Numerical Solution of Laplace Equation

By Gilberto E. Urroz, October 2004

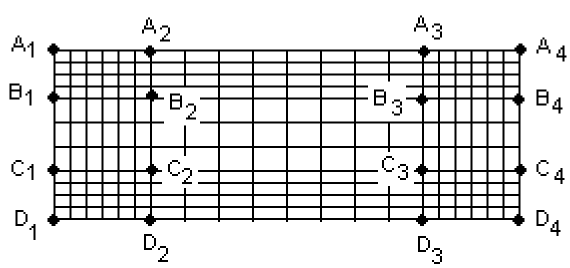
Laplace equation governs a variety of equilibrium physical phenomena such as temperature distribution in solids, electrostatics, inviscid and irrotational two-dimensional flow (potential flow), and groundwater flow. In order to illustrate the numerical solution of the Laplace equation we consider the distribution of temperature in a two-dimensional, rectangular plate, where the temperature is maintained at given values along the four boundaries to the plate (i.e., Dirichlet-type boundary conditions). The Laplace equation, for this case, is written as

$$\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} = 0 . \qquad [0]$$

NOTE: While temperature distributions in solids are not of interest to most civil engineering applications, this situation provides a relatively simple physical phenomena that can be analyzed with Laplace's equation.

Temperature distribution in a rectangular plate

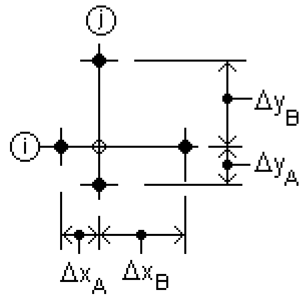
The figure below represents a thin rectangular solid body whose temperature distribution is to be determined by the solution of Laplace's equation.. The rectangular body is covered with a computational grid as shown in the figure.



Notice that we have arbitrarily divided the domain into 9 sub-domains (e.g., $A_1A_2B_1B_2$, etc.) in such a way that each sub-domain contains points on a grid with constant increments Δx_i and Δy_i. Here, the sub-index i describes a given sub-domain, e.g., sub-domain $A_1A_2B_1B_2$ could be identified as $i=1$, while sub-domain $A_2A_3B_2B_3$ would be $i = 2$, etc. Dirichlet boundary conditions for this case requires us to specify the temperature along the boundaries A_1A_4, A_1D_1, D_1D_4, and A_4D_4.

Finite difference approximation

To produce a numerical solution, we proceed to find the most general finite-difference approximation for the equation on a given interior grid point. We will focus in a point such as B_2 (or B_3, C_2, C_3), which represents the border point of four different sub-domains in the diagram above. The reason for selecting one of these points is that, at that point, the grid has different increments in both x and y, thus, being the most general case possible. This situation is illustrated in the following figure:



In order to find an approximation for the derivative T_{xx} we use the following equations:

$$T_{i-1,j} = T_{i,j} - T_x\,\Delta x_A + \frac{1}{2} T_{xx}\,\Delta x_A^{\,2} \qquad [1]$$

$$T_{i+1,j} = T_{i,j} + T_x\,\Delta x_B + \frac{1}{2} T_{xx}\,\Delta x_B^{\,2} \qquad [2]$$

Subtracting equation [1] from equation [2], and solving for $T_x = \partial T/\partial x$ while neglecting the terms involving $T_{xx} = \partial^2 T/\partial x^2$, results in

$$T_x = \frac{T_{i+1,j} - T_{i-1,j}}{\Delta x_B + \Delta x_A}. \qquad [3]$$

Adding equations [1] and [2], and solving for $T_{xx} = \partial^2 T/\partial x^2$, after replacing the expression for T_x, from [3], results in the following expression:

$$T_{xx} = \frac{2\left(T_{i-1,j} + T_{i+1,j} - 2\,T_{i,j} + \frac{(T_{i+1,j} - T_{i-1,j})\,(\Delta x_A - \Delta x_B)}{\Delta x_B + \Delta x_A}\right)}{\Delta x_A^{\,2} + \Delta x_B^{\,2}}$$

To simplify the expression we introduce the following definitions:

$$\alpha_x = \Delta x_A - \Delta x_B,\ \ \beta_x = \Delta x_A + \Delta x_B,\ r_x = \alpha_x / \beta_x,\ \ \gamma_x^2 = \Delta x_A^2 + \Delta x_B^2. \qquad [4].$$

Thus,

$$T_{xx} = \frac{2\,(T_{i-1,j} + T_{i+1,j} - 2\,T_{i,j} + r_x\,(T_{i+1,j} - T_{i-1,j}))}{\gamma_x^{\,2}} \qquad [5].$$

Similarly, by using a Taylor series expansion in y, we can obtain the following expression for the following derivatives in y:

$$T_y = \frac{T_{i,j+1} - T_{i,j-1}}{\Delta y_A + \Delta y_B} \qquad [6]$$

and

$$T_{yy} = \frac{2\,(T_{i,j-1} + T_{i,j+1} - 2\,T_{i,j} + r_y\,(T_{i,j+1} - T_{i,j-1}))}{\gamma_y^{\,2}} \qquad [7]$$

Where,

$$\alpha_y = \Delta y_A - \Delta y_B,\;\; \beta_y = \Delta y_A + \Delta y_B,\; r_y = \alpha_y / \beta_y,\;\; \gamma_y^2 = \Delta y_A^{\,2} + \Delta y_B^{\,2}. \qquad [8].$$

If we now replace the results of equations [5] and [7] into the Laplace equation, namely, $T_{xx}+T_{yy} = 0$, results in the following finite-difference approximation:

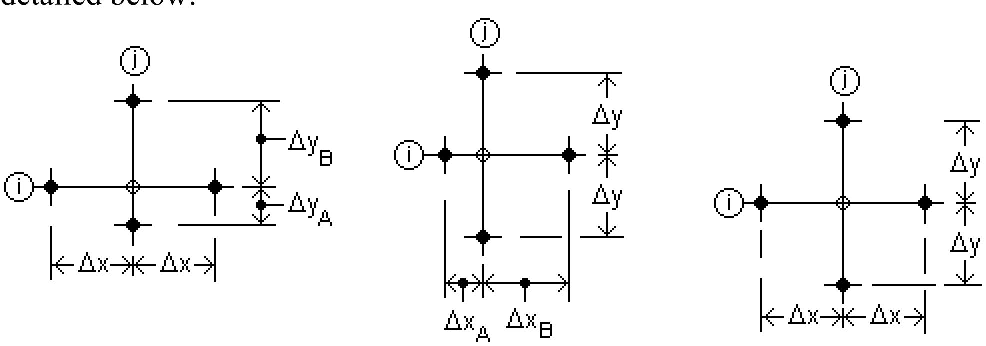
$$\frac{T_{i-1,j} + T_{i+1,j} - 2\,T_{i,j} + r_x\,(T_{i+1,j} - T_{i-1,j})}{\gamma_x^{\,2}} + \frac{(T_{i,j-1} + T_{i,j+1} - 2\,T_{i,j} + r_y\,(T_{i,j+1} - T_{i,j-1}))}{\gamma_y^{\,2}} = 0 \qquad [9]$$

Explicit solution to the finite-difference equation

An *explicit* solution for the value of the unknown T_{ij} at the center of the computational cell can be obtained from equation [9]:

$$T_{ij} = \frac{\gamma_y^2 (T_{i-1,j} + T_{i+1,j} + r_x (T_{i+1,j} - T_{i-1,j})) + \gamma_x^2 (T_{i,j-1} + T_{i,j+1} + r_y (T_{i,j+1} - T_{i,j-1}))}{2(\gamma_x^2 + \gamma_y^2)} \qquad [10]$$

As indicated earlier, the result in equation [10] represents the most general case for the explicit solution for a node in the computational domain with different increments in both x and y. With reference to the grid shown earlier, this corresponds to one of these points: B_2, B_3, C_2, or C_3. Other possibilities to consider are illustrated in the following figure and detailed below:



(a) Uniform x increment (b) Uniform y increment (c) Uniform grid in x,y

- Along lines B_1B_2, B_2B_3, B_3B_4, C_1C_2, C_2C_3, and C_3C_4 in page 1(except for the extreme points), where the values of the increment in x remain constant (see case (a) in the figure above): $\Delta x_A = \Delta x_B = \Delta x,\ r_x = \alpha_x = 0,\ \beta_x = 2\Delta x,\ \gamma_x^2 = 2\Delta x^2$, and

$$T_{ij} = \frac{\gamma_y^2 (T_{i-1,j} + T_{i+1,j}) + 2\Delta x^2 (T_{i,j-1} + T_{i,j+1} + r_y (T_{i,j+1} - T_{i,j-1}))}{2(2\Delta x^2 + \gamma_y^2)} \qquad [11]$$

- Along lines A_2B_2, B_2C_2, C_2D_2, A_3B_3, B_3C_3, and C_3D_3 in page 1 (except for the extreme points), where the values of the increment in x remain constant, (see case (b) in the figure above): $\Delta y_A = \Delta y_B = \Delta y,\ r_y = \alpha_y = 0,\ \beta_y = 2\Delta y,\ \gamma_y^2 = 2\Delta y^2$, and

$$T_{ij} = \frac{2\Delta y^2 (T_{i-1,j} + T_{i+1,j} + r_x (T_{i+1,j} - T_{i-1,j})) + \gamma_x^2 (T_{i,j-1} + T_{i,j+1})}{2(\gamma_x^2 + 2\Delta y^2)} \qquad [12]$$

- In the interior points of any of the nine sub-domains shown in the grid diagram above, except for the boundary lines (see case (c) in the figure above):

$$\Delta x_A = \Delta x_B = \Delta x,\ r_x = \alpha_x = 0,\ \beta_x = 2\Delta x,\ \gamma_x^2 = 2\Delta x^2,$$
$$\Delta y_A = \Delta y_B = \Delta y,\ r_y = \alpha_y = 0,\ \beta_y = 2\Delta y,\ \gamma_y^2 = 2\Delta y^2,$$

  and equation [10] simplifies to

$$T_{ij} = \frac{T_{i-1,j} + T_{i+1,j} + \beta^2 (T_{i,j+1} + T_{i,j+1})}{2(1+\beta^2)}, \qquad [13]$$

  where, $\beta = \Delta x/\Delta y$.

