The Orr-Sommerfeld Equation

A Numerical Approach to Fluid Stability

Seyed Mohammad Amin Hosseini

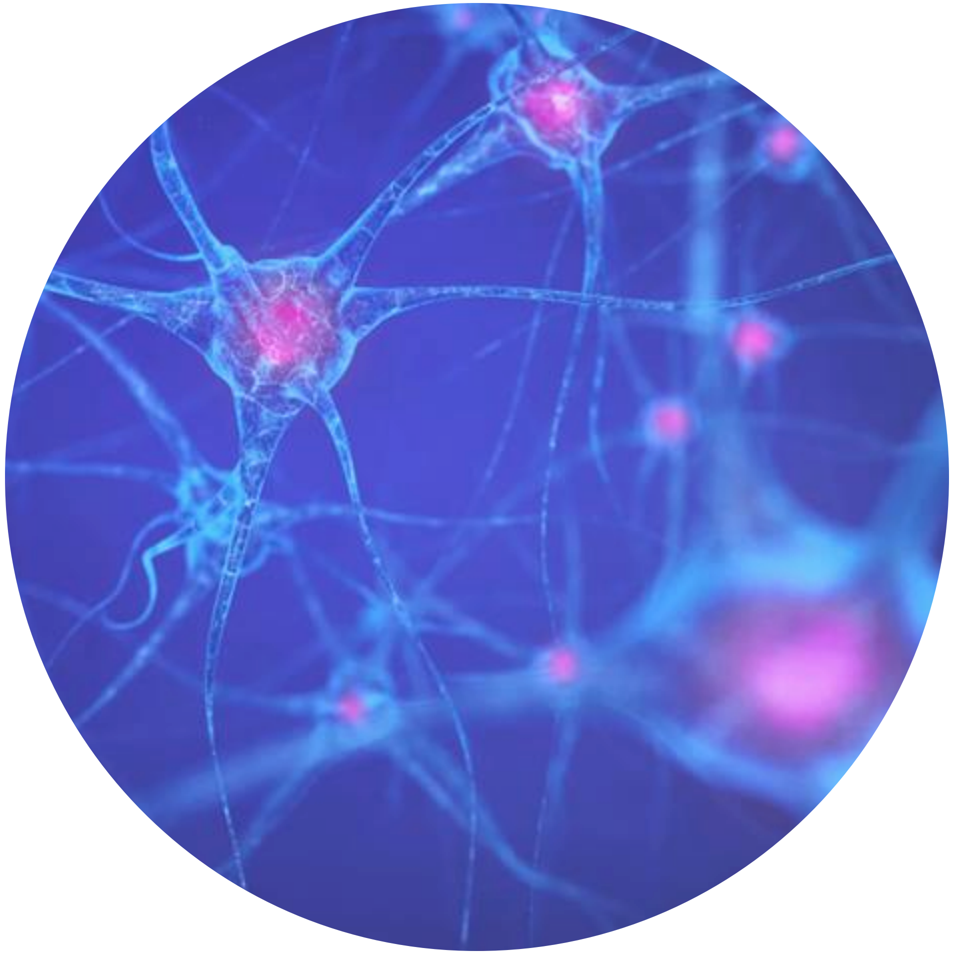
Prof.S.Salman Nourazar

Hydrodynamics Instability

Agenda

- Introduction
- Problem Statement
- Numerical Approach
- Results
- Future works

Introduction



Introduction

- The Orr-Sommerfeld equation describes the linear stability of parallel shear flows, such as **Poiseuille** and **Couette flow**.
- This equation helps determine whether small perturbations in the flow will decay (**stable**) or grow (**unstable**), leading to turbulence.
- Understanding flow stability is essential in engineering applications, including aerodynamics, pipeline flow, and industrial fluid transport.
- The objective of this study is to solve the Orr-Sommerfeld equation numerically using the **Finite Difference Method** (FDM) and analyze stability characteristics at different Reynolds numbers.

