), view(15,60)
```

Upon solving, the matrix **sol** is generates which is $20 \times 400 \times 2$. The first layer of this matrix cube is the voltage $V(x,t)$ in time and space ($\mathbf{u_1}$) and the second layer is the recovery field $W(x,t)$ in time and space ($\mathbf{u_2}$). The voltage, and the propagating wave is plotted using **waterfall**.

What is left is to define the PDE itself (**pdepe_fn_pde.m**), its boundary conditions (**pdepe_fn_bc.m**) and its initial conditions (**pdepe_fn_ic.m**). We begin by developing the function call associated with the PDE itself.

pdepe_fn_pde.m

```
function [c,f,s] = pde_fn_pde(x,t,u,DuDx,A,B,C)
c = [1; 1];              % c diagonal terms
f = [0.01; 0] .* DuDx;   % diffusion term

rhsV = u(1)*(A-u(1))*(u(1)-1) - u(2);
rhsW = B*u(1)-C*u(2);
s = [rhsV; rhsW];
```

Note that the specification of $c(x,t,u,u_x)$ in the PDE must be diagonal matrix. Thus the PDE call only allows for the placement of these diagonal elements into a vector that must have positive coefficients. The diffusion operator and the derivative are specified by **f** and **DuDx** respectively. Once the appropriate terms are constructed, the right-hand side is fully formed in the matrix **s**. Note

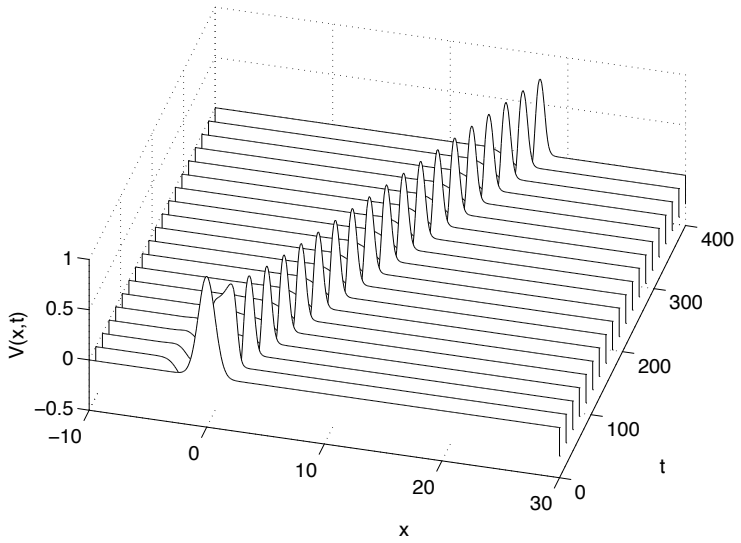

Figure 92: The solution of the Fitzhugh-Nagumo partial differential equation with no flux boundary conditions.

the similarity between this and the construction involved in **ode45**. We next implement the initial conditions.

pdepe_fn_ic.m

```
function u0 = pde_fn_ic(x,A,B,C)
u0 = [1*exp(-((x)/1).^2)
    0.2*exp(-((x+2)/1).^2)];
```

Arbitrary initial conditions can be applied, but preferably, one would implement initial conditions consistent with the boundary conditions. The boundary conditions are implemented in the following code.

pdepe_fn_bc.m

```
function [pl,ql,pr,qr] = pde_fn_bc(xl,ul,xr,ur,t,A,B,C)
pl = [0; 0];
ql = [1; 1];
pr = [0; 0];
qr = [1; 1];
```

Here the **pl** and **ql** are the functions $p(x,t,u)$ and $q(x,t)$ respectively at the left boundary point while **pr** and **qr** are the functions $p(x,t,u)$ and $q(x,t)$ at the right boundary point.

