Indian Institute of Technology Madras

AM6515
Boundary Layer Stability
Term Project Report

Numerical Approach to Solving Orr-Sommerfeld Eqution for Blasius Profile and Plane Poiseulle Flow

Submitted by: Abel Viji George, AE21B001

Contents

List of Figures

1 Introduction

The Orr-Sommerfeld equation is a fundamental differential equation in fluid dynamics, specifically in the study of hydrodynamic instability and the transition from laminar to turbulent flow in a viscous fluid. It describes the stability of small disturbances in a parallel shear flow and is useful in examining how these disturbances evolve, which can lead to the amplification of instabilities and the onset of turbulence.
The equation is derived from the Navier-Stokes equations under assumptions for incompressible and linearized disturbances in a parallel flow. Named after William McFadden Orr and Arnold Sommerfeld, it governs the evolution of infinitesimal perturbations superimposed on a base flow, such as plane Couette or Poiseuille flow. By studying the solutions of the Orr-Sommerfeld equation, researchers gain insight into the stability characteristics of the base flow, understanding whether disturbances will decay or grow over time.

2 Derivation of the Equation

2.1 Base Flow

The base flow is a parallel flow in the x - direction with velocity, $\vec{u_B} = \overline{U}(y)\hat{e_x}$, constant density, $\rho_B = \rho_0$, constant pressure, $p_B = p_0$. Further assumptions are the absence of gravity and 3-D perturbations. Using the argument of Squire's theorem, it is sufficient to consider 2-D perturbations. While linearizing, every quantity is given a perturbation.

$$\vec{u} = \vec{u_B} + u'$$
$$p = p_B + p'$$

The flow is bounded by y_1 and y_2.

2.2 Characteristic scales

The governing equations are written in a non-dimensional form:

$$\text{Spatial scale}: L, \quad \text{Velocity scale}: U$$
$$\text{Pressure scale}: \rho U^2, \quad \text{Time scale}: \frac{L}{U}$$

2.3 Non-dimensional continuity equation

$$\vec{\nabla}.\vec{u} = 0$$

Linearizing,

$$\vec{\nabla}.\vec{u}' = 0$$

2.4 Non-dimensional momentum equation

$$\frac{D\vec{u}}{Dt} = -\vec{\nabla}p + \frac{\vec{\nabla}^2\vec{u}}{R_e}$$

Linearizing,

x-momentum equation

$$\frac{\partial u'}{\partial t} + \overline{U}\frac{\partial u'}{\partial x} + v'\frac{\partial \overline{U}}{\partial y} = -\frac{\partial p'}{\partial x} + \frac{1}{R_e}\left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}\right)u'$$

y-momentum equation

$$\frac{\partial v'}{\partial t} + \overline{U}\frac{\partial v'}{\partial y} = -\frac{\partial p'}{\partial y} + \frac{1}{R_e}\left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}\right)v'$$

2.5 In terms of stream function, ψ

$$u' = \psi_y v' = -\psi_x$$

Substituting for u' and v' and eliminating p' from the momentum equations, we get,

$$\left(\frac{\partial}{\partial t} + \overline{U}\frac{\partial}{\partial x}\right)\nabla^2 - \psi_x\frac{d^2\overline{U}}{dy^2} = \frac{1}{R_e}\nabla^2\left(\psi_{xx} + \psi_{yy}\right)$$

2.6 Normal mode form of ψ

Since the flow is homogeneous in x-direction and inhomogeneous in y-direction, we can write a normal mode solution form for ψ:

$$\psi = \frac{\hat{\psi(y)}}{2} e^{i(kx-\omega t)} + C.C$$

where, $\hat{\psi}$ is the amplitude, k is the wavenumber in the x-direction and ω is the complex frequency. c is the wave velocity with $c = \frac{\omega}{k}$. Substituting in the pressure-eliminated-momentum equations, we get,

$$(\overline{U} - c)\left(\frac{d^2}{dy^2} - k^2\right)\hat{\psi} - \hat{\psi}\frac{d^2\overline{U}}{dy^2} = \frac{1}{ikR_e}\left(\frac{d^2}{dy^2} - k^2\right)^2 \hat{\psi}$$

which is the Orr-Sommerfeld equation.

3 Numerical Techniques to solve Orr-Sommerfeld equation

3.1 Shooting Method

- Write the O.S equation as
$$\Phi' = A(y)\Phi$$
where, $\Phi = \begin{pmatrix}\hat{\psi} & \frac{d\hat{\psi}}{dy} & \frac{d^2\hat{\psi}}{dy^2} & \frac{d^3\hat{\psi}}{dy^3}\end{pmatrix}^T$, $\Phi, = \frac{d\Phi}{dy}$ and $A(y) = \begin{bmatrix}0 & 1 & 0 & 0\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\\ a & 0 & b & 0\end{bmatrix}$ where a and b can be found from the O.S equation.

