Numerical Methods for the Solution of Elliptic Partial Differential Equations

David Keffer
Department of Chemical Engineering
University of Tennessee, Knoxville
September 1999

Table of Contents

Introduction

A linear elliptic PDE has the general form :

$$\nabla\left(\underline{c}(x,y,z)\cdot\nabla U(x,y,z)\right)+\underline{b}(x,y,z)\cdot\nabla U(x,y,z)+a(x,y,z)U(x,y,z)=f(x,y,z) \qquad (0.1)$$

which can be rewritten as:

$$\begin{aligned}\underline{c}(x,y,z)\cdot\nabla^2U(x,y,z)+\left[\nabla\underline{c}(x,y,z)+\underline{b}(x,y,z)\right]\cdot\nabla U(x,y,z)\\ +a(x,y,z)U(x,y,z)=f(x,y,z)\end{aligned} \qquad (0.2)$$

and which, when the coefficients are constants can be written as:

$$\underline{c}\cdot\nabla^2U(x,y,z)+\underline{b}\cdot\nabla U(x,y,z)+aU(x,y,z)=f(x,y,z) \qquad (0.3)$$

which, if we abandon matrix notation becomes

$$c_x\frac{\partial^2U}{\partial x^2}+c_y\frac{\partial^2U}{\partial y^2}+c_z\frac{\partial^2U}{\partial z^2}+b_x\frac{\partial U}{\partial x}+b_y\frac{\partial U}{\partial y}+b_z\frac{\partial U}{\partial z}+aU=f \qquad (0.4)$$

if we have variation in U in three spatial dimensions, or

$$c_x\frac{\partial^2U}{\partial x^2}+c_y\frac{\partial^2U}{\partial y^2}+b_x\frac{\partial U}{\partial x}+b_y\frac{\partial U}{\partial y}+aU=f \qquad (0.5)$$

if we have variation in U in two spatial dimensions. If we have variation in U in only one spatial dimension, then we have

$$c_x\frac{\partial^2U}{\partial x^2}+b_x\frac{\partial U}{\partial x}+aU=f \qquad (0.6)$$

which is an ordinary differential equation, which we already know how to solve, numerically and analytically.

Some simple and famous examples of elliptic equations are

- the two-dimensional Laplace equation: $\frac{\partial^2T}{\partial x^2}+\frac{\partial^2T}{\partial y^2}=0$ (0.7)
- the three-dimensional Laplace equation: $\nabla^2U=0$ (0.8)
- the two-dimensional Poisson equation: $\frac{\partial^2T}{\partial x^2}+\frac{\partial^2T}{\partial y^2}=f(x,y)$ (0.9)

1. Single Linear Elliptic PDEs

A. Method of Transformation to a system of parabolic PDEs

An elliptic PDE does not have time as an independent variable. It is a PDE because it has at least two independent spatial dimensions. Because it does not have a time dependence, we don't naturally think to solve it using a numerical integration method, like the Runge-Kutta method, that we used for ODEs, as well as for parabolic and hyperbolic PDEs.

Of course, we could use that technique. The mathematics of our solution methodology doesn't care whether our independent variable is labeled t for time or x for space. Therefore, we could take the two-dimensional version of equation 0.2

$$\underline{c} \cdot \nabla^2 U + \left[\nabla \underline{c} + \underline{b}\right] \cdot \nabla U + aU = f \tag{1.1}$$

which can be rewritten as

$$c_x \frac{\partial^2 U}{\partial x^2} + c_y \frac{\partial^2 U}{\partial y^2} + \left(\frac{\partial c_x}{\partial x} + b_x\right)\frac{\partial U}{\partial x} + \left(\frac{\partial c_y}{\partial y} + b_y\right)\frac{\partial U}{\partial y} + aU = f \tag{1.2}$$

where I have decided not to write the explicit (x,y,z) dependence of a, b, c, f, and U, and consider y as t and x as x and solve it as we did a hyperbolic PDE. First we break the single second order PDE into two first order PDEs using the substitution:

$$U_1 = U \text{ and } \quad U_2 = \frac{\partial U}{\partial y} \tag{1.3}$$

Then we can write the first equation as

$$\frac{\partial U_1}{\partial y} = U_2 \tag{1.4}$$

and the second equation as

$$c_y \frac{\partial^2 U}{\partial y^2} = f - c_x \frac{\partial^2 U}{\partial x^2} + -\left(\frac{\partial c_x}{\partial x} + b_x\right)\frac{\partial U}{\partial x} - \left(\frac{\partial c_y}{\partial y} + b_y\right)\frac{\partial U}{\partial y} - aU \tag{1.5}$$

$$\frac{\partial U_2}{\partial y} = \frac{1}{c_y}\left[f - c_x \frac{\partial^2 U_1}{\partial x^2} + -\left(\frac{\partial c_x}{\partial x} + b_x\right)\frac{\partial U_1}{\partial x} - \left(\frac{\partial c_y}{\partial y} + b_y\right)U_2 - aU_1\right] \tag{1.6}$$

Equation (1.4) and (1.6) form a system of two linear parabolic PDEs which are entirely consistent with the linear elliptic PDE of equation (1.1).

