Hydrodynamics and Cavitation Numerical Assignment

Pin-Yuan Tseng[1,*]

[1]Energy Science and Technology MSc., ETH Zurich
[*]tsengy@ethz.ch

ABSTRACT

This report is part of the numerical assignment for the **Hydrodynamics and Cavitation (151-0125-00L)** course. The assignment consists of three distinct problems, each addressing a classical hydrodynamics instability or nonlinear flow structure: the **capillary bridge**, the **distortion of acoustic waves**, and the **Rayleigh–Plateau instability**.
The first part investigates the capillary bridge formed between two circular loops. The numerical solution is used to analyze the stability of the liquid bridge and the conditions under which a physically meaningful solution exists. Key relationships are explored, including how the contact angle and neck radius vary with loop separation, and how a critical distance determines the limit of stable bridge formation.
The second part investigates the formation of shock waves in high-amplitude one-dimensional acoustic wave propagation. The governing dynamics are initially described by the Westervelt equation, which is subsequently simplified to the nonlinear Burgers' equation for numerical analysis. Three key plots are presented across the sub-problems: the temporal evolution of the pressure profile, the time evolution of pressure amplitude and its spatial gradient, and the shock formation distance as a function of the initial wave amplitude and frequency.
The third part focuses on the Rayleigh–Plateau instability, which explains the breakup of a liquid jet into droplets due to surface tension. The linear stability analysis is used to identify the most unstable wavenumber. Numerical plots are generated to visualize the radial deformation of the jet and the pressure perturbation field within the liquid column at a given time.
Throughout the report, all problems are solved using Python and visualized to provide physical insight into the flow behaviors.

1 Answers and Discussion

1.1 Capillary Bridge

The capillary bridge describes the shape where surface tension balances the pressure difference across a liquid interface connecting two solid boundaries. As shown in figure 1, It typically arises in axisymmetric configurations where a liquid spans between two surfaces (e.g., two circular loops or plates), forming a bridge-like meniscus. The shape of the interface is governed by the Young–Laplace equation, and depends on geometric parameters such as the height H, contact radius R, and the boundary contact angle β. The bridge exists only within certain geometric limits, beyond which surface tension can no longer support the structure, leading to instability or rupture.

In this problem, all geometric quantities such as the height H, contact radius $\bar{R}$, radial profile $r(z)$, and surface area S are presented in a dimensionless form. This is because the governing equations have been non-dimensionalized with respect to a characteristic length scale, typically chosen to be the contact radius $\bar{R}$. As a result, all lengths are expressed relative to $\bar{R}$, and the results describe the relative evolution of the capillary bridge shape rather than its absolute dimensions. This allows for a general analysis that can be rescaled to any physical system of interest.

1.1.1 Shape Profiles for K_1 and K_2

To solve the transcendental equation 1, we numerically found two positive roots for a given value of $\bar{R} = 1$ and $H = 1$. The smaller root K_1 and the larger root K_2 both yield mathematically valid shapes. With K_1 and K_2, two different surface areas can be calculated using equation 2 and 3.

$$\bar{R} = K \cosh\left(\frac{H}{2K}\right) \tag{1}$$

$$r(z) = K \cosh\left(\frac{z}{K}\right) \tag{2}$$

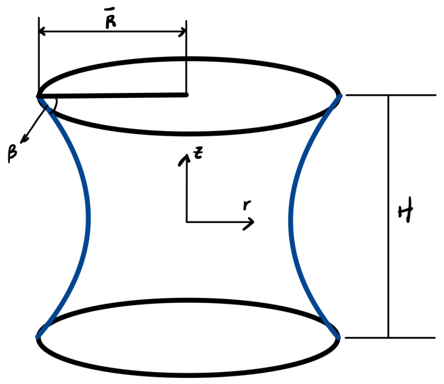

Figure 1. Structure of Capillary Bridge with height H, contact radius R, and the boundary contact angle β.

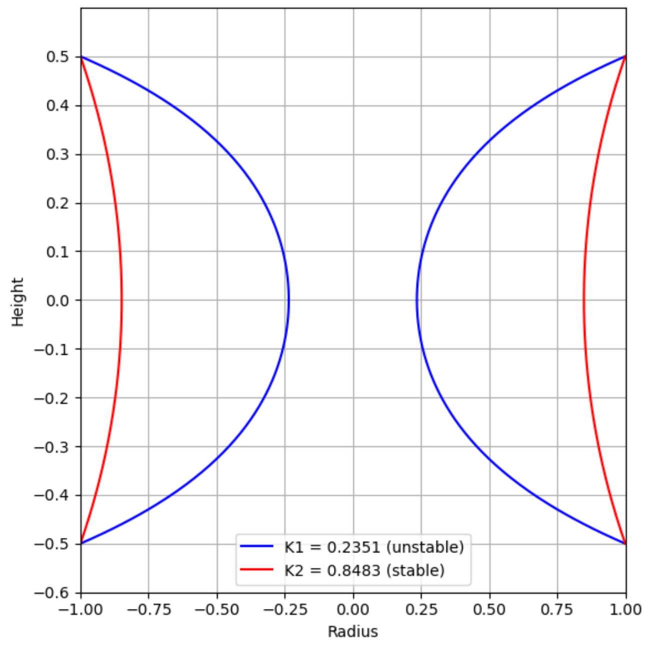

Figure 2. Comparison of capillary bridge profiles corresponding to two solutions of the shape parameter K that satisfy the boundary conditions for a liquid bridge between two loops. The blue curve represents the unstable solution ($K_1 = 0.2351$), while the red curve shows the stable solution ($K_2 = 0.8483$). The stable profile minimizes surface area and corresponds to the physically favored configuration.

$$S = 2\pi \int_{-H/2}^{H/2} r(z)\sqrt{1+(r'(z))^2}\,dz \tag{3}$$

Root	K	S	status
1	0.2351	6.8457	unstable
2	0.8483	5.9918	stable

Table 1. Corresponding values of K and surface area based on two roots found.

However, the profile generated by K_1 corresponds to a capillary bridge with larger area, while K_2 leads to a smaller surface area. The result is presented in figure 2. As was shown in class, the surface tension makes the system with a smaller area physically stable. With K_2, we can now calculate radius at various heights with equation 2.

1.1.2 Critical Height

The critical height H_{crit} marks the maximum possible separation between the two loops for which a stable capillary bridge can still exist. Physically, it represents the limit beyond which surface tension is no longer sufficient to support the liquid interface. At this point, the two roots of the shape equation converge, i.e., $K_1 = K_2$, and no real solution exists for larger H. In our case, for a fixed contact radius $\bar{R} = 1$, we incrementally increased H and numerically solved for K. The two distinct roots K_1 and K_2 eventually merged at $H_{\text{crit}} = 1.33$, where $K_1 = K_2 \approx 0.553$.

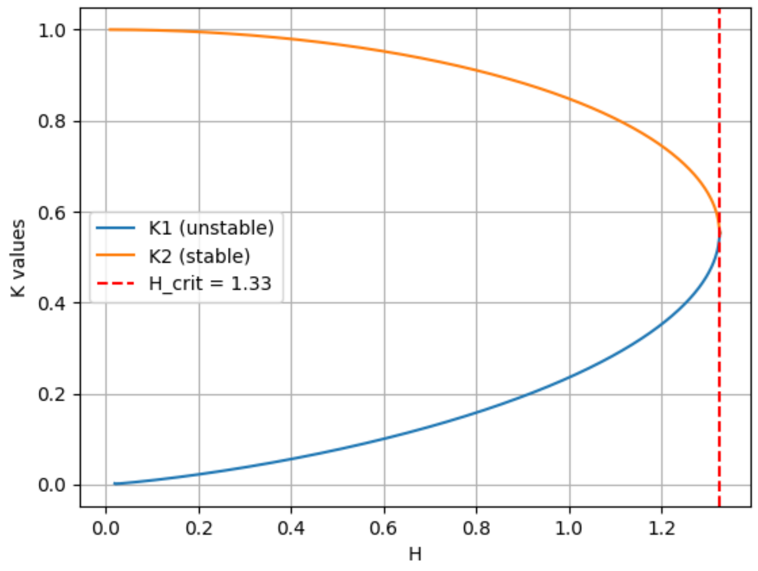

Figure 3. Evolution of K_1 and K_2 as a function of H. The critical height $H_{\text{crit}} = 1.33$ is reached when the two solutions merge.

