9.6 Solving the Heat Equation using the Crank-Nicholson Method

The one-dimensional heat equation was derived on page 165. Let's generalize it to allow for the direct application of heat in the form of, say, an electric heater or a flame:

$$\frac{\partial T(x,t)}{\partial t} = D\frac{\partial^2 T(x,t)}{\partial x^2} + P_{applied}(x,t)\,. \tag{9.97}$$

The new term $P_{applied}(x,t)$ is the power applied (i.e. the rate at which heat energy is applied) at point x at time t. It is worthwhile to compare this generalized heat equation with the generalized wave equation, (9.93). The major difference is that the heat equation has a *first* time derivative whereas the wave equation has a *second* time derivative (if we ignore resistance).

We now wish to approximate the derivatives with finite differences. There are several possible schemes for doing this and we must see which one works best. The first scheme is called **Forward Time Centered Space** (FTCS) because it uses a forward difference to approximate the time derivative and a centered second difference to approximate the second space derivative. Thus:

$$\underbrace{\frac{T_j^{n+1} - T_j^n}{\Delta t}}_{\text{forward from time } n} = D\underbrace{\frac{T_{j+1}^n - 2T_j^n + T_{j-1}^n}{(\Delta x)^2}}_{\text{centered at } j \text{ at time } n} + P_j^n\,, \tag{9.98}$$

The definitions of n, j, Δt and Δx are the same as on page 197 and $T(x,t) \equiv T_j^n$ and $P_{applied}(x,t) \equiv P_j^n$. Solving for T_j^{n+1} gives

$$T_j^{n+1} = T_j^n + \alpha\left(T_{j+1}^n - 2T_j^n + T_{j-1}^n\right) + \Delta t \cdot P_j^n\,, \tag{9.99}$$

where $\alpha = D\dfrac{\Delta t}{(\Delta x)^2}$ is a dimensionless constant that measures the speed of the simulation.

Like Eq. (9.81) for waves, Eq. (9.99) for heat flow is an *explicit equation* – meaning that unknown grid point values at timestep n+1 can be calculated directly from known values at timestep n. Unfortunately the scheme is unstable unless Δt is made so small that the simulation is impossibly slow.

The second scheme is **Backward Time Centered Space** (BTCS). It uses a backward difference to approximate the time derivative:

$$\underbrace{\frac{T_j^{n+1} - T_j^n}{\Delta t}}_{\text{backward from time } n+1} = D\underbrace{\frac{T_{j+1}^{n+1} - 2T_j^{n+1} + T_{j-1}^{n+1}}{(\Delta x)^2}}_{\text{centered at } j \text{ at time } n+1} + P_j^{n+1}\,, \tag{9.100}$$

Notice that this is a relation among 3 future values and one present value. Let us rearrange this so that all the future values are isolated on the LHS. This gives:

$$\underbrace{-\alpha T_{j-1}^{n+1} + (1+2\alpha)T_j^{n+1} - \alpha T_{j+1}^{n+1}}_{\text{3 unknowns}} = \underbrace{T_j^n + \Delta t \cdot P_j^{n+1}}_{\text{known value}}\,, \tag{9.101}$$

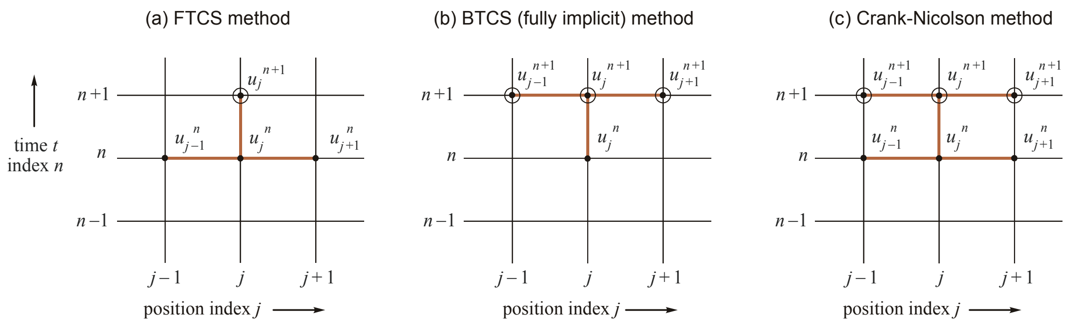

Figure 9.34 Stencils for three schemes for finite-differencing the heat equation.

We see that this is an *implicit equation* – to solve it means to solve a set of simultaneous linear equations at each timestep. Fortunately this is not a big problem since the system is tridiagonal. Also, as we will see later, this scheme is unconditionally stable.