- Since the O.S equation is a fourth order ODE, we can write the general solution as a combination of four linearly independent solutions:
$$\Phi(y) = \gamma_1\Phi^1(y) + \gamma_2\Phi^2(y) + \gamma_3\Phi^3(y) + \gamma_4\Phi^4(y)$$

- We set the boundary conditions for each of the four linearly independent solutions as follows: $\Phi^1(y_1) = \begin{pmatrix}0 & 0 & 0 & 1\end{pmatrix}^T$, $\Phi^2(y_1) = \begin{pmatrix}0 & 0 & 1 & 0\end{pmatrix}^T$, $\Phi^3(y_1) = \begin{pmatrix}0 & 1 & 0 & 0\end{pmatrix}^T$ and $\Phi^4(y_1) = \begin{pmatrix}1 & 0 & 0 & 0\end{pmatrix}^T$. Note that these will not satisfy the O.S equation! These just ensure that four linearly independent solutions are generated.

- The actual boundary condition for flow bounded by y_1 and y_2 is $\psi(\hat{y_1}) = 0$ and $\frac{d\hat{\psi}(y_1)}{dy} = 0$. So, the contribution of $\Phi^3(y)$ and $\Phi^4(y)$ should be 0. So, $\gamma_3 = 0$ and $\gamma_4 = 0$. So, general solution i of the form $\Phi(y) = \gamma_1\Phi^1(y) + \gamma_2\Phi^2(y)$.

- Taking an initial guess for c, we generate $\Phi_1(y)$ and $\Phi_2(y)$ at all locations using some integration scheme like the RK-4 method.

- Let $\Phi(y) = \begin{pmatrix}\hat{\psi} & \frac{d\hat{\psi}}{dy} & \frac{d^2\hat{psi}}{dy^2} & \frac{d^3\hat{\psi}}{dy^3}\end{pmatrix}^T = \begin{pmatrix}\Phi_1 & \Phi_2 & \Phi_3 & \Phi_4\end{pmatrix}^T$.

- At $y = y_2$, imposing the boundary conditions, $\psi(\hat{y_2}) = 0$ and $\frac{d\psi(\hat{y_2})}{dy} = 0$, we get,
$$\gamma_1\Phi_1^1(y_2) + \gamma_2\Phi_2^2(y_2) = 0$$
$$\gamma_1\Phi_2^1(y_2) + \gamma_2\Phi_2^2(y_2) = 0$$
Hence, for non-trivial solutions of γ_1 and γ_2, we can impose,
$$\det\begin{bmatrix}\Phi_1^1(y) & \Phi_1^2(y)\\ \Phi_2^1(y) & \Phi_2^2(y)\end{bmatrix}$$
$$= \Phi_1^1(y)\Phi_2^2(y) - \Phi_2^1(y)\Phi_1^2(y) = 0.$$

- Define $L = \Phi_1^1(y)\Phi_2^2(y) - \Phi_2^1(y)\Phi_1^2(y)$. The numerical problem has been reduced to finding a c for which L becomes zero. Having found such a c, we check if the imaginary part of it, c_i becomes positive for any k. In that case, there is an instability.

- Another way of tackling the problem is to use central differencing scheme to approximate the second and fourth derivatives of $\hat{\psi}$. This method is detailed in [2].
$$\bar{\phi}_i'' \approx \frac{\bar{\phi}_{i-1} - 2\bar{\phi}_i + \bar{\phi}_{i+1}}{h^2},$$
$$\bar{\phi}_i'''' \approx \frac{\bar{\phi}_{i-2} - 4\bar{\phi}_{i-1} + 6\bar{\phi}_i - 4\bar{\phi}_{i+1} + \bar{\phi}_{i+2}}{h^4}.$$

3.2 Spectral Methods

The Orr-Sommerfeld equation, after rearranging, reads:

$$\left(-Uk^2 - U'' - \frac{k^4}{ikRe}\right)\hat{\psi} + \left(U + \frac{2k^2}{ikRe}\right)D^2\hat{\psi} - \frac{1}{ikRe}D^4\hat{\psi} = c\left(D^2\hat{\psi} - k^2\hat{\psi}\right)$$

with the boundary conditions:

$$\hat{\psi}(\pm 1) = D\hat{\psi}(\pm 1) = 0$$

D is the derivative matrix.
Chebyshev Polynomials of the first kind are given by:

$$T_n(y) = cos(ncos^{-1}(y))$$

for $-1 \leq y \leq 1$.
We expand the eigenfunctions in a Chebyshev series:

$$\hat{\psi}(y) = \sum_{n=0}^{N} a_n T_n(y)$$

where $T_n(y)$ are Chebyshev polynomials of degree n.