Since we already know how to solve a system of parabolic PDEs, we are done. We use our code for solving a system of parabolic PDEs to solve the elliptic PDE.

Note to ourselves: No one solves Elliptic PDEs using this technique.
Why not?
People are crazy and they like to make things more complicated than they need to be. We have a technique for solving systems of parabolic PDEs that is guaranteed to work. The only place where it wouldn't work is where c_y equals 0 in equation (1.6).

A second reason people don't solve elliptic PDEs as systems of parabolic PDEs is because in doing so, you make a distinction between x and y. In the above formulation, we treat x as a spatial dimension. If we follow our previous work, then we use a centered finite difference method for the first and second partial derivatives of U with respect to x. The centered finite difference formulae are accurate to order h^2. Now, when we solved parabolic PDEs, we did not use a centered finite difference formulae for the time variable since that would have added an additional unknown to the problem. Instead, we used a second-order Runge-Kutta method.

The third and biggest reason that people don't solve elliptic PDEs as systems of parabolic PDEs is because a properly posed PDE has boundary conditions at $U(x, y = y_o) = U_{o,y}(x)$ and $U(x, y = y_f) = U_{f,y}(x)$. However, in order to solve the parabolic PDE, we need $U(x, y = y_o) = U_{o,y}(x)$ and $\frac{dU}{dy}(x, y = y_f) = \frac{dU}{dy}_{f,y}(x)$. Since we don't have the initial partial derivative information, we have to use a technique like the shooting method (see ChE 301 notes), which we used to solve an ODE BVP as an ODE IVP, in order to solve the elliptic PDE as a system of parabolic PDEs. The big drawback with this is that we now have to converge to the correct final boundary condition. Convergence is never a sure thing, if we can avoid any scheme that relies on convergence, we would like to.

B. Method of the Laplacian Difference Equation

For small systems, we use the centered finite difference formulae for both x and y:

$$\left(\frac{\partial T}{\partial x}\right)_{i,k} \approx \frac{T_{i+1,k} - T_{i-1,k}}{x_{i+1} - x_{i-1}} = \frac{T_{i+1,k} - T_{i-1,k}}{2\Delta x} \tag{1.7}$$

$$\left(\frac{\partial T}{\partial y}\right)_{i,k} \approx \frac{T_{i,k+1} - T_{i,k-1}}{y_{k+1} - y_{k-1}} = \frac{T_{i,k+1} - T_{i,k-1}}{2\Delta y} \tag{1.8}$$

$$\left(\frac{\partial^2 T}{\partial x^2}\right)_{i,k} \approx \frac{\left(\frac{\partial T}{\partial x}\right)_{i+1,k} - \left(\frac{\partial T}{\partial x}\right)_{i,k}}{x_{i+1} - x_i} = \frac{\left(\frac{\partial T}{\partial x}\right)_{i+1,k} - \left(\frac{\partial T}{\partial x}\right)_{i,k}}{\Delta x} \tag{1.9}$$

$$\left(\frac{\partial^2 T}{\partial y^2}\right)_{i,k} \approx \frac{\left(\frac{\partial T}{\partial y}\right)_{i,k+1} - \left(\frac{\partial T}{\partial y}\right)_{i,k}}{y_{k+1} - y_i} = \frac{\left(\frac{\partial T}{\partial y}\right)_{i,k+1} - \left(\frac{\partial T}{\partial y}\right)_{i,k}}{\Delta y} \tag{1.10}$$

$$\left(\frac{\partial^2 T}{\partial x^2}\right)_{i,k} \approx \frac{\left(\frac{T_{i+1,k}-T_{i,k}}{\Delta x}\right)-\left(\frac{T_{i,k}-T_{i-1,k}}{\Delta x}\right)}{\Delta x} = \frac{T_{i+1,k}-2T_{i,k}+T_{i-1,k}}{\Delta x^2} \tag{1.11}$$

$$\left(\frac{\partial^2 T}{\partial y^2}\right)_{i,k} \approx \frac{\left(\frac{T_{i,k+1}-T_{i,k}}{\Delta y}\right)-\left(\frac{T_{i,k}-T_{i,k-1}}{\Delta y}\right)}{\Delta y} = \frac{T_{i,k+1}-2T_{i,k}+T_{i,k-1}}{\Delta y^2} \tag{1.12}$$

If we substitute these formulae into equation (1.2), we have a system of linear algebraic equations, which we know how to solve. We have $m_x \cdot m_y$ unknowns where m_x is the number of nodes where the temperature is unknown in the x-dimension and m_y is the number of nodes where the temperature is unknown in the y-dimension. The mapping of the unknowns from (x,y) coordinates in the spatial dimension to the one-dimensional ordering of the matrix which appears in our system of linear algebraic equations is exactly analogous to the mapping that was laid out for the same mapping when solving a single parabolic PDE with variation in 2 spatial dimensions, so see those notes on the 505 website.

