8 Partial Differential Equations

8.1 INTRODUCTION

Different engineering disciplines solve different types of problems in their respective fields. For mechanical engineers, they may need to solve the temperature change within a solid when it is heated by the interior heat sources or due to a rise or decrease of its *boundary* temperatures. For electrical engineers, they may need to find the voltages at all circuit joints of a computer chip board. Temperature and voltage are the variables in their respective fields. Hence, they are called *field variables*. It is easy to understand that the value of the field variable is *space-dependent* and *time-dependent*. That is to say, that we are interested to know the *spatial* and *temporal* changes of the field variable. Let us denote the field variable as ϕ. and let the independent variables be x_i which could be the time t, or, the space coordinates as x, y, and z. In order not to overly complicate the discussion, we introduce the general two-dimensional partial differential equation which governs the field variable in the form of:

$$A(x_1, x_2)\frac{\partial^2\phi}{\partial x_1^2} + B(x_1, x_2)\frac{\partial^2\phi}{\partial x_1 \partial x_2} + C(x_1, x_2)\frac{\partial^2\phi}{\partial x_2^2} = F\left(x_1, x_2, \phi, \frac{\partial\phi}{\partial x_1}, \frac{\partial\phi}{\partial x_2}\right) \tag{1}$$

where the coefficient functions A, B, and C in the general cases are dependent on the variables x_1 and x_2, and the right-hand-side function F, called *forcing function* may depend not only on the independent variables x_1 and x_2 but may also depend on the first derivatives of ϕ. There are innumerable of feasible solutions for Equation 1. However, when the initial and/or boundary conditions are specified, only particular solution(s) would then be found appropriate.

In this chapter, we will discuss three simple cases when A, B, and C are all constants. The first case is a two-dimensional, *steady-state heat conduction* problem involving temperature as the field variable and only the spatial distribution of the temperature needs to be determined, Equation 1 is reduced to a *parabolic* partial differential equation named after *Poisson* and *Laplace* when the forcing function F is not equal to, or, equal to zero, respectively. This is a case when the coefficient functions in Equation 1 are related by the condition $B^2–4AC<0$.

The second case is a one-dimensional, *transient heat conduction* problem. Again, the field variable is the temperature which is changing along the longitudinal x-axis

of a straight rod and also in time. That is, x_1 becomes x and x_2 become the time t. Equation 1 is reduced to an *elliptical* partial differential equation. This is a case when $B^2–4AC = 0$.

The third case is the study of the vibration of a tightened string. The field variable is the lateral deflection of this string whose shape is changing in time. Equation 1 is reduced to a *hyperbolic* partial differential equation. If x is the longitudinal axis of the string, then same as in the second case, the two independent variables are x and t. This is a case when $B^2–4AC>0$.

The reason that these problems are called parabolic, elliptical, and hyperbolic is because their characteristic curves have such geometric features. Readers interested in exploring these features should refer to a textbook on partial differential equations.

Details will be presented regarding how the forward, backward, and central differences discussed in Chapter 4 are to be applied for approximating the first and second derivative terms appearing in Equation 1. Repetitive algorithms can be devised to facilitate programming for straight-forward computation of the spatial and temporal changes of the field variable. Numerical examples are provided to illustrate how these changes can be determined by use of either **QuickBASIC**, **FORTRAN**, **MATLAB**, or, **Mathematica** programs.

Although explanation of the procedure for numerical solution of these three types of problems is given only for the simple one- and two-dimensional cases, but its extension to the higher dimension case is straight forward. For example, one may attempt to solve the transient heat conduction problem of a thin plate by having two space variables instead of one space variable for a long rod. The steady-state heat conduction problem of a thin plate can be extended for the case of a three-dimensional solid, and the string vibration problem can be extended to a two-dimensional membrane problem.

8.2 PROGRAM PARABPDE — NUMERICAL SOLUTION OF PARABOLIC PARTIAL DIFFERENTIAL EQUATIONS

The program **ParabPDE** is designed for numerically solving engineering problems governed by parabolic partial differential equation in the form of:

$$\frac{\partial \phi}{\partial t} = a \frac{\partial^2 \phi}{\partial x^2} \tag{1}$$

and ϕ is a function of t and x and satisfies a certain set of supplementary conditions. Equation 1 is called a parabolic partial differential equation. For example, ϕ could be the temperature, T, of a longitudinal rod shown in Figure 1 and the parameter a in Equation 1 could be equal to $k/c\rho$ where k, c, and ρ are the thermal conductivity, specific heat, and specific weight of the rod, respectively. To make the problem more specific, the rod may have an initial temperature of 0°F throughout and it is completely insulated around its lateral surface and also at its right end. If its left end is to be maintained at 100°F beginning at the time t = 0, then it is of interest to know

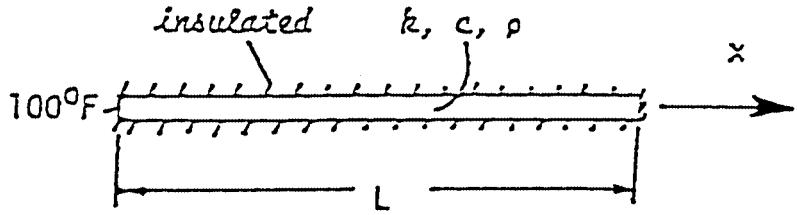


FIGURE 1. ϕ could be the temperature, T, of a longitudinal rod.

how the temperatures along the entire length of the rod will be changing as the time progresses. This is therefore a transient heat conduction problem. One would like to know how long would it take to have the entire rod reaching a uniform temperature of 100°F.

If the rod is made of a single material, k/cρ would then be equal to a constant. Analytical solution can be found for this simple case.[1] For the general case that the rod may be composed of a number of different materials and the physical properties k, c, and ρ would not only depend on the spatial variable x but may also depend on the temporal variable t. The more complicated the variation of these properties in x and t, the more likely no analytical solution is possible and the problem can only be solved numerically. The finite-difference approximation of Equation 1 can be achieved by applying the forward difference for the first derivative with respective to t and central difference for the second derivative with respect to x as follows (for t at t_i and x at x_j):

$$\frac{\partial T}{\partial t} \doteq \frac{T_{i+1,j} - T_{i,j}}{\Delta t} \quad \text{and} \quad \frac{\partial^2 T}{\partial x^2} \doteq \frac{T_{i,j-1} - 2T_{i,j} + T_{i,j+1}}{(\Delta x)^2}$$

If k/c ρ is changing in time and also changing from one location to another, we could designate it as $a_{i,j}$. As a consequence, Equation 1 can then be written as:

$$\frac{T_{i+1,j} - T_{i,j}}{\Delta t} = a_{i,j} \frac{T_{i,j-1} - 2T_{i,j} + T_{i,j+1}}{(\Delta x)^2} \tag{2}$$

Since the initial temperature distribution T is known, the above expression suggests that for a numerical solution we may select an appropriate increment in t, Δt, and the temperature be determined at a finite number of stations, N. It is advisable to have these stations be equally spaced so that the increment Δx is equal to L/(N–1) where L is the length of the rod, and the instants are to be designated as $t_1 = 0$, $t_2 = t,\ldots$, $t_i = (i-1)\Delta t$, and the stations as $x_1 = 0$, $x_2 = \Delta x,\ldots$, $x_j = (j-1)\Delta x,\ldots$, and $x_N = (N-1)\Delta x = L$. The task at hand is then to find $T(t_i,x_j)$ for $i = 1,2,\ldots$ and $j = 1,2,\ldots,N$. It can be noticed from Equation 2 that the there is only one temperature at t_{i+1} and can be expressed in terms of those at the preceding instant t_i as:

$$T_{i+1,j} = T_{i,j} + \frac{a_{i,j}\Delta t}{(\Delta x)^2}\left(T_{i,j-1} - 2T_{i,j} + T_{i,j+1}\right) \tag{3}$$

Equation 3 is to be used for j = 2 through j = N–1. For the last station, j = N, which is insulated,the temperatures on both side of this station can be assumed to be equal (the station N + 1 is a fictitious one!). The modified equation for this particular station is:

$$T_{i+1,N} = T_{i,N} + \frac{2a_{i,N}\Delta t}{(\Delta x)^2}\left(T_{i,N-1} - T_{i,N}\right) \tag{4}$$

For generating the temperatures of the rod at N stations for any specified time increment Δt until the temperatures are almost all equal to 100°F throughout, the program ParabPDE has been applied. It is listed below along with a typical printout of the results.

FORTRAN Version

```
C  Program ParabPDE - Parabolic Partial Differential Equation
C                     solved numerically using a transient
C                     heat-conduction problem.
C  For a simple illustrat'n, let kdt/c(rho)(dx)**2=1 AND N=11.
      DIMENSION T(11),TN(11)
      DATA T/100.,10*0./,TN(1)/100./
      DATA RK,C,RHO,DT,DX,N/0.037,0.212,168.,1.,.1,11/
      C1=RK/C/RHO*DT/(DX)**2
      NM1=N-1
      WRITE (*,1)
    1 FORMAT(' At what temperature differential (for all',
     *       ' stations in degree F should'
     *      /'   the computation be terminated?')
      READ (*,*) TDF
      WRITE (*,2)
    2 FORMAT(3X,'X1',5X,'X2',5X,'X3',5X,'X4',5X,'X5',5X,'X6',
     *       5X,'X7',5X,'X8',5X,'X9',4X,'X10',4X,'X11')
      WRITE (*,5) (T(IP),IP=1,N)
    5 FORMAT(11F7.1)
      TM=0.
    8 DO 15 J=2,N
   15 TN(J)=T(J)+C1*(T(J-1)-2*T(J)+T(J+1))
      TN(N)=T(N)+2*C1*(T(N-1)-T(N))
      WRITE (*,5) (TN(IP),IP=1,N)
      TM=TM+DT
      DO 25 J=2,N
      IF (ABS(TN(J)-T(J)).GT.TDF) GOTO 30
   25 CONTINUE
      WRITE (*,27) TM
   27 FORMAT(' It takes ',E12.5,' seconds.')
      GOTO 99
   30 DO 35 J=2,N
   35 T(J)=TN(J)
      GOTO 8
   99 END
```

Sample Output

```
 At what temperature differential (for all stations in degree F should
   the computation be terminated?
1
   X1      X2      X3      X4      X5      X6      X7      X8      X9      X10     X11
  100.0    65.7    37.5    18.5     7.8     2.8      .9      .2      .1      .0      .0
 It takes  .24000E+02 seconds.

 At what temperature differential (for all stations in degree F should
   the computation be terminated?
0.5
   X1      X2      X3      X4      X5      X6      X7      X8      X9      X10     X11
  100.0    75.5    53.2    34.9    21.3    12.0     6.2     3.0     1.4      .6      .4
 It takes  .50000E+02 seconds.

 At what temperature differential (for all stations in degree F should
   the computation be terminated?
0.1
   X1      X2      X3      X4      X5      X6      X7      X8      X9      X10     X11
  100.0    93.9    88.0    82.3    77.1    72.5    68.5    65.3    63.0    61.6    61.1
 It takes  .46400E+03 seconds.

 At what temperature differential (for all stations in degree F should
   the computation be terminated?
0.05
   X1      X2      X3      X4      X5      X6      X7      X8      X9      X10     X11
  100.0    97.0    94.0    91.2    88.5    86.2    84.2    82.6    81.5    80.8    80.5
 It takes  .73500E+03 seconds.

 At what temperature differential (for all stations in degree F should
   the computation be terminated?
0.01
   X1      X2      X3      X4      X5      X6      X7      X8      X9      X10     X11
  100.0    99.4    98.8    98.2    97.7    97.2    96.9    96.5    96.3    96.2    96.1
 It takes  .13650E+04 seconds.

 At what temperature differential (for all stations in degree F should
   the computation be terminated?
0.005
   X1      X2      X3      X4      X5      X6      X7      X8      X9      X10     X11
  100.0    99.7    99.4    99.1    98.9    98.6    98.4    98.3    98.1    98.1    98.1
 It takes  .16360E+04 seconds.

 At what temperature differential (for all stations in degree F should
   the computation be terminated?
0.001
   X1      X2      X3      X4      X5      X6      X7      X8      X9      X10     X11
  100.0    99.9    99.9    99.8    99.8    99.7    99.7    99.7    99.6    99.6    99.6
 It takes  .22640E+04 seconds.

 At what temperature differential (for all stations in degree F should
   the computation be terminated?
0.0005
   X1      X2      X3      X4      X5      X6      X7      X8      X9      X10     X11
  100.0    99.9    99.9    99.9    99.9    99.8    99.8    99.8    99.8    99.8    99.8
 It takes  .25370E+04 seconds.

 At what temperature differential (for all stations in degree F should
   the computation be terminated?
0.0001
   X1      X2      X3      X4      X5      X6      X7      X8      X9      X10     X11
  100.0   100.0   100.0   100.0   100.0   100.0   100.0   100.0   100.0   100.0   100.0
 It takes  .31570E+04 seconds.
```

QuickBASIC Version

```
'  Program PDEParab - Parabolic Partial Differential Equation solved numerically
'                     using a transient heat-conduction problem.
'  For a simple illustrat'n, let kdt/c(rho)(dx)**2=1 AND N=11.
     DIM T(11), TN(11)
     T(1)=100!
     FOR I=2 TO 11
         T(I)=0!
         NEXT I
     TN(1)=100!
     READ RK,C,RHO,DT,DX,N
     DATA 0.037,0.212,168.,1.,.1,11
     C1 = RK / C / RHO * DT / (DX) ^ 2
     NM1 = N - 1
     PRINT " At what temperature differential (for all";
     PRINT " stations)', in degree F should"
     INPUT "   the computation be terminated? ", TDF
     PRINT SPC(3);"X1";SPC(5);"X2";SPC(5);"X3";SPC(5);"X4";
     PRINT SPC(5);"X5";SPC(5);"X6";SPC(5);"X7";SPC(5);"X8";
     PRINT SPC(5);"X9";SPC(4);"X10";SPC(4);"X11"
     FOR ip=1 TO N
         PRINT USING "#####.#";T(ip);
         NEXT ip
     PRINT
     TM = 0!
 8 FOR J = 2 TO N - 1
       TN(J) = T(J) + C1*(T(J-1) - 2*T(J)+T(J+1))
       NEXT J
   TN(N) = T(N) + 2 * C1 * (T(N - 1) - T(N))
   FOR ip=1 TO N
       PRINT USING "#####.#"; TN(ip);
       NEXT ip
   PRINT
   TM    = TM + DT
   FOR J = 2 TO N
       IF (ABS(TN(J) - T(J)) > TDF) THEN 30
25     NEXT J
       PRINT USING " It takes ##.#####^^^^ seconds."; TM
       GOTO 99
30 FOR J    = 2 TO N
       T(J) = TN(J)
       NEXT J
       GOTO 8
99 END
```

MATLAB Applications

A **MATLAB** version of **ParabPDE** can be created easily by converting the **QuickBASIC** program. The m file may be arranged as follows:

```
          function T=ParabPDE(N)
% Parabolic Partial Differential Eq. solved numerically using a transient heat-conduction
%   problem.  For a simple illustration, let kdt/c(rho)(dx)**2=1 AND N=11.
  T(1) =100; for I=2:N, T(I)=0; end
  TN(1)=100; Rk=0.037; C=0.212; Rho=168; Dt=1; Dx=0.1;
  C1   = Rk/C/Rho*Dt/Dx^2; NM1=N-1;
  fprintf('At what temperature differential (for all stations), in degree F')
  TDF=input('  should the computation be terminated? ');
  TM = 0; ExitFlag=0;
  while ExitFlag==0; Jump=0;
        for J=2:N-1
            TN(J)=T(J)+C1*(T(J-1)-2*T(J)+T(J+1));
        end
        TN(N)=T(N)+2*C1*(T(N-1)-T(N));
        TM=TM+Dt;
```

```
        for J=2:N
            if abs(TN(J)-T(J))>TDF, Jump=1; break
            end
        end
        if Jump==0, fprintf('It takes %12.5e seconds.\n',TM), ExitFlag=1; break
           else T=TN;
        end
  end
```

