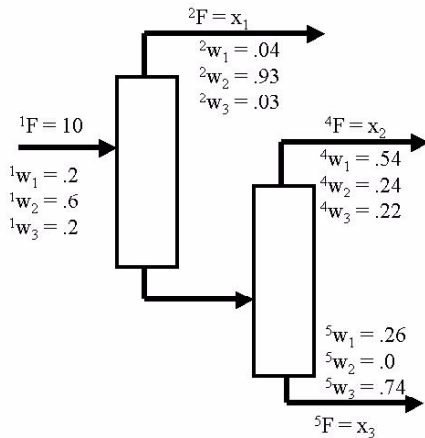


**10.34 Numerical Methods Applied to Chemical Engineering
Homework #1. Linear Algebraic Equation Sets**

SOLUTION

1. Solution of a mass balance problem

In this problem, we are given the following separation system:



We are interested in calculating the unknown mass flow rates of each outlet stream. To do so, we define the unknowns as $x_1 = {}^2F$, $x_2 = {}^4F$, and $x_3 = {}^5F$. By setting up mass balances for (a) the total mass flow rate, (b) the mass flow rate of species 1, and (c) the mass flow rate of species 2, we obtain the following set of three linear algebraic equations for the three unknown outlet flow rates

$$\begin{aligned} x_1 + x_2 + x_3 &= 10 \\ 0.04x_1 + 0.54x_2 + 0.26x_3 &= 0.2 * (10) \\ 0.93x_1 + 0.24x_2 &= 0.6 * (10) \end{aligned}$$

The same system of equations can be written in matrix form

$$\begin{pmatrix} 1 & 1 & 1 \\ 0.04 & 0.54 & 0.26 \\ 0.93 & 0.24 & 0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 10 \\ 2 \\ 6 \end{pmatrix}$$

or in a more compact augmented matrix form

$$(A, \underline{b}) = \begin{bmatrix} 1 & 1 & 1 & 10 \\ 0.04 & 0.54 & 0.26 & 2 \\ 0.93 & 0.24 & 0 & 6 \end{bmatrix}$$

Examining the entries in the first column of the augmented matrix, we notice that the largest entry occurs in row 1 and therefore no interchanging of rows is required at this stage.

Now we want to perform a row operation in order to put a zero in the (2,1) position.

$$\lambda_{21} = \bar{a}_{21}/\bar{a}_{11} = 0.04/1 = 0.04$$

$$\begin{aligned} (A^{(2,1)}, \bar{b}^{(2,1)}) &= \left[\begin{array}{cccc} 1 & 1 & 1 & 10 \\ 0.04 - 0.04*1 & 0.54 - 0.04*1 & 0.26 - 0.04*1 & 2 - 0.04*10 \\ 0.93 & 0.24 & 0 & 6 \end{array} \right] \\ &= \left[\begin{array}{cccc} 1 & 1 & 1 & 10 \\ 0 & 0.5 & 0.22 & 1.6 \\ 0.93 & 0.24 & 0 & 6 \end{array} \right] \end{aligned}$$

Now we want to put a zero in the (3,1) position.

$$\lambda_{31} = \bar{a}_{31}/\bar{a}_{11} = 0.93/1 = 0.93$$

$$\begin{aligned} (A^{(3,1)}, \bar{b}^{(3,1)}) &= \left[\begin{array}{cccc} 1 & 1 & 1 & 10 \\ 0 & 0.5 & 0.22 & 1.6 \\ 0.93 - 0.93*1 & 0.24 - 0.93*1 & 0 - 0.93*1 & 6 - 0.93*10 \end{array} \right] \\ &= \left[\begin{array}{cccc} 1 & 1 & 1 & 10 \\ 0 & 0.5 & 0.22 & 1.6 \\ 0 & -0.69 & -0.93 & -3.3 \end{array} \right] \end{aligned}$$

Looking through the entries in column 2, rows 2 and 3, we observe that the entry with largest absolute value occurs in the third row, and therefore we want to interchange rows 2 and 3.