The derivatives of the eigenfunctions are obtained by differentiating the expansion. For the second derivative, for example:

$$D^2\hat{\psi}(y) = \sum_{n=0}^{N} a_n T_n''(y),$$

and similarly for the fourth derivative. Substituting this expansion into the Orr-Sommerfeld equation, we get:

$$\left(U(y)k^2 - U''(y) - \frac{k^4}{ik\mathrm{Re}}\right)\sum_{n=0}^{N} a_n T_n(y) + \left(U(y) + \frac{2k^2}{ik\mathrm{Re}}\right)\sum_{n=0}^{N} a_n T_n''(y)$$
$$- \frac{1}{ik\mathrm{Re}}\sum_{n=0}^{N} a_n T_n''''(y) = c\left(\sum_{n=0}^{N} a_n T_n''(y) - k^2\sum_{n=0}^{N} a_n T_n(y)\right) \quad (1)$$

We require this equation to be satisfied at the Gauss-Lobatto collocation points $y_j = \cos\left(\frac{\pi j}{N}\right)$. This allows us to use the recurrence relations to evaluate the derivatives of the Chebyshev polynomials.

The discretized boundary conditions read:

$$\sum_{n=0}^{N} a_n T_n(1) = 0, \quad \sum_{n=0}^{N} a_n T_n'(1) = 0,$$
$$\sum_{n=0}^{N} a_n T_n(-1) = 0, \quad \sum_{n=0}^{N} a_n T_n'(-1) = 0.$$

The final result is a generalized eigenvalue problem of the form:

$$Aa = cBa$$

Where we solve for the eigen value, c which is the complex velocity.
For detailed explanation of the method refer [1].

4 Results and Discussions

A brief discussion is presented for two flow profiles - the Blasius profile and the Plane Poiseulle flow. The results obtained from the numerical code to solve the Orr-Sommerfeld equation using the shooting and spectral methods are given along with some observations.

4.1 Blasius Profile

A 2-D Blasius profile can be expressed as:

$$\overline{U} = f'(\eta), \quad \eta = y\sqrt{\frac{U_\infty}{2\nu x}},$$
$$f''' + ff'' = 0,$$
$$f(0) = 0, \quad f'(0) = 0, \quad f'(\infty) = 1.$$

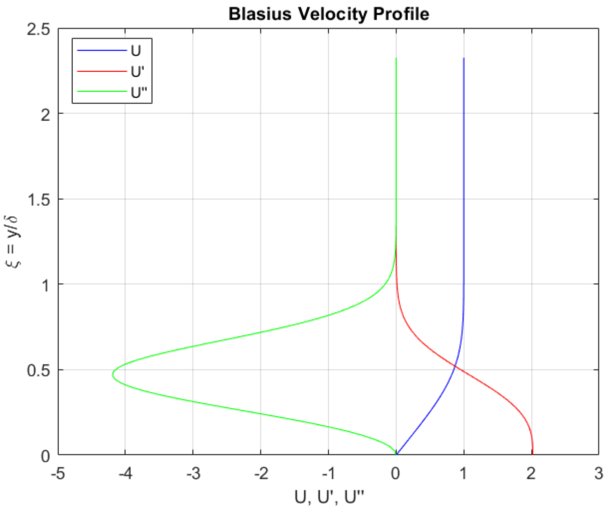
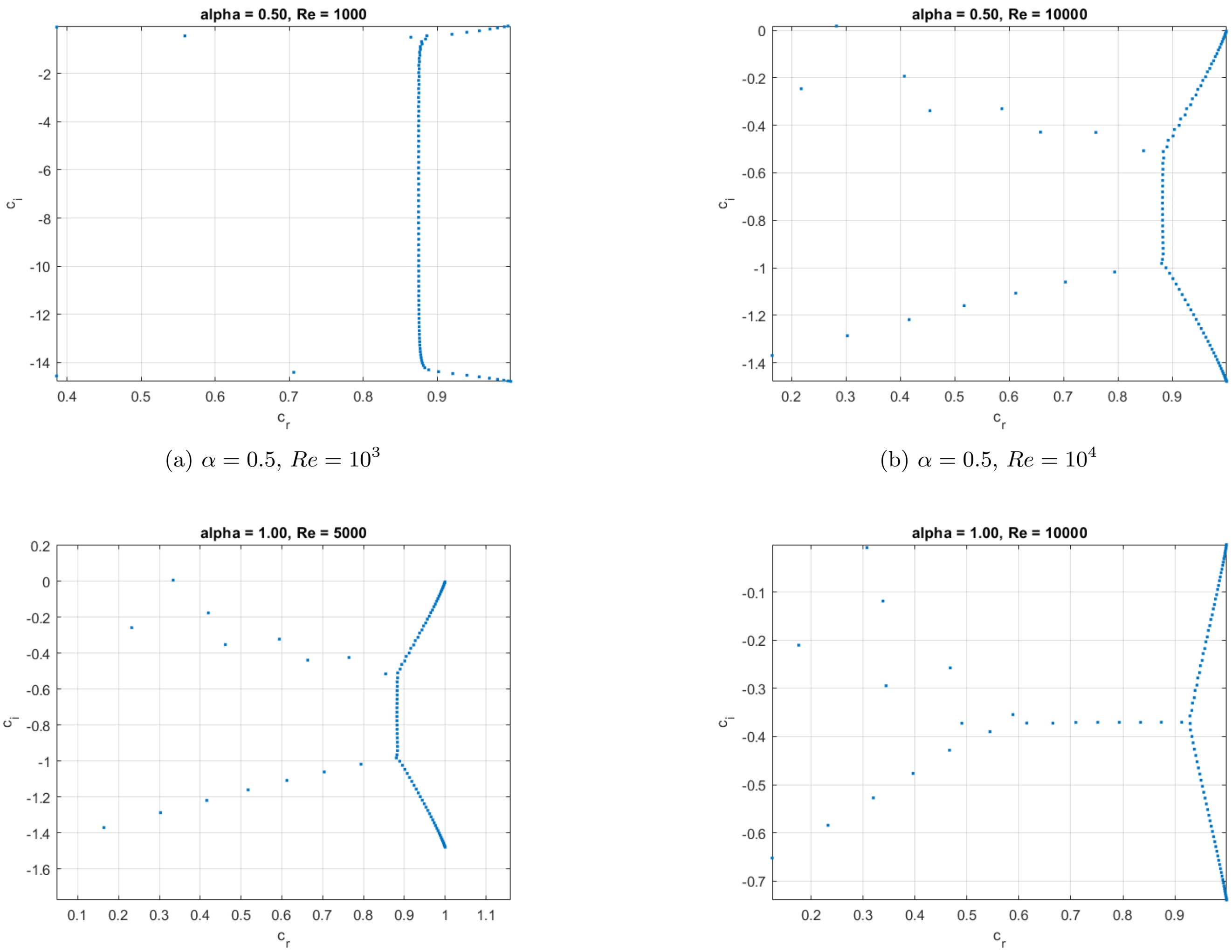


