

HW 8: Mini Computer Project - PWR Modeling

Arnav Goyal

12 November 2025

Contents

Contents	2
1 Temperature in Fluid Equations	1
2 Temperature in Rod Equations	2
3 Results	4
4 Code	8

1 Temperature in Fluid Equations

In a PWR, the liquid should remain sub-cooled the entire time - χ_e should remain below 0 for the entire channel.

The heat generation in the fuel is given with Equation 1 and all of the variables are present in the code.

$$q'(z) = q'_0 \cdot \cos(\pi z/H_e) \left[\frac{W}{m} \right] \quad (1)$$

This can be converted to other measured fluxes for the rod surfaces (q''), or heat generation within the fuel pellet (q''') using the relations in Equation 2.

$$q[W] = q' \left[\frac{W}{m} \right] \cdot \Delta z[m] = q'' \left[\frac{W}{m^2} \right] \cdot (2\pi r_{fuel} \cdot \Delta z)[m^2] = q''' \left[\frac{W}{m^3} \right] \cdot (\pi r_{fuel}^2 \cdot \Delta z)[m^3] \quad (2)$$

This is the mass conservation equation for the fluid, it is used to ensure that G , the momentum of the fluid or ρv , is constant.

$$\frac{d\rho}{dz} + \frac{d\rho v}{dz} = 0$$

This is the momentum conservation equation for the fluid, it is used to calculate the pressure drop along the channel.

$$\begin{aligned} \frac{d\rho v}{dt} + \frac{d\rho v v}{dz} &= -\frac{dP}{dz} - \frac{\tau_f \xi_w}{A_f} - \rho g \sin(\theta) \\ -\frac{dP}{dz} &= G^2 \frac{d}{dz} \frac{1}{\rho} + \frac{\tau_f \xi_w}{A_f} + \rho g + \frac{d\rho v}{dt} \end{aligned}$$

Where $\tau_f = \frac{1}{2} f \frac{G^2}{\rho}$, and $f = f_{1\phi} = 0.316 Re^{-0.25}$

$$\begin{aligned} -\frac{dP}{dz} &= \frac{1}{2} (0.316 Re^{-0.25}) \frac{G^2}{\rho} \frac{\xi_w}{A_f} + \rho g \\ \frac{dP}{dz} &= \frac{P^{i+1} - P^i}{\Delta z} \end{aligned}$$

This is the energy conservation equation for the fluid, it is used to calculate the change in equilibrium quality along the channel, the corresponding heat

transfer coefficients (h), and the change in temperature along the channel.

$$\begin{aligned}
\frac{d}{dt}\rho_m h + \frac{d}{dz}\rho_m v h &= \frac{q''\xi_w}{A_f} + \frac{dP}{dt} + q''' \\
\frac{dh}{dz} &= \frac{q''\xi_w}{A_f G} \\
\cancel{\frac{d}{dt}\rho_m h} + G \frac{d}{dz} h &= \frac{q''\xi_w}{A_f} + \cancel{\frac{dP}{dt}} + \cancel{q'''} \\
\frac{dh}{dz} &= \frac{q''\xi_w}{A_f G} \\
h &= \chi_e \cdot h_{fg} + h_f \\
\cancel{\frac{dh_f}{dz}} + h_{fg} \frac{d\chi_e}{dz} &= \frac{q''\xi_w}{A_f G} \\
\frac{d\chi_e}{dz} &= \frac{q''\xi_w}{A_f \cdot G \cdot h_{fg}} = \frac{\chi_e^{i+1} - \chi_e^i}{\Delta z}
\end{aligned}$$

2 Temperature in Rod Equations

Region 1 is for $r < r_{fuel}$ corresponding to the fuel pellet where heat is being generated. Region 2 corresponds to the gap between the pellet and clad and Region 3 corresponds to the cladding. The dimensions of these different regions are all given in the code.