The point is that this is trivial and works so long as we don't have too many unknowns.

Note: this technique only works if the problem is linear, but the linear problem can be of the general form of equation (1.1).

C. Liebmann's method

For larger systems, where we don't have the ability to solve the system of linear equations, we can apply a Gauss-Seidel approximation, which when applied to PDEs is known as Liebmann's method. Here we just write:

$$T_{i,k} \approx \frac{T_{i+1,k}+T_{i-1,k}+T_{i,k+1}+T_{i,k-1}}{4} \tag{1.13}$$

Because the matrix is diagonally dominant, repeated applications of this approximation will converge to the true solution.

This alone would work. It is slow. Therefore, people try to speed it up by using the following enhancement after each application of equation (1.11)

$$T_{i,k}^{new} = \lambda T_{i,k}^{j} + (1-\lambda)\lambda T_{i,k}^{j-1} \tag{1.14}$$

where λ is a relaxation parameter with a value between 1 and 2, and $T_{i,k}^{j}, T_{i,k}^{j-1}$ are the values from the present and previous iteration.

This process of (1.11) and (1.12) is cycled through for each node at which the solution unknown and then repeated until the solution profile no longer changes.

Now, convergence, even guaranteed convergence, is an iffy business. It requires initial guesses. I will do everything in my power to avoid having to make initial guesses. Therefore, if at all possible, I will use method B above.

Note, this method will work for Laplace's equation only.

An Example of solving Laplace's Equation using Liebmann's method.

Consider a rectangle plate with the following Dirichlet boundary conditions:

$$\begin{aligned} T(x=0,y) &= 100 \\ T(x=1,y) &= 0 \\ T(x,y=0) &= 50 \\ T(x,y=1) &= 75 \end{aligned}$$

We discretize this grid for n_x and n_y intervals and solve for a variety of values of λ. We therefore have $(n_x+1)(n_y+1)$ total nodes and $(n_x-1)(n_y-1)$ unknown nodes. (The remaining nodes have values given by the boundary conditions. Our initial guess is that all interior nodes are 0. Below we show a table of results for the different values of λ. Our error is calculated as the root mean square error.

$$err = \sqrt{\frac{\sum_{i=2}^{n_x}\sum_{j=2}^{n_y}\left(U_{i,j}^{new} - U_{i,j}^{old}\right)^2}{(n_x-1)(n_y-1)}}$$

We stop the iterative solution procedure when the error is less than $1.0 \cdot 10^{-6}$ or we exceed 100 iterations.

Below are the errors for $n_x = n_y = 4$. (So we have $3^2 = 9$ unknown nodes.) We see that a value of $\lambda = 1.2$ gave the quickest (14 iterations) convergence.

```
lamdba = 1.000000 iteration = 27 , error = 0.000001
lamdba = 1.100000 iteration = 20 , error = 0.000001
lamdba = 1.200000 iteration = 14 , error = 0.000000
lamdba = 1.300000 iteration = 17 , error = 0.000000
lamdba = 1.400000 iteration = 21 , error = 0.000001
lamdba = 1.500000 iteration = 27 , error = 0.000001
lamdba = 1.600000 iteration = 36 , error = 0.000001
lamdba = 1.700000 iteration = 52 , error = 0.000001
lamdba = 1.800000 iteration = 81 , error = 0.000001
lamdba = 1.900000 iteration = 100 , error = 0.001554
```

Below are the errors for $n_x = n_y = 10$. (So we have $9^2 = 81$ unknown nodes.) We see that a value of $\lambda = 1.6$ gave the quickest (40 iterations) convergence.

```
 lamdba = 1.000000 iteration = 100 , error = 0.000230
 lamdba = 1.100000 iteration = 100 , error = 0.000027
 lamdba = 1.200000 iteration = 100 , error = 0.000002
 lamdba = 1.300000 iteration = 82 , error = 0.000001
 lamdba = 1.400000 iteration = 63 , error = 0.000001
 lamdba = 1.500000 iteration = 43 , error = 0.000001
 lamdba = 1.600000 iteration = 40 , error = 0.000000
 lamdba = 1.700000 iteration = 51 , error = 0.000001
 lamdba = 1.800000 iteration = 82 , error = 0.000000
 lamdba = 1.900000 iteration = 100 , error = 0.001717
```

Below are the errors for $n_x = n_y = 50$. (So we have $49^2 = 2401$ unknown nodes.) We see that none of the values of λ converged within the acceptable tolerance in less than 100 iterations. But we see that we had the smallest error after 100 iterations with a value of $\lambda = 1.9$.

