Solving the Heat Equation

Background

We consider a thin, perfectly insulated, rod of length L [m] (see Figure 1). Time-varying temperatures are imposed on each end. At the left end the temperature is given by $left(t)$, while the at the right end the temperature is given by $right(t)$.

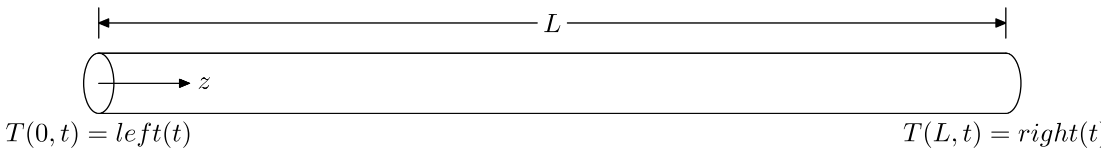

Figure 1: A thin, perfectly-insulated, rod with specified, time-varying, temperatures imposed on both ends.

We are interested in modeling the temperature along the entire rod as a function of location and time: $T(x, t)$. The initial temperature of the rod is uniformly 0. That is,

$$T(x, 0) = left(0) = right(0) = 0 \tag{1}$$

for all $0 \leqslant x \leqslant L$.

Mathematics

Temperature as a function of time and locations is described by a well-known partial differential equation. Not surprisingly, this partial differential equation is known as the *Heat Equation*[1].

$$\frac{\partial T}{\partial t} = \frac{k}{c_p \rho} \frac{\partial^2 T}{\partial x^2} \tag{2}$$

where

- T is the temperature,
- t is the time,
- k is the *thermal conductivity*[2],
- c_p is the *specific heat capacity*[3],
- ρ is the *mass density* of the material, and
- x is the space coordinate.

[1] See, for example, http://en.wikipedia.org/wiki/Heat_equation.

[2] See, for example, http://en.wikipedia.org/wiki/Thermal_conductivity. According to this page, the thermal conductivity for Portland Cement is approximately 0.29 [$W/(mKelvin)$], which that of concrete and stone is approximately 1.7 [$W/mKelvin$].

[3] See, for example, http://en.wikipedia.org/wiki/Specific_heat_capacity.

The coefficient $\frac{k}{c_p \rho}$ is called the *thermal diffusivity*[4] and is often denoted using α. This substitution visually simplifies the partial differential equation to

$$\boxed{\frac{\partial T}{\partial t} = \alpha \frac{\partial^2 T}{\partial x^2}} \tag{3}$$

This is the form that we will be working with in this problem.

MATLAB

The heat equation is the a *parabolic partial differential equation*[5], and is amenable to a numerical solution using various methods. In fact, MATLAB has a built-in function for doing just that: `pdepe`.

```
pdepe  Solve initial-boundary value problems for parabolic-elliptic PDEs in 1-D.
    SOL = pdepe(M,PDEFUN,ICFUN,BCFUN,XMESH,TSPAN) solves initial-boundary
    value problems for small systems of parabolic and elliptic PDEs in one
    space variable x and time t to modest accuracy. There are npde unknown
    solution components that satisfy a system of npde equations of the form

    c(x,t,u,Du/Dx) * Du/Dt = x^(-m) * D(x^m * f(x,t,u,Du/Dx))/Dx + s(x,t,u,Du/Dx)

    Here f(x,t,u,Du/Dx) is a flux and s(x,t,u,Du/Dx) is a source term. m must
    be 0, 1, or 2, corresponding to slab, cylindrical, or spherical symmetry,
    respectively. The coupling of the partial derivatives with respect to
    time is restricted to multiplication by a diagonal matrix c(x,t,u,Du/Dx).
    The diagonal elements of c are either identically zero or positive.
    An entry that is identically zero corresponds to an elliptic equation and
    otherwise to a parabolic equation. There must be at least one parabolic
    equation. An entry of c corresponding to a parabolic equation is permitted
    to vanish at isolated values of x provided they are included in the mesh
    XMESH, and in particular, is always allowed to vanish at the ends of the
    interval. The PDEs hold for t0 <= t <= tf and a <= x <= b. The interval
    [a,b] must be finite. If m > 0, it is required that 0 <= a. The solution
    components are to have known values at the initial time t = t0, the
    initial conditions. The solution components are to satisfy boundary
    conditions at x=a and x=b for all t of the form

        p(x,t,u) + q(x,t) * f(x,t,u,Du/Dx) = 0

    q(x,t) is a diagonal matrix. The diagonal elements of q must be either
    identically zero or never zero. Note that the boundary conditions are
    expressed in terms of the flux rather than Du/Dx. Also, of the two
    coefficients, only p can depend on u.
```

[4]See, for example, `http://en.wikipedia.org/wiki/Thermal_diffusivity`. This page gives various values for numerous materials, but nothing for concrete. You may also find `http://www.wbdg.org/ccb/ARMYCOE/COESTDS/crd_c37.pdf` interesting, if a bit dated.