Explicit solution for uniform grid increments

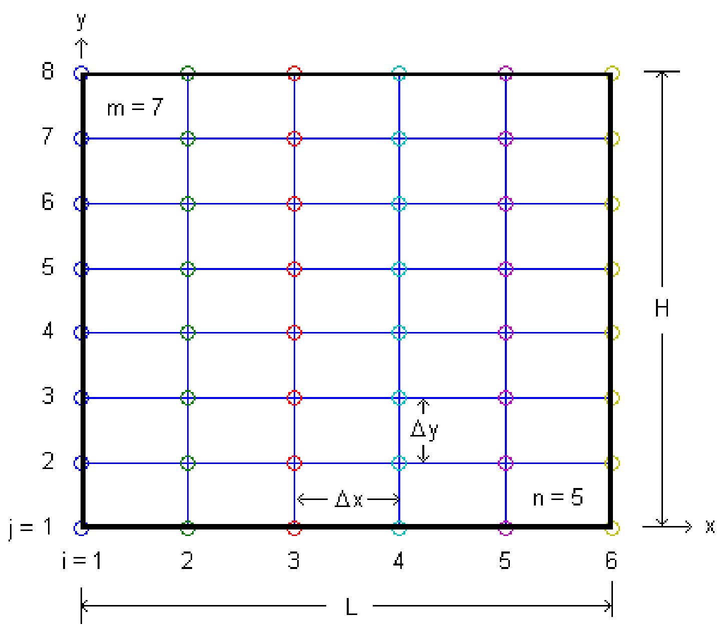
Consider a rectangular domain where the increments in both x and y are uniform. The appropriate equation to use for an explicit solution to the Laplace equation is equation [13]. The solution will start by loading the boundary conditions, and then calculating the values of T_{ij} in the interior points of the domain. While we are initially tempted to calculate T_{ij} only once with equation [13], it should be mentioned that these values are only a fist approximation to the solution. In practice, an iterative process must be performed to improve the results until each value T_{ij} converges to a solution. We should, therefore, add an additional index, k, representing the current iteration, to each solution value. The solution values will now be referred to as T_{ij}^k, and equation [13] will be modified to read:

$$T_{i,j}^{k+1} = \frac{T_{i-1,j}^k + T_{i+1,j}^k + \beta^2 (T_{i,j-1}^k + T_{i,j+1}^k)}{2(1+\beta^2)}. \qquad [13\text{-a}]$$

The iterative process should be repeated until convergence is achieved in every interior point of the domain, or until a maximum number of iterations, say, 1000, have been performed. Convergence can be achieved, for example, if, given a tolerance value ε, the maximum difference between two consecutive iterations is less than the tolerance, i.e., if

$$\max_{i,j} | T_{i,j}^{k+1} - T_{i,j}^{k} | \leq \varepsilon .$$

Consider, as an example, a rectangular domain of length $L = 5\ cm$, and height $H = 3.5\ cm$, with increments $\Delta x = 1\ cm$, and $\Delta y = 0.5\ cm$, as illustrated in the figure below.



There will be $n = L/\Delta x$ sub-intervals in x, and $m = H/\Delta y$ sub-intervals in y, with

$$x_i = (i-1)\ \Delta x, \text{ for } i = 1, 2, \ldots, n+1,$$

and

$$y_j = (j-1)\Delta y, \text{ for } j = 1, 2, \ldots, m+1.$$

The boundary conditions are given as follows: $T_{ij} = 5$ along the left and right sides of the domain, while the temperatures are given by the function $T_b(x) = 5 \cdot x \cdot (1-x)$ for the top and bottom sides of the domain, respectively.

Solution is achieved by using function *LaplaceExplicit.m* in Matlab :

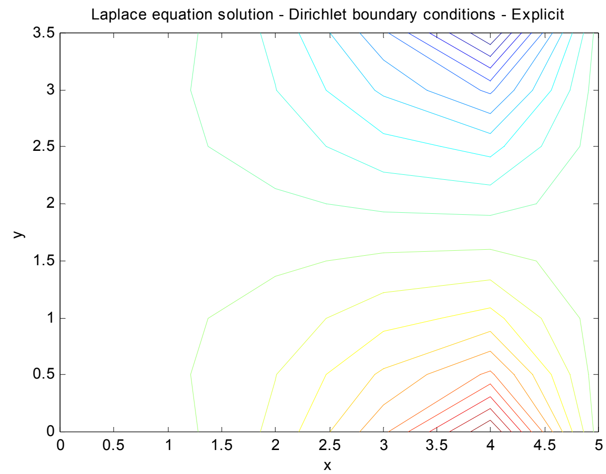
```
function [x,y,T]= LaplaceExplicit(n,m,Dx,Dy)
echo off;
numgrid(n,m);
R = 5.0;
T = R*ones(n+1,m+1); % All T(i,j) = 1 includes all boundary conditions
x = [0:Dx:n*Dx];y=[0:Dy:m*Dy]; % x and y vectors
for i = 1:n              % Boundary conditions at j = m+1 and j = 1
```

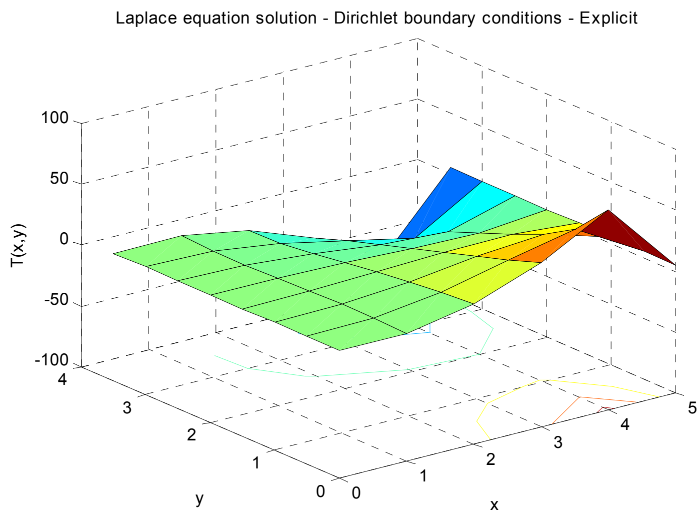
```
   T(i,m+1) = T(i,m+1)+ R*x(i)*(1-x(i));
   T(i,1)  = T(i,1)  + R*x(i)*(x(i)-1);
end;
TN = T; % TN = new iteration for solution
err = TN-T;
% Parameters in the solution
beta = Dx/Dy;
denom = 2*(1+beta^2);
% Iterative procedure
epsilon = 1e-5;    % tolerance for convergence
imax    = 1000;    % maximum number of iterations allowed
k       = 1;       % initial index value for iteration
% Calculation loop
while k<= imax
   for i = 2:n
       for j = 2:m
          TN(i,j)=(T(i-1,j)+T(i+1,j)+beta^2*(T(i,j-1)+T(i,j+1)))/denom;
          err(i,j) = abs(TN(i,j)-T(i,j));
       end;
    end;
    T = TN; k = k + 1;
    errmax = max(max(err));
    if errmax < epsilon
       [X,Y] = meshgrid(x,y);
       figure(2);contour(X,Y,T',20);xlabel('x');ylabel('y');
       title('Laplace equation solution - Dirichlet boundary conditions
- Explicit');

figure(3);surfc(X,Y,T');xlabel('x');ylabel('y');zlabel('T(x,y)');
       title('Laplace equation solution - Dirichlet boundary conditions
- Explicit');
       fprintf('Convergence achieved after %i iterations.\n',k);
       fprintf('See the following figures:\n');
       fprintf('==========================\n');
       fprintf('Figure 1 - sketch of computational grid \n');
       fprintf('Figure 2 - contour plot of temperature \n');
       fprintf('Figure 3 - surface plot of temperature \n');
       return
    end;
 end;
 fprintf('\n No convergence after %i iterations.',k);
```

To activate the function for the case illustrated in the figure above we use:

```
» [x,y,T] = LaplaceExplicit(5,7,1,0.5)
```

The solution is returned in the vectors *x* and *y*, and in matrix *T*. The function produces three plots: a sketch of the grid (similar to the figure above), the solution as a contours, and the solution as a surface. The last two figures are shown next:





Successive over-relaxation (SOR)

A solution as that provided by equation [13] is referred as a *relaxation* solution, since the value at each node of the solution domain is slowly "relaxed" into a convergent solution. A way to accelerate the convergence is by improving the current iteration at point *(i,j)* by using as many values of the current iteration as possible. For example, if we are "sweeping" the solution grid by rows, (i.e., by letting the sub-index i vary slower than the sub-index j), at each point T_{ij} we would already know the value $T_{i,j-1}^{k+1}$. Thus, the following version of equation [13] will be used in the solution (notice the different time levels involved k and $k+1$):

$$T_{i,j}^{k+1} = \frac{T_{i-1,j}^{k} + T_{i+1,j}^{k} + \beta^2 (T_{i,j-1}^{k+1} + T_{i,j+1}^{k})}{2(1+\beta^2)}. \qquad [14]$$

