PDEs that model diffusion, technically classified as *parabolic* PDEs, can admit traveling wave solutions as we demonstrate in the following analysis. The linear diffusion equation is

\[ \frac{\partial u}{\partial t} = D \frac{\partial^2 u}{\partial x^2} \]  

with the initial condition (IC)

\[ u(x, t = 0) = f(x) \]  

For a traveling wave solution, we consider

\[ U(\xi) = u(k(x - Dt)); \quad \xi = k(x - Dt) \]  

Equation (3.3), when substituted into eq. (3.1) gives

\[ \frac{dU}{d\xi} \frac{\partial \xi}{\partial t} = D \frac{d}{d\xi} \left( \frac{dU}{d\xi} \frac{\partial \xi}{\partial x} \right) \frac{\partial \xi}{\partial x} \]

\[ \frac{dU}{d\xi} (-kD) = D \frac{d^2 U}{d\xi^2} k^2 \]  

Equation (3.4) is a second-order ODE that can be integrated once to give (after cancellation of \( D \))

\[ k \frac{dU}{d\xi} + U = C_1 \]  

If we impose the conditions \( U(\xi) = \frac{dU}{d\xi} = 0, \xi \to \infty \), the integration constant is \( C_1 = 0 \). A second integration gives

\[ U(\xi) = Ce^{-\xi/k} \]

which satisfies the condition \( U(\xi) = 0, \xi \to \infty \) (with \( k = 1 \)). Thus,

\[ u(x, t) = Ce^{-(x-Dt)} \]
For the IC for eq. (3.1) and eq. (3.2), we take

\[ u(x, t = 0) = f(x) = e^{-x} \]  
(3.7)

so \( C = 1 \) in eq. (3.6).

Since eq. (3.1) is second order in \( x \), it requires two boundary conditions (BCs), which we take as

\[ u(x = 0, t) = e^{Dt}; \quad u(x \to \infty) = 0 \]  
(3.8)  
(3.9)

Equation (3.6) (with \( C = 1 \)) is the analytical solution that will be used to evaluate the numerical solution.

Before proceeding to the Matlab routines, we confirm that eq. (3.6) is the analytical solution to eq. (3.1) with IC (3.7) and BCs (3.8) and (3.9). Substitution of eq. (3.6) into eq. (3.1) gives (with \( C = 1 \))

<table>
<thead>
<tr>
<th>Terms in PDE eq. (3.1)</th>
<th>Terms from eq. (3.6)</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \frac{\partial u}{\partial t} )</td>
<td>( D e^{-(x-Dt)} )</td>
</tr>
<tr>
<td>( -D \frac{\partial^2 u}{\partial x^2} )</td>
<td>( -D e^{-(x-Dt)} )</td>
</tr>
</tbody>
</table>

Sum of terms 0 0

By inspection, eq. (3.6) satisfies IC (3.7) and BCs (3.8) and (3.9). Thus, we are assured that eq. (3.6) is a valid test of the numerical solution.

The Matlab routines closely resemble those of Chapter 2. Here, we list a few details pertaining to eqns. (3.1) and (3.6)–(3.9). First, the ODE routine pde_1.m is

```matlab
function ut=pde_1(t,u)

global xl xu x n ncall
global D
uxx=dss044(xl,xu,n,u,ux,nl,nu); 

for i=1:n
    ut(i)=D*uxx(i);
end
```
ut(1)=0;
ut(n)=0;
ut=ut';

% Increment calls to pde_1
ncall=ncall+1;

**LISTING 3.1:** Function pde_1.m for eq. (3.1).

We can note the following points about pde_1.m:

- The function and some global parameters are first defined.

  ```matlab
  function ut=pde_1(t,u)
  % Function pde_1 computes the t derivative vector for the linear
  % diffusion equation
  %
  % global xl xu x n ncall
  % global D
  
  % The second derivative in eq. (3.1), \( u_{xx} \), is then computed using the function
  dss044 with *Dirichlet BCs* (3.8) and (3.9) specified \( n_{l}=1, \ n_{u}=1 \) at grid points \( 1 \) corresponding to \( x = 0 \) and \( n = 51 \) corresponding to \( x = \infty \) (subsequently set in
  function initial_1.m).
  
  Note that \( u_{x}(1) = 0 \) is used only to satisfy the calling requirements of dss044 (in Matlab, all input arguments must have a value); \( u_{x} \) is not actually used in dss044 for Dirichlet
  BCs.

- Equation (3.1) is then programmed

  ```matlab
  for i=1:n
      ut(i)=D*uxx(i);
  end
  ut(1)=0;
ut(n)=0;
ut=ut';

  % Increment calls to pde_1
  ncall=ncall+1;
  
  Since Dirichlet BCs are used, the derivatives in \( t \) are set to zero at the boundaries (so that the ODE integrator does not move the boundary values away from their
  prescribed values). A transpose is included to meet the requirements of the ODE
integrator ode15s. Finally, the counter for the number of calls to pde_1.m is incremented.