Problem Statement

Mathematical Formulation

- The Orr-Sommerfeld equation is derived from the Navier-Stokes equations for incompressible fluid flow with small perturbations.
- The equation is given by:

$$\frac{1}{i\kappa Re}\left[\frac{d^2}{dy^2}-\kappa^2\right]^2 v=(U-c)\left(\frac{d^2}{dy^2}-\kappa^2\right)v-U''v$$

And the B.C. of :

$$v=v'=0\ @\ y=-1,1$$

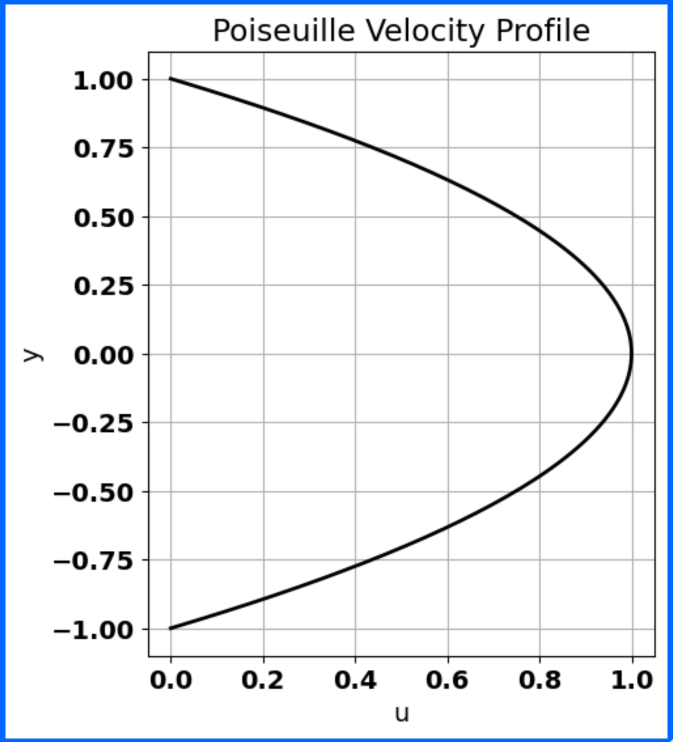
$c=c_r+ic_i$ is the complete wave speed

Problem Statement

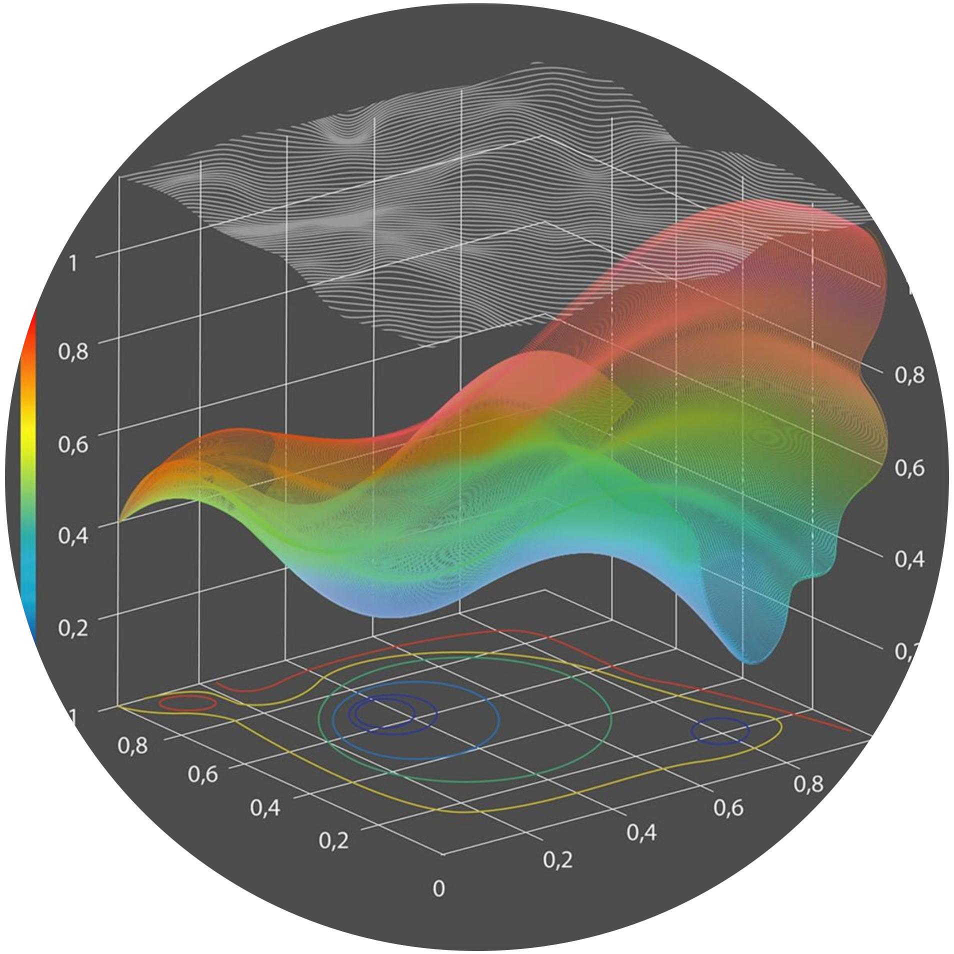
- The imaginary part of determines stability:
  - If $c_i < 0$ disturbances decay (stable flow).
  - If $c_i > 0$ disturbances grow (unstable flow leading to transition to turbulence).
  - In this research we worked on **Poiseuille Flow Profile** in which the velocity profile :

