9

Maxwell's Equations

This partial differential equation (PDE) application introduces the following mathematical concepts and computational methods:

1. We start with a general PDE system in three dimensions (3D) that, with some simplifying assumptions, is reduced to a 1D linear PDE. Here is some terminology:
   (a) The starting point of this application is a classic PDE system, Maxwell's equations of electromagnetic (EM) field theory.
   (b) After reduction of these equations to 1D, followed by some additional simplifications, we arrive at the *damped wave equation* (DWE).
2. The use of coordinate-free PDEs that can then be specialized to a particular coordinate system; for the following analysis, this is Cartesian coordinates.
3. Spatial convergence of the DWE numerical solution by h- and p-refinement.
4. The effect of the method of lines (MOL) finite-difference (FD) approximations on the bandwidth and sparsity of the ordinary differential equation (ODE) Jacobian matrix.
5. A general method for the construction of PDE test problems is illustrated by a specific example for the DWE.
6. The physical significance of the DWE (which we can say without exaggeration is *profound*).

We start the analysis with the *differential form of Maxwell's equations for EM fields*:

$$\nabla \times \mathbf{H} = \mathbf{J} + \mathbf{J_d} = \mathbf{J} + \frac{\partial \mathbf{D}}{\partial t} \tag{9.1a}$$

$$\nabla \times \mathbf{E} = -\frac{\partial \mathbf{B}}{\partial t} \tag{9.1b}$$

$$\nabla \bullet \mathbf{D} = \rho \tag{9.1c}$$

$$\nabla \bullet \mathbf{B} = 0 \tag{9.1d}$$

$$\frac{\partial \rho}{\partial t} + \nabla \bullet \mathbf{J} = 0 \tag{9.1e}$$

Table 9.1. Variables, parameters, and operators in Eqs. (9.1a)–(9.1e)

$\mathbf{H}$	Magnetic field intensity (amps/m)
$\mathbf{E}$	Electric field intensity (volts/m)
$\mathbf{D}$	Electric flux density (coulombs/m^2)
$\mathbf{B}$	Magnetic flux density (webers/m^2)
$\mathbf{J}$	Electric current density (amps/m^2)
$\mathbf{J_d}$	Displacement current density (amps/m^2)
ρ	Charge density (coulombs/m^3)
t	Time (s)
$\times$	Curl vector operator
$\bullet$	Vector dot product
∇	Del vector operator (1/m)

where boldface indicates a vector (except for the vector dot product $\bullet$, which we use for clarity in place of the less apparent dot $\cdot$). The variables, parameters, and operators in Eqs. (9.1a)–(9.1e), with a set of representative units, are given in Table 9.1.

Equation system (9.1a)–(9.1e) clearly has more dependent variables than equations. We therefore use some *constitutive equations* to provide the required additional relationships between the dependent variables:

$$\mathbf{D} = \epsilon \mathbf{E}, \tag{9.1f}$$

$$\mathbf{B} = \mu \mathbf{H}, \tag{9.1g}$$

$$\mathbf{J} = \sigma \mathbf{E} \tag{9.1h}$$

where

ϵ capacitivity or permittivity (farads/m)
μ inductivity or permeability (henrys/m)
σ conductivity (mohs/m)

Equations (9.1a)–(9.1h) are a complete set of PDEs (number of dependent variables = number of equations). We now proceed to combine these equations and finally obtain a single equation in $\mathbf{E}$.

If Eqs. (9.1f), (9.1g), and (9.1h) are substituted into Eq. (9.1a),

$$(1/\mu)\nabla \times \mathbf{B} = \sigma \mathbf{E} + \epsilon \frac{\partial \mathbf{E}}{\partial t} \tag{9.2}$$

Differentiation of Eq. (9.2) with respect to t (assuming a linear, homogeneous, isotropic medium) gives

$$(1/\mu)\nabla \times \frac{\partial \mathbf{B}}{\partial t} = \sigma \frac{\partial \mathbf{E}}{\partial t} + \epsilon \frac{\partial^2 \mathbf{E}}{\partial t^2} \tag{9.3}$$

where the order of the LHS differentiation with respect to t and ∇ (space) has been interchanged.

Substitution of Eq. (9.1b) into Eq. (9.3) gives

$$-\nabla \times (\nabla \times \mathbf{E}) = \mu\sigma\frac{\partial \mathbf{E}}{\partial t} + \mu\epsilon\frac{\partial^2 \mathbf{E}}{\partial t^2} \tag{9.4}$$

The identity

$$\nabla \times (\nabla \times \mathbf{J}) = \nabla(\nabla \bullet \mathbf{J}) - \nabla^2 \mathbf{J}$$

with $\nabla \bullet J = 0$ (constant charge density in Eq. (9.1e)) and Eq. (9.1h) gives

$$\nabla \times (\nabla \times \mathbf{E}) = -\nabla^2 \mathbf{E}$$

Substitution into Eq. (9.4) finally gives a single equation for **E**

$$\mu\epsilon\frac{\partial^2 \mathbf{E}}{\partial t^2} + \mu\sigma\frac{\partial \mathbf{E}}{\partial t} = \nabla^2 \mathbf{E} \tag{9.5}$$

Equation (9.5) is the *time-dependent Maxwell equation for the electric field*, **E**. It also applies to **H** and **J** in place of **E**. Equation (9.1h) represents *Ohm's law*. For the case of a nonconductor where $\sigma = 0$, Eq. (9.5) reduces to the *wave equation*.

Equation (9.5) is both *hyperbolic* (from $\partial^2 \mathbf{E}/\partial t^2$ and $\nabla^2 \mathbf{E}$) and *parabolic* (from $\partial \mathbf{E}/\partial t$ and $\nabla^2 \mathbf{E}$). As expressed in terms of ∇, it is *coordinate independent*. If it is reduced to 1D in *Cartesian coordinates*, we have

$$\mu\epsilon\frac{\partial^2 \mathbf{E}}{\partial t^2} + \mu\sigma\frac{\partial \mathbf{E}}{\partial t} = \frac{\partial^2 \mathbf{E}}{\partial x^2} \tag{9.6}$$

Equation (9.6), a *linear, constant-coefficient PDE*, is the starting point for the MOL analysis. It is *second order in t and x*. We take as the two required initial conditions (ICs)

$$\mathbf{E}(x, t = 0) = \cos(\pi x), \tag{9.7a}$$

$$\frac{\partial \mathbf{E}(x, t = 0)}{\partial t} = 0 \tag{9.7b}$$

The two required boundary conditions (BCs) we take to be *homogeneous Neumann* are

$$\frac{\partial \mathbf{E}(x = 0, t)}{\partial x} = 0, \tag{9.8a}$$

$$\frac{\partial \mathbf{E}(x = 1, t)}{\partial x} = 0 \tag{9.8b}$$

Equations (9.6)–(9.8) are the complete specification of the PDE problem for the following MOL analysis.