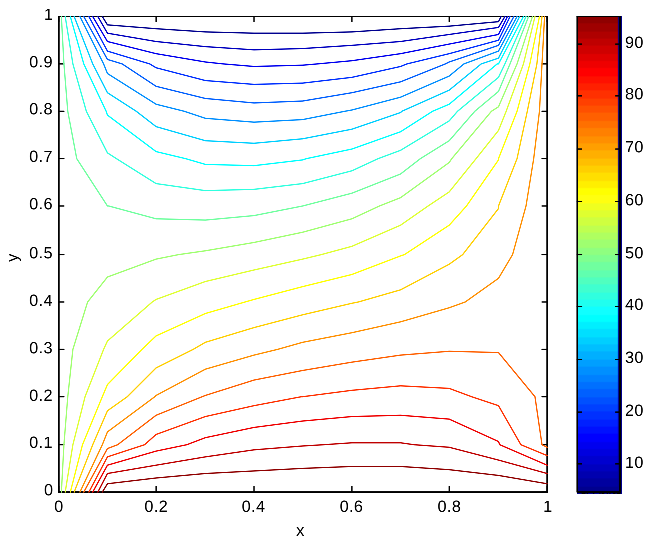
```
 lamdba = 1.000000 iteration = 100 , error = 0.151001
 lamdba = 1.100000 iteration = 100 , error = 0.157871
 lamdba = 1.200000 iteration = 100 , error = 0.164691
 lamdba = 1.300000 iteration = 100 , error = 0.170279
 lamdba = 1.400000 iteration = 100 , error = 0.172255
 lamdba = 1.500000 iteration = 100 , error = 0.166663
 lamdba = 1.600000 iteration = 100 , error = 0.147557
 lamdba = 1.700000 iteration = 100 , error = 0.107225
 lamdba = 1.800000 iteration = 100 , error = 0.042887
 lamdba = 1.900000 iteration = 100 , error = 0.024803
```

What if we change our initial guess of the interior nodes to be the average of the boundary conditions values (100 + 0 + 50 + 75)/4 = 56.25. Below are the errors for $n_x = n_y = 10$. (So we have $9^2 = 81$ unknown nodes.) We see that a value of $\lambda = 1.6$ gave the quickest (39 iterations) convergence. It took one iteration less than the initial guess with all interior nodes equal to zero.

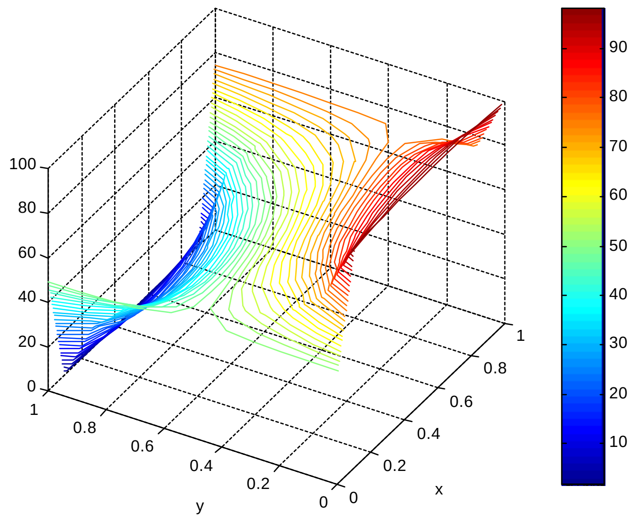
```
 lamdba = 1.000000 iteration = 100 , error = 0.000016
 lamdba = 1.100000 iteration = 100 , error = 0.000002
 lamdba = 1.200000 iteration = 89 , error = 0.000001
 lamdba = 1.300000 iteration = 73 , error = 0.000001
 lamdba = 1.400000 iteration = 57 , error = 0.000001
 lamdba = 1.500000 iteration = 40 , error = 0.000001
 lamdba = 1.600000 iteration = 39 , error = 0.000001
 lamdba = 1.700000 iteration = 51 , error = 0.000001
 lamdba = 1.800000 iteration = 78 , error = 0.000001
 lamdba = 1.900000 iteration = 100 , error = 0.000593
```

What did we learn from this? The value of λ is not only dependent on the boundary conditions but on the size of discretization and the initial guesses as well.

A two-dimensional color contour plot of the converged solution is given below.



A 3-dimensional color contour plot of the converged solution is given below.



