NPRE 449: Homework 9

Mini-CP

University of Illinois at Urbana-Champaign

Department of Nuclear, Plasma, and Radiological Engineering

Author

Joseph Specht

jspecht3

Professor and Teaching Assistants

Caleb Brooks

Muhammad Sohaib Malik

November 22, 2024

Contents

1 Fuel Pin Governing Equations **1**

1.1 General Heat Diffusion Equation 1

1.2 Fuel Temperature 2

1.3 Gap Temperature 3

1.4 Cladding Temperature 3

1.5 Boundary Conditions 3

2 Heat Flux **6**

2.1 Derivation 6

2.2 Heat Transfer 7

3 Fluid Continuity Equations **8**

3.1 Mass Balance 8

3.2 Momentum Balance 8

3.3 Energy/Enthalpy Balance 10

4 Equilibrium Quality **10**

4.1 Derivation 10

4.2 Fluid Temperature 11

5 Numerical Discretization **12**

6 Results **12**

7 Appendix **14**

1 Fuel Pin Governing Equations

1.1 General Heat Diffusion Equation

For this problem, the fuel pin is split into three concentric rings: the fuel, the gap, and the cladding. For the gap region the thermal conductivity and not the convective heat transfer coefficient is given. Therefore, the governing equation for all three regions is given by the general heat diffusion equation. The general heat diffusion equation is given as:

$$\nabla \cdot k\nabla T + q''' = \rho c_p \frac{\partial T}{\partial t} \tag{1}$$

As this problem is steady state, the partial derivative with respect to time is equal to 0.

$$\nabla \cdot k\nabla T + q''' = 0 \tag{2}$$

Additionally, the changes in fuel, gap, and clad properties with temperature are negligible. Therefore, the thermal conductivity k can be pulled outside the ∇.

$$k\nabla^2 T + q''' = 0 \tag{3}$$

Rearranging.

$$\nabla^2 T = -\frac{q'''}{k} \tag{4}$$

Expanding.

$$\frac{1}{r}\frac{\partial}{\partial r} r \frac{\partial T}{\partial r} + \frac{1}{r^2}\frac{\partial^2 T}{\partial \theta^2} + \frac{\partial^2 T}{\partial^2 z} = -\frac{q'''}{k} \tag{5}$$

Assume negligible axial conduction and the temperature profile is only a function of the radius r, which cancels each partial derivative that is not with respect to r.

$$\frac{1}{r}\frac{\partial}{\partial r} r \frac{\partial T}{\partial r} = -\frac{q'''}{k} \tag{6}$$

Eq. 6 is the general solution for the temperature profile in a region with the assumptions previously mentioned applied. However, each region has specific considerations that need to be made.

1.2 Fuel Temperature

For the fuel temperature, the linear heat generation rate is given as:

$$q'(z) = q'_0 \sin\left(\frac{z\pi}{H}\right) \tag{7}$$

The linear and volumetric heat generation rates can be equated using conservation of energy.

$$\dot{E}_{gen} = \dot{E}_{gen} \tag{8a}$$

$$q'''V = q'L \tag{8b}$$

$$q'''AL = q'L \tag{8c}$$

$$q''' = \frac{q'}{A} \tag{8d}$$

$$q''' = \frac{q'}{\left(\frac{\pi}{4}D_f^2\right)} \tag{8e}$$

$$q''' = \frac{4q'}{\pi D_f^2} \tag{8f}$$

$$q''' = \frac{4q'_0}{\pi D_f^2}\sin\left(\frac{z\pi}{H}\right) = \frac{q'_0}{A}\sin\left(\frac{z\pi}{H}\right) \tag{8g}$$

Now, the volumetric heat generation is seen to not be a function of r, so the fuel temperature can be integrated with respect to r without concern:

$$\frac{1}{r}\frac{\partial}{\partial r} r \frac{\partial T_f}{\partial r} = -\frac{q'''}{k_f} \tag{9a}$$

$$\frac{\partial}{\partial r} r \frac{\partial T_f}{\partial r} = -\frac{q''' r}{k_f} \tag{9b}$$

$$r\frac{\partial T_f}{\partial r} = -\frac{q''' r^2}{2k_f} + C_1 \tag{9c}$$

$$\frac{\partial T_f}{\partial r} = -\frac{q''' r}{2k_f} + \frac{C_1}{r} \tag{9d}$$

$$T_f = -\frac{q''' r^2}{4k_f} + C_1 \ln(r) + C_2 \tag{9e}$$

1.3 Gap Temperature

Applying Eq. 6 to the cladding, the volumetric heat generation cancels out.

