Solving Partial Differential Equations With `MATLAB` and the `pde1dM` Function

Bill Greene

Version 0.2, July 27, 2020

1 Overview

The `pde1dM` function solves systems of partial differential equations (PDE) and, optionally, coupled ordinary differential equations (ODE) of the following form:

$$c(x,t,u,\frac{\partial u}{\partial x})\frac{\partial u}{\partial t} = x^{-m}\frac{\partial}{\partial x}\left(x^m f(x,t,u,\frac{\partial u}{\partial x})\frac{\partial u}{\partial x}\right) + s(x,t,u,\frac{\partial u}{\partial x}) \tag{1}$$

$$F(t,v,\dot{v},\tilde{x},\tilde{u},\partial\tilde{u}/\partial x,\tilde{f},\partial\tilde{u}/\partial t,\partial^2\tilde{u}/\partial x\partial t) = 0 \tag{2}$$

where

x	Independent spatial variable.
t	Independent variable, time.
u	Vector of dependent variables defined at every spatial location.
c	Vector of diagonal entries of the so-called "mass matrix". Note the entries can be functions of x, t, u, and du/dx.
f	Vector of flux entries. Note the entries can be functions of x, t, u, and du/dx.
s	Vector of source entries. Note the entries can be functions of x, t, u, and du/dx.
m	Allows for problems with spatial cylindrical or spherical symmetry. If $m = 0$, no symmetry is assumed, i.e. a basic Cartesian coordinate system. If $m = 1$, cylindrical symmetry is assumed. If $m = 2$, spherical symmetry is assumed.
F	Vector defining a system of ODE in so-called implicit form. That is, a function of the indicated variables that must equal zero for a solution.
v	Vector of dependent ODE variables.
$\tilde{x}$	Vector of spatial locations where the ODE system is coupled to the PDE system. That is, the PDE variables are evaluated at these specific values of x so they can be used in defining the system of ODE.

Equation (1) is defined on the interval $a \leq x \geq b$. The number of PDE in the problem will be denoted as N and the number of ODE (which may be zero) will be denoted as M.

The complete definition of the problem to be solved includes initial conditions (values of the solution variables at $t = 0$) and, for the PDE variables, boundary conditions. The boundary conditions are defined at the ends of the spatial domain– $x = a$ and $x = b$. These boundary conditions take the following form

$$p(x,t,u) + q(x,t)f(x,t,u,du/dx) = 0 \tag{3}$$

where p and q must be defined by the user at both ends of the domain.

2 Calling `pde1dM`

The basic calling sequence for `pde1dM` when there are no ODE is

```
solution = pde1dM(m,pdeFunc,icFunc,bcFunc,meshPts,timePts)
```

`m`	Defines the spatial coordinate system type, as described above. Default is `m=0`.
`pdeFunc`	Handle to a user-written function that describes the system of PDE to be solved. This function is described in detail below.
`icFunc`	Handle to a user-written function that describes the initial conditions for the system of PDE. This function is described in detail below.
`bcFunc`	Handle to a user-written function that describes the boundary conditions for the system of PDE. This function is described in detail below.
`meshPts`	Vector of x locations defining the spatial mesh. The first entry in `meshPts` must equal the beginning of the interval, a, the last point must equal b, and the values of the intermediate points must be monotonically increasing. The accuracy of the solution depends on the density of the points in this mesh. The spacing between points need not be uniform; it is often advantageous to prescribe a higher density of points in places where the solution is changing rapidly.
`timePts`	Vector of time points where it is desired to output the solution. The density of points in this vector has no effect on the accuracy of solution. Often the number of points is determined by the number required to produce a smooth plot of the solution as a function of time.

This function signature is identical to that of the `MATLAB pdepe` function.

