1 Introduction to partial differential equation integration in space and time

1.1 Introduction

The analysis of biomedical systems using mathematical models expressed as partial differential equations (PDEs) is the central theme of this book. This is done not with mathematical analysis such as theorems and proofs, but rather, through example applications to illustrate computational methods for the numerical solution of the model equations, and the details of implementing these numerical methods in computer codes. Example applications are taken from the recent literature: antibody binding kinetics, acid-mediated tumor growth, retinal O_2 transport, hemodialyzer dynamics, epidermal wound healing, drug distribution from a polymer matrix. Each chapter covers a self-contained example.

The numerical solution of the model equations is through a single, well-established procedure for PDEs, the method of lines (MOL), which has been applied to all of the major classes of PDE (parabolic, hyperbolic, and elliptic). Basically, the spatial derivatives in the PDEs are replaced with algebraic approximations; in the present book, the approximations are finite differences (FDs) although other approximations could easily be accommodated within the MOL format, e.g., finite elements, finite volumes, spectral methods, Galerkin methods such as collocation. The final result is a set of routines that implement the MOL calculations for each particular application; the format of these routines is the same throughout the set of applications.

In each example, we follow a series of steps:

- Statement of the mathematical model as a PDE system;
- Some background concerning the model and the original source (references);
- Discussion of the numerical methods (algorithms) for the MOL solution of the model equations;
- Listing of MATLAB routines that implement the MOL solution with a detailed discussion of the routines, a few lines at a time, to emphasize the connection of the programming to the model equations;
- Discussion of the numerical solution of the model equations, including the origin of any unusual features of the solution and an assessment of the accuracy of the solution;
- Concluding summary and discussion of extensions of the model and the MOL algorithms.

Since a common framework is used for all of the applications, we provide in this chapter an introductory discussion of MOL analysis and associated routines. Then, when additional explanation and clarification of computational details are required in the discussion of the applications, references back to Chapter 1 are included. We start with a discussion of hyperbolic PDEs that typically model convection.

1.2 Hyperbolic PDEs

We start the discussion of the numerical solution of PDEs with what is perhaps the simplest PDE, but which, somewhat ironically, is also one of the most difficult to integrate numerically, the linear advection equation (derived in Appendix 1):

$$\frac{\partial u}{\partial t} + v\frac{\partial u}{\partial z} = 0, \tag{1.1}$$

where

Variable	Definition
u	Dependent variable
z	Spatial (boundary value) independent variable
t	Temporal (initial value) independent variable
v	Constant, typically velocity

Equation (1.1) is a partial differential equation since it has more than one independent variable, that is, t and z. We mean by a solution to eq. (1.1), the dependent variable, u, as a function of the independent variables, t and z, that is, $u(z,t)$. The solution may be either a mathematical function, which is called an analytical, exact or closed form solution, or a numerical solution, in which case the solution $u(z,t)$ is in numerical form (as $u(z,t)$ in tabular numerical format or a graph of $u(z,t)$ plotted against t and z).

As a point of notation, we have adopted u as the designation of the dependent variable which is in accordance with much of the literature pertaining to PDEs. For a system of n simultaneous PDEs, we will use $u_1, u_2, ..., u_n$, that is, a vector of n dependent variables. However, in applications, the dependent variables can be, for example, concentrations of various chemical species, temperature, and velocity. Although different symbols for these chemical and physical entities would seem useful (for clarity), we will stay with u as a consistent representation of the PDE-dependent variable(s).

With regard to an application of eq. (1.1), u could represent a chemical composition or concentration for flow at velocity v through a circular channel or tube, which could represent, for example, an artery. z is the distance measured along the tube and t is time. Thus, in computing a numerical solution to eq. (1.1) ($u(z,t)$ in numerical format), we would obtain the concentration as a function of z and t.

Equation (1.1) is a hyperbolic PDE, one of three geometric classifications: hyperbolic, parabolic, and elliptic. Equation (1.1) has constant coefficients since the coefficient v is a constant.

