10.34. Numerical Methods Applied to Chemical Engineering

Homework # 1. Linear Algebraic Equation Sets

Assigned Friday 9/6/02, Due Friday 9/13/02

1. Solution of a mass balance problem

For the following separation system, we know the inlet mass flowrate (in *Kg/hr*) and the mass fractions of each species in the inlet (stream 1) and each outlet (streams 2, 4, and 5).

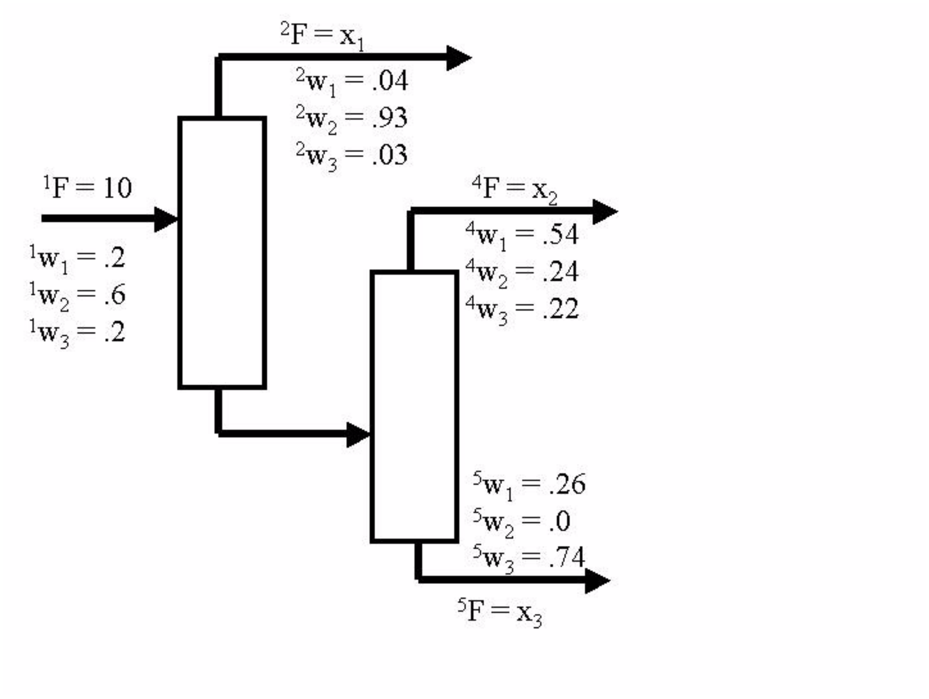

FIGURE 1. Process diagram for separation system

Here we use the notation that ^iF is the mass flow rate of stream # i, and ^iw_j is the mass fraction of species # j in stream # i.

We want to calculate the unknown mass flow rates of each outlet stream. If we define the unknowns as

$$x_1 = {}^2F,\quad x_2 = {}^4F,\quad x_3 = {}^5F \qquad \text{(EQ 1)}$$

and set up the mass balances for

1. the total mass flow rate
2. the mass flow rate of species 1
3. the mass flow rate of species 2

we obtain a set of three linear algebraic equations for the three unknown outlet flow rates.

Convert this set of equations to matrix form, and solve the system by hand using Gaussian elimination with partial pivoting.

Make sure that you clearly identify what calculations you perform at each row and pivot operation.

2. Solution of a 1-D transport problem with finite differences

Background

In this problem, we consider the use of the method of finite differences to convert a boundary value problem from fluid mechanics into a set of linear algebraic equations. We will then modify the standard Gaussian elimination method to solve the resulting linear system in a very efficient manner.

Consider the case of a Newtonian fluid undergoing laminar, pressure-driven flow between two parallel, infinite flat plates separated by a distance B (see figure below). The bottom plate is stationary and the top plate moves at a constant velocity V_{up}. We know the value of the constant dynamic pressure gradient, $\Delta P / \Delta x$, $P = p - \underline{g} \bullet \underline{r}$, and wish to calculate the resulting velocity profile.