For solving the sample transient temperature problem, this m file can be called and interactive **MATLAB** instructions can be entered through keyboard to obtain the temperature distribution of the rod at various times:

```
>> format compact, T=feval('a:ParabPDE',11)
 At what temperature differential (for all stations), in degree F
    should the computation be terminated? 1
 It takes 2.40000e+001 seconds.
 T =
   Columns 1 through 7
   100.0000   65.0005   36.5231   17.5928    7.2408    2.5459    0.7659
   Columns 8 through 11
    0.1976    0.0438    0.0085    0.0027
>> Tsave(:,1)=T';
>> T=feval('a:ParabPDE',11)
 At what temperature differential (for all stations), in degree F
    should the computation be terminated? 0.1
 It takes 4.62000e+002 seconds.
 T =
   Columns 1 through 7
   100.0000   93.8964   87.9431   82.2870   77.0671   72.4122   68.4367

   Columns 8 through 11
    65.2385   62.8962   61.4673   60.9871
>> Tsave(:,2)=T';
>> T=feval('a:ParabPDE',11)
 At what temperature differential (for all stations), in degree F
    should the computation be terminated? 0.01
 It takes 1.36100e+003 seconds.
 T =
   Columns 1 through 7
   100.0000   99.3897   98.7944   98.2288   97.7069   97.2413   96.8438
   Columns 8 through 11
    96.5239   96.2896   96.1467   96.0987
>> Tsave(:,3)=T';
>> T=feval('a:ParabPDE',11)
 At what temperature differential (for all stations), in degree F
    should the computation be terminated? 0.001
 It takes 2.26000e+003 seconds.
 T =
   Columns 1 through 7
   100.0000   99.9390   99.8794   99.8229   99.7707   99.7241   99.6844
   Columns 8 through 11
    99.6524   99.6290   99.6147   99.6099
>> Tsave(:,4)=T';
>> T=feval('a:ParabPDE',11)
 At what temperature differential (for all stations), in degree F
    should the computation be terminated? 0.0001
 It takes 3.15900e+003 seconds.
 T =
   Columns 1 through 7
   100.0000   99.9939   99.9879   99.9823   99.9771   99.9724   99.9684
   Columns 8 through 11
    99.9652   99.9629   99.9615   99.9610
>> plot(Tsave), xlabel('Station Number'), ylabel('Temperature, in Fahrenheits')
>> text(3.5,30,'t=24 sec  '), text(7.5,60,'t=462 sec ')
>> text(4  ,95,'t=1361 sec'), text(9  ,96,'t>2260 sec')
```

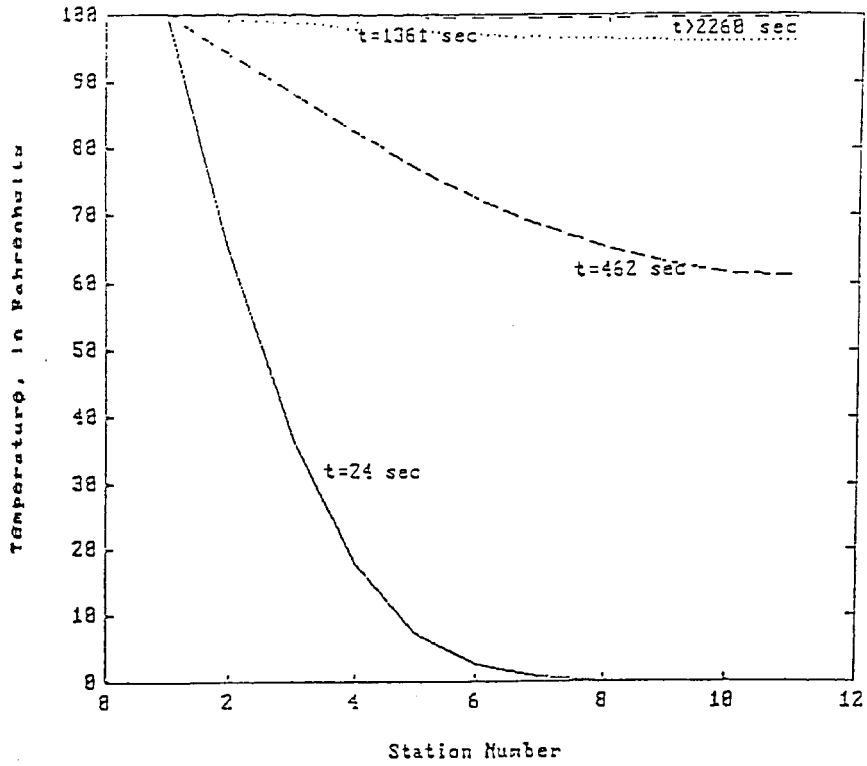


FIGURE 2. A composite graph with axes labels and markings of the curves by making use of the **MATLAB** commands **xlabel**, **ylabel**, and **text**.

Notice that results of temperature distributions which have been terminated using five differentials of 1, 0.1, 0.01, 0.001, and 0.0001 Fahrenheit are saved in Tsave and then later plotted. Since T is a row matrix, the transpose of T, T′, is stored in an appropriate column of Tsave. If the temperatures were rounded, the rod reaches a uniform temperature distribution of 100°F when the required temperature differential is selected to be 0.001°F. If the fifth curve for the temperature differential equal to 0.0001 is plotted, it will be too close to the forth curve and also the plot function provides only four line types (solid, broken, dot, and center lines), the fifth set of results is therefore not saved in Tsave. Figure 2 shows a composite graph with axes labels and markings of the curves by making use of the MATLAB commands xlabel, ylabel, and text.

MATHEMATICA APPLICATIONS

The heat conduction problem of an insulated rod previously discussed in the versions for **FORTRAN**, **QuickBASIC**, and **MATLAB** can be solved by application of **Mathematica** as follows:

```
In[1]: = (n = 11; rk = 0.037; c = 0.212; rho = 168; dt = 1; dx = 0.1;
                        c1 = rk/c/rho*dt/dx^2; nm1 = n–1;)

In[2]: = (t = Table[0,{n}]; tn = Table[0,{n}]; t[[1]] = 100; tn[[1]] = 100; tdf = 1;
      tm = 0; flag1 = 0;

In[3]: = (While[flag1 = = 0, Do[tn[[j]] = t[[j]] + c1*(t[[j–1]]–2*t[[j]] + t[[j + 1]]),
                        {j,2,nm1}];
        tn[[n]] = t[[n]] + 2*c1*(t[[nm1]]-t[[n]]);tm = tm + dt;flag1 = 1;
          Do[If[Abs[tn[[I]]-t[[I]]]>tdf, flag1 = 0; Break,
                        Continue],{i,n}];
          Do[t[[I]] = tn[[I]],{i,n}]]; Print["t = ",tm]; Print[N[tn,3]])

Out[3] = t = 24
   {100., 65.7, 37.5, 18.5, 7.83, 2.85, 0.892, 0.241, 0.0561, 0.0116, 0.00394}
```

The temperature distribution of the heated rod after 24 seconds is same as obtained by the **FORTRAN**, **QuickBASIC**, and **MATLAB** versions when every component of two consecutive temperature distribution, kept as {t} and {tn}, differ no more than the allowed set value of tdf = 1 degree in *In[2]*. Any component has a difference exceeding the tdf value will cause **flag1** to change from a value of 1 to 0 and the **Break** command in the second Do loop in *In[2]* to exit and to continue the iteration. **flag1** is created to control the **While** command which determines when the iteration should be terminated. The N[tn,3] instructs the components of {tn} be printed with 3 significant figures.

When tdf is changed to a value of 0.5, **Mathematica** can again be applied to yield

```
In[4]: = t = (Table[0,{n}]; tn = Table[0,{n}]; t[[1]] = 100; tn[[1]] = 100;
      tdf = 0.5; tm = 0; flag1 = 0;)

In[5]: = (While[flag1 = = 0,Do[tn[[j]] = t[[j]] + c1*(t[[j–1]]–2*t[[j]] + t[[j + 1]]),
                        {j,2,nm1}];
        tn[[n]] = t[[n]] + 2*c1*(t[[nm1]]-t[[n]]);tm = tm + dt;flag1 = 1;
        Do[If[Abs[tn[[I]]-t[[I]]]>tdf, flag1 = 0; Break,
            Continue],{i,n}];
        Do[t[[I]] = tn[[I]],{i,n}]]; Print["t = ",tm]; Print[N[tn,3]])

Out[5]: = t = 49
   {100., 75.5, 53.2, 34.9, 21.3, 12., 6.24, 3., 1.35, 0.617, 0.413}
```

Notice that **Mathematica** takes only 49 seconds, one second less than that required for the **FORTRAN**, **QuickBASIC**, and **MATLAB** versions. The reason is that **Mathematica** keeps more significant digits in carrying out all computations. To show more on the effect of changing the tdf value, the following **Mathematica** runs are provided:

```
In[6]: = t = (Table[0,{n}]; tn = Table[0,{n}]; t[[1]] = 100; tn[[1]] = 100;
          tdf = 0.1; tm = 0; flag1 = 0;)

In[7]: = (While[flag1 == 0, Do[tn[[j]] = t[[j]] + c1*(t[[j–1]]–2*t[[j]] + t[[j + 1]]),
                                {j,2,nm1}];
                            tn[[n]] = t[[n]] + 2*c1*(t[[nm1]]-t[[n]]);
                            tm = tm + dt; flag1 = 1;
                            Do[If[Abs[tn[[I]]-t[[I]]]>tdf, flag1 = 0; Break,
                                                    Continue],{i,n}];
                            Do[t[[I]] = tn[[I]],{i,n}]]; Print["t = ",tm];
                                Print[N[tn,3]])

Out[7]: = t = 462
    {100., 93.9, 88., 82.3, 77.1, 72.5, 68.5, 65.3, 63., 61.6, 61.1}

In[8]: = t = (Table[0,{n}]; tn = Table[0,{n}]; t[[1]] = 100; tn[[1]] = 100;
          tdf = 0.05; tm = 0; flag1 = 0;)

In[9]: = (While[flag1 == 0, Do[tn[[j]] = t[[j]] + c1*(t[[j–1]]–2*t[[j]] + t[[j + 1]]),
                              {j,2,nm1}];
                            tn[[n]] = t[[n]] + 2*c1*(t[[nm1]]-t[[n]]);
                            tm = tm + dt; flag1 = 1;
                            Do[If[Abs[tn[[I]]-t[[I]]]>tdf, flag1 = 0; Break,
                                                    Continue],{i,n}];
                            Do[t[[I]] = tn[[I]],{i,n}]]; Print["t = ",tm];
                                Print[N[tn,3]])

Out[9]: = t = 732
    {100., 97., 94., 91.2, 88.5, 86.2, 84.2, 82.6, 81.5, 80.8, 80.5}

In[10]: = t = (Table[0,{n}]; tn = Table[0,{n}]; t[[1]] = 100; tn[[1]] = 100;
          tdf = 0.0001; tm = 0; flag1 = 0;

In[11]: = (While[flag1 == 0, Do[tn[[j]] = t[[j]] + c1*(t[[j–1]]–2*t[[j]] + t[[j + 1]]),
                                {j,2,nm1}];
                              tn[[n]] = t[[n]] + 2*c1*(t[[nm1]]-t[[n]]);
                              tm = tm + dt; flag1 = 1;
                              Do[If[Abs[tn[[I]]-t[[I]]]>tdf, flag1 = 0; Break,
                                                      Continue],{i,n}];
                  Do[t[[I]] = tn[[I]],{i,n}]]; Print["t = ",tm]; Print[N[tn,3]])

Out[11]: = t = 3159
    {100., 100., 100., 100., 100., 100., 100., 100., 100., 100., 100.}
```

For tdf = 0.1 and tdf = 0.05, **Mathematica** continues to take lesser time than the other version; but when tdf = 0.0001, **Mathematica** needs two additional seconds. The

reason is that seven significant figures are required in the last case, rounding may have resulted in earlier termination of the iteration when the **FORTRAN**, Quick-**BASIC**, and **MATLAB** versions are employed.

8.3 PROGRAM RELAXATN — SOLVING ELLIPTICAL PARTIAL DIFFERENTIAL EQUATIONS BY RELAXATION METHOD

The program **Relaxatn** is designed for solving engineering problems which are governed by elliptical partial differential equation of the form:

$$\frac{\partial^2 \phi}{\partial x^2} + \frac{\partial^2 \phi}{\partial y^2} = F(x, y) \tag{1}$$

where ϕ is called the field function and F(x,y) is called forcing function. When the steady-state heat conduction of a two-dimensional domain is considered,[2] then the field function becomes the temperature distribution, T(x,y), and the forcing function becomes the heat-source function, Q(x,y). If the distribution of T is influenced only by the temperatures at the boundary of the domain, then Q(x,y) = 0 and Equation 1 which is often called a Poisson equation is reduced to a Laplace equation:

$$\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} = 0 \tag{2}$$

The second-order, central-difference formulas (read the program **DiffTabl**) can be applied to approximate the second derivatives in the above equation at an arbitrary point $x = x_i$ and $y = y_j$ in the domain as:

$$\left.\frac{\partial^2 T}{\partial x^2}\right|_{\text{at } x_i, y_j} \doteq \frac{T_{i-1,j} - 2T_{i,j} + T_{i+1,j}}{(\Delta x)^2}$$

and

$$\left.\frac{\partial^2 T}{\partial y^2}\right|_{\text{at } x_i, y_j} \doteq \frac{T_{i,j-1} - 2T_{i,j} + T_{i,j+1}}{(\Delta y)^2}$$

By substituting both of the above equations into Equation 2 and taking equal increments in both x- and y-directions, the reduced equation is, for $\Delta x = \Delta y$,

$$T_{i,j} = \frac{1}{4}\left(T_{i-1,j} + T_{i+1,j} + T_{i,j-1} + T_{i,j+1}\right) \tag{3}$$

The result is expected when the temperature distribution reaches a steady state because it states that the temperature at any point should be equal to the average of its surrounding temperatures.

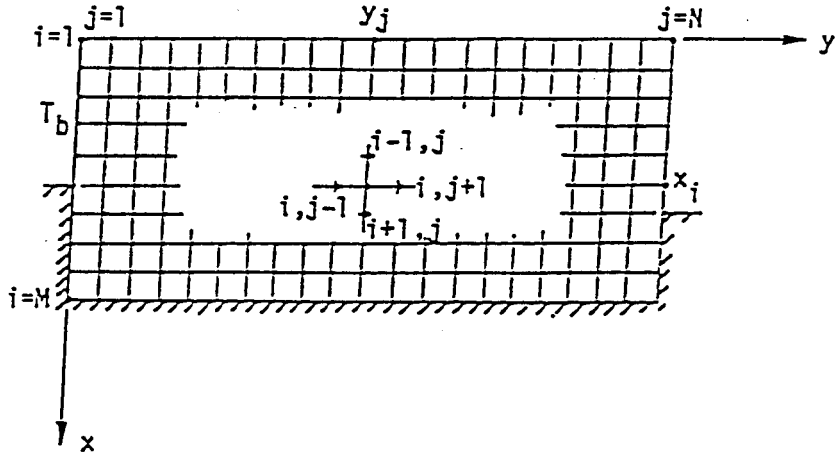


FIGURE 3. A plate, which initially (at time t = 0) had a temperature equal to 0°F throughout, insulated along a portion of its boundary and suddenly heated at its upper left boundary to maintain a linearly varying temperature T_b.

Before we proceed further, it is appropriate at this time to introduce a numerical case. Suppose that a plate which initially (at time t = 0) has a temperature equal to 0°F throughout and is insulated along a portion of its boundary, is suddenly heated at its upper left boundary to maintain a linearly varying temperature T_b as shown in Figure 3. If this heating process is to be maintained, we are then interested in knowing the temperature distribution, changed from its initial state of uniformly equal to 0°F if given sufficient time to allow it to reach an equilibrium (steady) state. Numerically, we intend to calculate the temperatures, denoted as a matrix [T], at a selected number of locations. Therefore, the plate is first divided into a gridwork of M rows and N columns along the x- and y-directions, respectively, as indicated in Figure 1. The directions of x- and y-axes are so selected for the convenience of associating them with the row and column of the temperature matrix [T] which is of order M by N. The values of M and N should be so decided such that the increments Δx and Δy are equal in order to apply Equation 3. To be more specific, let M = 10 and N = 20 and the linear temperature variation along the upper left boundary T_b be:

$$T_{i,1} = 10(i-1) \qquad \text{for} \qquad i = 1,2,\ldots,6 \tag{4}$$

Here, $T_{i,j}$ is to be understood as the temperature at the location (x_i,y_j). Equation 4 describes the temperature along the left boundary $y = y_1$ but only for x_i in the range of i = 1 to i = 6.

Since the temperatures at the stations which are on the insulated boundaries of the plate are also involved, these unknown temperatures need to be treated differently. By an insulated boundary, it means that there is no heat transfer normal to the boundary. Since the heat flow is depended on the temperature difference across that

insulated boundary, mathematically it requires that $\partial T/\partial n = 0$ there, n being the normal direction. At the vertical boundaries $x = x_M$, we have $\partial T/\partial n = \partial T/\partial x = 0$ since x is the normal direction. Based on the central difference and considering two increments in the x direction, at a y_jth station we can have:

$$\left.\frac{\partial T}{\partial x}\right|_{\text{at } (x_M, y_j)} \doteq \frac{T_{M+1,j} - T_{M-1,j}}{2(\Delta x)} = 0$$

or

$$T_{M+1,j} = T_{M-1,j} \tag{5}$$

Since x_{M+1} is below the bottom boundary of the plate shown in Figure 3, there is no need to calculate the temperatures there, however Equation 5 enables the temperatures at the stations along the bottom boundary of the plate $T_{M,j}$ for j = 1 to N to be averaged. Returning to Equation 4, we notice that if Equation 5 is substituted into it, the resulting equation which relates only to *three* neighboring temperatures is:

$$T_{M,j} = \frac{1}{4}\left(2T_{M-1,j} + T_{M,j-1} + T_{M,j+1}\right) \quad \text{for} \quad j = 2,3,\ldots,N-1 \tag{6}$$

Notice that j = 1 and j = N are not covered in Equation 6. These two cases concerning the insulated stations at the left and right bottom corners of the plate will be discussed after we address the two vertical, insulated boundaries $y = y_1$ and $y = y_N$.