$$\begin{aligned}
\nabla k \nabla T_1 &= -q''' \\
\frac{1}{r} \frac{d}{dr} r \frac{dT_1}{dr} &= -\frac{q'(z)}{\pi r_{fuel}^2 \cdot k_{fuel}} \\
\frac{d}{dr} r \frac{dT_1}{dr} &= -\frac{q'(z)}{\pi r_{fuel}^2 \cdot k_{fuel}} r \\
r \frac{dT_1}{dr} &= -\frac{q'(z)}{\pi r_{fuel}^2 \cdot k_{fuel}} \frac{r^2}{2} + C_1 \\
\frac{dT_1}{dr} &= -\frac{q'(z)}{\pi r_{fuel}^2 \cdot k_{fuel}} \frac{r}{2} + \frac{C_1}{r} \\
T_1(r) &= -\frac{q'(z)}{\pi r_{fuel}^2 \cdot k_{fuel}} \frac{r^2}{4} + C_1 \cdot \ln(r) + C_2
\end{aligned}$$

$$\begin{aligned}
\nabla k \nabla T_2 &= 0 \\
\frac{1}{r} \frac{d}{dr} r \frac{dT_2}{dr} &= 0 \\
\frac{dT_2}{dr} &= \frac{C_3}{r} \\
T_2(r) &= C_3 \ln(r) + C_4 \\
\nabla k \nabla T_3 &= 0 \\
\frac{1}{r} \frac{d}{dr} r \frac{dT_3}{dr} &= 0 \\
\frac{dT_3}{dr} &= \frac{C_5}{r} \\
T_3(r) &= C_5 \ln(r) + C_6
\end{aligned}$$

Once general equations are solved for, the boundary conditions can be applied to each vertical slice (Δz) to get the temperatures at each point in the rod.

B.C.s

$$\begin{aligned}
1. \frac{dT_1}{dr} \Big|_{r=0} &= 0 \text{ due to symmetry} \\
&\text{So, } C_1 = 0 \\
2. -k_{fuel} \frac{dT_1}{dr} \Big|_{r=r_{fuel}} &= -k_{gap} \frac{dT_2}{dr} \Big|_{r=r_{fuel}} \\
-k_{fuel} \left(-\frac{q'(z)}{\pi r_{fuel}^2 \cdot k_{fuel}} \frac{r_{fuel}}{2} \right) &= -k_{gap} \frac{C_3}{r_{fuel}} \\
\left(\frac{-q'(z)}{2\pi \cdot k_{gap}} \right) &= C_3 \\
3. -k_{gap} \frac{dT_2}{dr} \Big|_{r=r_{c,i}} &= -k_{clad} \frac{dT_3}{dr} \Big|_{r=r_{c,i}} \\
-k_{gap} \frac{C_3}{r_{c,i}} &= -k_{clad} \frac{C_5}{r_{c,i}} \\
\frac{k_{gap} C_3}{k_{clad}} &= C_5
\end{aligned}$$

$$\begin{aligned}
4. -k_{clad} \frac{dT_3}{dr} \Big|_{r=r_{c,o}} &= h(T_3(r_{c,o}) - T_f) \\
-k_{clad} \frac{C_5}{r_{c,0}} &= h(C_5 \ln(r_{c,o}) + C_6 - T_f) \\
C_5 \left(\frac{-k_{clad}}{h \cdot r_{c,0}} - \ln(r_{c,o}) \right) + T_f &= C_6 \\
5. T_2(r_{c,i}) &= T_3(r_{c,i}) \\
C_3 \ln(r_{c,i}) + C_4 &= C_5 \ln(r_{c,i}) + C_6 \\
C_3 \ln(r_{c,i}) \left(\frac{k_{gap}}{k_{clad}} - 1 \right) + C_6 &= C_4 \\
6. T_1(r_{fuel}) &= T_2(r_{fuel}) \\
-\frac{q'(z)}{\pi r_{fuel}^2 \cdot k_{fuel}} \frac{r_{fuel}^2}{4} + C_2 &= C_3 \ln(r_{fuel}) + C_4 \\
C_2 &= C_3 \ln(r_{fuel}) + C_4 + \frac{q'(z)}{4\pi \cdot k_{fuel}} \\
C_1 &= 0 \tag{3} \\
C_3 &= \frac{-q'(z)}{2\pi \cdot k_{gap}} \tag{4} \\
C_5 &= \frac{k_{gap} C_3}{k_{clad}} \tag{5} \\
C_6 &= C_5 \left(\frac{-k_{clad}}{h \cdot r_{c,0}} - \ln(r_{c,o}) \right) + T_f \tag{6} \\
C_4 &= C_3 \ln(r_{c,i}) \left(\frac{k_{gap}}{k_{clad}} - 1 \right) + C_6 \tag{7} \\
C_2 &= C_3 \ln(r_{fuel}) + C_4 + \frac{q'(z)}{4\pi \cdot k_{fuel}} \tag{8}
\end{aligned}$$

3 Results

This first plot, Fig 1 shows the equilibrium quality throughout the channel for the fluid, with the step size decreasing allowing for finer lines and a more accurate solution. The final Δz of 0.02 m was chosen for all of the parameters calculated for this homework. Once it was selected, the converged solution for the equilibrium quality was given in Fig. 2.