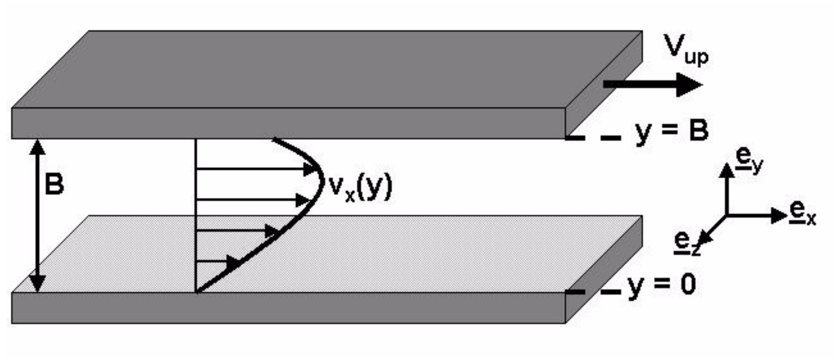

FIGURE 2. Pressure-driven flow between two infinite, parallel, flat plates

If we assume a velocity profile of the form

$$\underline{v}(\underline{r}, t) = v_x(y)\underline{e}_x \qquad \text{(EQ 2)}$$

the equation of continuity for an incompressible fluid, $\nabla \bullet \underline{v} = 0$, is satisfied automatically and the Navier-Stokes equation,

$$\rho \frac{D\underline{v}}{Dt} = -\nabla P + \mu \nabla^2 \underline{v} \qquad \text{(EQ 3)}$$

reduces to

$$0 = -\left(\frac{\Delta P}{\Delta x}\right) + \mu \frac{d^2 v_x}{dy^2} \tag{EQ 4}$$

subject to the no-slip boundary conditions

$$v_x(y{=}0) = 0 \qquad v_x(y{=}\mathrm{B}) = V_{up} \tag{EQ 5}$$

This is a classic problem from fluid mechanics that is solved easily by integrating the differential equation twice and using the boundary conditions to specify the values of the constants of integration. The resulting solution is

$$v_x(y) = V_{up}\left(\frac{y}{B}\right) + \frac{1}{2\mu}\left(\frac{\Delta P}{\Delta x}\right)(y^2 - yB) \tag{EQ 6}$$

We now wish to employ a numerical method to "solve" this problem by converting the boundary value problem into a set of algebraic equations. For this particular problem, there is little actual need to do so since an analytical solution is available. We shall find, however, that the technique that we develop here can be used to obtain numerical approximations to the solution of boundary value problems even when no analytical solution exists.

For this example, we use the conceptually-simple method of finite differences that is based on the following definition of the derivative of a differentiable function $f(x)$,

$$\frac{df}{dx} = \lim_{\Delta x \to 0} \frac{f(x+\Delta x) - f(x-\Delta x)}{2\Delta x} = \lim_{\Delta x \to 0} \frac{f(x+\Delta x) - f(x)}{\Delta x} = \lim_{\Delta x \to 0} \frac{f(x) - f(x-\Delta x)}{\Delta x} \tag{EQ 7}$$

In the limit as $\Delta x \to 0$, all three finite difference approximations agree. In the method of finite differences, we use finite, but "small", values of Δx in one of these approximations to convert the derivatives appearing in a differential equation to algebraic forms. We shall study this method in more detail later; however, for now we merely note that the first approximation formula given above, the central difference approximation, is more accurate than the one-sided differences.