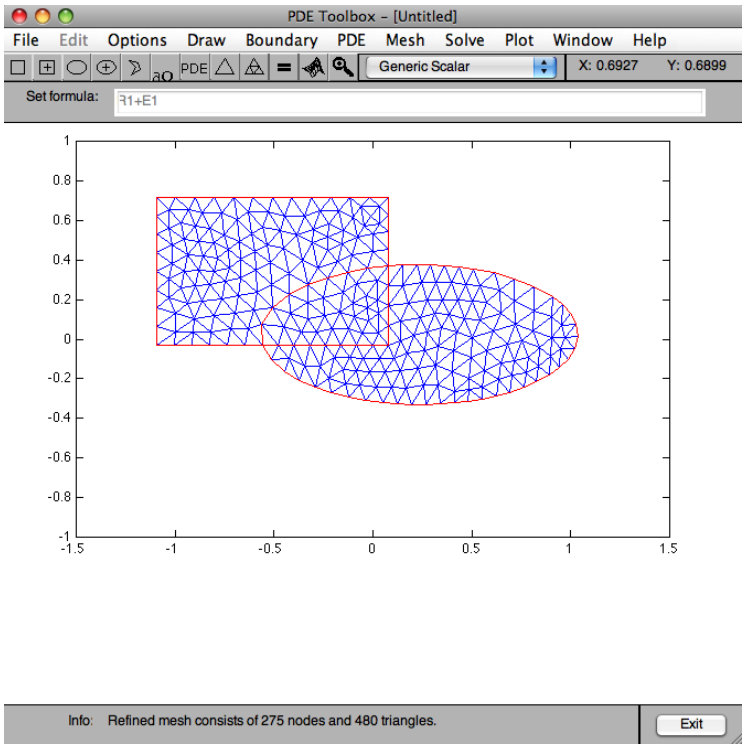

Figure 93: Graphical user interface of the partial differential equations GUI. The interface is activated by the command **pdetool**.

Figure 92 demonstrates the formation of a spike and its propagation using the **pdepe** methodology. Note that the time step used is not determined by the initial time vector used for output. Rather, just like the adaptive differential equation steppers, a step size Δt is chosen to ensure a default accuracy of 10^{-6}. The spatial discretization, however, is absolutely critical as the method uses the user specified mesh in space. Thus it is imperative to choose a fine enough mesh in order to avoid numerical inaccuracies and instabilities.

11.4 MATLAB Partial Differential Equations Toolbox

In addition to the **pdepe** function call, MATLAB has a finite element based PDE solver. Unlike **pdepe**, which provides solutions to one-dimensional parabolic and elliptic type PDEs, the PDE toolbox allows for the solution of linear, two-dimensional PDE systems of parabolic, elliptic and hyperbolic type along with

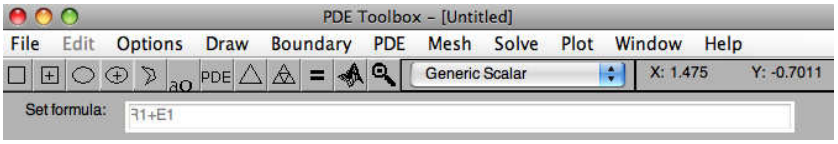

Figure 94: User interface for PDE specification along with boundary conditions and domain. The icons can be activated from left to right in order to (i) specify the domain, (ii) specify the boundary conditions, (iii) specify the PDE, (iv) implement a mesh, and (v) solve the PDE.

eigenvalue problems. In particular, the toolbox allows for the solution of

$$elliptic: \quad Lu = f(x,y,t) \tag{11.4.1a}$$
$$eigenvalue: \quad Lu = \lambda d(x,y,t)u \tag{11.4.1b}$$
$$parabolic: \quad d(x,y,t)\frac{\partial u}{\partial t} + Lu = f(x,y,t) \tag{11.4.1c}$$
$$hyperbolic: \quad d(x,y,t)\frac{\partial^2 u}{\partial t^2} + Lu = f(x,y,t) \tag{11.4.1d}$$

where L denotes the spatial linear operator

$$Lu = -\nabla \cdot (c(x,y,t)\nabla u) + a(x,y,t)u \tag{11.4.2}$$

where x and y are on a given domain Ω. The functions $d(x,y,t)$, $c(x,y,t)$, $f(x,y,t)$ and $a(x,y,t)$ can be fairly arbitrary, but should be, generically, well behaved on the domain Ω.

The boundary of the domain, as will be shown, can be quite complicated. On each portion of the domain, boundary conditions must be specified which are of the Dirichlet or Neumann type. In particular, each portion of the boundary must be specified as one of the following:

$$Dirichlet: \quad h(x,y,t)u = r(x,y,t) \tag{11.4.3a}$$
$$Neumann: \quad c(x,y,t)(\mathbf{n}\cdot\nabla u) + q(x,y,t)u = g(x,y,t) \tag{11.4.3b}$$

where $h(x,y,t)$, $r(x,y,t)$, $c(x,y,t)$, $q(x,y,t)$ and $g(x,y,t)$ can be fairly arbitrary and $\mathbf{n}$ is the unit normal vector on the boundary. Thus the flux across the boundary is specified with the Neumann boundary conditions.