The next measurements included temperatures in different parts of the system as a function of z ; Fig. 3 includes the center line temperature, wall and fluid temperature and the saturation temperature (based on the pressure).

This enlarged plot, in Fig. 4, shows a clear, albeit small, difference between the fluid temperature and the wall temperature. The difference makes intuitive

sense as the heat will flow from hot to cold and the fluid is there as a coolant so the wall should be hotter than the coolant. Both lines are very similar due to the effective heat transfer through convection and do not reach T_{sat} which implies that there is only 1ϕ flow and heat transfer.

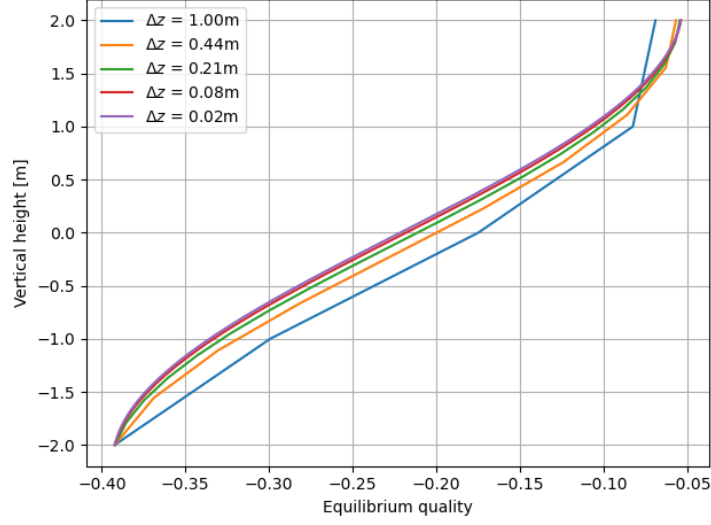


Fig. 1: χ_e convergence based on Δz where smaller step sizes are tested to find a good balance between computational intensity and accuracy

The final plot is pressure versus the change in height in the channel. This was done assuming constant properties for density throughout the channel and calculating the change in pressure pretty easily. Fig. 5 gives this relationship, assuming the density doesn't change from the inlet and neither does the friction factor (both used to calculate the change in pressure for a given Δz).