A main program for the MOL solution of Eqs. (9.6)–(9.8) is given in Listing 9.1 (in accordance with the usual naming convention, the dependent variable is *u* rather than the *E* of Eqs. (9.6)–(9.8) in the subsequent programming).

```
%
% Clear previous files
  clear all
  clc
%
%
% Parameters shared with the ODE routine
  global  ncall ndss c1 c2 xl xu n
%
% Select case
%
%   Case 1 - second order FDs
%
%   Case 2 - fourth order FDs
%
%   Case 3 - sixth order FDs
%
  for ncase=1:3
%
% Problem parameters
  eps=1.0;
  mu=1.0;
  sigma=1.0;
  c1=sigma/eps;
  c2=1.0/(mu*eps);
%
% Boundaries, number of grid points
  xl=0.0;
  xu=1.0;
  n=101;
%
% Initial condition
  t0=0.0;
  dx=(xu-xl)/(n-1);
  for i=1:n
    x(i)=xl+(i-1)*dx;
    u0(i)=cos(pi*x(i));
    u0(i+n)=0.0;
  end
%
% Independent variable for ODE integration
  tf=1.0;
  tout=[t0:0.2:tf]';
  nout=6;
  ncall=0;
%
% ODE integration
```

```
  reltol=1.0e-04; abstol=1.0e-04;
  options=odeset('RelTol',reltol,'AbsTol',abstol);
  if    (ncase==1) ndss=42;
  elseif(ncase==2) ndss=44;
  elseif(ncase==3) ndss=46; end
%
% Implicit (sparse stiff) integration
  S=jpattern_num;
  pause
  options=odeset(options,'JPattern',S)
  [t,u]=ode15s(@pde_1,tout,u0,options);
%
% One vector to two vectors
  for it=1:nout
  for i=1:n
    u1(it,i)=u(it,i);
    u2(it,i)=u(it,i+n);
  end
  end
%
% Display selected output
  fprintf('\n  ncase = %2d  ndss = %2d  c1 = %4.2f
         c2 = %4.2f\n\n', ncase,ndss,c1,c2);
  for it=1:nout
    fprintf('     t       x         u(x,t)        ut(x,t)\n');
    for i=1:10:n
      fprintf('%6.2f%8.3f%15.6f%15.6f\n',t(it),x(i), ...
             u1(it,i),u2(it,i));
    end
    fprintf('\n');
  end
  fprintf('  ncall = %4d\n\n',ncall);
%
% Plot numerical solution
  figure(ncase)
  plot(x,u1,'-')
  xlabel('x')
  ylabel('u(x,t)')
  title('Maxwell's equation; t = 0, 0.2,... 1.0')
% print -deps -r300 pde.eps; print -dps -r300 pde.ps;
  print -dpng -r300 pde.png
%
% Next case
  end
```

Listing 9.1. Main program `pde_1_main.m` for the solution of Eqs. (9.6)–(9.8)

We can note the following points about this main program:

1. A *global* area is specified to share parameters with other routines. A `for` loop then steps through three cases for FDs of second, fourth, and sixth order. The parameters of Eq. (9.6) are set numerically and passed to the ODE routine as global parameters `c1` and `c2`. We will later return to the significance of these parameters.

```
%
% Clear previous files
  clear all
  clc
%
%
% Parameters shared with the ODE routine
  global  ncall ndss c1 c2 xl xu n
%
% Select case
%
%   Case 1 - second order FDs
%
%   Case 2 - fourth order FDs
%
%   Case 3 - sixth order FDs
%
  for ncase=1:3
%
% Problem parameters
  eps=1.0;
  mu=1.0;
  sigma=1.0;
  c1=sigma/eps;
  c2=1.0/(mu*eps);
```

2. The boundaries and number of grid points in x are set. Then the grid in x is used to define the ICs (Eqs. 9.7). Note that $u(x, t = 0)$ is stored in array `u0(i)` for $1 \leq i \leq n$ and the derivative $\partial u(x, t = 0)/\partial t$ is also stored in `u0(i)` for $n + 1 \leq i \leq 2n$, so that the 1D array (vector) of dependent variables is initialized as required by `ode15s`.

```
%
% Boundaries, number of grid points
  xl=0.0;
  xu=1.0;
```

```
  n=101;
%
% Initial condition
  t0=0.0;
  dx=(xu-xl)/(n-1);
  for i=1:n
    x(i)=xl+(i-1)*dx;
    u0(i)=cos(pi*x(i));
    u0(i+n)=0.0;
  end
```

3. The t interval is $0 \leq t \leq 1$ with an output interval of 0.2 so that six outputs are displayed (including the initial point $t = 0$). For the three cases `ncase=1,2,3` (from the `for` loop at the beginning), `ndss` is passed as a global parameter to select one of the differentiation routines `dss042, dss044, dss046`, respectively (called by the ODE routine `pde_1` discussed next). The sparse matrix version of `ode15s` integrates the `n=(2)(101)=202` ODEs using the ODE routine `pde_1`.

```
%
% Independent variable for ODE integration
  tf=1.0;
  tout=[t0:0.2:tf]';
  nout=6;
  ncall=0;
%
% ODE integration
  reltol=1.0e-04; abstol=1.0e-04;
  options=odeset('RelTol',reltol,'AbsTol',abstol);
  if    (ncase==1) ndss=42;
  elseif(ncase==2) ndss=44;
  elseif(ncase==3) ndss=46; end
%
% Implicit (sparse stiff) integration
  S=jpattern_num;
  pause
  options=odeset(options,'JPattern',S)
  [t,u]=ode15s(@pde_1,tout,u0,options);
```

4. The array u returned by `ode15s` with the ODE solutions is transformed into two 1D arrays (with $u(x, t)$ and $\partial u(x, t)/\partial t$) to facilitate the handling of the output.

```
%
% One vector to two vectors
  for it=1:nout
  for i=1:n
    u1(it,i)=u(it,i);
    u2(it,i)=u(it,i+n);
  end
  end
```

5. Selected tabular output and plots of the solution are then produced.

```
%
% Display selected output
  fprintf('\n  ncase = %2d  ndss = %2d  c1 = %4.2f
         c2 = %4.2f\n\n', ncase,ndss,c1,c2);
  for it=1:nout
    fprintf('     t       x         u(x,t)        ut(x,t)\n');
    for i=1:10:n
      fprintf('%6.2f%8.3f%15.6f%15.6f\n',t(it),x(i),
             u1(it,i),u2(it,i));
    end
    fprintf('\n');
  end
  fprintf('  ncall = %4d\n\n',ncall);
%
% Plot numerical solution
  figure(ncase)
  plot(x,u1,'-')
  xlabel('x')
  ylabel('u(x,t)')
  title('Maxwell's equation; t = 0, 0.2,... 1.0')
% print -deps -r300 pde.eps; print -dps -r300 pde.ps;
  print -dpng -r300 pde.png
%
% Next case
  end
```

The final `end` statement terminates the `for` loop (for `ncase=1,2,3`).