Our differential equation in this example involves the second derivative of the velocity; therefore, we need to construct an algebraic approximation to this higher-order derivative. We place a grid of N points along the computational domain $y \in [0,B]$ (see figure below) at the locations,

$$y_j = j(\Delta y) \qquad \Delta y = \frac{B}{N+1} \qquad j = 1, 2, \ldots, N \tag{EQ 8}$$

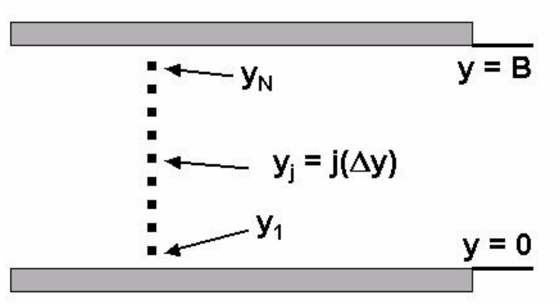

FIGURE 3. Placement of grid points for finite difference computation

At grid point j, we use a central-difference formula to approximate the local value of the second derivative of the velocity,

$$\left.\frac{d^2 v_x}{dy^2}\right|_{y_j} \approx \frac{\left.\left(\frac{dv_x}{dy}\right)\right|_{y_j + (\Delta y)/2} - \left.\left(\frac{dv_x}{dy}\right)\right|_{y_j - (\Delta y)/2}}{\Delta y} \tag{EQ 9}$$

Here, the values of the first derivatives are evaluated at the mid-points between the grid locations. We then use yet other central-difference formulas for these mid-point values,

$$\left.\left(\frac{dv_x}{dy}\right)\right|_{y_j + (\Delta y)/2} \approx \frac{v_x(y_{j+1}) - v_x(y_j)}{\Delta y} \tag{EQ 10}$$

$$\left.\left(\frac{dv_x}{dy}\right)\right|_{y_j - (\Delta y)/2} \approx \frac{v_x(y_j) - v_x(y_{j-1})}{\Delta y} \tag{EQ 11}$$

to obtain the following approximation of the second derivative at each grid point,

$$\left.\frac{d^2 v_x}{dy^2}\right|_{y_j} \approx \frac{v_x(y_{j+1}) - 2v_x(y_j) + v_x(y_{j-1})}{(\Delta y)^2} \tag{EQ 12}$$

In general, this algebraic approximation of the second derivative is not exact, and one must reduce the value of Δy by increasing the number of grid points N until the magnitude of the approximation error is below some acceptable value. For this particular problem, since the true solution is a simple quadratic function, we are lucky and this algebraic approximation is exact.

To "solve" a boundary value problem using the method of finite diffferences, we formulate a set of N algebraic equations for the set of N unknowns $\{y_1, y_2, \ldots, y_N\}$. For each grid point, we need a separate algebraic equation, that we obtain by requiring the differential equation to hold locally,

$$0 = -\left(\frac{\Delta P}{\Delta x}\right) + \mu \frac{d^2 v_x}{dy^2}\bigg|_{y_j} \tag{EQ 13}$$

If we insert the central-difference approximation for the second derivative, the algebraic equation for grid point j is

$$0 = -\left(\frac{\Delta P}{\Delta x}\right) + \mu \frac{v_x(y_{j+1}) - 2v_x(y_j) + v_x(y_{j-1})}{(\Delta y)^2} \tag{EQ 14}$$

We write this in a more compact form by defining the column vector $\underline{v}$ as

$$\underline{v} = \begin{bmatrix} v_1 \\ v_2 \\ : \\ v_N \end{bmatrix} = \begin{bmatrix} v_x(y_1) \\ v_x(y_2) \\ : \\ v_x(y_N) \end{bmatrix} \tag{EQ 15}$$

so that the algebraic equation for grid point j becomes

$$v_{j+1} - 2v_j + v_{j-1} = \frac{(\Delta y)^2}{\mu}\left(\frac{\Delta P}{\Delta x}\right) \tag{EQ 16}$$