Equation (1.1) is first order since both the derivative in t, $(\partial u/\partial t)$, and the derivative in z, $(\partial u/\partial z)$, are first order. Equation (1.1) is also first degree or linear since the dependent variable u and its derivatives are to the first power. If a PDE is not first degree, it is nonlinear. For example, the PDE $(\partial u/\partial t) + v\,(\partial u/\partial z)^2 = 0$ is nonlinear since the derivative in z is to the second power; it could also be termed second degree (the terms "order" and "power" are easily confused).

In summary, eq. (1.1) is a linear, first order, constant coefficient, hyperbolic PDE. This type of description with words is important not only to describe or classify the PDE, but also because it often suggests a method of analytical or numerical solution.

Since eq. (1.1) is first order in t, it requires one auxiliary condition in t, which is an initial condition (IC) (in general, the required number of ICs equals the order of the highest order derivative in the initial value independent variable; in this case, one IC for the first order derivative $(\partial u/\partial t)$). t is an initial value variable since it starts from an initial (prescribed) value and then proceeds indefinitely, that is, over the semi-infinite interval $0 \le t \le \infty$ or $-\infty \ge t \ge 0$, or the infinite interval $-\infty \ge t \ge \infty$. Another explanation for "initial" is that t usually represents time in an application.

Also, since eq. (1.1) is first order in z, it requires one auxiliary condition in z, which is a boundary condition (BC). z is a boundary value variable since it starts from a boundary (prescribed) value, z_l, and then proceeds to a second value, z_u usually over the finite interval $z_l \le t \le z_u$ (although either z_l or z_u can be ∞ or $-\infty$). Another explanation for "boundary" is that z usually represents space in an application with physical boundaries (z_l and z_u represent the locations of physical boundaries such as surfaces).

For the IC for eq. (1.1), we take

$$u(t = 0, z) = f(z). \tag{1.2}$$

Note that eq. (1.2) is stated for $t = 0$ and $f(z)$ is a prescribed function of z.

For the BC for eq. (1.1), we take

$$u(t, z = 0) = g(t). \tag{1.3}$$

Note that eq. (1.3) is stated for $z = 0$ and $g(t)$ is a prescribed function of t.

Equations (1.1) to (1.3) constitute a complete, well-posed PDE problem and we now seek a numerical solution for particular choices of $f(z)$ and $g(t)$. Many numerical methods for PDEs can be considered and we focus on one approach, the numerical method of lines (MOL). The basic idea in MOL analysis is to replace the boundary value (spatial) derivatives with algebraic approximations. This effectively removes these derivatives from the PDE and since only the initial value independent variable remains, e.g., t, the PDE has been converted to a system of approximating ordinary differential equations (ODEs) that can be integrated by standard, well-established numerical algorithms for initial value ODEs; this is the essence of MOL analysis. We next consider MOL analysis applied to eqs. (1.1) to (1.3) through a series of computer routines.

1.2.1 Spatial integration

To compute a numerical solution to eqs. (1.1) to (1.3), we have to express these equations in a way that a computer can accept or accommodate. Basically, a computer can only do arithmetic and store the results, but fortunately, it does these tasks with remarkable speed and reliability. A computer, however, cannot accept a PDE such as eq. (1.1) directly. In other words, a computer cannot integrate a PDE in the way a human being would, using rather sophisticated mathematical methods. Rather, we have to state the PDE in a way that requires only the operations a computer can do, namely arithmetic.

Therefore, the first challenge in computing a numerical solution is to state the PDE in an alternative form that can be programmed. In particular, integration in space and time (with respect to t and z in eq. (1.1)) is required. Thus, we now consider some numerical methods for temporal and spatial integration.

In the case of spatial integration, we can replace the derivative $(\partial u/\partial x)$ in eq. (1.1) with an algebraic approximation such as

$$\frac{\partial u_i}{\partial z} \approx \frac{u_i - u_{i-1}}{\Delta z}, \tag{1.4a}$$

where i is an index indicating position along a grid in z; Δz is the spacing between the grid points. $i = 1$ corresponds to $z = z_l = 0$ and $i = n$ corresponds to $z = z_u = z_L$ (note that we have selected the spatial interval $0 \le z \le z_L$, but any other interval could be considered, finite, semi-infinite, or infinite).