$$(\bar{A}^{(3,1)}, \bar{b}^{(3,1)}) = \left[\begin{array}{cccc} 1 & 1 & 1 & 10 \\ 0 & -0.69 & -0.93 & -3.3 \\ 0 & 0.5 & 0.22 & 1.6 \end{array} \right]$$

Now we are ready to perform the last row operation and put a zero in the (3,2) position.

$$\lambda_{32} = \bar{a}_{32}/\bar{a}_{22} = 0.5/(-0.69) = -0.7246$$

$$\begin{aligned}
 \begin{pmatrix} \bar{A} \\ \bar{b} \end{pmatrix}^{(3,2)} &= \begin{bmatrix} 1 & 1 & 1 & 10 \\ 0 & -0.69 & -0.93 & -3.3 \\ 0 & 0.5 - (-0.7246)(-0.69) & 0.22 - (-0.7246)(-0.93) & 1.6 - (-0.7246)(-3.3) \end{bmatrix} \\
 &= \begin{bmatrix} 1 & 1 & 1 & 10 \\ 0 & -0.69 & -0.93 & -3.3 \\ 0 & 0 & -0.4539 & -0.7913 \end{bmatrix}
 \end{aligned}$$

Finally, using backward substitution we get the solution

$$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 5.8238 \\ 2.4330 \\ 1.7433 \end{pmatrix}$$

The function that performs the Gaussian elimination is:

```

% show_Gauss_elim.m
%
% function [x,iflag] = show_Gauss_elim(A,b);
%
% This MATLAB m-file shows the method of forward Gaussian elimination
% with backward substitution to directly solve a set of linear equations.
%
% Written by K.J. Beers. 9/7/2001

function [x,iflag] = show_Gauss_elim(A,b);

disp('RUNNING show_Gauss_elim.m');

iflag = 0;

% First, we make copies of A and b that will be modified during
% the solution process.

Ac = A;
bc = b;

% We now set the size of A and initialize the
Ndim = length(b);
if(size(A,1) ~= Ndim)
    error('A # rows wrong');
    iflag = -1;
    return;
elseif (size(A,2) ~= Ndim)
    error('A # columns wrong');
    iflag = -2;
    return;
end
% We next initialize the solution vector to be all zeros.

```

```

x = linspace(0,0,Ndim)';

disp('Initial augmented system matrix : ');
disp([Ac,bc]);

disp('Strike any key to continue ...');
pause;

% Now, we perform forward Gaussian elimination with partial pivoting.

for i=1:(Ndim-1) % sum over columns from left

    disp('Iteration # : '); disp(i);

    % We now determine if a pivot is to be conducted. First, we
    % identify the row at and below row i that has the largest
    % magnitude element in column i.
    [max,ipivot] = max(abs(Ac(i:Ndim,i)));
    ipivot = ipivot + i - 1; % correct for shortened length of Ac(i:Ndim,i)
    if(ipivot ~= i)
        disp(['Performing pivot between rows ', int2str(i), ...
            ' and ', int2str(ipivot)]);
        temp = Ac(i,:);
        Ac(i,:) = Ac(ipivot,:);
        Ac(ipivot,:) = temp;
        temp2 = bc(i);
        bc(i) = bc(ipivot);
        bc(ipivot) = temp2;
        disp([Ac, bc]);
        disp('Strike any key to continue ...');
        pause;
    end

    % Then, we perform the row operations for this column.
    for j=(i+1):Ndim % elements of column i below diagonal

        lambda = Ac(j,i)/Ac(i,i);
        Ac(j,i:Ndim) = Ac(j,i:Ndim) - lambda*Ac(i,i:Ndim);
        bc(j) = bc(j) - lambda*bc(i);

        disp('After row operation, system is : ');
        disp([Ac, bc]);
        disp('Strike any key to continue ...');
        pause;
    end
end

end

% Now, the A matrix is in triangular form, so backward substitution can
% be used to solve the system.

for i=Ndim:-1:1 % sum from bottom
    sum = 0;
    for j=(i+1):Ndim % sum over components already calculated
        sum = sum + Ac(i,j)*x(j);
    end
end