We next review the output from `pde_1_main` before discussing the ODE routine `pde_1`. The abbreviated tabulated output is given in Table 9.2.

Table 9.2. Selected output from `pde_1_main` for `ncase=1,2,3`

```
options =

             AbsTol: 1.0000e-004
                BDF: []
             Events: []
        InitialStep: []
           Jacobian: []
          JConstant: []
           JPattern: [202x202 double]
               Mass: []
       MassConstant: []
       MassSingular: []
           MaxOrder: []
            MaxStep: []
        NormControl: []
          OutputFcn: []
          OutputSel: []
             Refine: []
             RelTol: 1.0000e-004
              Stats: []
         Vectorized: []
   MStateDependence: []
         MvPattern: []
      InitialSlope: []


 ncase =  1  ndss = 42  c1 = 1.00  c2 = 1.00

    t       x         u(x,t)        ut(x,t)
0.00   0.000       1.000000       0.000000
0.00   0.100       0.951057       0.000000
0.00   0.200       0.809017       0.000000
0.00   0.300       0.587785       0.000000
0.00   0.400       0.309017       0.000000
0.00   0.500       0.000000       0.000000
0.00   0.600      -0.309017       0.000000
0.00   0.700      -0.587785       0.000000
0.00   0.800      -0.809017       0.000000
0.00   0.900      -0.951057       0.000000
0.00   1.000      -1.000000       0.000000

    t       x         u(x,t)        ut(x,t)
0.20   0.000       0.820991      -1.673205
0.20   0.100       0.780810      -1.591313
0.20   0.200       0.664197      -1.353649
```

(*continued*)

Table 9.2 (*continued*)

```
0.20   0.300       0.482567      -0.983482
0.20   0.400       0.253701      -0.517047
0.20   0.500      -0.000000      -0.000000
0.20   0.600      -0.253701       0.517047
0.20   0.700      -0.482567       0.983482
0.20   0.800      -0.664197       1.353649
0.20   0.900      -0.780810       1.591313
0.20   1.000      -0.820991       1.673205
        .                          .
        .                          .
        .                          .

 Output for t = 0.4, 0.6, 0.8 is removed

        .                          .
        .                          .
        .                          .
   t       x         u(x,t)         ut(x,t)
1.00   0.000      -0.602024      -0.077062
1.00   0.100      -0.572559      -0.073288
1.00   0.200      -0.487048      -0.062321
1.00   0.300      -0.353861      -0.045258
1.00   0.400      -0.186035      -0.023821
1.00   0.500       0.000000      -0.000000
1.00   0.600       0.186035       0.023821
1.00   0.700       0.353861       0.045258
1.00   0.800       0.487048       0.062321
1.00   0.900       0.572559       0.073288
1.00   1.000       0.602024       0.077062

ncall =  244

        .                          .
        .                          .
        .                          .

  Options output from ode15s is removed

        .                          .
        .                          .
        .                          .

ncase =  2  ndss = 44  c1 = 1.00  c2 = 1.00

   t       x         u(x,t)         ut(x,t)
0.00   0.000       1.000000       0.000000
```

```
0.00   0.100       0.951057       0.000000
0.00   0.200       0.809017       0.000000
0.00   0.300       0.587785       0.000000
0.00   0.400       0.309017       0.000000
0.00   0.500       0.000000       0.000000
0.00   0.600      -0.309017       0.000000
0.00   0.700      -0.587785       0.000000
0.00   0.800      -0.809017       0.000000
0.00   0.900      -0.951057       0.000000
0.00   1.000      -1.000000       0.000000

   t       x         u(x,t)        ut(x,t)
0.20   0.000       0.820969      -1.673268
0.20   0.100       0.780788      -1.591373
0.20   0.200       0.664178      -1.353702
0.20   0.300       0.482553      -0.983522
0.20   0.400       0.253693      -0.517068
0.20   0.500       0.000000       0.000000
0.20   0.600      -0.253693       0.517068
0.20   0.700      -0.482553       0.983522
0.20   0.800      -0.664178       1.353702
0.20   0.900      -0.780788       1.591373
0.20   1.000      -0.820969       1.673268
        .                          .
        .                          .
        .                          .

 Output for t = 0.4, 0.6, 0.8 is removed

        .                          .
        .                          .
        .                          .
   t       x         u(x,t)        ut(x,t)
1.00   0.000      -0.601934      -0.076826
1.00   0.100      -0.572473      -0.073066
1.00   0.200      -0.486975      -0.062154
1.00   0.300      -0.353808      -0.045157
1.00   0.400      -0.186008      -0.023741
1.00   0.500      -0.000000      -0.000000
1.00   0.600       0.186008       0.023741
1.00   0.700       0.353808       0.045157
1.00   0.800       0.486975       0.062154
1.00   0.900       0.572473       0.073066
1.00   1.000       0.601934       0.076826

ncall =  246
```

(*continued*)

Table 9.2 (*continued*)

```
        .                          .
        .                          .
        .                          .

  Options output from ode15s is removed

        .                          .
        .                          .
        .                          .

ncase =  3  ndss = 46  c1 = 1.00  c2 = 1.00

   t       x         u(x,t)        ut(x,t)
0.00   0.000       1.000000       0.000000
0.00   0.100       0.951057       0.000000
0.00   0.200       0.809017       0.000000
0.00   0.300       0.587785       0.000000
0.00   0.400       0.309017       0.000000
0.00   0.500       0.000000       0.000000
0.00   0.600      -0.309017       0.000000
0.00   0.700      -0.587785       0.000000
0.00   0.800      -0.809017       0.000000
0.00   0.900      -0.951057       0.000000
0.00   1.000      -1.000000       0.000000

   t       x         u(x,t)        ut(x,t)
0.20   0.000       0.820969      -1.673268
0.20   0.100       0.780788      -1.591373
0.20   0.200       0.664178      -1.353702
0.20   0.300       0.482553      -0.983522
0.20   0.400       0.253693      -0.517068
0.20   0.500      -0.000000       0.000000
0.20   0.600      -0.253693       0.517068
0.20   0.700      -0.482553       0.983522
0.20   0.800      -0.664178       1.353702
0.20   0.900      -0.780788       1.591373
0.20   1.000      -0.820969       1.673268
        .                          .
        .                          .
        .                          .

 Output for t = 0.4, 0.6, 0.8 is removed

        .                          .
        .                          .
        .                          .
   t       x         u(x,t)        ut(x,t)
1.00   0.000      -0.601934      -0.076826
```

```
1.00   0.100      -0.572473      -0.073066
1.00   0.200      -0.486975      -0.062154
1.00   0.300      -0.353808      -0.045157
1.00   0.400      -0.186008      -0.023741
1.00   0.500      -0.000000      -0.000000
1.00   0.600       0.186008       0.023741
1.00   0.700       0.353808       0.045157
1.00   0.800       0.486975       0.062154
1.00   0.900       0.572473       0.073066
1.00   1.000       0.601934       0.076826

ncall =  248
```

