Solving Partial Differential Equations With the `pde1dm` Function and MATLAB/Octave

Bill Greene

Version 1.4, August 2, 2023

Contents

1 Overview **2**

2 Calling `pde1dm` **2**

3 User-Defined Functions **4**
3.1 User-Defined Functions When The Problem Has Only PDE 4
3.1.1 PDE Definition Function 4
3.1.2 PDE Boundary Condition Function 4
3.1.3 PDE Initial Condition Function 4
3.2 User-Defined Functions When The Problem Has Both PDE and ODE 5
3.2.1 ODE Definition Function 5
3.2.2 ODE Initial Condition Function 5
3.2.3 PDE Definition Function 5
3.2.4 PDE Boundary Condition Function 5

4 Examples **6**
4.1 Heat Conduction in a Rod 6
4.1.1 Code Required by `pde1dm` 6
4.1.2 Improving Performance with Vectorized Mode 8
4.2 Simple Coupling of PDE and ODE Equations 9
4.3 Nonlinear Heat Equation with Periodic Boundary Conditions 12
4.4 Dynamics of An Elastic Structural Beam 17
4.4.1 PDE Definition 17
4.4.2 Boundary Conditions 18
4.4.3 Free Vibration of a Simply Supported Beam 18
4.5 Melting of Ice 20
4.5.1 Governing Equations 21
4.5.2 Moving Mesh 21
4.5.3 Implementation and Results 21

A Appendix: Comparison Between `pde1dm` and `MATLAB` `pdepe` **25**

1 Overview

The `pde1dm` function works in either MATLAB or Octave and solves systems of partial differential equations (PDE) and, optionally, coupled ordinary differential equations (ODE) of the following form:

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

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

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. This set of PDE variables is denoted $\tilde{u}$.

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 the initial time) 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.
`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.
`solution`	Values of the PDE variables at each time and mesh point. The size of this output matrix is number of time points $\times$ number of mesh points $\times$ N.

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 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.

These function signatures are identical to the `MATLAB pdepe` function.

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.
`odeSolution`	Matrix of the values of ODE variables at the time points. The size of this output matrix is M $\times$ number of time points.

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 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 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)
```

The user must return the c, f, and s matrices defined in equation (1). The size of these matrices is N $\times$ number of entries in the x vector.

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 the initial time, are assumed to be zero at all future times. Entries in the `qLeft` and `qRight` vectors that are non-zero at the initial time, 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 the initial time) 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 Heat Conduction in a Rod

The first example we will consider is the heat conduction in a rod where the temperature at the left end is prescribed as 100 $^\circ C$ and all other surfaces of the rod are insulated. The initial temperature of all other points in the rod is zero. The material is copper with density, $\rho = 8940\ kg/m^3$; specific heat, $c_p = 390\ J/(kg\ ^\circ C)$ and thermal conductivity, $k = 385\ W/(m\ ^\circ C)$.

The PDE describing this behavior is

$$\rho c_p \frac{\partial T}{\partial t} = \frac{\partial}{\partial x}\left(k \frac{\partial T}{\partial x}\right) \tag{4}$$

With appropriate changes in coefficients, this equation describes a wide variety of physical behavior so is particularly appropriate as a first example.

4.1.1 Code Required by `pde1dm`

The complete code for this example is shown below. This section highlights some key pieces of this code.

```
n=12; % number of nodes in x
L=1; % length of the bar in m
x = linspace(0,L,n);
```

This code snippet defines the x-locations of the nodes. This example uses twelve nodes but since the accuracy of the solution depends on the mesh it is important to verify that the mesh is sufficiently refined.

```
t = linspace(0,2000,30); % number of time points for output of results
```

This code snippet defines the points in time where we would like to save the solution. The accuracy of the solution is not affected by this choice.

The PDE, equation (4), is defined by matching the terms with those in the general equation (1). The variable m is zero because we have a rectangular Cartesian coordinate system. The resulting code is

```
function [c,f,s] = heatpde(x,t,u,DuDx)
rho=8940; % material density
cp=390; % material specific heat
k=385; % material thermal conductivity
c = rho*cp;
f = k*DuDx;
s = 0;
end
```

The actual name for this function is arbitrary. The boundary conditions are defined at each end to match equation (3). At the left end, we are simply prescribing the value of temperature so $q = 0$ and p is defined as the temperature at the left end minus the prescribed value of temperature, $Tl - T0$. A common mistake is to simply set $p = T0$ instead of in the form required by (3), $Tl - T0$! The right end is insulated so the heat flux, $k\partial T/\partial x$, is zero. Note carefully that we have defined the f result in the `heatpde` function as the heat flux. We want `pde1dm` to maintain this quantity as zero at the right end so, referring to equation (3), $p = 0$ and $q = 1$. The resulting code is

```
function [pl,ql,pr,qr] = heatbc(xl,Tl,xr,Tr,t)
T0=100; % temperature at left end, degrees C
```