$$\frac{1}{r}\frac{\partial}{\partial r}r\frac{\partial T_g}{\partial r} = 0 \tag{10}$$

Solving for the temperature in the gap.

$$\frac{1}{r}\frac{\partial}{\partial r}r\frac{\partial T_g}{\partial r} = 0 \tag{11a}$$

$$\frac{\partial}{\partial r}r\frac{\partial T_g}{\partial r} = 0 \tag{11b}$$

$$r\frac{\partial T_g}{\partial r} = C_3 \tag{11c}$$

$$\frac{\partial T_g}{\partial r} = \frac{C_3}{r} \tag{11d}$$

$$T_g = C_3 \ln(r) + C_4 \tag{11e}$$

1.4 Cladding Temperature

The temperature distribution in the cladding follows the same function form as the temperature distribution in the gap as both regions have no volumetric heat generation.

$$T_c = C_5 \ln(r) + C_6 \tag{12}$$

1.5 Boundary Conditions

With the temperature profiles in each region:

$$T_f = -\frac{q'''r^2}{4k_f} + C_1 \ln(r) + C_2, \quad 0 \le r \le R_{fuel} \tag{13a}$$

$$T_g = C_3 \ln(r) + C_4, \quad R_{fuel} < r \le R_{fuel} + t_{gap} \tag{13b}$$

$$T_c = C_5 \ln(r) + C_6, \quad R_{fuel} + t_{gap} < r \le R_{rod} \tag{13c}$$

and the six boundary conditions for the six unknown variables:

$$i) \quad T_f(r=0) \neq \infty \tag{14a}$$

$$ii) \quad T_c(r=R_{rod}, z) = \frac{q''(z)}{h} + T_{fluid}(z) \tag{14b}$$

$$iii) \quad T_f(r=R_{fuel}) = T_g(r=R_{fuel}) \tag{14c}$$

$$iv) \quad T_g(r=R_{fuel}+t_{gap}) = T_c(r=R_{fuel}+t_{gap}) \tag{14d}$$

$$v) \quad q''(r=R_{fuel}) = -k_f \frac{dT_f}{dr} = \bigg|_{r=R_{fuel}} = -k_g \frac{dT_g}{dr}\bigg|_{r=R_{fuel}} \tag{14e}$$

$$vi) \quad q''(r=R_{fuel}) = -k_g \frac{dT_g}{dr} = \bigg|_{r=R_{fuel}+t_{gap}} = -k_c \frac{dT_c}{dr}\bigg|_{r=R_{fuel}+t_{gap}} \tag{14f}$$

With the boundary conditions and temperature profiles in each region, the solution for each can be found. The first boundary condition is applied:

$$T_f(r=0) = -\frac{q'''0}{4k_f} + C_1 \ln(0) + C_2 \neq \infty \tag{15a}$$

$$0 + C_1\infty + C_2 \neq \infty \tag{15b}$$

$$C_1 = 0 \tag{15c}$$

$$T_f = -\frac{q'''r^2}{4k_f} + C_2 \tag{15d}$$

Next, the second boundary condition is applied:

$$T_c(r=R_{rod}, z=0) = T_{fluid}(z) = C_5 \ln(R_{rod}) + C_6 \tag{16}$$

Currently, nothing can be done with the second boundary condition as there are two unknowns. Applying the third boundary condition:

$$-\frac{q'''R_{fuel}^2}{4k_f} + C_2 = C_3 \ln(R_{fuel}) + C_4, \tag{17}$$

Again, nothing can be done with the third boundary condition. Applying the fourth boundary condition:

$$C_3 \ln(R_{fuel}+t_{gap}) + C_4 = C_5 \ln(R_{fuel}+t_{gap}) + C_6 \tag{18}$$

The fifth boundary condition:

$$-k_f \frac{d}{dr}\left(\frac{q'''r^2}{4k_f} + C_2\right)\bigg|_{r=R_{fuel}} = -k_g \frac{d}{dr}\left(C_3 \ln(r) + C_4\right)\bigg|_{r=R_{fuel}} \tag{19a}$$

$$-k_f \left(\frac{q'''r}{2k_f}\right)\bigg|_{r=R_{fuel}} = -k_g \left(\frac{C_3}{r}\right)\bigg|_{r=R_{fuel}} \tag{19b}$$

