```
# -*- coding: utf-8 -*-
"""
Created on Tue Sep 30 15:49:36 2025

@author: coled
"""

import numpy as np

# ---------- part d: variable friction ----------


def sutherland(T, T0=273.15, mu0=1.716e-5):
    return mu0 * (T/T0)**1.5 * (T0 + 110.4) / (T + 110.4)

def Re(mu, u, rho, D):
    return rho * u * D / mu


def friction_func(D, rho, u, mu, epsilon=1.0e-3):
    den = np.log10( (epsilon/(3.7*D)) + 5.74/(Re(mu, u, rho, D)**0.9) )
    return 0.0625 / (den**2)

def pressure_isen(M, gamma=1.4):          # p/p0
    return (1.0 + 0.5*(gamma-1.0)*M**2) ** (-gamma/(gamma-1.0))

def temp_ratio_isen(M, gamma=1.4):         # T/T0
    return 1.0 / (1.0 + 0.5*(gamma-1.0)*M**2)

def u_func(M, T, gamma=1.4, R=287.0):
    return M * np.sqrt(gamma * R * T)


def dm_dx(M, f, x, gamma=1.4):

    c1 = (1 + (gamma-1)*0.5 * M **2) / ( (M**2 - 1))
    c2 = (gamma*M**2) * f / ( np.exp(1/(x+1)))
    c3 = -2 / ((x+1)**2)
    return M * c1 * (c3-c2)

def rho_func(mdot, u, A):
    return mdot / (u * A)


dx = 1.0e-3
xmin, xmax = 0.0, 100.0
n_samples = int((xmax - xmin) / dx)

x   = np.linspace(xmin, xmax, n_samples)
M   = np.zeros_like(x)
D   = np.zeros_like(x)
u   = np.zeros_like(x)
A   = np.zeros_like(x)
T   = np.zeros_like(x)
rho = np.zeros_like(x)
p   = np.zeros_like(x)
```

```python
p0_array = np.zeros_like(x)
T0_array = np.zeros_like(x)
f    = np.zeros_like(x)
Re_array = np.zeros_like(x)
mu   = np.zeros_like(x)

# geometry
for i in range(len(x)):
    D[i] = 2.0 * np.exp(1.0/(x[i] + 1.0))
    A[i] = np.pi * (0.5 * D[i])**2

# givens
T0 = 600.0
p0 = 200.0e3
R  = 287.0

# inlet state
M[0]   = 6.0
T[0]   = T0 * temp_ratio_isen(M[0])
u[0]   = u_func(M[0], T[0])
p[0]   = p0 * pressure_isen(M[0])
rho[0] = p[0] / (R * T[0])
mdot   = rho[0] * u[0] * A[0]
mu[0]  = sutherland(T[0])
Re_array[0] = Re(mu[0], u[0], rho[0], D[0])
f[0]   = friction_func(D[0], rho[0], u[0], mu[0])

tol = 1e-3

# integrate M(x) while updating f(x)
def find_choke(M=M, x=x, f=f, mdot=mdot):
    i_break = len(x) - 1
    for i in range(len(x) - 1):
        # march M with current f[i]
        M[i+1] = M[i] + dx * dm_dx(M[i], f[i], x[i])

        # update properties from M[i+1]
        T[i+1]   = T0 * temp_ratio_isen(M[i+1])
        u[i+1]   = u_func(M[i+1], T[i+1])
        rho[i+1] = rho_func(mdot, u[i+1], A[i+1])
        mu[i+1]  = sutherland(T[i+1])
        Re_array[i+1] = Re(mu[i+1], u[i+1], rho[i+1], D[i+1])
        f[i+1] = friction_func(D[i+1], rho[i+1], u[i+1], mu[i+1])

        if (abs(M[i+1] - 1.0) < tol) or (M[i+1] < 1.0):
            i_break = i + 1
            break

    return (x[i], M, i_break)

L1, check_M, i_break = find_choke()
avg_f = np.mean(f[:i_break+1])
```