1.1.3 Evolution of Neck Radius and Contact Angle vs Height

As the loop separation H increases, both the neck radius $r(0)$ and contact angle β evolve accordingly. These trends help identify the critical configuration beyond which the capillary bridge becomes unstable. The radius of the neck $r(0)$ is calculated with equation 2 with $z = 0$. The contact angle β is assessed with equation 4. Both numbers are computed with a larger and more stable K_2.

$$\beta = \pi/2 - \arctan[\sinh(H/2K)] \tag{4}$$

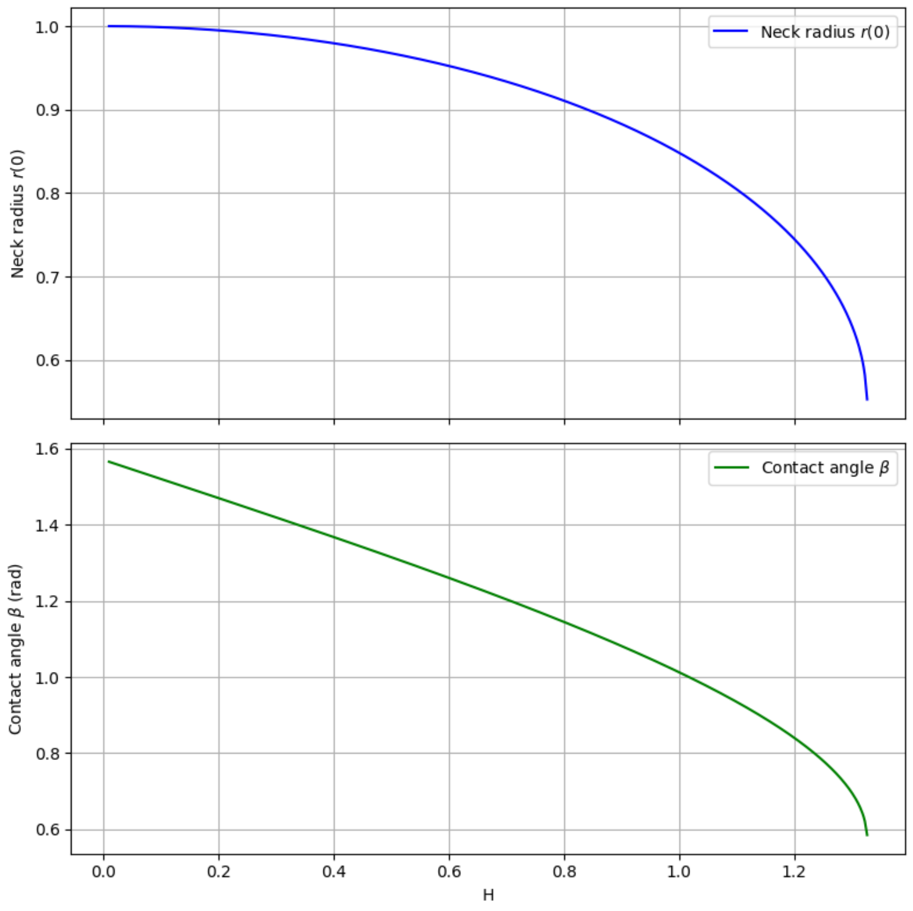

Figure 4. Evolution of Neck Radius $r(0)$ and Contact Angle β as a function of Height. Both $r(0)$ and β decreases until H reaches Hcrit

Frequency f_0 $[MHz]$	1
Amplitude p_a $[kPa]$	100
Density ρ_0 $[kg/m^3]$	1000
Sound Speed c_0 $[m/s]$	1500
β	3.5
δ $[m^2/s]$	4.3E-6

Table 2. Properties of fluid and sound wave

1.2 Distortion of Acoustic Waves

This section presents the numerical results obtained from solving the Burgers' equation for high-amplitude acoustic wave propagation. Physically, the problem investigates the nonlinear distortion of a sinusoidal pressure wave as it propagates through a viscous medium. The original governing equation, the Westervelt equation (5), captures the interplay between nonlinearity and thermoviscous damping. Under the assumption of a moving reference frame and one-way wave propagation, it reduces to a simplified form, equation (6) resembling Burgers' equation (7).

From a physical perspective, when a sound wave has sufficiently large amplitude, its positive and negative pressure phases do not travel at the same speed. This is a consequence of the nonlinear dependence of wave speed on pressure: regions of higher pressure (compression) propagate faster than regions of lower pressure (rarefaction). As a result, the wave begins to steepen over time, forming a sharp front—a precursor to a shock wave.

In the absence of dissipation, this steepening would eventually lead to a discontinuity in pressure (a true shock). However, due to the presence of viscous dissipation (characterized by the diffusivity parameter δ), the wavefront is smoothed out over time. This competition between nonlinear steepening and diffusive smoothing defines the wave dynamics observed in the simulations. The evolution of the wave profile is discussed in the first sub-problem. Pressure amplitude, and pressure gradient reveals two distinct regimes: an initial nonlinear growth where steepening dominates, followed by a dissipative regime where the wave energy attenuates and the profile gradually returns toward a sinusoidal shape. The third sub-problem also allows comparison between theoretical (lossless) and simulated (viscous) shock formation distances, providing further insight into the role of amplitude and frequency in nonlinear acoustic wave propagation.

This problem is solved based on the parameters in Table 2.

$$\frac{\partial^2 p}{\partial x^2}-\frac{1}{c_0^2}\frac{\partial^2 p}{\partial t^2}+\frac{\beta}{\rho_0 c_0^4}\frac{\partial^2 p^2}{\partial t^2}+\frac{\delta}{c_0^4}\frac{\partial^3 p}{\partial t^3}=0 \tag{5}$$

$$\frac{\partial p}{\partial t}=-\frac{\beta}{\rho_0 c_0}p\frac{\partial p}{\partial \chi}+\frac{\delta}{2}\frac{\partial^2 p}{\partial \chi^2} \tag{6}$$

$$\frac{\partial s}{\partial t}=-ms\frac{\partial s}{\partial \chi}+d\frac{\partial^2 s}{\partial \chi^2} \tag{7}$$

$$\frac{\partial \hat{s}}{\partial t}=-ikm\hat{s}^2-k^2 d\hat{s} \tag{8}$$

1.2.1 *Temporal Evolution of the Signal Profile*

The signal is defined as $S=p/p_a$, which indicates the normalized pressure. Since simplified Westervelt equation (6) has to match Burgers' equation (7). The non-linearity factor m and the diffusivity factor d must be solved first. After derivation, $m=-(\beta p_a)/(\rho_0 c_0)$ and $d=\delta/2$. With this setup, the solution of Burgers' equation can now be solved by iteratively switching between the spatial (7) and spectral (8) domains.

