Lecture 38

Insulated Boundary Conditions

Insulation

In many of the previous sections we have considered fixed boundary conditions, i.e. $u(0) = a$, $u(L) = b$. We implemented these simply by assigning $u_0^j = a$ and $u_n^j = b$ for all j.

We also considered variable boundary conditions, such as $u(0,t) = g_1(t)$. For example, we might have $u(0,t) = \sin(t)$ which could represents periodic heating and cooling of the end at $x = 0$.

A third important type of boundary condition is called the *insulated* boundary condition. It is so named because it mimics an insulator at the boundary. Physically, the effect of insulation is that no heat flows across the boundary. This means that the temperature gradient is zero, which implies that we should require the mathematical boundary condition $u'(L) = 0$.

To use it in a program, we must replace $u'(L) = 0$ by a discrete version. Recall that in our discrete equations we usually have $L = x_n$. Recall from the section on numerical derivatives, that there are three different ways to replace a derivative by a difference equation, left, right and central differences. The three of them at x_n would be

$$u'(x_n) \approx \frac{u_n - u_{n-1}}{h} \approx \frac{u_{n+1} - u_n}{h} \approx \frac{u_{n+1} - u_{n-1}}{2h}.$$

If x_n is the last node of our grid, then it is clear that we cannot use the right or central difference, but are stuck with the first of these. Setting that expression to zero implies

$$u_n = u_{n-1}.$$

This restriction can be easily implemented in a program simply by putting a statement `u(n+1)=u(n)` inside the loop that updates values of the profile. However, since this method replaces $u'(L) = 0$ by an expression that is only accurate to first order, it is not very accurate and is usually avoided.

Instead we want to use the most accurate version, the central difference. For that we should have

$$u'(L) = u'(x_n) = \frac{u_{n+1} - u_{n-1}}{2h} = 0.$$

or simply

$$u_{n+1} = u_{n-1}.$$

However, u_{n+1} would represent $u(x_{n+1})$ and x_{n+1} would be $L + h$, which is outside the domain. This, however, is not an obstacle in a program. We can simply extend the grid to one more node, x_{n+1}, and let u_{n+1} always equal u_{n-1} by copying u_{n-1} into u_{n+1} whenever u_{n-1} changes. The point x_{n+1} is "fictional", but a computer does not know the difference between fiction and reality! This idea is carried out in the calculations of the next section and illustrated in Figure 38.1.

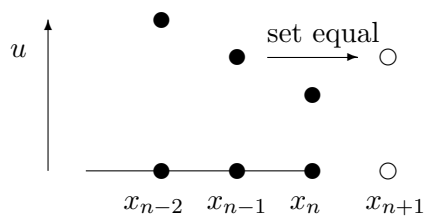

Figure 38.1: Illustation of an insulated boundary condition using a fictional point x_{n+1} with $u_{n+1} = u_{n-1}$.

A way to think of an insulated boundary that makes sense of the point $L + h$ is to think of two bars joined end to end, where you let the second bar be mirror image of the first bar. If you do this, then no heat will flow across the joint, which is exactly the same effect as insulating.

Another practical way to implement an insulated boundary is to let the grid points straddle the boundary. For example suppose we want to impose insulated boundary at the left end of a bar, i.e. $u'(0) = 0$, then you could let the first two grid points be at $x_0 = -h/2$ and $x_1 = h/2$. Then you can let

$$u_0 = u_1.$$

This will again force the central difference at $x = 0$ to be 0.

Implementation in a linear equation by elimination

Consider the BVP

$$u_{xx} = -1 \quad \text{with} \quad u(0) = 5 \quad \text{and} \quad u'(1) = 0. \tag{38.1}$$

This represents the steady state temperature of a bar with a uniformly applied heat source, with one end held at a fixed temperature and the other end insulated.