```
pl = Tl-T0;
ql = 0;
pr = 0;
qr = 1;
end
```

Defining the initial conditions is straightforward. We want the initial temperature of the rod to be zero everywhere except at the left end where it should match the boundary condition

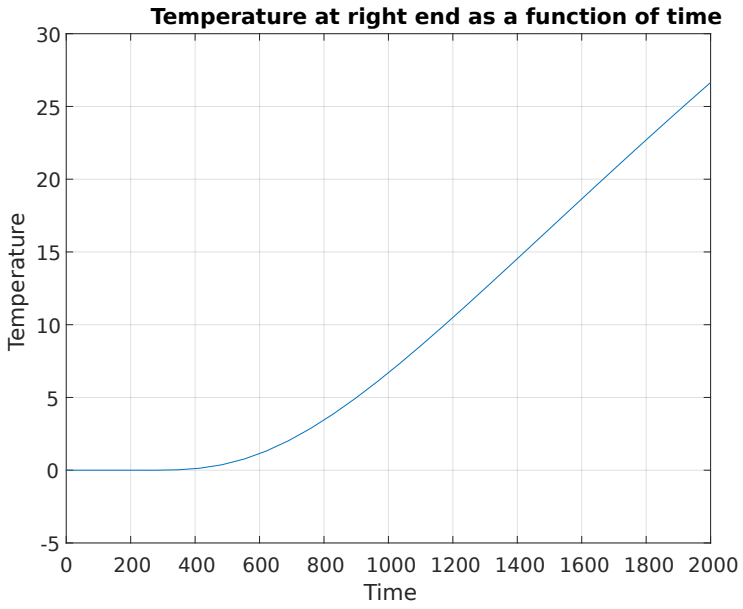
```
function u0 = heatic(x)
T0=100; % temperature at left end, degrees C
if x==0
u0=T0;
else
u0=0;
end
end
```

The full code for this example is

```
function heatConduction
% transient heat conduction in a rod
n=12; % number of nodes in x
L=1; % length of the bar in m
x = linspace(0,L,n);
t = linspace(0,2000,30); % number of time points for output of results
m=0; % rectangular Cartesian coordinate system
u = pde1dm(m, @heatpde,@heatic,@heatbc,x,t);

figure; plot(t, u(:,end)); grid on;
xlabel('Time'); ylabel('Temperature');
title('Temperature␣at␣right␣end␣as␣a␣function␣of␣time');

figure; plot(x, u(end,:)); grid on;
xlabel('x'); ylabel('Temperature');
title('Temperature␣along␣the␣length␣at␣final␣time');

end

function [c,f,s] = heatpde(x,t,u,DuDx)
rho=8940; % material density
cp=390; % material specific heat
k=385; % material thermal conductivity
c = rho*cp;
f = k*DuDx;
s = 0;
end

function [pl,ql,pr,qr] = heatbc(xl,Tl,xr,Tr,t)
T0=100; % temperature at left end, degrees C
pl = Tl-T0;
ql = 0;
pr = 0;
```

```
qr = 1;
end

function u0 = heatic(x)
T0=100; % temperature at left end, degrees C
if x==0
  u0=T0;
else
  u0=0;
end
end
```

When executed, this script produces the following two figures



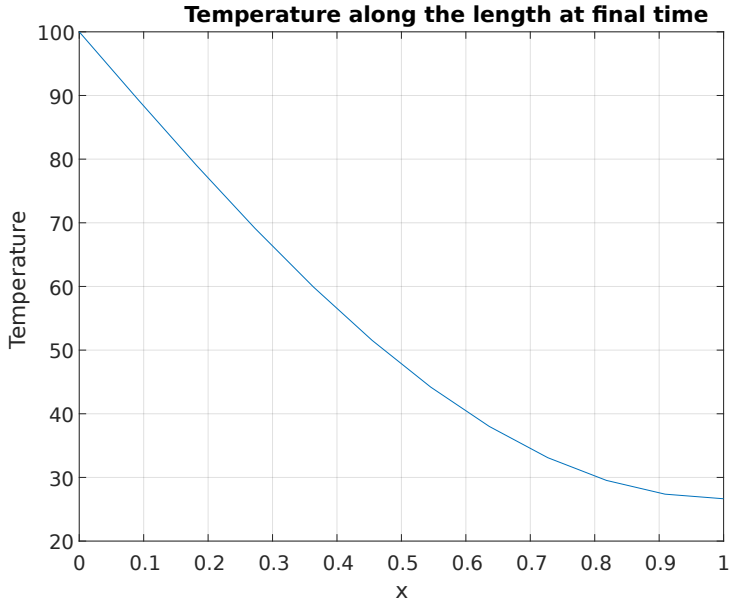
4.1.2 Improving Performance with Vectorized Mode

Execution time can often be substantially reduced by using the vectorized option. The following changes to the script above are required. The heatpde function is replaced with

```
function [c,f,s] = heatpde(x,t,u,DuDx)
rho=8940; % material density
cp=390; % material specific heat
k=385; % material thermal conductivity
nx=length(x);
c = rho*cp*ones(1,nx);
f = k*DuDx;
s = zeros(1,nx);
end
```

And pde1dm is called as follows