If we assemble these equations in matrix form, we obtain the system

$$\begin{bmatrix} -2 & 1 & & & & & \\ 1 & -2 & 1 & & & & \\ & 1 & -2 & 1 & \dots & & \\ & & & : & & & \\ & & & & 1 & -2 & 1 \\ & & & & & 1 & -2 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ : \\ v_{N-1} \\ v_N \end{bmatrix} = \begin{bmatrix} G - v_0 \\ G \\ G \\ : \\ G \\ G - v_{N+1} \end{bmatrix} \tag{EQ 17}$$

where

$$G = \frac{(\Delta y)^2}{\mu}\left(\frac{\Delta P}{\Delta x}\right) \tag{EQ 18}$$

and

$$v_0 = v_x(y=0) \qquad v_{N+1} = v_x(y=B) \tag{EQ 19}$$

To enforce the no-slip boundary conditions, we merely set in the right-hand side vector,

$$v_0 = 0 \qquad v_{N+1} = V_{up} \tag{EQ 20}$$

By this technique, we have converted the original differential equation into a set of algebraic equations for the values of the velocity at each grid point. The question now is how to solve this set of equations. We note that the matrix in the system has a special structure - the only non-zero elements are located along the main diagonal and on the diagonals immediately above and below. Such a matrix is said to be *tridiagonal*. We now consider how acknowledgement of this special structure allows us to solve the set of equations in a fraction of the time required by brute-force Gaussian elimination.

First, from the in-class discussion of Gaussian elimination, we note that the number of FLOP's (Floating Point OPeration's) required to perform the row operations in Gaussian elimination for a system of N unknowns is $2N^3/3$. If we want, in general, to obtain an accurate solution of the differential equation, we may need to use a grid of 100 or more points, so that the number of FLOP's required for Gaussian elimination is on the order of 1 million. In addition to CPU time, the memory requirements of storing the matrix are significant, since a matrix for a system with N unknowns contains N^2 elements, each requiring its own location in memory. As N increases, these numbers become much worse - the number of FLOP's required to perform full Gaussian elimination on a system of 1000 unknowns is on the order of 1 billion.

Because nearly all of the elements of the matrix are already zero, much of the effort of brute-force Gaussian elimination is a waste of time. In fact, we only need to perform one row operation per column to zero the element immediately below the diagonal. Also, since each row only contains three non-zero values, the number of FLOP's required for each row operation is a small number, not on the order of N as is the case in general. For this system, we find that we can remove two of the nested for loops of the Gaussian elimination algorithm so that the total number of FLOP's required to solve the system scales only linearly with N. This is a very important point, as this trick of taking advantage of the structure makes feasible the solution of this set of equations even when the number of grid points is very large.

Also, note that we do not need N^2 locations in memory to store the matrix, but rather need only allocate three column vectors with N components each. One of these column vectors is used to store the matrix values along the main diagonal, and two are allocated for the values in the diagonals immediately above and below.

Your assignment is the following :

A. Write a MATLAB m-file routine that solves a tridiagonal system of linear equations. As input, the routine takes four column vectors of length N. The first three contain the matrix elements along the main diagonal and the diagonals immediately above and below. The fourth is the vector of "right-hand-side" values.

B. Using this routine, solve the linear system above for the case where the fluid is water ($\rho = 10^3 (Kg/m^3)$, $\mu = 10^{-3} Pa \bullet s$), the velocity of the upper plate is 1 *mm/min*, and the separation between plates is 5 *mm*. For these conditions, make a plot of the maximum fluid velocity vs. the pressure gradient, making sure to stop your calculations when the Reynolds' number (defined in terms of the maximum velocity) becomes of the order 1 and the laminar flow solution is no longer expected to be valid. You may use the analytical solution to determine the magintude of pressure gradients to use, but use the numerical method to compute the velocity profile. Submit with your results a MATLAB program showing how the calculations were performed.