If we use 4 equally spaced intervals, then

$$m = 4 \quad \text{and} \quad L = 1 \quad \Rightarrow \quad h = \frac{L}{m} = \frac{1}{4},$$

and

$$x_0 = 0,\ x_1 = .25,\ x_2 = .5,\ x_3 = .75,\ x_4 = 1, \text{ and } x_5 = 1.25.$$

The point $x_5 = 1.25$ is outside the region and thus fictional. The boundary condition at $x_0 = 0$ is implemented as

$$u_0 = 5.$$

For the insulated condition, we will require

$$u_5 = u_3.$$

This makes the central difference for $u'(x_4)$ be zero. We can write the differential equation as a difference equation

$$\frac{u_{i-1} - 2u_i + u_{i+1}}{h^2} = -1$$

or

$$u_{i-1} - 2u_i + u_{i+1} = -0.0625, \quad i = 1, 2, 3, 4.$$

For $i = 1$, recalling that $u_0 = 5$, we have

$$5 - 2u_1 + u_2 = -.0625 \quad \text{or} \quad -2u_1 + u_2 = -5.0625.$$

For $i = 2$ and $i = 3$ we have

$$u_1 - 2u_2 + u_3 = -.0625 \quad \text{and} \quad u_2 - 2u_3 + u_4 = -.0625.$$

For $i = 4$ we have

$$u_3 - 2u_4 + u_5 = -.0625\,.$$

Note that we now have 5 unknowns in our problem: $u_1, \ldots, u_5$. However, from the boundary condition $u_5 = u_3$ and so we can eliminate u_5 from our $i = 4$ equation and write

$$2u_3 - 2u_4 = -.0625\,.$$

Summarizing, we can put the unknown quantities in a vector $\mathbf{u} = (u_1, u_2, u_3, u_4)'$ and write the equations as a matrix equation $A\mathbf{u} = \mathbf{b}$ where

$$A = \begin{pmatrix} -2 & 1 & 0 & 0 \\ 1 & -2 & 1 & 0 \\ 0 & 1 & -2 & 1 \\ 0 & 0 & 2 & -2 \end{pmatrix}$$

and $\mathbf{b} = (-5.0625, -.0625, -.0625, -.0625)'$. Solve this system and plot the results:

```
≫ u = A\b
≫ u = [5 ; u]
≫ x = 0:.25:1
≫ plot(x,u,'d')
```

Then interpolate with a spline.

The exact solution of this BVP is:

$$U(x) = 5 + x - .5x^2.$$

Use `hold on` and plot this function on the same graph to compare:

```
≫ xx = 0:.01:1;
≫ uu = 5 + xx - .5*xx.^2;
≫ hold on
≫ plot(xx,uu,'r')
```

You should see that our approximate solution is almost perfect!

Insulated boundary conditions in time-dependent problems

To implement the insulated boundary condition in an explicit difference equation with time, we need to copy values from inside the region to fictional points just outside the region. Note that you copy the *new* value from inside the region to the false point during each time step, i.e. inside the loop, but after the values are updated at the real points. See Figure 38.2 for an illustration.