$$-\frac{q''' R_{fuel}}{2} = \frac{k_g C_3}{R_{fuel}} \tag{19c}$$

$$C_3 = -\frac{q''' R_{fuel}^2}{2k_g} \tag{19d}$$

The sixth boundary condition:

$$-k_g \frac{d}{dr}\left(C_3 \ln(r) + C_4\right)\bigg|_{r=R_{fuel}+t_{gap}} = -k_f \frac{d}{dr}\left(C_5 \ln(r) + C_6\right)\bigg|_{r=R_{fuel}+t_{gap}} \tag{20a}$$

$$-k_g \left(\frac{C_3}{r}\right)\bigg|_{r=R_{fuel}+t_{gap}} = -k_f \left(\frac{C_5}{r}\right)\bigg|_{r=R_{fuel}+t_{gap}} \tag{20b}$$

$$\frac{k_g C_3}{R_{fuel}+t_{gap}} = \frac{k_f C_5}{R_{fuel}+t_{gap}} \tag{20c}$$

$$k_g C_3 = k_f C_5 \tag{20d}$$

With the equations for the unknown constant formulated, each constant is given as follows:

$$C_1 = 0 \tag{21a}$$

$$C_3 = -\frac{q''' R_{fuel}^2}{2k_g} \tag{21b}$$

$$C_5 = \frac{k_g}{k_c} C_3 \tag{21c}$$

$$C_6 = T_{fluid}(z) - C_5 \ln\left(R_{rod}\right) \tag{21d}$$

$$C_4 = C_5 \ln\left(R_{fuel} + t_{gap}\right) - C_3 \ln\left(R_{fuel} + t_{gap}\right) + C_6 \tag{21e}$$

$$C_2 = \frac{q''' R_{fuel}^2}{4k_f} + C_3 \ln\left(R_{fuel}\right) + C_4 \tag{21f}$$

Now, each constant is given in terms of the knowns of the problem. This means the temperature profile at the inlet in the solid regions can be calculated. Again, each region is given by the following equations:

$$T_f = -\frac{q'''r^2}{4k_f} + C_2, \quad 0 \le r \le R_{fuel} \tag{22a}$$

$$T_g = C_3 \ln(r) + C_4, \quad R_{fuel} < r \leq R_{fuel} + t_{gap} \tag{22b}$$

$$T_c = C_5 \ln(r) + C_6, \quad R_{fuel} + t_{gap} < r \leq R_{rod} \tag{22c}$$

2 Heat Flux

2.1 Derivation

To determine the heat flux as a function of z, an energy balance is performed over a differential height dh of the fuel element. The heat flux is assumed to be uniform in angle and only a function of r and z. However, there is no axial conduction, so separation of variables can be applied to the heat flux. With the aforementioned assumptions, an energy balance can be performed on a a differential height dh of the fuel rod:

$$\dot{E}_{stored} = \dot{E}_{in} - \dot{E}_{out} + \dot{E}_{gen} \tag{23}$$

However, there this problem is steady-state and heat is only flowing out of the fuel rod:

$$\dot{E}_{gen} = \dot{E}_{out} \tag{24}$$

In all subsequent equations, the heat flux is evaluated at the clad outer surface. Therefore, the energy out will be over the surface area of the cladding while the energy generation occurs in the fuel section:

$$q'''V_f = q''A_c \tag{25a}$$

$$q'''A_f L = q''\xi_h L \tag{25b}$$

$$\frac{q'}{A_f} A_f = q''\xi_h \tag{25c}$$

$$q' = q''\xi_h \tag{25d}$$

$$q'' = \frac{q'}{\xi_h} \tag{25e}$$

$$q'' = \frac{q'_0}{\xi_h} \sin\left(\frac{\pi z}{H}\right) \tag{25f}$$

2.2 Heat Transfer

The one-phase heat transfer into a bulk fluid is given by Newton's Law:

$$q'' = h\left(T_w - T_{fluid}\right) \tag{26}$$

However, the temperature of the wall and the fluid are the equivalent at the clad outer surface. If the wall temperature is higher than the fluid saturation temperature, the resultant heat transfer will be two-phase. In two-phase heat transfer, Newton's law does not apply. Instead, the homogeneous equilibrium model for heat transfer is applied, which is given as:

$$q'' = \left\{\left[Fh_{FC}\left(T_w - T_{fluid}\right)\right]^2 + \left[Sh_{NB}\left(T_w - T_{sat}\right)\right]^2\right\}^{\frac{1}{2}} \tag{27a}$$