$$U = 1 - y^2$$

    - Grid discretization from $y = -1$ to $y = 1$



Numerical Approach



Code Structure

This function generates the parabolic velocity profile $U = 1 - y^2$ and its second derivative $U'' = -2$

Channel domain: y = -1 to 1

Grid spacing

Parabolic velocity profile

Second derivative

```
def get_poiseuille_profile(N):  1 usage

    y = np.linspace(-1,  stop: 1, N)
    dy = y[1] - y[0]
    u = 1 - y**2
    ddu = -2 * np.ones_like(y)
    return u, ddu, y, dy
```

N is the Number of grid points which is 201

Code Structure

This function is the core of your implementation, as it numerically solves the Orr-Sommerfeld equation using the **Finite Difference Method (FDM).**

The grid has N points, but the first two and last two points are excluded to enforce no-slip ($v = 0$) and ($v' = 0$) boundary conditions at $y = \pm 1$. This leaves $N1 = N - 4$ interior points.

```
def solve_orr_sommerfeld(u, ddu, dy, Re, K, N):  2 usages

    N1 = N - 4
    E = np.zeros(N1, dtype=complex)
    C = np.zeros(N1, dtype=complex)
    A = np.zeros(N1, dtype=complex)
    B = np.zeros(N1, dtype=complex)
    D = np.zeros(N1, dtype=complex)

    for i in range(N1):
        j = i + 2
        E[i] = 1j / (Re * K * dy**4)
        C[i] = 1j / (Re * K) * (-4 / dy**4 - 2 * K**2 / dy**2) + u[j] / dy**2
        A[i] = (1j / (Re * K) * (6 / dy**4 + 4 * K**2 / dy**2 + K**4) -
                2 * u[j] / dy**2 - u[j] * K**2 - ddu[j])
        B[i] = 1j / (Re * K) * (-4 / dy**4 - 2 * K**2 / dy**2) + u[j] / dy**2
        D[i] = 1j / (Re * K) / dy**4

    AMT = (np.diag(A[::-1], k=0) +
           np.diag(C[::-1][:-1], k=1) +
           np.diag(E[::-1][:-2], k=2) +
           np.diag(B[::-1][1:], k=-1) +
           np.diag(D[::-1][2:], k=-2))
    BCAP = np.ones(N1) / dy**2
    ACAP = (-2 / dy**2 - K**2) * np.ones(N1)
    CCAP = np.ones(N1) / dy**2
    BMT = (np.diag(ACAP[::-1], k=0) +
           np.diag(CCAP[::-1][:-1], k=1) +
           np.diag(BCAP[::-1][1:], k=-1))
    c, v = eig(AMT, BMT)
    return c, v
```