Since the approximation (RHS) of eq. (1.4a) is a ratio of a difference in u over a difference in z, it is termed a finite difference approximation. If eq. (1.4a) is substituted in eq. (1.1), we have

$$\frac{du_i(t)}{dt} + v\frac{u_i(t) - u_{i-1}(t)}{\Delta z} = 0,\ i = 1,2,...,n. \tag{1.4b}$$

Note that the use of eq. (1.4a) eliminated the derivative in z in eq. (1.1) and thus only one independent variable, t, remains. Therefore, the derivative in eq. (1.4b) is an ordinary or total derivative in t. Also, eq. (1.4b) is an ODE at grid point i, and i is an index over the grid in z that ranges over the values $1 \le i \le n$. In other words, eq. (1.4b) is a system of n ODEs that approximates the PDE, eq. (1.1).

Equations (1.4b) require n ICs which follow from eq. (1.2) as

$$u_i(t = 0) = f(z(i)),\ i = 1,2,...,n. \tag{1.4c}$$

Also, BC (1.3) becomes

$$u_1(t) = g(t). \tag{1.4d}$$

Thus, we do not require the ODE of eqs. (1.4b) at $i = 1$ corresponding to $z = 0$ (because of BC (1.3)) and we therefore have to integrate only $n - 1$ ODEs. An alternative is to use at $i = 1$ the ODE

$$\frac{du_1(t)}{dt} = 0, \tag{1.4e}$$

so that the integration in t does not move $u_1(t)$ away from its value prescribed by eq. (1.4d). Either approach, $n-1$ ODEs without eq. (1.4e) or n ODEs with eq. (1.4e), can be used in the numerical integration with respect to t.

At this point, we will assume that we have an algorithm programmed for the numerical integration of eqs. (1.4b) subject to the ICs of eqs. (1.4c) and BC (1.4d), and postpone temporarily a discussion of temporal integration (with respect to t). Equations (1.4b), (1.4c), and (1.4d) therefore constitute the MOL formulation of eqs. (1.1) to (1.3). After the MOL programming is discussed and some numerical results are reviewed, we will return to the matter of the temporal integration of eqs. (1.4b).

Finally, for the case $v<0$ (flow right to left), eqs. (1.4b), (1.4d), and (1.4e) become,

$$\frac{du_i(t)}{dt}+v\frac{u_{i+1}(t)-u_i(t)}{\Delta z}=0,\ i=1,2,...,n, \tag{1.4f}$$

$$u_n(t)=g(t), \tag{1.4g}$$

$$\frac{du_n(t)}{dt}=0. \tag{1.4h}$$

This form of the MOL approximation of eq. (1.1) is discussed subsequently.