$$F = \left[1 + \chi Pr\left(\frac{\rho_f}{\rho_g} - 1\right)\right]^{0.35} \tag{27b}$$

$$S = \left(1 + 0.55F^{0.1}Re^{0.16}\right)^{-1} \tag{27c}$$

$$h_{FC} = 0.023\left(\frac{k_f}{D_h}\right)Re^{0.8}Pr^{0.4} \tag{27d}$$

$$h_{NB} = 55\left(\frac{P}{P_c}\right)^{0.12} q''^{2/3}\left(-\frac{P}{P_c}\right)^{-0.55} M_w^{-0.5} \tag{27e}$$

where M_w is the molecular weight of water, which is 18, and P_c is the thermodynamic critical pressure. The heat transfer in the two-phase region is the contribution of heat transfer from forced convection (via h_{FC}) and heat transfer from nuclear boiling (via h_{NB}).

As the heat flux is known, the formulations for the heat flux in each region can be solved for the fluid temperature. In one-phase heat transfer:

$$T_{fluid} = \frac{q''}{h} - T_w \tag{28}$$

In two-phase heat transfer:

$$T_{fluid} = T_w - \frac{1}{Fh_{fc}}\left\{q''^2 - \left[Sh_{NB}\left(T_w - T_{sat}\right)\right]^2\right\} \tag{29}$$

With these equations, the fluid temperature can be calculated in both one-phase and two-phase heat transfer at the clad outer surface.

3 Fluid Continuity Equations

3.1 Mass Balance

The general mass conservation equation for a fluid is:

$$\frac{\partial}{\partial t}\rho + \frac{\partial}{\partial z}\rho v = 0 \tag{30}$$

However, this problem is steady state, so any partial derivative with respect to time cancels out. We can also define the mass flux as ρv, which means the general mass conservation equation simplifies down to:

$$\frac{\partial}{\partial z}\rho v = 0 \tag{31a}$$

$$\frac{\partial}{\partial z}G = 0 \tag{31b}$$

$$G = constant \tag{31c}$$

Therefore, the mass flux is constant over the entire channel, which is helpful for future equations.

3.2 Momentum Balance

The general momentum balance is:

$$\frac{\partial}{\partial t}\rho v + \frac{\partial}{\partial z}\rho v^2 = -\frac{\partial P}{\partial z} - \frac{\tau_f \xi_w}{A_{fluid}} - \rho g \sin(\theta) \tag{32}$$

This problem is steady-state, therefore any partial derivative with respect to time cancels out.

$$\frac{\partial}{\partial z}\rho v^2 = -\frac{\partial P}{\partial z} - \frac{\tau_f \xi_w}{A_{fluid}} - \rho g \sin(\theta) \tag{33}$$

By rearranging the equation above, stating the channel is vertical with $\theta = 90°$, and applying a simplification for the turbulent sheer stress τ_F:

$$\tau_f = \frac{f \rho v^2}{2} = \frac{f G^2}{2\rho} \tag{34}$$

The momentum conservation equation becomes:

$$-\frac{\partial P}{\partial z} = G^2 \frac{\partial}{\partial z}\frac{1}{\rho} + \frac{f G^2 \xi_w}{2\rho A_{fluid}} + \rho g \tag{35}$$

From this equation, the pressure drop as a function of z is obtained along the channel. The friction factor f is also given by the following equation:

$$f = 0.316Re^{-\frac{1}{4}} \tag{36}$$

However, in Eq. 35, there is no cogent notion of ρ in two-phase flow. Therefore, ρ will be represented as ρ_m, an approximation of the two-phase density. First, the relation between the specific volume and density is known:

$$\frac{1}{\rho_m} = \nu_m \tag{37}$$

The formulation for ν_m is also given as:

$$\nu_m = \tag{38a}$$

$$\chi\nu_g + (1-\chi)\nu_f = \tag{38b}$$

$$\nu_f + (\nu_g - \nu_f)\,\chi = \tag{38c}$$

$$\nu_f + \chi\nu_{fg} \tag{38d}$$

Using these equations, the relation between the density can be represented in terms of known specific volumes and the vapor quality:

$$\rho_m = \frac{1}{\nu_m} = \frac{1}{\nu_f + \chi\nu_{fg}} \tag{39}$$

Now, the equation for pressure drop (Eq. 35) can be written in terms of the vapor quality:

$$-\frac{\partial P}{\partial z} = G^2 \frac{\partial}{\partial z}(\nu_f + \chi\nu_{fg}) + \frac{fG^2\xi_w}{2A_{fluid}}(\nu_f + \chi\nu_{fg}) + \frac{g}{(\nu_f + \chi\nu_{fg})} \tag{40}$$

