1 Introduction

The equations that govern open channel flow at an inclination of θ degrees with the horizontal can be transformed into a Poisson equation by scaling the streamwise velocity u as

$$U(Y,Z) = \frac{u(Y,Z)}{L^2 \rho g \sin(\theta/\mu)} , \tag{1}$$

where L is the length of the square channel, ρ is the fluid density, g is gravitational acceleration, and μ is the fluid's dynamic viscosity. The cross-sectional dimensions in y and z are also normalized by $Y = y/L$ and $Z = Z/L$. Through these scaling procedures, the governing equations map onto a unit square as

$$U_{,YY} + U_{,ZZ} = -1 , \tag{2}$$
$$U(0,Z) = 0 ,$$
$$U_{,Y}(1,Z) = 0 ,$$
$$U(Y,0) = U(Y,1) = 0 . \tag{3}$$

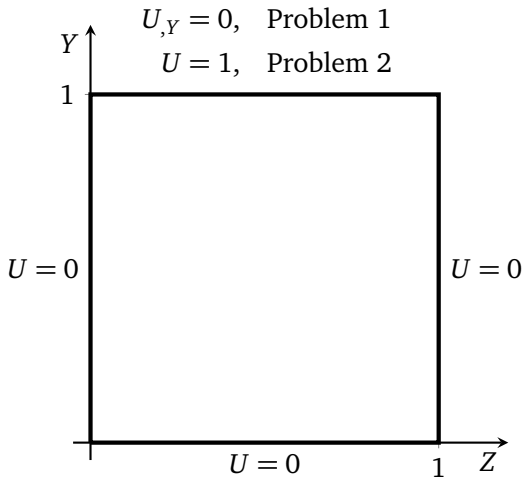

Figure 1: Boundary value problem for Homework 6. No-slip conditions are imposed at three side walls. For Problem 2, the upper boundary is a moving wall.

1.1 Problem 1

Numerically integrate (2) with the stated boundary conditions (3) using the ADI method with $N = M = 101$ grid points in each direction. Implement LU decomposition to solve the tridiagonal systems, and determine convergence by a reduction of the original error by three orders of magnitude.

Plot the contours of U in the Y-Z plane at convergence.

1.2 Problem 2

Change the upper boundary condition to represent a solid lid moving at a constant velocity with

$$U_{,Y}(1,Z) = 0 \quad \longrightarrow \quad U(1,Z) = 1 . \tag{4}$$

Solve this problem using the successive over-relaxation (SOR) method. As in Problem 1, define convergence as a reduction by three orders of magnitude of the initial error in the maximum norm.

Obtain the best estimate for the acceleration parameter ω by numerical experimentation. That is, plot the number of iterations required for convergence as a function of ω, and determine the value of ω that minimizes this function. How does this value of ω compare to the theoretical value?

Plot the contours of U in the Y-Z plane at convergence, and compare the results to Problem 1.

2 METHODOLOGY

2.1 PROBLEM 1

The alternating-direction implicit (ADI) method assumes a pseudo-time derivative ($\partial/\partial t$), such that (2) becomes

$$T_{,t} = U_{,YY} + U_{,ZZ} - \xi = 0\,, \tag{5}$$

where ξ is the inhomogeneous source term. Our solution for (2) can be interpreted as the steady-state solution to (5) as $t \to \infty$. This equation is parabolic in space and elliptic in time. Of course, the transient is not physical, so it acceptable to advance the solution in time using the fully implicit Euler method.