An additional argument, `options`, may be included to change various parameters controlling the behavior of `pde1dM` .

```
solution = pde1dM(m,pdeFunc,icFunc,bcFunc,meshPts,timePts,options)
```

This argument is a `MATLAB` structure array. The name, default value, and purpose of the supported options are

Name	Default	Purpose
RelTol	1e-3	Relative tolerance for converged solution
AbsTol	1e-6	Absolute tolerance for converged solution
Vectorized	false	If set to true, `pdeFunc` is called with a vector of x-values and is expected to return values of c, f, and s for all of these x-values. Setting this option to true substantially improves performance.
MaxSteps	10000	Maximum number of time steps allowed

When ODE are included in the problem definition, three additional arguments to the `pde1dM` function are required.

```
[solution,odeSolution] = pde1dM(m,pdeFunc,icFunc,bcFunc,meshPts,timePts,...
                         odeFunc, odeIcFunc,xOde)
```

`odeFunc`	Handle to a user-written function that describes the system of ODE to be solved. This function is described in detail below.
`odeIcFunc`	Handle to a user-written function that describes the initial conditions for the system of ODE. This function is described in detail below.
`xOde`	Vector of x locations where the systems of PDE and ODE interact. At these x locations, the values of various PDE variables are made available for use in defining the system of ODE. Any reasonable number of x locations may be defined and they need not coincide with the mesh points in the PDE definition.

The same `options` argument, described above, may also be included when there are ODE.

```
[solution,odeSolution] = pde1dM(m,pdeFunc,icFunc,bcFunc,meshPts,timePts,...
                        odeFunc, odeIcFunc,xOde,options)
```

3 User-Defined Functions

The user-written `MATLAB` functions, mentioned above, that define the systems of PDE and ODE are discussed in more detail in this section. In the description of these functions, a function name is chosen which is somewhat descriptive of the function's purpose; however the user is free to choose any allowable `MATLAB` function name for these functions.

3.1 User-Defined Functions When The Problem Has Only PDE

3.1.1 PDE Definition Function

```
[c, f, s] = pdeFunc(x, t, u, DuDx)
```

3.1.2 PDE Boundary Condition Function

```
[pLeft, qLeft, pRight, qRight] = bcFunc(xLeft, uLeft, xRight, uRight, t)
```

The user must return the q and p vectors defined in equation (3) at the `Left` and `Right` ends of the domain. The input variables, `xLeft, uLeft, xRight, uRight, t` may be used in defining `pLeft, qLeft, pRight, qRight`. (Note that `xLeft` will equal a and `xRight` will equal b). Entries in the `qLeft` and `qRight` vectors that are zero at $t = 0$, are assumed to be zero at all future times. Entries in the `qLeft` and `qRight` vectors that are non-zero at $t = 0$, are assumed to be non-zero at all future times.

3.1.3 PDE Initial Condition Function

```
u0=pdeIcFunc(x)
```

The complete specification of a PDE system requires that initial conditions (solution at $t = 0$) be defined by the user. The function must return the initial condition, `u0`, at spatial location `x`. `u0` is a vector which has length N. Formally, the boundary conditions returned from `bcFunc` and the initial conditions returned from `pdeIcFunc` should agree but this is not strictly required by `pde1dM` .

3.2 User-Defined Functions When The Problem Has Both PDE and ODE

The following two functions are required only if the problem definition includes ODE.

3.2.1 ODE Definition Function

```
F=odeFunc(t,v,vdot,x,u,DuDx,f, dudt, du2dxdt)
```

The system of ODE is defined by equation (2). The input variables to `odeFunc` are

`t`	time
`v`	vector of ODE values
`vDot`	derivative of ODE variables with respect to time
`x`	Vector of spatial locations where the ODE couple with the PDE variables. This is referred to as $\tilde{x}$ in equation (2).
`u`	values of the PDE variables at the x locations
`DuDx`	derivatives of the PDE variables with respect to x, evaluated at the x locations
`f`	values of the flux defined by the PDE definition evaluated at the x locations
`dudt`	derivatives of the PDE variables with respect to time, evaluated at the x locations
`du2dxdt`	second derivatives of the PDE variables with respect to x and time, evaluated at the x locations

The vector `F` must be returned.

3.2.2 ODE Initial Condition Function

```
v0=odeIcFunc()
```

A vector (length M) of initial values of the ODE variables, `v0`, must be returned.