An approach referred to as *successive over-relaxation (SOR)* weights the values of the solution at point T_{ij} at iteration levels k and $k+1$ by using weighting factors $(1-\omega)$ and ω, where ω is known as the *over-relaxation parameter*. Thus, the formula to use is:

$$T_{i,j}^{k+1} = (1-\varpi)T_{i,j}^{k} + \frac{\varpi}{2(1+\beta^2)}(T_{i-1,j}^{k+1} + T_{i+1,j}^{k} + \beta^2(T_{i,j-1}^{k+1} + T_{i,j+1}^{k})) . \qquad [15]$$

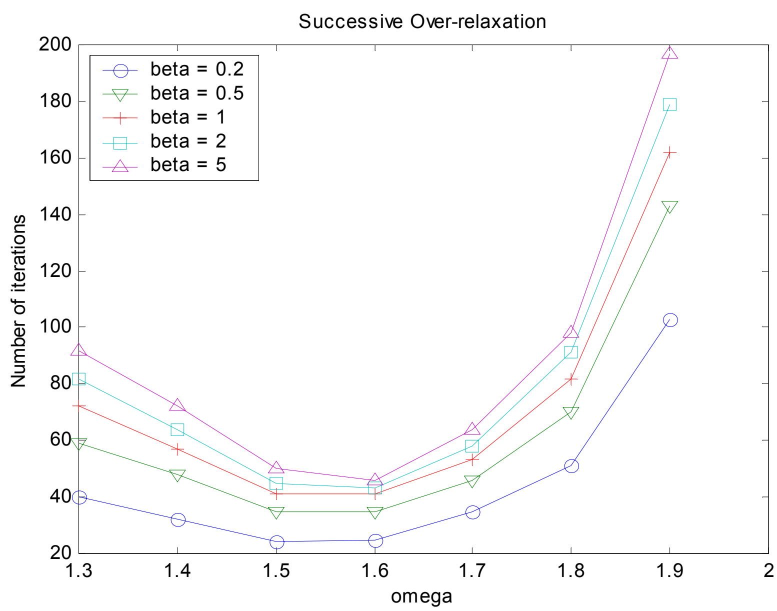
The following function, *LaplaceSOR.m*, calculates the solution for temperature distribution in the rectangular domain of page 5 by using successive over-relaxation as indicated by equation [15].

```
function [x,y,T,k]= LaplaceSOR(n,m,Dx,Dy,omega)
echo off;
numgrid(n,m);
R = 5.0;
T = R*ones(n+1,m+1); % All T(i,j) = 1 includes all boundary conditions
x = [0:Dx:n*Dx];y=[0:Dy:m*Dy]; % x and y vectors
for i = 1:n           % Boundary conditions at j = m+1 and j = 1
   T(i,m+1) = T(i,m+1)+ R*x(i)*(1-x(i));
   T(i,1) = T(i,1) + R*x(i)*(x(i)-1);
end;
TN = T; % TN = new iteration for solution
err = TN-T;
% Parameters in the solution
beta = Dx/Dy;
denom = 2*(1+beta^2);
% Iterative procedure
epsilon = 1e-5;    % tolerance for convergence
imax    = 1000;    % maximum number of iterations allowed
k       = 1;       % initial index value for iteration
% Calculation loop
while k<= imax
   for i = 2:n
       for j = 2:m
          TN(i,j)=(1-omega)*T(i,j)+omega*(TN(i+1,j)+TN(i-
1,j)+beta^2*(T(i,j+1)+TN(i,j-1)))/denom;
          err(i,j) = abs(TN(i,j)-T(i,j));
       end;
    end;
    T = TN; k = k + 1;
    errmax = max(max(err));
    if errmax < epsilon
       [X,Y] = meshgrid(x,y);
       figure(2);contour(X,Y,T',20);xlabel('x');ylabel('y');
       title('Laplace equation solution - Dirichlet boundary conditions
- Explicit');

figure(3);surfc(X,Y,T');xlabel('x');ylabel('y');zlabel('T(x,y)');
       title('Laplace equation solution - Dirichlet boundary conditions
- Explicit');
       fprintf('Convergence achieved after %i iterations.\n',k);
       fprintf('See the following figures:\n');
       fprintf('==========================\n');
```

```
        fprintf('Figure 1 - sketch of computational grid \n');
        fprintf('Figure 2 - contour plot of temperature \n');
        fprintf('Figure 3 - surface plot of temperature \n');
        return
    end;
end;
fprintf('\n No convergence after %i iterations.',k);
```

An exercise using different values of and different values of ω was attempted to elucidate the effect of ω produce the smallest number of iterations. The results for a rectangular grid with $n = 10, m = 10$, and different values of $\beta = \Delta x/\Delta y$, are shown next.

The results from this sensitivity analysis for successive over-relaxation in a rectangular domain, indicate that values of ω between 1.5 and 1.6 produce the smallest number of iterations for the solution, regardless of the value of β.



Alternative-direction successive over-relaxation (ADSOR)

This approach tries to improve the solution further by sweeping first by rows, producing intermediate values of the solution that are referred to by the iteration number *k+1/2*, i.e.,

$$T_{i,j}^{k+1/2} = (1-\varpi_1)T_{i,j}^{k} + \frac{\varpi_1}{2(1+\beta^2)}(T_{i-1,j}^{k+1/2} + T_{i+1,j}^{k} + \beta^2(T_{i,j-1}^{k+1/2} + T_{i,j+1}^{k})), \quad [16]$$

before, sweeping by columns to calculate:

$$T_{i,j}^{k+1} = (1-\varpi_2)T_{i,j}^{k+1/2} + \frac{\varpi_2}{2(1+\beta^2)}(T_{i-1,j}^{k+1} + T_{i+1,j}^{k+1/2} + \beta^2(T_{i,j-1}^{k+1} + T_{i,j+1}^{k+1/2})), \quad [17]$$

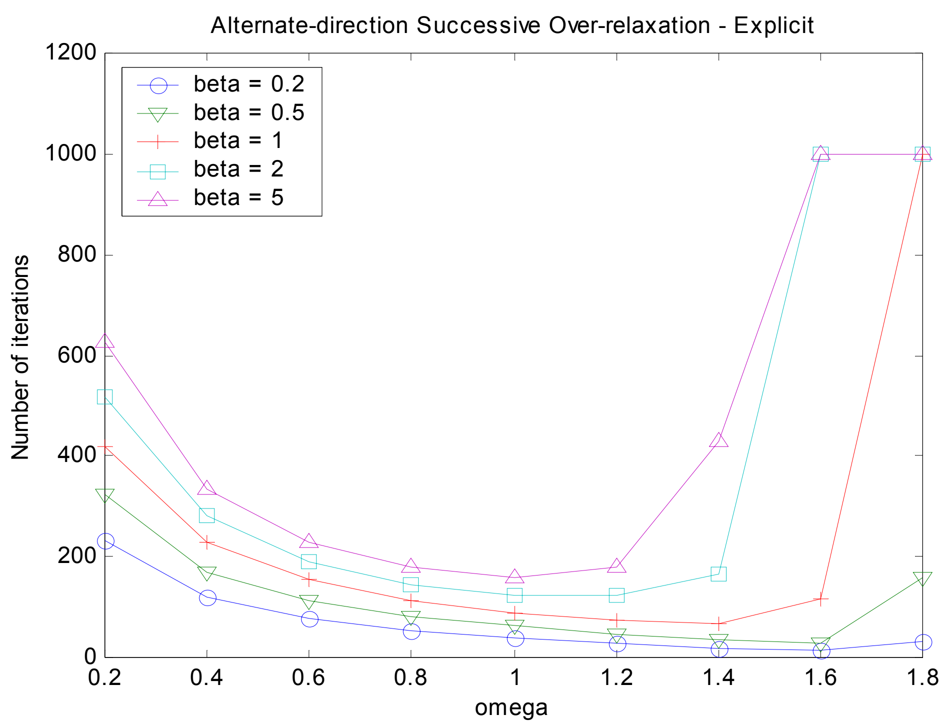
While the method indicated by equations [16] and [17] allow for the use of two different SOR parameters, namely, ω_1 and ω_2, a single value can be used, i.e., $\omega = \omega_1 = \omega_2$.

The following function, *LaplaceADSOR.m*, calculates the solution by using alternate-direction successive over-relaxation as indicated by equations [16] and [17].

```
function [x,y,T,k]= LaplaceADSOR(n,m,Dx,Dy,omega1,omega2)
echo off;
numgrid(n,m);
R = 5.0;
T = R*ones(n+1,m+1); % All T(i,j) = 1 includes all boundary conditions
x = [0:Dx:n*Dx];y=[0:Dy:m*Dy]; % x and y vectors
for i = 1:n           % Boundary conditions at j = m+1 and j = 1
   T(i,m+1) = T(i,m+1)+ R*x(i)*(1-x(i));
   T(i,1) = T(i,1) + R*x(i)*(x(i)-1);
end;
TN = T; % TN = new iteration for solution
TI = T; % TI = intermediate solution step
err = TN-T;
% Parameters in the solution
beta = Dx/Dy;
denom = 2*(1+beta^2);
% Iterative procedure
epsilon = 1e-5;   % tolerance for convergence
imax    = 1000;   % maximum number of iterations allowed
k       = 1;      % initial index value for iteration
% Calculation loop
while k<= imax
   for i = 2:n    % Sweeping by rows
       for j = 2:m
          TI(i,j)=(1-omega1)*T(i,j)+omega1*(T(i+1,j)+TI(i-
1,j)+beta^2*(T(i,j+1)+TI(i,j-1)))/denom;
       end;
    end;
   TN = TI;
   for j = 2:m   % Sweeping by columns
       for i = 2:n
           TN(i,j)=(1-omega2)*TI(i,j)+omega2*(TI(i+1,j)+TN(i-
1,j)+beta^2*(T(i,j+1)+TN(i,j-1)))/denom;
           err(i,j) = abs(TN(i,j)-T(i,j));
        end;
   end;
   T = TN; k = k + 1;
   errmax = max(max(err));
   if errmax < epsilon
       [X,Y] = meshgrid(x,y);
       figure(2);contour(X,Y,T',20);xlabel('x');ylabel('y');
       title('Laplace equation solution - Dirichlet boundary conditions
- Explicit');

figure(3);surfc(X,Y,T');xlabel('x');ylabel('y');zlabel('T(x,y)');
       title('Laplace equation solution - Dirichlet boundary conditions
- Explicit');
       fprintf('Convergence achieved after %i iterations.\n',k);
       fprintf('See the following figures:\n');
```

```
        fprintf('=========================\n');
        fprintf('Figure 1 - sketch of computational grid \n');
        fprintf('Figure 2 - contour plot of temperature \n');
        fprintf('Figure 3 - surface plot of temperature \n');
        return
    end;
end;
fprintf('\n No convergence after %i iterations.',k);
```

An exercise using different values of β and different values of $\omega = \omega_1 = \omega_2$ was attempted to figure out the effect of ω in the number of iterations required for convergence. The results for a rectangular grid with $n = 10$, $m = 10$, and different values of $\beta = \Delta x/\Delta y$, are shown next.



