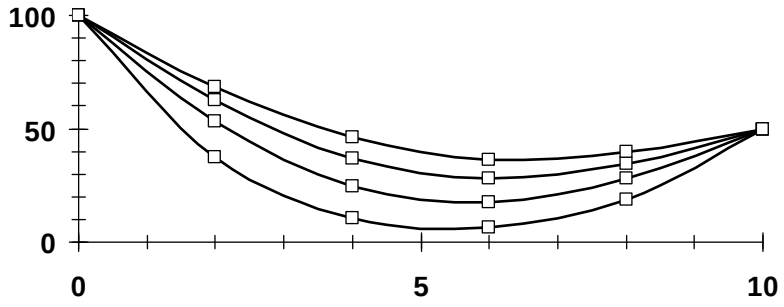
CHAPTER 30

30.1 The key to approaching this problem is to recast the PDE as a system of ODEs. Thus, by substituting the finite-difference approximation for the spatial derivative, we arrive at the following general equation for each node

$$\frac{dT_i}{dt} = k\frac{T_{i-1} - 2T_i + T_{i+1}}{\Delta x^2}$$

By writing this equation for each node, the solution reduces to solving 4 simultaneous ODEs with Heun's method. The results for the first two steps along with some later selected values are tabulated below. In addition, a plot similar to Fig. 30.4, is also shown

t	*x* = 0	*x* = 2	*x* = 4	*x* = 6	*x* = 8	*x* = 10
0	100	0	0	0	0	50
0.1	100	2.043923	0.021788	0.010894	1.021962	50
0.2	100	4.005178	0.084022	0.042672	2.002593	50
•						
•						
•						
3	100	37.54054	10.27449	6.442319	18.95732	50
6	100	53.24294	24.66052	17.4603	27.92251	50
9	100	62.39032	36.64937	27.84901	34.34692	50
12	100	68.71331	46.03498	36.54213	39.5355	50



30.2 Because we now have derivative boundary conditions, the boundary nodes must be simulated. For node 0,

$$T_0^{l+1} = T_0^l + \lambda(T_1^l - 2T_0^l + T_{-1}^l) \qquad (i)$$

This introduces an exterior node into the solution at $i = -1$. The derivative boundary condition can be used to eliminate this node,

$$\left.\frac{dT}{dx}\right|_0 = \frac{T_1 - T_{-1}}{2\Delta x}$$

which can be solved for

$$T_{-1} = T_1 - 2\Delta x\frac{dT_0}{dx}$$

which can be substituted into Eq. (*i*) to give

$$T_0^{l+1} = T_0^l + \lambda\left(2T_1^l - 2T_0^l - 2\Delta x\frac{dT_0^l}{dx}\right)$$

For our case, $dT_0/dx = 1$ and $\Delta x = 2$, and therefore $T_{-1} = T_1 - 4$. This can be substituted into Eq. (*i*) to give,

$$T_0^{l+1} = T_0^l + \lambda(2T_1^l - 2T_0^l - 4)$$

A similar analysis can be used to embed the zero derivative in the equation for the n^{th} node,

$$T_n^{l+1} = T_n^l + \lambda(T_{n+1}^l - 2T_n^l + T_{n-1}^l) \qquad (ii)$$

This introduces an exterior node into the solution at $n + 1$. The derivative boundary condition can be used to eliminate this node,

$$\left.\frac{dT}{dx}\right|_n = \frac{T_{n+1} - T_{n-1}}{2\Delta x}$$

which can be solved for

$$T_{n+1} = T_{n-1} + 2\Delta x\frac{dT_n}{dx}$$

which can be substituted into Eq. (*ii*) to give

$$T_n^{l+1} = T_n^l + \lambda\left(2T_{n-1}^l - 2T_n^l + 2\Delta x\frac{dT_n}{dx}\right)$$

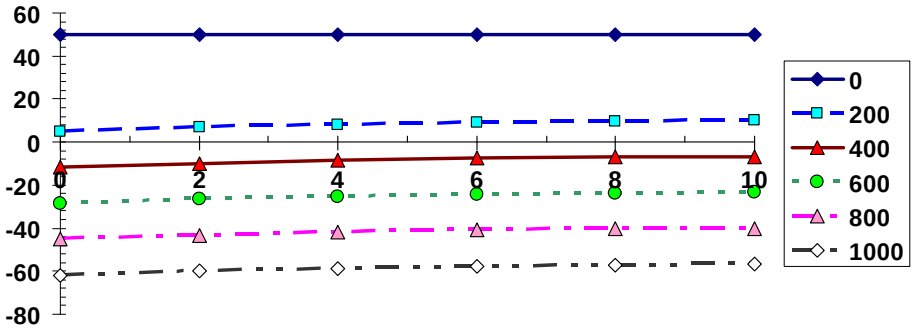
For our case, $n = 5$ and $dT_n/dx = 0$, and therefore

$$T_5^{l+1} = T_5^l + \lambda(2T_4^l - 2T_5^l)$$

Together with the equations for the interior nodes, the entire system can be solved with a step of 0.1 s. The results for some of the early steps along with some later selected values are tabulated below. In addition, a plot of the later results is also shown

t	***x* = 0**	***x* = 2**	***x* = 4**	***x* = 6**	***x* = 8**	***x* = 10**
0	50.0000	50.0000	50.0000	50.0000	50.0000	50.0000
0.1	49.9165	50.0000	50.0000	50.0000	50.0000	50.0000
0.2	49.8365	49.9983	50.0000	50.0000	50.0000	50.0000
0.3	49.7597	49.9949	50.0000	50.0000	50.0000	50.0000
0.4	49.6861	49.9901	49.9999	50.0000	50.0000	50.0000
0.5	49.6153	49.9840	49.9997	50.0000	50.0000	50.0000
•						
•						
•						
200	5.000081	6.800074	8.200059	9.200048	9.800042	10.00004

400	-11.6988	-9.89883	-8.49883	-7.49882	-6.89881	-6.69881
600	-28.4008	-26.6008	-25.2008	-24.2008	-23.6007	-23.4007
800	-45.1056	-43.3056	-41.9056	-40.9056	-40.3056	-40.1056
1000	-61.8104	-60.0104	-58.6104	-57.6104	-57.0104	-56.8104



Notice what's happening. The rod never reaches a steady state, because of the heat loss at the left end (unit gradient) and the insulated condition (zero gradient) at the right.

30.3 The solution for $\Delta t = 0.1$ is (as computed in Example 30.1),

t	*x* = 0	*x* = 2	*x* = 4	*x* = 6	*x* = 8	*x* = 10
0	100	0	0	0	0	50
0.1	100	2.0875	0	0	1.04375	50
0.2	100	4.087847	0.043577	0.021788	2.043923	50

For $\Delta t = 0.05$, it is

t	*x* = 0	*x* = 2	*x* = 4	*x* = 6	*x* = 8	*x* = 10
0	100	0.000000	0.000000	0.000000	0.000000	50
0.05	100	1.043750	0.000000	0.000000	0.521875	50
0.1	100	2.065712	0.010894	0.005447	1.032856	50
0.15	100	3.066454	0.032284	0.016228	1.533227	50
0.2	100	4.046528	0.063786	0.032229	2.023265	50

To assess the differences between the results, we performed the simulation a third time using a more accurate approach (the Heun method) with a much smaller step size ($\Delta t = 0.001$). It was assumed that this more refined approach would yield a prediction close to true solution. These values could then be used to assess the relative errors of the two Euler solutions. The results are summarized as

	x = 0	*x* = 2	*x* = 4	*x* = 6	*x* = 8	*x* = 10
Heun (h = 0.001)	100	4.006588	0.083044	0.042377	2.003302	50
Euler (h = 0.1)	100	4.087847	0.043577	0.021788	2.043923	50
Error relative to Heun		2.0%	47.5%	48.6%	2.0%	
Euler (h = 0.05)	100	4.046528	0.063786	0.032229	2.023265	50
Error relative to Heun		1.0%	23.2%	23.9%	1.0%	

Notice, that as would be expected for Euler's method, halving the step size approximately halves the global relative error.