We can note the following details about this output:

1. The options selected before the ODE integrator, `ode15s`, is called are summarized. Note in particular the 202×202 ODE Jacobian matrix. The structure of this matrix is displayed graphically, as discussed subsequently.

```
options =

              AbsTol: 1.0000e-004
                 BDF: []
              Events: []
         InitialStep: []
            Jacobian: []
           JConstant: []
            JPattern: [202x202 double]
                Mass: []
        MassConstant: []
        MassSingular: []
            MaxOrder: []
             MaxStep: []
         NormControl: []
           OutputFcn: []
           OutputSel: []
              Refine: []
              RelTol: 1.0000e-004
               Stats: []
          Vectorized: []
    MStateDependence: []
          MvPattern: []
       InitialSlope: []
```

2. The ICs, Eqs. (9.7a) and (9.7b), are verified. This is an important check (if the ICs are not correct, the solution will certainly not be correct).

```
ncase =  1  ndss = 42  c1 = 1.00  c2 = 1.00

   t       x         u(x,t)        ut(x,t)
0.00   0.000       1.000000       0.000000
0.00   0.100       0.951057       0.000000
0.00   0.200       0.809017       0.000000
0.00   0.300       0.587785       0.000000
0.00   0.400       0.309017       0.000000
0.00   0.500       0.000000       0.000000
0.00   0.600      -0.309017       0.000000
0.00   0.700      -0.587785       0.000000
0.00   0.800      -0.809017       0.000000
0.00   0.900      -0.951057       0.000000
0.00   1.000      -1.000000       0.000000
```

3. The solution evolves in a way that is not easily visualized from the tabulated output, and we will therefore rely on the subsequent plotted output to better understand the solution. However, we can check how the solution varies with the order of the FD approximations. For example, we have from `dss042` (second-order FDs), `dss044` (fourth-order FDs), and `dss046` (sixth-order FDs) the following output at $t = 0.2$.

```
dss042 (ncase = 1):

   t       x         u(x,t)        ut(x,t)
0.20   0.000       0.820991      -1.673205
0.20   0.100       0.780810      -1.591313
0.20   0.200       0.664197      -1.353649
0.20   0.300       0.482567      -0.983482
0.20   0.400       0.253701      -0.517047
0.20   0.500      -0.000000      -0.000000
0.20   0.600      -0.253701       0.517047
0.20   0.700      -0.482567       0.983482
0.20   0.800      -0.664197       1.353649
0.20   0.900      -0.780810       1.591313
0.20   1.000      -0.820991       1.673205

dss044 (ncase = 2):

   t       x         u(x,t)        ut(x,t)
0.20   0.000       0.820969      -1.673268
0.20   0.100       0.780788      -1.591373
```

```
0.20   0.200       0.664178      -1.353702
0.20   0.300       0.482553      -0.983522
0.20   0.400       0.253693      -0.517068
0.20   0.500       0.000000       0.000000
0.20   0.600      -0.253693       0.517068
0.20   0.700      -0.482553       0.983522
0.20   0.800      -0.664178       1.353702
0.20   0.900      -0.780788       1.591373
0.20   1.000      -0.820969       1.673268
```

```
dss046 (ncase = 3):

   t       x         u(x,t)        ut(x,t)
0.20   0.000       0.820969      -1.673268
0.20   0.100       0.780788      -1.591373
0.20   0.200       0.664178      -1.353702
0.20   0.300       0.482553      -0.983522
0.20   0.400       0.253693      -0.517068
0.20   0.500      -0.000000       0.000000
0.20   0.600      -0.253693       0.517068
0.20   0.700      -0.482553       0.983522
0.20   0.800      -0.664178       1.353702
0.20   0.900      -0.780788       1.591373
0.20   1.000      -0.820969       1.673268
```

We observe that through this *p-refinement* (change in the order of the FDs approximations),

(a) The agreement between the three solutions is better than four figures (and is six figures for `dss044` compared to `dss046`). Thus, through this *p*-refinement, we have some confidence that the solutions are accurate to at least four figures (and we arrived at this conclusion without the benefit of an analytical solution).

(b) The symmetry around $x = 0.5$ is as expected (if this was not observed, this would be an indication of an error in the differentiation routines or the MOL solution). Of course, this symmetry also suggests that the solution need only be computed over the interval $0 \leq x \leq 0.5$, which would result in a twofold reduction in the number of ODEs.

(c) The numerical solution requires a modest computational effort.

```
dss042: ncall =  244

dss044: ncall =  246

dss046: ncall =  248
```

Also, at least for this problem, the use of higher-order FDs did not result in a substantially greater number of calls to `pde_1`. However, within each call, the number of arithmetic operations increases with the order of the FDs (the weighted sums in the FDs have more terms with increasing order).

(d) As an incidental point, if the cos in IC (9.7a) is changed to sin, the corresponding numerical solution becomes substantially more difficult to compute. This is because with this change, IC (9.7a) becomes inconsistent with BCs (9.8a) and (9.8b). In other words, the slope of $u(x, t = 0)$ at $x = 0, 1$ is not zero initially from IC (9.7a) (with sin), but changes discontinuously to zero from BCs (9.8a) and (9.8b) for $t > 0$. The discontinuities at the boundaries cause computational difficulties as reflected in the solution (which has some small spatial oscillation) and a substantial increase in `ncall`. Generally, *numerical methods do not like discontinuous functions* (except for those that are designed specifically for discontinuous functions).

The plotted output from `pde_1_main` follows. First, Figure 9.1a is a map of the Jacobian matrix produced by the sparse version of `ode15s` (the Jacobian matrix is required by the ODE integration algorithm in `ode15s` to provide stability for the numerical integration of stiff ODEs).