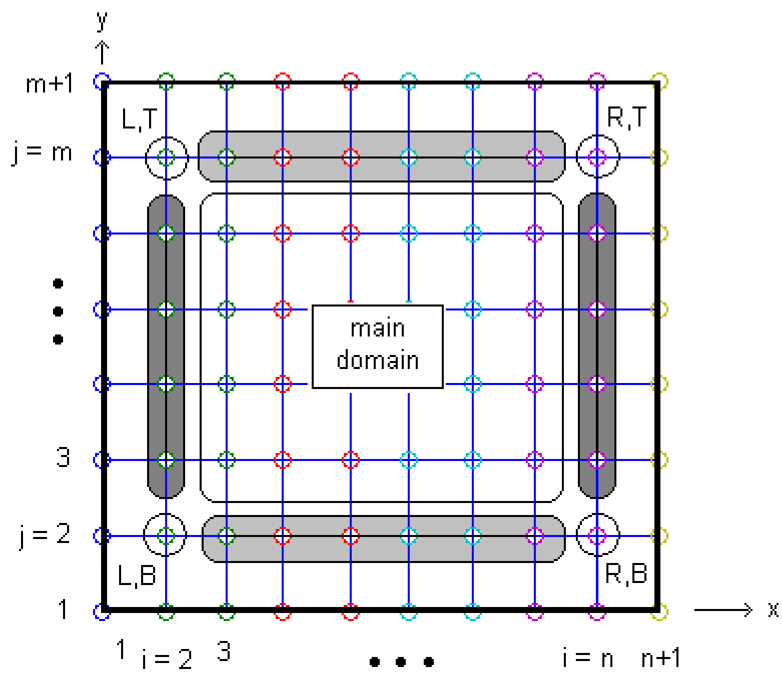
The value of ω that minimizes the number of iterations for convergence is obviously a function of β. The smaller the value of β, the higher the value of w required to minimize the number of iterations. For $\beta = 0.2$, for example, $\omega = 1.6$ minimizes the number of iterations for convergence, while for $\beta = 5$, the value of ω that minimizes the number of iterations is close to 1.0.

An implicit solution

Implicit solutions typically consist of solving a number of simultaneous algebraic equations involving the unknown values T_{ij} in the interior points of the solution domain. For example, for points in the domain not contiguous to a boundary, i.e., for $i = 2, 3, \ldots, n-1$, and $j = 2, 3, \ldots, m+1$, the algebraic solutions to solve result from re-writing equation [13] to read:

$$T_{i-1,j} + T_{i+1,j} - 2(1+\beta^2)T_{ij} + \beta^2(T_{i,j-1} + T_{i,j+1}) = 0\,. \qquad [18]$$

We'll refer to these points as the *main domain.* There are *(n-3)(m-3)* points in the main domain, thus, producing *(n-3)(m-3)* equations. The main domain and other sub-domains of interest are shown in the following figure.



For points contiguous to boundaries, the following equations apply:

- Left bottom corner (L,B): $T_{1,2} + T_{3,2} - 2(1+\beta^2)T_{2,2} + \beta^2(T_{2,1} + T_{2,3}) = 0\,.$
- Left top corner (L,T): $T_{1,m} + T_{3,m} - 2(1+\beta^2)T_{2,m} + \beta^2(T_{2,m-1} + T_{2,m+1}) = 0\,.$
- Right bottom corner (R,B): $T_{n-1,2} + T_{n+1,2} - 2(1+\beta^2)T_{n,2} + \beta^2(T_{n,1} + T_{n,3}) = 0\,.$
- Right top corner (R,T): $T_{n-1,m} + T_{n+1,m} - 2(1+\beta^2)T_{n,m} + \beta^2(T_{n,m-1} + T_{n,m+1}) = 0\,.$
- Along the line *i = 2*: $T_{1,j} + T_{3,j} - 2(1+\beta^2)T_{2,j} + \beta^2(T_{2,j-1} + T_{2,j+1}) = 0\,.$
- Along the line *i = n*: $T_{n-1,j} + T_{n+1,j} - 2(1+\beta^2)T_{n,j} + \beta^2(T_{n,j-1} + T_{n,j+1}) = 0\,.$
- Along the line *j = 2*: $T_{i-1,2} + T_{i+1,2} - 2(1+\beta^2)T_{i,2} + \beta^2(T_{i,1} + T_{i,3}) = 0\,.$
- Along the line *j = m*: $T_{i-1,m} + T_{i+1,m} - 2(1+\beta^2)T_{i,m} + \beta^2(T_{i,m-1} + T_{i,m+1}) = 0\,.$

Notice that in these equations the values of $T_{1,2}$, $T_{2,1}$, $T_{1,m}$, $T_{2,m+1}$, $T_{n+1,2}$, $T_{n,1}$, $T_{n+1,m}$, $T_{n,m+1}$, $T_{1,j}$, $T_{n+1,j}$, $T_{i,1}$, and $T_{i,m+1}$ are known.

The corners, (L,B), (L,T), (R,B), and (R,T), produce 4 more equations besides the *(n-3)(m-3)* equations from the *main domain*. Lines *i = 2* and *i = n* produce *(n-3)* equations each, while the lines *j = 2* and *j = m* produce *(m-3)* equations each. Thus, the total number of equations produced is $(n\text{-}3)(m\text{-}3) + 4 + 2(n\text{-}3) + 2(m\text{-}3) = (n\text{-}1)(m\text{-}1)$, which corresponds to the number of unknowns $(n+1\text{-}2)(m+1\text{-}2) = (n\text{-}1)(m\text{-}1)$. Therefore, the system of equations should be, at least, in principle, uniquely determined.

The system of equations can be cast as a matrix equation where the unknowns are the values T_{ij} in the interior points of the solution domain. A difficulty that arises at this point is trying to write the matrix equation in terms of unknown variables having a single sub-index. This difficulty can be overcome by replacing the unknown T_{ij} with the unknown X_k where $k = (j\text{-}2)(n\text{-}1) + i - 1$. This way, the variables X_k take the place of the variables T_{ij} so that $X_1 = T_{2,2}$, $X_2 = T_{2,3}$, etc., resulting in $(n\text{-}1)(m\text{-}1)$ variables X_k.