3. Solution of a 2-D boundary value problem with finite differences

The advantage to be gained by exploiting matrix structure becomes even more significant when solving boundary value problems in multiple dimensions. In this problem we consider the solution of boundary value problems in multiple spatial dimensions, and the use of the built-in MATLAB elimination solver for sparse systems.

We wish to calculate the velocity profile $v_z(x, y)$ for laminar, pressure-driven flow of a Newtonian fluid through a long rectangular duct. The geometry of the problem, with the no-slip boundary conditions, is diagrammed in the figure below,

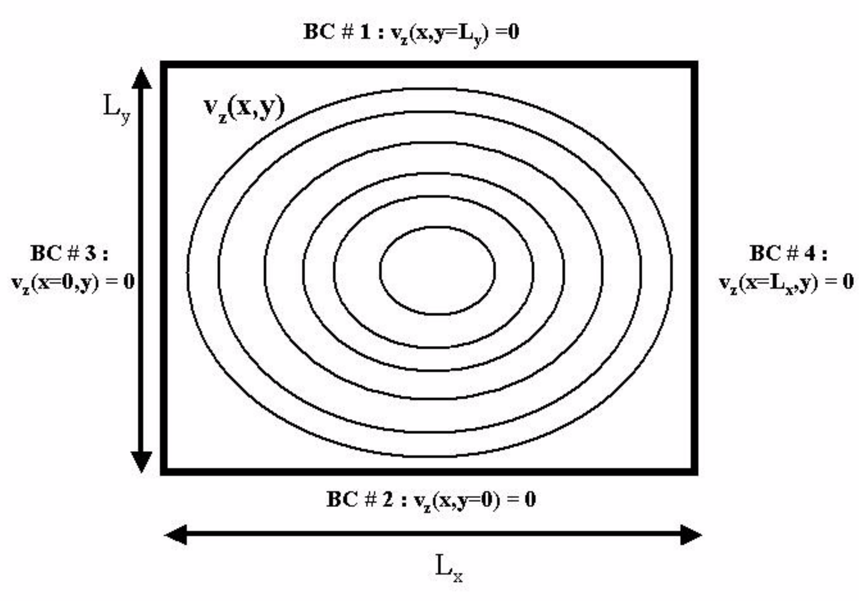
BC # 1 : $v_z(x,y=L_y) = 0$
L_y
$v_z(x,y)$
BC # 3 : $v_z(x=0,y) = 0$
BC # 4 : $v_z(x=L_x,y) = 0$
BC # 2 : $v_z(x,y=0) = 0$
L_x

FIGURE 4. Geometry of pressure-driven flow down a rectangular duct

For a constant dynamic pressure gradient, $\Delta P/\Delta z$, the Navier-Stokes equation for this problem reduces to

$$\frac{\partial^2 v_z}{\partial x^2} + \frac{\partial^2 v_z}{\partial y^2} = \mu^{-1}\left(\frac{\Delta P}{\Delta z}\right) \qquad \text{(EQ 21)}$$

subject to the no-slip boundary conditions noted in the diagram above.

Again, to obtain a numerical solution to this problem using finite differences, we place a computational grid over the domain. For this 2-D problem, we will find the bookeeping a bit easier if we add grid points on the boundaries as well. This increases the number of unknowns, but not by a significant fraction. We use a grid of N_x points in the *x* direction and N_y points in the *y* direction, so that the total number of points on the 2-D grid is N_xN_y. The numbering system used to identify each point in the grid is outlined in the diagram below. For the grid point at (x_i, y_j), we store the local value of the velocity in a column vector of dimension N_xN_y at a "master index" position $k = (i-1)N_y + j$. Note in this diagram the master indices of the neighboring grid points in the *x* and *y* directions. We assume a uniform spacing between grid points in the *x* and *y* directions respectively of

$$\Delta x = L_x/(N_x - 1) \qquad \Delta y = L_y/(N_y - 1) \tag{EQ 22}$$

so that the positions of the grid points are $x_i = (i-1)(\Delta x)$, $y_j = (j-1)(\Delta y)$.