```

```

end
x(i) = (bc(i)-sum)/Ac(i,i);
end

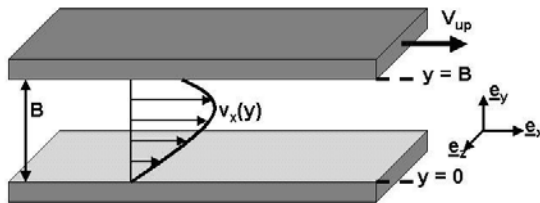
disp(' '); disp('The solution is : '); x % print out results

disp('FINISHED show_Gauss_elim.m');

```

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

In this problem we 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, $\Delta P/\Delta x$, and wish to calculate the resulting velocity profile.



If we assume that the flow is one-dimensional, i.e. $v_y = v_z = 0$ and $v_x = v_x(y)$, the equation of continuity is satisfied automatically and the Navier-Stokes equation reduces to

$$0 = -\frac{dP}{dx} + \mu \frac{d^2 v_x}{dy^2} \quad \text{subject to the no-slip boundary conditions}$$

$$v_x(y=0) = 0 \quad v_x(y=B) = V_{up}$$

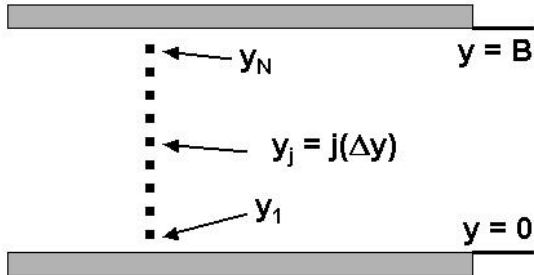
This differential equation is amenable to analytical solution, which has the form

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

The goal of this exercise is to solve the problem numerically using central finite difference technique. The exact solution can be used to check the results we obtain using our numerical method. We can place a grid of N points along the computational domain $y \in [0, B]$ (see figure below). To solve the boundary value problem using the method of

finite differences, we formulate a set of N algebraic equations for the set of N unknowns $\{y_1, y_2, \dots, y_N\}$.

$$y_j = j(\Delta y) \quad \Delta y = \frac{B}{N+1} \quad j = 1, 2, \dots, N$$



After going through the discretization process, we get that the algebraic equation to be satisfied at each grid point j is

$$v_{j+1} - 2v_j + v_{j-1} = \frac{(\Delta y)^2}{\mu} \frac{dP}{dx} \equiv G \quad \text{and}$$

$$v_0 = v_x(y=0) = 0 \quad v_{N+1} = v_x(y=B) = V_{up}$$

The resulting matrix (see EQ 17 in the HW1 handout) is tridiagonal, and therefore it would be a waste of effort to use brute-force Gaussian elimination to solve the system since most of the entries in that matrix are already zero. In the Gaussian elimination algorithm for this system, we can remove one of the nested loops, which will allow us to get to the solution in roughly N number of FLOPS rather than N^3 for a dense matrix.

In this problem we are not given the number of points N . I chose to use $N=100$ ($N=1000$ in calculating maximum pressure gradient from the analytical solution). We need to make sure to stop our calculations when the Reynolds number becomes of the order 1 and the laminar flow solution is no longer expected to be valid. We do know that the value of dP/dx must be negative to ensure fluid flow in the positive x -direction. The smallest possible pressure gradient, zero, will result in planar Couette flow (see lower curve on the velocity vs. position plot). One way to determine the absolute value of the maximum allowable pressure gradient is by taking advantage of the analytical solution, which is available for this simple one-dimensional problem.