The implicit solution is implemented in function *LaplaceImplicit.m*. The function uses sparse matrices, since a large number of elements of the matrix of coefficients for the system of equations are zero. A graph of the solution domain, diagrams of the matrix of coefficients and of the right-hand side vector, as well as graphics of the solution are produced by the function.

```
function [x,y,T]= LaplaceImplicit(n,m,Dx,Dy)
echo off;
% The following function calculates index k for X(k) corresponding to
% variable T(i,j), such that k = (j-1)*(n-1)+i-1
k = inline('(j-2)*(n-1)+i-1','i','j','n');
numgrid(n,m); % Shows numerical grid
R = 5.0;
T = R*ones(n+1,m+1); % All T(i,j) = 1 includes all boundary conditions
x = [0:Dx:n*Dx];y=[0:Dy:m*Dy]; % x & y points in solution domain
for i = 1:n          % Boundary conditions at j = m+1 and j = 1
   T(i,m+1) = T(i,m+1)+ R*x(i)*(1-x(i));
   T(i,1) = T(i,1) + R*x(i)*(x(i)-1);
end;
beta = Dx/Dy;                % Parameters of the solution
denom = -2*(1+beta^2);
kk = (n-1)*(m-1);
A = zeros(kk,kk); b = zeros(kk,1);
kvL = []; kvR = []; kvC = []; kvB = []; kvT = [];
%main domain
for i = 3:n-1
   for j = 3:m-1
       ke=k(i,j,n);kL=k(i-1,j,n);kR=k(i+1,j,n);
       kC=k(i,j,n);kB=k(i,j-1,n);kT=k(i,j+1,n);
       A(ke,kL) = 1; A(ke,kR) = 1; A(ke,kC) = denom;
       A(ke,kB) = beta^2; A(ke,kT)=beta^2; b(ke) = 0;
   end;
end;
%Left-Bottom corner
i=2;j=2;ke=k(i,j,n);kR=k(i+1,j,n);kC=k(i,j,n);kT=k(i,j+1,n);b(ke)=-T(i-
1,j)-beta^2*T(i,j-1);
A(ke,kR) = 1; A(ke,kC) = denom;A(ke,kT)=beta^2;
%Left-Top corner
i=2;j=m;ke=k(i,j,n);kR=k(i+1,j,n);kC=k(i,j,n);kB=k(i,j-1,n);b(ke)=-T(i-
1,j)-beta^2*T(i,j+1);
A(ke,kR) = 1; A(ke,kC) = denom;A(ke,kB) = beta^2;
%Right-Bottom corner
```

```
i=n;j=2;ke=k(i,j,n);kL=k(i-1,j,n);kC=k(i,j,n);kT=k(i,j+1,n);b(ke)=-
T(i+1,j)-beta^2*T(i,j-1);
A(ke,kL) = 1; A(ke,kC) = denom;A(ke,kT)=beta^2;
%Right-Top corner
i=n;j=m;ke=k(i,j,n);kL=k(i-1,j,n);kC=k(i,j,n);kB=k(i,j-1,n);b(ke)=-
T(i+1,j)-beta^2*T(i,j+1);
A(ke,kL) = 1; A(ke,kC) = denom; A(ke,kB) = beta^2;
%i=2 (left column)
i=2;
for j = 3:m-1
   ke=k(i,j,n);kR=k(i+1,j,n);kC=k(i,j,n);kB=k(i,j-
1,n);kT=k(i,j+1,n);b(ke)=-T(i-1,j);
   A(ke,kR) = 1; A(ke,kC) = denom; A(ke,kB) = beta^2; A(ke,kT)=beta^2;
end;
%i=n (right column)
i=n;
for j = 3:m-1
   ke=k(i,j,n);kL=k(i-1,j,n);kC=k(i,j,n);kB=k(i,j-
1,n);kT=k(i,j+1,n);b(ke)=-T(i+1,j);
   A(ke,kL) = 1; A(ke,kC) = denom; A(ke,kB) = beta^2; A(ke,kT)=beta^2;
end;
%j=2 (bottom row)
j=2;
for i = 3:n-1
   ke=k(i,j,n);kL=k(i-
1,j,n);kR=k(i+1,j,n);kC=k(i,j,n);kT=k(i,j+1,n);b(ke)=-beta^2*T(i,j-1);
   A(ke,kL) = 1; A(ke,kR) = 1; A(ke,kC) = denom; A(ke,kT)=beta^2;
end;
%j=m (top row)
j=m;
for i = 3:n-1
   ke=k(i,j,n);kL=k(i-1,j,n);kR=k(i+1,j,n);kC=k(i,j,n);kB=k(i,j-
1,n);b(ke)=-beta^2*T(i,j+1);
   A(ke,kL) = 1; A(ke,kR) = 1; A(ke,kC) = denom; A(ke,kB) = beta^2;
end;
% Create sparse matrix A and sparse vector b
As = sparse(A);
figure(2);spy(As);title('Matrix of coefficients'); % Picture of sparse
matrix A
bs = sparse(b);
figure(3);spy(b);title('Right-hand side vector');  % Picture of sparse
vector b
XX = As\bs;     % Solve using left division
% Convert solution back to T(i,j), i.e., T(i,j) = X(k), with k = j-
1)*(n-1)+i-1
for i = 2:n
   for j = 2:m
      ke = k(i,j,n); T(i,j) = XX(ke);
   end;
end;
[X,Y] = meshgrid(x,y); % Generate grid data for contour plot and
surface plot
figure(4);contour(X,Y,T',20);xlabel('x');ylabel('y');
title('Laplace equation solution - Dirichlet boundary conditions -
Implicit');
figure(5);surfc(X,Y,T');xlabel('x');ylabel('y');zlabel('T(x,y)');
```

```
title('Laplace equation solution - Dirichlet boundary conditions - 
Implicit');
% Indicate list of figures produced
fprintf('See the following figures:\n');
fprintf('==========================\n');
fprintf('Figure 1 - sketch of computational grid \n');
fprintf('Figure 2 - diagram of matrix of coefficients \n');
fprintf('Figure 3 - diagram of right-hand side vector \n');
fprintf('Figure 4 - contour plot of temperature \n');
fprintf('Figure 5 - surface plot of temperature \n');
```

An example is calculated by using:

```
» [x,y,T] = LaplaceImplicit(10,10,1,1);
```

Results for this case are shown below.

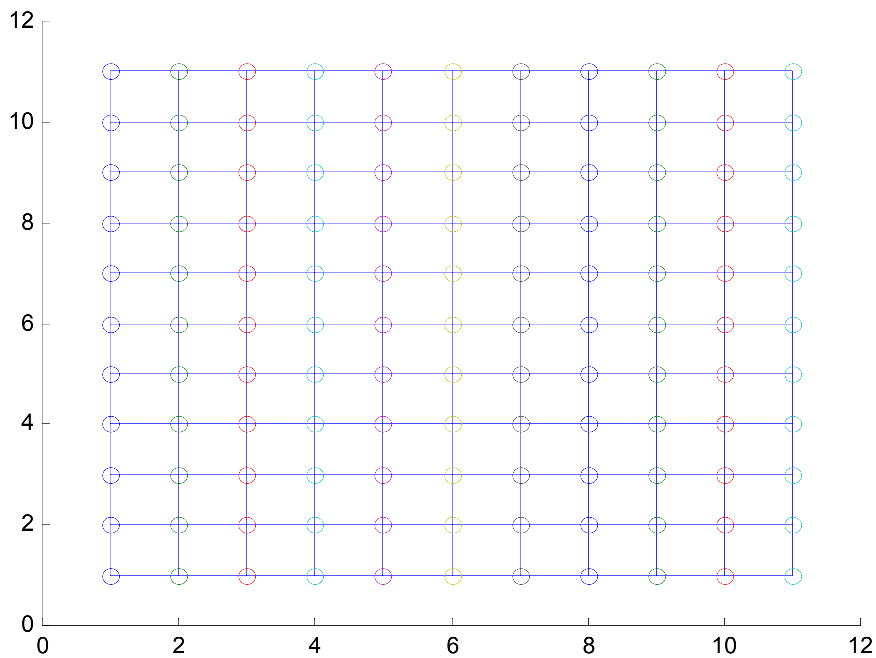
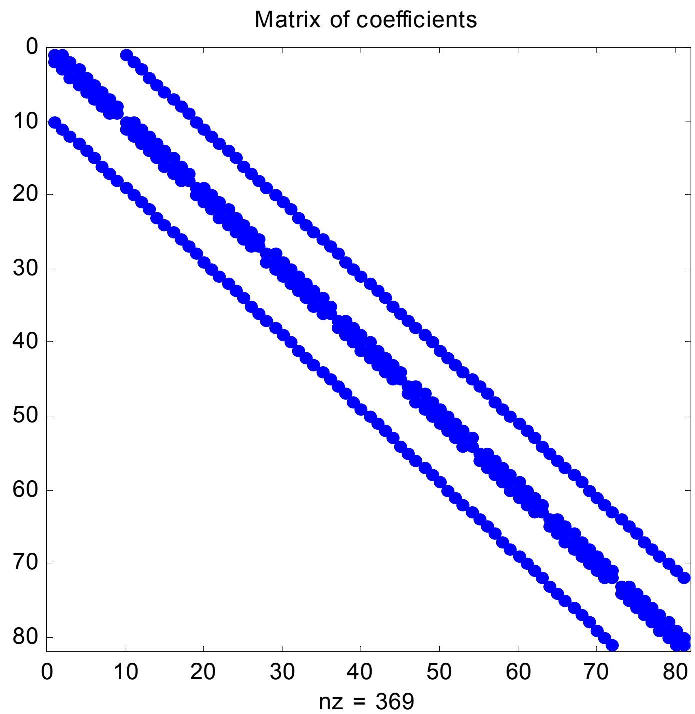
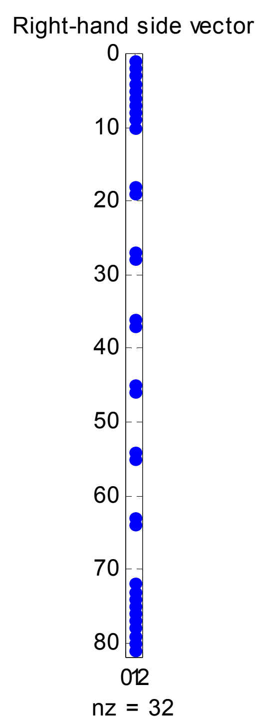


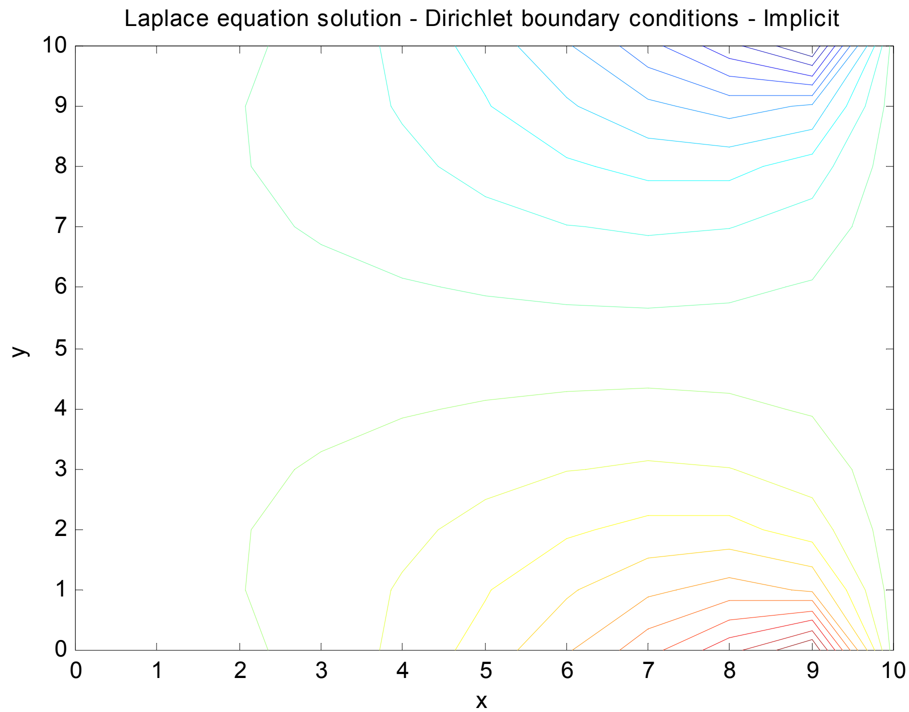
Figure showing the solution domain.



