

### 2.6.3 Numerical Solution by Gauss-Seidel Iteration

A finite difference solution results in a system of algebraic equations that must be solved simultaneously. In Section 2.6.2, we looked at placing these equations into a matrix equation ( $\underline{\underline{A}}\underline{\underline{X}}=\underline{\underline{b}}$ ) that was solved by a single matrix inversion (or the equivalent mathematical manipulation). The matrix  $\underline{\underline{A}}$  is large ( $M \times N$  x  $M \times N$ ) but sparse and therefore MATLAB can handle it relatively efficiently. However, an alternative technique, Gauss-Seidel iteration, can also be used to approximately solve the system of equations using an iterative technique. The Gauss-Seidel method requires much less memory than the direct matrix solution method and in some cases it may require less computational effort as well.

The algebraic equations that are obtained by the numerical model do not need to be placed into a matrix equation form in order to carry out Gauss-Seidel iteration. Rather, the temperature matrix is initially set up using a set of reasonable guess values for the temperature at each node. The finite difference equation that results from an energy balance on each control volume is solved explicitly for the temperature of the node,  $T_{i,j}$ , as a function of the temperatures of the surrounding nodes. Using this set of equations, each node in the matrix  $T_{i,j}$  is calculated in a logical order using the current values of the temperature of the surrounding nodes. The difference between the original and recalculated value of each temperature is computed and the maximum value of this error is recorded for each iteration. The process is repeated (i.e., the Gauss-Seidel process is iterated) until the numerical error associated with the iteration is less than some tolerance (i.e., the iteration is complete). The Gauss-Seidel iteration process is illustrated using the EPL mask problem that was discussed in Section 2.6.2.

The finite difference equations that describe the energy balances on the internal nodes were given by Eq. (2-211) and do not change:

$$\begin{aligned} & \frac{k \Delta y t h}{\Delta x} (T_{i-1,j} - T_{i,j}) + \frac{k \Delta y t h}{\Delta x} (T_{i+1,j} - T_{i,j}) + \\ & \frac{k \Delta x t h}{\Delta y} (T_{i,j-1} - T_{i,j}) + \frac{k \Delta x t h}{\Delta y} (T_{i,j+1} - T_{i,j}) + \\ & \dot{q}_{inc}'' p d \Delta x \Delta y = 2 \bar{h} \Delta x \Delta y (T_{i,j} - T_{\infty}) \\ & \text{for } i = 2..(M-1) \text{ and } j = 2..(N-1) \end{aligned} \quad (E4-1)$$

The boundary conditions are also the same:

$$T_{1,j} = T_{mh} \text{ for } j = 1..N \quad (E4-2)$$

$$T_{m,j} = T_{mh} \text{ for } j = 1..N \quad (E4-3)$$

$$T_{i,1} = T_{mh} \text{ for } i = 2..(M-1) \quad (E4-4)$$

#### E4: Section 2.6.3 Numerical Solution by Gauss-Seidel Iteration

$$T_{i,n} = T_{mh} \text{ for } i = 2..(M-1) \quad (E4-5)$$

Rather than arranging these equations into the form of a matrix equation, they are solved explicitly for the temperature at each node. The boundary conditions given by Eqs. (E4-2) through (E4-5) are already in this format. The energy balance equations given by Eq. (E4-1) are solved for  $T_{i,j}$ :

$$T_{i,j} = \frac{\left[ 2\bar{h} \Delta x \Delta y T_{\infty} + \dot{q}_{inc}'' pd \Delta x \Delta y + \frac{k \Delta y th}{\Delta x} (T_{i-1,j} + T_{i+1,j}) + \frac{k \Delta x th}{\Delta y} (T_{i,j-1} + T_{i,j+1}) \right]}{\left[ 2 \frac{k \Delta y th}{\Delta x} + 2 \frac{k \Delta x th}{\Delta y} + 2\bar{h} \Delta x \Delta y \right]} \quad (E4-6)$$

for  $i = 2..(M-1)$  and  $j = 2..(N-1)$

The solution using Gauss-Seidel iteration is implemented in the MATLAB function EPL\_Mask\_GS. The header, input section, and grid setup sections remain the same as they were in the function EPL\_Mask that was developed in Section 2.6.2:

```
function[x,y,T_C]=EPL_Mask_GS(M,N)

%[]=EPL_Mask_GS
%
% This function determines the temperature distribution in an EPL_Mask using Gauss-Seidel iteration
%
% Inputs:
% M - number of nodes in the x-direction (-)
% N - number of nodes in the y-direction (-)

%INPUTS
W=0.001;           % width of membrane (m)
th=0.75e-6;        % thickness of membrane (m)
q_dot_flux=5000;   % incident energy (W/m^2)
k=150;             % conductivity (W/m-K)
T_mh=20+273.2;     % mask holder temperature (K)
T_infinity=20+273.2; % ambient air temperature (K)
h_bar=20;          % heat transfer coefficient (W/m^2-K)

Biot=h_bar*th/k;    % Biot number

%Setup grid
for i=1:M
    x(i,1)=(i-1)*W/(M-1);
end
DELTAx=W/(M-1);
for j=1:N
    y(j,1)=(j-1)*W/(N-1);
end
DELTAy=W/(N-1);
```