1.2.2 A basic MOL format

We start the discussion of MOL programming with a basic format for the programming of eqs. (1.4b), (1.4c), (1.4d), and (1.4e) (or eqs. (1.4f), (1.4g), and (1.4h)). A main program follows.

```
  clc
  clear all
%
%  Linear advection equation
%
%  The linear advection equation
%
%    ut + v*uz = 0                                          (1)
%
%  is integrated by the method of lines (MOL) subject to
%  the IC
%
%    u(z,t=0) = f(z)                                        (2)
%
%  BC
%
%    u(z=0,t) = g(t)                                        (3)
%
%  We consider in particular f(z) = 0, g(t) = 1 corresponding
%  to the Heaviside unit step function, h(t); the solution to
%  eqs. (1) to (3) is
%
%    u(z,t)=h(t - z/v)                                      (4)
```

```
%
% which is used to evaluate the numerical (MOL) solution.
%
%  The numerical algorithms are:
%
%    z (spatial, boundary value) integration: Two point upwind
%      (2pu)
%
%    t (temporal, initial value) integration: Explicit Euler
%
  global z dz zL v n ncase ncall
%
% Grid (in z)
  zL=1; n=51; dz=0.02;
  z=[0:dz:zL];
%
% Level of output
%
%   Detailed output - ip = 1
%
%   Brief (IC) output - ip = 2
%
  ip=2;
%
% Step through cases
%
%   ncase = 1: v > 0
%
%   ncase = 2: v < 0
%
  for ncase=1:2
    if ncase==1 v= 1; end
    if ncase==2 v=-1; end
%
% Parameters for Euler integration
  nsteps=20;
  h=0.001;
%
% Initial condition
  for i=1:n
    u(i)=0;
  end
  t=0;
%
% Write ncase, h, v
  fprintf('\n\n ncase = %5d    h = %10.3e    v = %4.2f\n\n',ncase,h,v);
%
% Write heading
  if(ncase==1)
    fprintf('   t     zL    t-zL/|v|   u(zL,t)   ua(zL,t)   diff\n');
  end
```

```
%
% Write heading
  if(ncase==2)
    fprintf('   t     zL    t-zL/|v|    u(0,t)    ua(0,t)   diff\n');
  end
%
% Display numerical, analytical solutions at t = 0
  if(t <  zL/abs(v)) ua=0;   end
  if(t >  zL/abs(v)) ua=1;   end
  if(t == zL/abs(v)) ua=0.5; end
  if(ncase==1)
    diff=u(n)-ua;
    fprintf('%5.2f%7.2f%10.2f%10.3f%10.3f%10.4f\n',...
            t,zL,t-zL/abs(v),u(n),ua,diff);
  end
  if(ncase==2)
    diff=u(1)-ua;
    fprintf('%5.2f%7.2f%10.2f%10.3f%10.3f%10.4f\n',...
            t,zL,t-zL/abs(v),u(1),ua,diff);
  end
%
% Store solution for plotting
  if(ncase==1)
     uplot(1,1)=u(n);
    uaplot(1,1)=ua;
     tplot(1)=t;
  end
  if(ncase==2)
     uplot(2,1)=u(1);
    uaplot(2,1)=ua;
  end
%
% nout output points
  nout=101;
  ncall=0;
  for iout=2:nout
%
%   Euler integration
    u0=u; t0=t;
    [u,t]=euler(u0,t0,h,nsteps);
%
%   Numerical, analytical solutions
    if(t <  zL/abs(v)) ua=0;   end
    if(t >  zL/abs(v)) ua=1;   end
    if(t == zL/abs(v)) ua=0.5; end
    if(ip==1)
      if(ncase==1)
        diff=u(n)-ua;
        fprintf('%5.2f%7.2f%10.2f%10.3f%10.3f%10.4f\n',...
                t,zL,t-zL/abs(v),u(n),ua,diff);
      end
```

```
      if(ncase==2)
        diff=u(1)-ua;
        fprintf('%5.2f%7.2f%10.2f%10.3f%10.3f%10.4f\n',...
                t,zL,t-zL/abs(v),u(1),ua,diff);
      end
    end
%
%   Store solution for plotting
    if(ncase==1)
       uplot(1,iout)=u(n);
      uaplot(1,iout)=ua;
       tplot(iout)=t;
    end
    if(ncase==2)
       uplot(2,iout)=u(1);
      uaplot(2,iout)=ua;
       tplot(iout)=t;
    end
%
% Next output
  end
%
% Plots for ncase = 1, 2
  if(ncase==1)
    figure(1);
    plot(tplot,uplot(1,:),'-o');
    axis([0 2 0 1]);
    ylabel('u(zL,t),ua(zL,t)');xlabel('t');
    title('ncase = 1; num - o; anal - line');
    hold on
    plot(tplot,uaplot(1,:),'-');
  end
  if(ncase==2)
    figure(2);
    plot(tplot,uplot(2,:),'-o');
    axis([0 2 0 1]);
    ylabel('u(0,t),ua(0,t)');xlabel('t');
    title('ncase = 2; num - o; anal - line');
    hold on
    plot(tplot,uaplot(2,:),'-');
  end
%
% Next case
fprintf ('\n ncall = %4d\n', ncall/);
  end
```