%%
 % USING ANALYTICAL SOLUTION TO CALCULATE maximum dP/dx

```
Npts=1000;           % Number of grid points
B = 5e-3;           % Separation b/w plates
Vup = 1/6e4;        % Velocity of the upper plate
density = 1e3;      % Fluid density
visc = 1e-3;        % Fluid viscosity
```

```

delta_y = B/(Npts+1);
dp_dx = -0.06;

% Calculate exact solution
for i = 1:Npts
    y = i*delta_y;
    exact_soln(i) = Vup*(y/B)+1/(2*visc)*dp_dx*(y^2-y*B);
end

Vmax = max(abs(exact_soln));
Re = density*Vmax*B/visc
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

By varying the value of dp_dx in the code above, we can determine that $\Delta P/\Delta x = -0.06$ will result in the Reynolds number of 0.9796, which is close enough to 1.

There is another way to calculate maximum allowable pressure gradient using the analytical solution. By differentiating the expression for velocity profile and setting the derivative equal to zero, we can solve for the value of y, y^* , at which velocity is maximum. Of course, if $\Delta P/\Delta x$ is close to zero, the velocity profile is linear (for $\Delta P/\Delta x=0$) or nearly linear otherwise, and maximum velocity will occur at the upper plate.

$$\frac{dv_x}{dy} = \frac{V_{up}}{B} + \frac{1}{2\mu} \left(\frac{\Delta P}{\Delta x} \right)_{\max} (2y^* - B) = 0$$

$$\Rightarrow y^* = \frac{1}{2} \left[-\frac{V_{up}}{B} \frac{(2\mu)}{(\Delta P/\Delta x)_{\max}} + B \right]$$

By definition, $Re = \frac{\rho V_{\max} B}{\mu}$, and since we want $Re \sim 1$,

$$V_{\max} = \frac{\mu}{\rho B} = V_{up} \left(\frac{y^*}{B} \right) + \frac{1}{2\mu} \left(\frac{\Delta P}{\Delta x} \right)_{\max} (y^{*2} - y^* B)$$

The expression for $(\Delta P/\Delta x)_{\max}$ is then

$$\left(\frac{\Delta P}{\Delta x} \right)_{\max} = \left[\frac{\mu}{\rho B} - V_{up} \frac{y^*}{B} \right] \left[\frac{2\mu}{y^{*2} - y^* B} \right]$$

which is a non-linear equation in $\Delta P/\Delta x$. Solving for $\Delta P/\Delta x$ using MAPLE, we get three roots: 0, -0.0000290, and -0.0613. The root we want to use is the one with the largest absolute value.

Problem 2 code:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
% 10.34 Fall 2002
```

```
% Homework 1, Problem 2
```

```
% INPUTS
```

```
Npts=100;           % Number of grid points
B = 5e-3;           % Separation b/w plates
Vup = 1/6e4;        % Velocity of the upper plate
density = 1e3;      % Fluid density
visc = 1e-3;        % Fluid viscosity
delta_y = B/(Npts+1);
dp_dx_max = -0.06; % Maximum allowable pressure gradient for laminar flow
N_press = 10;       % Number of pts on the velocity vs. dP/dx plot in part (b)
press_grad = linspace(dp_dx_max,0,N_press);
Vmax_vec = zeros(N_press,1);
```

```
% Since we need to vary pressure in part (b), I'm putting all calculations under one
% big 'for' loop. This way, we can generate a velocity vs. pressure gradient plot by