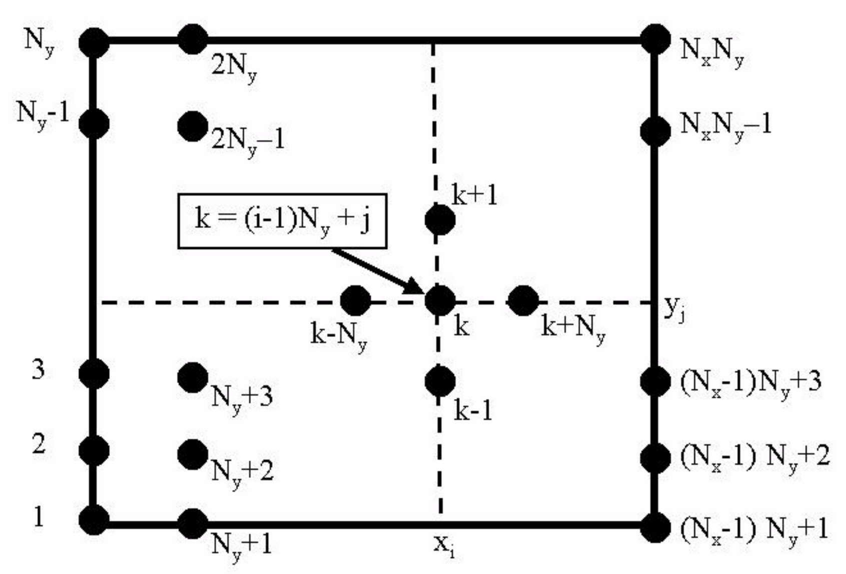

FIGURE 5. Placement of grid points for duct flow problem

We now need to obtain a set of N_xN_y algebraic equations to solve for the velocity at each grid point. For each boundary point, we simply enforce the no-slip boundary conditions. For example, the MATLAB code to set the coefficients of the matrix and right-hand-side vector for all boundary points along boundary # 1 is

```
j = Ny;   % for boundary along y = Ly
for i=1:Nx
    k = (i-1)*Ny + j;
    A(k,k) = 1; % this enforces a simple equality
```

```
        b(k) = 0; % the value of the velocity along the boundary
    end
```

For the interior points, we require that the differential equation be satisfied locally,

$$\left.\frac{\partial^2 v_z}{\partial x^2}\right|_{(x_i,\, y_j)} + \left.\frac{\partial^2 v_z}{\partial y^2}\right|_{(x_i,\, y_j)} = \mu^{-1}\left(\frac{\Delta P}{\Delta z}\right) \qquad \text{(EQ 23)}$$

Let (P) signfify the point at (x_i, y_j) with master index $k = (i-1)N_y + j$. The points immediately to the "north" (N), "south" (S), "east" (E), and "west" (W) (and the values of their master indices) are shown in the figure below.