Figure 5 shows the evolution of the normalized pressure signal over time. At $t=0$, the wave is purely sinusoidal, representing a linear acoustic wave without distortion. However, as time progresses, nonlinear effects (represented by the $-mS\partial S/\partial x$ term in the Burgers' equation (7)) begin to distort the waveform. Physically, this occurs because the local sound speed increases with pressure, meaning that the compressive parts of the wave (positive pressure) propagate faster than the rarefactive parts (negative pressure). As a result, the wavefront begins to steepen, and by $t\approx 1.6$ ms, a clear shock-like front is observed where

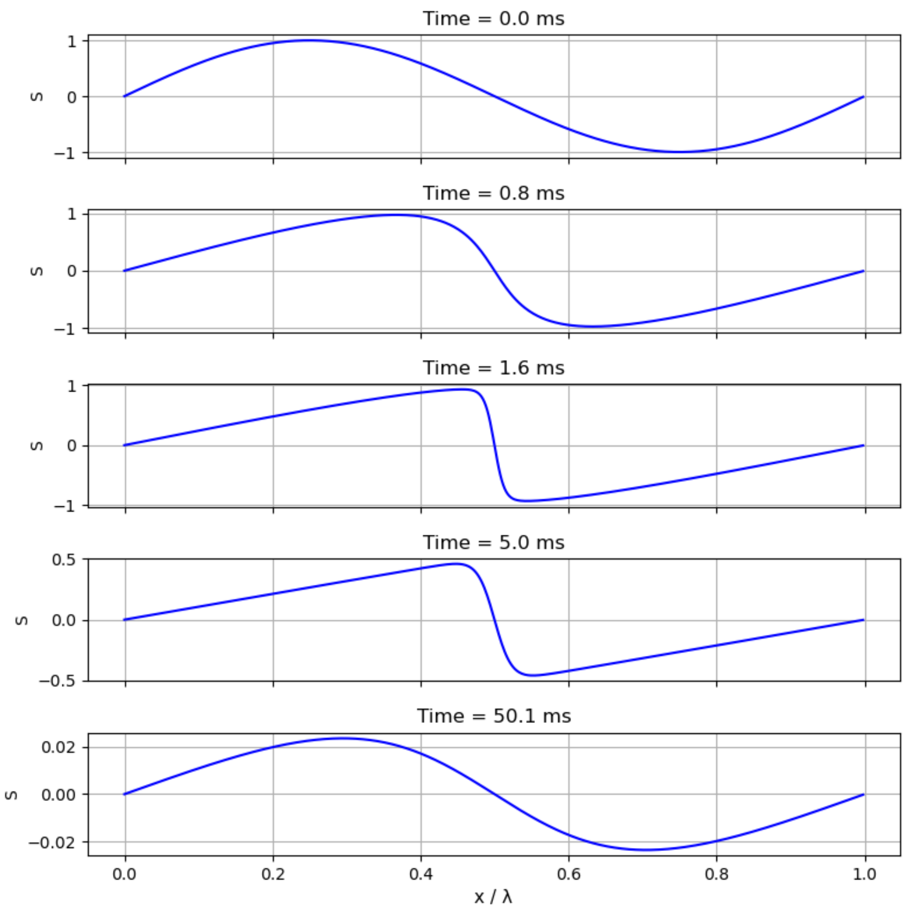

Figure 5. Temporal evolution of the normalized pressure signal $S = p/p_a$ over one wavelength (x/λ) at five selected time instances. Initially ($t = 0.0$ ms), the wave is a pure sinusoid. As time progresses to $t = 0.8$ ms, nonlinear steepening begins to appear on the wavefront. By $t = 1.6$ ms, the waveform develops a near-discontinuous shock front due to strong nonlinear effects. At $t = 5.0$ ms, dissipative effects dominate, significantly reducing the amplitude and smoothing the profile. Finally, at $t = 50.1$ ms, the signal has been strongly attenuated and resembles a low-amplitude sinusoidal wave due to diffusion. The x-axis is normalized by wavelength λ.

the gradient becomes extremely sharp. Eventually, dissipative effects (modeled by the diffusion term $d\,\partial^2 S/\partial x^2$) dominate, and the wave begins to smooth out. At $t = 50.1$ ms, the waveform has lost almost all of its original amplitude and appears sinusoidal again, but with much lower energy. This process illustrates a classic balance between nonlinearity and diffusion: nonlinear steepening leads to shock formation, while viscous damping spreads and attenuates the wave. The evolution mirrors the physical behavior of high-amplitude acoustic waves in thermoviscous media such as water, as described by the Westervelt equation (5).

1.2.2 Amplitude and Pressure Gradient Evolution

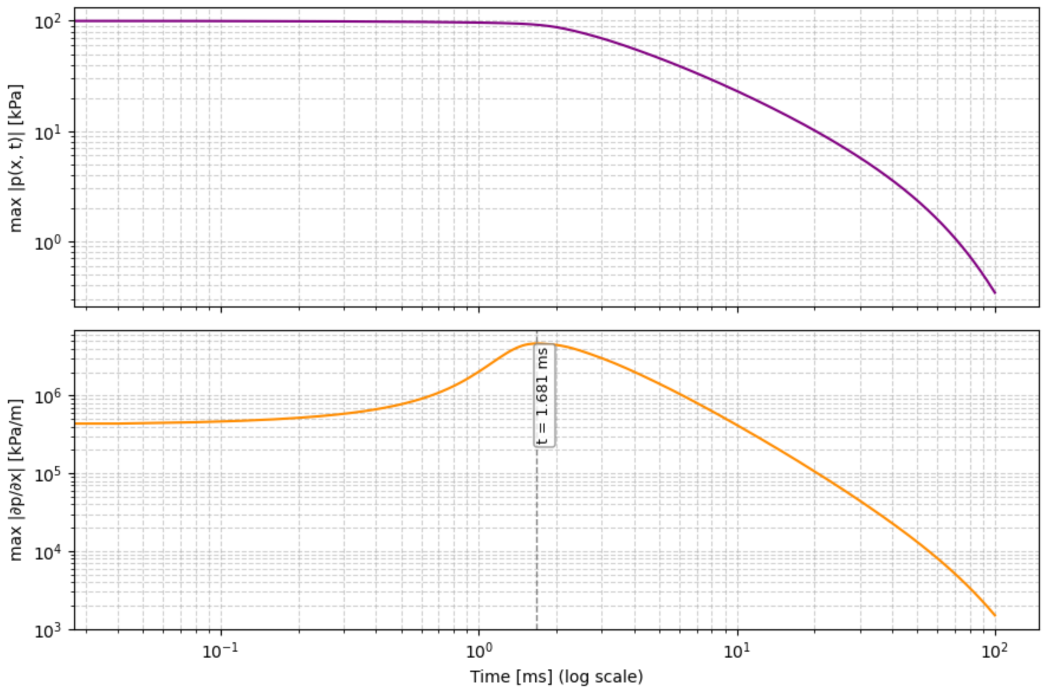

Figure 6. Time evolution of the pressure amplitude (top) and pressure gradient amplitude (bottom) of a high-amplitude acoustic wave. Log-log scale is used on both axes. The dashed line indicates the time of maximum pressure gradient amplitude.

Figure 6 illustrates the temporal evolution of the pressure amplitude, $|p(x,t)|$, and the pressure gradient amplitude, $|\partial p/\partial x|$, over time. Both quantities are plotted on logarithmic scales. The pressure amplitude remains nearly constant in the early phase and gradually decays after the wavefront steepening occurs. The pressure gradient amplitude, on the other hand, exhibits a sharp peak at $t \approx 1.68$ ms, which is marked by a vertical dashed line.