The third scheme is called the **Crank-Nicolson method**. It takes the average of (9.98) and (9.100):

$$\frac{T_j^{n+1}-T_j^n}{\Delta t} = \frac{1}{2}\left(D\,\frac{T_{j+1}^{\,n}-2T_j^{\,n}+T_{j-1}^{\,n}}{(\Delta x)^2}+P_j^{\,n} + D\,\frac{T_{j+1}^{\,n+1}-2T_j^{\,n+1}+T_{j-1}^{\,n+1}}{(\Delta x)^2}+P_j^{\,n+1}\right) \tag{9.102}$$

Although this is again an implicit equation, it has the advantage of being symmetrical in time and is thus *more accurate* than either FTCS or BTCS. It is helpful to compare the stencils for the three schemes. They are shown in Fig. 9.34. Solving (9.102) for the new temperatures in terms of old temperatures gives

$$-\frac{\alpha}{2}T_{j-1}^{\,n+1}+(1+\alpha)T_j^{\,n+1}-\frac{\alpha}{2}T_{j+1}^{\,n+1} = \frac{\alpha}{2}T_{j-1}^{\,n}+(1-\alpha)T_j^{\,n}+\frac{\alpha}{2}T_{j+1}^{\,n}+\frac{\Delta t}{2}\left(P_j^{\,n}+P_j^{\,n+1}\right). \tag{9.103}$$

or

$$\underbrace{-\frac{\alpha}{2}T_{j-1}^{\,n+1}+(1+\alpha)T_j^{\,n+1}-\frac{\alpha}{2}T_{j+1}^{\,n+1}}_{\text{3 unknowns}} = \underbrace{R_j^{\,n}}_{\text{known value}}, \tag{9.104}$$

where

$$R_j^{\,n} = \frac{\alpha}{2}T_{j-1}^{\,n}+(1-\alpha)T_j^{\,n}+\frac{\alpha}{2}T_{j+1}^{\,n}+\frac{\Delta t}{2}\left(P_j^{\,n}+P_j^{\,n+1}\right). \tag{9.105}$$

The quantity R_j^n that appears on the RHS is a known quantity at the beginning of each timestep. To become familiar with Eq. (9.104) suppose that there are 5 grid points numbered 0 to 4. Suppose also that the boundary conditions are that the temperature equals a at the left boundary and b at the right boundary (Dirichlet conditions). Then we must solve this matrix equation:

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ -\frac{\alpha}{2} & 1+\alpha & -\frac{\alpha}{2} & 0 & 0 \\ 0 & -\frac{\alpha}{2} & 1+\alpha & -\frac{\alpha}{2} & 0 \\ 0 & 0 & -\frac{\alpha}{2} & 1+\alpha & -\frac{\alpha}{2} \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} T_0^{n+1} \\ T_1^{n+1} \\ T_2^{n+1} \\ T_3^{n+1} \\ T_4^{n+1} \end{bmatrix} = \begin{bmatrix} a \\ R_1^n \\ R_2^n \\ R_3^n \\ b \end{bmatrix}. \tag{9.106}$$

The first and last equations in it are just the boundary conditions. The middle three equations are (9.104). Only the three shaded bands are non-zero. This is why the system is called **tridiagonal**.

As usual, if we have Neumann boundary conditions (i.e. insulated boundaries) then we need special versions of (9.104). For the left boundary we get it by letting $j = 0$ in (9.104) and using the fact that $T_{-1}^{n+1} \equiv T_1^{n+1}$. This gives

$$\left(1+\alpha\right)T_0^{n+1} - \alpha T_1^{n+1} = R_0^{\,n}, \text{ where } R_0^{\,n} = \left(1-\alpha\right)T_0^{\,n} + \alpha T_1^{\,n} + \frac{\Delta t}{2}\left(P_0^{\,n} + P_0^{\,n+1}\right) \tag{9.107}$$

For the right boundary we let $j = J$ in (9.104) and use the fact that $T_{J+1}^{\,n+1} \equiv T_{J-1}^{\,n+1}$. This gives

$$-\alpha T_{J-1}^{\,n+1} + \left(1+\alpha\right)T_J^{\,n+1} = R_J^{\,n}, \text{ where } R_J^{\,n} = \left(1-\alpha\right)T_J^{\,n} + \alpha T_{J-1}^{\,n} + \frac{\Delta t}{2}\left(P_J^{\,n} + P_J^{\,n+1}\right) \tag{9.108}$$

Thus for Neumann boundary conditions we must solve this matrix equation:

$$\begin{bmatrix} 1+\alpha & -\alpha & 0 & 0 & 0 \\ -\frac{\alpha}{2} & 1+\alpha & -\frac{\alpha}{2} & 0 & 0 \\ 0 & -\frac{\alpha}{2} & 1+\alpha & -\frac{\alpha}{2} & 0 \\ 0 & 0 & -\frac{\alpha}{2} & 1+\alpha & -\frac{\alpha}{2} \\ 0 & 0 & 0 & -\alpha & 1+\alpha \end{bmatrix} \begin{bmatrix} T_0^{\,n+1} \\ T_1^{\,n+1} \\ T_2^{\,n+1} \\ T_3^{\,n+1} \\ T_4^{\,n+1} \end{bmatrix} = \begin{bmatrix} R_0^{\,n} \\ R_1^{\,n} \\ R_2^{\,n} \\ R_3^{\,n} \\ R_4^{\,n} \end{bmatrix}. \tag{9.109}$$

The first and last equations are the Neumann the boundary conditions, (9.107) and (9.108). The middle three equations are (9.104). Again the matrix is tridiagonal.