Figure 9.1a lists the number of the ODE dependent variable (out of 202 ODEs) along the bottom and the number of the ODE down the side. For example, if dependent variable 25 appears in ODE 26, a marker will appear at the coordinates (`25 (horizontal), 26 (vertical)`); thus, the map indicates which dependent variables appear in the RHS of which ODEs.

We can note the following details about the map in Figure 9.1a:

1. The map is sparse with mostly zeros. In fact, only 505 of the 202^2 =40,804 elements of the Jacobian matrix are nonzero (1.238% are nonzero). This degree of sparsity is not unusual in applications, and clearly demonstrates the utility of a sparse matrix integrator such as `ode15s` that works primarily with only the nonzero elements.
2. The nonzero elements appear in bands. The lower left band reflects the FDs of `dss042`; it is three elements wide because the `dss042` FDs use three values of the 202 dependent variables to compute numerical derivatives (they are three-point FD approximations).

The plotted solution is in Figure 9.2, which Figure 9.2 indicates that

1. The solution starts out with a half cosine wave according to IC (9.7a).
2. As expected, the solution is symmetric with respect to $x = 0.5$, so that, again, the solution only over half the interval $0 \le x \le 1$ is actually required and the use of symmetry could reduce the number of ODEs by one half.
3. The solution approaches $u(x, t = \infty) = 0$. This is expected because of the term $\mu\sigma\partial\mathbf{E}/\partial t$ in Eq. (9.6) that provides damping of the solution.

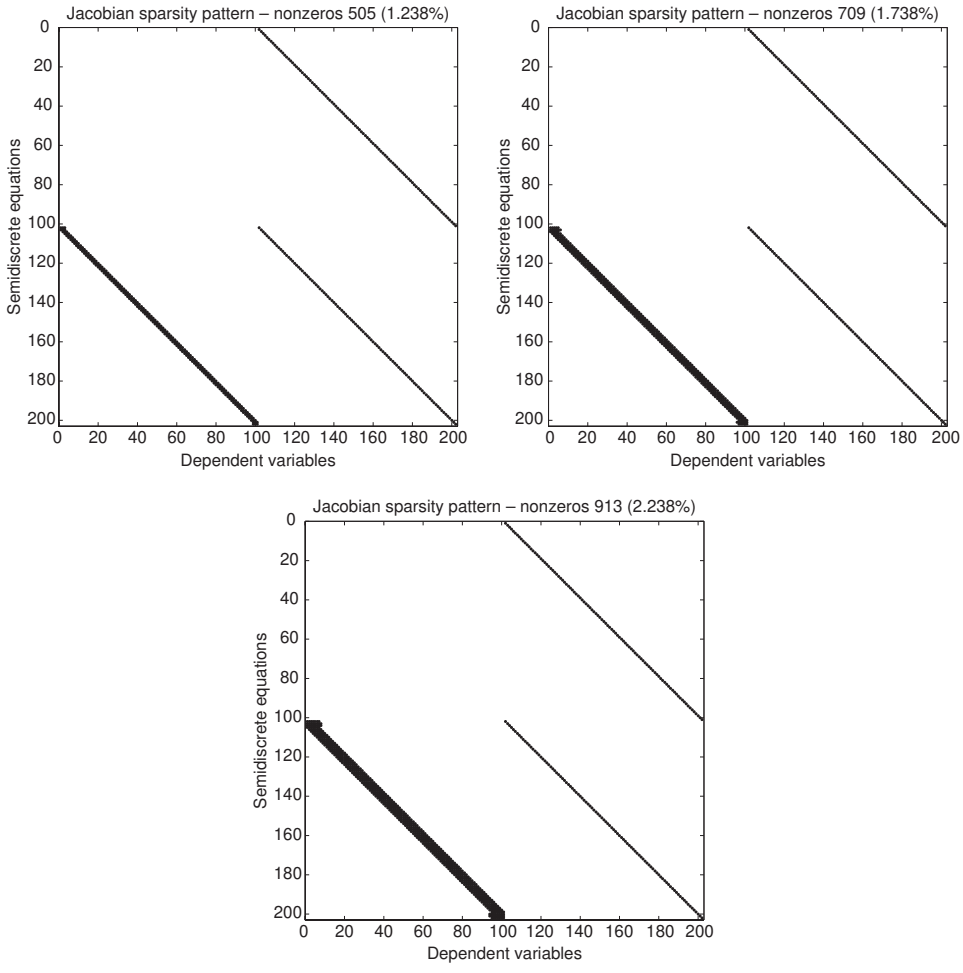

Figure 9.1. Jacobian matrix map for the MOL/ODE approximation of Eqs. (9.6)–(9.8), (a) ncase=1, (b) ncase=2, and (c) ncase=3

To complete the picture of the output from `pde_1_main`, we also include the Jacobian maps and plots for `ncase=2,3` (from `dss044, dss046`) in Figures 9.1b and 9.1c. These figures indicate that the width of the lower left band of the Jacobian map increases with the order of the FD approximation, which is to be expected since the higher order (of the FDs) is achieved by using more values of the dependent variable in calculating the numerical derivatives. Still, the Jacobian matrices are quite sparse.

The plots of the solutions for `ncase=2,3` are indistinguishable from Figure 9.2 for `ncase=1` and are therefore not presented here. However, to elucidate the numerical solution, we also include some 3D plots produced with the following code:

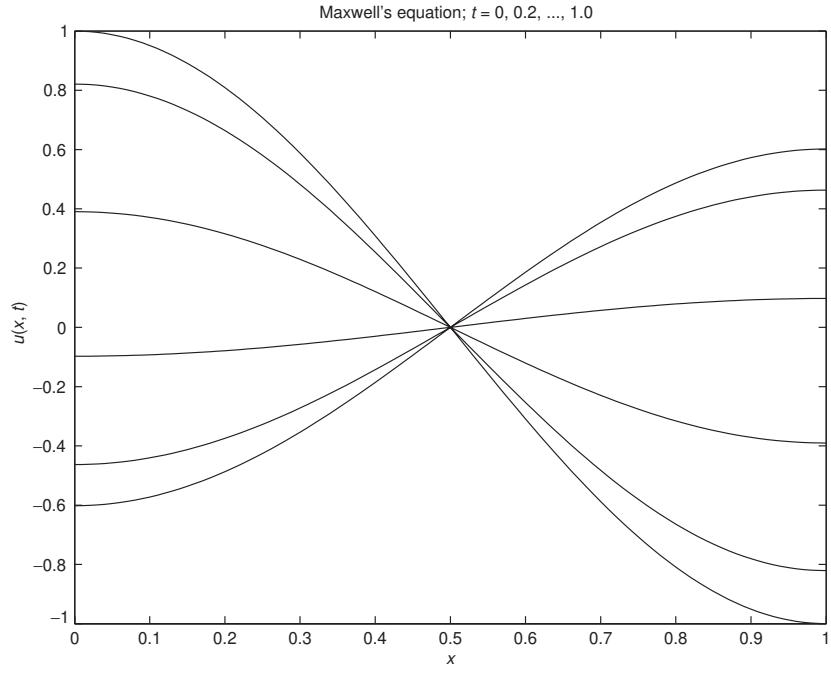

Figure 9.2. Plot of the numerical solution for Eqs. (9.6)–(9.8) from pde_1_main and pde_1 for ncase=1

```
% 3D Plots

figure(3);
surfl(x,t,u1); shading interp;
title('E(x,t)');
xlabel('x');
ylabel('t');
zlabel('E(x,t)');
axis tight
print -dpng -r300 fig3.png;

figure(4);
surfl(x,t,u2); shading interp;
title('\partial/\partialt E(x,t)');
xlabel('x');
ylabel('t');
zlabel('\partial/\partialt E(x,t)');
axis tight
rotate3d on
print -dpng -r300 fig4.png;
```

The resulting 3D plots are shown in Figures 9.3 and 9.4.

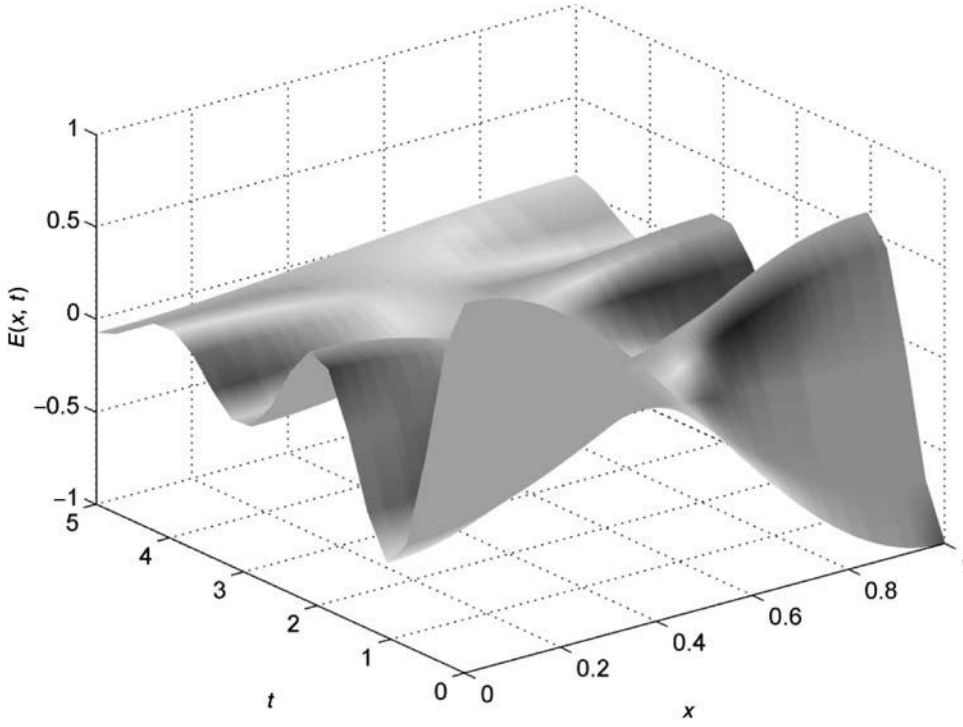

Figure 9.3. MOL/ODE solution of Eqs. (9.6)–(9.8), ncase=1, u1(x,t)

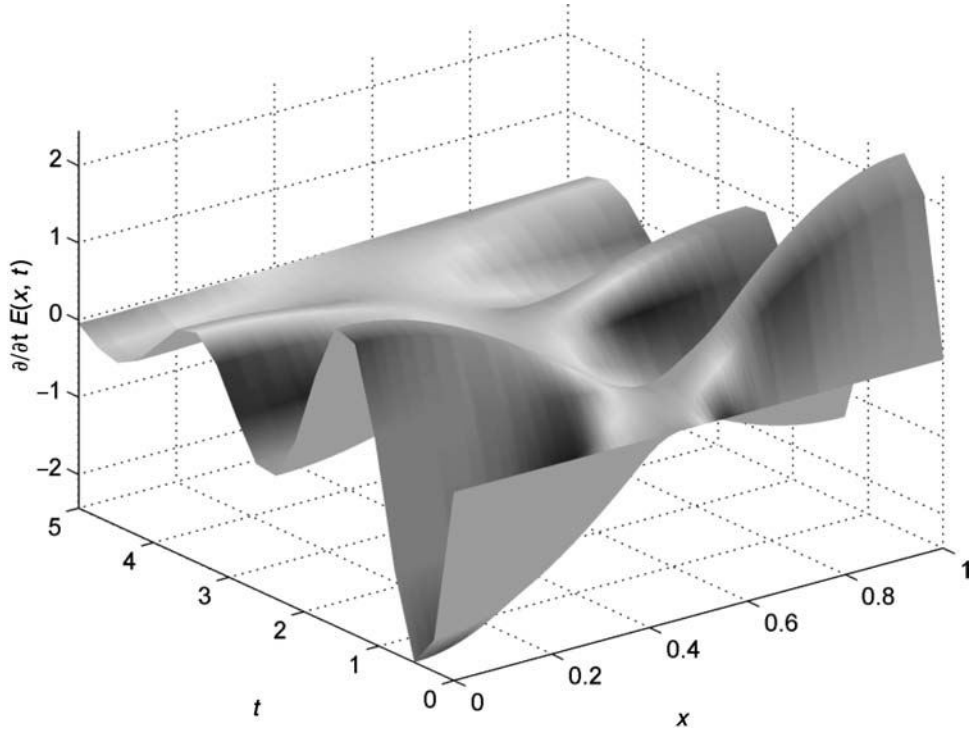

Figure 9.4. MOL/ODE solution of Eqs. (9.6)–(9.8), ncase=1, u2(x,t)