For the boundaries $y = y_1$, $\partial T/\partial n$ becomes $\partial T/\partial y$. Again, we can apply the central difference for double y increments to obtain:

$$\left.\frac{\partial T}{\partial y}\right|_{\text{at } (x_1, y_1)} \doteq \frac{T_{i,0} - T_{i,2}}{2(\Delta y)} = 0$$

or

$$T_{i,0} = T_{i,2} \tag{7}$$

Thus, the modified Equation 3 for the left insulated boundary is:

$$T_{i,1} = \frac{1}{4}\left(T_{i-1,1} + T_{i+1,1} + 2T_{i,2}\right) \quad \text{for} \quad i = 7,8,9 \tag{8}$$

In a similar manner, we can derive for the right insulated boundary $y = y_N$

$$\left.\frac{\partial T}{\partial y}\right|_{\text{at } (x_i, y_N)} \doteq \frac{T_{i,N+1} - T_{i,N-1}}{2(\Delta y)} = 0$$

or

$$T_{i,N+1} = T_{i,N-1} \tag{9}$$

and

$$T_{i,N} = \frac{1}{4}\left(T_{i-1,N} + T_{i+1,N} + 2T_{i,N-1}\right) \quad \text{for } i = 8,9 \tag{10}$$

Having derived Equations 6, 8, and 10, it is easy to deduce the two special equation for the corner insulated stations to be:

$$T_{M,1} = \frac{1}{2}\left(T_{M,2} + T_{M-1,1}\right) \tag{11}$$

and

$$T_{M,N} = \frac{1}{2}\left(T_{M,N-1} + T_{M-1,N}\right) \tag{12}$$

We have derived all equations needed for averaging the temperature at any station of interest including those at the insulated boundaries by utilizing those at its neighboring stations. It suggests that a continuous upgrading process can be developed which assumes that the neighboring temperatures are known. This so-called *relaxation* method starts with an initial assumed distribution of temperature $[T^{(0)}]$ and continues to use Equations 3 and 6 to 12 until the differences at all locations are small enough. **Mathematica**lly, the process terminates when:

$$\sum_{i=1}^{M}\sum_{j=1}^{N}\left|T_{i,j}^{(k+1)} - T_{i,j}^{(k)}\right| < \varepsilon \tag{13}$$

where ϵ is a prescribed tolerance of accuracy and $k = 0,1,2,\ldots$ is the number of sweeps in upgrading the temperature distribution. Superscripts (k + 1) and (k) refer to the improved and previous distributions, respectively. The order of sweep will affect how the temperatures should be upgraded. For example, if the temperatures are to be re-averaged from top to bottom and left to right, referring to Figure 1, then Equation 3 is to be modified as:

$$T_{i,j}^{(k+1)} = \frac{1}{4}\left(T_{i-1,j}^{(k+1)} + T_{i+1,j}^{(k)} + T_{i,j-1}^{(k+1)} + T_{i,j+1}^{(k)}\right) \tag{14}$$

Notice that the neighboring temperatures in the row above, i–1, and in the column to the left, j–1, have already been upgraded while those in the row below, i + 1, and in the column to the right, j + 1, are yet to be upgraded. Similar modifications are to be made to Equations 6 to 12 during relaxation.

The program **Relaxatn** is developed according to the relaxation method described above. For solving the problem shown in Figure 3, both **FORTRAN** and **QuickBASIC** versions are made available for interactively specifying the tolerance. Sample results are presented below.

FORTRAN Version

```
C  Program Relaxatn - Steady-state temperature distribution determined
C                     by application of relaxation method.
      DIMENSION T(10,20)
      DATA T/200*0./
C  Initially all temperatures are equal to zero degree F.
C  The left, upper boundary is heated.
      WRITE (*,101)
  101 FORMAT(1X,'Program Relaxatn - Relaxation method applied for ',
     *       'solving heat conduction problem')
      WRITE (*,103)
  103 FORMAT(1X,'Enter the tolerance, epsilon, for termination : ')
      READ (*,*) Epsilon
      DO 1 I=2,6
    1 T(I,1)=(I-1)*10.
C  Relaxation proceeds.
      NR=1
    2 WRITE (*,3) NR
    3 FORMAT(1X,'Sweep #',I5)
      DO 25 I=1,10
   25 WRITE (*,30) (T(I,J),J=1,20)
   30 FORMAT(20F4.0)
      D=0
      DO 4 I=2,6
      DO 4 J=2,19
      TS=T(I,J)
      T(I,J)=.25*(T(I-1,J)+T(I+1,J)+T(I,J-1)+T(I,J+1))
    4 D=D+ABS(TS-T(I,J))
      DO 8 I=7,9
      TS=T(I,1)
      T(I,1)=.25*(T(I-1,1)+T(I+1,1)+2*T(I,2))
      D=D+ABS(TS-T(I,1))
      DO 6 J=2,19
      TS=T(I,J)
      T(I,J)=.25*(T(I-1,J)+T(I+1,J)+T(I,J-1)+T(I,J+1))
    6 D=D+ABS(TS-T(I,J))
      IF (I.EQ.7) GOTO 8
      TS=T(I,20)
      T(I,20)=.25*(T(I-1,20)+T(I+1,20)+2*T(I,19))
      D=D+ABS(TS-T(I,20))
    8 CONTINUE
      TS=T(10,1)
      T(10,1)=.5*(T(9,1)+T(10,2))
      D=D+ABS(TS-T(10,1))
      DO 10 J=2,19
      TS=T(I,J)
      T(10,J)=.25*(2*T(9,J)+T(10,J-1)+T(10,J+1))
   10 D=D+ABS(TS-T(10,J))
      TS=T(10,20)
      T(10,20)=.5*(T(9,20)+T(10,19))
      D=D+ABS(TS-T(10,20))
      IF (D.LT.EPSILON) GOTO 20
      NR=NR+1
      GO TO 2
```

```
C  Program Relaxatn - Steady-state temperature distribution determined
C                     by application of relaxation method.
      DIMENSION T(10,20)
      DATA T/200*0./
C  Initially all temperatures are equal to zero degree F.
C  The left, upper boundary is heated.
      WRITE (*,101)
  101 FORMAT(1X,'Program Relaxatn - Relaxation method applied for ',
     *          'solving heat conduction problem')
      WRITE (*,103)
  103 FORMAT(1X,'Enter the tolerance, epsilon, for termination : ')
      READ (*,*) Epsilon
      DO 1 I=2,6
    1 T(I,1)=(I-1)*10.
C  Relaxation proceeds.
      NR=1
    2 WRITE (*,3) NR
    3 FORMAT(1X,'Sweep #',I5)
      DO 25 I=1,10
   25 WRITE (*,30) (T(I,J),J=1,20)
   30 FORMAT(20F4.0)
      D=0
      DO 4 I=2,6
      DO 4 J=2,19
      TS=T(I,J)
      T(I,J)=.25*(T(I-1,J)+T(I+1,J)+T(I,J-1)+T(I,J+1))
    4 D=D+ABS(TS-T(I,J))
      DO 8 I=7,9
      TS=T(I,1)
      T(I,1)=.25*(T(I-1,1)+T(I+1,1)+2*T(I,2))
      D=D+ABS(TS-T(I,1))
      DO 6 J=2,19
      TS=T(I,J)
      T(I,J)=.25*(T(I-1,J)+T(I+1,J)+T(I,J-1)+T(I,J+1))
    6 D=D+ABS(TS-T(I,J))
      IF (I.EQ.7) GOTO 8
      TS=T(I,20)
      T(I,20)=.25*(T(I-1,20)+T(I+1,20)+2*T(I,19))
      D=D+ABS(TS-T(I,20))
    8 CONTINUE
      TS=T(10,1)
      T(10,1)=.5*(T(9,1)+T(10,2))
      D=D+ABS(TS-T(10,1))
      DO 10 J=2,19
      TS=T(I,J)
      T(10,J)=.25*(2*T(9,J)+T(10,J-1)+T(10,J+1))
   10 D=D+ABS(TS-T(10,J))
      TS=T(10,20)
      T(10,20)=.5*(T(9,20)+T(10,19))
      D=D+ABS(TS-T(10,20))
      IF (D.LT.EPSILON) GOTO 20
      NR=NR+1
      GO TO 2
```

Sample Results

The program **Relaxatn** is first applied for an interactively entered value of equal to 100. Only one relaxation needs to be implemented as shown below. The temperature distribution for Sweep #1 is actually the initial assumed distribution. One cannot assess how accurate this distribution is. The second run specifies that be

equal to 1. The results show that 136 relaxation steps are required. For giving more insight on how the relaxation has proceeded, Sweeps #10, #30, #50, #100, and #137 are presented for interested readers. It clearly indicates that a tolerance of equal to 100 is definitely inadequate.

```
Program Relaxatn - Relaxation method applied for solving heat conduction problem

Enter the tolerance, epsilon, for termination :
100

Sweep #    1
 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
10.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
20.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
30.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
40.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
50.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
Sweep #    2
 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
10.  3.  1.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
20.  6.  2.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
30.  9.  3.  1.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
40. 12.  4.  1.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
50. 16.  5.  1.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
13.  7.  3.  1.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
 3.  3.  1.  1.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
 1.  1.  1.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
 0.  1.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
Stop - Program terminated.


Program Relaxatn - Relaxation method applied for solving heat conduction problem

Enter the tolerance, epsilon, for termination :
1

Sweep #   10

 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
10.  7.  5.  3.  2.  1.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
20. 14.  9.  6.  3.  2.  1.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
30. 20. 13.  8.  5.  3.  1.  1.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
40. 26. 16.  9.  5.  3.  2.  1.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
50. 27. 16. 10.  6.  3.  2.  1.  1.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
25. 19. 13.  8.  5.  3.  2.  1.  1.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
14. 13.  9.  7.  4.  3.  2.  1.  1.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
10.  9.  7.  5.  4.  2.  2.  1.  1.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
 8.  8.  7.  5.  4.  2.  2.  1.  1.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.

Sweep #   30

 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
10.  8.  6.  4.  3.  2.  2.  1.  1.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
20. 15. 12.  8.  6.  4.  3.  2.  1.  1.  1.  0.  0.  0.  0.  0.  0.  0.  0.
30. 23. 17. 12.  9.  6.  4.  3.  2.  1.  1.  1.  0.  0.  0.  0.  0.  0.  0.
40. 29. 20. 14. 10.  7.  5.  4.  3.  2.  1.  1.  1.  0.  0.  0.  0.  0.  0.
50. 31. 22. 16. 11.  8.  6.  4.  3.  2.  1.  1.  1.  0.  0.  0.  0.  0.  0.
31. 26. 20. 15. 12.  9.  6.  5.  3.  2.  2.  1.  1.  0.  0.  0.  0.  0.  0.
23. 21. 18. 14. 11.  9.  6.  5.  3.  2.  2.  1.  1.  1.  0.  0.  0.  0.  0.
20. 19. 16. 14. 11.  9.  7.  5.  4.  3.  2.  1.  1.  1.  0.  0.  0.  0.  0.
19. 18. 16. 14. 11.  9.  7.  5.  4.  3.  2.  1.  1.  1.  0.  0.  0.  0.  0.

Sweep #   50

 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
10.  8.  6.  5.  4.  3.  2.  2.  1.  1.  1.  0.  0.  0.  0.  0.  0.  0.  0.
```

```
20. 16. 12.  9.  7.  5.  4.  3.  2.  2.  1.  1.  1.  0.  0.  0.  0.  0.  0.  0.
30. 23. 18. 13. 10.  8.  6.  4.  3.  2.  2.  1.  1.  1.  0.  0.  0.  0.  0.  0.
40. 29. 22. 16. 12.  9.  7.  5.  4.  3.  2.  2.  1.  1.  1.  0.  0.  0.  0.  0.
50. 33. 24. 18. 14. 11.  8.  6.  5.  4.  3.  2.  2.  1.  1.  1.  0.  0.  0.  0.
33. 28. 23. 18. 14. 11.  9.  7.  5.  4.  3.  2.  2.  1.  1.  1.  0.  0.  0.  0.
26. 24. 21. 18. 15. 12.  9.  7.  6.  4.  3.  3.  2.  1.  1.  1.  1.  0.  0.  0.
23. 22. 20. 17. 14. 12. 10.  8.  6.  5.  4.  3.  2.  1.  1.  1.  1.  0.  0.  0.
22. 22. 20. 17. 14. 12. 10.  8.  6.  5.  4.  3.  2.  2.  1.  1.  1.  0.  0.  0.

Sweep #  100

 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
10.  8.  7.  5.  4.  3.  3.  2.  2.  1.  1.  1.  1.  1.  0.  0.  0.  0.  0.  0.
20. 16. 13. 10.  8.  6.  5.  4.  3.  3.  2.  2.  1.  1.  1.  1.  0.  0.  0.  0.
30. 24. 18. 14. 11.  9.  7.  6.  5.  4.  3.  2.  2.  2.  1.  1.  1.  0.  0.  0.
40. 30. 23. 18. 14. 11.  9.  7.  6.  5.  4.  3.  3.  2.  2.  1.  1.  1.  0.  0.
50. 34. 25. 20. 16. 13. 11.  9.  7.  6.  5.  4.  3.  2.  2.  1.  1.  1.  0.  0.
35. 30. 25. 21. 17. 14. 12.  9.  8.  6.  5.  4.  3.  3.  2.  2.  1.  1.  1.  0.
29. 27. 24. 20. 17. 15. 12. 10.  8.  7.  6.  5.  4.  3.  2.  2.  1.  1.  1.  1.
26. 25. 23. 20. 17. 15. 12. 10.  9.  7.  6.  5.  4.  3.  2.  2.  2.  1.  1.  1.
25. 24. 23. 20. 17. 15. 13. 11.  9.  7.  6.  5.  4.  3.  2.  2.  2.  1.  1.  1.

Sweep #  137

 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
10.  8.  7.  5.  4.  3.  3.  2.  2.  1.  1.  1.  1.  1.  0.  0.  0.  0.  0.  0.
20. 16. 13. 10.  8.  7.  5.  4.  3.  3.  2.  2.  1.  1.  1.  1.  0.  0.  0.  0.
30. 24. 18. 14. 12.  9.  8.  6.  5.  4.  3.  2.  2.  2.  1.  1.  1.  0.  0.  0.
40. 30. 23. 18. 14. 12. 10.  8.  6.  5.  4.  4.  3.  2.  2.  1.  1.  1.  0.  0.
50. 34. 26. 20. 16. 13. 11.  9.  7.  6.  5.  4.  3.  2.  2.  1.  1.  1.  0.  0.
35. 30. 25. 21. 17. 15. 12. 10.  8.  7.  6.  5.  4.  3.  3.  2.  2.  1.  1.  0.
29. 27. 24. 21. 18. 15. 13. 11.  9.  7.  6.  5.  4.  3.  3.  2.  2.  1.  1.  1.
26. 26. 23. 21. 18. 15. 13. 11.  9.  8.  6.  5.  4.  4.  3.  2.  2.  1.  1.  1.
26. 25. 23. 21. 18. 16. 13. 11.  9.  8.  7.  5.  4.  4.  3.  2.  2.  2.  1.  1.

Stop - Program terminated.
```

QuickBASIC Version

```
'  Program Relaxatn - Steady-state temperature distribution determined
'                     by application of relaxation method.
     CLS : CLEAR : KEY OFF: DIM T(10,20)
     FOR I = 1 TO 10: FOR J = 1 TO 20: T(I,J) = 0!: NEXT J: NEXT I
'  Initially all temperatures are equal to zero degree F.
'  The left, upper boundary is heated.
   PRINT "Program Relaxatn - Relaxation method applied for solving heat conduction problem "
   INPUT "Enter the tolerance, epsilon, for termination : ", EPSILON
      FOR I = 2 TO 6: T(I,1) = (I-1)*10!: NEXT I
'  Relaxation proceeds.
      NR = 1
2     PRINT : PRINT "Sweep #", NR
      FOR I = 1 TO 10: FOR J = 1 TO 20: PRINT USING " ##."; T(I,J); : NEXT J: NEXT I
      D = 0
      FOR I=2 TO 6
          FOR J=2 TO 19: TS=T(I,J)
              T(I,J)=.25*(T(I-1,J)+T(I+1,J)+T(I,J-1)+T(I,J+1))
              D = D + ABS(TS - T(I,J)): NEXT J:                NEXT I
      FOR I = 7 TO 9: TS = T(I, 1)
          T(I,1) = .25 * (T(I-1,1) + T(I+1,1) + 2 * T(I,2)): D = D + ABS(TS-T(I,1))
          FOR J  = 2 TO 19: TS=T(I,J): T(I,J)=.25*(T(I-1,J)+T(I+1,J)+T(I,J-1)+T(I,J+1))
              D  = D + ABS(TS - T(I,J)): NEXT J: IF I = 7 THEN 8
```

```
        TS = T(I,20): T(I,20) = .25 * (T(I-1,20)+T(I+1,20)+2*T(I,19))
        D  = D + ABS(TS - T(I,20))
8       NEXT I: TS = T(10,1): T(10,1)=.5*(T(9,1)+T(10,2)): D=D+ABS(TS-T(10,1))
    FOR J = 2 TO 19: TS=T(I,J): T(10,J)=.25*(2*T(9,J)+T(10,J-1)+T(10,J+1))
        D = D + ABS(TS - T(10,J)): NEXT J
    TS = T(10,20): T(10,20)=.5*(T(9,20)+T(10,19)): D=D+ABS(TS-T(10,20))
    IF D < EPSILON THEN 20 ELSE NR=NR+1: GOTO 2
20 END
```