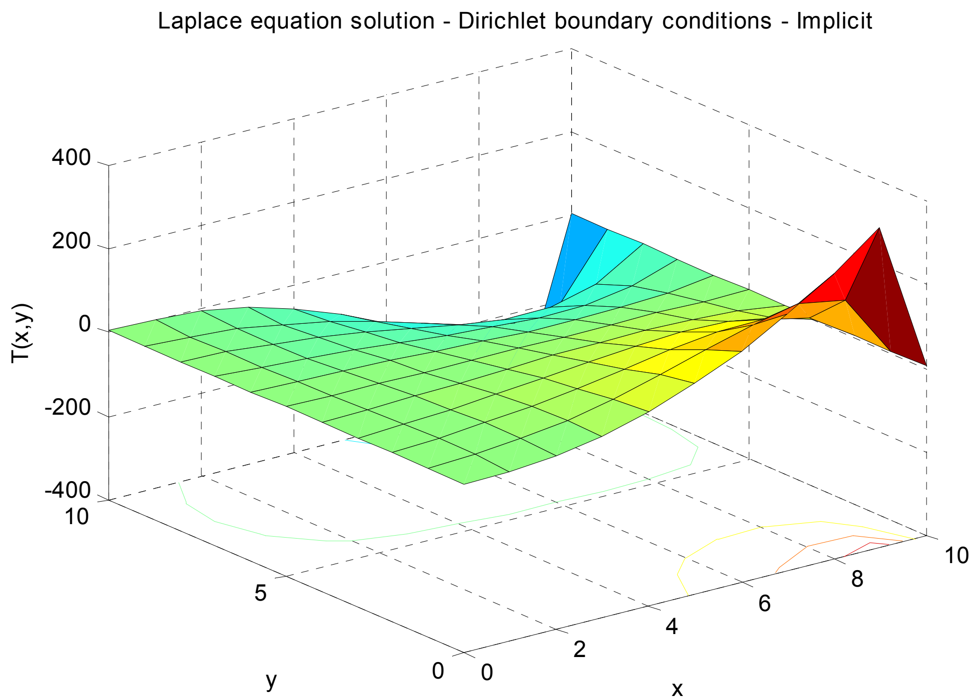
Spy graph of sparse matrix of coefficients.



Spy graph of sparse right-hand side vector.



Contour plot of temperatures



Surface plot of temperatures

Successive over-relaxation semi-implicit scheme

The main difficulty of the implicit approach described earlier is that not all domains produce systems of equations that can be easily re-cast into a sparse matrix equation. In order to simplify the solution, a semi-implicit successive over-relaxation scheme is proposed. This scheme is such that, in each sweep by rows, a tri-diagonal system of equations is produced. This tri-diagonal system can be easily solved using the Thomas algorithm (described later).

The equation to be used in each interior point, sweeping by rows, is:

$$T_{i,j}^{k+1} = \frac{T_{i-1,j}^{k+1} + T_{i+1,j}^{k+1} + \beta^2 (T_{i,j-1}^{k+1} + T_{i,j+1}^{k})}{2(1+\beta^2)},$$

which can be re-cast as

$$T_{i-1,j}^{k+1} - 2(1+\beta^2) T_{i,j}^{k+1} + T_{i+1,j}^{k+1} = -\beta^2 (T_{i,j-1}^{k+1} + T_{i,j+1}^{k}). \quad [19]$$

For each value of j, as we let $i = 2, 3, \ldots, n$, we produce $(n-1)$ equations in $(n-1)$ unknowns. These are solved using the Thomas algorithm. Then, the value of j is incremented, so that $j = 2, 3, \ldots, m$, and new solutions calculated for each value of j. The process is iterative, and convergence should be checked by using, for example, the criteria:

$$\max_{i,j} | T_{i,j}^{k+1} - T_{i,j}^{k} | \le \varepsilon . \quad [20]$$

Alternate-direction successive over-relaxation semi-implicit scheme

In this scheme tri-diagonal systems of equations are produced by first sweeping by rows, to produce intermediate values $T_{ij}^{k+1/2}$, and then sweeping by columns to produce the new iteration values T_{ij}^{k+1}. The equations to use are:

$$T_{i,j}^{k+1/2} = (1-\varpi_1) T_{i,j}^{k} + \frac{\varpi_1}{2(1+\beta^2)} (T_{i-1,j}^{k+1/2} + T_{i+1,j}^{k} + \beta^2 (T_{i,j-1}^{k+1/2} + T_{i,j+1}^{k})), \quad [21]$$

before, sweeping by columns to calculate:

$$T_{i,j}^{k+1} = (1-\varpi_2) T_{i,j}^{k+1/2} + \frac{\varpi_2}{2(1+\beta^2)} (T_{i-1,j}^{k+1} + T_{i+1,j}^{k+1/2} + \beta^2 (T_{i,j-1}^{k+1} + T_{i,j+1}^{k+1/2})), \quad [22]$$

Thomas (or double sweep) algorithm

This algorithm is described briefly in Vreughdenhill's *Computational Hydraulics*, section *7.4* (pages 40 and 41) for the specific case of the implicit Crank-Nicholson method for the diffusion equation. In this section, the Thomas, or double-sweep, algorithm is described for the numerical solution of Laplace's equation in a rectangular domain.

The algorithm solves a tri-diagonal system of equations of the form

$$a_k h_{k-1} + b_k h_k + c_k h_{k+1} = d_k, \qquad [23]$$

for $k = 1, 2, \ldots, N+1$, with $a_1 = 0$ and $c_{N+1} = 0$. Thus, only the main diagonal of the matrix of coefficients and the two adjacent diagonals in this system are non-zero (hence, the name *tri-diagonal*).

The values of the constants a_k, b_k, c_k, and d_k are obtained from the implicit equations with the appropriate definitions of the variables h_k. For example, if we were to obtain the tri-diagonal equations for the implicit equation given in [19], namely,

$$T_{i-1,j} - 2(1+\beta^2)T_{ij} + T_{i+1,j} = -\beta^2(T_{i,j-1} + T_{i,j+1}), \qquad [19]$$

while sweeping by rows (i.e., j is constant), make the replacements $k = i$, $N = n$, $h_{k-1} = T_{i-1,j}$, $h_k = T_{i,j}$, and $h_{k+1} = T_{i+1,j}$ in equation [23]. Thus, the constants to use in this case would be

$$a_k = c_k = 1,\ b_k = -2(1+\beta^2),\ \text{and}\ d_k = -\beta^2(T_{i,j-1}+T_{i,j+1}), \qquad [19\text{-b}]$$

for $k = 2, \ldots, N-1$ and a fixed value of j. The equations and constants corresponding to $k=i=1$ and $k = i = n+1$, will be determined by the boundary conditions of the problem at $x = x_1$ and $x = x_{n+1}$, respectively. (NOTE: it is assumed that the solution grid has n sub-intervals in x and m sub-intervals in y).

If we were to use Thomas algorithm to solve the implicit equation in [19] while sweeping by columns (i.e., i remains constant), you can prove that the proper replacement of indices and variables would be $k = j$, $N = m$, $h_{k-1} = T_{i,j-1}$, $h_k = T_{i,j}$, and $h_{k+1} = T_{i,j+1}$ in equation [23]. The corresponding constants in equation [23] for this case would be:

$$a_k = c_k = 1,\ b_k = -2(1+\beta^2),\ \text{and}\ d_k = -\beta^2(T_{i-1,j}+T_{i+1,j}), \qquad [18\text{-c}]$$

for $k = 2, \ldots, m-1$ and a fixed value of j. The equations and constants corresponding to $k=j=1$ and $k = j = m+1$, will be determined by the boundary conditions of the problem at $y = y_1$ and $y = y_{m+1}$, respectively. (NOTE: using equations [21] and [22], the constants in [23] would have a more complicated expression).

The algorithm postulates the following relationship for the solution of the system of equations [23]:

$$h_k = e_k + f_k h_{k+1}, \qquad [24]$$

where

$$e_k = \frac{d_k - a_k e_{k-1}}{b_k + a_k f_{k-1}}, \qquad e_1 = \frac{d_1}{b_1} \qquad [25]$$

$$f_k = \frac{-c_k}{b_k + a_k f_{k-1}}, \qquad f_1 = -\frac{c_1}{b_1} \qquad [26]$$

The algorithm should proceed as follows:

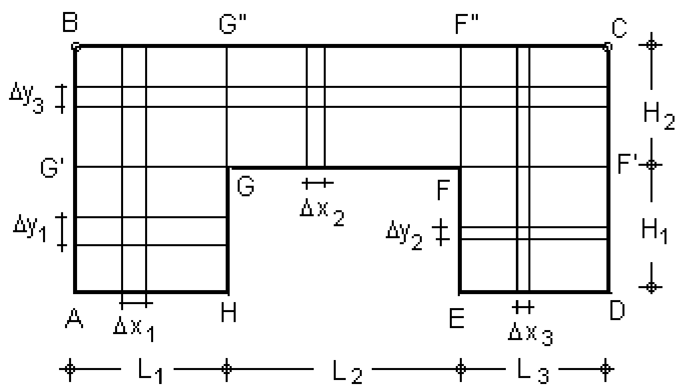
- Apply upstream boundary condition (i.e., at $k = 1$) to calculate e_1 and f_1.
- Calculate the coefficients e_k, f_k for $k = 2, \ldots, N$.
- Apply downstream boundary condition to calculate h_{N+1}.
- Perform a backward sweep calculating the values of h_k from equation [24] for $k = N, N-1, \ldots 2$.