```
options.Vectorized='on';
u = pde1dm(m, @heatpde,@heatic,@heatbc,x,t,options);
```



4.2 Simple Coupling of PDE and ODE Equations

The Numerical Algorithms Group (NAG) provides a function, `d03phf`, for solving coupled systems of PDE and ODE equations (reference [3]). Their simple example coupling a single PDE with a single PDE will be shown here.

The single PDE, is

$$v^2 \frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} + xv \frac{\partial v}{\partial t} \frac{\partial u}{\partial x} \tag{5}$$

defined on the domain from $x = 0$ to $x = 1$
and the single ODE is

$$\frac{\partial v}{\partial t} = vu(1) + \frac{\partial u}{\partial x}(1) + t + 1 \tag{6}$$

where $u(1)$ is the solution at $x = 1$ and $\partial u/\partial x(1)$ is the x-derivative at $x = 1$.

The left boundary condition at $x = 0$ is

$$\frac{\partial u}{\partial x} = -ve^t \tag{7}$$

The right boundary condition at $x = 1$ is

$$\frac{\partial u}{\partial x} = -v\dot{v} \tag{8}$$

An analytic solution to this problem is

$$u = \quad e^{(1-x)t} - 1 \tag{9}$$

$$v = \qquad t \tag{10}$$

$$\tag{11}$$

The initial conditions on u and v in the `pde1dm` solution are computed from this exact solution.

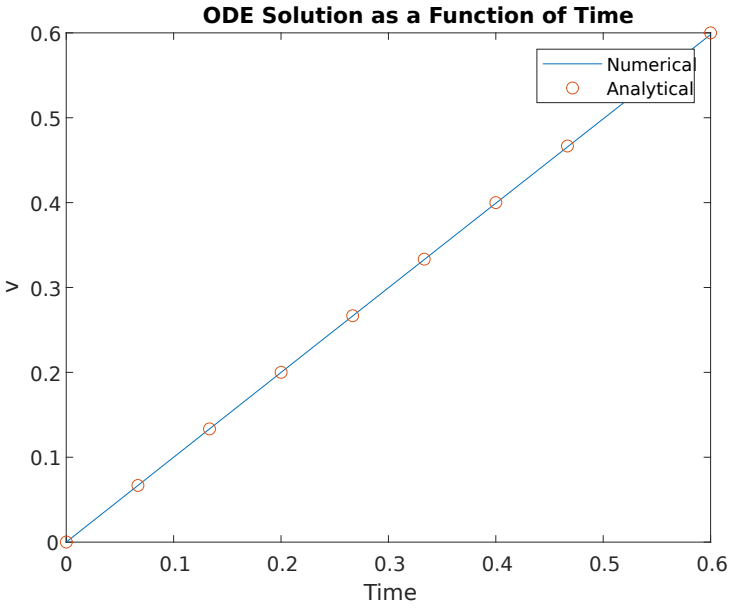
The code for this example is shown below

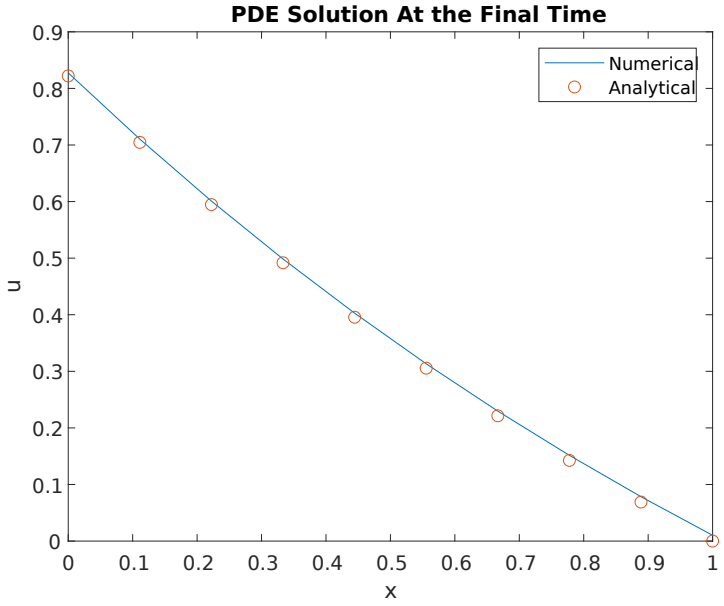
```
function nagD03phfExample
n=10;
L=1;
x = linspace(0,L,n);
t0=1e-4;
t=linspace(t0, .6, 10);
m = 0;
xOde = L;
icF = @(x) icFunc(x,t0);
odeIcF = @() odeIcFunc(t0);
opts.vectorized='on'; % speed up computation
[u,uode] = pde1dm(m, @pdeFunc,icF,@bcFunc,x,t,@odeFunc, odeIcF,xOde,opts);
va=vAnal(t);
ua=uAnal(t,x);

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)
% for vectorized mode, we return coefficients at
% multiple x locations
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);
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)
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
```

The code produces the following output and figures