Sample Results

```
Program Relaxatn - Relaxation method applied for solving heat conduction problem

Enter the tolerance, epsilon, for termination : 10

Sweep #         43
  0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
 10.   8.   6.   5.   3.   3.   2.   1.   1.   1.   1.   0.   0.   0.   0.   0.   0.   0.   0.
 20.  16.  12.   9.   7.   5.   4.   3.   2.   1.   1.   1.   0.   0.   0.   0.   0.   0.   0.
 30.  23.  17.  13.  10.   7.   5.   4.   3.   2.   2.   1.   1.   1.   0.   0.   0.   0.   0.
 40.  29.  21.  16.  12.   9.   7.   5.   4.   3.   2.   1.   1.   1.   1.   0.   0.   0.   0.
 50.  32.  23.  17.  13.  10.   8.   6.   4.   3.   2.   2.   1.   1.   1.   0.   0.   0.   0.
 33.  28.  22.  17.  14.  11.   8.   6.   5.   4.   3.   2.   1.   1.   1.   1.   0.   0.   0.
 25.  24.  20.  17.  14.  11.   9.   7.   5.   4.   3.   2.   2.   1.   1.   1.   0.   0.   0.
 22.  21.  19.  16.  14.  11.   9.   7.   5.   4.   3.   2.   2.   1.   1.   1.   0.   0.   0.
 21.  21.  19.  16.  14.  11.   9.   7.   5.   4.   3.   2.   2.   1.   1.   0.   0.   0.   0.
Press any key to continue
```

Irregular Boundaries

Practically, there are cases where the domain of heat conduction have boundaries which are quite irregular geometrically as illustrated in Figure 4. For such cases, the equation derived based on the relaxation method, Equation 3, which states that the temperature at any point has the average value of those at its four neighboring points if they are equally apart, has to be modified. The modified equation can be derived using a simple argument applied in both x and y directions. For example, consider the temperature at the point G, T_G, in Figure 4. First, let us investigate the horizontal, y direction (for convenience of associating x and y with the row and column indices i and j, respectively as in Figure 2). We observe that T_G is affected more by the temperature at the point C, T_C, than by that at the point I, T_I because the point C is closer to the point G than the point I. Since the closer the point, the greater the influence, based on linear variation of the temperature we can then write:

$$T_G = \frac{\Delta y}{\Delta y + \beta' \Delta y} T_C + \frac{\beta' \Delta y}{\Delta y + \beta \Delta y} T_I = \frac{1}{1+\beta'} T_C + \frac{\beta'}{1+\beta'} T_I \tag{15}$$

where the increment from point I to point G is the regular increment Δy while that between G and C is less and equal to $\beta' \Delta y$ with β' having a value between 0 and 1. Similarly, along the vertical, x direction and considering the points B, G, and H and a regular increment Δx, we can have:

$$T_G = \frac{\Delta x}{\Delta x + \alpha' \Delta x} T_B + \frac{\alpha' \Delta x}{\Delta x + \alpha' \Delta x} T_H = \frac{1}{1+\alpha'} T_B + \frac{\alpha'}{1+\alpha'} T_H \tag{16}$$

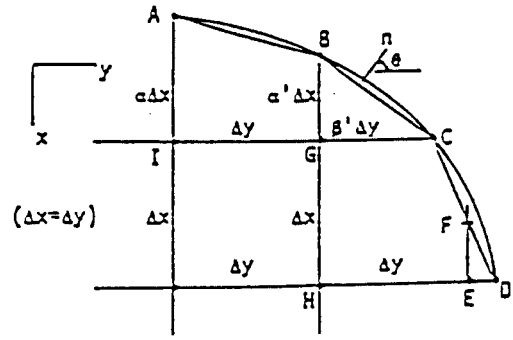


FIGURE 4. There are cases where the domain of heat conduction have boundaries which are quite irregular geometrically.

where like β', α' has a value between 0 and 1. As often is the case, the regular increments Δx and Δy are taken to be equal to each other for the simplicity of computation. Equations 15 and 16 can then be combined and by taking both x and y directions into consideration, an averaging approach leads to:

$$T_G = \frac{1}{2}\left(\frac{1}{1+\alpha'}T_B + \frac{\alpha'}{1+\alpha'}T_H + \frac{1}{1+\beta'}T_C + \frac{\beta'}{1+\beta'}T_I\right) \tag{17}$$

For every group of five points such as B, C, G, H, and I in Figure 4 situated at any irregular boundary, the values of α' and β' have to be measured and Equation 17 is to be used during the relaxation process if the boundary temperatures are known.

If some points along an irregular boundary are insulated such as the points B and C in Figure 4, we need to derive new formula to replace Equation 6 or Equation 10. The insulated condition along BC requires $\partial T \partial n = 0$ where n is the direction normal to the cord BC when the arc BC is approximated linearly. If the values of ‘ and ‘ are known, we can replace the condition $\partial T/\partial n = 0$ with

$$\frac{\partial T}{\partial n} = \frac{\partial T dX}{\partial X dn} + \frac{\partial T dY}{\partial Y dn} = \frac{\partial T}{\partial X}\sin\theta + \frac{\partial T}{\partial Y}\cos\theta = \alpha'\frac{\partial T}{\partial X} + \beta'\frac{\partial T}{\partial Y} = 0 \tag{18}$$

The remainder of derivation is left as a homework problem.

MATLAB Application

A **Relaxatn.m** file can be created to perform interactive **MATLAB** operations and generate plots of the temperature distributions during the course of relaxation. This file may be prepared as follows:

```
function [SumOfDs,T]=Relaxatn(T)
        SumOfDs=0;
        for i=2:6
            for j=2:19
                Tsave=T(i,j);
                T(i,j)=.25*(T(i-1,j)+T(i+1,j)+T(i,j-1)+T(i,j+1));
                SumOfDs=abs(Tsave-T(i,j));
            end
        end
        for i=7:9
            Tsave=T(i,1);
            T(i,1)=.25*(T(i-1,1)+T(i+1,1)+2*T(i,2));
            SumOfDs=abs(Tsave-T(i,1));
            for j=2:19
                Tsave=T(i,j);
                T(i,j)=.25*(T(i-1,j)+T(i+1,j)+T(i,j-1)+T(i,j+1));
                SumOfDs=abs(Tsave-T(i,j));
            end
            if i>7 Tsave=T(i,20);
                   T(i,20)=.25*(T(i-1,20)+T(i+1,20)+2*T(i,19));
                   SumOfDs=abs(Tsave-T(i,20));
            end
        end
        Tsave=T(10,1);
        T(10,1)=.5*(T(9,1)+T(10,2));
        SumOfDs=abs(Tsave-T(10,1));
        for j=2:19
            Tsave=T(10,j);
            T(10,j)=.25*(2*T(9,j)+T(10,j-1)+T(10,j+1));
            SumOfDs=abs(Tsave-T(10,j));
        end
        Tsave=T(10,20);
        T(10,20)=.5*(T(9,20)+T(10,19));
        SumOfDs=abs(Tsave-T(10,20));
```

This file can be applied to solve the sample problem run by first specifying the boundary temperatures described in Equation 4 to obtain an initial distribution by entering the **MATLAB** instructions:

```
>> T=zeros(10,20); for I=2:6, T(i,1)=(I-1)*10; end, format compact, NR=1;
>> fprintf('Sweep # %3.0f \n',NR), T
   Sweep #   1
   T =
   Columns 1 through 12

     0     0     0     0     0     0     0     0     0     0     0     0
    10     0     0     0     0     0     0     0     0     0     0     0
    20     0     0     0     0     0     0     0     0     0     0     0
    30     0     0     0     0     0     0     0     0     0     0     0
    40     0     0     0     0     0     0     0     0     0     0     0
    50     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
```

```
Columns 13 through 20
     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0
```

The fprintf command enables a label be added, in which the format %3.0f requests 3 columns be provided without the decimal point for printing the value of NR, and \n requests that next printout should be started on a new line. The Relaxatn.m can now be utilized to perform the relaxations. Let first perform one relaxation by entering

```
>> NR = NR + 1; fprintf('Sweep # %3.0f \n',NR), [D,T] = feval(A:Relaxatn',T]
```

The resulting display of the error defined in Equation 13 and the second temperature distribution is:

```
Sweep #   2
D =
   7.0137e-008
T =
Columns 1 through 7
         0         0         0         0         0         0         0
  10.0000    2.5000    0.6250    0.1563    0.0391    0.0098    0.0024
  20.0000    5.6250    1.5625    0.4297    0.1172    0.0317    0.0085
  30.0000    8.9063    2.6172    0.7617    0.2197    0.0629    0.0179
  40.0000   12.2266    3.7109    1.1182    0.3345    0.0993    0.0293
  50.0000   15.5566    4.8169    1.4838    0.4546    0.1385    0.0419
  12.5000    7.0142    2.9678    1.1104    0.3912    0.1324    0.0436
   3.1250    2.5348    1.3731    0.6209    0.2530    0.0964    0.0350
   0.7813    0.8290    0.5505    0.2929    0.1365    0.0582    0.0233
   0.3906    0.5512    0.4033    0.2473    0.1300    0.0616    0.0271
Columns 8 through 14
         0         0         0         0         0         0         0
   0.0006    0.0002    0.0000    0.0000    0.0000    0.0000    0.0000
   0.0023    0.0002    0.0002    0.0000    0.0000    0.0000    0.0080
   0.0050    0.0014    0 0004    0.0001    0.0000    0.0000    0.0000
   0.0086    0.0025    0.0007    0.0002    0.0001    0.0000    0.0000
   0.0126    0.0038    0.0011    0.0003    0.0001    0.0000    0.0000
   0.0141    0.0045    0.0014    0.0004    0.0001    0.0000    0.0000
   0.0123    0.0042    0.0014    0.0005    0.0001    0.0000    0.0000
   0.0089    0.0033    0.0012    0.0004    0.0001    0.0000    0.0000
   0.0112    0.0044    0.0017    0.0006    0.0002    0.0001    0.0000
Columns 15 through 20
         0         0         0         0         0         0
   0.0000    0.0000    0.0000    0.0000    0.0000         0
   0.0000    0.0000    0.0000    0.0000    0.0000         0
   0.0000    0.0000    0.0000    0.0000    0.0000         0
   0.0000    0.0000    0.0000    0.0000    0.0000         0
   0.0000    0.0000    0.0000    0.0000    0.0000         0
   0.0000    0.0000    0.0000    0.0000    0.0000         0
   0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
   0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
   0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
```

In case that we need to have the 30th temperature distribution by performing 29 consecutive relaxations, we enter:

```
>> for NR = 3:30; [D,T] = feval(A:Relaxatn',T]; end
>> fprintf('Sweep # %3.0f \n',NR),D,T
```

The resulting display of the error defined in Equation 13 and the 30th temperature distribution is:

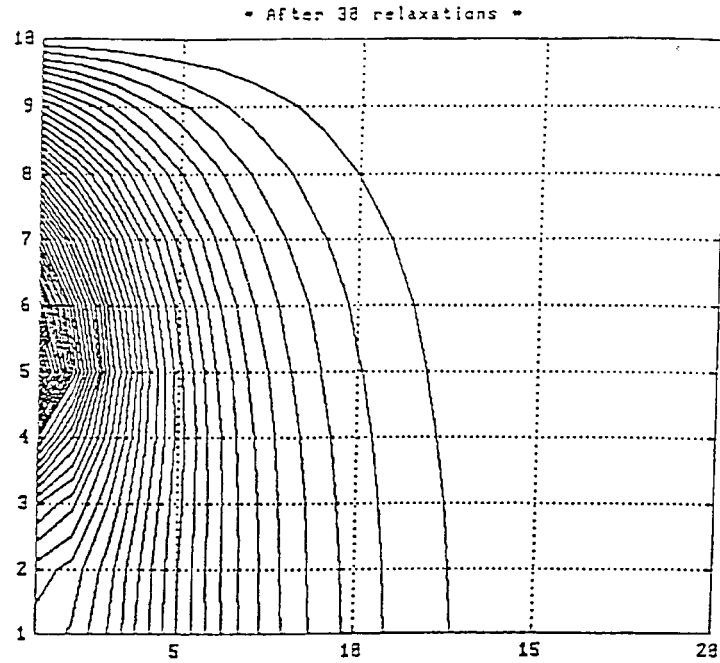
```
     Sweep #  30
D =
    0.0082
T =
Columns 1 through 7
       0         0         0         0         0         0         0
 10.0000    7.8240    5.8971    4.3264    3.1130    2.2070    1.5452
 20.0000   15.4564   11.5265    8.4017    6.0310    4.2789    3.0045
 30.0000   22.5767   16.5012   11.9001    8.5147    6.0506    4.2671
 40.0000   28.5013   20.2189   14.4310   10.3309    7.3837    5.2490
 50.0000   31.4309   21.7354   15.5950   11.3147    8.2054    5.9135
 31.0236   25.8271   20.0781   15.2923   11.5089    8.6376    6.2913
 23.0526   21.2258   17.9193   14.4421   11.2972    0.6354    6.5749
 19.5060   18.6290   16.4481   13.7563   11.0644    8.6354    6.5749
 18.5571   17.8914   16.0265   13.5837   11.0448    8.6947    6.6665
Columns 8 through 14
       0         0         0         0         0         0         0
  1.0695    0.7318    0.4949    0.3307    0.2182    0.1422    0.0914
  2.0881    1.4358    0.9762    0.6559    0.4353    0.2853    0.1845
  2.9830    2.0645    1.4132    0.9560    0.6387    0.4213    0.2742
  3.7013    2.5840    1.7836    1.2163    0.8189    0.5441    0.3567
  4.2188    2.9802    2.0778    1.4298    0.9707    0.6500    0.4293
  4.5613    3.2612    2.2985    1.5968    1.0934    0.7379    0.4909
  4.7682    3.4517    2.4581    1.7229    1.1891    0.8083    0.5414
  4.8998    3.5816    2.5715    1.8153    1.2609    0.8623    0.5809
  4.9975    3.6720    2.6490    1.8784    1.3105    0.9001    0.6091
Columns 15 through 20
       0         0         0         0         0         0
  0.0580    0.0361    0.0220    0.0126    0.0058         0
  0.1177    0.0738    0.0451    0.0259    0.0120         0
  0.1760    0.1111    0.0683    0.0395    0.0185         0
  0.2305    0.1465    0.0908    0.0529    0.0250         0
  0.2794    0.1790    0.1119    0.0662    0.0320         0
  0.3219    0.2078    0.1315    0.0799    0.0414         0
  0.3574    0.2326    0.1494    0.0948    0.0606    0.0447
  0.3858    0.2529    0.1643    0.1074    0.0749    0.0650
  0.4063    0.2676    0.1751    0.1159    0.0828    0.0739
```

To obtain a plot of this temperature distribution after the initial temperature distribution has been relaxed 29 times, with gridwork and title as shown in Figure 5a, the interactive **MATLAB** instructions entered are:

```
>> V= 0:1:50; contour(T,V'), grid, title('* After 30 relaxations *')
```

Notice that 51 contours having values 0 through 50 with an increment of 1 defined in the row matrix V. In Figure 5a, the contour having a value equal to 0 is

(a)



(b)

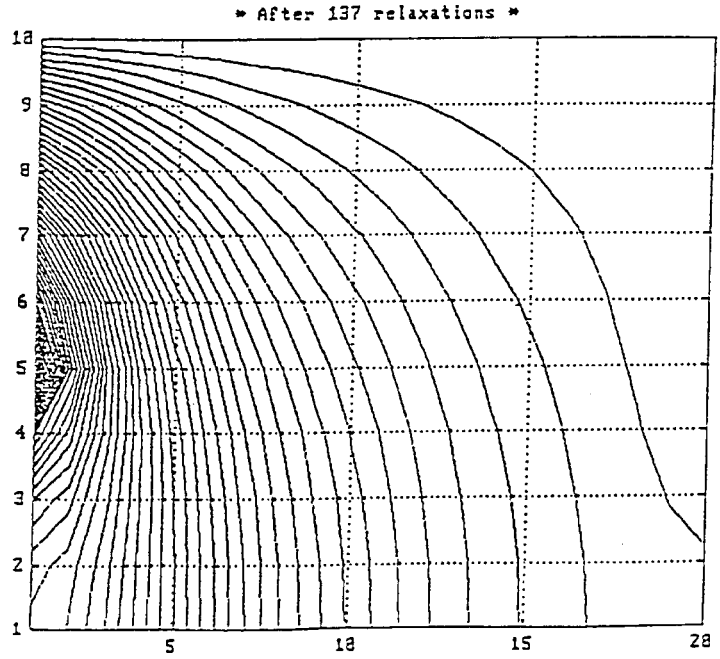


FIGURE 5. After 38 relaxations (a) and after 137 relaxations (b).

along the upper edge (Y = 10) and right edge (X = 20), the first *curved* contour of the right has a value equal to 1, and the values of the contours are increased from right to left until the point marked "5" which has a temperature equal to 50 is reached. It should be noted that along the left edge, the uppermost point marked "10" has a temperature equal to zero and the temperatures are increased linearly (as for the initial conditions) to 50 at the point marked with "5", and from that point down to the point marked "1" the entire lower portion of the left edge is insulated.