30.4 The approach described in Example 30.2 must be modified to account for the zero derivative at the right hand node ($i = 5$). To do this, Eq. (30.8) is first written for that node as

$$-\lambda T_4^{l+1} + (1+2\lambda)T_5^{l+1} - \lambda T_6^{l+1} = T_5^l \qquad (i)$$

The value outside the system ($i = 6$) can be eliminated by writing the finite difference relationship for the derivative at node 5 as

$$\left.\frac{dT}{dx}\right|_5 = \frac{T_6 - T_4}{2\Delta x}$$

which can be solved for

$$T_6 = T_4 + 2\Delta x \left.\frac{dT}{dx}\right|_5$$

For our case, $dT/dx = 0$, so $T_6 = T_4$ and Eq. (i) becomes

$$-2\lambda T_4^{l+1} + (1+2\lambda)T_5^{l+1} = T_5^l$$

Thus, the simultaneous equations to be solved at the first step are

$$\begin{bmatrix} 1.04175 & -0.020875 & & & \\ -0.020875 & 1.04175 & -0.020875 & & \\ & -0.020875 & 1.04175 & -0.020875 & \\ & & -0.020875 & 1.04175 & -0.020875 \\ & & & -0.04175 & 1.04175 \end{bmatrix} \begin{Bmatrix} T_1^1 \\ T_2^1 \\ T_3^1 \\ T_4^1 \\ T_5^1 \end{Bmatrix} = \begin{Bmatrix} 2.0875 \\ 0 \\ 0 \\ 0 \\ 0 \end{Bmatrix}$$

which can be solved for

$$\begin{Bmatrix} 2.004645 \\ 0.040186 \\ 0.000806 \\ 1.62\times10^{-5} \\ 6.47\times10^{-7} \end{Bmatrix}$$

For the second step, the right-hand side is modified to reflect these computed values of T at $t = 0.1$,

$$\begin{bmatrix} 1.04175 & -0.020875 & & & \\ -0.020875 & 1.04175 & -0.020875 & & \\ & -0.020875 & 1.04175 & -0.020875 & \\ & & -0.020875 & 1.04175 & -0.020875 \\ & & & -0.04175 & 1.04175 \end{bmatrix} \begin{Bmatrix} T_1^1 \\ T_2^1 \\ T_3^1 \\ T_4^1 \\ T_5^1 \end{Bmatrix} = \begin{Bmatrix} 4.092145 \\ 0.040186 \\ 0.000806 \\ 1.62\times10^{-5} \\ 6.47\times10^{-7} \end{Bmatrix}$$

which can be solved for

$$\begin{Bmatrix} 3.930497 \\ 0.117399 \\ 0.003127 \\ 7.83\times10^{-5} \\ 3.76\times10^{-6} \end{Bmatrix}$$

30.5 The solution is identical to Example 30.3, but with 9 interior nodes. Thus, the simultaneous equations to be solved at the first step are

$$\begin{bmatrix} 2.167 & -0.0835 & & & & & & & \\ -0.0835 & 2.167 & -0.0835 & & & & & & \\ & -0.0835 & 2.167 & -0.0835 & & & & & \\ & & -0.0835 & 2.167 & -0.0835 & & & & \\ & & & -0.0835 & 2.167 & -0.0835 & & & \\ & & & & -0.0835 & 2.167 & -0.0835 & & \\ & & & & & -0.0835 & 2.167 & -0.0835 & \\ & & & & & & -0.0835 & 2.167 & -0.0835 \\ & & & & & & & -0.0835 & 2.167 \end{bmatrix} \begin{Bmatrix} T_1 \\ T_2 \\ T_3 \\ T_4 \\ T_5 \\ T_6 \\ T_7 \\ T_8 \\ T_9 \end{Bmatrix} = \begin{Bmatrix} 16.7 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 8.35 \end{Bmatrix}$$

which can be solved for

$$\begin{Bmatrix} 7.717983 \\ 0.297836 \\ 0.011493 \\ 0.000444 \\ 0.000026 \\ 0.000222 \\ 0.005747 \\ 0.148918 \\ 3.858992 \end{Bmatrix}$$

For the second step, the right-hand side is modified to reflect these computed values of T at $t = 0.1$,

$$\begin{bmatrix} 2.167 & -0.0835 & & & & & & & \\ -0.0835 & 2.167 & -0.0835 & & & & & & \\ & -0.0835 & 2.167 & -0.0835 & & & & & \\ & & -0.0835 & 2.167 & -0.0835 & & & & \\ & & & -0.0835 & 2.167 & -0.0835 & & & \\ & & & & -0.0835 & 2.167 & -0.0835 & & \\ & & & & & -0.0835 & 2.167 & -0.0835 & \\ & & & & & & -0.0835 & 2.167 & -0.0835 \\ & & & & & & & -0.0835 & 2.167 \end{bmatrix} \begin{Bmatrix} T_1 \\ T_2 \\ T_3 \\ T_4 \\ T_5 \\ T_6 \\ T_7 \\ T_8 \\ T_9 \end{Bmatrix} = \begin{Bmatrix} 30.8719 \\ 1.1913 \\ 0.04597 \\ 0.001775 \\ 0.000103 \\ 0.00089 \\ 0.002299 \\ 0.59567 \\ 15.436 \end{Bmatrix}$$

which can be solved for

$$\begin{Bmatrix} 14.28889 \\ 1.102814 \\ 0.063836 \\ 0.003288 \\ 0.000238 \\ 0.001650 \\ 0.031918 \\ 0.551407 \\ 7.144443 \end{Bmatrix}$$

30.6 Using the approach followed in Example 30.5, Eq. (30.20) is applied to nodes (1,1), (1,2), and (1,3) to yield the following tridiagonal equations

$$\begin{bmatrix} 2.167 & -0.0835 & \\ -0.0835 & 2.167 & -0.0835 \\ & -0.0835 & 2.167 \end{bmatrix} \begin{Bmatrix} T_{1,1} \\ T_{1,2} \\ T_{1,3} \end{Bmatrix} = \begin{Bmatrix} 5.01 \\ 5.01 \\ 15.03 \end{Bmatrix}$$

which can be solved for

$$T_{1,1} = 2.415074 \quad T_{1,2} = 2.67624 \quad T_{1,3} = 7.038978$$

In a similar fashion, tridiagonal equations can be developed and solved for

$$T_{2,1} = 0.1044726 \quad T_{2,2} = 0.2962096 \quad T_{2,3} = 4.906547$$

and

$$T_{3,1} = 2.01846 \quad T_{3,2} = 2.278894 \quad T_{3,3} = 6.827404$$

For the second step to $t = 10$, Eq. (30.22) is applied to nodes (1,1), (2,1), and (3,1) to yield

$$\begin{bmatrix} 2.167 & -0.0835 & \\ -0.0835 & 2.167 & -0.0835 \\ & -0.0835 & 2.167 \end{bmatrix} \begin{Bmatrix} T_{1,1} \\ T_{1,2} \\ T_{1,3} \end{Bmatrix} = \begin{Bmatrix} 9.660297 \\ 0.216232 \\ 8.065132 \end{Bmatrix}$$

which can be solved for

$$T_{1,1} = 4.47395 \quad T_{1,2} = 0.416205 \quad T_{1,3} = 3.737833$$

Tridiagonal equations for the other rows can be developed and solved for

$$T_{2,1} = 5.050756 \quad T_{2,2} = 0.815685 \quad T_{2,3} = 4.292810$$

and

$$T_{3,1} = 13.462935 \quad T_{3,2} = 9.820288 \quad T_{3,3} = 12.869439$$

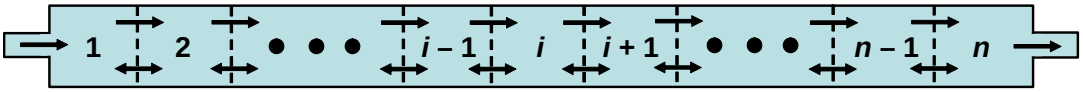
Thus, the result at the end of the first step can be summarized as

	$i = 0$	$i = 1$	$i = 2$	$i = 3$	$i = 4$
$j = 4$	90	120	120	120	85
$j = 3$	60	13.46293	9.820288	12.86944	50
$j = 2$	60	5.050756	0.815685	4.29281	50
$j = 1$	60	4.47395	0.416205	3.737833	50
$j = 0$	30	0	0	0	25

The computation can be repeated and the results for $t = 2000$ s are shown below:

	$i = 0$	$i = 1$	$i = 2$	$i = 3$	$i = 4$
$j = 4$	90	120	120	120	85
$j = 3$	60	80.71429	83.83929	77.14286	50
$j = 2$	60	59.01786	57.5	54.73214	50
$j = 1$	60	37.85714	32.41071	34.28571	50
$j = 0$	30	0	0	0	25

30.7 Although this problem can be modeled with the finite-difference approach (see Sec. 32.1), the control-volume method provides a more straightforward way to handle the boundary conditions. Thus, the tank is idealized as a series of control volumes:



The boundary fluxes and the reaction term can be used to develop the discrete form of the advection-diffusion equation for the interior volumes as

$$\Delta x\frac{dc_i^l}{dt} = -D\frac{c_i^l - c_{i-1}^l}{\Delta x} + D\frac{c_{i+1}^l - c_i^l}{\Delta x} + U\frac{c_i^l + c_{i-1}^l}{2} - U\frac{c_{i+1}^l + c_i^l}{2} - k\Delta x c_i^l$$

or dividing both sides by Δx,

$$\frac{dc_i^l}{dt} = D\frac{c_{i+1}^l - 2c_i^l + c_{i-1}^l}{\Delta x^2} - U\frac{c_{i+1}^l - c_{i-1}^l}{2\Delta x} - kc_i^l$$

which is precisely the form that would have resulted by substituting centered finite difference approximations into the advection-diffusion equation.

For the first boundary node, no diffusion is allowed up the entrance pipe and advection is handled with a backward difference,

$$\Delta x\frac{dc_1^l}{dt} = D\frac{c_2^l - c_1^l}{\Delta x} + Uc_0^l - U\frac{c_2^l + c_1^l}{2} - k\Delta x c_1^l$$

or dividing both sides by Δx,

$$\frac{dc_1^l}{dt} = D\frac{c_2^l - c_1^l}{\Delta x^2} + U\frac{2c_0^l - c_2^l - c_1^l}{2\Delta x} - kc_1^l$$

For the last boundary node, no diffusion is allowed through the exit pipe and advection out of the tank is again handled with a backward difference,

$$\Delta x\frac{dc_n^l}{dt} = D\frac{c_{n-1}^l - c_n^l}{\Delta x} + U\frac{c_n^l + c_{n-1}^l}{2} - Uc_n^l - k\Delta x c_n^l$$

or dividing both sides by Δx,

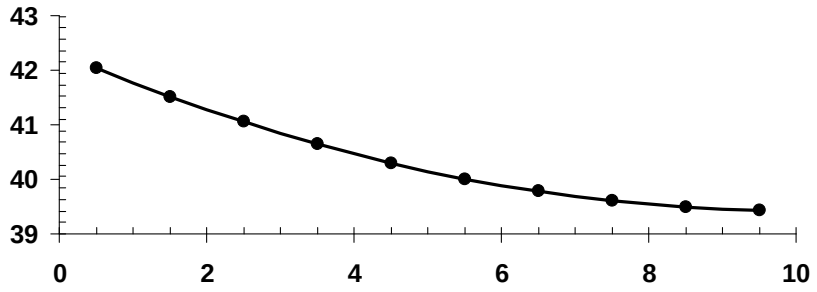
$$\frac{dc_n^l}{dt} = D\frac{c_{n-1}^l - c_n^l}{\Delta x^2} + U\frac{c_{n-1}^l - c_n^l}{2\Delta x} - kc_n^l$$

By writing these equations for each equally-spaced volume, the PDE is transformed into a system of ODEs. Explicit methods like Euler's method or other higher-order RK methods can then be used to solve the system.

The results with an initial condition that the reactor has zero concentration with an inflow concentration of 100 (using Euler with a step size of 0.005) for $t = 100$ are

x	0.5	1.5	2.5	3.5	4.5	5.5	6.5	7.5	8.5	9.5
c	42.0320	41.5128	41.0509	40.6463	40.2989	40.0087	39.7760	39.6008	39.4836	39.4248

A plot of the results is shown below:



30.8 Here is a VBA program that implements the explicit method. It is set up to duplicate Example 30.1.

```
Option Explicit
Sub Explicit()
Dim i As Integer, j As Integer, np As Integer, ns As Integer
Dim Te(20) As Double, dTe(20) As Double, tpr(20) As Double, Tepr(20, 20) As Double
Dim k As Double, dx As Double, L As Double, tc As Double, tf As Double
Dim tp As Double, t As Double, tend As Double, h As Double, tol As Double
Dim x As Double
tol = 0.000001
L = 10
ns = 5
dx = L / ns
k = 0.835
Te(0) = 100
Te(5) = 50
tc = 0.1
tf = 12
tp = 3
np = 0
tpr(np) = t
For i = 0 To ns
  Tepr(i, np) = Te(i)
Next i
Do
  tend = t + tp
  If tend > tf Then tend = tf
  h = tc
  Do
    If t + h > tend Then h = tend - t
    Call Derivs(Te, dTe, ns, dx, k)
    For j = 1 To ns - 1
      Te(j) = Te(j) + dTe(j) * h
    Next j
    t = t + h
    If t >= tend Then Exit Do
  Loop
  np = np + 1
  tpr(np) = t
  For j = 0 To ns
    Tepr(j, np) = Te(j)
  Next j
  If t + tol >= tf Then Exit Do
Loop
Sheets("sheet1").Select
Range("a4:bb5000").ClearContents
Range("a4").Select
ActiveCell.Value = "time"
x = 0
For j = 0 To ns
  ActiveCell.Offset(0, 1).Select
```

```
  ActiveCell.Value = "x = " & x
  x = x + dx
Next j
Range("a5").Select
For i = 0 To np
  ActiveCell.Value = tpr(i)
  For j = 0 To ns
    ActiveCell.Offset(0, 1).Select
    ActiveCell.Value = Tepr(j, i)
  Next j
  ActiveCell.Offset(1, -ns - 1).Select
Next i
Range("a5").Select
End Sub

Sub Derivs(Te, dTe, ns, dx, k)
Dim j As Integer
For j = 1 To ns - 1
  dTe(j) = k * (Te(j - 1) - 2 * Te(j) + Te(j + 1)) / dx ^ 2
Next j
End Sub
```

When the program is run, the result is

	A	B	C	D	E	F	G	H	I
1									
2								RUN	
3									
4	time	x = 0	x = 2	x = 4	x = 6	x = 8	x = 10		
5	0	100	0	0	0	0	50		
6	3	100	37.8557	10.24094	6.378532	19.09671	50		
7	6	100	53.44055	24.75211	17.49043	27.99545	50		
8	9	100	62.52413	36.78229	27.93798	34.40585	50		
9	12	100	68.81906	46.1739	36.65582	39.59994	50		

30.9 This VBA program is set up to either use Dirichlet or gradient boundary conditions depending on the values of the parameters *istrt* and *iend*. It is set up to solve the first few steps of Prob. 30.2.

```
Option Explicit

Sub EulerPDE()
Dim i As Integer, j As Integer, np As Integer, ns As Integer
Dim istrt As Integer, iend As Integer
Dim Te(2000) As Double, dTe(2000) As Double
Dim tpr(20000) As Double, Tepr(2000, 20000) As Double
Dim k As Double, dx As Double, L As Double, tc As Double, tf As Double
Dim tp As Double, t As Double, tend As Double, h As Double
Dim dTedx(2000) As Double, tol As Double, x As Double
tol = 0.000001
L = 10
ns = 5
dx = L / ns
k = 0.835
dTedx(0) = 1
istrt = 0
dTedx(ns) = 0
iend = ns
Te(0) = 25
Te(1) = 25
Te(2) = 25
Te(3) = 25
Te(4) = 25
Te(5) = 25
tc = 0.1
tf = 0.5
tp = 0.1
```

```
np = 0
tpr(np) = t
For i = 0 To ns
  Tepr(i, np) = Te(i)
Next i
Do
  tend = t + tp
  If tend > tf Then tend = tf
  h = tc
  Do
    If t + h > tend Then h = tend - t
    Call Derivs(Te, dTe, istrt, iend, ns, dx, k, dTedx)
    For j = istrt To iend
      Te(j) = Te(j) + dTe(j) * h
    Next j
    t = t + h
    If t >= tend Then Exit Do
  Loop
  np = np + 1
  tpr(np) = t
  For j = 0 To ns
    Tepr(j, np) = Te(j)
  Next j
  If t + tol >= tf Then Exit Do
Loop
Sheets("sheet1").Select
Range("a4:bb5000").ClearContents
Range("a4").Select
ActiveCell.Value = "time"
x = 0
For j = 0 To ns
  ActiveCell.Offset(0, 1).Select
  ActiveCell.Value = "x = " & x
  x = x + dx
Next j
Range("a5").Select
For i = 0 To np
  ActiveCell.Value = tpr(i)
  For j = 0 To ns
    ActiveCell.Offset(0, 1).Select
    ActiveCell.Value = Tepr(j, i)
  Next j
  ActiveCell.Offset(1, -ns - 1).Select
Next i
Range("a5").Select
End Sub

Sub Derivs(Te, dTe, istrt, iend, ns, dx, k, dTedx)
Dim j As Integer
If istrt = 0 Then
  dTe(0) = k * (2 * Te(1) - 2 * Te(0) - 2 * dx * dTedx(0)) / dx ^ 2
End If
For j = 1 To ns - 1
  dTe(j) = k * (Te(j - 1) - 2 * Te(j) + Te(j + 1)) / dx ^ 2
Next j
If iend = ns Then
  dTe(ns) = k * (2 * Te(ns - 1) - 2 * Te(ns) + 2 * dx * dTedx(ns)) / dx ^ 2
End If
End Sub
```