% running this simulation only once.
```

```
for iter = 1:N_press
```

```
    dp_dx = press_grad(iter);
    G = delta_y^2/visc*dp_dx;
```

```
    diag = zeros(Npts,1);
    up_diag = zeros(Npts,1);
    low_diag = zeros(Npts,1);
    rhs = zeros(Npts,1);
    soln = zeros(Npts,1);
    exact_soln = zeros(Npts,1);
```

```
% Here are our four column vectors to serve as inputs in building matrix A.
```

```
for i=1:Npts
```

```
    diag(i) = -2;           % diagonal elements of A
    up_diag(i) = 1;        % upper diagonal elements of A
    low_diag(i) = 1;       % lower diagonal elements of A
    rhs(i) = G;            % right hand side vector
```

```
end
```

```
rhs(Npts) = G-Vup;
```

```
% Using the sparse matrix format, we allocate memory
% for the matrix obtained by central finite differences.
```

```
nzA = 3*Npts;           % # of non-zero elements
A = spalloc(Npts,Npts,nzA); % allocates memory, initializes to 0
```

```
% Now we can fill in the entries of A
```

```
% Internal rows
```

```
for i = 2:Npts-1
```

```
    A(i,i) = diag(i);
    A(i,i-1) = low_diag(i);
    A(i,i+1) = up_diag(i);
```

```
end
```



```

% External rows
i = 1;
A(i,i) = diag(i);
A(i,i+1) = up_diag(i);

i = Npts;
A(i,i) = diag(i);
A(i,i-1) = low_diag(i);

% With matrix A and the right hand side vector assembled, we are now ready to solve
% this linear system using Gaussian elimination.

for j = 1:(Npts-1)
    lambda = A(j+1,j)/A(j,j);
    A(j+1,j) = A(j+1,j) - lambda*A(j,j);
    A(j+1,j+1) = A(j+1,j+1) - lambda*A(j,j+1);
    rhs(j+1) = rhs(j+1) - lambda*rhs(j);
end

% All that is left to do now is backward substitution. Due to the tridiagonal nature of the matrix,
% backward substitution will also scale linearly with N. We will not have, after elimination, non-zero
% values running from the diagonal all the way across the row as is the case in general.

soln(Npts) = rhs(Npts)/A(Npts,Npts);
for i = (Npts-1):-1:1
    soln(i) = (rhs(i)-A(i,i+1)*soln(i+1))/A(i,i);
end

Vmax = max(abs(soln)) % Maximum velocity calculated numerically
Vmax_vec(iter) = Vmax;
Re = density*Vmax*B/visc

% Calculate exact solution
for i = 1:Npts
    y = i*delta_y;
    exact_soln(i) = Vup*(y/B)+1/(2*visc)*dp_dx*(y^2-y*B);
end

% Compare numerical results with the exact solution (for completeness and to make sure that the
% boundary conditions were implemented properly, I'm including boundary points).
soln_plot = zeros(Npts+2,1);
soln_plot(1) = 0;
soln_plot(Npts+2) = Vup;
soln_plot(2:Npts+1) = soln(1:Npts);

exact_soln_plot = zeros(Npts+2,1);
exact_soln_plot(1) = 0;
exact_soln_plot(Npts+2) = Vup;
exact_soln_plot(2:Npts+1) = exact_soln(1:Npts);

yposition = (0:delta_y:B)';
plot(yposition,soln_plot,yposition,exact_soln_plot,'*');
xlabel('y'); ylabel('Velocity');
title('Comparison of the exact and the numerical solutions');
legend('Numerical Solution','Exact Solution');
hold on;