The MATLAB code we used to solve this problem is given below.

```
function U = ell_liebmann
clear all
%
%  Solve Laplace's equation using Liebmann's method
%
%  Assume Dirichlet Boundary Conditions
%
%  Author:  David Keffer
%  Department of Chemical Engineering
%  University of Tennessee, Knoxville
%  Last Updated:  October 24, 2000
%

%solve for several values of lambda
lambdav = [1.0:0.1:1.9];
for k = 1:1:length(lambdav)
   lambda = lambdav(k);
    % number of intervals
    nx_int = 10;
    ny_int = 10;
    % number of nodes
    nx = nx_int + 1;
    ny = ny_int + 1;
    %  grid points in x and y
    xo = 0.0;
    xf = 1.0;
    yo = 0.0;
    yf = 1.0;
    dx = (xf - xo)/nx_int;
    dy = (yf - yo)/ny_int;
    xgrid = [xo:dx:xf];
    ygrid = [yo:dy:yf];
    % initialize U and Uold
    U = zeros(nx,ny);
    Uold = zeros(nx,ny);
    %  Fill in Uold with four dirichlet BCs
    for j = 1:1:ny
    % BC for x = xo
        Uold(1,j) = 100.0;
    % BC for x = xf
    Uold(nx,j) = 0.0;
    % BC for y = yo
        Uold(j,1) = 50.0;
    % BC for y = yf
    Uold(j,ny) = 75.0;
    end
    %  fill in the interior nodes of Uold with initial guess
    for i = 2:1:nx-1
    for j = 2:1:ny-1
        %Uold(i,j) = 0.0;
        Uold(i,j) = 56.25;
       end
    end
    U = Uold;
    %
    %  iteratively solve
    %
    maxit = 100;
    err = 100;
    tol = 1.0e-6;
    icount = 0;
    while (icount < maxit & err > tol)
    icount = icount + 1;
    Uold = U;
    for i = 2:1:nx-1
         for j = 2:1:ny-1
            % apply Liebmann's method
           U(i,j) = (U(i+1,j) + U(i-1,j) + U(i,j+1) + U(i,j-1) )*0.25;
           U(i,j) = lambda*U(i,j) + (1-lambda)*Uold(i,j);
        end
      end
      % calculate error
    err = 0.0;
    for i = 2:1:nx-1
       for j = 2:1:ny-1
          err = err + (U(i,j) - Uold(i,j))^2;
       end
    end
    err = sqrt(err/((nx-2)*(ny-2)));
    end
    fprintf(1,' lamdba = %f iteration = %i , error = %f \n',lambda, icount, err);
    % plot
    xmin = xgrid(1);
    xmax = xgrid(nx);
    ymin = ygrid(1);
    ymax = ygrid(ny);
    ncontourlines = 50;
    Umin = min(min(U));
    Umax = max(max(U));
    if (abs(Umin-Umax) < 1.0e-8);
      fprintf(1,'Solution is a flat plane with value = %f \n',Umin)
   else
      plot_dimensions = 1;
      if (plot_dimensions == 3)
           contour3(xgrid, ygrid,U,ncontourlines);
       axis([xmin xmax ymin ymax Umin Umax])
      else
         contour(xgrid, ygrid,U,ncontourlines);
       axis([xmin xmax ymin ymax])
      end
      xlabel('x');
    ylabel('y');
    colorbar
    end
end
```

2. Single Nonlinear Elliptic PDE

Now the trouble starts. The Method of the Laplacian Difference Equation only works for linear elliptic PDEs. The Method of Liebmann only works for Laplace's equation. So we have to use the method of transformation to a system of parabolic PDEs.

The Method of Transformation to a system of parabolic PDEs is our best bet for at least two reasons. First, we already have a code that does 90% of (A). It won't take much effort to add an exterior loop which converges on the final boundary condition. Second, in (A) we have to make initial guesses for the partial derivative $\frac{dU}{dy}(x, y = y_f) = \frac{dU}{dy}_{f,y}(x)$ at each of m_x nodes.

In this method, we begin by rewriting the PDE in the same way that we did for the linear case, namely, by converting it to a system of two parabolic PDEs.

$$\frac{\partial U^{(1)}}{\partial y} = U^{(2)} \tag{2.1}$$

$$\frac{\partial U^{(2)}}{\partial y} = \frac{1}{c_y}\left[f - c_x \frac{\partial^2 U^{(1)}}{\partial x^2} + -\left(\frac{\partial c_x}{\partial x} + b_x \right)\frac{\partial U^{(1)}}{\partial x} - \left(\frac{\partial c_y}{\partial y} + b_y \right)U^{(2)} - aU^{(1)} \right] \tag{2.2}$$