This behavior reflects two distinct physical regimes:

- **Nonlinear steepening regime (pre-shock):** Before the peak in gradient, nonlinear effects dominate the wave dynamics. Higher-pressure regions propagate faster than lower-pressure regions, leading to wavefront steepening. This results in a rapid increase in the pressure gradient amplitude, a precursor to shock formation.
- **Dissipative decay regime (post-shock):** After the shock forms, viscous and thermal diffusion mechanisms begin to dominate. These dissipative effects smooth out the pressure gradient and reduce the overall amplitude. Both the pressure amplitude and gradient decrease rapidly, indicating energy loss and the transition into a diffusion-controlled regime.

In summary, the figure demonstrates the transition from a nonlinear-dominated to a diffusion-dominated regime, with the peak in pressure gradient marking the onset of shock formation.

1.2.3 Shock Formation Distance and Lossless Theory Comparison

The shock formation distance $\bar{x}$ corresponds to the propagation distance at which the pressure gradient becomes maximum, indicating the onset of shock-like behavior. This distance is numerically extracted for different combinations of initial wave amplitudes and frequencies, as shown in Figure 7. In the plot, the shock formation distance decreases with increasing initial wave amplitude (P_a) and frequency (f_0). This matches physical intuition and the trend in Equation (9):

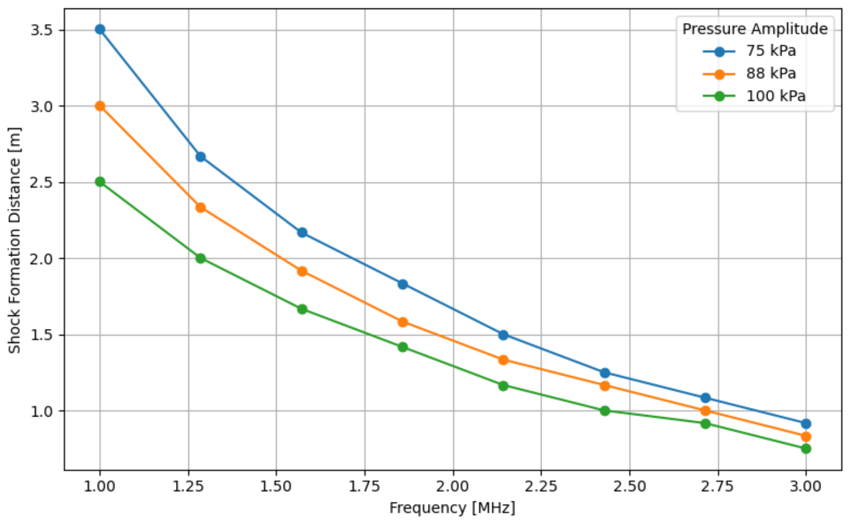

Figure 7. Shock formation distance as a function of initial wave frequency for various pressure amplitudes. The distances decrease with increasing P_a and f_0, consistent with theoretical expectations.

- A higher **amplitude** P_a introduces stronger nonlinearity in the wave, causing faster wavefront steepening and hence a shorter shock formation distance.
- A higher **frequency** f_0 leads to a shorter wavelength $\lambda = c_0/f_0$, which reduces the distance to form the steepening structure.

The theoretical prediction for shock formation distance in the *lossless case* ($\delta = 0$) is given by Equation (9). For $P_a = 100kPa$ and $f_0 = 1MHz$, the theoretical shock formation distance is: $\bar{x}_{\text{theory}} = 1.5347m$. The distance numerically obtained in the case with loss ($\delta > 0$) is: $\bar{x}_{\text{numerical}} = 2.5014m$

$$\bar{x}_{\text{theory}} = \frac{\rho_0 c_0^2}{\beta P_a k_0} \tag{9}$$

The discrepancy arises due to the inclusion of *viscous damping* ($\delta > 0$) in the numerical model. This term counteracts the nonlinear steepening by diffusing the waveform, delaying the shock formation. As a result, the shock forms at a greater distance compared to the idealized lossless case. Additionally, the theoretical formula assumes an instantaneous formation based on Burgers' equation, while the numerical simulation resolves the full temporal evolution, including transient effects and finite dissipation.

1.3 Rayleigh–Plateau Instability

The Rayleigh–Plateau instability describes the spontaneous breakup of a liquid jet or column due to surface tension effects. When small axisymmetric perturbations are introduced on the surface of a cylindrical liquid jet, surface tension tends to minimize surface area. As a result, certain wavelengths grow over time, eventually leading to pinch-off and droplet formation.

This study analyzes the linear stability of a water jet with physical parameters $\rho = 1000\ \text{kg/m}^3$, $\gamma = 72\ \text{mN/m}$, and initial radius $R_0 = 20$ mm. The jet moves downward at constant velocity $V_0 = 1$ m/s, and radial and axial coordinates are denoted by r and x, respectively. A reference frame x_{ref} is adopted that moves with the jet.

In the first part, the dispersion relation yields the dimensionless growth rate $\tau\,\text{Im}(\omega)$ as a function of the dimensionless wavenumber kR_0, from which the most unstable mode k_{crit} is determined. The associated pressure profile is expressed using modified Bessel functions. Numerically, the domain $x_{\text{ref}} \in [0, 0.5]$ m and initial perturbation amplitude $h = 2$ mm are used. In the second part, the radial and pressure perturbation fields at $t = 0$ are visualized using the critical mode.

1.3.1 Relation of Growth Rate and Wavelength

The dimensionless growth rate of the Rayleigh–Plateau instability is computed using the dispersion relation:

$$\tau = \sqrt{\frac{\rho R_0^3}{\gamma}} \tag{10}$$

$$\omega^2 = -\frac{1}{\tau^2} \cdot \left(\frac{kR_0 I_1(kR_0)}{I_0(kR_0)} \right) \cdot \left(1 - (kR_0)^2\right) \tag{11}$$

where I_0 and I_1 are the modified Bessel functions of the first kind. The imaginary part of ω is extracted and multiplied by τ to obtain the dimensionless growth rate $\tau\,\mathrm{Im}(\omega)$. A range of kR_0 values is sampled, and the peak of $\tau\,\mathrm{Im}(\omega)$ identifies the critical wavenumber k_{crit}.

As shown in Figure 8, the growth rate reaches a maximum of 0.3433 at $k_{\mathrm{crit}} = 0.6974/R_0$. For $R_0 = 20$ mm, this corresponds to a physical wavenumber $k_{\mathrm{crit}} \approx 34.87\ \mathrm{m}^{-1}$, and a critical wavelength $\lambda_{\mathrm{crit}} \approx 18$ cm. This mode grows fastest and dominates the breakup pattern. For $kR_0 > 1$, the growth rate becomes zero, indicating that a cylindrical jet is energetically more favorable than droplet breakup. This outcome is consistent with surface tension minimizing the surface area.

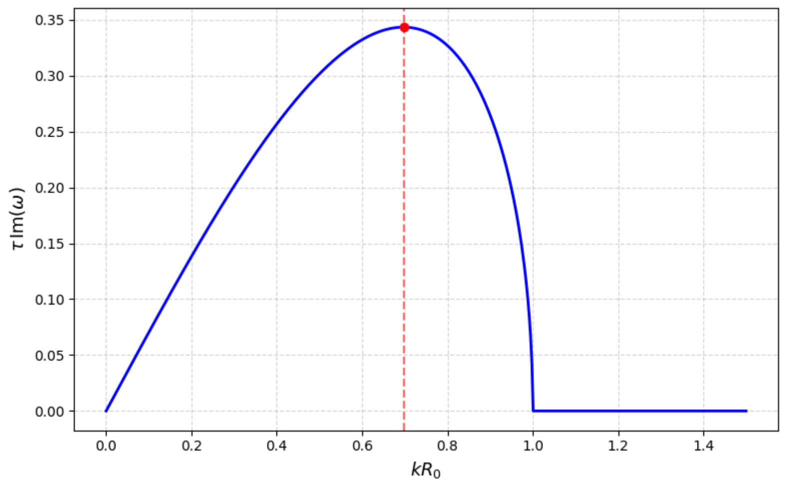

Figure 8. Dimensionless growth rate $\tau\,\mathrm{Im}(\omega)$ as a function of dimensionless wavenumber kR_0. The maximum occurs at $k_{\mathrm{crit}} = 0.6974/R_0$, corresponding to $\lambda_{\mathrm{crit}} \approx 18$ cm.