Figure 1: Blasius Profile

- Growth Rate (c_i) vs Wave Speed (c_r) plot.



(a) $\alpha = 0.5,\ Re = 10^3$

(b) $\alpha = 0.5,\ Re = 10^4$

(c) $\alpha = 1,\ Re = 5 \times 10^3$

(d) $\alpha = 1,\ Re = 10^4$

Figure 2: Plots using the Shooting Method

It can be seen that for the wavenumber, $\alpha = 0.5$, $Re = 10^3$ results in a stable system whereas $Re = 10^4$ has at least one c_i being positive.
For $\alpha = 1$, $Re = 5 \times 10^3$ yields unstable eigen values, whereas $Re = 10^4$ shows occurrence of negative imaginary parts of complex velocity.

- Neutral Stability Curves

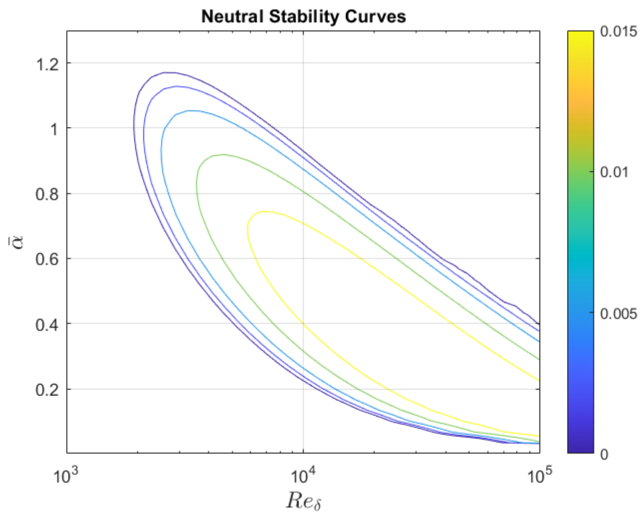


Figure 3: Curves representing $c_i = 0$ for $Re \in [10^3, 10^5]$ and $\alpha \in [0.001, 1.3]$

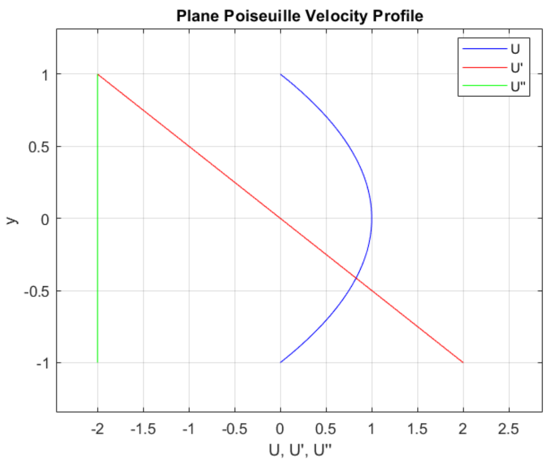
The critical Reynolds Number, $Re_c \approx 2030$.