Now, our PDE is nonlinear so the form may not be as we have written it in equation (2.2). Let's be a little more general and rewrite equations (2.1) and (2.2) as general nonlinear functions:

$$\frac{\partial U^{(1)}}{\partial y} = K^{(1)}\left(x, y, U^{(1)}, U^{(2)}, \frac{\partial U^{(1)}}{\partial x}, \frac{\partial U^{(2)}}{\partial x}, \frac{\partial^2 U^{(1)}}{\partial x^2}, \frac{\partial^2 U^{(2)}}{\partial x^2} \right) \tag{2.3}$$

$$\frac{\partial U^{(2)}}{\partial y} = K^{(2)}\left(x, y, U^{(1)}, U^{(2)}, \frac{\partial U^{(1)}}{\partial x}, \frac{\partial U^{(2)}}{\partial x}, \frac{\partial^2 U^{(1)}}{\partial x^2}, \frac{\partial^2 U^{(2)}}{\partial x^2} \right) \tag{2.4}$$

A comment on notation: we will write $U^{(\ell)}(x_i, y_j)$ as $U^{(\ell)}{}_i^j$ so that

j superscripts designate temporal (y) increments
i subscripts designate spatial (x) increments
ℓ and k superscripts inside parentheses designate different functions

To recap what we did in the parabolic PDE case, we first discretized time and space. Second we used the second order Classical Runge-Kutta method to solve the time component of the PDE like an ODE.

$$U^{(\ell)}{}_i^{j+1} = U^{(\ell)}{}_i^j + \frac{\Delta y}{2}\left[K^{(\ell)}{}_i^{j+1} + K^{(\ell)}{}_i^j \right] \tag{2.5}$$

where $K^{(\ell)}{}_i^j$ is the partial derivative of $U^{(\ell)}{}_i^j$ with respect to y,

$$K^{(\ell)j}_{i} = K^{(\ell)}\left(x_i, y_j, \left\{U^j\right\}, \left\{\frac{\partial U}{\partial x}^j\right\}, \left\{\frac{\partial^2 U}{\partial x^2}^j\right\} \right) \tag{2.6}$$

The braces in equation (2.6) now stand for the complete set over both position i and function (ℓ). The second function in equation (2.5) is given by $K(x_i, y_{j+1}, U_i^j + \Delta y K_i^j)$

$$K^{(\ell)j+1}_{i} = K^{(\ell)}\left(x_i, y_{j+1}, \left\{U^{j+1}\right\}, \left\{\frac{\partial U}{\partial x}^{j+1}\right\}, \left\{\frac{\partial^2 U}{\partial x^2}^{j+1}\right\} \right) \tag{2.7}$$

where the value of U is given by

$$U_i^{j+1} \approx U_i^j + \Delta y K_i^j \tag{2.8}$$

Note: this temperature is used not only for the explicit temperatures but is also used in the finite difference formulae to obtain the first and second spatial partial derivatives.

From this point on, we proceed in generally same manner as we did for a system of parabolic PDEs. Hopefully we see that in treating an elliptic equation as a set of parabolic PDEs, we are performing the same transformation that we did in solving the hyperbolic PDE. In this transformation we will that we are missing certain “boundary conditions”.

Remember, the elliptic PDE on a rectangular domain gives boundary conditions (in this case Dirichlet) of the form:

$$U(x = x_o, y) = f_1(y) \qquad U(x = x_f, y) = f_2(y).$$

$$U(x, y = y_o) = f_3(x) \qquad U(x, y = y_f) = f_4(x).$$

In order to solve this as a system of parabolic PDEs, we need boundary conditions on $U^{(1)} = U$ and on $U^{(2)} = \frac{\partial U}{\partial y}$. We have the boundary conditions on $U^{(1)}$ from the original set of BCs. We can obtain the boundary conditions on $U^{(2)}$ by realizing that $U^{(2)} = \frac{\partial U^{(1)}}{\partial y}$ so

$$U^{(1)}(x = x_o, y) = f_1(y) \qquad U^{(1)}(x = x_f, y) = f_2(y).$$

$$U^{(2)}(x = x_o, y) = \frac{\partial f_1(y)}{\partial y} \qquad U^{(1)}(x = x_f, y) = \frac{\partial f_2(y)}{\partial y}.$$

As for the initial conditions, we have one and we have to guess the other along the lines of the "shooting method" to obtain the fourth boundary condition, $U(x, y = y_f) = f_4(x)$. Thus our initial conditions become:

$$U(x, y = y_o) = f_3(x) \qquad \frac{\partial U}{\partial y}(x, y = y_o) = f_{guess}(x).$$

We are going to have to make guesses for the value of the slope at all of the nodes. (That stinks but that's life.)