1.3.2 Radial Deformation and Pressure Perturbation Field

To visualize the most unstable mode, the critical wavenumber k_{crit} is used along with a perturbation amplitude $h = 2$ mm. The radial deformation and pressure field are evaluated at $t = 0$ in the moving frame $x_{\mathrm{ref}} = x - V_0 t$.

The radial deformation is calculated as:

$$R'(x_{\mathrm{ref}}, t) = h \exp\left[i \left(\left(k - \frac{\omega}{V_0} \right) x_{\mathrm{ref}} - kV_0 t \right) \right], \quad R = R_0 + \Re(R') \tag{12}$$

The pressure perturbation field is computed using:

$$p'(x_{\mathrm{ref}}, r, t) = \hat{p}(r) \exp\left[i \left(\left(k - \frac{\omega}{V_0} \right) x_{\mathrm{ref}} - kV_0 t \right) \right] \tag{13}$$

where the radial profile is defined as:

$$\hat{p}(r) = A\, I_0(kr), \quad \text{with} \quad \hat{p}(R) = \gamma \left(k^2 h - \frac{h}{R_0^2} \right) \tag{14}$$

Numerically, the interface $R(x_{\mathrm{ref}})$ is evaluated and used to define the radial bounds of the pressure field. The pressure is then computed inside the deformed jet using the Bessel function profile. The plot in Figure 9 shows the pressure distribution with the jet boundary overlaid.

$$\Delta p = \gamma \left(\frac{1}{R_{\mathrm{radial}}} + \frac{1}{R_{\mathrm{axial}}} \right) \tag{15}$$

The pressure field at $t = 0$ shows a clear sinusoidal variation along the axial direction with wavelength $\lambda_{\text{crit}} = 2\pi/k_{\text{crit}} \approx 18cm$, corresponding to the most unstable mode. The pressure varies more prominently along the axial direction than in the radial direction due to the Bessel function profile $I_0(kr)$ and the dominant contribution of radial curvature in the Young-Laplace equation (15). The pressure distribution reveals the physical mechanism of the Rayleigh–Plateau instability: regions with smaller radius (necks) exhibit higher pressure, while regions with larger radius (bulges) have lower pressure. This pressure gradient drives fluid flow from necks toward bulges, creating a positive feedback loop that amplifies the initial perturbation. Over time (not shown here), this exponential growth eventually leads to jet breakup into droplets separated by approximately one critical wavelength.

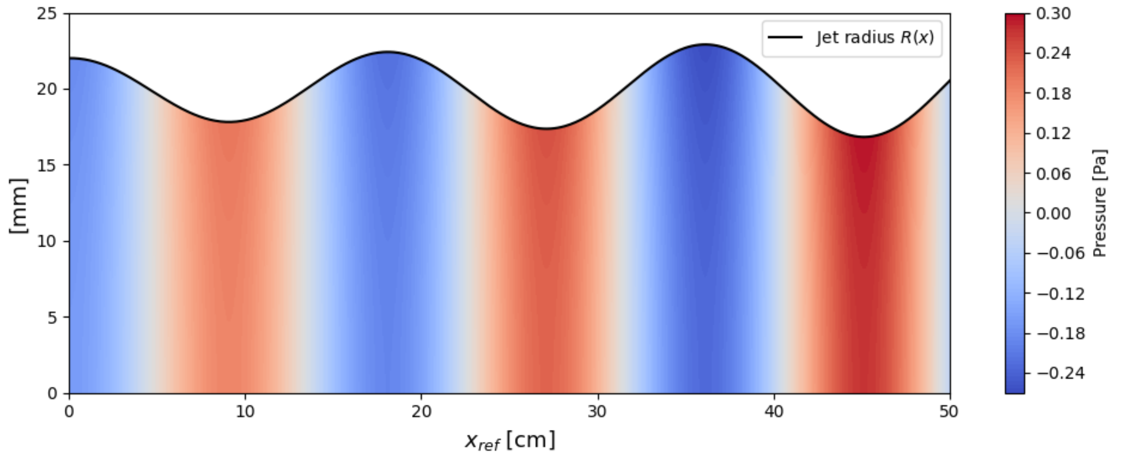

Figure 9. Pressure perturbation field $p'(x_{\text{ref}}, r)$ at $t = 0$, with jet interface $R(x)$ shown as a black line, showing the spatial variation of the most unstable mode.

2 Appendix

2.1 Question 1a

```python
# Question 1a - Capillary Bridge - Show why the larger K value is the stable solution
import numpy as np
import scipy.integrate as spi
import scipy.optimize as spo

R = 1.0  # Radius of the loop
H = 1.0  # Height of the capillary bridge

def equation_K(K):
    """Equation to solve for K given R and H"""
    return K * np.cosh(H/(2*K)) - R

def K_solution():
    """Solve for K using the given R and H"""
    K0_1 = 0.01      # Initial guess (smaller value)
    K0_2 = 100       # Another initial guess (larger value)
    K_sol_1 = spo.fsolve(equation_K, K0_1)
    K_sol_2 = spo.fsolve(equation_K, K0_2)
    return K_sol_1[0], K_sol_2[0]

def radius(z):
    """Calculate the radius of the capillary bridge at height z"""
    K1, K2 = K_solution()
    K = max(K1, K2) # Choose the larger K for the stable solution
    return K * np.cosh(z / K)

def contact_angle():
    """Calculate the contact angle (beta) at the loop"""
    K1, K2 = K_solution()
    K = max(K1, K2) # Choose the larger K for the stable solution
    beta = np.pi/2 - np.arctan(np.sinh(H/(2*K)))
    return beta

def area(K):
    """Calculate the surface area of the capillary bridge"""
    integrand = lambda z: K * np.cosh(z/K) * np.sqrt(1 + (np.sinh(z/K))**2)
    S, _ = spi.quad(integrand, -H/2, H/2) # Integrate from -H/2 to H/2
    return 2 * np.pi * S

if __name__ == "__main__":
    K1, K2 = K_solution()
    print("K1 value:", K1, "  K2 value:", K2)       #Question 1a
    print("Surface Area with K1:", area(K1))        #Question 1a
    print("Surface Area with K2:", area(K2))        #Question 1a
```

Listing 1. Shape profiles of the capillary bridge corresponding to the two solutions of the transcendental equation