3.2.3 PDE Definition Function

```
[c, f, s] = pdeFunc(x, t, u, DuDx, v, vDot)
```

The PDE function, `pdeFunc`, is identical to the definition above except that, when ODE are included, two additional input variables are provided. These are the values of the ODE variables, `v` and their derivatives with respect to time, `vDot`.

3.2.4 PDE Boundary Condition Function

```
[pLeft, qLeft, pRight, qRight] = bcFunc(xLeft, uLeft, xRight, uRight,
                                        t,v,vDot)
```

The boundary condition function, `bcFunc`, is identical to the definition above except that, when ODE are included, two additional input variables are provided. These are the values of the ODE variables, `v` and their derivatives with respect to time, `vDot`.

4 Examples

4.1 Simple Coupling of PDE and ODE Equations

The Numerical Algorithms Group (NAG) provides a function, `do3phf`, for solving coupled systems of PDE and ODE equations (reference [2]). Their simple example coupling a single PDE with a single PDE will be shown here.

The single PDE, using the notation of reference [2], is

(4)

(5)

The `MATLAB` code for this example is shown below

```
function nagD03phfExample
n=10;
L=1;
x = linspace(0,L,n);
t0=1e-4;
t=[t0 .1*2.^[1:5]];
t=linspace(t0, .6, 10);
m = 0;
xOde = L;
icF = @(x) icFunc(x,t0);
odeIcF = @() odeIcFunc(t0);
opts.vectorized='on';
[u,uode] = pde1dM(m, @pdeFunc,icF,@bcFunc,x,t,@odeFunc, odeIcF,xOde,opts);
va=vAnal(t);
ua=uAnal(t,x);

figure; plot(t, uode(:), t, va, 'o');
xlabel('Time'); ylabel('v');
legend('Numerical', 'Analytical');
title 'ODE␣Solution␣as␣a␣Function␣of␣Time';
figure; plot(x, u(end,:), x, uAnal(t(end), x), 'o');
xlabel('x'); ylabel('u');
legend('Numerical', 'Analytical');
title 'PDE␣Solution␣At␣the␣Final␣Time';
fprintf('Maximum␣error␣in␣ODE␣valriable=%10.2e\n', max(abs(uode(:)-va(:))));
fprintf('Maximum␣error␣in␣PDE␣valriable=%10.2e\n', max(abs(u(:)-ua(:))));

end

function [c,f,s] = pdeFunc(x,t,u,DuDx,v,vdot)
nx = length(x);
c = repmat(v(1)^2,1,nx);
f = DuDx;
s = x.*DuDx*v*vdot;
end

function u0 = icFunc(x, t0)
u0 = uAnal(t0, x);
%prtVec('u0', u0);
end

function [pl,ql,pr,qr] = bcFunc(xl,ul,xr,ur,t,v,vdot)
pl = v(1)*exp(t);
ql = 1;
pr = v(1)*vdot(1);
qr = 1;
end

function f=odeFunc(t,v,vdot,x,u,DuDx)
%fprintf('x=%f\n', x);
f=v*u + DuDx + 1 + t - vdot(1);
end
```

```
function v0=odeIcFunc(t0)
v0=vAnal(t0);
end

function v=vAnal(t)
  % analytical soluton for ode variable
  v=t;
end

function u=uAnal(t,x)
  % analytical soluton for pde variable
  u=exp(t'*(1-x))-1;
end
```

4.2 Nonlinear Heat Equation with Periodic Boundary Conditions

For the solution to be periodic, the following two conditions must hold

$$u(a) = u(b) \tag{6}$$

and

$$\frac{\partial u}{\partial x}(a) = \frac{\partial u}{\partial x}(b) \tag{7}$$

It is not possible to apply such boundary conditions using the standard PDE boundary condition mechanism. However reference [1] shows how a periodic boundary condition can be prescribed by adding an ODE to the system. This example from reference [1] is presented here. This example has the added benefit that a simple analytical solution is available to compare with the numerical results.