```

```
end % for the 'dP/dx' loop
```

```
% Finally, we can plot maximum velocity as a function of pressure gradient, as required  
% in problem 2(b).
```

```
figure;
```

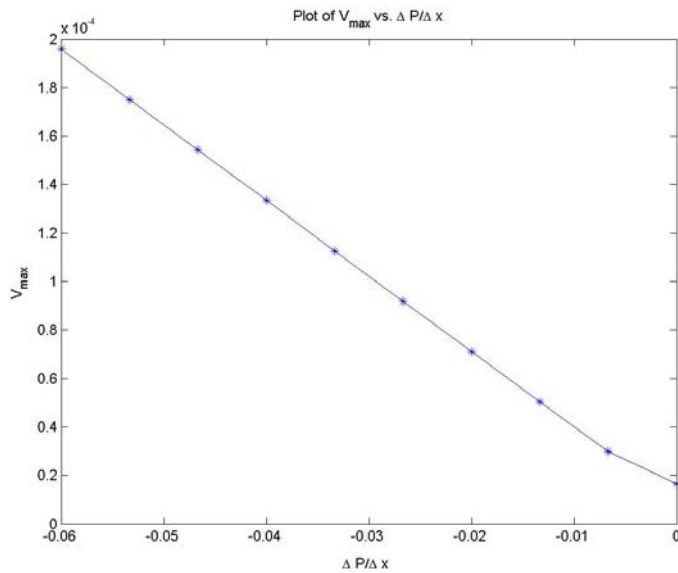
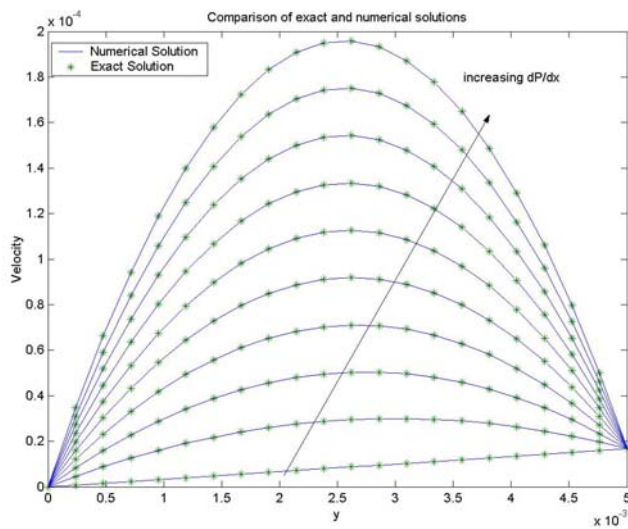
```
plot(press_grad,Vmax_vec,'*-');
```

```
title('Plot of  $V_{\max}$  vs.  $\Delta P/\Delta x$ ');
```

```
xlabel('\Delta P/\Delta x'); ylabel('V_{\max}');
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

The output of this code is two plots: (a) velocity vs. position for both analytical and numerical solutions, and (b) maximum velocity vs. pressure gradient.



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

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

$$\frac{d^2 v_z}{dx^2} + \frac{d^2 v_z}{dy^2} = \frac{1}{\mu} \left(\frac{dP}{dx} \right)$$

subject to no-slip boundary conditions on the four walls of the duct. 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 bookkeeping 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 $N_x N_y$. The numbering system used to identify each point in the grid is outlined in Figure 5 of the HW1 handout. For the grid point at (x_i, y_j) , we store the local value of the velocity in a column vector of dimension $N_x N_y$ at a “master index” position $k = (i-1)N_y + j$. Note in the 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

$$\Delta x = L_x / (N_x - 1) \quad \Delta y = L_y / (N_y - 1)$$

Using the finite difference approximations outlined on page 10 of the assignment sheet, we get the following algebraic equation for the interior grid point with master index k

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

which can be rewritten as

$$A_{k,k+N_y} v_{k+N_y} + A_{k,k} v_k + A_{k,k+1} v_{k+1} + A_{k,k-N_y} v_{k-N_y} + A_{k,k-1} v_{k-1} = b_k$$

where

$$\begin{aligned} A_{k,k+N_y} &= 1/(\Delta x)^2 & A_{k,k+1} &= 1/(\Delta y)^2 \\ A_{k,k} &= -2/(\Delta x)^2 - 2/(\Delta y)^2 & A_{k,k-1} &= 1/(\Delta y)^2 \\ A_{k,k-N_y} &= 1/(\Delta x)^2 & b_k &= \mu^{-1} (\Delta P / \Delta z) \end{aligned}$$

(A) For this part, we are asked to calculate the total number of FLOPs required to perform the row operations of Gaussian elimination by taking into account the banded structure of the matrix. Since we’re looking for an order of magnitude estimate, we can make certain approximations in this count. For one, we will forget that rows of matrix A

corresponding to boundary elements only have one entry. Here is an outline of a “rough” Gaussian elimination algorithm for a matrix containing 5 diagonals with the distance between the main and outer diagonals being N_y .

```
%%%%%%%%%
```

```
for i = 1: N_x*N_y
    :
    for j = (i + 1): (i + N_y)
        :
        for k = i: (i + 2*N_y)
            :
        end
    end
end
end
```

```
%%%%%%%%%
```

For this algorithm we see that the number of FLOPS will scale with $N_x N_y^3$, which is a significant saving as compared with $N_x^3 N_y^3$, which is what is required to perform Gaussian elimination on a dense matrix of the same dimensions.

(B) For this part, we need to write a MATLAB program to solve for the velocity profile using the sparse matrix format and the built-in elimination solver.