2.2 Question 1b

```python
# Question 1b - Capillary Bridge - Show the existence of H_critical
import numpy as np
import matplotlib.pyplot as plt
import scipy.optimize as spo

R = 1.0                                # Radius of the loop
H_vals = np.linspace(0.01, 2, 1000) # H values to test, from 0.01 to 2
K1_list = []                           # Store K1 values
K2_list = []                           # Store K2 values
H_valid = []                           # Store valid H values where two solutions exist

def equation_K(K, H):
    """Equation to solve for K given R and H"""
    return K * np.cosh(H/(2*K)) - R

def K_solution():
    """Solve for K using the given R and H"""
    for H in H_vals:
        try:
            # Solve for K with two different initial guesses
            K0_1 = 0.0025
            K0_2 = 2.0
            K1 = spo.fsolve(equation_K, K0_1, args=(H, ))[0]
            K2 = spo.fsolve(equation_K, K0_2, args=(H, ))[0]

            # when K1 and K2 converge, we reach the critical height
            err = abs(K1-K2)
            if err < 5e-4 and H > 0.1:  # the setting of H > 0.1 is to avoid K1 = K2 at very small H
                break                   # H critical found

            # Store valid solutions
            K1_list.append(K1)
            K2_list.append(K2)
            H_valid.append(H)

        except Exception:
            continue

    return K1_list, K2_list, H_valid

if __name__ == "__main__":
    K1_list, K2_list, H_valid = K_solution()
    H_crit = H_valid[-1]
    print(f"Critical H = {H_crit:.6f}")

    # Plotting the results
    plt.plot(H_valid[5:], K1_list[5:], label="K1 (unstable)")
    plt.plot(H_valid, K2_list, label="K2 (stable)")
    plt.axvline(H_crit, color='r', linestyle='--', label=f"H_crit = {H_crit:.2f}")
    plt.xlabel("H")
    plt.ylabel("K␣values")
    plt.legend()
    plt.grid(True)
    plt.show()
```

Listing 2. critical height H_{crit}

2.3 Question 1c

```
# Question 1c - Capillary Bridge - Plot neck radius vs H and contact angle vs H
import numpy as np
import matplotlib.pyplot as plt
import scipy.optimize as spo

R = 1.0                                # Radius of the loop
H_vals = np.linspace(0.01, 2, 1000)  # H values to test, from 0.01 to 2
K1_list = []                           # Store K1 values
K2_list = []                           # Store K2 values
H_valid = []                           # Store valid H values where two solutions exist
beta_list = []                         # Store contact angle values
neck_radius_list = []                  # Store neck radius values

def equation_K(K, H):
    """Equation to solve for K given R and H"""
    return K * np.cosh(H/(2*K)) - R

def K_solution():
    """Solve for K using the given R and H"""
    for H in H_vals:
        try:
            # Solve for K with two different initial guesses
            K0_1 = 0.0025
            K0_2 = 2.0
            K1 = spo.fsolve(equation_K, K0_1, args=(H, ))[0]
            K2 = spo.fsolve(equation_K, K0_2, args=(H, ))[0]

            # when K1 and K2 converge, we reach the critical height
            err = abs(K1-K2)
            if err < 5e-4 and H > 0.1:  # the setting of H > 0.1 is to avoid K1 = K2 at very small H
                break                   # H critical found

            # Store valid solutions
            K1_list.append(K1)
            K2_list.append(K2)
            H_valid.append(H)

        except Exception:
            continue

    return K1_list, K2_list, H_valid

def contact_angle(K2_list, H_valid):
    """Calculate the contact angle (beta) at the loop"""
    for H, K in zip(H_valid, K2_list):
        beta = np.pi/2 - np.arctan(np.sinh(H / (2*K)))
        beta_list.append(beta)
    return beta_list

def neck_radius(K2_list):
    """Calculate the neck radius of the capillary bridge"""
    neck_radius_list = K2_list # at z=0, since cosh(0)=1
    return neck_radius_list

if __name__ == "__main__":
    K1_list, K2_list, H_valid = K_solution()
    beta_list = contact_angle(K2_list, H_valid)
    r_neck_list = neck_radius(K2_list)

    # --- Plotting in subplots ---
```

```python
    fig, axs = plt.subplots(2, 1, figsize=(8, 8), sharex=True)

    # Subplot 1: Neck radius
    axs[0].plot(H_valid, r_neck_list, color='blue', label="Neck radius $r(0)$")
    axs[0].set_ylabel("Neck radius $r(0)$")
    axs[0].legend()
    axs[0].grid(True)

    # Subplot 2: Contact angle
    axs[1].plot(H_valid, beta_list, color='green', label="Contact angle $\\beta$")
    axs[1].set_xlabel("H")
    axs[1].set_ylabel("Contact angle $\\beta$ (rad)")
    axs[1].legend()
    axs[1].grid(True)

    plt.tight_layout()
    plt.show()
```

Listing 3. Evolution of the neck radius $r(0)$ and the boundary contact angle β as functions of bridge height H.

2.4 Question 2a

```python
import numpy as np
import scipy.integrate as spi
import matplotlib.pyplot as plt

# Physical Parameters
f0 = 1E6         # frequency
Pa = 100E3       # pressure Amplitude (Reference)
rho0 = 1000      # water density [kg*m^-3]
c0 = 1500        # speed of sound in water [m/s]
beta = 3.5       # non-linearity coefficient
delta = 4.3E-6   # sound diffusivity

# Derived Parameters
Lambda = c0 / f0                        # wave length [m]
k0 = 2 * np.pi / Lambda                 # wavenumber
m = (beta / (rho0 * c0)) * Pa           # non dimensionalized m by mutiplying Pa
d = delta / 2                           # non dimensionalized diffusivity

# Space and Time
timespan = [0, 0.1]                                 # total time span [s]
timestep = np.linspace(0, timespan[1], 1000)        # descretized time range from 0-0.1s
NX = 512                                            # number of spatial points
X = np.linspace(0, Lambda, NX, endpoint=False)      # spatial domain from 0 to lambda
dX = X[1] - X[0]                                    # spatial step size

# Initial Condition
S0 = np.sin(k0 * X)

def BurgersEquation(t, S, X, m, d):
    '''Input in spatial, solved in spectral, transform again to spatial as output'''
    '''The output is an array of S at timestep t'''
    #Wave number discretization
    N_x = X.size
    dx = X[1] - X[0]
    k = 2*np.pi*np.fft.fftfreq(N_x, d = dx)

    #Spatial derivative in the Fourier domain
    S_hat = np.fft.fft(S)
    S_hat_x = 1j*k*S_hat
    S_hat_xx = -k**2*S_hat

    # Back to spatial domain
    S_x = np.fft.ifft(S_hat_x)
    S_xx = np.fft.ifft(S_hat_xx)

    # Compute Burgers equation time derivative
    S_t = -m*S*S_x + d*S_xx

    # Ensure real output
    S_t = S_t.real
    return S_t

def rhs(t, S):
    return BurgersEquation(t, S, X, m, d)

if __name__ == "__main__":
    sol = spi.solve_ivp(rhs,
                   timespan,
                   S0,
                   t_eval = timestep,
                   method='BDF')
    # print(sol)
    indices = [0, 8, 16, 50, 500]  # Different time spot
    fig, axes = plt.subplots(len(indices), 1, figsize=(8, 8), sharex=True)

    for i, idx in enumerate(indices):
        t = sol.t[idx]*1000
        S_profile = sol.y[:, idx]

        ax = axes[i]
        ax.plot(X / Lambda, S_profile, color='b')
        ax.set_ylabel("S")
        ax.set_title(f"Time␣=␣{t:.1f}␣ms")
        ax.grid(True)

    axes[-1].set_xlabel("x␣/␣  ", fontsize=11)
    plt.tight_layout()
    plt.show()
```

Listing 4. Python implementation of the pseudo-spectral solver for the Burgers' equation to simulate nonlinear acoustic wave distortion.