We will enforce equation (7) by defining an ODE variable, v, such that

$$\frac{\partial u}{\partial x}(a) = v \tag{8}$$

$$\frac{\partial u}{\partial x}(b) = v \tag{9}$$

The ODE used to enforce equation (6) is simply

$$u(a) - u(b) = 0 \tag{10}$$

Clearly this doesn't **look** like an ODE because neither v nor $\partial v/\partial t$ is present. However it does satisfy the form defined in equation (2);

The MATLAB code for this example is shown below

```
function schryer_ex4
n=15;
x = linspace(-pi,pi,n);
t=linspace(0, 3*pi/2.5, 10);
m = 0;
% We need the PDE solution at the right and left ends
% to define the ODE (constraint equation).
xOde = [pi -pi]';
% Set vectorized mode to improve performance
```

```
opts.vectorized='on';
[u,vOde] = pde1dM(m,@pdeFunc,@icFunc,@bcFunc,x,t,@odeFunc,@odeIcFunc,xOde,opts);

% Compute analytical solution for comparison.
ua=uAnal(t,x);

figure; plot(x, u(end,:), x, uAnal(t(end), x), 'o'); grid on;
xlabel('x'); ylabel('u'); title('Solution at Final Time');
legend('Numerical', 'Analytical');

figure; plot(t, u(:,end), t, u(:,1), t, uAnal(t, x(1)), 'o'), grid on;
xlabel('Time'); ylabel('u'); title('Solution at End Points');
legend('Right End', 'Left End', 'Analytical');

figure; plot(t, vOde); grid on;
xlabel('Time'); ylabel('v'); title('ODE Variable as a Function of Time');

end

function [c,f,s] = pdeFunc(x,t,u,DuDx,v,vdot)
nx=length(x);
c = ones(1,nx);
f = DuDx;
cx=cos(x);
st=sin(t);
gxt=cx*cos(t)+cx*st+cx.^3*st^3;
s = -u.^3 + gxt;
end

function u0 = icFunc(x)
u0 = uAnal(0, x);
end

function [pl,ql,pr,qr] = bcFunc(xl,ul,xr,ur,t,v,vdot)
% du/dx at right end must equal
% du/dx at the left end. Use the ODE
% variable to enforce this.
pl = v(1);
ql = 1;
pr = v(1);
qr = 1;
end

function f=odeFunc(t,v,vdot,x,u,DuDx)
% The solution at the right end must equal
% the solution at the left end.
f=u(1)-u(2);
end

function v0=odeIcFunc()
v0=0;
end
```

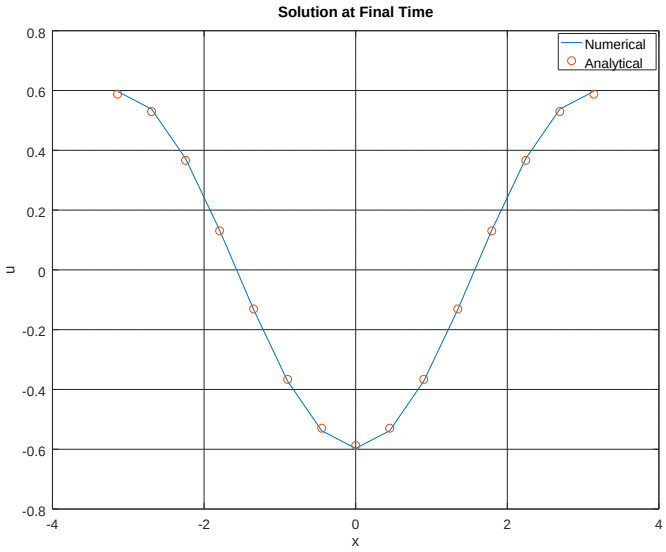

Figure 1: Solution at the final time as a function of x.

```
function u=uAnal(t,x)
u=sin(t)'*cos(x);
end
```

From Figure 1 we see that the solution at the final time compares well with the analytical solution. We also see that the solutions at the right and left ends agree.

Figure 2 shows that the solutions at the right and left ends are equal at all times and that they agree with the analytical solution.