The ADI method breaks the problem into two directions and solves each over two half-time steps. Discretizing the problem using second-order central differences, defining the inhomogeneous source term as $\xi(Y,Z) = -1$, and letting the grid spacing in both directions be equal ($h \equiv \Delta y = \Delta z$), we obtain

$$U^{n+1/2}_{i+1,j} - (2+\rho)U^{n+1/2}_{i,j} + U^{n+1/2}_{i-1,j} = -U^{n}_{i,j+1} + (2-\rho)U^{n}_{i,j} - U^{n}_{i,j-1} - h^2\xi_{i,j}\,, \tag{6}$$

$$U^{n+1}_{i,j+1} - (2+\rho)U^{n+1}_{i,j} + U^{n+1}_{i,j-1} = -U^{n+1/2}_{i+1,j} + (2-\rho)U^{n+1/2}_{i,j} - U^{n+1/2}_{i-1,j} - h^2\xi_{i,j}\,, \tag{7}$$

where $\rho = h^2/\Delta t$, and subscript commas indicate a separation of spatial indices rather than differentiation. The first pass loops over all j values, and for each j solves for U at all i-locations. The second equation does the opposite. Initial guesses of the solution along the domain must be provided, but the RHS is always known and solution proceeds in the standard manner for implicit central differences. Boundary conditions are incorporated in the standard fashion as well, by modifying the RHS and diagonal terms as needed. To determine convergence, we define our error norm for the n^{th} iteration as

$$\epsilon_n = \frac{1}{\epsilon_1}\sum_{i=1}^{N}\sum_{j=1}^{N}\left|U^{n}_{i,j} - U^{n-1}_{i,j}\right|\,, \quad n = 2,3,\ldots\,, \tag{8}$$

and cease solution when $\epsilon_n \leq 10^{-3}$. Here, ϵ_1 is defined as the RHS sums evaluated for $n = 1$, and the initial guess is defined as $U^0_{i,j} = 0$.

Peaceman and Rachford (1955) show that the ADI method converges for any value of the iteration parameter ρ. The optimal value is a function of the iteration number k:

$$\rho_k = 4\sin^2\frac{k\pi}{2N}\,, \qquad k = 1,2,\ldots,n \quad \text{(until convergence)}\,. \tag{9}$$

Note that since we change ρ on each iteration, the **L** and **U** matrices involved in LU-decomposition need to be re-assembled with each iteration, and we sacrifice some of the efficiency gains of LU-decomposition. The details of LU-decomposition will not be discussed here, as they were presented in full in Homework 4.

2.2 PROBLEM 2

For the SOR method, we perform "column sweeps" over the whole domain until convergence is achieved. That is, we loop over $i = 2,\ldots,N-1$ and apply the following SOR central difference equation to each grid point with $j = 2,\ldots,M-1$:

$$U^{n}_{i,j} = (1-\omega)\,U^{n-1}_{i,j} + \frac{\omega}{4}\left(U^{n}_{i-1,j} + U^{n}_{i,j-1} + U^{n-1}_{i+1,j} + U^{n-1}_{i,j+1} - h^2\xi_{i,j}\right)\,, \tag{10}$$

where ω is the relaxation parameter, which typically has an optimal value in the range $[1.7, 1.9]$.

The theoretical optimum value of ω is given by

$$\omega_{\text{opt}} = \frac{8 - 4\sqrt{4-\alpha^2}}{\alpha^2}, \qquad \alpha = \cos(\pi/M) + \cos(\pi/N), \tag{11}$$

and for our grid with $M = N = 101$, we obtain $\alpha = 1.9990$ and $\omega_{\text{opt}} = 1.9397$.

We define our convergence criterion exactly the same as in Problem 1, though we could have used the maximum norm, which is defined as

$$\epsilon_n = \frac{1}{\epsilon_1} \max_{i,j} \left| U^n_{i,j} - U^{n-1}_{i,j} \right|, \tag{12}$$

where ϵ_1 is defined as the RHS max operation evaluated for $n = 1$. The initial guess is defined as $U^0_{i,j} = 0$ except for when $j = N$, in which case the upper boundary condition is imposed. Do note that when we actually *solve* this BVP, the upper boundary condition needs to be $U = -1$, since in a right-handed coordinate system, positive X points into the page when looking at the Z-Y plane. In other words, the given $U = 1$ BC corresponds to the upper boundary moving into the page, which in the context of our solver that uses the Z-Y plane, is represented by $U = -1$.

3 Results

3.1 Problem 1

A contour plot of the solution for U obtained from the ADI method is presented in Figure 2.