For obtaining the 137th sweep, we can continue to call the service **Relaxatn.m** by similarly applying the **MATLAB** instructions as follows:

```
>> for NR = 31:137; [D,T] = feval(A:Relaxatn',T]; end
>> fprintf('Sweep # %3.0f \n',NR),D,T
```

The resulting display of the error defined in Equation 13 and the 137th temperature distribution is:

```
Sweep # 137
D =
    0.0039
T =
Columns 1 through 7
       0         0         0         0         0         0         0
 10.0000    8.2092    6.6034    5.2663    4.1979    3.3592    2.7027
 20.0000   16.2354   12.9413   10.2680    8.1709    6.5415    5.2714
 30.0000   23.7946   18.6638   14.7005   11.6842    9.3736    7.5788
 40.0000   30.2843   23.2266   18.1958   14.5030   11.7020    9.5196
 50.0000   34.1232   25.7729   20.3657   16.4440   13.4267   11.0289
 34.9702   30.4465   25.3896   21.0650   17.9476   14.5495   12.1007
 29.0063   27.3172   24.2898   21.0253   17.9476   15.1931   12.7873
 26.4439   25.5428   23.4447   20.8176   18.0951   15.5084   13.1623
 25.7091   24.9832   23.1469   20.7428   18.1272   15.6043   13.2838
Columns 8 through 14
       0         0         0         0         0         0         0
  2.1860    1.7755    1.4457    1.1780    0.9585    0.7764    0.6235
  4.2718    2.4760    2.8351    2.3135    1.8846    1.5384    1.2289
  6.1633    5.0311    4.1147    3.3652    2.7466    2.2313    1.7974
  7.7842    6.3834    5.2398    4.2979    3.5164    2.8631    2.3120
  9.0865    7.4942    6.1784    5.0849    4.1717    3.4053    2.7578
 10.0569    8.3464    6.9123    5.7084    4.6961    3.8432    3.1218
 10.7141    8.9421    7.4354    6.1586    5.0785    4.1464    3.3929
 11.0913    9.2934    7.7490    6.4315    5.3122    4.3639    3.5615
 11.2170    9.4124    7.8565    6.5259    5.3937    4.4336    3.6212
Columns 15 through 20
       0         0         0         0         0         0
  0.4931    0.3792    0.2770    0.1819    0.0904         0
  0.9731    0.7498    0.5490    0.3616    0.1802         0
  1.4266    1.1028    0.8111    0.5375    0.2696         0
  1.8410    1.4302    1.0601    0.7107    0.3619         0
  2.2050    1.7248    1.2945    0.8874    0.4688         0
  2.5078    1.9790    1.5129    1.0807    0.6278         0
  2.7380    2.1811    1.7060    1.3007    0.9651    0.7498
  2.8836    2.3131    1.8388    1.4589    1.1884    1.0729
  2.9358    2.3610    1.8876    1.5164    1.2645    1.1687
```

Figure 5b shows the 137th temperature distribution when the interactive **MATLAB** instructions entered are:

```
>> V = 0:1:50; contour(T,V'), grid, title('* After 137 relaxations *')
```

Notice that area near the insulated boundaries at the right-lower corner has finally reached a steady-state temperature distribution, i.e., changes of the entire temperature distribution will be insignificant if more relaxations were pursued.

Mathematica Applications

To apply the relaxation method for finding the steady-state temperature distribution of the heated plate already solved by the **FORTRAN**, **QuickBASIC**, and **MATLAB** versions, here we make use of the **Do**, **If**, and **While** commands of **Mathematica** to generate similar results through the following interactive operations:

```
In[1]: = t = Table[0,{10},{20}]; eps = 100; nr = 0; d = eps + 1;

In[2]: = Do[t[[i,1]] = (I–1)*10,{i,2,6}];

In[3]: = (While[d>eps, d = 0;nr = nr + 1;
    Do[Do[ts = t[[i,j]]; t[[i,j]] = .25*(t[[I–1,j]] + t[[I + 1,j]]
          + t[[i,j–1]] + t[[i,j + 1]]);
          d = Abs[ts-t[[i,j]]] + d;,{j,2,19}],{i,2,6}];
       Do[ts = t[[i,1]]; t[[i,1]] = .25*(t[[I–1,1]] + t[[I + 1,1]]
          + 2*t[[i,2]]); d = Abs[ts-t[[i,1]]] + d;
         Do[ts = t[[i,j]]; t[[i,j]] = .25*(t[[I–1,j]] + t[[I + 1,j]]
          + t[[i,j–1]] + t[[i,j + 1]]);
          d = Abs[ts-t[[i,j]]] + d;,{j,2,19}];
          If[i = = 7,Continue, ts = t[[i,20]];
          t[[i,20]] = .25*(t[[I–1,20]] + t[[I + 1,20]] + 2*t[[i,19]]);
          d = Abs[ts-t[[i,20]]] + d;],{i,7,9}];
          ts = t[[10,1]]; t[[10,1]] = .5*(t[[9,1]] + t[[10,2]]);
          d = Abs[ts-t[[10,1]]] + d;
          Do[ts = t[[10,j]]; t[[10,j]] = .25*(2*t[[9,j]] + t[[10,j–1]]
               + t[[10,j + 1]]); d = Abs[ts-t[[10,j]]] + d;,{j,2,19}];
            ts = t[[10,20]]; t[[10,20]] = .5*(t[[9,20]] + t[[10,19]]);
            d = Abs[ts-t[[10,20]]] + d;])

In[4]: = Print["Sweep #",nr]; Round[N[t,2]]

Out[4] = Sweep #2

    {{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {10, 4, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {20, 9, 3, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
```

{30, 13, 5, 2, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
{40, 18, 7, 3, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
{50, 20, 8, 3, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
{17, 11, 5, 2, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
{ 6, 5, 3, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
{ 2, 2, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
{ 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}}

Notice that *In[2]* initializes the boundary temperatures, nr keeps the count of how many sweeps have been performed, and the function Round is employed in *In[4]* to round the temperature value to a two-digit integer. When the total temperature differences, d, is limited to eps = 100 degrees, the t values obtained after two sweeps are slightly different from those obtained by the other versions, this is again because **Mathematica** keeps more significant digits in all computation steps than those in the **FORTRAN**, **QuickBASIC**, and **MATLAB** programs. By changing the eps value from 100 degrees to 1 degree, **Mathematica** also takes 137 sweeps to converge as in the **FORTRAN**, **QuickBASIC**, and **MATLAB** versions:

In[5]: = t = Table[0,{10},{20}]; eps = 1; nr = 0; d = eps + 1;

In[6]: = Do[t[[i,1]] = (I–1)*10,{i,2,6}];

In[7]: = Print[“Sweep #”,nr]; Round[N[t,2]]

Out[7] = Sweep #137

{{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
{10, 8, 7, 5, 4, 3, 3, 2, 2, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0},
{20, 16, 13, 10, 8, 7, 5, 4, 3, 3, 2, 2, 2, 1, 1, 1, 1, 0, 0, 0},
{30, 24, 19, 15, 12, 9, 8, 6, 5, 4, 3, 3, 2, 2, 1, 1, 1, 1, 0, 0},
{40, 30, 23, 18, 15, 12, 10, 8, 6, 5, 4, 4, 3, 2, 2, 1, 1, 1, 0, 0},
{50, 34, 26, 20, 16, 13, 11, 9, 7, 6, 5, 4, 3, 3, 2, 2, 1, 1, 0, 0},
{35, 30, 25, 21, 17, 15, 12, 10, 8, 7, 6, 5, 4, 3, 3, 2, 2, 1, 1, 0},
{29, 27, 24, 21, 18, 15, 13, 11, 9, 7, 6, 5, 4, 3, 3, 2, 2, 1, 1, 1},
{26, 26, 23, 21, 18, 16, 13, 11, 9, 8, 6, 5, 4, 4, 3, 2, 2, 1, 1, 1},
{26, 25, 23, 21, 18, 16, 13, 11, 9, 8, 7, 5, 4, 4, 3, 2, 2, 2, 1, 1}}

The above results all agree with those obtained earlier.

Warping Analysis of a Twisted Bar with Noncircular Cross Section

As another example of applying the relaxation method for engineering analysis, consider the case of a long bar of uniform rectangular cross section twisted by the two equal torques (T) at its ends. The cross section of the twisted bar becomes warped as shown in Figure 6. If z-axis is assigned to the longitudinal direction of the bar, to find the amount of warping at any point (x,y) of the cross sectioned surface, denoted as W(x,y), the relaxation method can again be employed because

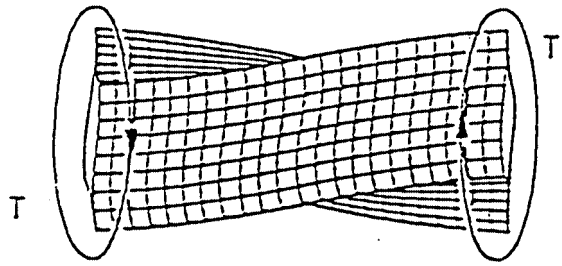


FIGURE 6. In the case of a bar of uniform rectangular cross section twisted by the two equal torques (T) at its ends, the cross section of the twisted bar becomes warped as shown.

W(x,y) is governed by the Laplace equation.[3] Due to anti-symmetry of W(x,y), that is W(–x,y) = W(x,–y) = –W(x,y), only one-fourth of the cross section needs to be analyzed. Let us consider a rectangular region $0 \le x \le a$ and $0 \le y \le b$. It is obvious that the anti-symmetry leads to the conditions W(x,0) = 0 and W(0,y) = 0 along two of the four linear boundaries. To derive the boundary conditions along the right side x = a and the upper side y = b, we have to utilize the relationships between the warping function W(x,y) and shear stresses τ_{xz} and τ_{yz} which are:

$$\tau_{xz} = G\theta\left(\frac{\partial W}{\partial x} - y\right) \quad \text{and} \quad \tau_{yz} = G\theta\left(\frac{\partial W}{\partial y} + x\right) \tag{19,20}$$

where G and θ are the shear rigidity and twisting angle of the bar, respectively. Along the boundaries x = a and y = b, the shear stress should be tangential to the lateral surface of the twisted bar requires:

$$\left(\frac{\partial W}{\partial x} - y\right)\frac{dy}{ds} - \left(\frac{\partial W}{\partial y} + x\right)\frac{dx}{ds} = 0 \tag{21}$$

where s is the variable changed along the boundary. Along the upper boundary y = b, dy/ds = 0 and dx/ds = 1 and along the right boundary x = a, dx/ds = 0 and dy/ds = 1. Consequently, Equation 19 reduces to:

$$\frac{\partial W}{\partial y} = -x \quad \text{for } y = b \quad \text{and} \quad \frac{\partial W}{x} = y \quad \text{for } x = a \tag{22,23}$$

To apply the relaxation method for solving W(x,y) which satisfies the Laplace equation for $0<x<a$ and $0<y<b$ and the boundary conditions W(0,y) = W(x,0) = 0

and Equations 22 and 23, let us partition the rectangular region $0 \le x \le a$ and $0 \le y \le b$ into a network of MxN using the increments $\Delta x = a/(M-1)$ and $\Delta y = b/(N-1)$. In finite-difference forms, Equations 22 and 23 yield, respectively:

$$W\left(x_i, y = y_N \equiv b\right) = W\left(x_i, b - \Delta y\right) - x_i \Delta y \qquad \text{for } i = 2,3,\ldots,M-1 \tag{24}$$

and

$$W\left(x = x_M \equiv a, y_j\right) = W\left(a - \Delta x, y_j\right) + y_j \Delta x \qquad \text{for } j = 2,3,\ldots,N-1 \tag{25}$$

where $x_i = (i-1)\Delta x$ and $y_j = (j-1)\Delta y$. Equations 24 and 25 can be combined to yield a finite-difference formula for relaxing the corner point (x_M, y_N). That is:

$$W\left(x_M \equiv a, y_N \equiv b\right) = .5\left[W\left(a, y_{N-1}\right) + W\left(x_{M-1}, b\right) + b\Delta x - a\Delta y\right] \tag{26}$$

Program **Relaxatn.m** has been modified to develop a new m file **Warping.m** for the warping analysis which uses input values for a, b, M, N, and which is needed in Equation 13 for termination of the relaxation. **MATLAB** statements for sample application of **Warping.m** are listed below along with the m-file itself. The **mesh** plot of the warping surface W(x,y) obtained after 131 sweeps of relaxation by initially assuming W(x,y) = 0 throughout the entire region and an ϵ value equal to 100 is shown in Figure 7.

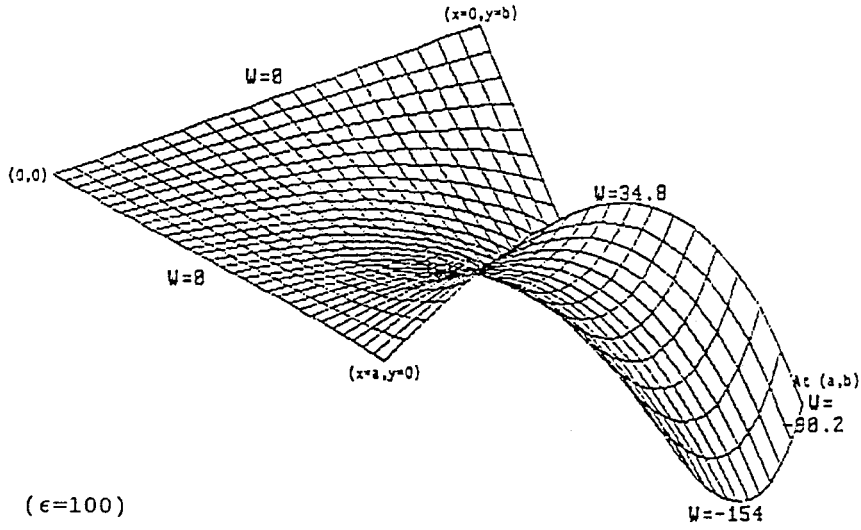


FIGURE 7. The result for ϵ = 100 when a = 30 and b = 20 means that each of the 600 gridpoints is allowed to have, on average, a difference equal to 1/6 during two consecutive relaxations.

```
>>W=zeros(31,21); [W,n]=warping(30,20,31,21,100,W), mesh(W)

function [W,Nrelax]=Warping(a,b,M,N,Epsilon,W)
        dx=a/(M-1); dy=b/(N-1); Nrelax=1; ExitFlag=0;
        while ExitFlag==0;
          SumOfDs=0; Wsave=W(M,N);
          %   Eq. (26)
              W(M,N)=.5*(W(M,N-1)+W(M-1,N)+b*dx-a*dy);
          SumOfDs=SumOfDs+abs(Wsave-W(M,N));
          for jr=2:N-1; j=N+1-jr;
              Wsave=W(M,j);
          %   Eq. (25)
              W(M,j)=W(M-1,j)+(j-1)*dy*dx;
              SumOfDs=SumOfDs+abs(Wsave-W(M,j));
          end
          for ir=2:M-1; i=M+1-ir;
              Wsave=W(i,N);
          %   Eq. (24)
              W(i,N)=W(i,N-1)-(i-1)*dx*dy;
              SumOfDs=SumOfDs+abs(Wsave-W(i,N));
              for jr=2:N-1; j=N+1-jr;
                  Wsave=W(i,j);
              %   Eq. (8)
                  W(i,j)=.25*(W(i-1,j)+W(i+1,j)+W(i,j-1)+W(i,j+1));
                  SumOfDs=SumOfDs+abs(Wsave-W(i,j));
              end
          end
          if SumOfDs<Epsilon, ExitFlag=1; break
             else fprintf('No. of Sweep = %3.0f \n',Nrelax)
                  fprintf('Total W differences = %12.5e \n',SumOfDs)
                  Nrelax=Nrelax+1;
          end
        end
```

Notice that the variable Nrelax keeps track how many sweeps have been performed during the relaxation procedure, it is an output argument like W which can be printed if desired. The exit flag, FlagExit, is initially set equal to zero and changed to a value of unity when the total error, SumOfDs, is less than and allowing the relaxation to be terminated.