The subfunction pd\_f also remains the same:

```
function[pd]=pd_f(x,y,W)

% [pd]=pd_f(x,y,W)
%
% This sub-function returns the pattern density of the EPL mask
%
% Inputs:
% x - x-position (m)
% y - y-position (m)
% W - dimension of mask (m)
% Output:
% pd - pattern density (-)

    pd=0.1+0.5*x*y/W^2;
end
```

The temperature matrix is initialized with all of its values set equal to the mask structure temperature,  $T_{mh}$ .

```
T=ones(M,N)*T_mh; % initialize temperature matrix
```

The temperatures along the boundary are specified by Eqs. (E4-2) through (E4-5) and do not need to be part of the iteration since they do not depend on the values of the adjacent nodes; note that alternative boundary conditions (e.g., convection or heat flux) must be included within the iteration process.

```
% boundary conditions
for j=1:M
    T(1,j)=T_mh;
    T(M,j)=T_mh;
end
for i=2:(M-1)
    T(i,1)=T_mh;
    T(i,N)=T_mh;
end
```

The iteration must be placed in a while loop so that it continues until the temperature distribution stops changing by more than a specified tolerance between subsequent iterations. The convergence criteria,  $tol$ , is specified in terms of the maximum change in any of the nodal temperatures during the iteration. Therefore, the while loop terminates when the value of the variable  $err$ , which is defined as the maximum absolute value in the change between successive iterations, is less than the variable  $tol$ . The value of the variable  $err$  is initially set to a value that is larger than the tolerance in order to ensure that at least one iteration occurs.

```
tol=0.00001; % stopping criteria – max. change in temperature (K)
err=999; % initial value of the error - set > tol
while(err>tol)

end
```

#### E4: Section 2.6.3 Numerical Solution by Gauss-Seidel Iteration

Within the while loop, the value of the maximum error associated with the iteration (the variable `err_max`) is set to 0. The initial temperature of each node is recorded in the scalar variable `T_old` and the nodal temperature is re-assigned using Eq. (E4-6). The absolute value of the difference between the variable `T_old` and `T(i,j)` is computed and assigned to the variable `err`. If the local error, `err`, is greater than the current value of `err_max` then `err_max` is replaced with `err`. In this way, the value of `err_max` at the conclusion of the iteration will be the maximum change in temperature that occurred during the iteration.

```
err_max=0;      % initialize maximum value of the error
% internal node energy balances
for i=2:(M-1)
    for j=2:(N-1)
        T_old=T(i,j); % old value of temperature
        T(i,j)=(2*h_bar*DELTAx*DELTAy*T_infinity+q_dot_flux*...
            pd_f(x(i),y(j),W)*DELTAx*DELTAy+k*DELTAy*th*...
            (T(i-1,j)+T(i+1,j))/DELTAx+k*DELTAx*th*...
            (T(i,j-1)+T(i,j+1))/DELTAy)/(2*k*DELTAy*...
            th/DELTAx+2*k*DELTAx*th/DELTAy+2*h_bar*DELTAx*DELTAy);
        err=abs(T_old-T(i,j)); % compute error between old and new temps.
        if (err>err_max)
            err_max=err;
        end
    end
end
end
```

The temperature is converted to degree Celsius:

```
T_C=T-273.15;      % temperature in C
```

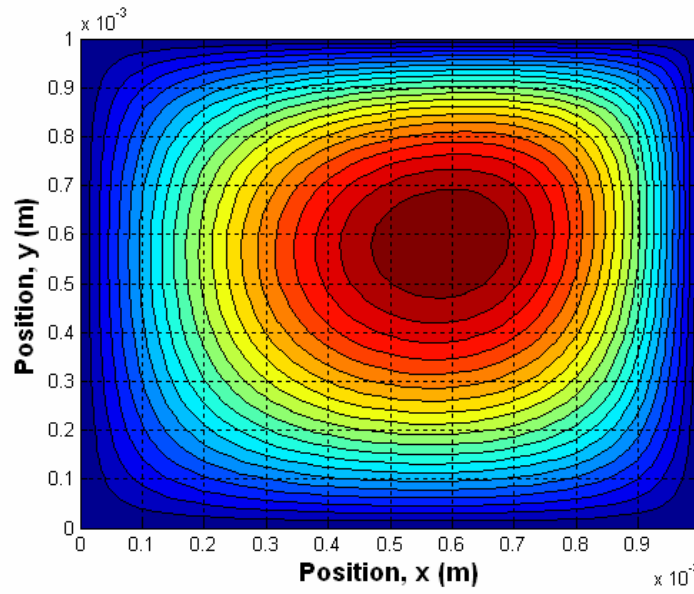
Run the function from the command window by entering:

```
>> M=50;
>> N=50;
>> [x,y,T_C]=EPL_Mask_GS(M,N);
```

Use the results to create a filled contour plot by entering

```
>> contourf(x,y,T_C);
```

The contour plot is shown in Figure E4-1.



**Figure E4-1: Contour plot of temperature distribution obtained using Gauss-Seidel iteration.**

The Gauss-Seidel technique is intuitive and easy to use on large or small problems. However, you should beware that the convergence tolerance must be carefully selected. Because the convergence criterion is based on the error between iterations, the same convergence criteria may lead to very different results. The direct solution technique presented in Section 2.6.2 is generally preferred over an iterative technique like Gauss-Seidel iteration if a tool such as MATLAB that provides a computationally efficient means of solving sparse matrices is available.