Code Structure

```
def solve_orr_sommerfeld(u, ddu, dy, Re, K, N):  2 usages

    N1 = N - 4
    E = np.zeros(N1, dtype=complex)
    C = np.zeros(N1, dtype=complex)
    A = np.zeros(N1, dtype=complex)
    B = np.zeros(N1, dtype=complex)
    D = np.zeros(N1, dtype=complex)

    for i in range(N1):
        j = i + 2
        E[i] = 1j / (Re * K * dy**4)
        C[i] = 1j / (Re * K) * (-4 / dy**4 - 2 * K**2 / dy**2) + u[j] / dy**2
        A[i] = (1j / (Re * K) * (6 / dy**4 + 4 * K**2 / dy**2 + K**4) -
                2 * u[j] / dy**2 - u[j] * K**2 - ddu[j])
        B[i] = 1j / (Re * K) * (-4 / dy**4 - 2 * K**2 / dy**2) + u[j] / dy**2
        D[i] = 1j / (Re * K) / dy**4

    AMT = (np.diag(A[::-1], k=0) +
           np.diag(C[::-1][:-1], k=1) +
           np.diag(E[::-1][:-2], k=2) +
           np.diag(B[::-1][1:], k=-1) +
           np.diag(D[::-1][2:], k=-2))
    BCAP = np.ones(N1) / dy**2
    ACAP = (-2 / dy**2 - K**2) * np.ones(N1)
    CCAP = np.ones(N1) / dy**2
    BMT = (np.diag(ACAP[::-1], k=0) +
           np.diag(CCAP[::-1][:-1], k=1) +
           np.diag(BCAP[::-1][1:], k=-1))
    c, v = eig(AMT, BMT)
    return c, v
```

These arrays will hold the finite difference coefficients

Each term represents a discretization of the Orr-Sommerfeld equation

Fourth Derivative Term:

$$v^{(4)} \approx \frac{v_{i-2} - 4v_{i-1} + 6v_i - 4v_{i+1} + v_{i+2}}{\Delta y^4}$$

second Derivative Term:

$$v'' \approx \frac{v_{i-1} - 2v_i + v_{i+1}}{\Delta y^2}$$

This constructs a pentadiagonal matrix

Code Structure

```
AMT = (np.diag(A[::-1], k=0) +
       np.diag(C[::-1][:-1], k=1) +
       np.diag(E[::-1][:-2], k=2) +
       np.diag(B[::-1][1:], k=-1) +
       np.diag(D[::-1][2:], k=-2))
BCAP = np.ones(N1) / dy**2
ACAP = (-2 / dy**2 - K**2) * np.ones(N1)
CCAP = np.ones(N1) / dy**2
BMT = (np.diag(ACAP[::-1], k=0) +
       np.diag(CCAP[::-1][:-1], k=1) +
       np.diag(BCAP[::-1][1:], k=-1))
c, v = eig(AMT, BMT)
return c, v
```

The pentadiagonal matrix **AMT** represents the left-hand side of the Orr-Sommerfeld equation:

$$AMT\,.v = c\,.BMT\,.v$$

The tridiagonal matrix **BMT** represents the right-hand side of the Orr-Sommerfeld equation.