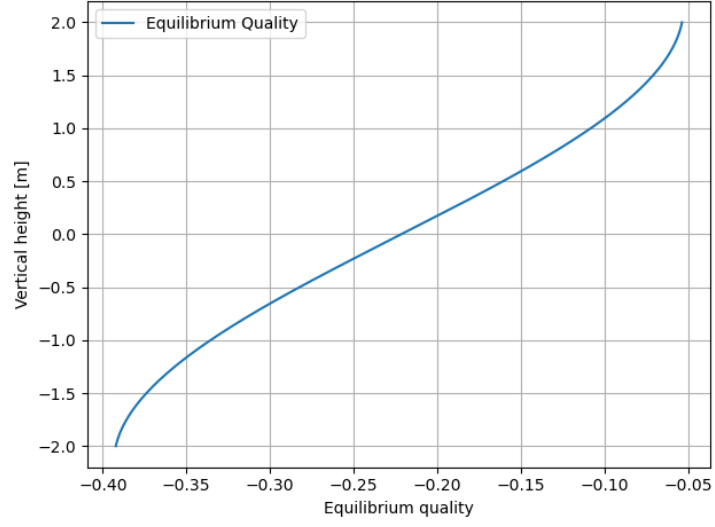


Fig. 2: χ_e , equilibrium quality, as the height in the channel increases

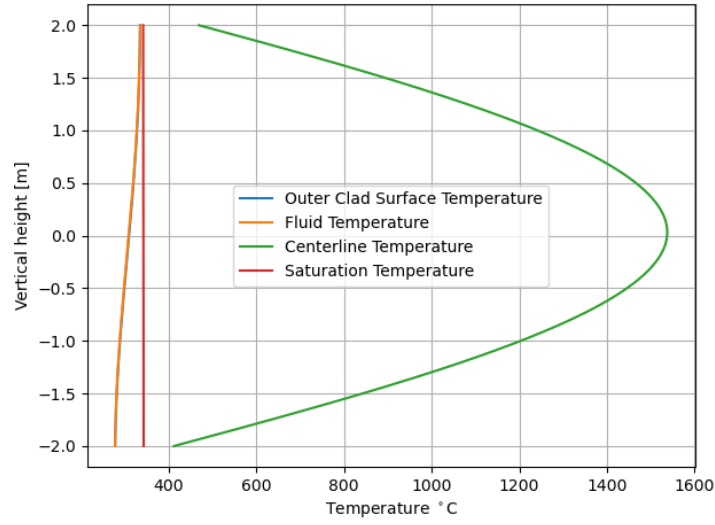


Fig. 3: Temperatures of different parts of the system as a function of the height of the channel, the overlap between fluid and wall is enlarged in Fig. 4

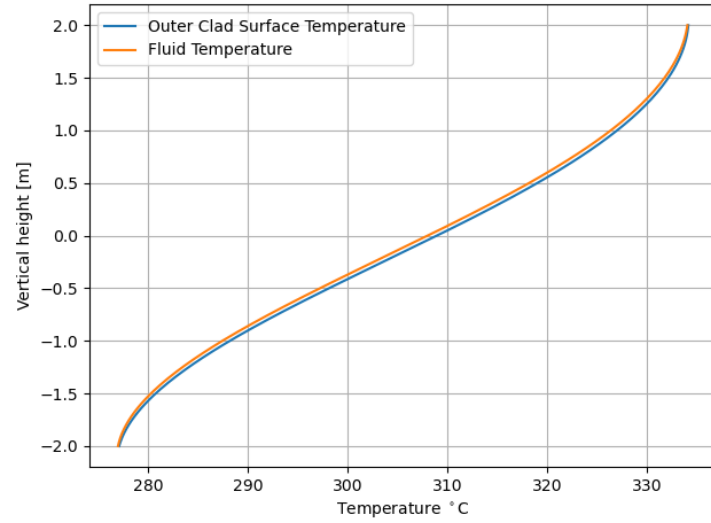


Fig. 4: Temperatures of fluid and wall as a function of the height of the channel

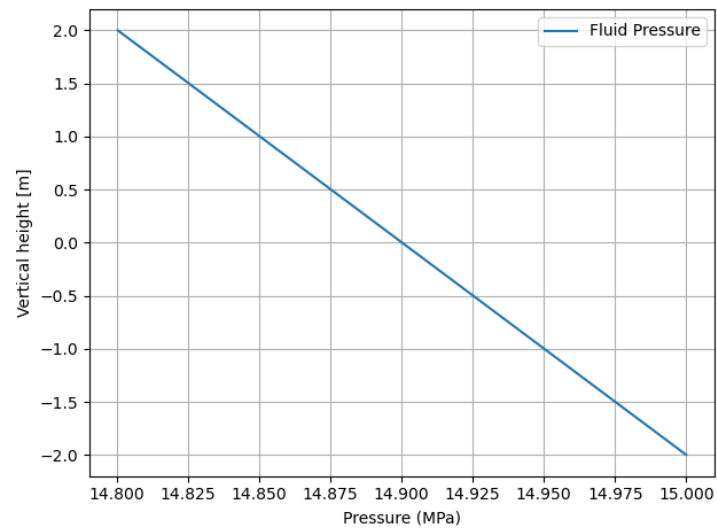


Fig. 5: Fluid pressure of the system as a function of the height of the channel, the range ends up being [14.8 - 15 MPa]