3. Systems of Linear Elliptic PDEs

Using the Method of the Laplacian Difference Equation, a single linear elliptic PDE can be converted to a system of $m_x \cdot m_y$ linear algebraic equations. Analogously a system of n coupled linear elliptic PDEs can be converted to a system of $n \cdot m_x \cdot m_y$ linear algebraic equations. it really doesn't matter how big the system of equations is, the methodology needed to solve it remains unchanged. The only limitation is computational power and memory requirements, which we are assuming to be sufficient for our problems.

Example: Consider the problem

$$\frac{\partial^2 U^{(1)}}{\partial x^2} + \frac{\partial^2 U^{(1)}}{\partial y^2} = f_1(x,y) + a_1(x,y)U^{(1)} + b_1(x,y)U^{(2)} \tag{3.1}$$

$$\frac{\partial^2 U^{(2)}}{\partial x^2} + \frac{\partial^2 U^{(2)}}{\partial y^2} = f_2(x,y) + a_2(x,y)U^{(1)} + b_2(x,y)U^{(2)} \tag{3.2}$$

on the rectangle defined by $0 \le x \le 1$ and $0 \le y \le 1$ with the boundary conditions:

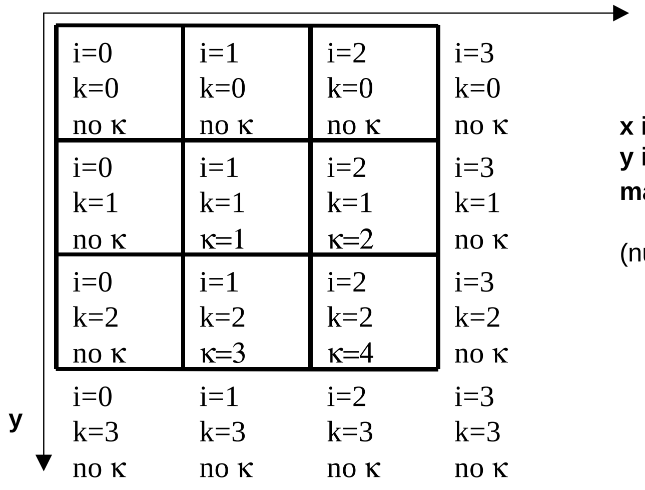
$$U^{(1)}(x = x_o, y) = h_1(y) \text{ , } U^{(1)}(x = x_f, y) = h_2(y), \quad U^{(1)}(x, y = y_o) = h_3(x), \; U^{(1)}(x, y = y_f) = h_4(x) \tag{3.3}$$

$$U^{(2)}(x = x_o, y) = g_1(y) \text{ , } U^{(2)}(x = x_f, y) = g_2(y), \quad U^{(2)}(x, y = y_o) = g_3(x), \; U^{(2)}(x, y = y_f) = g_4(x) \tag{3.4}$$

with $m_x = 3$ and $m_y = 3$.

Pictorially we have four nodes with 2 unknowns each:

node numbering



x index: i = 0,1,2,m_x=3
y index: k = 0,1,2,m_y=3
matrix index: $\kappa = $ 1,2,3,m_u=4

(numbers refer to nodes located to the upper left)

Using the finite-centered difference formulae in equations (1.6)-(1.10) for the partial derivatives in equations (3.1) and (3.2) we have

$$\frac{U^{(1)}{}_{i+1,k} - 2U^{(1)}{}_{i,k} + U^{(1)}{}_{i-1,k}}{\Delta x^2} + \frac{U^{(1)}{}_{i,k+1} - 2U^{(1)}{}_{i,k} + U^{(1)}{}_{i,k-1}}{\Delta y^2} = f_1(x,y) + a_1(x,y)U^{(1)} + b_1(x,y)U^{(2)} \tag{3.5}$$

$$\frac{U^{(2)}{}_{i+1,k} - 2U^{(2)}{}_{i,k} + U^{(2)}{}_{i-1,k}}{\Delta x^2} + \frac{U^{(2)}{}_{i,k+1} - 2U^{(2)}{}_{i,k} + U^{(2)}{}_{i,k-1}}{\Delta y^2} = f_2(x,y) + a_2(x,y)U^{(1)} + b_2(x,y)U^{(2)} \tag{3.6}$$

We can write an equation such as (3.5) and (3.6) for each node where $U^{(1)}$ and $U^{(2)}$ are unknown. In this case, since we have four unknown nodes, we have 8 unknown variables and 8 equations. We can easily solve the 8 by 8 matrix.

$$\underline{\underline{J}}\,\underline{x} = \underline{R} \tag{3.7}$$

where the vector of unknowns is

$$\underline{x} = \begin{bmatrix} U^{(1)}_{1,1} \\ U^{(1)}_{2,1} \\ U^{(1)}_{1,2} \\ U^{(1)}_{2,2} \\ U^{(2)}_{1,1} \\ U^{(2)}_{2,1} \\ U^{(2)}_{1,2} \\ U^{(2)}_{2,2} \end{bmatrix}$$