The eigenvalues and eigenvectors are computed

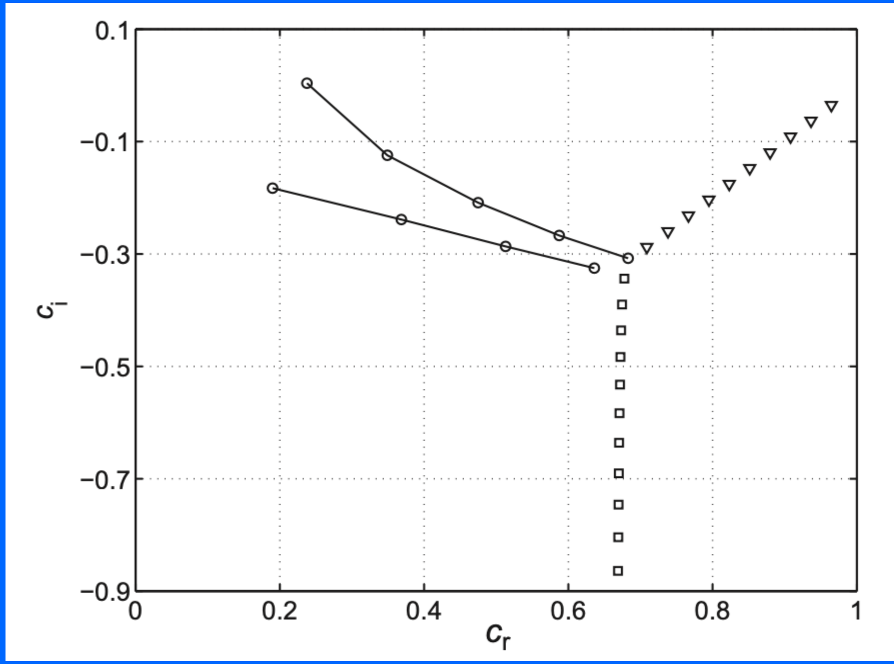
c is Complex eigenvalues
v is Eigenvectors

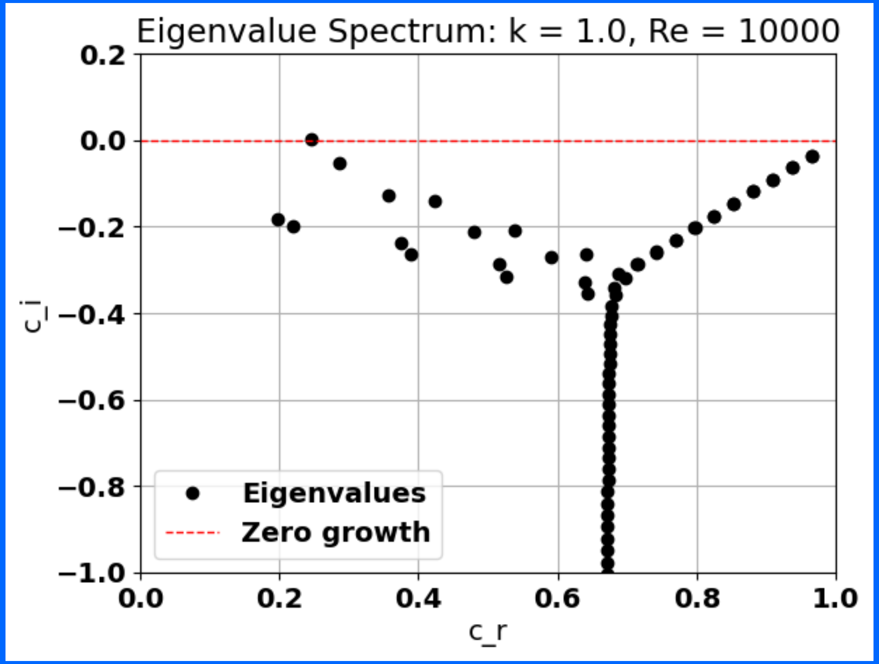
Results


analysis
something
breaking
definition
synthesis
material
learn
opposed
examination
parts