```
%%%%%%%%%
```

```
% 10.34 Fall 2002
```

```
% Homework 1, Problem 3
```

```
% duct_flow_2D.m
```

```
%
```

```
% This MATLAB program employs finite differences
```

```
% to compute the velocity profile of a Newtonian
```

```
% fluid during laminar, pressure-driven flow
```

```
% through a rectangular duct.
```

```
%
```

```
% K. Beers. MIT ChE. 9/13/2002
```

```
function iflag_main = duct_flow_2D();
```

```
iflag_main = 0;
```

```
% First, ask for problem parameters
```

```
Lx = input('Enter width in x-direction : ');
```

```

Ly = input('Enter width in y-direction : ');
dp_dz = input('Enter dynamic pressure gradient : ');
mu = input('Enter viscosity : ');
Nx = input('Enter number of x-dir. grid points : ');
Ny = input('Enter number of y-dir. gridpoints : ');

% Now, set 2D grid of points
dx = Lx/(Nx-1);
x = linspace(0,Lx,Nx);
dy = Ly/(Ny-1);
y = linspace(0,Ly,Ny);

% allocate memory for A matrix and b vector
A = spalloc(Nx*Ny,Nx*Ny,5*Nx*Ny);
b = zeros(Nx*Ny,1);

% specify algebraic equations for each grid point

% first, set the interior grid points
r = dx^2/dy^2;
G = -dp_dz/mu*dx^2;
for i=2:(Nx-1)
for j=2:(Ny-1)
    k = (i-1)*Ny+j;
    A(k,k-Ny) = -1;
    A(k,k-1) = -r;
    A(k,k) = 2*(1+r);
    A(k,k+1) = -r;
    A(k,k+Ny) = -1;
    b(k) = G;
end
end

% next, enforce the boundary conditions
% BC # 1, x = 0
i = 1;
for j=1:Ny
    k = (i-1)*Ny+j;
    A(k,k) = 1;
    b(k) = 0;
end
% BC # 2, x = Lx
i = Nx;
for j=1:Ny
    k = (i-1)*Ny+j;
    A(k,k) = 1;
    b(k) = 0;
end
% BC # 3, y = 0
j = 1;
for i=1:Nx
    k = (i-1)*Ny+j;
    A(k,k) = 1;
    b(k) = 0;
end
% BC # 4, y = Ly
j = Ny;

```

```

for i=1:Nx
    k = (i-1)*Ny+j;
    A(k,k) = 1;
    b(k) = 0;
end

% make plot of matrix structure
figure;
spy(A);
title('2D duct flow : Structure of A matrix');

% Now, use the built-in elimination solver to obtain the vector of velocity values.
v = A\b;

% Finally, we make a filled contour plot of
% the solution.

% Currently, the velocities are stacked into a single column vector. The first step is to
% extract the values into a matrix VV, whose (i,j) element contains the velocity at grid point (xi,yj)
VV = zeros(Nx,Ny);
k = 0;
for i=1:Nx
    for j=1:Ny
        k = k + 1;
        VV(i,j) = v(k);
    end
end

% MATLAB's 3-D plot routines expect a different grid numbering system than that used, as can be seen by
% executing the following three lines of code
% [XX,YY] = meshgrid(x,y);
% XX(1:3,1:3), YY(1:3,1:3),
%
% This shows that XX(i,j) and YY(i,j) contain the values x(j) and y(i) respectively.
%
% Therefore, we must call contourf with the transpose of the VV matrix defined above.
%
% We can check that we are using the correct numbering system by executing the following
% lines of code,
% TT = zeros(Nx,Ny);
% for i=1:Nx
%     for j=1:Ny
%         TT(i,j) = x(i) + 5*y(j);
%     end
% end
% figure;
% surf(x,y,TT'); xlabel('x'); ylabel('y');

% We now make the filled contour plot using the correct grid point numbering scheme.
figure;
contourf(x,y,VV',20);
axis equal; colorbar;
xlabel('x'); ylabel('y');
title(['v_z(x,y) ', ...
    '\mu = ', num2str(mu), ...
    '\Delta P/\Delta z = ', num2str(dp_dz), ...
    '\ N_x = ', int2str(Nx), ...

```

```
', N_y = ', int2str(Ny));
```

```
iflag_main = 1;
```

```
save duct_flow_2D.mat;
```

```
return;
```

