```python
# -*- coding: utf-8 -*-
"""
Created on Mon Nov 10 14:33:33 2025

@author: coled
"""

# IFM HW26: Viscous Shock Solver

import numpy as np
from scipy.interpolate import PchipInterpolator
import matplotlib.pyplot as plt

# some relevant function definitions

# u2 in terms of u1, gamma, M1
def u2_M1(u1, gamma, M1):
    gam_term = (gamma-1.0)/(gamma+1.0)
    M_term = 1.0 + 2/((gamma-1.0)*M1**2)
    return u1*gam_term*M_term

# M1 from some initial conditions
def M1_int(u1, gamma, R, T1):
    return np.sqrt(u1**2/(gamma*R*T1))

# big daddy
def x_u(gamma, mu, rho1, R, T1, u1, u, xin=0, uin=2999):
    M1 = M1_int(u1, gamma, R, T1)
    u2 = u2_M1(u1, gamma, M1)
    t1 = 8*gamma*mu/(3*(gamma+1)*rho1*u1)
    t2 = u2/(u1-u2) * np.log((u-u2)/(uin-u2))
    t3 = u1/(u1-u2) * np.log((u1-u)/(u1-uin))
    return xin - t1*(t2-t3)

# interpolation to build inverse u_x
def u_x(xq, x_sorted, u_sorted):
    pchip = PchipInterpolator(x_sorted, u_sorted)
    return pchip(xq)

# rho from u(x)
def rho_func(rho1, u1, u):
    return rho1*u1/u
# T from u(x)
def T_func(T1, u1, u, cp):
    return T1 + (u1**2 - u**2)/(2.0*cp)

# entropy change function
def s_func(rho1, rho, gamma, p, p1):
    t1 = (rho1/rho)**gamma
    t2= p/p1
    return np.log(t1*t2)

def p_func(rho, R, T):
    return rho*R*T

# irreversibility production rate
def Idot_func(mu, T, cp, u, x):
```

```
    dTdx = np.gradient(T, x, edge_order = 2)
    dudx = np.gradient(u, x, edge_order=2)

    t1 = 4*mu/(3*T**2)
    t2 = cp*dTdx**2
    t3 = T*dudx**2
    return t1*(t2+t3)

# now define some initial conditions

u1 = 3000
p1 = 10**5
T1 = 300
xin = 0
uin = 2999

# for helium, gamma=5/3, R = 2077 J/kg*K, mu = 1.86*10^-5
gamma = 5/3
R = 2077
cp = gamma*R/(gamma-1)
mu = 1.86*10**-5

# back out rho
rho1 = p1/(R*T1)

# initial mach number, fully shocked u2
M1 = M1_int(u1, gamma, R, T1)
u2 = u2_M1(u1, gamma, M1)

# Take u as an array over N points; slightly change u1, u2 to avoid singularities
tol = 1e-8
N = 20000
u = np.linspace(u1-tol, u2+tol, N)

# find the x over those u's
x = x_u(gamma, mu, rho1, R, T1, u1, u)
# sort to ensure constant increase
idx = np.argsort(x)
x = x[idx]
u = u[idx]

# evaluate state on xgrid of choice
xq = np.linspace(x[0], x[-1], 5000)
uq = u_x(xq, x, u)

# plug in to exisiting functions to get desired values
rho = rho_func(rho1, u1, uq)
T = T_func(T1, u1, uq, cp)
p = p_func(rho, R, T)
s = s_func(rho1, rho, gamma, p, p1)
idot = Idot_func(mu, T, cp, uq, xq)


# plot answers
fig, ax = plt.subplots(1, 3)

#u(x)
```

```
ax[0].plot(xq, uq)
ax[0].set_title("u(x)")
ax[0].set_xlabel("x(m)")
ax[0].set_ylabel("u (m/s)")

# s function
ax[1].plot(xq, s)
ax[1].set_title("(s(x)-s)/cv")
ax[1].set_xlabel("x(m)")
ax[1].set_ylabel("s(x)-...")

# idot
ax[2].plot(xq, idot)
ax[2].set_title("Idot")
ax[2].set_xlabel("x(m)")
ax[2].set_ylabel("Idot")
```