Eigenvalue Spectrum

These plots are Eigenvalue Spectrum for $\kappa = 1.0\ ,\ Re = 10000$

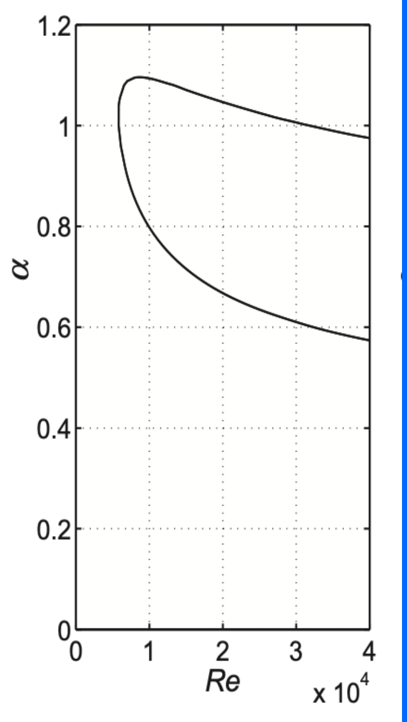


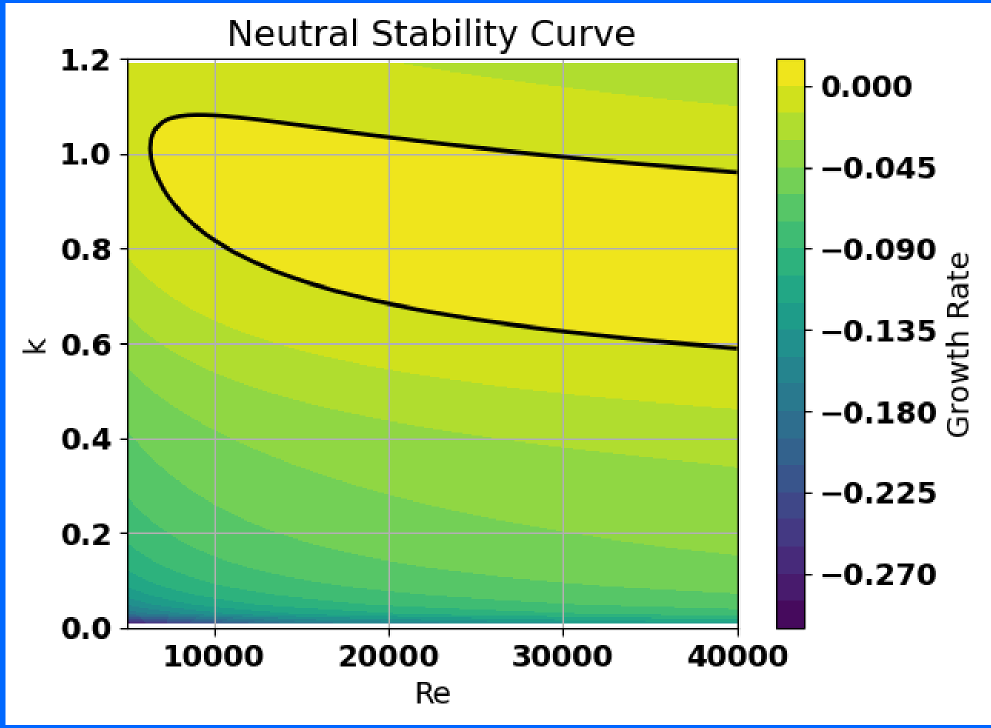


[1] Theory and Computation of Hydrodynamic Stability - W.O. criminale

Results

The black contour line represents the neutral stability curve, where disturbances neither grow nor decay ($c_i = 0$). Inside the enclosed region, $c_i > 0$, meaning the flow is unstable, while outside it, the flow remains stable.





Future Work

Future Work

1. **Machine Learning Approach:**

   ✓ Replace FDM with **Physics-Informed Neural Networks (PINNs)** for solving the Orr-Sommerfeld equation.

2. **Extended Flow Configurations:**

   ✓ Analyze stability in **Blasius Boundary Layer Flow** $2f''' + ff'' = 0$

   ✓ Investigate on **Falkner–Skan Flow** $2f''' + ff'' + \beta(1 - f'^2) = 0$

Thank you

Seyed Mohammad Amin Hosseini

09934995672

Amin.hosseini1@aut.ac.ir