The matrix is

$$\underline{\underline{J}} = \begin{bmatrix} J_{1,1}^{ul} & \frac{1}{\Delta x^2} & \frac{1}{\Delta y^2} & 0 & J_{1,1}^{ur} & 0 & 0 & 0 \\ \frac{1}{\Delta x^2} & J_{2,1}^{ul} & 0 & \frac{1}{\Delta y^2} & 0 & J_{2,1}^{ur} & 0 & 0 \\ \frac{1}{\Delta y^2} & 0 & J_{1,2}^{ul} & \frac{1}{\Delta x^2} & 0 & 0 & J_{1,2}^{ur} & 0 \\ 0 & \frac{1}{\Delta y^2} & \frac{1}{\Delta x^2} & J_{2,2}^{ul} & 0 & 0 & 0 & J_{2,2}^{ur} \\ J_{1,1}^{ll} & 0 & 0 & 0 & J_{1,1}^{lr} & \frac{1}{\Delta x^2} & \frac{1}{\Delta y^2} & 0 \\ 0 & J_{2,1}^{ll} & 0 & 0 & \frac{1}{\Delta x^2} & J_{2,1}^{lr} & 0 & \frac{1}{\Delta y^2} \\ 0 & 0 & J_{1,2}^{ll} & 0 & \frac{1}{\Delta y^2} & 0 & J_{1,2}^{lr} & \frac{1}{\Delta x^2} \\ 0 & 0 & 0 & J_{2,2}^{ll} & 0 & \frac{1}{\Delta y^2} & \frac{1}{\Delta x^2} & J_{2,2}^{lr} \end{bmatrix} \tag{3.5}$$

where

$$J_{i,j}^{ul} = -\frac{2}{\Delta x^2} - \frac{2}{\Delta y^2} - a_1(x_i, y_j) \tag{3.6}$$

$$J_{i,j}^{ll} = -a_2(x_i, y_j) \tag{3.7}$$

$$J_{i,j}^{ur} = -b_1(x_i, y_j) \tag{3.8}$$

$$J_{i,j}^{lr} = -\frac{2}{\Delta x^2} - \frac{2}{\Delta y^2} - b_2(x_i, y_j) \tag{3.9}$$

The residual has the form:

$$\underline{R} = \begin{bmatrix} f_1(x_1,y_1) - \frac{1}{\Delta x^2} h_1(y_1) - \frac{1}{\Delta y^2} h_3(x_1) \\ f_1(x_2,y_1) - \frac{1}{\Delta x^2} h_2(y_1) - \frac{1}{\Delta y^2} h_3(x_2) \\ f_1(x_1,y_2) - \frac{1}{\Delta x^2} h_1(y_2) - \frac{1}{\Delta y^2} h_4(x_1) \\ f_1(x_2,y_2) - \frac{1}{\Delta x^2} h_2(y_2) - \frac{1}{\Delta y^2} h_4(x_2) \\ f_2(x_1,y_1) - \frac{1}{\Delta x^2} g_1(y_1) - \frac{1}{\Delta y^2} g_3(x_1) \\ f_2(x_2,y_1) - \frac{1}{\Delta x^2} g_2(y_1) - \frac{1}{\Delta y^2} g_3(x_2) \\ f_2(x_1,y_2) - \frac{1}{\Delta x^2} g_1(y_2) - \frac{1}{\Delta y^2} g_4(x_1) \\ f_2(x_2,y_2) - \frac{1}{\Delta x^2} g_2(y_2) - \frac{1}{\Delta y^2} g_4(x_2) \end{bmatrix} \tag{3.10}$$

We solve this system of linear algebraic equations for the solution.

4. Systems of Nonlinear Elliptic PDEs

Using the Method of Transformation to a system of parabolic PDEs, a single nonlinear elliptic PDE can be converted to a system of 2 parabolic PDEs, which is solved as a system of $2m_x$ nonlinear ODEs. Analogously a system of n coupled nonlinear elliptic PDEs can be converted to a system of $2n \cdot m_x$ nonlinear ODEs. It really doesn't matter how big the system of equations is, the methodology needed to solve it remains unchanged. The only limitation is computational power and memory requirements, which we are assuming to be sufficient for our problems.

As before, we solve the system of $2n \cdot m_x$ nonlinear ODEs, based on an initial guess of the initial condition of the slope. We see how well that guess delivers us to the final boundary conditions. We then revise our guess of the initial condition of the slope and try again until we obtain the final boundary condition. Methods for updating the guess are given in the ChE 301 notes on the shooting method.