```
Maximum error in ODE valriable= 2.01e-03
Maximum error in PDE valriable= 1.02e-02
```





4.3 Nonlinear Heat Equation with Periodic Boundary Conditions

Here is another example showing the usefulness of being able to couple ODE equations with the PDE equations. For the solution to be periodic, the following two conditions must hold

$$u(a) = u(b) \tag{12}$$

and

$$\frac{\partial u}{\partial x}(a) = \frac{\partial u}{\partial x}(b) \tag{13}$$

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 (13) by defining an ODE variable, v, such that

$$\frac{\partial u}{\partial x}(a) = v \tag{14}$$

$$\frac{\partial u}{\partial x}(b) = v \tag{15}$$

The ODE used to enforce equation (12) is simply

$$u(a) - u(b) = 0 \tag{16}$$

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 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
```

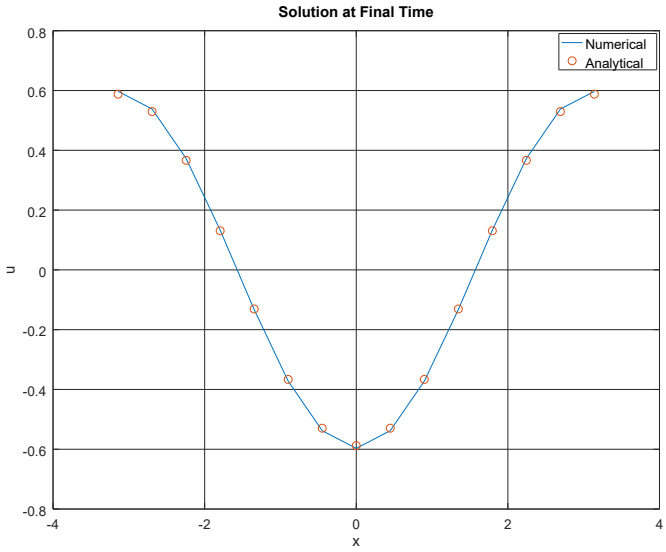


Figure 1: Solution at the final time as a function of x.

```
% the solution at the left end.
f=u(1)-u(2);
end

function v0=odeIcFunc()
v0=0;
end

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

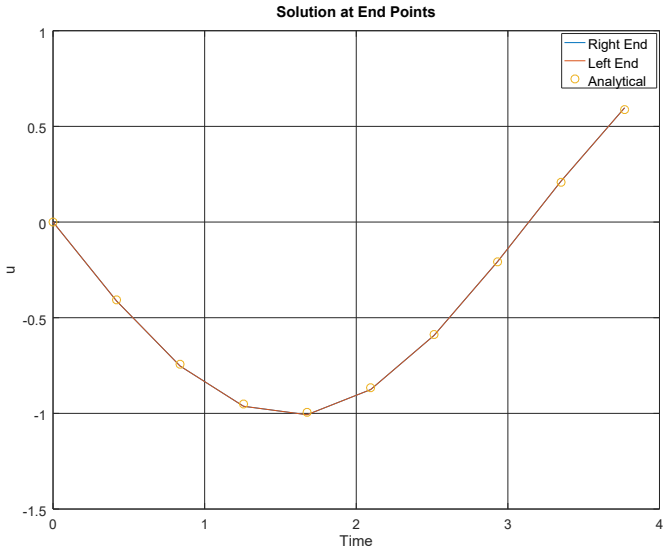


Figure 2: Solution at the two ends as a function of time.

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.

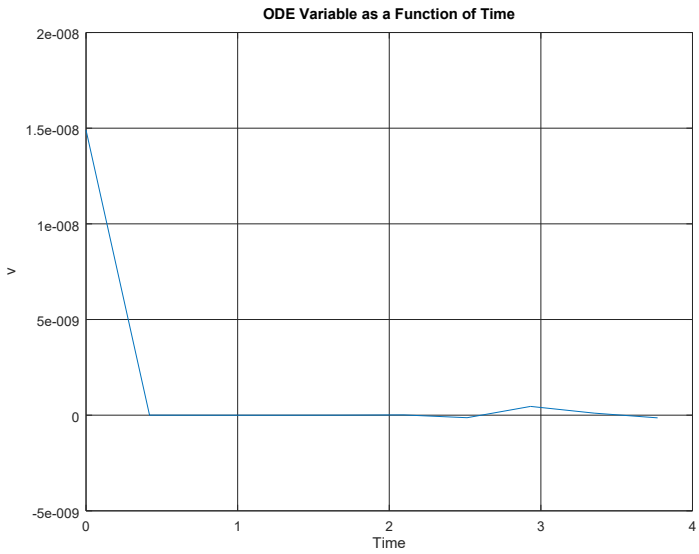


Figure 3: Solution of the ODE equation as a function of time.

4.4 Dynamics of An Elastic Structural Beam

This example shows how the structural dynamics of a beam can be analyzed with `pde1dm` . The classical PDE for transient deflections of a beam is second order in time and fourth order in space. The form of PDE that `pde1dm` accepts is first order in time and second order in space. One of the specific goals of this example is to show how the beam equation can be converted to a form acceptable to `pde1dm` .

The classical equation for the deflection of a beam is

$$\frac{\partial^2}{\partial x^2}(EI\frac{\partial^2 w}{\partial x^2}) + N\frac{\partial^2 w}{\partial x^2} + \rho A\frac{\partial^2 w}{\partial t^2} = F \tag{17}$$

The variables in this equation are:

w	Transverse deflection of the beam
x	Coordinate along the beam axis
t	Time
E	Modulus of elasticity of the material
I	Moment of inertia of the beam cross section
N	Prescribed axial force in the beam
ρ	Density of the material
A	Cross sectional area of the beam
F	Distributed transverse loading on the beam

To convert this equation into a form acceptable to `pde1dm` , we first define the following auxiliary variables

$$\frac{\partial^2 w}{\partial x^2} = K \tag{18}$$

and

$$\frac{\partial w}{\partial t} = \dot{w} = V \tag{19}$$

With these two definitions equation (17) can be rewritten as this system of three PDE

$$\left\{ \begin{array}{c} \dot{w} \\ \rho A\dot{V} \\ 0 \end{array} \right\} = \left\{ \begin{array}{c} 0 \\ -EI\frac{\partial^2 K}{\partial x^2} - N\frac{\partial^2 w}{\partial x^2} \\ -\frac{\partial^2 w}{\partial x^2} \end{array} \right\} + \left\{ \begin{array}{c} V \\ F \\ K \end{array} \right\} \tag{20}$$

The dependent variables in this system are

$$u = \left\{ \begin{array}{c} w \\ V \\ K \end{array} \right\} \tag{21}$$

4.4.1 PDE Definition

`pde1dm` requires that the user write a function with the following signature to define the system of PDE.