4.2 Plane Poiseulle Flow

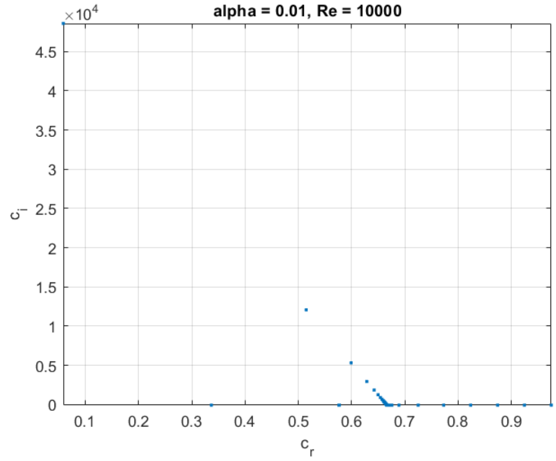
Plane Poiseulle flow is a type of parallel flow with a profile given by:

$$u(y) = U\left(1 - y^2\right)$$

where, $-1 \leq y \leq 1$ represents the veritical bounds and U is the flow velocity at the center ($y = 0$).



(a) Plane Poiseulle Flow profile



(b) Growth Rate vs Wave Speed plot using Shooting method

Figure 4

Solving the same way for Plane Poiseulle flow resulted in Non sensical results. This could be due to the failure of the shooting method to capture the strong gradient near the walls for the poiseulle flow. So, we move onto Spectral Methods

4.2.1 Spectral Method Solutions

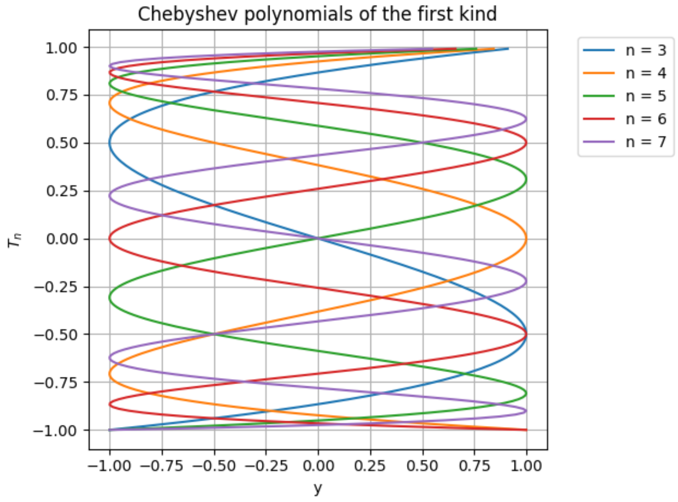
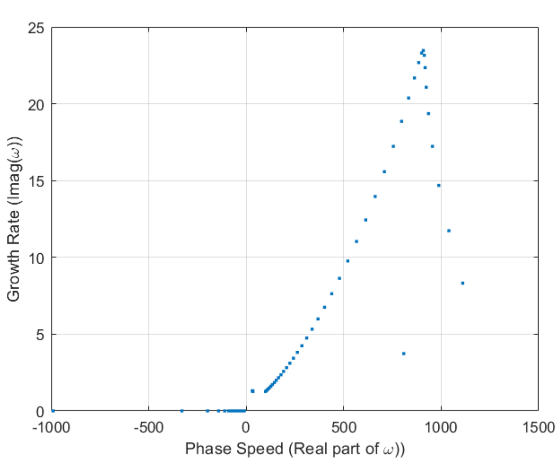
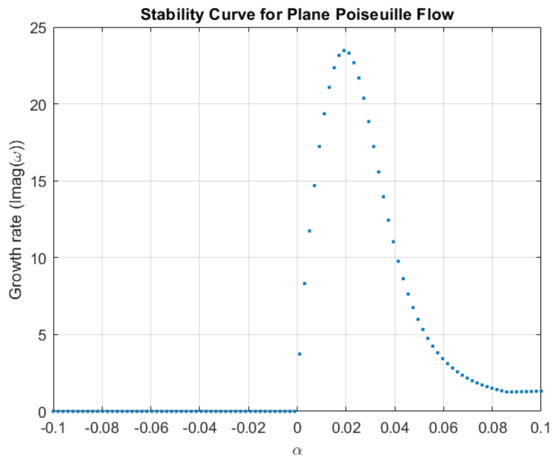


Figure 5: Chebyshev Polynomials of the first kind for n = 3 to 7

- Plots using Spectral Method



(a) Growth rate vs phase speed plot



(b) Growth rate vs wavenumbers

Figure 6: $\alpha \in [-0.1, 0.1]$, $Re = 10^4$

A range of wavenumbers are taken and the eigen values are computed for each of the wavenumber. The maximum growth rate among each eigen value is stored and plotted.
These plots can be used to study the most unstable wavenumber (wavenumber for which we have the maximum growth rate) and also the wavenumber at which even the slightest change can lead to instability. The plots suggest distinct peaks where there is the maximum instability.
It can be seen that there is a sudden growth just after $\alpha \approx 0$ and the most unstable wavenumber is $\alpha \approx 0.02$.