In addition to the PDE itself and its boundary conditions, the initial condition must be specified for the parabolic and hyperbolic manifestation of the equation. The initial condition is a function specified over the entire domain Ω.

Although the PDE toolbox is fairly general in terms of its non-constant coefficient specifications, it is a fairly limited solver given that it can only solve linear problems in two-dimensions with up to two space and time derivatives. If considering a problem that can be specified by the conditions laid out above,

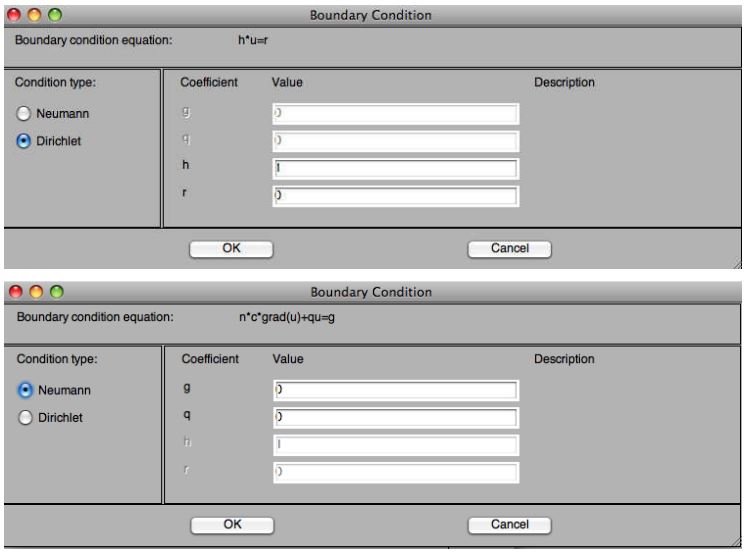

Figure 95: User interface for PDE boundary condition specification. Simply double click on a given boundary and specify Dirichlet (top figure) or Neumann (bottom figure) on the boundary selected. Note that the boundary will appear red if it is Dirichlet and blue if it is Neumann.

then it can provide a fast and efficient solution method with very little development costs and overhead. Parenthetically, it should be noted that a wide variety of other professional grade tools exist that also require specifying the above constraints. And in general, every one of these PDE toolboxes is limited by a constrained form of PDE and boundary conditions. To access the toolbox in MATLAB, simply type

```
pdetool
```

in MATLAB and a graphical user interface (GUI) will appear guiding one through the setup of the PDE solver.

Figure 93 shows the basic user interface associated with the MATLAB PDE toolbox GUI. An initial domain and mesh are displayed for demonstration purposes. In what follows, a step-by-step outline of the use of the GUI will be presented. Indeed, the GUI allows for a natural progression that specifies (i) the domain on which the PDE is to be considered, (ii) the specific PDE to be solved, (iii) the boundary conditions on each portion of the domain, (iv) the resolution and grid for which to generate the solution. Generically, each of these steps is performed by following the graphical icons aligned on the top of the GUI (See Fig. 94).

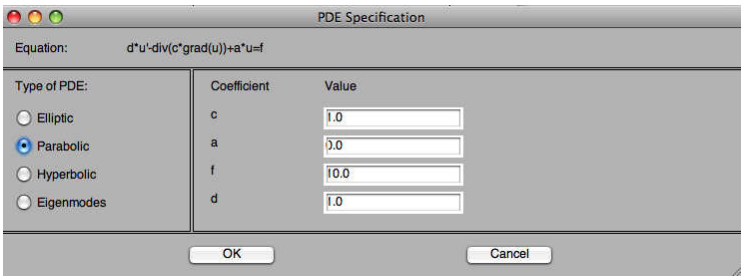

Figure 96: User interface for PDE specification. Four choices are given representing the equations given in (11.4.1).