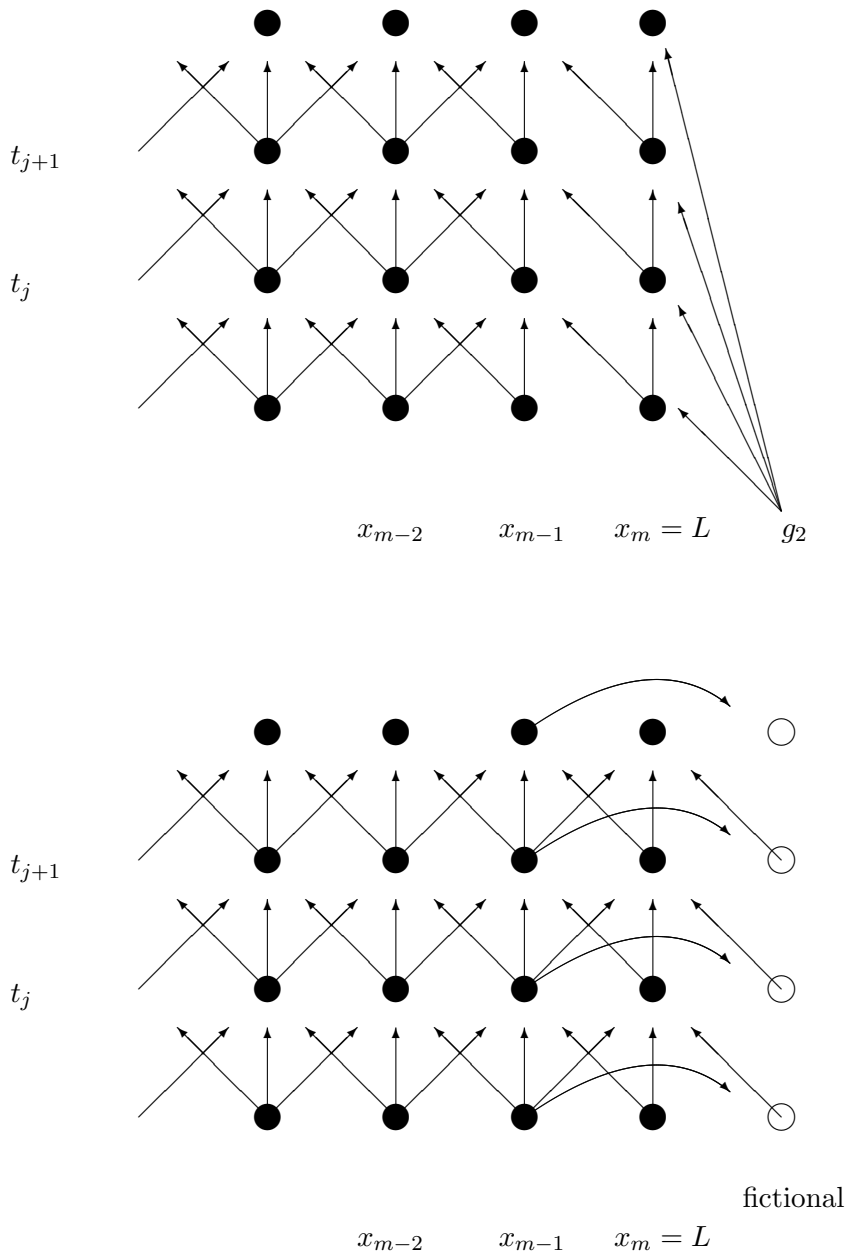

Figure 38.2: Illustration of information flow for the explicit method near the right boundary at $x = L$. The top figure shows a fixed boundary condition, where $u_{m,j}$ is set to be $g_2(t_j)$. The bottom figure shows an insulating boundary condition. Now $u_{m,j}$ is updated in the same way as the general $u_{i,j}$ and an additional entry $u_{m+1,j}$ is used with its value set by copying $u_{m-1,j}$.

An example

The steady state temperature $u(r)$ (given in polar coordinates) of a disk subjected to a radially symmetric heat load $g(r)$ and cooled by conduction to the rim of the disk and radiation to its environment is determined by the boundary value problem

$$\frac{\partial^2 u}{\partial r^2} + \frac{1}{r}\frac{\partial u}{\partial r} = d(u^4 - u_b^4) - g(r) \quad \text{with} \quad u(R) = u_R \quad \text{and} \quad u'(0) = 0. \tag{38.2}$$

Here u_b is the (fixed) background temperature and u_R is the (fixed) temperature at the rim of the disk.

The program `myheatdisk.m` implements these equations for parameter values $R = 5$, $d = .1$, $u_R = u_b = 10$ and $g(r) = (r-5)^2$. Notice that the equations have a singularity (discontinuity) at $r = 0$. How does the program avoid this problem? How does the program implement $u_R = 10$ and $u'(0) = 0$? Run the program.

Exercises

38.1 Redo the calculations for the BVP (38.1) except do not include the fictional point x_5. Instead, let x_4 be the last point and impose the insulated boundary by requiring $u_4 = u_3$. (Keep $m = 4$ and $h = 1/4$. Your system of equations should be 3×3.) Compare this solution with the true solution and the better approximation in the lecture. Illustrate this comparison on a single plot.

38.2 Modify the program `myheat.m` to have an insulated boundary at $x = L$ (rather than $u(L,t) = g_2(t)$). You will need to change the domain to: `x = 0:h:L+h`, change the dimensions of all the other objects to fit this domain and implement the insulation (copy) step inside the loop (see the section 'Insulated boundary conditions in time-dependent problems' and the figure).

Run the program with $L = 2\pi$, $T = 20$, $c = .6$, $g_1(t) = \sin(t)$ and $f(x) = -\sin(x/4)$. Set $m = 20$ and experiment with n. Get a plot when the program is stable. Turn in your program and plot.