The IC of eq. (3.7) is programmed in inital_1.m listed next.

```matlab
function u0=inital_1(t0)
    % Function inital_1 sets the initial condition for the linear advection equation
    % Spatial domain and initial condition
    xl= 0;
    xu=10;
    n=51;
    dx=(xu-xl)/(n-1);
    % IC from analytical solution
    for i=1:n
        x(i)=xl+(i-1)*dx;
        u0(i)=ua_1(x(i),0.0);
    end
end
```

**LISTING 3.2: Function inital_1.m for IC (3.7).**

We can note the following points about inital_1.m:

- The function and some global parameters are first defined.

```matlab
function u0=inital_1(t0)
    % Function inital_1 sets the initial condition for the linear advection equation
    % Spatial domain and initial condition
    xl= 0;
    xu=10;
    n=51;
    dx=(xu-xl)/(n-1);
    % IC from analytical solution
    for i=1:n
        x(i)=xl+(i-1)*dx;
        u0(i)=ua_1(x(i),0.0);
    end
end
```

- The grid in \( x \) is then defined over the interval \( 0 \leq x \leq 10 \) for 51 points.

```matlab
function u0=inital_1(t0)
    % Function inital_1 sets the initial condition for the linear advection equation
    % Spatial domain and initial condition
    xl= 0;
    xu=10;
    n=51;
    dx=(xu-xl)/(n-1);
    % IC from analytical solution
    for i=1:n
        x(i)=xl+(i-1)*dx;
        u0(i)=ua_1(x(i),0.0);
    end
end
```

As the grid in \( x \) is defined in the for loop, function ua_1 (listed next) is called (for \( t = 0 \)) to define IC (3.7).
\( x = 10 \) effectively defines a boundary at \( x = \infty \). This can be inferred from the analytical solution, eq. (3.6), since with \( C = D = 1 \),

\[
 u(x = 10, t) = e^{-(10-Dt)}
\]

which for the present example is small for any value of \( t \) we consider (\( t \leq 4 \)). In other words, we have essentially implemented BC (3.9).

Function \texttt{ua}1.m is a straightforward implementation of the analytical solution, eq. (3.6).

\begin{verbatim}
function uanal=ua_1(x,t)
    global D
    uanal=exp(-(x-D*t));
end
\end{verbatim}

\textbf{LISTING 3.3:} Function \texttt{ua}1.m for analytical solution (3.6).

\textbf{FIGURE 3.1:} Numerical solution to eq. (3.1) (lines) with the analytical solution superimposed (circles) using five-point FD approximations in dss044 [1].
The main program, `pde_lmain`, is essentially the same as `pde_lmain` of Listing 2.1 and therefore it is not listed here. The problem parameter is set in the statements.

```plaintext
global D
D=1;
```

The main program produces the same three figures and tabulated output as in Chapter 2, which are now reviewed. Also, the Jacobian matrix routine `jpattern_num_1.m` is the same as `jpattern_num_1.m` in Chapter 2 and is therefore not reproduced here. Figure 3.1 indicates good agreement between the analytical and numerical solutions. Figure 3.2 is a 3D plot of the numerical solution. The map of the ODE Jacobian matrix, Fig. 3.3, reflects the banded structure of the ODEs produced by `dss044`.

The tabular analytical and numerical solutions given in Table 3.1 also reflect the good agreement between these two solutions. The computational effort reflected in `ncall = 150` is quite modest.

In summary, the solution of eq. (3.1) subject to IC (3.2) or (3.7) and BCs (3.8) and (3.9) is straightforward. This is to be expected, since parabolic problems tend to produce smooth solutions; that is, the propagation of steep fronts as in the case of the linear advection eq. (2.1) is generally not a problem.
However, because we have followed a numerical approach, extensions of the problem are straightforward. For example, if the problem included a second-order chemical reaction so that eq. (3.1) becomes

$$\frac{\partial u}{\partial t} = D \frac{\partial^2 u}{\partial x^2} + u^2$$

(3.11)

the preceding coding of `pde_1` in Listing 3.1 would be simply changed to

```c
ut(i)=D*uxx(i)+u(i)^2;
```

Although the change in the coding is trivial, the comparison of the numerical and analytical solutions becomes much more difficult because the analytical solution is not readily available. In other words, although we generally discuss PDE problems in the subsequent chapters for which analytical solutions are available to evaluate the numerical solutions,
### Table 3.1: Tabular numerical and analytical solutions