```
[c,f,s] = pdefun(x,t,u,dudx)
```

The details of the input and output arguments to this function are defined in the `pde1dm` documentation. A `pdefun` function defining equations (20) and (21) takes the following form

```
c = [1 rho*A 0]';
f = [0 -E*I*DuDx(3)-N*DuDx(1) -DuDx(1)]';
s = [u(2) F u(3)]';
```

4.4.2 Boundary Conditions

`pde1dm` requires that the user write a function with the following signature to define the boundary conditions at the left and right ends of the spatial region.

```
[pl,ql,pr,qr] = bcfun(xl,ul,xr,ur,t)
```

In the structural analysis of beams, three types of boundary conditions are frequently considered: free, simple support, and clamped. The two ends of the beam may have any combination of these three basic types. In the code snippets below, u, p, and q represent the value of the corresponding variable at either the left or right end.

Free

```
p = [0 0 u(3)]';
q = [1 1 0]';
```

Simple Support

```
p = [u(1) u(2) u(3)]';
q = [0 0 0]';
```

Clamped

```
p = [u(1) u(2) 0]';
q = [0 0 1]';
```

4.4.3 Free Vibration of a Simply Supported Beam

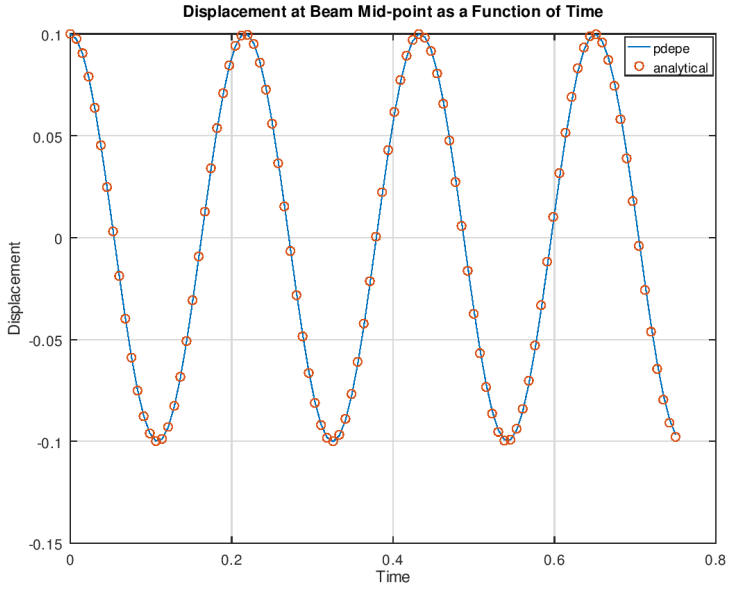
As a simple example we will consider a beam that is initially displaced but is otherwise unloaded and has simple support boundary conditions at both ends. The initial displacement is chosen to be a half sin wave over the length of the beam. This is the eigenvector for the lowest vibration frequency and this allows for a particularly simple analytical solution. We will compare the analytical solution with the solution obtained from `pde1dm` .

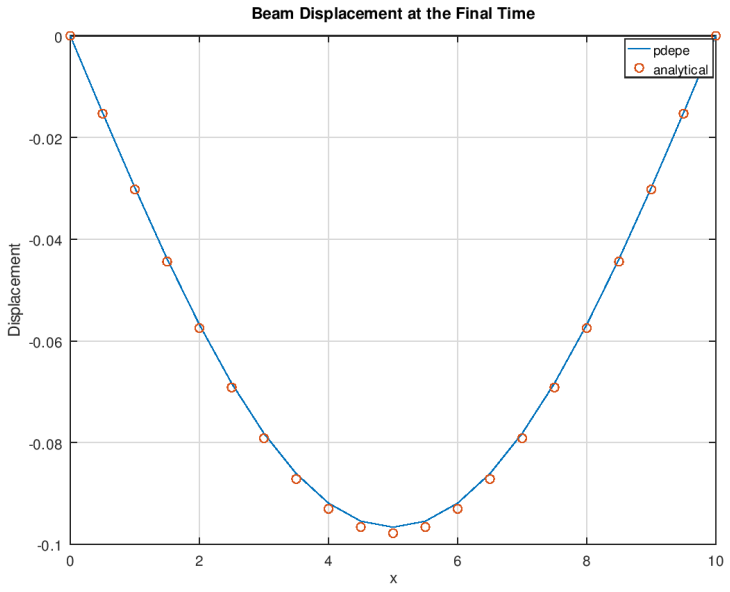
The complete listing of the `MATLAB` code for this example is shown below.