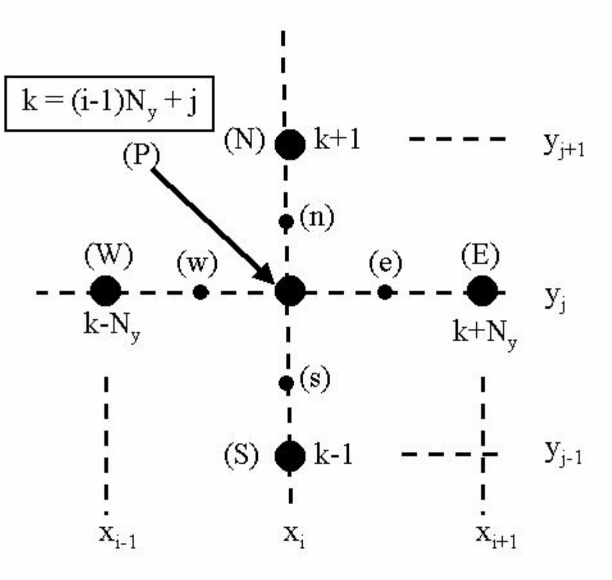

FIGURE 6. Local 2-D grid geometry for finite difference approximations

Let the values of the velocities at each of these grid points be $\{v_{(P)}, v_{(N)}, v_{(S)}, v_{(E)}, v_{(W)}\}$. To approximate the value at (P) of the second-order partial derivative with respect to *x*, we first apply a central difference approximation based on the values of the first derivative at the mid-point locations (e) and (w),

$$\left.\frac{\partial^2 v_z}{\partial x^2}\right|_{(P)} \approx \frac{\left.\left(\frac{\partial v_z}{\partial x}\right)\right|_{(e)} - \left.\left(\frac{\partial v_z}{\partial x}\right)\right|_{(w)}}{\Delta x} \qquad \text{(EQ 24)}$$

For each of the mid-point first derivatives, we similarly use central difference approximations,

$$\left(\frac{\partial v_z}{\partial x}\right)\bigg|_{(e)} \approx \frac{v_{(E)} - v_{(P)}}{\Delta x} \qquad \left(\frac{\partial v_z}{\partial x}\right)\bigg|_{(w)} \approx \frac{v_{(P)} - v_{(W)}}{\Delta x} \tag{EQ 25}$$

This yields the following approximation for the second partial with respect to *x* at (P),

$$\left.\frac{\partial^2 v_z}{\partial x^2}\right|_{(P)} \approx \frac{v_{(E)} - 2v_{(P)} + v_{(W)}}{(\Delta x)^2} \tag{EQ 26}$$

Similarly, the second partial with respect to *y* at (P) is approximated by,

$$\left.\frac{\partial^2 v_z}{\partial y^2}\right|_{(P)} \approx \frac{v_{(N)} - 2v_{(P)} + v_{(S)}}{(\Delta y)^2} \tag{EQ 27}$$

If we now replace this “compass” notation for the grid velocities with the master index labels, the approximation formulas become

$$\left.\frac{\partial^2 v_z}{\partial x^2}\right|_{(x_i,\, y_j)} \approx \frac{v_z(x_{i+1}, y_j) - 2v_z(x_i, y_j) + v_z(x_{i-1}, y_j)}{(\Delta x)^2} = \frac{v_{k+N_y} - 2v_k + v_{k-N_y}}{(\Delta x)^2} \tag{EQ 28}$$

$$\left.\frac{\partial^2 v_z}{\partial y^2}\right|_{(x_i,\, y_j)} \approx \frac{v_z(x_i, y_{j+1}) - 2v_z(x_i, y_j) + v_z(x_i, y_{j-1})}{(\Delta y)^2} = \frac{v_{k+1} - 2v_k + v_{k-1}}{(\Delta y)^2} \tag{EQ 29}$$

Using these finite difference approximations, the algebraic equation for the interior grid point with master index k is

$$\frac{v_{k+N_y} - 2v_k + v_{k-N_y}}{(\Delta x)^2} + \frac{v_{k+1} - 2v_k + v_{k-1}}{(\Delta y)^2} = \mu^{-1}\left(\frac{\Delta P}{\Delta z}\right) \tag{EQ 30}$$

The resulting matrix for this problem is highly structured, but now in addition to the non-zero elements on the diagonals immediately above (N) and immediately below (S) the principal diagonal, we have two additional non-zero diagonals for the (E) and (W) points that are offset N_y diagonals from the principal diagonal (see figure below). This matrix is said to be banded, since the only non-zero elements occur within diagonals offset by a few (compared to N_xN_y) diagonals from the main one. As in the case of a tridiagonal matrix, great savings in CPU time can be achieved by recogonizing the fact during Gaussian elimination that all elements located on diagonals offset by more than N_y positions from the principal diagonal are guaranteed to be zero. For brute-force Gaussian elimination, the number of FLOP’s required to solve the problem is approximately $(2/3)(N_xN_y)^3$. If $N_x = N_y = 100$, the total number of operations required is on the order of 10^{12}, one trillion

FLOP's! By taking into account the banded structure, however, this calculation can be performed many orders of magnitude faster.