When the program is run, the result is

	A	B	C	D	E	F	G	H	I
1								RUN	
2									
3									
4	time	x = 0	x = 2	x = 4	x = 6	x = 8	x = 10		
5	0	25	25	25	25	25	25		
6	0.1	24.9165	25	25	25	25	25		
7	0.2	24.83649	24.99826	25	25	25	25		
8	0.3	24.75974	24.99492	24.99996	25	25	25		
9	0.4	24.68606	24.99011	24.99986	25	25	25		
10	0.5	24.61525	24.98397	24.99966	25	25	25		

30.10 Here is a VBA program that implements the implicit method. It is set up to duplicate Example 30.2.

```
Option Explicit

Sub Implicit()
Dim np As Integer, ns As Integer, i As Integer, j As Integer, n As Integer
Dim Te(10) As Double, dTe(10) As Double, tpr(100) As Double
Dim Tepr(10, 100) As Double, Tei As Double
Dim k As Double, dx As Double, L As Double, tc As Double, tf As Double
Dim tp As Double, t As Double, tend As Double, h As Double, lambda As Double
Dim e(10) As Double, f(10) As Double, g(10) As Double
Dim r(10) As Double, x(10) As Double, xrod As Double
Dim tol As Double
tol = 0.000001
L = 10#
ns = 5
dx = L / ns
k = 0.835
Te(0) = 100#
Te(ns) = 50#
Tei = 0
For i = 1 To ns - 1
  Te(i) = Tei
Next i
t = 0
np = 0
tpr(np) = t
For i = 0 To ns
  Tepr(i, np) = Te(i)
Next i
tc = 0.1
tp = 0.1
tf = 0.5
Do
  tend = t + tp
  If tend > tf Then tend = tf
  h = tc
  Do
    If t + h > tend Then h = tend - t
    lambda = k * h / dx ^ 2
    f(1) = 1 + 2 * lambda
    g(1) = -lambda
    r(1) = Te(1) + lambda * Te(0)
    For j = 2 To ns - 2
      e(j) = -lambda
      f(j) = 1 + 2 * lambda
      g(j) = -lambda
      r(j) = Te(j)
    Next j
    e(ns - 1) = -lambda
    f(ns - 1) = 1 + 2 * lambda
    r(ns - 1) = Te(ns - 1) + lambda * Te(ns)
    Call Tridiag(e, f, g, r, Te, ns - 1)
    t = t + h
    If t >= tend Then Exit Do
```

```
  Loop
  np = np + 1
  tpr(np) = t
  For j = 0 To ns
    Tepr(j, np) = Te(j)
  Next j
  If t + tol >= tf Then Exit Do
Loop
Sheets("sheet1").Select
Range("a4:bb5000").ClearContents
Range("a4").Select
ActiveCell.Value = "time"
xrod = 0
For j = 0 To ns
  ActiveCell.Offset(0, 1).Select
  ActiveCell.Value = "x = " & xrod
  xrod = xrod + dx
Next j
Range("a5").Select
For i = 0 To np
  ActiveCell.Value = tpr(i)
  For j = 0 To ns
    ActiveCell.Offset(0, 1).Select
    ActiveCell.Value = Tepr(j, i)
  Next j
  ActiveCell.Offset(1, -ns - 1).Select
Next i
Range("a5").Select
End Sub

Sub Tridiag(e, f, g, r, x, n)
Dim k As Integer
For k = 2 To n
  e(k) = e(k) / f(k - 1)
  f(k) = f(k) - e(k) * g(k - 1)
Next k
For k = 2 To n
  r(k) = r(k) - e(k) * r(k - 1)
Next k
x(n) = r(n) / f(n)
For k = n - 1 To 1 Step -1
  x(k) = (r(k) - g(k) * x(k + 1)) / f(k)
Next k
End Sub
```

When the program is run, the result is

	A	B	C	D	E	F	G	H	I
1									
2								RUN	
3									
4	time	x = 0	x = 2	x = 4	x = 6	x = 8	x = 10		
5	0	100	0	0	0	0	50		
6	0.1	100	2.004653	0.040589	0.020899	1.002339	50		
7	0.2	100	3.930536	0.118963	0.061827	1.965327	50		
8	0.3	100	5.781512	0.232491	0.121937	2.890926	50		
9	0.4	100	7.561235	0.378704	0.200404	3.781003	50		
10	0.5	100	9.273172	0.555286	0.296424	4.637332	50		

30.11 Here is a VBA program that implements the Crank-Nicolson method. It is set up to duplicate Example 30.3.

```
Option Explicit
Sub CrankNic()
Dim np As Integer, ns As Integer, i As Integer, j As Integer, n As Integer
Dim Te(10) As Double, dTe(10) As Double, tpr(100) As Double
```

```
Dim Tepr(10, 100) As Double, Tei As Double
Dim k As Double, dx As Double, L As Double, tc As Double, tf As Double
Dim tp As Double, t As Double, tend As Double, h As Double, lambda As Double
Dim e(10) As Double, f(10) As Double, g(10) As Double
Dim r(10) As Double, x(10) As Double, xrod As Double
Dim tol As Double
tol = 0.000001
L = 10#
ns = 5
dx = L / ns
k = 0.835
Te(0) = 100#
Te(ns) = 50#
Tei = 0
t = 0
np = 0
tpr(np) = t
For i = 0 To ns
  Tepr(i, np) = Tei
Next i
tc = 0.1
tf = 0.5
tp = 0.1
Do
  tend = t + tp
  If tend > tf Then tend = tf
  h = tc
  Do
    If t + h > tend Then h = tend - t
    lambda = k * h / dx ^ 2
    f(1) = 2 * (1 + lambda)
    g(1) = -lambda
    r(1) = lambda * Te(0) + 2 * (1 - lambda) * Te(1) + lambda * Te(2)
    r(1) = r(1) + lambda * Te(0)
    For j = 2 To ns - 2
      e(j) = -lambda
      f(j) = 2 * (1 + lambda)
      g(j) = -lambda
      r(j) = lambda * Te(j - 1) + 2 * (1 - lambda) * Te(j) + lambda * Te(j + 1)
    Next j
    e(ns - 1) = -lambda
    f(ns - 1) = 2 * (1 + lambda)
    r(ns - 1) = lambda * Te(ns - 2) + 2 * (1 - lambda) * Te(ns - 1)
    r(ns - 1) = r(ns - 1) + lambda * Te(ns) + lambda * Te(ns)
    Call Tridiag(e, f, g, r, Te, ns - 1)
    t = t + h
    If t >= tend Then Exit Do
  Loop
  np = np + 1
  tpr(np) = t
  For j = 0 To ns
    Tepr(j, np) = Te(j)
  Next j
  If t + tol >= tf Then Exit Do
Loop
Sheets("sheet1").Select
Range("a4:bb5000").ClearContents
Range("a4").Select
ActiveCell.Value = "time"
xrod = 0
For j = 0 To ns
  ActiveCell.Offset(0, 1).Select
  ActiveCell.Value = "x = " & xrod
  xrod = xrod + dx
Next j
Range("a5").Select
For i = 0 To np
  ActiveCell.Value = tpr(i)
  For j = 0 To ns
```

```
    ActiveCell.Offset(0, 1).Select
    ActiveCell.Value = Tepr(j, i)
  Next j
  ActiveCell.Offset(1, -ns - 1).Select
Next i
Range("a5").Select
End Sub

Sub Tridiag(e, f, g, r, x, n)
Dim k As Integer
For k = 2 To n
  e(k) = e(k) / f(k - 1)
  f(k) = f(k) - e(k) * g(k - 1)
Next k
For k = 2 To n
  r(k) = r(k) - e(k) * r(k - 1)
Next k
x(n) = r(n) / f(n)
For k = n - 1 To 1 Step -1
  x(k) = (r(k) - g(k) * x(k + 1)) / f(k)
Next k
End Sub
```