Figure 7 is the result for $\epsilon = 100$ when a = 30 and b = 20 which means that each of the 600 gridpoints is allowed to have, on average, a difference equal to 1/6 during two consecutive relaxations. The W values are found to be in the range of –154 and 24.8 for = 100. When is set equal to 1 which means that each gridpoint is allowed to have, on the average, a difference of 1/600, the mesh plot of W is shown in Figure 8. The W values are now all equal to or less than zero.

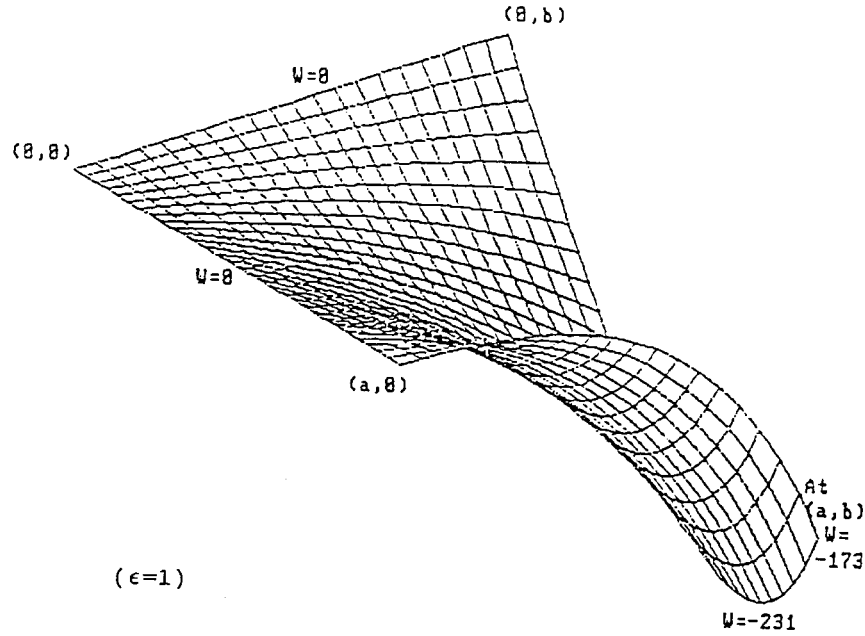


FIGURE 8.

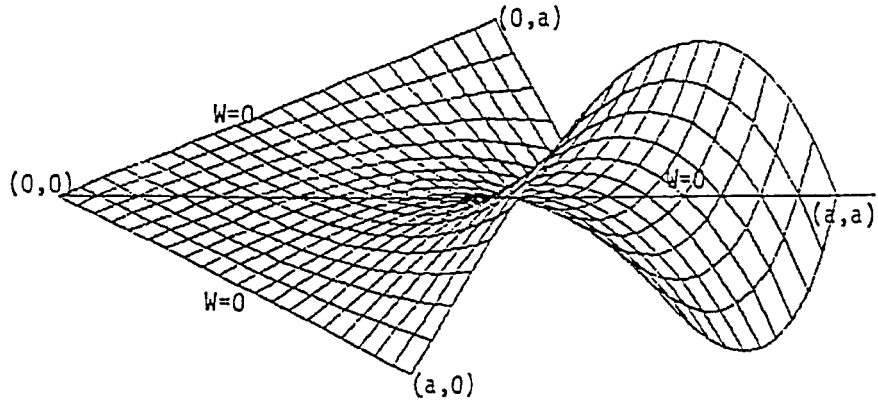


FIGURE 9.

It is noteworthy that if the cross section of the twisted rod is square, that is when a = b, there is no warping along the x = y line as illustrated by the mesh plot shown in Figure 9. This case is left as a homework problem for the readers to work out the details.

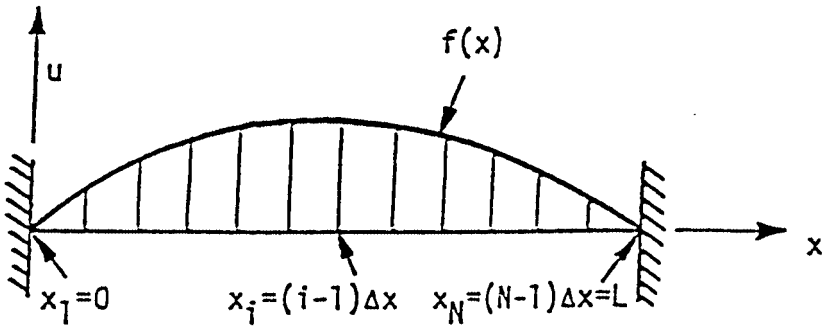


FIGURE 10. The problem of a tightened string.

8.4 PROGRAM WAVEPDE — NUMERICAL SOLUTION OF WAVE PROBLEMS GOVERNED BY HYPERBOLIC PARTIAL DIFFERENTIAL EQUATIONS

Program **WavePDE** is designed for numerical solution of the wave problems governed by the hyperbolic partial differential equation.[1] Consider the problem of a tightened string shown in Figure 10. The lateral displacement u satisfies the equation:

$$\frac{\partial^2 u}{\partial t^2} = a^2 \frac{\partial^2 u}{\partial x^2} \tag{1}$$

where x is the variable along the longitudinal direction of the string and a is the wave velocity related to the tension T in the string and the mass m of the string by the equation:

$$a^2 = \frac{T}{m} \tag{2}$$

Together with the governing equation of u, Equation 1, there are also so-called *initial conditions* prescribed which may be expressed as:

$$u(t=0,x) = f(x) \quad \text{and} \quad \frac{\partial u(t=0,x)}{\partial t} = v_0(x) \tag{3,4}$$

f(x) in Equation 3 describe the initial position while $v_0(x)$ describes the initial velocity of the tightened string. And, also there are so-called *boundary conditions* which in the case of a string at both ends are:

$$u(t, x = x_1 = 0) = 0 \quad \text{and} \quad u(t, x = x_N = L) = 0 \tag{5,6}$$

If the string is made of a single material, T/m would then be equal to a constant. Analytical solution can be found for this simple case. For the general case that the string may be composed of a number of different materials and the mass m is then a function of the spatial variable x. The more complicated the variation of these properties in x and t, the more likely no analytical solution possible and the problem can only be solved numerically.

The finite-difference approximation of Equation 1 can be achieved by applying the central difference for the second derivatives for both with respect to the space variable x and the time variable t. If we consider the displacement u only a finite number of stations, say N, defined with a spatial increment Δx and using a time increment of Δt, then specially, for t at t_i and x at x_j, we can have:

$$\frac{\partial^2 u}{\partial t^2} = \frac{u_{i-1,j} - 2u_{i,j} - u_{i+1,j}}{(\Delta t)^2}$$

and

$$\frac{\partial^2 u}{\partial x^2} = \frac{u_{i,j-1} - 2u_{i,j} + u_{i,j+1}}{(\Delta x)^2}$$

Substituting the above expressions into Equation 1, we obtain:

$$u_{i+1,j} = -u_{i=1,j} + 2u_{i,j} + \left(\frac{a\Delta t}{\Delta x}\right)^2 \left(u_{i,j-1} - 2u_{i,j} + u_{i,j+1}\right) \tag{7}$$

Equation 7 is to be applied for j = 2 through j = N–1. Initially for $t = t_1 = 0$, Equation 7 can be applied for i = 2, then $u_{i-1,j} \equiv u(t_1,x_j)$ term is simply $f(x_j)$ and can be calculated using Equation 3, and the $u_{i,j-1}$, $u_{i,j}$, and $u_{i,j+1}$ terms can be calculated using the forward-difference approximations of Equation 4 which are, for k = 2,3,…,N–1

$$u(t_2, x_k) = u(t_1, x_k) + v_0(x_k)\Delta t = f(x_k) + v_0(x_k)\Delta t \tag{8}$$

Since both f(x) and $v_0(x)$ are prescribed, use of Equation 7 enables all u to be calculated at $t = t_3$ and at all in-between stations x_j, for j = 2,3,…,N–1. When u values at $t = t_2$ and $t = t_3$ are completely known, Equation 7 can again be utilized to compute u at $t = t_4$ for i = 4 and so forth. Because the string is tightened, we expect the magnitudes of u values to continuously decrease in time. That is, for a specified tolerance, we may demand that the computation be terminated when

$$\max_{j=2\sim N-1} \left|u_{i,j}\right| < \varepsilon \tag{9}$$

It should be noted that the obtained u values are only for the selected increments Δt and Δx. Whether or not the results will change if either increment is reduced, need be tested.

Equation 7 relates the displacement at x_j at $t = t_{i+1}$ to the displacements at the same location x_j at two *previous* instants t_i and t_{i-1}, and also those of its left and right neighboring points at one time increment earlier, $t = t_i$. This is an approximation which ignores the influences of the displacements of its left and right neighboring points at the present time $t = t_{i+1}$, namely $u_{i+1,j-1}$ and $u_{i+1,j+1}$. These influences can be taken into consideration if the central-difference approximation for the curvature term in the x domain is applied for $t = t_{i+1}$ instead of at $t = t_i$ in derivation of Equation 7. Reader is urged to work Problem 5 to find the effect of this change.

A computer program called **WavePDE** has been developed for generating the deflection of the string at N stations for any specified time increment Δt until the deflection are almost all equal to zero throughout. The program allows interactive specification of the values of a, L, Δt, and Δx, and requires the user to define FUNCTION Subprograms F(X) and V0(X). Both FORTRAN and QuickBASIC versions are listed below along with a sample application.

QuickBASIC Version

```
function [SumOfDs,T]=Relaxatn(T)
        SumOfDs=0;
        for i=2:6
            for j=2:19
                Tsave=T(i,j);
                T(i,j)=.25*(T(i-1,j)+T(i+1,j)+T(i,j-1)+T(i,j+1));
                SumOfDs=abs(Tsave-T(i,j));
            end
        end
        for i=7:9
            Tsave=T(i,1);
            T(i,1)=.25*(T(i-1,1)+T(i+1,1)+2*T(i,2));
            SumOfDs=abs(Tsave-T(i,1));
            for j=2:19
                Tsave=T(i,j);
                T(i,j)=.25*(T(i-1,j)+T(i+1,j)+T(i,j-1)+T(i,j+1));
                SumOfDs=abs(Tsave-T(i,j));
            end
            if i>7 Tsave=T(i,20);
                   T(i,20)=.25*(T(i-1,20)+T(i+1,20)+2*T(i,19));
                   SumOfDs=abs(Tsave-T(i,20));
            end
        end
        Tsave=T(10,1);
        T(10,1)=.5*(T(9,1)+T(10,2));
        SumOfDs=abs(Tsave-T(10,1));
        for j=2:19
            Tsave=T(10,j);
            T(10,j)=.25*(2*T(9,j)+T(10,j-1)+T(10,j+1));
            SumOfDs=abs(Tsave-T(10,j));
        end
```

```
end
Tsave=T(10,20);
T(10,20)=.5*(T(9,20)+T(10,19));
SumOfDs=abs(Tsave-T(10,20));
```

Sample Application

The two function subprograms F and V0 listed in the program WavePDE are prepared for a string that has a length equal to 32 cm and is fastened at its two ends. At time t = 0, the string is lifted at x = 24 cm upward up 2 cm and then released from rest. That is, for Equations 3 and 4, we have a particular case of f(x) = x/12 in cm for $0 \le x \le 24$ cm and f(x) = 8(x/4) in cm for $24 < x \le 32$, and $v_0(x) = 0$. Suppose that the wave velocity, a in Equation 1, is equal to 90 cm/sec^2. We may be interested in knowing the lateral displacements at 15 stations between its two ends equally spaced at an increment of $\Delta x = 2$ cm for t>0. To perform the step-by-step calculation according to Equation 7, let us compute these displacements using a time increment of $\Delta t = 0.001$ second and proceed until t reaches 2 second or when the maximum displacement u_{max} is less than or equal to 0.001. In view of the small Δt, the results are to be printed after t has been increased by 0.1 second. An interactive run of the program WavePDE is presented below:

```
Program WavePDE - Wave Motion governed by Partial Differential Equation.
Have you created the supporting FUNCTION Subprograms F(X) & V0(X)?
Enter Y if they are done; otherwise press <Ctrl Break> : Y
Enter the value of a**2 : 8100
Enter the values of Delta t and Delta X : 0.001,2
Enter the number of in-between stations, N : 15
Enter the value of Epsilon : 1.e-1
Enter an ending time : 2
Enter a time increment for printing : 0.1

Time =      0.00000
      0.00000E+00     1.66667E-01     3.33333E-01     5.00000E-01     6.66667E-01
      8.33330E-01     1.00000E+00     1.16667E+00     1.33333E+00     1.50000E+00
      1.66667E+00     1.83333E+00     2.00000E+00     1.50000E+00     1.00000E+00
      5.00000E-01     0.00000E+00
Time =      0.10000
      0.00000E+00     1.66667E-01     3.33327E-01     4.99972E-01     6.66622E-01
      8.33034E-01     9.96975E-01     1.14341E+00     1.20900E+00     1.07281E+00
      7.94835E-01     6.92063E-01     4.98757E-01     3.59021E-01     1.31218E-01
      9.69535E-02     0.00000E+00
Time =      0.20000
      0.00000E+00     1.63165E-01     3.14672E-01     4.22497E-01     4.18258E-01
      2.40004E-01    -2.97670E-02    -1.84360E-01    -2.85290E-01    -5.15269E-01
     -6.31962E-01    -7.46900E-01    -7.49567E-01    -5.64395E-01    -2.87552E-01
     -1.67763E-01     0.00000E+00
Time =      0.30000
      0.00000E+00    -4.49038E-01    -8.17692E-01    -1.01222E+00    -1.11154E+00
     -1.30979E+00    -1.47038E+00    -1.47346E+00    -1452023E+00    -1.21779E+00
     -9.50311E-01    -8.11794E-01    -6.90822E-01    -5.14203E-01    -2.97612E-01
     -1.93379E-01     0.00000E+00
Time =      0.40000
      0.00000E+00    -4.20338E-01    -8.80266E-01    -1.25924E+00    -1.54300E+00
     -1.63233E+00    -1.48975E+00    -1.44211E+00    -1.36224E+00    -1.16807E+00
     -9.80121E-01    -8.28685E-01    -6.99694E-01    -4.55598E-01    -3.70615E-01
     -1.46329E-01     0.00000E+00
Time =      0.50000
      0.00000E+00     2.41942E-01     4.10856E-01     2.85292E-01     3.48808E-02
      1.04799E-02    -8.50767E-02    -3.54692E-01    -6.20998E-01    -7.73858E-01
     -8.03896E-01    -8.02862E-01    -6.00058E-01    -5.28044E-01    -3.08007E-01
     -1.77244E-01     0.00000E+00
```

```
Time =      0.60000
      0.00000E+00      9.71458E-02      3.40795E-01      4.81283E-01      6.95600E-01
      9.75958E-01      1.09782E+00      1.01151E+00      8.96584E-01      8.55300E-01
      6.80964E-01      5.46788E-01      2.18061E-01      5.21467E-02     -7.60440E-02
     -5.90514E-02      0.00000E+00
Time =      0.69999
      0.00000E+00      1.82785E-01      2.88539E-01      5.23221E-01      7.05613E-01
      8.19889E-01      9.58409E-01      1.11394E+00      1.38274E+00      1.59416E+00
      1.82468E+00      1.73587E+00      1.62435E+00      1.43918E+00      1.15483E+00
      6.62980E-01      0.00000E+00
Time =      0.79999
      0.00000E+00      2.06798E-01      2.97231E-01      4.80511E-01      7.23746E-01
      8.07089E-01      9.12777E-01      1.17106E+00      1.20114E+00      1.23721E+00
      1.09641E+00      8.92447E-01      6.88054E-01      5.65833E-01      4.80817E-01
      2.72947E-01      0.00000E+00
Time =      0.89999
      0.00000E+00      1.63375E-01      2.95364E-01      3.78608E-01      4.93753E-01
      4.38827E-01      2.94639E-01      7.43454E-02     -1.05731E-01     -4.12107E-01
     -5.38549E-01     -5.10644E-01     -5.64354E-01     -5.70825E-01     -5.37805E-01
     -3.22995E-01      0.00000E+00

Time =      0.99999
      0.00000E+00     -2.20420E-01     -4.59750E-01     -7.26144E-01     -1.05391E+00
     -1.15153E+00     -1.30936E+00     -1.33672E+00     -1.28539E+00     -1.31363E+00
     -1.19968E+00     -8.88411E-01     -5.39097E-01     -4.63126E-01     -3.50094E-01
     -1.97370E-01      0.00000E+00
Time =      1.10000
      0.00000E+00     -5.64074E-01     -9.91795E-01     -1.41255E+00     -1.64226E+00
     -1.80542E+00     -1.69046E+00     -1.47052E+00     -1.24728E+00     -1.03373E+00
     -1.06005E-00     -8.77848E-01     -6.51027E-01     -4.84975E-01     -3.61382E-01
     -1.27226E-01      0.00000E+00
Time =      1.20000
      0.00000E+00     -9.72051E-03      1.85691E-02      1.95219E-01      1.34458E-01
     -1.24663E-01     -4.45530E-01     -7.12247E-01     -8.02365E-01     -8.67517E-01
     -7.87895E-01     -7.98698E-01     -6.37735E-01     -4.78853E-01     -3.23825E-01
     -1.64963E-01      0.00000E+00
Time =      1.30000
      0.00000E+00      1.34638E-01      3.56986E-01      6.11524E-01      8.14188E-01
      9.11534E-01      8.98481E-01      9.45475E-01      8.16344E-01      6.76114E-01
      4.66450E-01      2.24011E-01     -6.47397E-02     -7.60462E-02     -7.50079E-02
     -1.34715E-01      0.00000E+00
Time =      1.40000
      0.00000E+00      2.27415E-01      4.29710E-01      4.54726E-01      5.69692E-01
      7.20758E-01      1.06781E+00      1.30335E+00      1.47015E+00      1.55998E+00
      1.66749E+00      1.64235E+00      1.53506E+00      1.37629E+00      1.02935E+00
      4.84091E-01      0.00000E+00
Time =      1.50001
      0.00000E+00      2.07191E-01      3.19376E-01      5.14515E-01      5.60746E-01
      8.68705E-01      1.07440E+00      1.12295E+00      1.08696E+00      1.18324E+00
      1.26952E+00      1.23464E+00      1.16266E+00      8.80416E-01      5.14480E-01
      2.67528E-01      0.00000E+00

Time =      1.60002
      0.00000E+00      1.21418E-01      3.38192E-01      3.67370E-01      4.97966E-01
      5.29223E-01      5.31940E-01      2.78686E-01      6.51729E-02     -4.83741E-02
     -2.87173E-01     -9.95912E-01     -8.37233E-01     -5.53924E-01     -1.56820E-01
     -2.05631E-01      0.00000E+00
Time =      1.70002
      0.00000E+00     -1.06744E-01     -2.69642E-01     -4.68614E-01     -7.33737E-01
     -9.10616E-01     -1.19498E+00     -1.31302E+00     -1.25113E+00     -1.20241E+00
     -1.08212E+00     -9.95912E-01     -8.37233E-01     -5.53924E-01     -1.56820E-01
     -1.65612E-01      0.00000E+00
Time =      1.80003
      0.00000E+00     -5.84820E-01     -1.20072E+00     -1.54521E+00     -1.65254E+00
     -1.72072E+00     -1.73798E+00     -1.57696E+00     -1.37549E+00     -1.15779E+00
     -9.02859E-01     -7.17306E-01     -7.00641E-01     -5.41795E-01     -3.65590E-01
     -1.65612E-01      0.00000E+00
Time =      1.90003
      0.00000E+00      7.24233E-02     -3.24723E-02     -1.85636E-01     -1.95257E-01
     -4.46355E-01     -5.88320E-01     -7.38593E-01     -9.34458E-01     -1.01370E+00
     -9.17116E-01     -7.40747E-01     -5.44435E-01     -4.80090E-01     -3.61236E-01
     -2.02691E-01      0.00000E+00
```

```
Time =     2.00004
     0.00000E+00     3.56727E-01     5.43352E-01     7.00064E-01     6.67818E-01
     8.18378E-01     8.30461E-01     8.55482E-01     7.11126E-01     4.01830E-01
     9.60796E-02    -4.12570E-02    -8.85658E-02    -1.59569E-01    -1.91057E-01
    -9.68752E-02     0.00000E+00
```