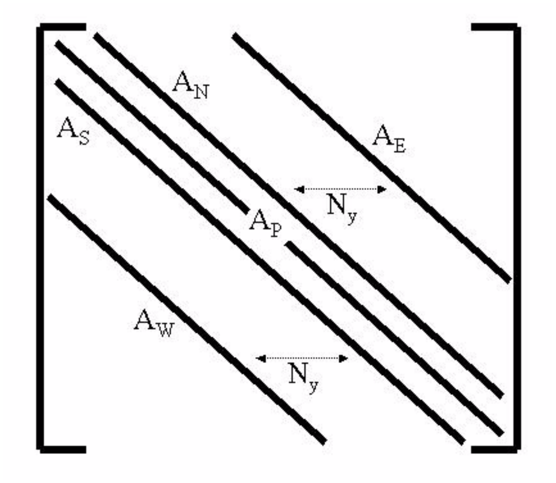

FIGURE 7. Structure of non-zero elements in banded matrix from a 2-D finite difference problem

Your assignment is the following :

A. For this problem, calculate the total number of FLOP's required to perform the row operations of Gaussian elimination, if we take into account the banded structure of the matrix.

B. When solving problems in MATLAB, we can use the built-in Gaussian elimination solver with the sparse matrix notation system to take advantage of such structure without explicitly writing a new Gaussian elimination routine.

We can store compactly in memory any matrix that is sparse (i.e. that is mostly filled with zeros) by allocating for each non-zero element three spaces (two integers and a floating point real number) in memory to record

1. the row number of the non-zero element
2. the column number of the non-zero element
3. the non-zero value of the element

In MATLAB, one can very easily use this sparse-matrix format after learning a few simple commands. The following MATLAB code generates a sparse-matrix representation of the matrix from problem 2 of this homework assignment.

```
% first, allocate memory
num_nz = 3*N;  % upper bound of number of non-zero elements
A = spalloc(N,N,num_nz);  % allocate space in memory
```

```
% set non-zero elements in first row
A(1,1) = -2;
A(1,2) = 1;
% set non-zero elements in last row
A(N,N-1) = 1;
A(N,N) = -2;
% set non-zero elements in “interior” rows
for j=2:(N-1)
    A(j,j-1) = 1;
    A(j,j) = -2;
    A(j,j+1) = 1;
end
spy(A);  make plot to show structure of non-zero elements
```

Then, using the MATLAB commands, we can solve problem 2 by entering the commands,

```
% set right-hand side vector from known value of G
b = G*ones(N,1);
b(N) = b(N) - Vup;
% call built-in MATLAB ellimination solver
v = A\b;
```

Here, when the elimination solver is called with a matrix stored in the sparse matrix format, the routine attempts to perform row and column interchanges to make the matrix as tightly banded as possible, and then applies Gaussian elimination. You get the memory and CPU time advantage of a specially-written banded matrix solver, with much less effort.

For this problem, write a MATLAB program to solve for the velocity profile using the sparse matrix format and the built-in elimination solver. For the following values of the parameters, make a filled countour plot (see command **contourf**) of the velocity profile.

$$\mu = 1 \quad L_x = 1.5 \quad L_y = 1 \quad N_x = 51 \quad N_y = 51 \quad \frac{\Delta P}{\Delta z} = -10^{-4} \qquad \text{(EQ 31)}$$

Consult the on-line MATLAB help utilities **(helpdesk, helpwin**) for further details about these commands. Submit the MATLAB source code and the output plot.