When the program is run, the result is

	A	B	C	D	E	F	G	H	I
1									
2								RUN	
3									
4	time	x = 0	x = 2	x = 4	x = 6	x = 8	x = 10		
5	0	0	0	0	0	0	0		
6	0.1	100	2.045029	0.021018	0.010669	1.022516	50		
7	0.2	100	4.007269	0.082578	0.042232	2.003647	50		
8	0.3	100	5.890904	0.181791	0.093808	2.945504	50		
9	0.4	100	7.699891	0.315951	0.164539	3.850092	50		
10	0.5	100	9.437972	0.482524	0.253588	4.71932	50		

30.12 Here is VBA code to solve this problem. The Excel output is also attached showing values for the first two steps along with selected snapshots of the solution as it evolves in time.

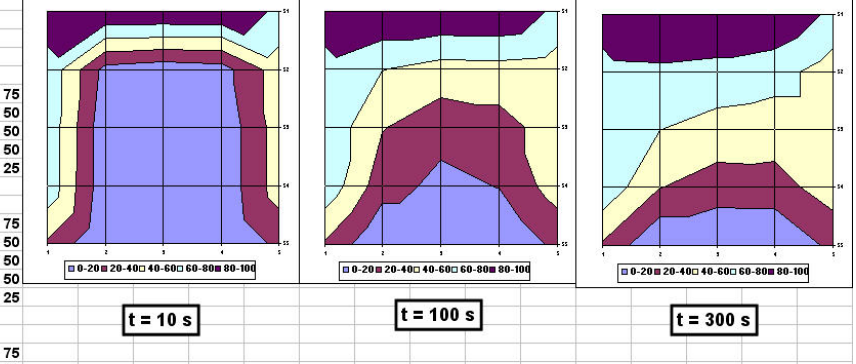
```
Option Explicit
Sub ADI()
Dim np As Integer, i As Integer, j As Integer
Dim nx As Integer, ny As Integer
Dim Lx As Double, dx As Double
Dim Ly As Double, dy As Double
Dim Te(10, 10) As Double, dTe(10, 10) As Double
Dim tpr(100) As Double, Tepr(10, 10, 100) As Double, Tei As Double
Dim k As Double
Dim dt As Double, ti As Double, tf As Double, tp As Double
Dim t As Double, tend As Double, h As Double
Dim lamx As Double, lamy As Double
Dim e(10) As Double, f(10) As Double, g(10) As Double
Dim r(10) As Double, Ted(10) As Double
'set computation parameters
Lx = 40
nx = 4
dx = Lx / nx
Ly = 40
ny = 4
dy = Ly / ny
k = 0.835
dt = 10
tf = 500
ti = 0
tp = 10
```

```
Tei = 0
'set top boundary
For i = 1 To nx - 1
  Te(i, ny) = 100
Next i
'set bottom boundary
For i = 1 To nx - 1
  Te(i, 0) = 0
Next i
'set left boundary
For j = 1 To ny - 1
  Te(0, j) = 75
Next j
'set right boundary
For j = 1 To ny - 1
  Te(nx, j) = 50
Next j
'set corners for plot
Te(0, 0) = (dy * Te(1, 0) + dx * Te(0, 1)) / (dy + dx)
Te(nx, 0) = (dy * Te(nx - 1, 0) + dx * Te(nx, 1)) / (dy + dx)
Te(0, ny) = (dy * Te(1, ny) + dx * Te(0, ny - 1)) / (dy + dx)
Te(nx, ny) = (dy * Te(nx - 1, ny) + dx * Te(nx, ny - 1)) / (dy + dx)
'set interior
For i = 1 To nx - 1
  For j = 1 To ny - 1
    Te(i, j) = Tei
  Next j
Next i
'save initial values for output
np = 0
t = ti
tpr(np) = t
For i = 0 To nx
  For j = 0 To ny
    Tepr(i, j, np) = Te(i, j)
  Next j
Next i
Do
  tend = t + tp
  If tend > tf Then tend = tf
  h = dt
  Do
    If t + h > tend Then h = tend - t
    'Sweep y
    lamx = k * h / dx ^ 2
    lamy = k * h / dy ^ 2
    For i = 1 To nx - 1
      f(1) = 2 * (1 + lamy)
      g(1) = -lamy
      r(1) = lamx * Te(i - 1, 1) + 2 * (1 - lamx) * Te(i, 1) + lamx * Te(i + 1, 1) _
            + lamy * Te(i, 0)
      For j = 2 To ny - 2
        e(j) = -lamy
        f(j) = 2 * (1 + lamy)
        g(j) = -lamy
        r(j) = lamx * Te(i - 1, j) + 2 * (1 - lamx) * Te(i, j) + lamx * Te(i + 1, j)
      Next j
      e(ny - 1) = -lamy
      f(ny - 1) = 2 * (1 + lamy)
      r(ny - 1) = lamx * Te(i - 1, ny - 1) + 2 * (1 - lamx) * Te(i, ny - 1) _
                                  + lamx * Te(i + 1, ny - 1) + lamy * Te(i, nx)
      Call Tridiag(e, f, g, r, Ted, nx - 1)
      For j = 1 To ny - 1
        Te(i, j) = Ted(j)
      Next j
    Next i
    t = t + h / 2
    'Sweep x
    For j = 1 To ny - 1
```

```
      f(1) = 2 * (1 + lamx)
      g(1) = -lamx
      r(1) = lamy * Te(1, j - 1) + 2 * (1 - lamy) * Te(1, j) + lamy * Te(1, j + 1) _
                                                              + lamx * Te(0, j)
      For i = 2 To nx - 2
        e(i) = -lamx
        f(i) = 2 * (1 + lamx)
        g(i) = -lamx
        r(i) = lamy * Te(i, j - 1) + 2 * (1 - lamy) * Te(i, j) + lamy * Te(i, j + 1)
      Next i
      e(nx - 1) = -lamx
      f(nx - 1) = 2 * (1 + lamx)
      r(nx - 1) = lamy * Te(nx - 1, j - 1) + 2 * (1 - lamy) * Te(nx - 1, j) _
                                   + lamy * Te(nx - 1, j + 1) + lamx * Te(ny, j)
      Call Tridiag(e, f, g, r, Ted, nx - 1)
      For i = 1 To nx - 1
        Te(i, j) = Ted(i)
      Next i
    Next j
    t = t + h / 2
    If t >= tend Then Exit Do
  Loop
  'save values for output
  np = np + 1
  tpr(np) = t
  For i = 0 To nx
    For j = 0 To ny
      Tepr(i, j, np) = Te(i, j)
    Next j
  Next i
  If t >= tf Then Exit Do
Loop
'output results back to sheet
Range("a5").Select
Range("a5:e2005").ClearContents
For k = 0 To np
  ActiveCell.Value = "t = " & tpr(k)
  ActiveCell.Offset(1, 0).Select
  For j = ny To 0 Step -1
    For i = 0 To nx
      ActiveCell.Value = Tepr(i, j, k)
      ActiveCell.Offset(0, 1).Select
    Next i
    ActiveCell.Offset(1, -nx - 1).Select
  Next j
  ActiveCell.Offset(1, 0).Select
Next k
Range("a5").Select
End Sub

Sub Tridiag(e, f, g, r, x, n)
Dim k As Integer
For k = 2 To n
  e(k) = e(k) / f(k - 1)
  f(k) = f(k) - e(k) * g(k - 1)
Next k
For k = 2 To n
  r(k) = r(k) - e(k) * r(k - 1)
Next k
x(n) = r(n) / f(n)
For k = n - 1 To 1 Step -1
  x(k) = (r(k) - g(k) * x(k + 1)) / f(k)
Next k
End Sub
```