The MOL implementation of Eqs. (9.6) and (9.8) is in the ODE routine pde_1 (see Listing 9.2).

```
  function ut=pde_1(t,u)
%
% Function pde_1 computes the t derivative vector for
% Maxwell's equation
%
  global  ncall ndss c1 c2 xl xu n
%
% One vector to two vectors
  for i=1:n
    u1(i)=u(i);
    u2(i)=u(i+n);
  end
%
% Calculate u1xx with ux(1) = ux(n) = 0 as BCs
  u1x(1)=0.0;
  u1x(n)=0.0;
  nl=2;
  nu=2;
  if    (ndss==42) u1xx=dss042(xl,xu,n,u1,u1x,nl,nu);
  % second order
  elseif(ndss==44) u1xx=dss044(xl,xu,n,u1,u1x,nl,nu);
  % fourth order
  elseif(ndss==46) u1xx=dss046(xl,xu,n,u1,u1x,nl,nu);
  % sixth order
  elseif(ndss==48) u1xx=dss048(xl,xu,n,u1,u1x,nl,nu);
  % eighth order
  elseif(ndss==50) u1xx=dss050(xl,xu,n,u1,u1x,nl,nu);
  % tenth order
  end
%
% PDE
  for i=1:n
    u1t(i)=u2(i);
    u2t(i)=-c1*u1t(i)+c2*u1xx(i);
  end
%
% Two vectors to one vector
  for i=1:n
    ut(i)  =u1t(i);
    ut(i+n)=u2t(i);
  end
```

```
  ut=ut';
%
% Increment calls to pde_1
  ncall=ncall+1;
```

Listing 9.2. pde_1 for the MOL solution of Eqs. (9.6)–(9.8)

We can note the following details about pde_1:

1. After the function is defined and a *global* area is set up to share parameters with the main program of Listing 9.1, the ODE dependent-variable vector u is divided into two vectors for $u(x, t)$ = u1) and $\partial u(x, t)/\partial t$ (= u2), respectively.

```
  function ut=pde_1(t,u)
%
% Function pde_1 computes the t derivative vector for
% Maxwell's equation
%
  global  ncall ndss c1 c2 xl xu n
%
% One vector to two vectors
  for i=1:n
    u1(i)=u(i);
    u2(i)=u(i+n);
  end
```

Equation (9.6), which is second order in t, is expressed as two PDEs first order in t (u = u1, $\partial u/\partial t$ = u2). This is a general procedure required for ODE/PDEs higher than first order in the initial value t since library ODEs integrators such as ode15s can accommodate only first-order ODEs; this procedure for replacing a ODE/PDE nth order in t with n ODE/PDEs first order in t is almost always possible and is easy to implement, so the limitation of library ODEs to first-order equations is not really a significant restriction.

2. The second derivative $\partial^2 u(x, t)/\partial x^2$ (= u1xx) is calculated by a call to one of the differentiation routines dss042, dss044, dss046 (for ncase=1,2,3, respectively). As this is done, BCs (9.8a) and (9.8b) are included.

```
%
% Calculate u1xx with ux(1) = ux(n) = 0 as BCs
  u1x(1)=0.0;
  u1x(n)=0.0;
  nl=2;
  nu=2;
```

```
if    (ndss==42) u1xx=dss042(xl,xu,n,u1,u1x,nl,nu);
% second order
elseif(ndss==44) u1xx=dss044(xl,xu,n,u1,u1x,nl,nu);
% fourth order
elseif(ndss==46) u1xx=dss046(xl,xu,n,u1,u1x,nl,nu);
% sixth order
elseif(ndss==48) u1xx=dss048(xl,xu,n,u1,u1x,nl,nu);
% eighth order
elseif(ndss==50) u1xx=dss050(xl,xu,n,u1,u1x,nl,nu);
% tenth order
end
```

3. The two first-order (in t) PDEs representing Eq. (9.6) are programmed.

```
%
% PDE
  for i=1:n
    u1t(i)=u2(i);
    u2t(i)=-c1*u1t(i)+c2*u1xx(i);
  end
  ut=ut';
%
% Increment calls to pde_1
  ncall=ncall+1;
```

Note that $\partial u/\partial t$ = `u1t` = `u2`, $\partial^2 u/\partial t^2$ = `u2t`.

4. The usual transpose is then included and the number of calls, `ncall`, to `pde_1` is incremented.

Finally, `S=jpattern_num` is listed in Listing 9.3 without a detailed discussion.

```
  function S=jpattern_num
%
% Set global variables
     global n
%
% Sparsity pattern of the Jacobian matrix based on a
% numerical evaluation
%
% Set independent, dependent variables for the calculation
% of the sparsity pattern
  tbase=0;
  for i=1:n
```

```
    ybase(i)=0.5;
    ybase(i+n)=0.5;
  end
  ybase=ybase';
%
% Compute the corresponding derivative vector
  ytbase=pde_1(tbase,ybase);
  fac=[];
  thresh=1e-16;
  vectorized='on';
  [Jac,fac]=numjac(@pde_1,tbase,ybase,ytbase,thresh,fac, ...
                   vectorized);
%
% Replace nonzero elements by "1" (so as to create a "0-1"
% map of the Jacobian matrix)
  S=spones(sparse(Jac));
%
% Plot the map
  figure
  spy(S);
  xlabel('dependent variables');
  ylabel('semi-discrete equations');
%
% Compute the percentage of non-zero elements
  [njac,mjac]=size(S);
  ntotjac=njac*mjac;
  non_zero=nnz(S);
  non_zero_percent=non_zero/ntotjac*100;
  stat=sprintf('Jacobian sparsity pattern - nonzeros
              %d (%.3f%%)', non_zero,non_zero_percent);
  title(stat);
```

Listing 9.3. `S=jpattern_num` Called by the Sparse Option of `ode15s`

We note a few points about `S=jpattern_num`:

1. It is set up for two PDEs (Eq. (9.6) expressed as two first-order PDEs, each approximated over n grid points), but can be readily extended to other numbers of PDEs.

```
    for i=1:n
      ybase(i)=0.5;
      ybase(i+n)=0.5;
    end
```

2. The Jacobian matrix is mapped using Matlab routine `numjac` that calls the ODE routine `pde_1` of Listing 9.2.

```
[Jac,fac]=numjac(@pde_1,tbase,ybase,ytbase,thresh, ...
                 fac,vectorized);
```

3. The remainder of `S=jpattern_num` plots the Jacobian map and computes and displays the number of nonzero elements (as in Figures 9.1a–9.1c).

In summary, our intention in presenting this chapter is to indicate how:

1. A particular PDE, in this case Maxwell's equation for the time-dependent electric field, $E(x, t)$, can be obtained as a special case of a more general PDE system, in this case the Maxwell's field equations.
2. The use of the differentiation routines facilitates changing the order of the FD approximations; all that is required is to change the number of the routine since the arguments remain the same. Since the order of numerical approximations is often given the symbol p, this procedure of changing the order of the FDs is termed *p-refinement*.
3. Additionally, the number of grid points can be changed (in `pde_1_main` of Listing 9.1). Since the grid spacing in numerical approximations is often given the symbol h, changing the number of grid points is termed *h-refinement*.
4. The effect of h- and p-refinement on the numerical solution can be observed through repeated solutions. If the solution remains essentially unchanged, for example, as in Table 9.2, then reasonable *spatial convergence* can be inferred. Note that this procedure does not require knowledge of an analytical solution (but, of course, it is not a mathematical proof of convergence).