2.5 Question 2b

```python
import numpy as np
import scipy.integrate as spi
import matplotlib.pyplot as plt

# Physical Parameters
f0 = 1E6          # frequency
Pa = 100E3        # pressure Amplitude (Reference)
rho0 = 1000       # water density [kg*m^-3]
c0 = 1500         # speed of sound in water [m/s]
beta = 3.5        # non-linearity coefficient
delta = 4.3E-6    # sound diffusivity

# Derived Parameters
Lambda = c0 / f0                        # wave length [m]
k0 = 2 * np.pi / Lambda                 # wavenumber
m = (beta / (rho0 * c0)) * Pa           # non dimensionalized m by mutiplying Pa
d = delta / 2                           # non dimensionalized diffusivity

# Space and Time
timespan = [0, 0.1]                                 # total time span [s]
timestep = np.linspace(0, timespan[1], 2500)        # descretized time range from 0-0.1s
NX = 512                                            # number of spatial points
X = np.linspace(0, Lambda, NX, endpoint=False)      # spatial domain from 0 to lambda
dX = X[1] - X[0]                                    # spatial step size

# Initial Condition
S0 = np.sin(k0 * X)

def BurgersEquation(t, S, X, m, d):
    '''Input in spatial, solved in spectral, transform again to spatial as output'''
    '''The output is an array of S at timestep t'''
    #Wave number discretization
    N_x = X.size
    dx = X[1] - X[0]
    k = 2*np.pi*np.fft.fftfreq(N_x, d = dx)

    #Spatial derivative in the Fourier domain
    S_hat = np.fft.fft(S)
    S_hat_x = 1j*k*S_hat
    S_hat_xx = -k**2*S_hat

    # Back to spatial domain
    S_x = np.fft.ifft(S_hat_x)
    S_xx = np.fft.ifft(S_hat_xx)

    # Compute Burgers equation time derivative
    S_t = -m*S*S_x + d*S_xx

    # Ensure real output
    S_t = S_t.real
    return S_t

def rhs(t, S):
    return BurgersEquation(t, S, X, m, d)

if __name__ == "__main__":
    sol = spi.solve_ivp(rhs,
                  timespan,
                  S0,
                  t_eval = timestep,
                  method='BDF')
    Amplitude = np.max(np.abs(sol.y * Pa/1000), axis=0)
    Gradient = np.max(np.abs(np.gradient(sol.y * Pa/1000, dX, axis=0)), axis=0)
    t_all = sol.t*1000
    Max_grad = np.max(Gradient)
    Max_grad_idx = np.argmax(Gradient)
    Max_grad_time = t_all[Max_grad_idx]

    # === Plot Amplitude and Gradient in subplots ===
    fig, axes = plt.subplots(2, 1, figsize=(9, 6), sharex=True)

    # Subplot 1: Amplitude vs Time
    axes[0].plot(t_all, Amplitude, color='purple')
    axes[0].set_ylabel("max␣|p(x,␣t)|␣[kPa]")
    # axes[0].set_title("Maximum Pressure Amplitude vs Time")
    axes[0].grid(True, which='both', linestyle='--', alpha=0.6)

    # Subplot 2: Gradient vs Time
    axes[1].plot(t_all, Gradient, color='darkorange')
    axes[1].set_ylabel("max␣|  p  /  x  |␣[kPa/m]")
    axes[1].set_xlabel("Time␣[ms]␣(log␣scale)")
    # axes[1].set_title("Maximum Pressure Gradient vs Time")
    axes[1].grid(True, which='both', linestyle='--', alpha=0.6)
    axes[1].axvline(Max_grad_time, color='gray', linestyle='--', linewidth=1)

    # Annotate maximum gradient point
    label_text = f"t␣=␣{Max_grad_time:.3f}␣ms"
    axes[1].text(Max_grad_time, Max_grad * 0.9,
                label_text, rotation=90, color='black',
                fontsize=9, va='top', ha='left',
                bbox=dict(boxstyle="round,pad=0.2", fc="white", ec="gray", alpha=0.8))

    # Set log scale on both axes
    axes[0].set_xscale('log')
    axes[1].set_xscale('log')
    axes[0].set_yscale('log')
    axes[1].set_yscale('log')

    plt.tight_layout()
    plt.show()
```

Listing 5. Computation of the temporal evolution of pressure amplitude and pressure gradient using the numerical solution of the Burgers' equation. Log-scale plots are generated to analyze two distinct propagation regimes.

2.6 Question 2c

```
import numpy as np
import scipy.integrate as spi
import matplotlib.pyplot as plt

# Physical Parameters
f0 = 1E6        # frequency
Pa = 100E3      # pressure Amplitude (Reference)
rho0 = 1000     # water density [kg*m^-3]
c0 = 1500       # speed of sound in water [m/s]
beta = 3.5      # non-linearity coefficient
delta = 4.3E-6  # sound diffusivity

# Derived Parameters
Lambda = c0 / f0                        # wave length [m]
k0 = 2 * np.pi / Lambda                 # wavenumber
m = (beta / (rho0 * c0)) * Pa           # non dimensionalized m by mutiplying Pa
d = delta / 2                           # non dimensionalized diffusivity

# Space and Time
timespan = [0, 0.1]                                 # total time span [s]
timestep = np.linspace(0, timespan[1], 1800)        # descretized time range from 0-0.1s
NX = 512                                            # number of spatial points
X = np.linspace(0, Lambda, NX, endpoint=False)      # spatial domain from 0 to lambda
dX = X[1] - X[0]                                    # spatial step size

# Initial Condition
S0 = np.sin(k0 * X)

def BurgersEquation(t, S, X, m, d):
    '''Input in spatial, solved in spectral, transform again to spatial as output'''
    '''The output is an array of S at timestep t'''
    #Wave number discretization
    N_x = X.size
    dx = X[1] - X[0]
    k = 2*np.pi*np.fft.fftfreq(N_x, d = dx)

    #Spatial derivative in the Fourier domain
    S_hat = np.fft.fft(S)
    S_hat_x = 1j*k*S_hat
    S_hat_xx = -k**2*S_hat

    # Back to spatial domain
    S_x = np.fft.ifft(S_hat_x)
    S_xx = np.fft.ifft(S_hat_xx)

    # Compute Burgers equation time derivative
    S_t = -m*S*S_x + d*S_xx

    # Ensure real output
    S_t = S_t.real
    return S_t

def rhs(t, S):
    return BurgersEquation(t, S, X, m, d)

def lossless_prop_x():
    return (rho0 * c0**2) / (beta * Pa * k0)

def prop_x(Max_grad_time):
    return Max_grad_time * c0

def max_grad_time(gradient, t_all):
```

```python
    Max_grad_idx = np.argmax(gradient)
    Max_grad_time = t_all[Max_grad_idx]
    return Max_grad_time


if __name__ == "__main__":
    sol = spi.solve_ivp(rhs,
                   timespan,
                   S0,
                   t_eval = timestep,
                   method='BDF')
    Amplitude = np.max(np.abs(sol.y * Pa/1000), axis=0)
    Gradient = np.max(np.abs(np.gradient(sol.y * Pa, dX, axis=0)), axis=0)
    t_all = sol.t
    Max_grad = np.max(Gradient)
    Max_grad_time = max_grad_time(Gradient, t_all)

    print(lossless_prop_x)
    print(prop_x(Max_grad_time))
    if lossless_prop_x() != prop_x(Max_grad_time):
        print("Shock␣formation␣distance␣does␣not␣match␣the␣lossless␣theoretical␣value.")
        print ("Lossless␣theoretical␣shock␣formation␣distance:␣{:.4f}␣m".format(lossless_prop_x()))
        print ("Numerical␣shock␣formation␣distance:␣{:.4f}␣m".format(prop_x(Max_grad_time)))
    else:
        print("Shock␣formation␣distance␣matches␣the␣lossless␣theoretical␣value.")
        print("Shock␣formation␣distance:␣{:.4f}␣m".format(lossless_prop_x()))

    Pa_values = np.linspace(75E3, 100E3, 3)
    f_values = np.linspace(1E6, 3E6, 8)

    # Create a 2D array to store shock distances: rows = Pa, cols = f
    shock_distance_matrix = np.zeros((len(Pa_values), len(f_values)))

    # Loop over pressure and frequency values
    for i, p in enumerate(Pa_values):
        for j, f in enumerate(f_values):
            k0 = 2 * np.pi * f / c0
            m = (beta / (rho0 * c0)) * p
            S0 = np.sin(k0 * X)

            def rhs_local(t, S):
                return BurgersEquation(t, S, X, m, d)

            sol = spi.solve_ivp(rhs_local,
                                timespan,
                                S0,
                                t_eval=timestep,
                                method='BDF')

            Gradient = np.max(np.abs(np.gradient(sol.y * p, dX, axis=0)), axis=0)
            shock_distance = prop_x(max_grad_time(Gradient, sol.t))
            shock_distance_matrix[i, j] = shock_distance

    # Plot shock distance vs frequency for each Pa
    plt.figure(figsize=(8, 5))
    for i, p in enumerate(Pa_values):
        plt.plot(f_values / 1e6,                  # Convert to MHz
                shock_distance_matrix[i, :],
                marker='o',
                label=f"{p/1000:.0f}␣kPa")

    plt.xlabel("Frequency␣[MHz]")
    plt.ylabel("Shock␣Formation␣Distance␣[m]")
    # plt.title("Shock Distance vs Frequency at Different Pa")
    plt.grid(True)
    plt.legend(title="Pressure␣Amplitude")
    plt.tight_layout()
    plt.show()
```