5 Conclusions

- A detailed derivation of the Orr-Sommerfeld equation is provided.
- Two numerical techniques are discussed for solving the Orr-Sommerfeld equations.
- The critical Reynolds Number, $Re_c \approx 2030$.
- For the Plane Poiseulle flow, it was seen that the numerical solutions yielded by the shooting method was very unstable and was not used,
- Using spectral method for the Plane Poiseulle flow, the most unstable wavenumber and the wavenumber above which instabilities can start to form can be identified.

6 Codes Used

6.1 Spectral Method to solve Orr-Sommerfeld equation for Plane Poiseulle Flow profile

```
% Plane Poiseuille flow Spectral Method
clear; clc; close all;

% Parameters
Re = 10000;           % Reynolds number
n = 100;              % Number of Chebyshev collocation points
alpha_range = linspace(-0.1, 0.1, 100); % Range of alpha (wavenumbers)
Re_range = linspace(2000, 10000, 20);  % Range of Reynolds numbers

% Chebyshev differentiation matrix
[D1, x] = chebyshev_diff_matrix(n);
D2 = D1^2;            % Second derivative
D2 = D2(2:end-1, 2:end-1); % Remove boundary rows for v'' BCs
I = eye(n-1);         % Identity matrix for inner points

% Base velocity profile for plane Poiseuille flow
U = diag(1 - x.^2);      % U(y) = 1 - y^2
U = U(2:end-1, 2:end-1); % Remove boundary rows/columns
U_dd = diag(-2 * ones(n-1, 1)); % U'' = -2 (constant for Poiseuille flow)

% Preallocate for storing results
growth_rates = zeros(1, length(alpha_range));
phase_speeds = zeros(1, length(alpha_range));
eigReal = [];
eigImag = [];
neutral_Re = [];

% Loop over wavenumbers
for j = 1:length(alpha_range)
    alpha = alpha_range(j);

    % Orr-Sommerfeld matrices
    L = -1i * alpha * Re * (D2 - alpha^2);  % Orr-Sommerfeld operator
    M = -1i * alpha * Re * (U * (D2 - alpha^2) - U_dd) + (D2 - alpha^2)^2; % RHS
    operator

    % Apply boundary conditions for v and v'
    A = [L, zeros(size(L));
         zeros(size(L)), I];
    B = [M, zeros(size(M));
         zeros(size(M)), I];

    % Solve generalized eigenvalue problem
    [eigVec, eigVal] = eig(A, B);
    eigVals = diag(eigVal);

    % Extract and store real and imaginary parts of eigenvalues
    eigReal = [eigReal; real(eigVals)];
    eigImag = [eigImag; imag(eigVals)];


    % Extract growth rate (imaginary part of eigenvalues)
    [maxGrowth, maxIdx] = max(imag(eigVals));
    growth_rates(j) = maxGrowth;
    phase_speeds(j) = real(eigVals(maxIdx)) / alpha;
end

% Plot results
figure;
plot(alpha_range, growth_rates, '.', 'LineWidth', 1.5);
```

```
xlabel('\alpha'); ylabel('Growth rate (Imag(\omega))');
title('Stability Curve for Plane Poiseuille Flow');
grid on;

figure;
plot(alpha_range, phase_speeds, '.', 'LineWidth', 1.5);
xlabel('\alpha'); ylabel('Phase speed (Real(\omega))');
grid on;

figure;
plot(phase_speeds, growth_rates, '.', 'LineWidth', 1.5);
xlabel('Phase Speed (Real part of \omega)'); ylabel('Growth Rate (Imag(\omega))');
grid on;

% Supporting function: Chebyshev differentiation matrix
function [D, x] = chebyshev_diff_matrix(n)
    % Returns the Chebyshev differentiation matrix D and grid points x
    x = cos(pi * (0:n) / n)'; % Chebyshev points
    c = [2; ones(n-1, 1); 2] .* (-1).^(0:n)'; % Scaling factors
    X = repmat(x, 1, n+1); % Repeated x-grid
    dX = X - X'; % Difference matrix
    D = (c * (1 ./ c)') ./ (dX + eye(n+1)); % Differentiation matrix
    D = D - diag(sum(D')); % Set diagonal elements
end
```

Listing 1: Spectral Method for Plane Poiseulle Flow

6.2 RK-4 scheme used for integration

```
% RK-4 solver
function [ t, w ] = cs_rk4( fun, t0, tf, y0, h )
t = [t0:h:tf];
w = zeros(length(y0),length(t)); % initialize w array
w(:,1) = y0;
for i = 1:(length(t)-1);
    ti = t(i);
    wi = w(:,i);
    k1 = h*fun(ti,wi);
    k2 = h*fun(ti+h/2,wi+k1/2);
    k3 = h*fun(ti+h/2,wi+k2/2);
    k4 = h*fun(ti+h,wi+k3);
    w(:,i+1) = wi + 1/6*(k1+2*k2+2*k3+k4);
end
end
```