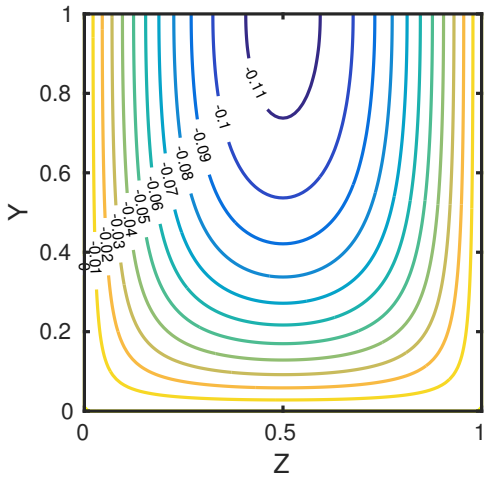

Figure 2: Contours of U for Problem 1.

3.2 Problem 2

Iterations to convergence as a function of ω, as well as a contour plot of the solution for U obtained from the SOR method is presented in Figure 3.

Experimentally, we find $\omega_{\text{opt}} = 1.9385 \pm 0.0015$, which is within error of our theoretical $\omega_{\text{opt}} = 1.9397$.

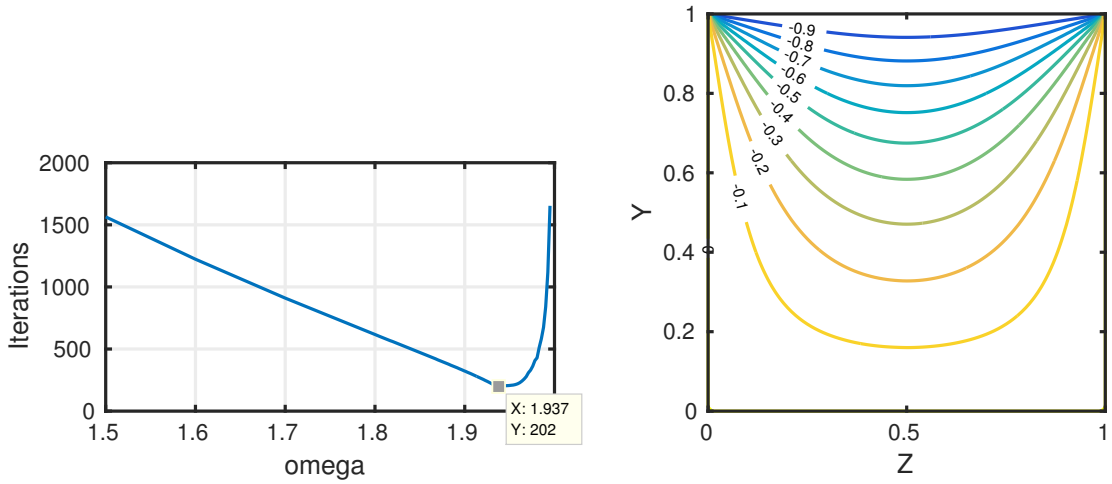

Figure 3: Convergence behavior and contours of U for Problem 2. Neighboring values of $\omega = 1.937$ and $\omega = 1.940$ both converge in 202 iterations.

4 Discussion

4.1 Problem 1

The ADI method converges very rapidly (within 5 iterations) using the variable time-step method based on σ_k. The contours obtained are reasonable for laminar flow in a duct of rectangular cross-section with an aspect ratio of 2. In Figure 2, we only see the lower half of this duct.

4.2 Problem 2

Substantial improvements are observed by using the SOR method compared to a simple Gauss-Seidel method, which the SOR method reduces to when $\omega = 1$. This value of ω is not shown in Figure 3 due to a high number of iterations (4142) to convergence.

5 References

No external references were used other than the course notes for this assignment.

Appendix: MATLAB Code

The following code listings generate all figures presented in this homework assignment. LU-decomposition code is omitted since it was already presented in Homework 4.