4 Code

Also available at Arnav Goyal's GitHub

The plotting lines commented out at the bottom can be changed at the users discretion to generate plots of their choosing.

```
import numpy as np
import matplotlib.pyplot as plt
from pyfluids import Fluid, FluidsList, Input

###variables###

H = 4 # m
He = 4.3 # m - extrapolated height
D_rod = 0.95 * 0.01 #m diameter
pitch = 1.26 * 0.01 #m distance between fuel pellets
D_fuel = 0.82 * 0.01 #m diameter
gap_thick = 0.006 * 0.01 #m space between fuel and clad
k_gap = 0.25 # W / m degC
k_fuel = 3.6 # W / m degC
k_clad = 21.5 # W / m degC

r_fuel = D_fuel/2
r_clad_i = r_fuel + gap_thick
r_clad_o = D_rod/2

G = 4000 #kg/ m2 s or density times velocity
q0 = 380*100 # W/m linear initial heat generation
P_low = 15*(10**6) #MPa at z=-H/2
T_f_low = 277 #degC at z=-H/2
g = 9.8 #m/s2 acceleration due to gravity

wet_perim = np.pi * D_rod #m
area = pitch**2 - (np.pi*((D_rod/2)**2))
D_equiv = 4*area / wet_perim

water = Fluid(FluidsList.Water).unspecify_phase()
#####

def tempQualProp(temp, qual, prop):
    """
    Parameters
    -----
    temp: int
        Degrees celsius
    qual: int
```

```

        Number between 0 and 100 for flow quality
prop: str
    one of the properties in the following list["vol", "intEnrg", "dynVisc", "enth", "rho"]
"""
water_state = water.with_state(Input.temperature(temp), Input.quality(qual))
if prop == "vol":
    return water_state.specific_volume #m3/kg
elif prop == "intEnrg":
    return water_state.internal_energy #J/kg
elif prop == "dynVisc":
    return water_state.dynamic_viscosity #Pa*s
elif prop == "enth":
    return water_state.enthalpy #J/kg
elif prop == "rho":
    return water_state.density #kg / m3

def presQualProp(pres, qual, prop):
    """
    Parameters
    -----
    pres: int
        pressure in pascals
    qual: int
        Number between 0 and 100 for flow quality
    prop: str
        one of the properties in the following list["vol", "intEnrg", "dynVisc", "enth", "rho"]
    """
    water_state = water.with_state(Input.pressure(pres), Input.quality(qual))

    if prop == "vol":
        return water_state.specific_volume #m3/kg
    elif prop == "intEnrg":
        return water_state.internal_energy #J/kg
    elif prop == "dynVisc":
        return water_state.dynamic_viscosity #Pa*s
    elif prop == "enth":
        return water_state.enthalpy # J/kg
    elif prop == "rho":
        return water_state.density #kg / m3

def tempPresProp(temp, pres, prop):
    """
    Parameters
    -----
    pres: int
        pressure in pascals

```

```

temp: int
    Degrees celsius
prop: str
    one of the properties in the following list["vol", "intEnrg", "dynVisc", "enth", "rho"]
"""
water_state = water.with_state(Input.temperature(temp), Input.pressure(pres))

if prop == "vol":
    return water_state.specific_volume #m3/kg
elif prop == "intEnrg":
    return water_state.internal_energy #J/kg
elif prop == "dynVisc":
    return water_state.dynamic_viscosity #Pa*s
elif prop == "enth":
    return water_state.enthalpy # J/kg
elif prop == "rho":
    return water_state.density #kg / m3

def calchfg(temp=0, pres=0):
    if temp != 0:
        return tempQualProp(temp, 100, "enth") - tempQualProp(temp, 0, "enth") #J/kg
    else:
        return presQualProp(pres, 100, "enth") - presQualProp(pres, 0, "enth") #J/kg

def reynolds(visc):
    return (G*D_equiv/visc) # Unitless

def frictionfactor(re):
    return 0.316 * re**(-0.25)

def deltaP(rho, f, drho):
    var = 0.5*f*(G**2/rho)*(wet_perim/area) + rho*g
    if drho != 0:
        return -1*(var + (G**2)/drho)
    else:
        return -1*var

def calcXe(pres, temp):
    h = tempPresProp(pres=pres, temp=temp, prop="enth")
    hfg = calchfg(pres = pres)
    hf = presQualProp(pres=pres, qual=0, prop="enth")
    #print(h, hfg, hf, type(hf))
    return (h-hf)/hfg

def calcNewTemp(Xe, pressure):
    hf = presQualProp(pres=pressure, qual=0, prop="enth")

```

```

hfg = calchfg(pres=pressure)
#print(Xe)
#print(Xe*hfg)
h = Xe*hfg + hf
#print(Xe, h, hf, hfg)

corresTemp = water.with_state(Input.enthalpy(h), Input.pressure(pressure)).temperature

return corresTemp, h # degrees Celsius

#####

#####

option = 1 # 1 : PWR, 2: BWR

fig,ax = plt.subplots()

if option == 1 :

    for numOfPoints in [200]:#[5, 10, 20, 50,200]:

        #####

        Ps = [P_low]
        T_f_s = [T_f_low]
        Xes = [calcXe(P_low, T_f_low)]
        #print(Xes[0], calcNewTemp(Xes[0], Ps[0]))

        #print(f"P_i = {P_low}, T_f_i = {T_f_low}, Xe_i={Xes[0]}, hfg_i = {calchfg(pres=Ps[0])}")

        rhos = [tempPresProp(T_f_low, P_low, "rho")]
        mus = [tempPresProp(T_f_low, P_low, "dynVisc")]
        #print(f"rho_i = {rhos[0]}, mu_i = {mus[0]}")

        Res = [reynolds(mus[0])]
        frics = [frictionfactor(Res[0])]

        hs = [tempPresProp(T_f_low, P_low, "enth")]

        #####

        zs, deltaz = np.linspace(-H/2, H/2, numOfPoints, retstep=True)