We conclude with two further ideas:

1. We did not use an analytical solution to evaluate the numerical solution in the preceding discussion; an analytical solution to Eqs. (9.6)–(9.8) would not be difficult to derive. However, we consider just briefly how analytical solutions might generally be derived that can be useful for testing numerical solutions of PDEs. This is done through an example application for Eqs. (9.6)–(9.8).
   (a) We begin by assuming an analytical solution that satisfies as much of the PDE problem as we can include a priori. For example, for Eq. (9.6), we might assume a solution of the form

$$E(x, t) = e^{at} \cos bt \cos \pi x \tag{9.9}$$

   where a and b are constants to be determined. The choice of the form of Eq. (9.9) is dictated by
      (i) e^{at}, $\cos bt$, and $\cos \pi x$, which when substituted in Eq. (9.6) differentiate to e^{at}, $\cos bt$, $\sin bt$, and $\cos \pi x$ that can then be manipulated so as to evaluate a and b in Eq. (9.9) (this is explained later).
      (ii) $\cos \pi x$ satisfies BCs (9.8a) and (9.8b) (and therefore Eq. (9.9) satisfies BCs (9.8a) and (9.8b) because they are homogeneous).
      (iii) $\mathbf{E}(x, t)$ from Eq. (9.9) satisfies IC (9.7a).

(b) Substitution of Eq. (9.9) into Eq. (9.6) gives

Term from Eq. (9.6)	Term from Eq. (9.9)
$\mu\epsilon\dfrac{\partial^2\mathbf{E}}{\partial t^2}$	$\mu\epsilon(-b^2\,e^{at}\,\cos bt - ab\,e^{at}\,\sin bt$ $-ab\,e^{at}\,\cos bt + a^2\,e^{at}\,\cos bt)\cos\pi x$
$\mu\sigma\dfrac{\partial\mathbf{E}}{\partial t}$	$\mu\sigma(-b\,e^{at}\,\sin bt + a\,e^{at}\,\cos bt)\cos\pi x$
$-\dfrac{\partial^2\mathbf{E}}{\partial x^2}$	$-\pi^2(e^{at}\,\cos bt)\cos\pi x$
$\sum = 0$	$\sum = 0(?)$

Collecting terms in $\sin bt$, we have

$$\mu\epsilon(-ab) + \mu\sigma(-b) = 0$$

or (assuming $b \neq 0$)

$$a = -\sigma/\epsilon \tag{9.10a}$$

Collecting terms in $\cos bt$, we have

$$\mu\epsilon(-b^2 - ab + a^2) + \mu\sigma(a) - \pi^2 = 0$$

or rearranging as a quadratic in b,

$$\mu\epsilon b^2 + a\mu\epsilon b - (\mu\epsilon a^2 - a\mu\sigma + \pi^2) = 0 \tag{9.10b}$$

Thus, Eqs. (9.10a) and (9.10b) can be used to solve for a and b in Eq. (9.9).

(c) So far, Eq. (9.9) satisfies Eqs. (9.6), (9.7a), and (9.8). Unfortunately, it does not satisfy Eq. (9.7b), and in fact, from Eq. (9.9)

$$\frac{\partial\mathbf{E}(x, t = 0)}{\partial t} = a\cos\pi x \tag{9.7c}$$

Thus, Eq. (9.9) is a solution to Eqs. (9.6), (9.7a), (9.7c), and (9.8) (and is therefore not a solution to the original problem, Eqs. (9.6), (9.7a), (9.7b), and (9.8). However, the aforementioned process is useful as it does provide an analytical solution to this related problem. Thus, it can be used as an alternative problem to test a candidate numerical algorithm prior to using it to solve the original problem. If the candidate algorithm provides an accurate solution to the related problem, it is likely to provide an accurate solution to the original problem – although this is not guaranteed.

(d) More generally, this approach to an analytical solution involves starting with a proposed analytical solution (e.g., Eq. (9.9)) and then differentiating it to arrive at a PDE (e.g., Eq. (9.6)) and associated set of ICs and BCs (e.g., Eqs. (9.7a), (9.7c), and (9.8)). This approach has been used extensively to generate analytical solutions that can be used to test numerical PDE solutions.

2. To conclude, the second idea begins with the special case $\sigma = 0$, for which Eq. (9.6) reduces to the *wave equation*.

$$\frac{\partial^2 \mathbf{E}}{\partial t^2} = \frac{1}{\mu\epsilon}\frac{\partial^2 \mathbf{E}}{\partial x^2} \tag{9.11}$$

But the coefficient of the derivative $\partial^2\mathbf{E}/\partial x^2$ is the square of the velocity, c, of the wave described by Eq. (9.11), that is,

$$c = \frac{1}{\sqrt{\mu\epsilon}} \tag{9.12}$$

In other words, EM waves travel at the velocity c given by Eq. (9.12) (here we have considered the electric field $\mathbf{E}$, but Eq. (9.6) also applies to the magnetic field $\mathbf{H}$, or in other words, Eq. (9.6) applies to EM fields). If we use values for μ and ϵ for free space, $\mu = 4\pi \times 10^{-7}$ H/m, $\epsilon = 10^{-9}/(36\pi)$ F/m, we have from Eq. (9.12)

$$c = \frac{1}{\sqrt{(4\pi \times 10^{-7})(10^{-9}/(36\pi))}} = 3 \times 10^8$$

which is the *speed of light in free space* in m/s. Thus, we come to the remarkable conclusion that Maxwell's equations predict that *the speed of EM waves is equal to the speed of light*; in fact, *light waves are EM waves*.

To bring these last two ideas together, for $\sigma = 0$, $a = 0$ and $b = \pi c$ from Eqs. (9.10a) and (9.10b), we obtain velocity c from Eq. (9.12). Thus, IC (9.7b) and IC (9.7c) are the same (for $a = 0$) and Eq. (9.9) becomes an analytical solution for the original problem, Eqs. (9.6), (9.7a), (9.7b), and (9.8). In other words, this analytical solution can be used to test the numerical solution from `pde_1_main` (with `sigma=0.0;`), `pde_1` and `S=jpattern_num`. This numerical solution will oscillate indefinitely, since for $\sigma = 0$, there is no damping in Eq. (9.6).