Listing 1: `Problem_1.m`

```
function [] = Problem_1()

    %%%%%%
    % Solves the inhomogeneous continuity equation for 2D fluid flow within a unit square
    % domain using the ADI method.
    %
    % Ryan Skinner, November 2015
    %%%
```

```
9
10      Set_Default_Plot_Properties();
11
12      %%%
13      % Define variables specific to the boundary-value problem.
14      %%%
15
16      % Solution domain: the closed interval [0,1]x[0,1]. Assume dx = dy = h.
17      N = 101;
18      x = linspace(0,1,N);
19      y = linspace(0,1,N);
20      h = x(2) - x(1);
21
22      % Source term.
23      xi = -1;
24
25      % Boundary conditions (u and uprime) defined as cardinal directions (n, s, w, e).
26      BC.us = 0;
27      BC.uw = 0;
28      BC.ue = 0;
29      BC.upn = 0; % This whole code assumes a value of zero.
30
31      % Initialize the solution, indexed by (x,y), and set BCs.
32      u = zeros(N,N);
33
34      % Iteraton parameter as a function of interation number.
35      rho = @(iter) 4 * sin(pi * iter / (2 * N))^2;
36
37      %%%
38      % Solve problem numerically.
39      %%%
40
41      % Solution norm.
42      epsilon = inf;
43      conv_crit = 0;
44
45      % Iteration number
46      n = 0;
47
48      % Iterate over time steps.
49      while epsilon > conv_crit
50          n = n + 1;
51
52          u_prev = u;
53          % Loop over j (horizontal slices).
54          for j = 2:N-1
55              [diag, sub, sup, rhs] = Assemble_ADI(u_prev(:,j-1:j+1), ...
56                                                   rho(n), h, xi, BC, 'horizontal');
57              [LUj.l, LUj.u] = LU_Decompose(diag, sub, sup);
58              [sol] = LU_Solve(sub, LUj.l, LUj.u, rhs);
59              u(:,j) = [BC.uw; sol; BC.ue];
60          end
61
62          u_half = u;
63          % Loop over i (vertical slices).
64          for i = 2:N-1
65              [diag, sub, sup, rhs] = Assemble_ADI(u_half(i-1:i+1,:)', ...
66                                                   rho(n), h, xi, BC, 'vertical');
67              [LUi.l, LUi.u] = LU_Decompose(diag, sub, sup);
68              [sol] = LU_Solve(sub, LUi.l, LUi.u, rhs);
69              u(i,:) = [BC.us; sol; sol(end)];
70          end
71
72          epsilon = sum(sum(abs(u_prev - u)));
73          fprintf('Iteration: %2i, Error Norm: %7.1e\n', n, epsilon);
74
75          if n == 1
76              conv_crit = 1e-3 * epsilon;
77          end
```

```
78
79      end
80
81      %%%
82      % Process results.
83      %%%
84
85      [C,h] = contour(x,y,u','LineWidth',2);
86      clabel(C,h,'FontSize',14,'LabelSpacing',1000);
87      axis('equal');
88      xlabel('Z');
89      ylabel('Y');
90
91      disp('Done.');
92      return
93
94  end
```

Listing 2: `Assemble_ADI.m`

```
1   function [diag, sub, sup, rhs] = Assemble_ADI( u_slice, rho, h, xi, BC, direction )
2
3       %%%%%%
4       % Assembles the LHS matrix and the RHS vector for the g-system.
5       %   diag -- diagonal
6       %    sub -- sub-diagonal
7       %    sup -- super-diagonal
8       %    rhs -- right-hand side vector
9       %
10      % Ryan Skinner, November 2015
11      %%%
12
13      N = max(size(u_slice));
14
15      diag_range = 2:N-1;
16       sub_range = 3:N-1;
17       sup_range = 2:N-2;
18
19      diag = - (2 + rho) * ones(length(diag_range),1);
20       sub =               ones(length(sub_range), 1);
21       sup =               ones(length(sup_range), 1);
22       rhs = -             u_slice(diag_range,3) ...
23             + (2 - rho) * u_slice(diag_range,2) ...
24             -             u_slice(diag_range,1) ...
25             - h^2 * xi;
26
27      % Account for boundary conditions.
28      if strcmp(direction,'vertical')
29          rhs(1)   = rhs(1) - BC.us;
30          diag(end) = - (1 + rho);
31          if BC.upn ~= 0
32              error('Non-zero Neumann BC not supported.');
33          end
34      elseif strcmp(direction,'horizontal')
35          rhs(1)   = rhs(1)   - BC.uw;
36          rhs(end) = rhs(end) - BC.ue;
37      else
38          error('Invalid direction specified.');
39      end
40
41  end
```