On the next page is Visual Basic code that solves the heat flow problem in Example 9.1 and Fig. 9.1. To load it into Excel follow the instructions given on page 194. To use it run the macro called *CrankNicolson*. To create a graph of the temperature profile select columns And B in the spreadsheet, click on the Insert tab, and choose a chart of the Scatter type. Problem Set 9.6 explores this code and makes modifications to it to solve other heat flow problems.

```
Public Sub CrankNicolson ( )
  Const TLeft = 5, TRight = 15, Alpha = 1                                          '[1]
  Const Iterations = 1, Right = 10
  Dim J As Integer, Timestep As Integer, UserResponse         'Index j runs from 0 to Right.  [2]
  Dim T(0 To Right)                                  'Various arrays:  Temperature distribution T.
  Dim A(0 To Right), B(0 To Right), C(0 To Right)              'Tridiagonal matrix's bands, A, B, C.
  Dim R(0 To Right), U(0 To Right)                             'R and U in equation: [Matrix]*U=R.

  For J = 0 To Right                  'Set up the tridiagonal matrix.  This has to be done just once.
    A(J) = -Alpha / 2                                                          'below-diagonal band.
    B(J) = 1 + Alpha                                                                 'diagonal band.
    C(J) = -Alpha / 2                                                          'above-diagonal band.
  Next J
  B(0) = 1:    C(0) = 0:    A(Right) = 0:    B(Right) = 1                  'For Dirichlet conditions.  [3]

  Timestep = 0

  For J = 0 To Right:    T(J) = 0:    Next J                       'Set up initial temperature profile.  [4]

  Cells(2, 1) = "Position": Cells(2, 2) = "Temperature"                   'Set up spreadsheet columns.
  For J = 0 To Right:   Cells(J + 3, 1) = J: Cells(J + 3, 2) = T(J): Next J
  MsgBox ("Timestep = 0, the initial temperature profile")

  Do                                                        'This is the loop.  Each loop does 1 timestep.
    For J = 1 To Right - 1                                      'Fill in the matrix R (the RHS matrix).
      R(J) = Alpha / 2 * (T(J - 1) + T(J + 1)) + (1 - Alpha) * T(J)
    Next J
    R(0) = TLeft:    R(Right) = TRight                                   'For Dirichlet conditions.  [5]

    Call TriDiagonalSolver(Right, A, B, C, R, U)

    Timestep = Timestep + 1

    For J = 0 To Right:    T(J) = U(J):    Next J              'Copy output of solver to temperature profile.

    For J = 0 To Right:   Cells(J + 3, 2) = T(J):   Next J                        'Update the spreadsheet.

    If Timestep Mod Iterations = 0 Then                         'Periodically ask the user whether to stop.
      UserResponse = MsgBox("Timestep = " & Timestep & " Continue?", vbYesNo)
    End If
  Loop Until UserResponse = vbNo
End Sub

Sub TriDiagonalSolver (J, A, B, C, R, U)
  'Tridiagonal system solver.  The input (unchanged) is J, A, B, C, R.  The output is array U.
  'Integer J, arrays A, B, C, R, U are defined in this picture:
```

$$\begin{bmatrix} b(0) & c(0) & 0 & 0 & 0 \\ a(1) & b(1) & c(1) & 0 & 0 \\ 0 & \cdots & \cdots & \cdots & 0 \\ 0 & 0 & a(J-1) & b(J-1) & c(J-1) \\ 0 & 0 & 0 & a(J) & b(J) \end{bmatrix} \begin{bmatrix} u(0) \\ u(1) \\ \cdots \\ u(J-1) \\ u(J) \end{bmatrix} = \begin{bmatrix} r(0) \\ r(1) \\ \cdots \\ r(J-1) \\ r(J) \end{bmatrix}$$

```
  Dim I, Beta, Gamma()
  ReDim Gamma(0 To J)

  Beta = B(0)
  U(0) = R(0) / Beta

  For I = 1 To J
    Gamma(I) = C(I - 1) / Beta
    Beta = B(I) - A(I) * Gamma(I)
    U(I) = (R(I) - A(I) * U(I - 1)) / Beta
  Next I

  For I = J - 1 To 0 Step -1
    U(I) = U(I) - Gamma(I + 1) * U(I + 1)
  Next I
End Sub
```