Listing 2: RK-4 Scheme

6.3 Shooting Method for solving the Orr-Sommerfeld equation for the Blasius profile

```
clear all;
close all;
clc;

%% INPUTS
h = 0.1;    % eta step size - use this to control accuracy of calculations

% Velocity Profile Inputs
xi = linspace(0,1,50);
use_Blasius = 1; % use Blasius velocity profile below
transform_u = 1; % transform velocity to xi coordinates

% Temporal Stability Inputs
run_temporal_analysis = 1 ; % 1 = run temporal analysis, 0 = skip
t_Re_min = 1e3; % minimum Re
```

```
t_Re_max = 1e5; % maximum Re
t_N_Re = 40;    % number of Re
t_Res = logspace(log10(t_Re_min), log10(t_Re_max), t_N_Re);

t_alpha_min = 0.001;    % minimum alpha for temporal analysis
t_alpha_max = 1.3;      % maximum alpha
t_N_alpha = 40;         % number of alpha to use
t_alphas = linspace(t_alpha_min,t_alpha_max,t_N_alpha);

%%
FIG_VEL_PROFILE = 1;
FIG_TEM_EIG = 2;
FIG_TEM_CONT = 3;

%% COMPUTE BLASIUS VELOCITY PROFILE
if ( use_Blasius )
    fprintf('Solving for Blasius Velocity Profile ... ');
    tic; % start counting elapsed time

    % Blasius ODE Definition:  f''' + ff'' = 0  ->  f''' = -ff''
    fun = @(eta,f) [ ...
        f(2); ...
        f(3); ...
        -f(1)*f(3); ...
        ];

    eta0 = 0;   % Initial eta station (@ wall )
    etaf = 10;  % Final eta station (@ "freestream")
    target = 1; % Freestream BC: f'(inf) = U(inf) = 1
    tol = 1e-6; % Shooting method tolerance

    f0 = [ 0; 0; .45 ]; % Initial condition for f, f', f'' at wall

    go = 1;
    while ( go == 1 ) % loop until converged
        % Standard 4th-order Runge-Kutta method
        [ eta, f ] = cs_rk4( fun, eta0, etaf, f0, h );

        % compute error, adjust f''(0)
        err = f(2,end) - target;
        if ( abs(err) < tol )
            go = 0;
        else
            f0(3,1) = f0(3,1) * ( target-err );
        end
    end
    U = f(2,:);              % U   = f'(eta)
    Up = f(3,:);             % U'  = f''(eta)
    Upp = -f(1,:).*f(3,:);   % U'' = f'''(eta)
    elapsed_time = toc;      % compute time elapsed
    fprintf('Done (%f sec)\n',elapsed_time);
end

%% TRANSFORM ETA TO XI
if ( transform_u )
    fprintf('Transforming Coordinates ... ');
    tic; % start counting elapsed time

    % find delta in terms of eta
    minval = 1; % initialize minimization variable
    index = 0;  % index of the edge of the Boundary Layer
    delta_vel = 0.999;  % percent of freestream velocity

    % find edge of Boundary layer, where U == delta_vel
```

```
    for n = 1:length(U)
        if ( abs(U(n)-delta_vel) < minval )
            index = n;
            minval = abs(U(n)-delta_vel);
        end
    end

    delta = eta(index); % Boundary layer thickness
    xi = eta/delta;     % define xi s.t. xi = 1 where eta = delta

    % transform to xi coordinate system
    % U = U;            % U(xi)   = U(eta)
    Up = Up*delta;      % U'(xi)  = U'(eta)*delta
    Upp = Upp*delta^2;  % U''(xi) = U''(eta)*delta^2

    elapsed_time = toc; % compute elapsed time
    fprintf('Done (%f sec)\n',elapsed_time);
end
NMAX = length(xi);  % look at all eigenvalues
% NMAX = index;       % only look at eigenvalues in B.L.

%% TEMPORAL STABILITY ANALYSIS
if ( run_temporal_analysis )
    fprintf('Temporal Stability Analysis ...\n');
    tic; % start counting elapsed time

    h = xi(2)-xi(1);    % xi step size
    for l = 1:length(t_Res)
        Re = t_Res(l);
        for m = 1:length(t_alphas)
            alpha = t_alphas(m);
            fprintf('\ta = %.2f, Re = %d ... ',alpha,Re);

            % generate the A matrix
            A = zeros(NMAX); % initialize A

            n = 2; % start at n = 2, go to n = nmax-1

            a1 = -U(n)*alpha^3 - Upp(n)*alpha + 1i*alpha^4/Re;
            a2 = U(n)*alpha - 2*1i*alpha^2/Re;
            a3 = 1i/Re;

            % 2nd order Central Difference Method
            A(n,n-1) = a2 - 4*a3/h^2;
            A(n,n)   = a1*h^2 - 2*a2 + 6*a3/h^2;
            A(n,n+1) = a2 - 4*a3/h^2;
            A(n,n+2) = a3/h^2;

            for n = 3:NMAX-2

                a1 = -U(n)*alpha^3 - Upp(n)*alpha + 1i*alpha^4/Re;
                a2 = U(n)*alpha - 2*1i*alpha^2/Re;
                a3 = 1i/Re;
                % 2nd order Central Difference Method
                A(n,n-2) = a3/h^2;
                A(n,n-1) = a2 - 4*a3/h^2;
                A(n,n)   = a1*h^2 - 2*a2 + 6*a3/h^2;
                A(n,n+1) = a2 - 4*a3/h^2;
                A(n,n+2) = a3/h^2;
            end

            n = NMAX-1;
            a1 = -U(n)*alpha^3 - Upp(n)*alpha + 1i*alpha^4/Re;
            a2 = U(n)*alpha - 2*1i*alpha^2/Re;
```

```
            a3 = 1i/Re;
            % 2nd order Central Difference Method
            A(n,n-2) = a3/h^2;
            A(n,n-1) = a2 - 4*a3/h^2;
            A(n,n)   = a1*h^2 - 2*a2 + 6*a3/h^2;
            A(n,n+1) = a2 - 4*a3/h^2;

            A = A(2:end-1,2:end-1); % remove first/last rows/cols of A (0 BCs)

            % generate B
            B = zeros(NMAX);    % initialize B
            for n = 2:NMAX-1

                b1 = -alpha^3;
                b2 = alpha;
                % 2nd order Central Difference Method
                B(n,n-1) = b2;
                B(n,n)   = b1*h^2 - 2*b2;
                B(n,n+1) = b2;
            end
            B = B(2:end-1,2:end-1); % remove first/last rows/cols of B (0 BCs)


            [V,e] = eig(B\A);   % invert B, compute eigenvalues of LHS
            e = diag(e);    % turn diagonal eigenvalue matrix into vector

            % store most unstable eigenvalue
            [maxe,cindex] = max(imag(e));
            fprintf('c = %f + %fi\n',real(e(cindex)),imag(e(cindex)));
            t_c(m,l) = e(cindex);

            % Plot eigenvalues of the discrete system
            figure(FIG_TEM_EIG)
            plot(real(e),imag(e),'.','MarkerSize',6)
            xlabel('c_r')
            ylabel('c_i')
            title(sprintf('alpha = %.2f, Re = %d',alpha,Re));
            axis tight

        end
    end
    elapsed_time = toc; % compute elapsed time
    %fprintf('Done (%f sec)\n',elapsed_time);
end

%% OUTPUT RESULTS
if ( run_temporal_analysis )
    cilevels = [0,.005,.01,.015,.0019];
    figure(FIG_TEM_CONT);
    contour(t_Res,t_alphas,imag(t_c),cilevels);
    set(gca,'XScale','log');
    xlabel('$Re_\delta$','Interpreter','latex','Fontsize',14);
    ylabel('$\bar{\alpha}$','Interpreter', 'latex','Fontsize',14);
    title('Neutral Stability Curves')
    colorbar
    grid on;
end

% uprop = ['k- ';'k: ';'k--'];
uprop = ['b';'r';'g'];
figure(FIG_VEL_PROFILE);
plot(U,xi,uprop(1,:),Up,xi,uprop(2,:),Upp,xi,uprop(3,:))
title('Blasius Velocity Profile')
xlabel('U, U'', U''''')
```

```
ylabel('\xi = y/\delta')%,'Interpreter','latex','FontSize',16)
legend('U','U''','U''''','location','best')
grid on;
```