Domain Specification

To begin, the domain is specified using a variety of drawing tools in the PDE toolbox. In particular, one can draw rectangles (boxes), ellipses (circles) and/or polygons. The first five icons of the top bar of the GUI, demonstrated in Fig. 94, shows the various options available. In particular, the first icon allows one to draw a rectangle (box) where two corners of the rectangle are specified (first icon) or the center and outside of the rectangle are specified (second icon). Similarly, the third and fourth icon specify the outsides of an ellipse (circle) or its center and semi-major/minor axes. The fifth icon allows one to specify a closed polygon for a boundary. One can also import a boundary shape and/or rotate a given object of interest. The **set formula** Fig. 94 allows you to specify wether the domains drawn are added together, or in case of generating holes in the domain, subtracted out. The rectangles, ellipses and polygons are denoted by Rk, Ek and Pk where k is the numbering of the object. Figure 93, for instance, shows the formula $R1 + E1$ since the domain is comprised of a rectangle and an ellipse.

Boundary Condition Specification

The sixth icon is given by $\partial\Omega$ and represents the specification of the boundary conditions. Either Dirichlet or Neumann are specified in the form of (11.4.3). Figure 95 shows the GUI that appears for specifying both types of boundary conditions. The default settings for Dirichlet boundary conditions are $h(x, y, t) = 1$ and $r(x, y, t) = 0$. The default settings for Neumann boundaries are $q(x, y, t) = g(x, y, t) = 0$.

PDE Specification

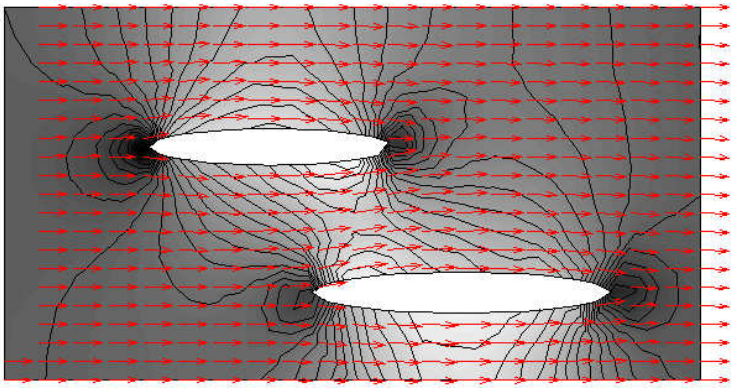

Figure 97: Plot of the solution of Laplace's equation with flux conditions on the left and right boundaries and no-flux on the top and bottom and internal boundaries. This represents a potential solution to the stream function equation where the gradient (velocity) field is plotted representing the flow lines.

The PDE to be solved is specified by the seventh icon labeled PDE. This will allow the user to specify one of the PDEs given in (11.4.1). Figure 96 demonstrates the PDE specification GUI and its four options. A parabolic PDE specification is demonstrated in the figure. The default settings are $c(x, y, t) = d(x, y, t) = 1$, $a(x, y, t) = 0$ and $f(x, y, t) = 10$

Grid Generation and Refinement

A finite element grid can be generated by applying the eighth icon (a triangle). This will automatically generate a set of triangles that cover the domain of interest where a higher density of triangles (discretization) points are automatically placed at locations where geometry changes occur. To refine the mesh, the ninth icon (a triangle within a triangle) can be executed. This refinement can be applied a number of times in order to achieve a desired grid resolution.

Solving the PDE and Plotting Results

The PDE can then be solved by executing the 10th icon which is an equals sign. This will solve the specified PDE with prescribed boundary conditions and domain. In order to specify initial conditions and/or the time domain, the solve button must be pushed and the **parameters** button from the drag down menu executed. For a parabolic PDE, the initial condition is $u(x, y, 0) = 0$, while for a hyperbolic PDE, the initial conditions are $u(x, y, 0) = u_x(x, y, 0) = 0$. The tolerance and accuracy for the time-stepping routine can also be adjusted

from this drag down menu. The solution will automatically appear once the computation is completed. The MATLAB icon, which is the eleventh icon, allows for full control of the plotting routines.

As an example, Laplace's equation is solved on a rectangular domain with elliptical internal boundary constraints. This can potentially model the flow field, in terms of the stream function, in a complicated geometry. The boundaries are all subject to Neumann boundary conditions with the top and bottom and internal boundaries specifying no-flux conditions. The left and right boundary impose a flow of the field from left-to-right. The gradient of the solution field, i.e. the velocity of the flow field, is plotted so that the steady state flow can be constructed over a complicated domain.