Figure 3 plots the solution of the ODE equation, v as a function of time. A simple differentiation of the analytical solution with respect to x shows that $\partial u/\partial x$ at the ends equals zero for all times. The figure shows that the numerical solution is also very close to zero for all times.

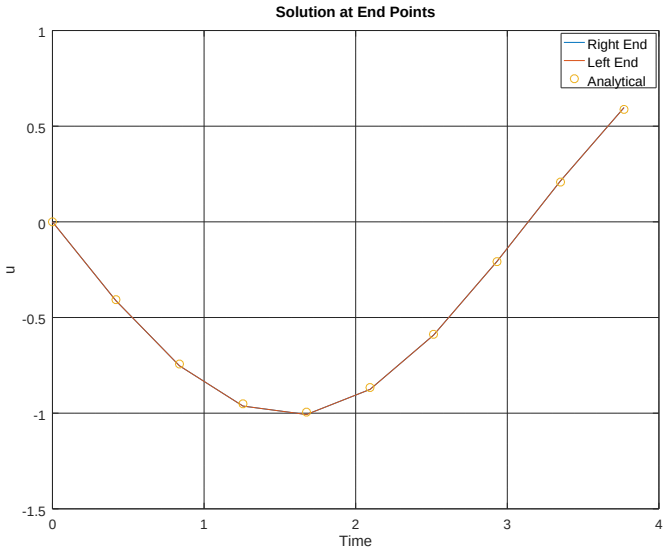

Figure 2: Solution at the two ends as a function of time.

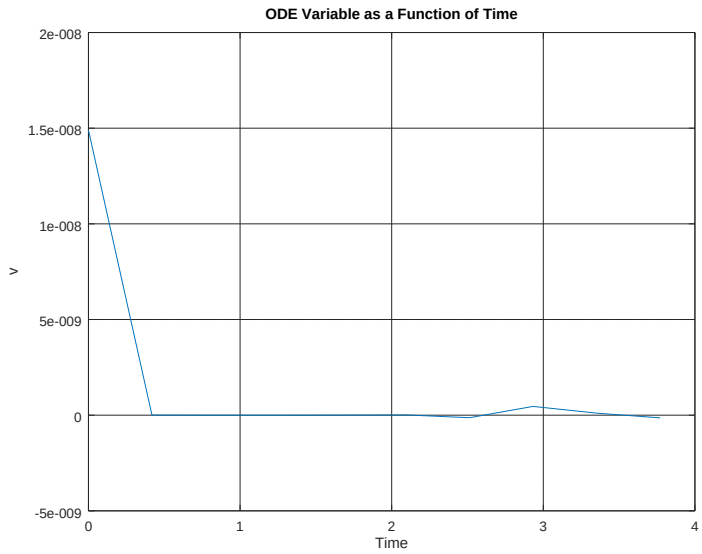

Figure 3: Solution of the ODE equation as a function of time.

A Appendix: Comparison Between pde1dM and MATLAB pdepe

The types of PDE solved by pde1dM and MATLAB pdepe are similar. The input to pde1dM is also similar to that of pdepe and, in many cases, the pdepe input for a problem can be used without changes in pde1dM . However, there are some significant differences between the two PDE solvers and this section describes those. It is primarily written for users who have a reasonable amount of experience with pdepe.

1. pdepe has interpolation functions specially designed to provide accurate solutions near $x = 0$ in problems with cylindrical ($m = 1$) and spherical ($m = 2$) coordinate systems. pde1dM can solve the PDE for such problems but the solution will generally be less accurate near $x = 0$.

2. pdepe allows the user to specify Events to be handled during the solution. pde1dM does not.

3. pdepe relies on the MATLAB ode15s ODE solver to solve initial value problem. pde1dM uses the ode15i solver.

References

[1] N. L. Schryer, POST- A Package for Solving Partial Differential Equations in One Space Variable, AT&T Bell Laboratories, August 30, 1984.

[2] NAG Fortran Library Routine Document D03PHF/D03PHA.