Listing 1.1 Main program `pde_1_main` for eqs. (1.1) to (1.3)

We can note the following details about this programming:

- Any previous files are cleared, the PDE problem is outlined as comments, and global variables are defined that can be shared with other routines.

```
  clc
  clear all
%
  Documentation comments are not repeated here to conserve space
%
  global z dz zL v n ncase ncall
```

- A grid of 51 points in z is defined over the interval $0 \leq z \leq 1$ with a uniform spacing of `dz=0.02` ($= \Delta z$ in eqs. (1.4a),(1.4b)).

```
%
% Grid (in z)
  zL=1; n=51; dz=0.02;
  z=[0:dz:zL];
```

  Note that we used the MATLAB utility for the definition of a 1D vector, `[]`, to define the grid `z`. `n` is not used here, but is defined numerically for later use. Also, a line of MATLAB code is usually terminated with a semicolon, `;`, to suppress the result from that line. If the semicolon is not used, the resulting displayed result can be useful when debugging some associated code (but also, excessive output can result, particularly if the numerical content of a vector or array is not suppressed, or if the line is executed repeatedly in a loop). Fifty-one grid points were selected to give adequate resolution of the numerical solution with respect to z. This number is a compromise between too small a value (and thus inadequate resolution in z) and too large a value (and thus excessive calculations and run times).
- An integer index, `ip`, is used to select a level of output later. For `ip=2`, only the IC is displayed numerically, but plots of the solution are produced. A `for` loop is then used to step through two cases corresponding to a positive velocity, `v=1` (with the solution traveling left to right) and a negative velocity, `v=-1` (with the solution traveling right to left). This use of a positive and then a negative velocity tests if the code works as expected for both cases.

```
%
% Level of output
%
%   Detailed output - ip = 1
%
%   Brief (IC) output - ip = 2
%
  ip=2;
%
% Step through cases
%
%   ncase = 1: v > 0
```

```
%
%   ncase = 2: v < 0
%
  for ncase=1:2
    if ncase==1 v= 1; end
    if ncase==2 v=-1; end
```

- When eq. (1.1) is approximated as a system of ODEs based on the MOL approach, these ODEs must then be integrated numerically (with respect to t). There are many initial value ODE integration algorithms to choose from, and here we start with the most basic of all such algorithms, the explicit Euler method. Later, we will consider some other ODE integration algorithms. The explicit Euler method steps along the solution with respect to t using an integration step `h=0.001`. After `nsteps =20` such steps, the solution is displayed. Thus, the numerical solution is displayed at $(20)(0.001) = 0.02$ intervals in t. One hundred intervals are used in the subsequent programming so that t covers the interval $0 \leq t \leq 2$. As might be expected, these integration parameters are problem dependent and are usually selected from a knowledge of the problem and also by some trial and error.

```
%
% Parameters for Euler integration
  nsteps=20;
  h=0.001;
```

- The IC of eq. (1.2) is then defined numerically.

```
%
% Initial condition
  for i=1:n
    u(i)=0;
  end
  t=0;
```

  We can note two points about this code:

  – The function $f(z)$ in eq. (1.2) is taken as $u(z, t = 0) = f(z) = 0$.
  – This zero function has been defined numerically in a `for` loop. This could also be done somewhat more compactly by using the MATLAB `zeros` function, that is, `u=zeros(1,n)` for a row vector of n zeros or `u=zeros(n,1)` for a column vector of n zeros. Either format would work in the code that follows (but this is not necessarily the case, especially when using MATLAB utilities that require a particular vector format). Note also that `t=0`, corresponding to the IC of eq. (1.2).

- A heading is displayed at the start of the numerical solution for the two cases `ncase=1,2`.

```
%
% Write ncase, h, v
  fprintf('\n\n ncase = %5d    h = %10.3e    v = %4.2f\n\n',ncase,h,v);
%
% Write heading
```