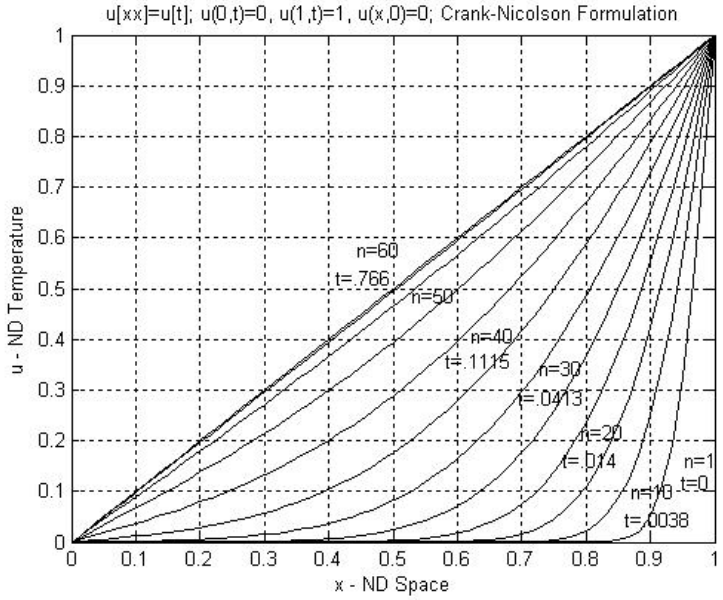
	A	B	C	D	E
1	Prob. 30.12				
2				RUN	
3					
4					
5	t = 0				
6	87.5	100	100	100	75
7	75	0	0	0	50
8	75	0	0	0	50
9	75	0	0	0	50
10	37.5	0	0	0	25
11					
12	t = 10				
13	87.5	100	100	100	75
14	75	13.112	8.3207	11.361	50
15	75	6.1683	0.8238	4.2359	50
16	75	5.58552	0.4782	3.7388	50
17	37.5	0	0	0	25
18					
19	t = 20				
20	87.5	100	100	100	75
21	75	23.467	15.889	20.334	50
22	75	11.8449	2.7523	8.3305	50
23	75	10.0556	1.4948	6.7745	50
24	37.5	0	0	0	25



30.13 MATLAB solution:

```
%PDE Parabolic Problem - Heat conduction in a rod
%   u[xx]=u[t]
%   BC u(0,t)=0   u(1,t)=1
%   IC u(x,0)=0   x<1
%   i=spatial index, from 1 to imax
%   imax = no. of x points
%   n=time index  from 1 to nmax
%   nmax = no. of time steps,
%   Crank-Nicolson Formulation
imax=61;
nmax=60;     % last time step = nmax+1
%  Constants
dx=1/(imax-1);
dx2=dx*dx;
dt=dx2;    % Setting dt to dx2 for good stability and results
%  Independent space variable
x=0:dx:1;
%  Sizing matrices
u=zeros(imax,nmax+1); t=zeros(1,nmax+1);
a=zeros(1,imax);  b=zeros(1,imax);
c=zeros(1,imax);  d=zeros(1,imax);
ba=zeros(1,imax); ga=zeros(1,imax);
up=zeros(1,imax);
%  Boundary Conditions
u(1,1)=0;
u(imax,1)=1;
%  Time step loop
%  n=1 represents 0 time, n+1 = next time step
t(1)=0;
for n=1:nmax
  t(n+1)=t(n)+dt;
  % Boundary conditions & Constants
  u(1,n+1)=0;
  u(imax,n+1)=1;
  dx2dt=dx2/dt;
  % coefficients
  b(2)=-2-2*dx2dt;
  c(2)=1;
  d(2)=(2-2*dx2dt)*u(2,n)-u(3,n);
  for i=3:imax-2
    a(i)=1;
    b(i)=-2-2*dx2dt;
    c(i)=1;
    d(i)=-u(i-1,n)+(2-2*dx2dt)*u(i,n)-u(i+1,n);
  end
  a(imax-1)=1;
```

```
  b(imax-1)=-2-2*dx2dt;
  d(imax-1)=-u(imax-2,n)+(2-2*dx2dt)*u(imax-1,n)-2;
  % Solution by Thomas Algorithm
  ba(2)=b(2);
  ga(2)=d(2)/b(2);
  for i=3:imax-1
    ba(i)=b(i)-a(i)*c(i-1)/ba(i-1);
    ga(i)=(d(i)-a(i)*ga(i-1))/ba(i);
  end
    % Back substitution step
  u(imax-1,n+1)=ga(imax-1);
  for i=imax-2:-1:2
    u(i,n+1)=ga(i)-c(i)*u(i+1,n+1)/ba(i);
  end
  dt=1.1*dt;
end
% end of time step loop
% Plot
% Storing plot value of u as up, at every 5 time steps, np=5
% j=time index
% i=space index
np=5;
for j=np:np:nmax
  for i=1:imax
    up(i)=u(i,j);
  end
  plot(x,up)
  hold on
end
grid
title('u[xx]=u[t]; u(0,t)=0, u(1,t)=1, u(x,0)=0; Crank-Nicolson Formulation')
xlabel('x - ND Space')
ylabel('u - ND Temperature')
hold off
gtext('n=60');gtext('n=50');gtext('n=40');gtext('n=30');
gtext('n=20');gtext('n=10');gtext('n=1');gtext('t=.766');
gtext('t=.1115');gtext('t=.0413');gtext('t=.014');
gtext('t=.0038');gtext('t=0')
```



30.14

$$\frac{\partial^2 u}{\partial r^2}+\frac{1}{r}\frac{\partial u}{\partial r}=\frac{\partial u}{\partial t}$$

Substituting of second order correct Crank-Nicolson analogues

$$\frac{\partial^2 u}{\partial r^2} = \frac{1}{2}\left[\frac{u_{i+1,n+1} - 2u_{i,n+1} + u_{i-1,n+1}}{\Delta r^2} + \frac{u_{i+1,n} - 2u_{i,n} + u_{i-1,n}}{\Delta r^2}\right]$$

$$\frac{\partial u}{\partial r} = \frac{1}{2}\left[\frac{u_{i+1,n+1} - u_{i-1,n+1}}{2\Delta r} + \frac{u_{i+1,n} - u_{i-1,n}}{2\Delta r}\right]$$

$$r = (i-1)\Delta r$$

$$\frac{\partial u}{\partial t} = \frac{u_{i,n+1} - u_{i,n}}{\Delta t}$$

into the governing equation give the following finite difference equations:

$$\left[1 - \frac{1}{2(i-1)}\right]u_{i-1,n+1} + \left[-2 - 2\frac{\Delta r^2}{\Delta t}\right]u_{i,n+1} + \left[1 + \frac{1}{2(i-1)}\right]u_{i+1,n+1} = \left[-1 + \frac{1}{2(i-1)}\right]u_{i-1,n} + \left[2 - 2\frac{\Delta r^2}{\Delta t}\right]u_{i,n} + \left[-1 - \frac{1}{2(i-1)}\right]u_{i+1,n}$$

For the end points:

$x = 1$ (i = R), substitute the value of $u_R = 1$ into the above FD equation
$x = 0$ ($i = 1$), set the FD analog to the first derivative = 0

$$\left[\frac{\partial u}{\partial r}\right]_{i=1} = \frac{1}{2}\left[\frac{u_{2,n+1} - u_{0,n+1}}{2\Delta r} + \frac{u_{2,n} - u_{0,n}}{2\Delta r}\right] = 0$$