Exercises – part I

[1]. (a) Write a function to implement the successive over-relaxation semi-implicit scheme described above. (b) Use the function to solve for the temperature distribution in the rectangular domain described earlier.

[2]. (a) Write a function to implement the alternate-direction successive over-relaxation semi-implicit scheme described above. (b) Use the function to solve for the temperature distribution in the rectangular domain described earlier. (c) Produce a plot showing the effect of the value of ω on the number of iterations required to achieve a solution for different values of $\beta = \Delta x/\Delta y$.

NOTE: The following figure applies to problems [3] and [4]:



[3]. (a) Use an explicit scheme without over-relaxation to solve for the temperature distribution in the domain shown above. (b) Produce a contour plot of the temperature.

Let $L_1 = L_3 = 5\ cm$, $L_2 = 10\ cm$, $\Delta x_1 = \Delta x_2 = \Delta x_3 = \Delta x = 1\ cm$, $H_1 = 3\ cm$, $H_2 = 5\ cm$, $\Delta y_1 = \Delta y_2 = \Delta y_3 = \Delta y = 0.5\ cm$.

Let the boundary conditions for the temperature be such that:

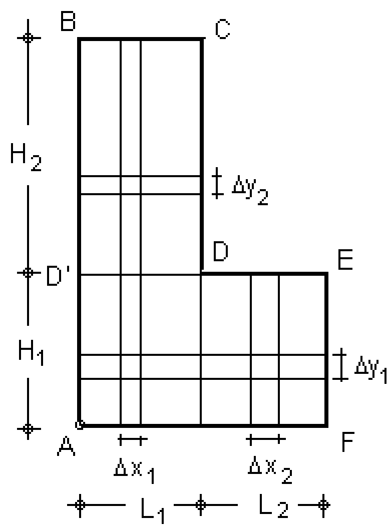
- $T = 10\ ^oC$ along BC.
- T decreases linearly from $10\ ^oC$ at B and C down to $0\ ^oC$ at A and D, respectively.
- T increases linearly from $0\ ^oC$ at A and D, to $5\ ^oC$ at H and E, respectively.
- T increases linearly from $5\ ^oC$ at H and E to $7.5\ ^o$ C at G and F, respectively.

- Along GF, T increases linearly from $7.5\ ^oC$ at G and F towards the mid-point of segment GF, where it reaches a maximum value of $10\ ^oC$.

[4]. (a) Use an explicit scheme without over-relaxation to solve for the temperature distribution in the domain shown above. (b) Produce a contour plot of the temperature.

Let $L_1 = L_3 = 5\ cm$, $L_2 = 10\ cm$, $\Delta x_1 = 1\ cm$, $\Delta x_2 = 0.5\ cm$, $\Delta x_3 = 1.25\ cm$, $H_1 = 4\ cm$, $H_2 = 2\ cm$, $\Delta y_1 = 1\ cm$, $\Delta y_2 = 1.25\ cm$, $\Delta y_3 = 0.5\ cm$. Use the same boundary conditions as in problem [3].

NOTE: The following figure applies to problems [5] and [6]:



[5]. (a) Use an explicit scheme without over-relaxation to solve for the temperature distribution in the domain shown above. (b) Produce a contour plot of the temperature.

Let $L_1 = 5\ cm$, $L_2 = 4\ cm$, $\Delta x_1 = \Delta x_2 = \Delta x = 1\ cm$, $H_1 = 3\ cm$, $H_2 = 2\ cm$, $\Delta y_1 = \Delta y_2 = \Delta y = 0.5\ cm$.

Let the boundary conditions for the temperature be such that:

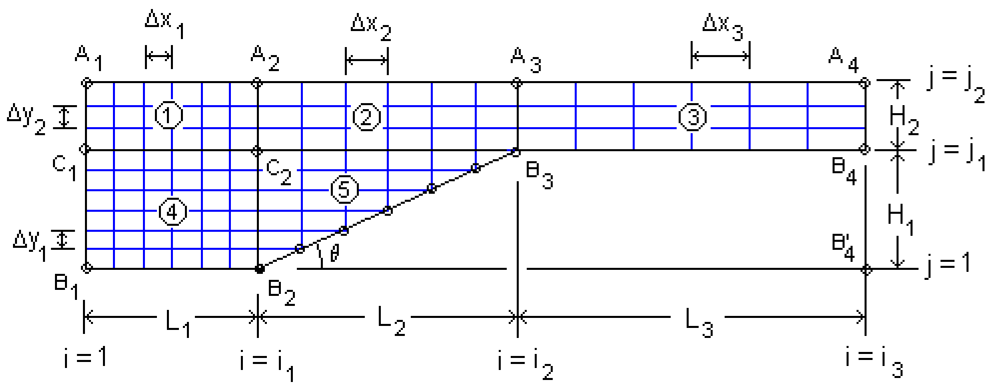
- $T = 10\ ^oC$ along BC.
- T decreases linearly from $10\ ^oC$ at C to $5\ ^oC$ at D.
- T decreases linearly from $5\ ^oC$ at D to $0\ ^oC$ at E.
- $T = 0\ ^oC$ along EF and FA.
- T increases linearly from $0\ ^oC$ at A to $10\ ^oC$ at B.

[6]. (a) Use an explicit scheme without over-relaxation to solve for the temperature distribution in the domain shown above. (b) Produce a contour plot of the temperature.

Let $L_1 = 5$ *cm*, $L_2 = 4$ *cm*, $\Delta x_1 = 1.25$ *cm*. $\Delta x_2 = 0.5$ *cm*, $H_1 = 3$ *cm*, $H_2 = 2$ *cm*, $\Delta y_1 = 1$ *cm*, $\Delta y_2 = 0.5$ *cm*.

Laplace equation solution in simple non-rectangular domains

The figure below shows a two-dimensional thin solid body formed by removing the trapezoidal shape $B_2B_3B_4B_4$' from the rectangular shape $A_1A_4B_1B_4$'. The resulting shape has side B_2B_3 tilted by an angle θ with respect to the horizontal line $B_1B_2B_4$'.



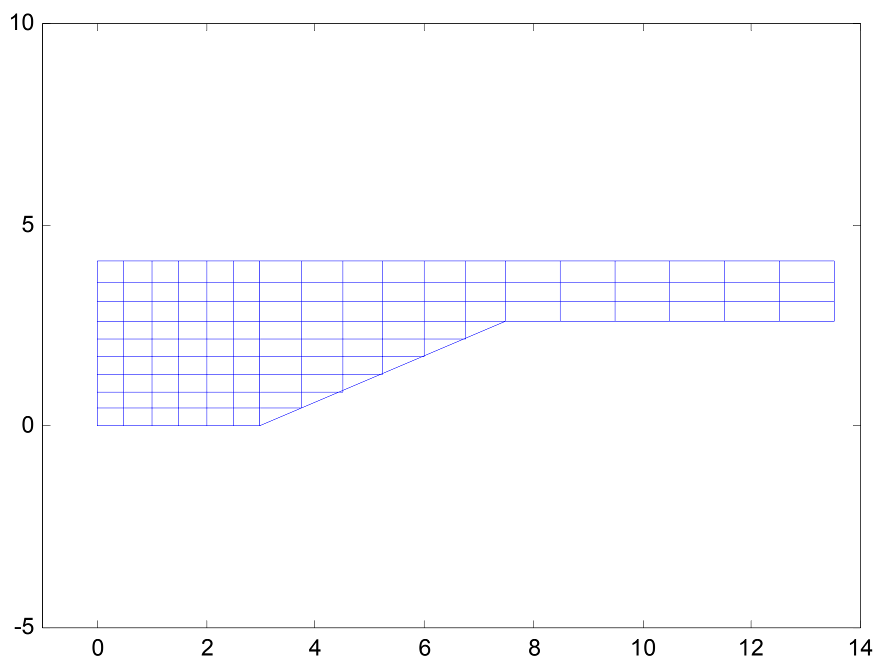
The computational grid attached to the irregular shape above is designed so that there are a number of equally spaced grid nodes along the inclined size B_2B_3. The rectangular grid with increments Δx_2 and Δy_1, that define the grid nodes along the inclined side, are related by $\tan\theta = \Delta y_1 / \Delta x_2$. The figure shows 5 different solution domains with varying size increments in x and y. Expressions for the finite-difference approximation for Laplace's equation in each of the domains, and on their boundaries, can be found, for example, by using equations [10] through [13].