```

```

qlins = q0*np.cos(np.pi*(zs/He)) # W / m
qdoubles = qlins/(np.pi*D_fuel)
#print(qlins[:10])

for i in range(1, len(zs)):
    #print(i)

    '''
    if i != 1 :
        drho = (rhos[i-1]-rhos[i-2])/deltaz
    else:
        drho = 0

    print(f"drho = {drho}")
    '''

    deltaPCurrent = deltaP(rhos[0], frics[0], 0) #delta P / delta z
    #print(f"delta P = {deltaPCurrent}")

    newP = deltaPCurrent*deltaz + Ps[i-1]
    #print(f"P = {newP}")
    Ps.append(newP)

    newhfg = calchfg(pres=Ps[i])
    #print(f"newhfg = {newhfg}")
    qdouble = qdoubles[i]

    #print(qdouble, qdouble*wet_perim/G*area)
    newXe = ((qdouble*wet_perim*deltaz)/(area*G*newhfg))+Xes[i-1]

    #print(f"hfg = {newhfg}, Xe = {newXe}")
    Xes.append(newXe)

    newT_f, newh = calcNewTemp(Xes[i], pressure=Ps[i])
    T_f_s.append(newT_f)
    hs.append(newh)

    #rhos.append(tempPresProp(pres=Ps[i], temp=T_f_s[i], prop="rho"))
    #print(rhos[i])
    #mus.append(tempPresProp(pres=Ps[i], temp=T_f_s[i], prop="dynVisc"))
    #Res.append(reynolds(mus[i]))
    #frics.append(frictionfactor(Res[i]))

```

```

        #print(f"P_{i} = {Ps[i]}, T_f_{i} = {T_f_s[i]}, Xe_{i}={Xes[i]}")
        #print(f"rho_{i} = {rhos[i]}, mu_{i} = {mus[i]}")

#####
###Temperature inside the fuel rod#####

T_c_in = []
T_c_out = []
T_f_c = []

for j in range(len(zs)):
    curh = hs[j]
    curT_f = T_f_s[j]
    qlin = qlins[j]

    c_3 = (-qlin)/(2*np.pi*k_gap)
    c_5 = k_gap * c_3 / k_clad
    c_6 = (c_5*((-k_clad/(curh*r_clad_o)) - np.log(r_clad_o))) + curT_f
    c_4 = (c_3 *np.log(r_clad_i)*((k_gap/k_clad)-1))+c_6
    c_2 = (c_3*np.log(r_fuel)) + c_4 + (qlin/(4*np.pi*k_fuel))

    T_f_c.append(c_2)

    T_c_in.append(c_3*np.log(r_clad_i) + c_4)
    T_c_out.append(c_5*np.log(r_clad_o) + c_6)

Tsats = []
for press in Ps:
    Tsats.append(water.dew_point_at_pressure(press).temperature)

#ax.plot(qlins, zs, label="q\'")
#ax.plot(np.array(Ps)/(10**6), zs, label="Fluid Pressure")

#ax.plot(Tsats, zs, label="Saturation Temperature")
ax.plot(T_f_s, zs, label="Fluid Temperature")
#ax.plot(T_f_c, zs, label="Centerline Temperature")
#ax.plot(T_c_in, zs, label="Inner Clad Surface Temperature")
ax.plot(T_c_out, zs, label="Outer Clad Surface Temperature")

#ax.plot(Xes, zs, label=f"Equilibrium Quality - $\Delta z$ = {deltaz:0.2f}m")

#ax.plot(rhos, zs, label="Densities")
#ax.plot(mus, zs, label="Dynamic Viscosities")

```

```
#ax.plot(Res, zs, label="reynolds")
#ax.plot(frics, zs, label = "frics")

#ax.set_xscale('log')
ax.set_ylabel(f"Vertical height [m]")
ax.set_xlabel(f"Equilibrium quality")
ax.grid()
ax.legend()
plt.tight_layout()

plt.show()
```