MATLAB Applications

A **MATLAB** version of **WavePDE.m** can be developed to run the sample problem as follows:

```
>>W=zeros(31,21); [W,n]=warping(30,20,31,21,100,W), mesh(W)

function [W,Nrelax]=Warping(a,b,M,N,Epsilon,W)
        dx=a/(M-1); dy=b/(N-1); Nrelax=1; ExitFlag=0;
        while ExitFlag==0;
          SumOfDs=0; Wsave=W(M,N);
          %   Eq. (26)
              W(M,N)=.5*(W(M,N-1)+W(M-1,N)+b*dx-a*dy);
          SumOfDs=SumOfDs+abs(Wsave-W(M,N));
          for jr=2:N-1; j=N+1-jr;
              Wsave=W(M,j);
          %   Eq. (25)
              W(M,j)=W(M-1,j)+(j-1)*dy*dx;
              SumOfDs=SumOfDs+abs(Wsave-W(M,j));
          end
          for ir=2:M-1; i=M+1-ir;
              Wsave=W(i,N);
          %   Eq. (24)
              W(i,N)=W(i,N-1)-(i-1)*dx*dy;
              SumOfDs=SumOfDs+abs(Wsave-W(i,N));
              for jr=2:N-1; j=N+1-jr;
                  Wsave=W(i,j);
              %   Eq. (8)
                  W(i,j)=.25*(W(i-1,j)+W(i+1,j)+W(i,j-1)+W(i,j+1));
                  SumOfDs=SumOfDs+abs(Wsave-W(i,j));
              end
          end
          if SumOfDs<Epsilon, ExitFlag=1; break
             else fprintf('No. of Sweep = %3.0f \n',Nrelax)
```

When this function is applied for generating displacements using a time increment of 0.001 and a storing increment of 0.1, the **MATLAB** commands, the data for plotting, and the resulting graph (Figure 2) having 21 curves each with 17 points are as follows:

```
>> format compact, UPlot=feval('a:WavePDE',8100,0,2,0.001,0.1,15,2,1.e-3)

UPlot =
  Columns 1 through 7
         0         0         0         0         0         0         0
    0.1667    0.1667    0.1632   -0.4490   -0.4204    0.2418    0.0971
```

```
    0.3333    0.3333    0.3148   -0.8175   -0.8804    0.4105    0.3406
    0.5000    0.5000    0.4227   -1.0120   -1.2594    0.2849    0.4811
    0.6667    0.6666    0.4185   -1.1152   -1.5432    0.0344    0.6953
    0.8333    0.8331    0.2402   -1.3096   -0.6326    0.0099    0.9756
    1.0000    0.9970   -0.0295   -1.4702   -1.4900   -0.0857    1.0973
    1.1667    1.1435   -0.1841   -1.4733   -1.4423   -0.3553    1.0109
    1.3333    1.2091   -0.2850   -1.4519   -1.3624   -0.6216    0.8959
    1.5000    1.0729   -0.5150   -1.2177   -1.1682   -0.7745    0.8546
    1.6667    0.7949   -0.6318   -0.9502   -0.9803   -0.8044    0.6803
    1.8333    0.6922   -0.7467   -0.8117   -0.8288   -0.8033    0.5461
    2.0000    0.4989   -0.7494   -0.6908   -0.6998   -0.6004    0.2175
    1.5000    0.3591   -0.5643   -0.5142   -0.4557   -0.5283    0.0517
    1.0000    0.1313   -0.2875   -0.2976   -0.3707   -0.3082   -0.0764
    0.5000    0.0961   -0.1677   -0.1934   -0.1463   -0.1773   -0.0592
         0         0         0         0         0         0         0
Columns 8 through 14
         0         0         0         0         0         0         0
    0.1828    0.2069    0.1636   -0.2201   -0.5642   -0.0102    0.1345
    0.2885    0.2974    0.2958   -0.4592   -0.9921    0.0177    0.3566
    0.5232    0.4808    0.3792   -0.7254   -1.4129    0.1941    0.6109
    0.7056    0.7241    0.4946   -1.0531   -1.6427    0.1331    0.8133
    0.8198    0.8075    0.4398   -1.1506   -1.8059   -0.1262    0.9104
    0.9584    0.9133    0.2958   -1.3084   -1.6910   -0.4472    0.8971
    1.1139    1.1717    0.0756   -1.3359   -1.4710   -0.7139    0.9439
    1.3827    1.2018   -0.1045   -1.2846   -1.2477   -0.8039    0.8147
    1.5941    1.2380   -0.4109   -1.3130   -1.0340   -0.8689    0.6744
    1.8246    1.0972   -0.5374   -1.1992   -1.0603   -0.7891    0.4647
    1.7357    0.8933   -0.5069   -0.8880   -0.8781   -0.7997    0.2224
    1.6242    0.6888   -0.5635   -0.5388   -0.6512   -0.6385    0.0661
    1.4390    0.5665   -0.5702   -0.4629   -0.4851   -0.4795   -0.0771
    1.1547    0.4813   -0.5374   -0.3499   -0.3615   -0.3242   -0.0758
    0.6629    0.2732   -0.3299   -0.1973   -0.1273   -0.1652   -0.1351
         0         0         0         0         0         0         0
Columns 15 through 21
         0         0         0         0         0         0         0
    0.2274    0.2074    0.1218   -0.1063   -0.5849    0.0718    0.3564
    0.4297    0.3197    0.3389   -0.2687   -1.2009   -0.0337    0.5427
    0.4546    0.5150    0.3684   -0.4673   -1.5454   -0.1875    0.6989
    0.5696    0.5614    0.4994   -0.7321   -1.6529   -0.1975    0.6663
    0.7206    0.8695    0.5310   -0.9089   -1.7211   -0.4488    0.8165
    1.0677    1.0753    0.5339   -1.1932   -1.7384   -0.5908    0.8282
    1.3031    1.1240    0.2809   -1.3114   -1.5774   -0.7410    0.8529
    1.4698    1.0881    0.0675   -1.2496   -1.3760   -0.9367    0.7084
    1.5595    1.1846   -0.0461   -1.2011   -1.1582   -1.0517    0.3991
    1.6669    1.2710   -0.2850   -1.0810   -0.9032   -0.9189    0.0934
    1.6417    1.2361   -0.4508   -0.9951   -0.7176   -0.7422   -0.0436
    1.5345    1.1640   -0.5788   -0.8366   -0.7009   -0.5456   -0.0906
    1.3757    0.8816   -0.6134   -0.5535   -0.5419   -0.4810   -0.1611
    1.0289    0.5153   -0.4800   -0.1565   -0.3658   -0.3619   -0.1921
    0.4838    0.2679   -0.2052    0.0041   -0.1657   -0.2030   -0.0974
         0         0         0         0         0         0         0
>> plot(Uplot), text(13.5,1.7,'(1)'), text(15,0.1,'(2)'), text(4.8,0.2,'(3)')
>> text(3.2,-0.9,'(4)')
```

Notice that solid, broken, dotted, and dot center lines and in that order are used repeatedly for plotting the 21 curves in Figure 11. The first four curves have been marked using the text command. Form the 21st column listed above, we observe that maximum initial displacement of 2 cm has only been reduced to 0.8529 cm after two seconds. It should be pointed out that in **WavePDE.m** the computation should also be ended when the sum of the absolute values of displacements is less than the specified tolerance.

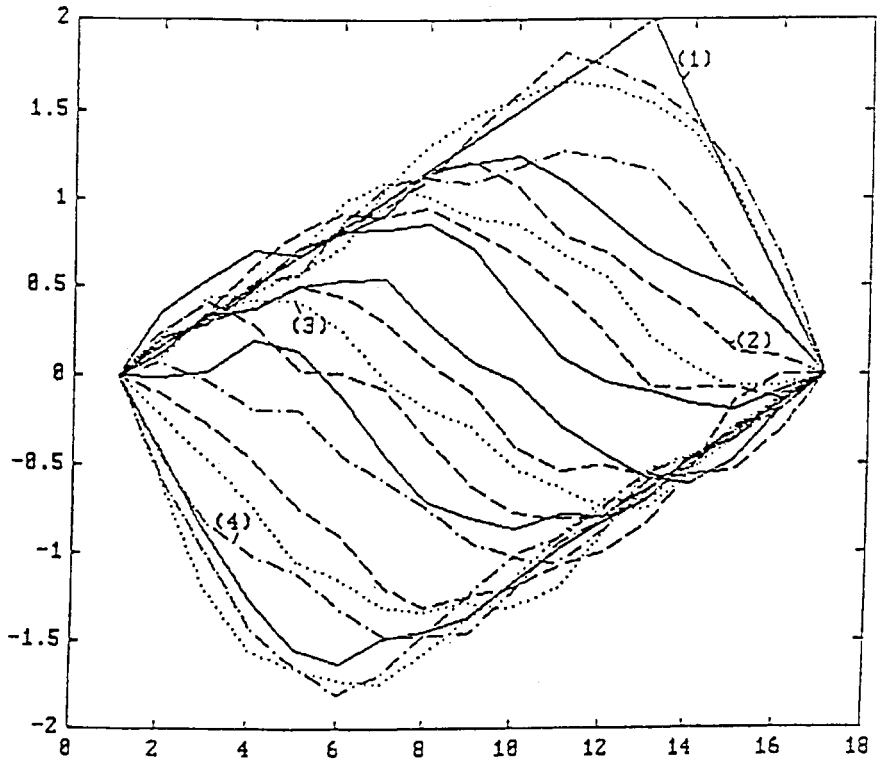


FIGURE 11. Solid, broken, dotted, and dot center lines are used repeatedly for plotting the 21 curves in this figure.

MATHEMATICA APPLICATION

Mathematica can be applied to investigate the string vibration problem by the following interactive operations:

In[1]: = f[fx_]: = If[x>24, 8x/4, x/12]

In[2]: = (asq = 81000; dt = 0.001; dx = 2: n = 15; eps = .1; tend = 2; dtp = 0.1;
v0 = 0; np = 100; c = (dt/dx)^2*asq; exitflagg = 0; t = 0)

In[3]: = u0 = Table[0.{j,n + 2}]; Do[x = (j–1)*dx; u0[[j]] = f[x];,{j,n + 2}];

In[4]: = (Print["t = ",N[t,3]," String's Displacements are: "];
Print[N[u03]])

Out[4]: = t = 0 String's Displacements are:
{0, 0.167, 0.333, 0.5, 0.667, 0.833, 1., 1.17, 1.33, 1.5, 1.67, 1.83,
2., 1.5, 1., 0.5, 0}

In[5]: = t = t + dt; u = u0 + dt*v0; un = Table[0.{j,n + 2}]; nc = 2;

In[6]: = (While[exitflag == 0, usum = 0; Do[un[[j + 1]] =-u0[[j + 1]] + c*u[[j]]
+ 2*(1–c)*u[[j + 1]] + c*u[[j + 2]];
usum = usum + Abs[un[[j + 1]]];,{j,n}];
t = t + dt; If[(usum<eps)||(t>tend), exitflag = 1; Break,
u0 = u; u = un;];
If[nc = = np, Print["t = ",N[t,3],
" String's Displacements are: "];
Print[N[u,3]]; nc = 1;, nc = nc + 1]])