<table>
<thead>
<tr>
<th>t</th>
<th>x</th>
<th>u(it, i)</th>
<th>u_anal(it, i)</th>
<th>err(it, i)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.00</td>
<td>0.00</td>
<td>1.000000</td>
<td>1.000000</td>
<td>0.000000</td>
</tr>
<tr>
<td>0.00</td>
<td>1.00</td>
<td>0.367879</td>
<td>0.367879</td>
<td>0.000000</td>
</tr>
<tr>
<td>0.00</td>
<td>2.00</td>
<td>0.135335</td>
<td>0.135335</td>
<td>0.000000</td>
</tr>
<tr>
<td>0.00</td>
<td>3.00</td>
<td>0.049787</td>
<td>0.049787</td>
<td>0.000000</td>
</tr>
<tr>
<td>0.00</td>
<td>4.00</td>
<td>0.018316</td>
<td>0.018316</td>
<td>0.000000</td>
</tr>
<tr>
<td>0.00</td>
<td>5.00</td>
<td>0.006738</td>
<td>0.006738</td>
<td>0.000000</td>
</tr>
<tr>
<td>0.00</td>
<td>6.00</td>
<td>0.002479</td>
<td>0.002479</td>
<td>0.000000</td>
</tr>
<tr>
<td>0.00</td>
<td>7.00</td>
<td>0.000912</td>
<td>0.000912</td>
<td>0.000000</td>
</tr>
<tr>
<td>0.00</td>
<td>8.00</td>
<td>0.000335</td>
<td>0.000335</td>
<td>0.000000</td>
</tr>
<tr>
<td>0.00</td>
<td>9.00</td>
<td>0.000123</td>
<td>0.000123</td>
<td>0.000000</td>
</tr>
<tr>
<td>0.00</td>
<td>10.00</td>
<td>0.000045</td>
<td>0.000045</td>
<td>0.000000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>t</th>
<th>x</th>
<th>u(it, i)</th>
<th>u_anal(it, i)</th>
<th>err(it, i)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.00</td>
<td>0.00</td>
<td>2.718282</td>
<td>2.718282</td>
<td>0.000000</td>
</tr>
<tr>
<td>1.00</td>
<td>1.00</td>
<td>1.000000</td>
<td>1.000000</td>
<td>-0.000000</td>
</tr>
<tr>
<td>1.00</td>
<td>2.00</td>
<td>0.367879</td>
<td>0.367879</td>
<td>-0.000001</td>
</tr>
<tr>
<td>1.00</td>
<td>3.00</td>
<td>0.135335</td>
<td>0.135335</td>
<td>-0.000001</td>
</tr>
<tr>
<td>1.00</td>
<td>4.00</td>
<td>0.049787</td>
<td>0.049787</td>
<td>-0.000000</td>
</tr>
<tr>
<td>1.00</td>
<td>5.00</td>
<td>0.018315</td>
<td>0.018316</td>
<td>-0.000000</td>
</tr>
<tr>
<td>1.00</td>
<td>6.00</td>
<td>0.006738</td>
<td>0.006738</td>
<td>-0.000000</td>
</tr>
<tr>
<td>1.00</td>
<td>7.00</td>
<td>0.002477</td>
<td>0.002479</td>
<td>-0.000002</td>
</tr>
<tr>
<td>1.00</td>
<td>8.00</td>
<td>0.000901</td>
<td>0.000912</td>
<td>-0.000010</td>
</tr>
<tr>
<td>1.00</td>
<td>9.00</td>
<td>0.000295</td>
<td>0.000335</td>
<td>-0.000040</td>
</tr>
<tr>
<td>1.00</td>
<td>10.00</td>
<td>0.000123</td>
<td>0.000123</td>
<td>0.000000</td>
</tr>
</tbody>
</table>

... output for t=2, 3 removed ... 

<table>
<thead>
<tr>
<th>t</th>
<th>x</th>
<th>u(it, i)</th>
<th>u_anal(it, i)</th>
<th>err(it, i)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.00</td>
<td>0.00</td>
<td>54.598150</td>
<td>54.598150</td>
<td>0.000000</td>
</tr>
<tr>
<td>4.00</td>
<td>1.00</td>
<td>20.085485</td>
<td>20.085537</td>
<td>-0.000052</td>
</tr>
<tr>
<td>4.00</td>
<td>2.00</td>
<td>7.388982</td>
<td>7.389056</td>
<td>-0.000074</td>
</tr>
<tr>
<td>4.00</td>
<td>3.00</td>
<td>2.718233</td>
<td>2.718282</td>
<td>-0.000048</td>
</tr>
<tr>
<td>4.00</td>
<td>4.00</td>
<td>0.999971</td>
<td>1.000000</td>
<td>-0.000029</td>
</tr>
<tr>
<td>4.00</td>
<td>5.00</td>
<td>0.367853</td>
<td>0.367879</td>
<td>-0.000026</td>
</tr>
<tr>
<td>4.00</td>
<td>6.00</td>
<td>0.135287</td>
<td>0.135335</td>
<td>-0.000048</td>
</tr>
<tr>
<td>4.00</td>
<td>7.00</td>
<td>0.049664</td>
<td>0.049787</td>
<td>-0.000123</td>
</tr>
<tr>
<td>4.00</td>
<td>8.00</td>
<td>0.017981</td>
<td>0.018316</td>
<td>-0.000334</td>
</tr>
<tr>
<td>4.00</td>
<td>9.00</td>
<td>0.005827</td>
<td>0.006738</td>
<td>-0.000911</td>
</tr>
<tr>
<td>4.00</td>
<td>10.00</td>
<td>0.002479</td>
<td>0.002479</td>
<td>0.000000</td>
</tr>
</tbody>
</table>

ncall=150
the intent is to demonstrate numerical procedures that can readily be extended to cases for which analytical solutions are not available. In fact, the possibility of solving numerically (at least in principle) a PDE problem of essentially any complexity is the principal reason for studying and using numerical methods.

Reference