[5]See, for example, `http://en.wikipedia.org/wiki/Parabolic_partial_differential_equation`.

```
The input argument M defines the symmetry of the problem. PDEFUN, ICFUN,
and BCFUN are function handles.

[C,F,S] = PDEFUN(X,T,U,DUDX) evaluates the quantities defining the
differential equation. The input arguments are scalars X and T and
vectors U and DUDX that approximate the solution and its partial
derivative with respect to x, respectively. PDEFUN returns column
vectors: C (containing the diagonal of the matrix c(x,t,u,Dx/Du)),
F, and S (representing the flux and source term, respectively).

U = ICFUN(X) evaluates the initial conditions. For a scalar X, ICFUN
must return a column vector, corresponding to the initial values of
the solution components at X.

[PL,QL,PR,QR] = BCFUN(XL,UL,XR,UR,T) evaluates the components of the
boundary conditions at time T. XL and XR are scalars representing the
left and right boundary points. UL and UR are column vectors with the
solution at the left and right boundary, respectively. PL and QL are
column vectors corresponding to p and the diagonal of q, evaluated at
the left boundary, similarly PR and QR correspond to the right boundary.
When m > 0 and a = 0, boundedness of the solution near x = 0 requires
that the flux f vanish at a = 0. pdepe imposes this boundary condition
automatically.
```

Making some sense of pdepe

Our problem at hand is

$$\boxed{\frac{\partial T}{\partial t} = \alpha \frac{\partial^2 T}{\partial x^2}} \tag{4}$$

As given in the MATLAB documentation, pdepe solves a partial differential equation of the following form:

$$c\left(x,t,u,\frac{\partial u}{\partial x}\right)\frac{\partial u}{\partial t} = x^{-m}\frac{\partial}{\partial x}\left(x^m f\left(x,t,u,\frac{\partial u}{\partial x}\right)\right) + s\left(x,t,u,\frac{\partial u}{\partial x}\right) \tag{5}$$

We are not working with an axis symmetric, or spherically symmetric, problem so $m = 0$, and (5) simplifies to

$$c\left(x,t,u,\frac{\partial u}{\partial x}\right)\frac{\partial u}{\partial t} = \frac{\partial}{\partial x}\left(f\left(x,t,u,\frac{\partial u}{\partial x}\right)\right) + s\left(x,t,u,\frac{\partial u}{\partial x}\right) \tag{6}$$

Our "c" function is simply a constant, $c = 1$, and we do not have an "s" function at all, $s = 0$. Thus, (6) simplifies to

$$\frac{\partial u}{\partial t} = \frac{\partial}{\partial x}\left(f\left(x,t,u,\frac{\partial u}{\partial x}\right)\right) \tag{7}$$

Looking at (7) we see that $f = \alpha\frac{\partial u}{\partial x}$, so we have

$$\frac{\partial u}{\partial t} = \frac{\partial}{\partial x}\left(\alpha\frac{\partial u}{\partial x}\right) = \alpha\frac{\partial^2 u}{\partial x^2} \tag{8}$$

Which is right where we wanted to be.

In summary,

- $m = 0$
- $u = T$
- $c = 1$
- $s = 0$
- $f = \alpha \frac{\partial u}{\partial x}$

An example

```
function T = Example( alpha, lambda1, lambda2 )
    m = 0;
    z = linspace(0,1,50);
    t = linspace(0,5,100);

    sol = pdepe(0, @pdefun, @icfun, @bcfun, z, t);
    % Extract the first solution component as T.
    T = sol(:,:,1);

    % A surface plot is often a good way to study a solution.
    surf(z,t,T)
    title('Numerical solution.')
    xlabel('Depth z')
    ylabel('Time t')

    % A solution profile can also be illuminating.
    figure
    plot(z,T(end,:))
    title('Solution at t = 2')
    xlabel('Depth z')
    ylabel('T(z,2)')

    % --------------------------------------------------------------
    function [c,f,s] = pdefun(x, t, T, dTdx)
        c = 1;
        f = alpha * dTdx;
        s = 0;
    end

    % --------------------------------------------------------------
    function T0 = icfun(z)
        T0 = 0;
    end

    % --------------------------------------------------------------
```

```
    function [pl,ql,pr,qr] = bcfun(zl, Tl, zr, Tr, t)
        pl = Tl - 2*sin(2*pi*t/lambda1);
        ql = 0;
        pr = Tr - sin(2*pi*t/lambda2);
        qr = 0;
    end
end
```