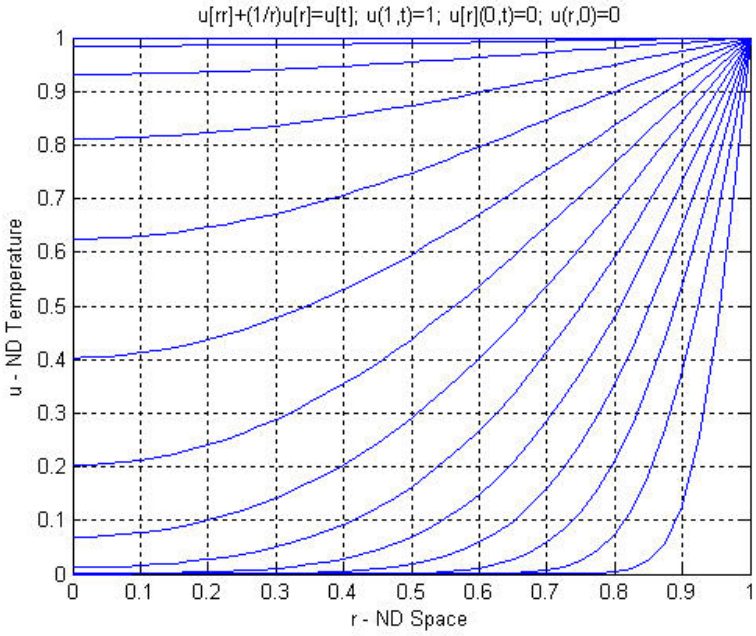
Also substitute in $i = 1$ into the finite difference equation and algebraically eliminate $u_{0,n+1} + u_{0,n}$ from the two equations and get the FD equation at $i = 1$:

$$\left[-2 - 2\frac{\Delta r^2}{\Delta t}\right]u_{1,n+1} + [2]u_{2,n+1} = -\left[2 - 2\frac{\Delta r^2}{\Delta t}\right]u_{1,n} + [-2]u_{2,n}$$

Here is a MATLAB implementation:

```
%PDE Parabolic Problem - Heat conduction in the radial direction in a circular rod
%  u[rr]+(1/r)u[r]=u[t]        0<r<1
%  BC   u(1,t)=1   u[r](0,t)=0
%  IC   u(r,0)=0          0<r<1
%  i=spatial index, from 1 to imax
%  imax = no. of r points (imax=21 for 20 dr spaces)
%  n=time index  from 1 to nmax
%  nmax = no. of time steps,
%  Crank-Nicolson Formulation
imax=41;
nmax=60;                                    % last time step = nmax+1
%  Constants
dr=1/(imax-1);
dr2=dr*dr;
dt=dr2; % Setting dt to dr2 for good stability and results
%  Independent space variable
r=0:dr:1;
```

```
%  Sizing matrices
u=zeros(imax,nmax+1); t=zeros(1,nmax+1);
a=zeros(1,imax);  b=zeros(1,imax);
c=zeros(1,imax);  d=zeros(1,imax);
ba=zeros(1,imax); ga=zeros(1,imax);
up=zeros(1,imax);
%  Boundary Conditions
u(imax,1)=1;
%  Time step loop
%  n=1 represents 0 time, new time = n+1
t(1)=0;
for n=1:nmax
  t(n+1)=t(n)+dt;
  % Boundary conditions & Constants
  u(imax,n+1)=1;
  dr2dt=dr2/dt;
  % coefficients
  b(1)=-2-2*dr2dt;
  c(1)=2;
  d(1)=(2-2*dr2dt)*u(1,n)-2*u(2,n);
  for i=2:imax-2
    a(i)=1-1/(2*(i-1));
    b(i)=-2-2*dr2dt;
    c(i)=1+1/(2*(i-1));
    d(i)=(-1+1/(2*(i-1)))*u(i-1,n)+(2-2*dr2dt)*u(i,n)+(-1-1/(2*(i-1)))*u(i+1,n);
  end
  a(imax-1)=1-1/(2*(imax-2));
  b(imax-1)=-2-2*dr2dt;
  d(imax-1)=(-1+1/(2*(imax-2)))*u(imax-2,n)+(2-2*dr2dt)*u(imax-1,n)-2*(1+1/(2*(imax-2)));
  % Solution by Thomas Algorithm
  ba(1)=b(1);
  ga(1)=d(1)/b(1);
  for i=2:imax-1
    ba(i)=b(i)-a(i)*c(i-1)/ba(i-1);
    ga(i)=(d(i)-a(i)*ga(i-1))/ba(i);
  end
  % Back substitution step
  u(imax-1,n+1)=ga(imax-1);
  for i=imax-2:-1:1
    u(i,n+1)=ga(i)-c(i)*u(i+1,n+1)/ba(i);
  end
  dt=1.1*dt;
end
% end of time step loop
% Plot
% Storing plot value of u as up, at every 5 time steps
% j=time index
% i=space index
istart=4;
for j=istart:istart:nmax+1
  for i=1:imax
    up(i)=u(i,j);
  end
  plot(r,up)
  hold on
end
grid
title('u[rr]+(1/r)u[r]=u[t]; u(1,t)=1; u[r](0,t)=0; u(r,0)=0')
xlabel('r - ND Space')
ylabel('u - ND Temperature')
hold off
```



30.15

$$\frac{\partial^2 u}{dx^2} + b\frac{\partial u}{\partial x} = \frac{\partial u}{\partial t}$$

Substituting of second order correct Crank-Nicolson analogues

$$\frac{\partial^2 u}{\partial x^2} = \frac{1}{2}\left[\frac{u_{i+1,n+1} - 2u_{i,n+1} + u_{i-1,n+1}}{\Delta x^2} + \frac{u_{i+1,n} - 2u_{i,n} + u_{i-1,n}}{\Delta x^2}\right]$$

$$\frac{\partial u}{\partial x} = \frac{1}{2}\left[\frac{u_{i+1,n+1} - u_{i-1,n+1}}{2\Delta x} + \frac{u_{i+1,n} - u_{i-1,n}}{2\Delta x}\right]$$

$$\frac{\partial u}{\partial t} = \frac{u_{i,n+1} - u_{i,n}}{\Delta t}$$

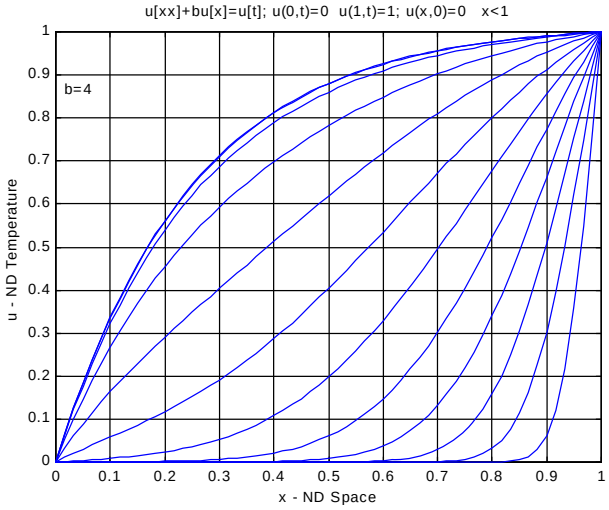
into the governing equation give the following finite difference equations

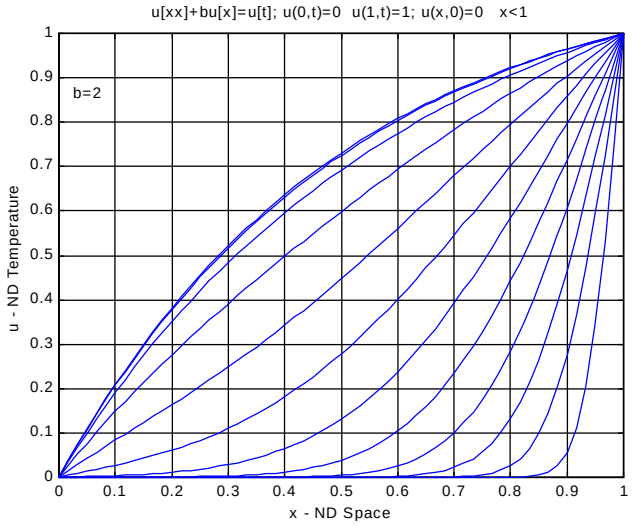
$$\left[1 - \frac{1}{2}b\Delta x\right]u_{i-1,n+1} + \left[-2 - 2\frac{\Delta x^2}{\Delta t}\right]u_{i,n+1} + \left[1 + \frac{1}{2}b\Delta x\right]u_{i+1,n+1} = \left[-1 + \frac{1}{2}b\Delta x\right]u_{i-1,n} + \left[2 - 2\frac{\Delta x^2}{\Delta t}\right]u_{i,n} + \left[-1 - \frac{1}{2}b\Delta x\right]u_{i+1,n}$$