Listing 3: `Problem_2.m`

```
1   function [] = Problem_2()
2
3       %%%%%%
4       % Solves the inhomogeneous continuity equation for 2D fluid flow within a unit square
```

```
    % domain using the SOR method.
    %
    % Ryan Skinner, November 2015
    %%%

    Set_Default_Plot_Properties();

    %%%
    % Define variables specific to the boundary-value problem.
    %%%

    % Solution domain: the closed interval [0,1]x[0,1]. Assume dx = dy = h.
    N = 101;
    x = linspace(0,1,N);
    y = linspace(0,1,N);
    h = x(2) - x(1);

    % Source term.
    xi = 1;

    % Boundary conditions (u and uprime) defined as cardinal directions (n, s, w, e).
    BC.us = 0;
    BC.uw = 0;
    BC.ue = 0;
    BC.un = -1;

    % Relaxation parameters to test.
    omega = [linspace(1.5,1.8,4), linspace(1.805,1.995,80)];

    n_to_converge = nan(length(omega));

    %%%
    % Solve problem numerically.
    %%%

    for i_omega = 1:length(omega)
        om = omega(i_omega);
        fprintf('\nWorking on omega = %6.3f\n', om);

        % Initialize the solution, indexed by (x,y), and set BCs.
        u = zeros(N,N);
        u(:,1)   = BC.us;
        u(:,end) = BC.un;
        u(1,:)   = BC.uw;
        u(end,:) = BC.ue;

        % Error and convergence measures.
        epsilon = inf;
        conv_crit = 0;

        % Iteration number
        n = 0;

        % Iterate over time steps.
        while epsilon > conv_crit

            n = n + 1;
            u_prev = u;

            for i = 2:N-1
                for j = 2:N-1
                    u(i,j) = (1 - om) * u_prev(i,j) ...
                             + (om/4) * (        u(i-1, j) ...
                                         +      u(i,   j-1) ...
                                         + u_prev(i+1, j) ...
                                         + u_prev(i,   j+1) ...
                                         - h^2 * xi );
                end
            end
```

```
74
75              % Calculate error norm.
76              epsilon = sum(sum(abs(u_prev - u)));
77
78              % Set convergence criterion based on first iteration.
79              if n == 1
80                  conv_crit = 1e-3 * epsilon;
81              end
82
83              % Abort if diverging.
84              if 1e-3*epsilon/conv_crit > 1e5
85                  error('Solution diverged.');
86              end
87
88              if (mod(n,10)  == 0) fprintf('.'); end
89              if (mod(n,500) == 0) fprintf('\n'); end
90          end
91
92          % Store and print convergence information.
93          n_to_converge(i_omega) = n;
94          fprintf('\nIteration: %2i, Error Norm: %7.1e\n', n, 1e-3*epsilon/conv_crit);
95
96      end
97
98      %%%
99      % Process results.
100     %%%
101
102     figure();
103     plot(omega,n_to_converge);
104     xlabel('omega');
105     ylabel('Iterations');
106     xlim([1.5,2]);
107     ylim([0,2000]);
108
109     figure();
110     [C,h] = contour(x,y,u','LineWidth',2);
111     clabel(C,h,'FontSize',14,'LabelSpacing',1000);
112     axis('equal');
113     xlabel('Z');
114     ylabel('Y');
115
116     figure();
117     surf(x,y,u');
118     xlabel('Z');
119     ylabel('Y');
120
121     disp('Done.');
122     return
123
124 end
```