Listing 6. Calculation of the shock formation distance for various initial wave amplitudes and frequencies, and comparison with the analytical lossless theory.

2.7 Question 3a

```python
# Question 3a - Rayleigh-Plateau Instability -
# Plot capillary timescale growth coefficient vs dimentionless wavenumber
# Show max growth rate and corresponding wavenumber

import numpy as np
from scipy.special import iv
import matplotlib.pyplot as plt

density = 1000           # Density of water (kg/m^3)
surface_tension = 0.072  # Surface tension of water (N/m)
R0 = 20e-3               # Radius of the jet (m)
jet_velocity = 1.0       # Velocity of the jet (m/s)
k_list = np.linspace(0/R0, 1.5/R0, 500) # Dimentionless wavenumber values

def capillary_time():
    '''Capillary timescale'''
    return np.sqrt(density * R0**3 / surface_tension)

def dispersion_omega(k):
    '''Dispersion relation   (k)'''
    omega_sq = (-1 / capillary_time()**2) * ((k*R0 * iv(1, k*R0)) / (iv(0, k*R0)) * (1 - (k*R0)**2))
    omega = np.lib.scimath.sqrt(omega_sq)
    return omega

if __name__ == "__main__":
    kr = k_list * R0                    # dimensionless wavenumber k * R0
    growth_rate = np.imag(dispersion_omega(k_list) * capillary_time()) # imaginary part of dimensionless growth rate    * t_cap

    mask = np.isfinite(growth_rate) # filter out non-finite values

    kr_plot = kr[mask]                # filtered dimensionless wavenumber
    growth_plot = growth_rate[mask] # filtered dimensionless growth rate

    max = np.nanargmax(growth_plot) # index of max growth rate
    kr_max = kr_plot[max]           # corresponding dimensionless wavenumber
    growth_max = growth_plot[max]   # max growth rate

    print (f'Max growth rate = {growth_max:.4f} at kR_0 = {kr_max:.4f}')

    plt.figure(figsize=(8,5))
    plt.plot(kr_plot, growth_plot, lw=2, color='b')
    plt.plot(kr_max, growth_max, 'ro')
    plt.axvline(kr_max, color='r', linestyle='--', alpha=0.6)

    plt.xlabel(r'$k R_0$', fontsize=13)
    plt.ylabel(r'$\tau \, \mathrm{Im}(\omega)$', fontsize=13)

    plt.grid(True, linestyle='--', alpha=0.5)
    plt.tight_layout()
    plt.show()
```

Listing 7. Python code for computing the Rayleigh-Plateau dispersion relation

2.8 Question 3b

```python
# Question 3b - Rayleigh-Plateau Instability -
# Plot radial deformation R = R0 + R'
# Plot pressure variation field p'(xref, r)

import numpy as np
```

```python
from scipy.special import iv
import matplotlib.pyplot as plt

density = 1000          # Density of water (kg/m^3)
surface_tension = 0.072 # Surface tension of water (N/m)
R0 = 20e-3              # Radius of the jet (m)
jet_velocity = 1.0      # Velocity of the jet (m/s)
k = 0.69739479/R0       # Dimentionless wavenumber
h = 2e-3                # Initial perturbation amplitude (m)
t = 0.0                 # Time (s)

x_ref = np.linspace(0, 0.5, 500)  # Reference x positions along the jet

def capillary_time():
    '''Capillary timescale'''
    return np.sqrt(density * R0**3 / surface_tension)

def dispersion_omega():
    '''Dispersion relation   (k)'''
    omega_sq = (-1 / capillary_time()**2) * ((k*R0 * iv(1, k*R0)) / (iv(0, k*R0)) * (1 - (k*R0)**2))
    omega = np.lib.scimath.sqrt(omega_sq)
    return omega

def radial_deformation():
    '''Radial deformation R = R0 + R' at reference positions x_ref and time t'''
    R_prime = h * np.exp(1j * ((k - dispersion_omega()/jet_velocity)*x_ref - k*jet_velocity*t))
    return R0 + R_prime.real

def pressure_variation():
    '''Pressure variation field p'(xref, r) at time t'''
    omega = dispersion_omega()
    phi = (k - omega / jet_velocity) * x_ref - k * jet_velocity * t  # shape of phase at x_ref

    R_profile = radial_deformation() # shape of interface along x_ref
    r_max = np.max(R_profile)
    r = np.linspace(0, r_max, 1000)  # create r grid up to max radius

    # Meshgrid for final 2D field
    p_prime_grid = np.full((len(x_ref), len(r)), np.nan)

    # Boundary condition
    pR = surface_tension * (k**2 * h - h / R0**2)

    for i, R_now in enumerate(R_profile):
        r_mask = r <= R_now  # mask for points inside the jet at this x_ref
        A = pR / iv(0, k * R_now)
        p_hat_r = A * iv(0, k * r[r_mask])
        p_prime_grid[i, r_mask] = np.real(p_hat_r * np.exp(1j * phi[i]))

    return r, p_prime_grid

if __name__ == "__main__":
    r, p_prime = pressure_variation()

    # Plot
    # Note:
    # Radial pressure variation is not obvious (smaller r has smaller aplitude p_hat_r)
    # Only axial direction is clear
    plt.figure(figsize=(10, 4))
    contour = plt.contourf(x_ref*100, r*1000, p_prime.T, 100, cmap='coolwarm')  # transpose for (r, x)
    plt.colorbar(contour, label='Pressure␣[Pa]')
    plt.xlabel(r'$x_{ref}$␣[cm]', fontsize=13)
    plt.ylabel(r'[mm]', fontsize=13)
    # plt.title("Pressure perturbation $p'(x_{ref, r)$ at $t=0$")
    plt.ylim(0, 25)
    plt.tight_layout()
    plt.plot(x_ref*100, radial_deformation()*1000, 'k-', label='Jet␣radius␣$R(x)$')
    plt.legend()
    plt.show()
```

Listing 8. Pressure perturbation field $p'(x_{\text{ref}}, r)$ and radial deformation R at $t = 0$ in the moving reference frame