```
function beamFreeVibration
E=200e9;
thick = .2;
width = .1;
I = width*thick^3/12;
EI=E*I;
A=thick*width;
L=10;
F=0;
N=0;
xMidPoint = L/2;
rho=7700; % density of steel, kg/m^3
m = 0;
amp=.1;
numElems=20; % even number of elements gives a node in the middle of the beam
elemLength = L/numElems;
numNodes = numElems + 1;
x = linspace(0,L,numNodes);
t=linspace(0, .75, 100);
```

```
% use anonymous functions to pass parameters to functions required by pde1dm
pde = @(x,t,u,DuDx) beampde(x,t,u,DuDx, EI, rho, A, F, N);
bc = @(xl,ul,xr,ur,t) beambc(xl,ul,xr,ur,t);
ic = @(x) beamic(x, L, amp);

sol = pde1dm(m,pde,ic,bc,x,t);

% plot the results

midNode = int32(numElems/2 + 1);
uMidPt = sol(:, midNode, 1);
uAnalVibr = analFreeVibr(L, EI, rho*A, amp, x, t);
figure; plot(t, uMidPt, t, uAnalVibr(:,midNode), 'o'); grid;
xlabel Time; ylabel Displacement;
title('Displacement␣at␣Beam␣Mid-point␣as␣a␣Function␣of␣Time');
legend('pde1dm', 'analytical');
figure; plot(x,sol(end,:,1), x, uAnalVibr(end, :), 'o'); grid;
xlabel x; ylabel Displacement;
title('Beam␣Displacement␣at␣the␣Final␣Time');
legend('pde1dm', 'analytical');

end

function [cr,fr,sr] = beampde(x,t,u,DuDx, EI, rho, A, F, N)
cr = [1 rho*A 0]';
fr = [0 -EI*DuDx(3)-N*DuDx(1) -DuDx(1)]';
sr = [u(2) F u(3)]';
end

function u0 = beamic(x, L, amp)
  % half sin wave initial condition
  s = sin(pi*x/L);
  u0 = [amp*s; 0; -amp*(pi/L)^2*s];
end

function [pl,ql,pr,qr] = beambc(xl,ul,xr,ur,t)
  pl = [ul(1) ul(2) ul(3)]';
  ql = [0 0 0]';
  pr = [ur(1) ur(2) ur(3)]';
  qr = [0 0 0]';
end