The following Matlab script can be used to produce the plot of the irregularly-shaped body shown above for specific values of the dimensions L_1, L_2, L_3, H_1, and H_2:

```
% Script for plotting irregularly-shaped body and computational grid
%   Calculating the geometry
Dx1 = 0.50; Dx2 = 0.75; Dx3 = 1.00;
L1  = 3.00; L2  = 4.50; L3  = 6.00;
n1  = L1/Dx1; n2 = L2/Dx2; n3 = L3/Dx3;
i1  = 1+n1;   i2 = i1+n2;  i3 = i2+n3;
theta = 30*pi/180; Dy1 = Dx2*tan(theta);
Dy2 = 0.5;
H1  = L2*tan(theta); H2 = 2.00;
m1  = round(H1/Dy1); m2 = round(H2/Dy2);
j1  = 1+m1; j2 = m1+m2;
%   Calculating coordinates
x = [0:Dx1:L1,L1+Dx2:Dx2:L1+L2,L1+L2+Dx3:Dx3:L1+L2+L3];
y = [0:Dy1:H1,H1+Dy2:Dy2:H1+H2];
%   Plot solution domain
figure(1);
plot([x(1),x(i1)],[y(1),y(1)],'-b');hold on;
plot([x(i1),x(i2)],[y(1),y(j1)],'-b');
```

```
plot([x(i2),x(i3)],[y(j1),y(j1)],'-b');
plot([x(i3),x(i3)],[y(j1),y(j2)],'-b');
plot([x(1),x(i3)],[y(j2),y(j2)],'-b');
plot([x(1),x(1)],[y(1),y(j2)],'-b');
axis([-1 14 -5 10]);
%   Plot solution grid
for j=2:j1
   plot([x(1),x(i1)],[y(j),y(j)]);
end;
for j=j1:j2-1
   plot([x(1),x(i3)],[y(j),y(j)]);
end;
for i = 2:i1
   plot([x(i),x(i)],[y(1),y(j2)]);
end;
for i = i2:i3-1
   plot([x(i),x(i)],[y(j1),y(j2)]);
end;
ii = [i1:1:i2]; jj = [1:1:j2];
for i = 1:n2
   plot([x(i1),x(ii(i))],[y(jj(i)),y(jj(i))]);
   plot([x(ii(i)),x(ii(i))],[y(jj(i)),y(j2)]);
end;
hold off;
```

The following figure shows the result:



Outline of an explicit solution with Dirichlet boundary conditions
For Dirichlet-type boundary conditions, the temperature T will be known in every grid point on the boundary. For example, along the inclined boundary B_2B_3, the values of T_{ij}, with $(i,j) = (i_1,1),(i_1+1,2), \ldots,(i_2,j_1)$ must be known.

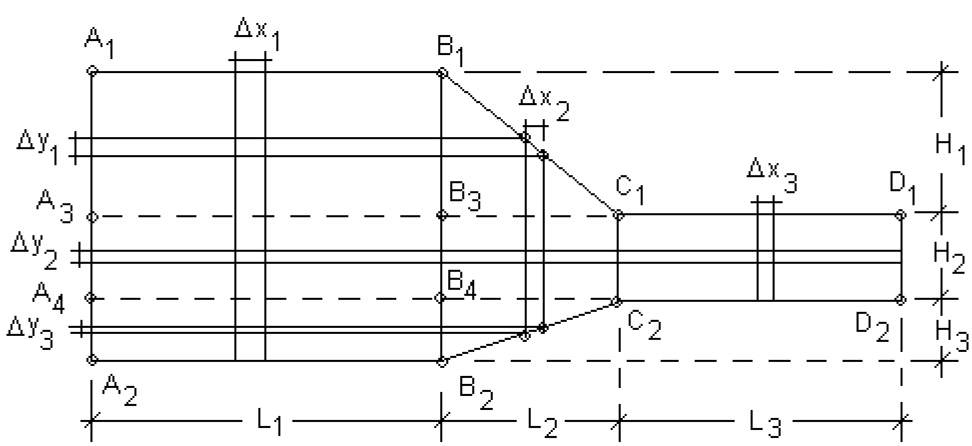
The algorithm for an implicit solution would proceed as follows:

- Load boundary conditions on all grid points on the boundaries.
- Use equation [10] for point C_2.
- Use equation [11] for the points on lines C_1C_2 and C_2B_3.
- Use equation [12] for points on the lines A_2C_2, C_2B_2, and A_3B_3.
- Finally, use equation [13] for interior points in each of the 5 domains.
- The solution is iterative, with a convergence criteria such as equation [20] used to determine when a solution has been achieved.

Thus, the explicit solution approach is very similar to that of a rectangular domain.

NOTE: Implicit solutions can also be set up following the approach used for rectangular domains after defining a solution grid as the one shown above.

Exercises – part II
[7]. For the irregularly-shaped domain shown below, (a) use an explicit method to solve the Laplace equation for temperature distribution (equation [0]) as specified below. The dimensions of the figure are: $L_1 = 10\ cm$, $L_2 = 5\ cm$, $L_3 = 8\ cm$, $H_1 = 5\ cm$, $H_2 = 3\ cm$, $H_3 = 2\ cm$, $\Delta x_1 = 1\ cm$, $\Delta x_2 = 0.5\ cm$, $\Delta x_3 = 1\ cm$, $\Delta y_1 = 0.5\ cm$, $\Delta y_2 = 0.25\ cm$, $\Delta y_3 = 0.2\ cm$. Boundary conditions: $T = 80\ ^oC$ along $A_2B_2C_2D_2$, $T = 60\ ^o$ along $A_1B_1C_1D_1$, T varies linearly along $A_1A_3A_4A_2$ as well as along D_1D_2, so that the temperature at the boundaries is continuous. (b) Produce a contour plot of the temperature distribution.



Derivation of Laplace's and Poisson's equation in heat transfer in solids
The heat flux (per unit area), q_x, along the x direction is given by the equation

$$q_x = -k \cdot \frac{\partial T}{\partial x}, \quad [27]$$

where k is the thermal conductivity of the material, and T is the temperature. Equation [27] indicates that the heat flux $q[J/m^2]$ is proportional to the temperature gradient, , and that heat flows in the direction of decreasing temperature. The units of k are $J/^oK \cdot m$.

It is possible to define a heat flux vector that accounts for heat fluxes in both the x and y directions for a two-dimensional case. Thus, the heat flux vector would be written as:

$$\mathbf{q} = q_x \mathbf{i} + q_y \mathbf{j} = -k \frac{\partial T}{\partial x}\mathbf{i} - k \frac{\partial T}{\partial y}\mathbf{j} = -k(\frac{\partial T}{\partial x}\mathbf{i} + \frac{\partial T}{\partial y}\mathbf{j}) = -k \cdot grad(T) = -k \cdot \nabla T. \quad [28]$$

In equation [12], the differential expression $\frac{\partial T}{\partial x}\mathbf{i} + \frac{\partial T}{\partial y}\mathbf{j}$ is referred to as the gradient of the temperature, i.e.,

$$grad(T) = \nabla T = \frac{\partial T}{\partial x}\mathbf{i} + \frac{\partial T}{\partial y}\mathbf{j}. \quad [29]$$

The differential operator ∇, called the *'del'* or *'nabla'* operator, is defined by:

$$\nabla[\;] = \frac{\partial[\;]}{\partial x}\mathbf{i} + \frac{\partial[\;]}{\partial y}\mathbf{j}. \quad [30]$$

When applied to a scalar function, such as temperature $T(x,y)$, the *del* operator produces a *gradient* (equation [29]). If applied to a vector function, e.g., to the heat flux vector from equation [28], through a *'dot'* or scalar product operation, we obtain the *divergence* of the vector function, e.g.,

$$div(\mathbf{q}) = \nabla \bullet \mathbf{q} = \frac{\partial q_x}{\partial x} + \frac{\partial q_y}{\partial y}. \quad [31]$$

The divergence of a gradient of a scalar function produces the *Laplacian* of the scalar function, e.g., for the scalar function $\phi(x,y,z)$,

$$\nabla^2 \phi = \nabla \bullet \nabla \phi = div(grad(\phi)) = \frac{\partial}{\partial x}\left(\frac{\partial \phi}{\partial x}\right) + \frac{\partial}{\partial x}\left(\frac{\partial \phi}{\partial y}\right) = \frac{\partial^2 \phi}{\partial x^2} + \frac{\partial^2 \phi}{\partial y^2}.$$

For the heat flux vector of equation [28], the divergence of that heat flux produces:

$$div(\mathbf{q}) = div(-k \cdot grad(T)) = \nabla \bullet (-k \cdot \nabla T) = -k \cdot \nabla^2 T. \quad [32]$$

NOTE: Equations [28] through [32] can be expanded to three dimensions by adding the corresponding derivatives with respect to z. The development presented here is strictly two-dimensional.

Poisson's equation
Consider a two-dimensional body of thickness Δz that produces *Q Joules* of heat per unit volume in every point within the body. This heat is transported by the heat fluxes in both the *x* and *y* directions such that the net flux out of a point (measured by the divergence of the flux vector) is equal to the heat produced *Q*, i.e.,

$$div(\mathbf{q}) = -k \cdot \nabla^2 T = Q\,. \qquad [33]$$

Re-writing equation [33], we find the following Poisson's equation for heat transfer with net heat production in every point:

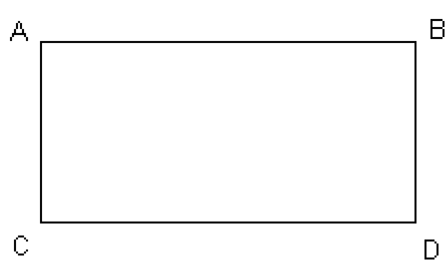
$$\nabla^2 T = \frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} = -\frac{Q}{k}\,. \qquad [34]$$

Laplace's equation
If there is no production of heat in the body of interest, then $Q = 0$, and Poisson's equation [34] reduces to Laplace's equation:

$$\nabla^2 T = \frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} = 0\,. \qquad [35]$$

Neumann boundary condition in heat transfer in solids
To incorporate a Neumann-type of boundary condition in heat transfer in solids we would need to postulate the heat flux at a given boundary. For example, if in the rectangular domain shown below, we indicate that the heat flux through, say, face BD is $q_x = q_o$, then the corresponding boundary condition is:

$$\left.\frac{\partial T}{\partial x}\right|_{BD} = -\frac{q_o}{k}\,. \qquad [36]$$



If face BD is thermally insulated, then no heat flux can occur through it, and the proper boundary condition is $\partial T/\partial x|_{BD} = 0$.

The handling of Neumann-type of boundary conditions in the solution of Laplace's equation is presented in detail in a separate document on two-dimensional potential flow solutions.