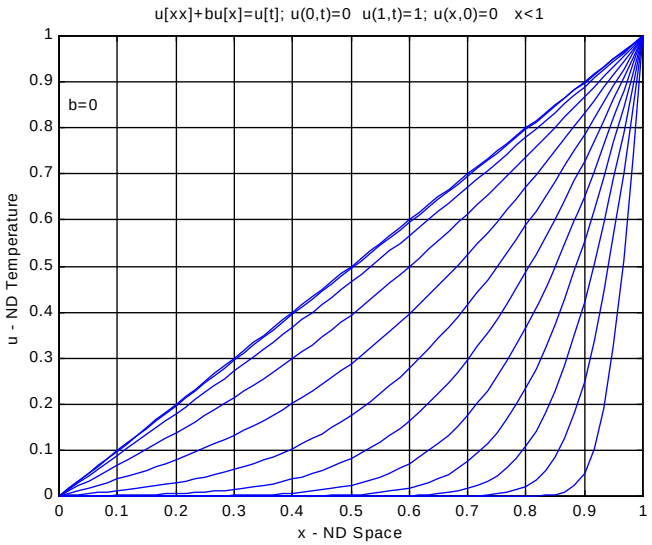
```
%PDE Parabolic Problem with a dispersion term
%  u[xx]+bu[x]=u[t]
%  BC u(0,t)=0  u(1,t)=1
%  IC      u(x,0)=0   x<1
%  i=spatial index, from 1 to imax
%  imax = no. of spatial points (imax=21 for 20 dx spaces)
%  n = time index, from 1 to nmax
%  nmax = no. of time steps
%  Crank-Nicholson formulation for the spatial derivatives
imax=61;
```

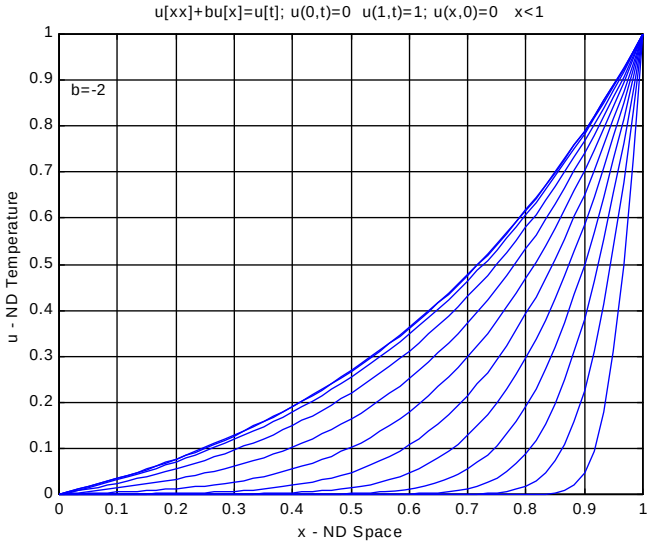
```
nmax=60;     % last time step = nmax+1
% constants
dx=1/(imax-1);
dx2=dx*dx;
dt=dx2;
% Parameters
B=-4;
% Independent spatial variable
x=0:dx:1;
% Sizing matrices
u=zeros(imax,nmax); t=zeros(1,nmax);
a=zeros(1,imax); b=zeros(1,imax);
c=zeros(1,imax); d=zeros(1,imax);
ba=zeros(1,imax); ga=zeros(1,imax);
up=zeros(1,imax);
% Boundary Conditions
u(1,1)=0;
u(imax,1)=1;
%  Time step loop
%  n=1 represents 0 time, new time = n+1
t(1)=0;
for n=1:nmax
  t(n+1)=t(n)+dt;
  %  Boundary conditions & constants
  u(1,n+1)=0;
  u(imax,n+1)=1;
  dx2dt=dx2/dt;
  % Coefficients
  b(2)=-2-2*dx2dt;
  c(2)=1+0.5*B*dx;
  d(2)=(-1-0.5*B*dx)*u(3,n)+(2-2*dx2dt)*u(2,n);
  for i=3:imax-2
    a(i)=1-0.5*B*dx;
    b(i)=-2-2*dx2dt;
    c(i)=1+0.5*B*dx;
    d(i)=(-1-0.5*B*dx)*u(i+1,n)+(2-2*dx2dt)*u(i,n)+(-1+0.5*B*dx)*u(i-1,n);
  end
  a(imax-1)=1-0.5*B*dx;
  b(imax-1)=-2-2*dx2dt;
  d(imax-1)=2*(-1-0.5*B*dx)+(2-2*dx2dt)*u(imax-1,n)+(-1+0.5*B*dx)*u(imax-2,n);
  % Solution by Thomas Algorithm
  ba(2)=b(2);
  ga(2)=d(2)/b(2);
  for i=3:imax-1
    ba(i)=b(i)-a(i)*c(i-1)/ba(i-1);
    ga(i)=(d(i)-a(i)*ga(i-1))/ba(i);
  end
  % Back substitution step
  u(imax-1,n+1)=ga(imax-1);
  for i=imax-2:-1:2
    u(i,n+1)=ga(i)-c(i)*u(i+1,n+1)/ba(i);
  end
  dt=1.1*dt;
end
% End of time step loop
%Plot
%Storing plot value of u as up, at ever 5 time steps
%  j=time index
%  i=speace index
for j=5:5:nmax
  for i=1:imax
      up(i)=u(i,j);
  end
  plot(x,up)
    hold on
end
grid
title('u[xx]+bu[x]=u[t]; u(0,t)=0  u(1,t)=1; u(x,0)=0   x<1')
xlabel('x - ND Space')
```

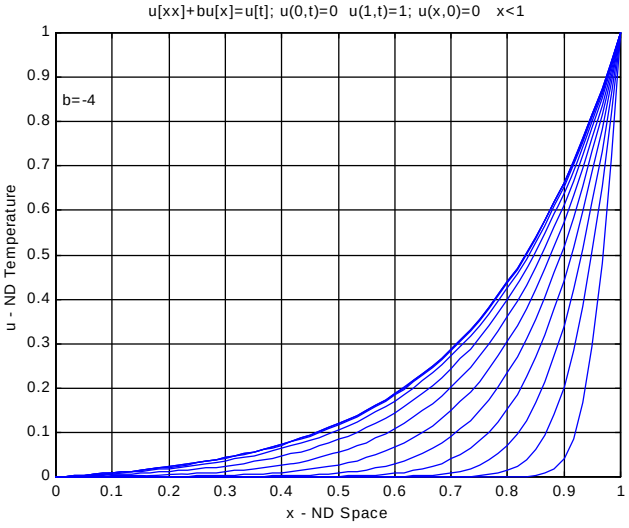
```
ylabel('u - ND Temperature')
hold off
gtext('b=-4')
```











30.16 We will solve this problem with the simple explicit method. Therefore, the interior nodes are handled in a standard fashion as

$$T_i^{l+1} = T_i^l + \lambda(T_{i+1}^l - 2T_i^l + T_{i-1}^l)$$

For the *n*th-node, the insulated condition can be developed by writing the balance as

$$T_n^{l+1} = T_n^l + \lambda(2T_{n-1}^l - 2T_n^l)$$

Finally, the convective boundary condition at the first node ($i = 0$) can be represented by first writing the general balance as

$$T_0^{l+1} = T_0^l + \lambda(T_1^l - 2T_0^l + T_{-1}^l) \qquad (i)$$

This introduces an exterior node into the solution at $i = -1$. The boundary condition can be used to eliminate this node. To do this, a finite difference representation of the condition can be written as

$$-k'\frac{T_1 - T_{-1}}{2\Delta x} = h\left(T_a - T_0\right)$$

which can be solved for

$$T_{-1} = T_1 + \frac{2\Delta x h}{k'}\left(T_a - T_0\right)$$

which can be substituted into Eq. (*i*) to give

$$T_0^{l+1} = T_0^l + \lambda\left(2T_1^l - 2T_0^l + \frac{2h\Delta x}{k'}(T_a - T_0^l)\right)$$

The entire system can be solved with a step of 1 s. A plot of the results is shown below. After sufficient time, the rod will approach a uniform temperature of 50°C.