function u=analFreeVibr(L, EI, rhoA, amp, x, t)
 omega=(pi/L)^2*sqrt(EI/rhoA);
 u=amp*cos(omega*t)'*sin(pi*x/L);
 end
```

The results are shown in the following two figures. As can be seen, agreement between the pde1dm and analytical solutions is very good.





4.5 Melting of Ice

This example shows the melting of a semi-infinite ice slab. We have chosen the somewhat unrealistic semi-infinite case because it has a simple analytical solution that we will compare with. Most of what is learned in this example can be applied to more realistic solidification and melting problems and also problems where moving coordinate systems are advantageous.

Problems of this type are often referred to as Stefan problems named after the physicist who studied melting and freezing in the 1800's. They are characterized by a moving "front" that separates one material phase from the other. They are discussed in many publications including [4].

The specific problem we will consider here is a slab of ice at zero degrees-C with a temperature of 25 degrees-C suddenly applied to the left ($x = 0$) face. The extent of the block in the other directions is assumed to be large. This allows the use of a one-dimensional model of heat transfer and implies that the block never fully melts during the time of the simulation. We model only the liquid phase that initially has length zero and lengthens as heat is transferred from the left edge into the ice. Such a model is referred to as a one-phase Stefan model.

4.5.1 Governing Equations

A key parameter of such a model is the evolving location of the liquid-solid interface. This is given by the Stefan equation, an ordinary differential equation (ODE) that relates the heat flux in the ice to the velocity of the interface location:

$$\rho l \frac{\partial X}{\partial t} = k \frac{\partial u}{\partial x} \tag{22}$$

where u is the temperature, X is the front location, ρ is the density, l is the latent heat, and k is the thermal conductivity.

This ODE is coupled to the classical partial differential equation (PDE) for heat conduction in the liquid.

$$\rho c \frac{\partial u}{\partial t} = \frac{\partial}{\partial x}\left(k \frac{\partial u}{\partial x}\right) \tag{23}$$

where c is the heat capacity of the fluid.

4.5.2 Moving Mesh

Because the analysis domain is continuously expanding as the analysis progresses it is necessary to move the mesh points at the same time. First, a parametric coordinate system is established that ranges from zero at the left end to one at the right end. Then, at any point in time the actual coordinates of any point in the domain can be defined as

$$x = \xi X(t) \tag{24}$$

where ξ is the parametric coordinate and X is determined from the solution to the Stefan equation. We use this relation to rewrite the heat equation, accounting for the moving mesh, as follows

$$x_\xi \frac{\partial u}{\partial t} = \left(\frac{k}{\rho c x_\xi} \frac{\partial u}{\partial \xi}\right)_\xi + \frac{1}{x_\xi} \frac{\partial u}{\partial \xi} \frac{\partial x}{\partial t} \tag{25}$$

ξ used as a subscript indicates partial differentiation, e.g. $x_\xi = \partial x / \xi$.

4.5.3 Implementation and Results

Equations (22) and (25) are coded in the functions `odeFunc` and `heatpde` respectively. A mesh is defined in parametric space

```
n=21;
% Define mesh in parametric space ranging from zero to one.
xi = linspace(0,1,n);
```

The PDE and ODE are coupled at the single point– the right end of the computational domain.

```
xOde = 1;
```

pde1dm passes u and $\partial u/\partial \xi$ at this location as arguments to odeFunc so they can be used in defining the Stefan equation. pde1dm also passes X and $\partial X/\partial t$ as arguments to heatpde so they can be used to define the transformation from parametric to the actual coordinates.

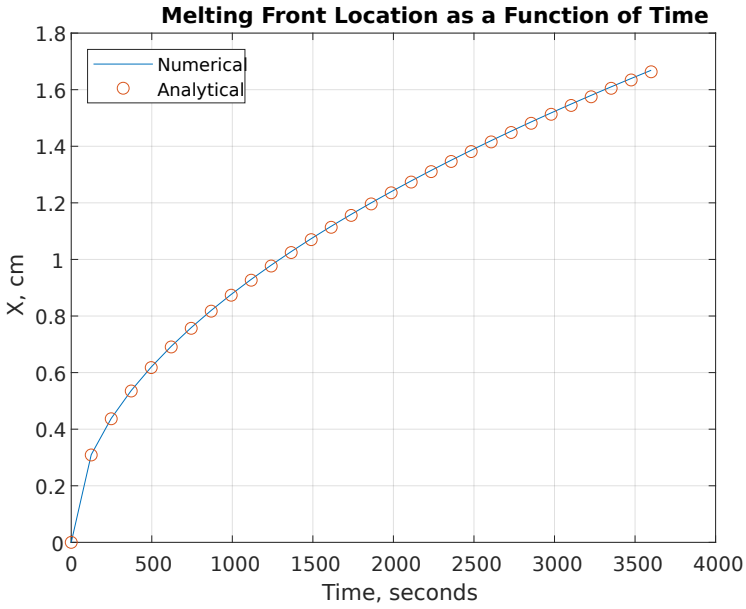
As mentioned above, a closed form analytic solution can be found for this simple melting problem. This solution can be found in section 2.1 of reference [4], for example. This is implemented in function stefanOnePhaseAnalSoln.

The complete listing of the MATLAB code for this example is shown below.

```
function meltingIce
% Melting of a semi-infinite block of ice
n=21;
% Define mesh in parametric space ranging from zero to one.
xi = linspace(0,1,n);

tfinal=3600; % simulate block melting for one hour
nt=30;
t = linspace(0,tfinal,nt);

% physical properties of water
rho=1; % density, gm/cm^3
cp=4.1868; % heat capacity
k=.564e-2; % thermal conductivity
latentHeat=333.4; % J/g
alpha=k/(rho*cp);
xOde = 1;
TL=25;
Tm=0;
St=cp*(TL-Tm)/latentHeat; % Stefan number

m=0;
opts.vectorized='on'; % use vectorized option to improve performance
% tighten the accuracy tolerances for ODE solver
opts.reltol=1e-5;
opts.abstol=1e-7;
pdeFunc = @(x,t,u,DuDx,X,Xdot) heatpde(x,t,u,DuDx,X,Xdot,alpha);
icFunc = @(x) heatic(x);
bcFunc = @(xl,ul,xr,ur,t) heatbcDir(xl,ul,xr,ur,t,TL,Tm);
odeIcF = @() odeIcFunc();
odef=@(t,X,Xdot,x,u,DuDxi) odeFunc(t,X,Xdot,x,u,DuDxi,latentHeat, rho, k);
[u,X] = pde1dm(m, pdeFunc,icFunc, bcFunc,xi,t,odef, odeIcF,xOde,opts);