Listing 3: Shooting Method for Blasius Profile

6.4 Plotting the Chebyshev Polynomials used for spectral Method

```
# Plotting the Chebyshev Polynomials
import numpy as np
import matplotlib.pyplot as plt

h = 0.01
y = np.arange(-1, 1 , h)
N = 5

def T_n(y, n):
    return np.cos(n * np.arccos(y))

for i in range(N):
    plt.plot(T_n(y, i), y, label = f'n = {i}')
    plt.xlabel('y')
    plt.ylabel('$T_n$')
    plt.title('Chebyshev polynomials of the first kind')
    plt.legend(loc='upper left', bbox_to_anchor=(1.05, 1))
    plt.grid(True)
    plt.tight_layout()
plt.show()
```

Listing 4: Plotting the Chebyshev Polynomials

The reader is directed to [3] for the full code on the spectral method for two phase parallel flow and [2] for the full code on shooting method for blasius profile which formed as a solid platform from which the provided codes are built.

References

[1] Peter J. Schmid and Dan S. Henningson. *Stability and Transition in Shear Flows*. Springer, 2001.

[2] C. Simpson. Temporal and spatial stability analysis of the orr-sommerfeld equation. https://www.cdsimpson.net/2015/04/temporal-and-spatial-stability-analysis.html#:~:text=This%20is%20a%20nonlinear%20ordinary,opposite%20boundary%20conditions%20are%20met, 2015.

[3] A. Singh. Chebyshev collocation code for solving two phase orr-sommerfeld eigenvalue problem. https://in.mathworks.com/matlabcentral/fileexchange/48862-chebyshev-collocation-code-for-solving-two-phase-orr-sommerfeld-eigenvalue-problem, 2014.