Out[6]: = t = 0.1 String's Displacements are:
{0. 0.167, 0.333, 0.5, 0.667, 0.833, 0.997, 1.14, 1.21, 1.07, 0.795,
0.692, 0.499, 0.359, 0.131, 0.0961, 0}

t = 0.2 String's Displacements are:
{0. 0.163, 0.315, 0.423, 0.418, 0.24, –0.0295, –0.184, –0.285,
–0.515, –0.632, –0.474, –0.749, –0.564, –0.287, –0.168, 0}

t = 0.3 String's Displacements are:
{0. –0.449, –0.818, –1.01, –1.12, –1.31, –1.47, –1.47, –1.45, –1.22,
–0.95, –0.812,–0.691, –0.514, –0.298, –0.193, 0}

t = 0.4 String's Displacements are:
{0. –0.42, –0.88, –1.26, –1.54, –1.63, –1.49, –1.44, –1.36, –1.17,
–0.98, –0.829, –0.7, –0.456, –0.371, –0.146, 0}

t = 0.5 String's Displacements are:
{0. 0.242, 0.411, 0.285, 0.0344, 0.00989, –0.0857, –0.355, –0.0622,
–0.774, –0.804, –0.803, –0.6, –0.528, –0.308, –.177, 0}

t = 0.6 String's Displacements are:
{0. 0.0971, 0.341, 0.481, 0.695, 0.976, 1.1, 1.01, 0.896, 0.855,
0.68, 0.546, 0.217, 0.0517, –0.0764, –0.0592, 0}

t = 0.7 String's Displacements are:
{0. 0.183, 0.289, 0.523, 0.706, 0.82, 0.958, 1.11, 1.38, 1.59, 1.82,
1.74, 1.62, 1.44, 1.15, 0.663, 0}

t = 0.8 String's Displacements are:
{0. 0.207, 0.297, 0.481, 0.724, 0.808, 0.913, 1.17, 1.2, 1.24, 1.1,
0.893, 0.689, 0.567, 0.481, 0.273, 0}

t = 0.9 String's Displacements are:
{0. 0.164, 0.296, 0.379, 0.495, 0.44, 0.296, 0.0756, –0.104, –0.411,
–0.537, –0.51, –0.563, –0.57, –0.537, –0.323, 0}

t = 1.0 String's Displacements are:
{0. –0.22, –0.459, –0.725, –1.05, –1.15, –1.31, –1.34, –1.28, –1.31
–1.2, –0.888, –0.539, –0.463, –0.53, –0.197, 0}

t = 1.1 String's Displacements are:
{0. –0.564, –0.992, –1.41, –1.64, –1.81, –1.69, –1.47, –1.25, –1.03,
–1.06, –0.878, –0.651, –0.485, –0.361, –0.127, 0}

t = 1.2 String's Displacements are:
{0. –0.0102, 0.0177, 0.194, 0.133, –0.126, –0.447, –0.714, –0.804,
–0.869, –0.789, –0.8, –0.639, –0.479, –0.324, –0.165, 0}

t = 1.3 String's Displacements are:
{0. 0.134, 0.357, 0.611, 0.813, 0.91, 0.897, 0.944, 0.815, 0.674,
0.465, 0.222, –0.0661, –0.0771, –0.0758, –0.135, 0}

t = 1.4 String's Displacements are:
{0. 0.227, 0.43, 0.455, 0.57, 0.721, 1.07, 1.3, 1.47, 1.56, 1.67,
1.64, 1.53, 1.38, 1.03, 0.484, 0}

t = 1.5 String's Displacements are:
{0. 0.207, 0.32, 0.515, 0.561, 0.869, 1.08, 1.12, 1.09, 1.18, 1.27,
1.24, 1.16, 0.882, 0.515, 0.268, 0}

t = 1.6 String's Displacements are:
{0. 0.122, 0.339, 0.368, 0.499, 0.531, 0.534, 0.281, 0.0675, –0.0461,
–0.285, –0.451, –0.579, –0.613, –0.48, –0.205, 0}

t = 1.7 String's Displacements are:
{0. –0.106, –0.269, –0.467, –0.732, –0.909, –1.19, –1.31, –1.25,
–1.2, –1.08, –0.995, –0.837, –0.533, –0.156, 0.00405, 0}

t = 1.8 String's Displacements are:
{0. –0.585, –1.2, –1.55, –1.65, –1.72, –1.74, –1.58, –1.38, –1.66,
–0.903, –0.718, –0.701, –0.542, –0.366, –0.166, 0}

t = 1.9 String's Displacements are:
{0. 0.0718, –0.0337, –0.188, –0.197, –0.4499, –0.591, –0.741, –0.937,
–1.02, –0.919, –0.742, –0.546, –0.481, –0.362, –0.203, 0}

t = 2.0 String's Displacements are:
{0. 0.0356, 0.543, 0.699, 0.666, 0.816, 0.828, 0.853, 0.708, 0.399,
0.0934, –0.0436, –0.0906, –0.161, –0.192, –0.0974, 0}

The results are in complete agreement with those obtained previously.

8.5 PROBLEMS

PARABPDE

1. Expand the program **ParabPDE** to print out the computed temperature distribution along the entire rod only when the temperature at an interactively specified location reaches an interactively entered value. The time required to reach this condition should be printed as shown below. Call this new program **ParaPDE1**.

   Enter the x value at which a desired temperature is to be specified: 6.0
   Enter the temperature to be reached, in °F: 50
   It takes X.XXXXXE + 00 seconds.

   Notice that a format of E13.5 is to be used to print out the time.
2. Consider the transient heat-conduction problem where $k/c\rho = 0.00104$ ft^2/sec, $\Delta t = 1$ sec, $\Delta x = 1$", and at the left end of the rod, $x = 0$, the temperature u is equal to 50°F initially but equal to 0°F for $t>0$. Compute the temperatures at $x = 1$", 2", and 3" when $t = 1$, 2, and 3 seconds.
3. Use the same data as in the program **ParabPDE**, but modify the program for the case when the right end is not insulated but is maintained at 0°F until $t = 100$ seconds and then is heated at 100°F afterwards. Print out the times required for the station at the mid-length reaches to 10°F, 20°F, at 10°F increments until 100°F.
4. Study the effect of changing the value of DT in the program **ParabPDE** on t_{final}, the time required for reaching the uniform distribution of 100°F throughout the rod. Tabulate DT vs. t_{final}.
5. Change the steady-state temperature from 100°F to 75°F and generate a plot similar to that shown in Figure 2.
6. For the rod shown in Figure 1, one-half of the surface insulation, for $x/L = 0.4$ to $x/L = 0.7$, is to be removed and the temperature for that portion of the rod is to be maintained at 70°F. Modify the program **ParabPDE** is accommodate for the computation needed to determine how long it takes to reach a stable temperature distribution.
7. Apply **MATLAB** and **Mathematica** (using **ParaPDE.m**) to solve the transient heat-conduction problem by decreasing Δx from 1 to 0.5.

RELAXATN

1. For the problem treated by the program **Relaxatn** except that the temperature is equal to 100°F along the top boundary, perform the relaxation by upgrading T starting from the top boundary instead of from the bottom boundary to expedite the convergence. Modify the program **Relaxatn** to perform this new sweeping process.
2. Round the right lower corner of the square plate with a radius equal to 5 and consider this corner as an insulated boundary. Modify the program

Relaxatn and rerun the problem to print out the converged temperature distribution.

3. Initially, the temperatures are assumed to be equal to zero everywhere in the plate shown in Figure 12 except those on the boundary. Carry out one complete relaxation (starting from the top row and from left to right, and then down to the next row and so on) cycle to upgrade the unknown temperatures.

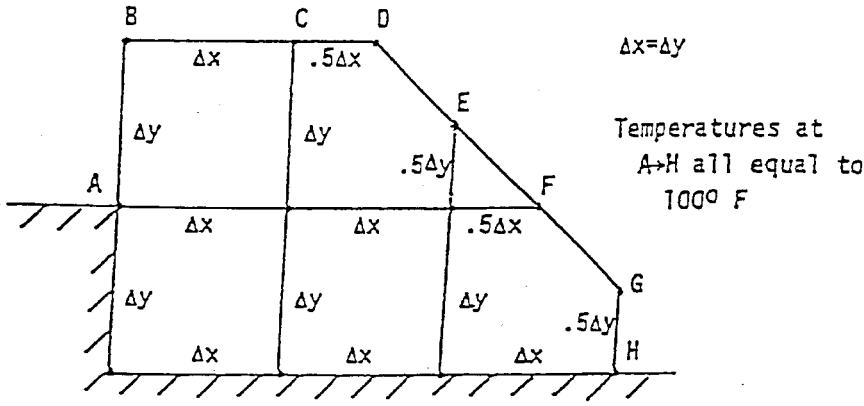


FIGURE 12. Problem 3.

4. Complete the derivation of relaxation equations for the temperatures at the points B and C shown in Figure 2 by use of Equation 18 and by considering the cord BC only. Derive the equation for the point B by averaging the effects of both cords AB and BC.
5. For the problem described in Problem 3, derive a matrix equation of order 5 for direct solution of the five unknown temperatures (two between points A and F, and three along the bottom insulted boundary) based on the finite-difference formulas, Equations 3, 6, 8, 10, 11, or, 12. Compare the resulting temperature distribution with that obtained in Problem 3.
6. Rework Problem 3 if the boundary DEFG is insulated but T_D and T_G remain equal to 100°F.
7. Initially, the temperatures are assumed to be equal to zero everywhere in the shown in Figure 13 except those on the boundary. Carry out one complete relaxation (starting from the top row and from left to right, and then down to the next row and so on) cycle to upgrade the unknown temperatures. The points A, B, and D are on a straight line.
8. Rework Problem 7 if the boundary ABCD is insulated but T_D remains equal to 100°F.
9. Following the same process as in Problem 8, but obtain the direct solution of the temperature distribution for Problem 7.

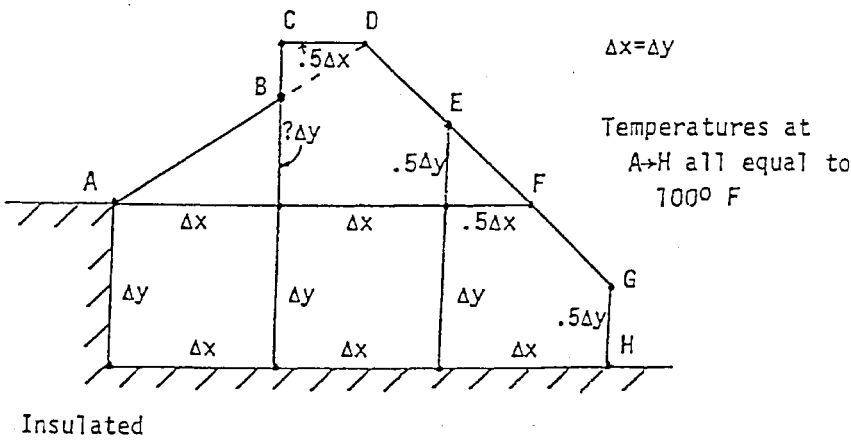


FIGURE 13. Prtoblem 7.

10. The warping of a twisted bar of uniform rectangular cross section already depicted by the mesh plot shown in Figures 5 and 6 also can be observed using the contour plotting capability of **MATLAB**. Apply program **Warping.m** and the command contour to generate a contour plot for a = 30 and b = 20 using = 100 and = 1 (Figures 14A and 14B, respectively).

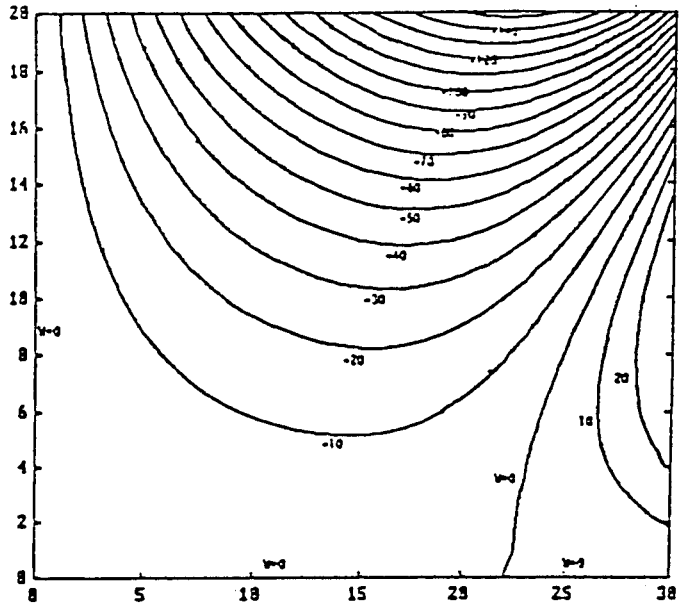


FIGURE 14A. Problem 10.

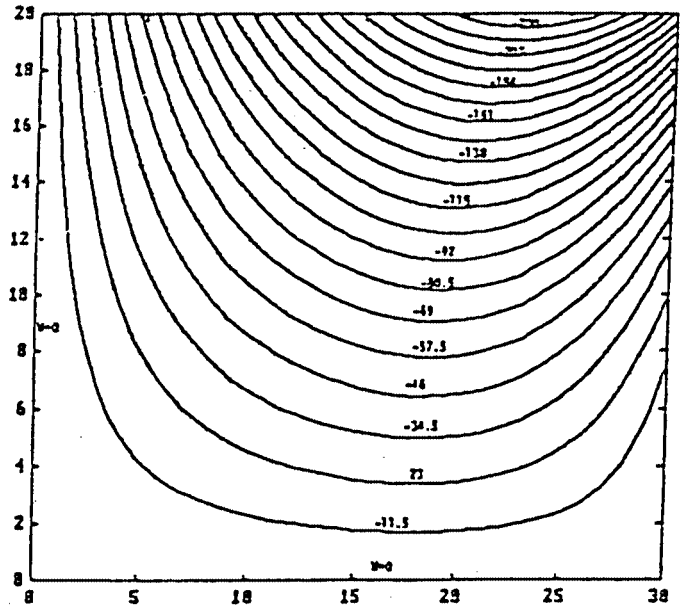


FIGURE 14B. Problem 10.

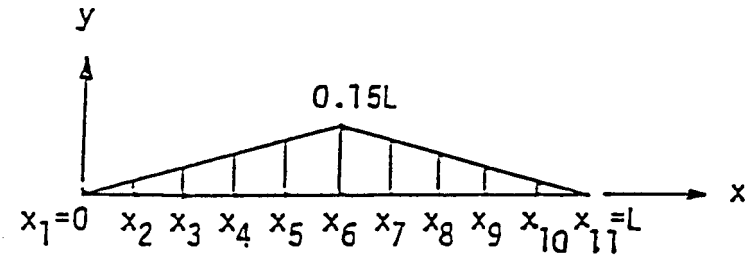


FIGURE 15. Problem 4.

11. Direct solution of the warping function W(X,Y) can also be obtained following the procedure described in Problem 8. For a rectangular cross section ($-a \le X \le a$ and $-b \le Y \le b$) of a twisted rod, the warping W(X,Y) needs to be found only for the upper right quadrant $0 \le X \le a$ and $0 \le Y \le b$ which in general can be divided into a gridwork of (M + 1)x(N + 1). The antisymmetry conditions W(X = 0,Y) = W(X,Y = 0) = 0 reduces to only (M + 1)x(N + 1)-(M + N + 1) = MxN unknowns, i.e., only solving those W's for X>0. That means we have to derive a system of MxN linear algebraic equations: along the boundaries X = a and Y = b, Equations 23 to 25 are to be used and for the interior grid points (0<X<a and 0<Y<b), Equation 3 is to be used. Generate such a matrix equation and then apply

the program Gauss to find these MxN W's. Compare the resulting W distribution with those obtained by the relaxation method shown in Figures 5 and 6 for a = 30 and b = 20.

12. Same as Problem 11 except for the case a = b = 20 and for comparing with Figure 7.
13. Solve the warping problem by **Mathematica**.

WavePDE

1. For the string problem analyzed in the Sample Application, modify the program slightly so that the times required for the string to have the magnitudes of its maximum displacements reduced to 0.8, 0.6, 0.4, and 0.2, and the corresponding deflected shapes can be printed.
2. Rearrange the subprogram FUNCTION **F** in the program **WavePDE** to consider the case of an initial, upward lifting the mid-third (8≤x≤16 cm) of the string by 1 cm. Rerun the program using the same input data as in the Sample Application.
3. Consider a string which is composed of two different materials even though it is subjected to a uniform tension T so that the left and right one-thirds (i.e., , 0≤x≤8 cm and 24≤x≤32 cm, respectively) of the string has a wave velocity a = 80 cm/sec while its mid-third (i.e., 8≤x≤16 cm) has a wave velocity a = 90 cm/sec. Modify and then rerun program **WavePDE** using the other input same as in the Sample Application.
4. A tightened string of length L equal to 1 ft is lifted as shown in 15 and is released with a velocity distribution $v = \partial y(t = 0,x)/\partial t = 2\sin\pi x/L$ in ft/sec. If the constant T/m appearing in Equation 2 is equal to 8,100 ft^2/sec^2, use a time increment $\Delta t = 0.0005$ sec and a space increment $\Delta x = 0.1L$ and apply Equation 7 to find the y values at t = 0.001 sec and for the stations x_2 and x_3.
5. In approximating Equation 1 by finite differences, we may keep the same approach for $\partial^2u/\partial t^2$ as in deriving Equation 7 but to apply the second central-difference formula for $\partial^2u/\partial t^2$ not at $t = t_i$ but at $t = t_{i+1}$. The resulting equation, for $C = (a\Delta t/\Delta x)^2$, is:

$$-Cu_{i+1,j-1} + (2+C)u_{i+1,j} - Cu_{i+1,j+1} = -u_{i-1,j} + 2u_{i,j}$$

   Derive a matrix equation for solving the unknowns u_j for j = 1,2,…,N–1, at $t = t_{i+1}$. Note that the boundary conditions are $u_{i+1,0} = u_{i+1,N} = 0$. Write a program **WavePDE.G** which uses the Gaussian Elimination method to solve this matrix equation and run it for the Sample problem to compare the results.
6. Change the **MATLAB** m file **WavePDE** to solve Problem 4.
7. Apply **Mathematica** to solve Problem 4.

8.6 REFERENCES

1. C. R. Wylie, Jr., *Advanced Engineering Mathematics*, Chapter 9, Second Edition, McGraw-Hill, New York, 1960.
2. J. P. Holman, *Heat Transfer*, McGraw-Hill, New York, 1963.
3. S. Timoshenko and J. N. Goodier, *Theory of Elasticity*, Chapter 11, Second Edition, McGraw-Hill, New York, 1951.