[XAnal,uAnal]=stefanOnePhaseAnalSoln(TL,Tm,St,alpha,t,xi);
x=xi'*X';
xAnal=xi'*XAnal';
figure; plot(x(:,end), u(end,:), xAnal(:,end), uAnal(end,:), 'o'); grid;
title(sprintf('Temperature in the slab at %d seconds', tfinal));
xlabel 'x, cm'; ylabel 'Temperature, degrees-C';
legend('Numerical', 'Analytical');

figure; plot(t, X, t, XAnal, 'o'); ylabel('X, cm'); grid;
xlabel 'Time, seconds';
title 'Melting Front Location as a Function of Time';
```

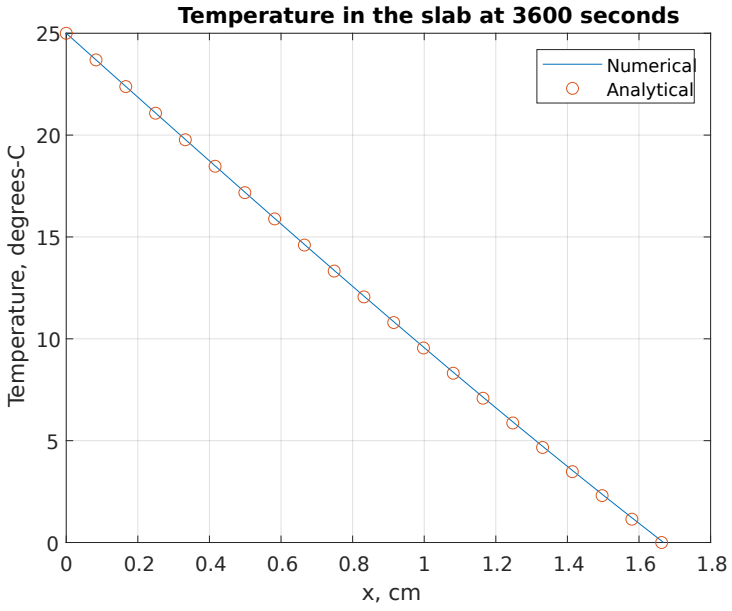
```
legend('Numerical', 'Analytical', 'Location','northwest');

end

function [c,f,s] = heatpde(xi,~,u,DuDxi,X,Xdot,alpha)
% x = xi*X(t)
dxDXi=X;
dxDt=Xdot*xi;
% evaluate the PDE at all xi locations (vectorized mode)
nx=length(xi);
c = dxDXi*ones(1,nx);
f = alpha*DuDxi/dxDXi;
s=DuDxi.*dxDt;
end

function u0 = heatic(x)
u0 = 0;
end

function [pl,ql,pr,qr] = heatbcDir(xl,ul,xr,ur,t,TL,Tm)
pl = ul-TL;
ql = 0;
pr = ur-Tm;
qr = 0;
end

function f=odeFunc(t,X,Xdot,x,u,DuDxi,latentHeat, rho, k)
dxDXi=X(1);
f1=rho*latentHeat*Xdot(1)+k*DuDxi/dxDXi;
f=f1;
end

function X0=odeIcFunc()
X0=1e-5; % start with a near-zero initial front position
end

function [frontLocation,temperature]=stefanOnePhaseAnalSoln(TL,Tm,St,alpha,t,xi)
lambda=sqrt(St/2);
f= @(lam) lam*exp(lam^2)*erf(lam)-St/sqrt(pi);
lambda=fzero(f, lambda);
frontLocation=2*lambda*sqrt(alpha*t');
nx=length(xi);
nt=length(t);
t=t(:)';
temperature=zeros(nt,nx);
for i=1:nt
  Xi=frontLocation(i)*xi(:)';
  temperature(i,:)=TL - (TL-Tm)*erf(Xi/(2*sqrt(alpha*t(i))))/erf(lambda);
  temperature(temperature<0) = Tm;
end
end
```

This figure shows the progression of the melting location as a function of time.



As can be seen, the agreement between the `pde1dm` solution and the closed form solution is very good.

This figure shows the temperature as a function of x-location in the liquid at the final analysis time (3600 seconds).



Agreement with the analytical solution is also quite good.

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 currently support events.

3. `pdepe` relies on the `MATLAB ode15s` ODE solver to solve initial value problem. `pde1dm` uses the `ode15i` solver.

4. The Octave version of `ode15i` cannot solve ODE with complex matrices so this limits `pde1dm` , when used with Octave, to only real PDE.

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] Partial Differential Equations in MATLAB 7.0

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

[4] Vasilios Alexiades and Alan D. Solomon, *Mathematical Modeling Of Melting And Freezing Processes*, (Taylor & Francis), 1993.