As the values for the specific volume of the fluid are at the flow entrance, the specific volumes are not a function of z. As such, the derivatives of the specific volumes with respect to z are 0.

$$-\frac{\partial P}{\partial z} = G^2\nu_{fg}\frac{\partial \chi}{\partial z} + \frac{fG^2\xi_w}{2A_{fluid}}(\nu_f + \chi\nu_{fg}) + \frac{g}{(\nu_f + \chi\nu_{fg})} \tag{41}$$

3.3 Energy/Enthalpy Balance

The general form of the energy balance equation is:

$$\frac{\partial}{\partial t}\rho h + \frac{\partial}{\partial z}\rho v h = \frac{q''\xi_h}{A_{fluid}} + \frac{\partial P}{\partial t} + q''' \tag{42}$$

However, this problem is steady-state with no volumetric heat generation in the fluid. Therefore, the energy conservation equation becomes:

$$\frac{\partial}{\partial z}\rho v h = \frac{q''\xi_h}{A_{fluid}} \tag{43}$$

Rearranging the above equation and substituting in for G gives:

$$\frac{\partial h}{\partial z} = \frac{q''\xi_h}{GA_{fluid}} \tag{44}$$

4 Equilibrium Quality

4.1 Derivation

Although there is a formulation for h, said formulation does not account is not in a useful formulation when considering two phase flow. Therefore, the enthalpy h can be broken in terms of other variables. First, the formulation for the equilibrium steam quality χ_e is given as:

$$\chi_e = \frac{h - h_{f,sat}}{h_{g,sat} - h_{f,sat}} = \frac{h - h_{f,sat}}{h_{fg,sat}} \tag{45}$$

Solving this equation for h gives:

$$h = h_{fg,sat}\chi_e + h_{f,sat} \tag{46}$$

From Eq. 44, the above equation can be substituted in to give:

$$\frac{\partial}{\partial z}\left(h_{fg,sat}\chi_e + h_{f,sat}\right) = \frac{q''\xi_h}{GA_{fluid}} \tag{47}$$

Then, the derivative is taken of the left-hand side:

$$\chi_e\frac{\partial h_{fg,sat}}{\partial z} + \frac{\partial h_{f,sat}}{\partial z} + h_{fg,sat}\frac{\partial \chi_e}{\partial z} = \frac{q''\xi_h}{GA_{fluid}} \tag{48}$$

Assuming each variable is continuously differentiable over the interval of interest allows the expansion of the derivatives with respect to z.

$$\chi_e \frac{\partial h_{fg,sat}}{\partial P}\frac{\partial P}{\partial z} + \frac{\partial h_{f,sat}}{\partial P}\frac{\partial P}{\partial z} + h_{fg,sat}\frac{\partial \chi_e}{\partial z} = \frac{q''\xi_h}{GA_{fluid}} \tag{49}$$

This equation can be rearranged to solve for derivative of χ_e with respect to z:

$$h_{fg,sat}\frac{\partial \chi_e}{\partial P} = \frac{q''\xi_h}{GA_{fluid}} - \left(\chi_e \frac{\partial h_{fg,sat}}{\partial P}\frac{\partial P}{\partial z} + \frac{\partial h_{f,sat}}{\partial P}\frac{\partial P}{\partial z}\right) \tag{50a}$$

$$\frac{\partial \chi_e}{\partial P} = \frac{1}{h_{fg,sat}}\left[\frac{q''\xi_h}{GA_{fluid}} - \left(\chi_e \frac{\partial h_{fg,sat}}{\partial P}\frac{\partial P}{\partial z} + \frac{\partial h_{f,sat}}{\partial P}\frac{\partial P}{\partial z}\right)\right] \tag{50b}$$

$$\frac{\partial \chi_e}{\partial z} = \frac{1}{h_{fg,sat}}\left[\frac{q''\xi_h}{GA_{fluid}} - \left(\chi_e \frac{\partial h_{fg,sat}}{\partial P} + \frac{\partial h_{f,sat}}{\partial P}\right)\frac{\partial P}{\partial z}\right] \tag{50c}$$

Which gives the final form of the equation:

$$\frac{d\chi_e}{dP} = \frac{1}{h_{fg,sat}}\left[\frac{q''\xi_h}{GA_{fluid}} - \left(\chi_e \frac{dh_{fg,sat}}{dP} + \frac{dh_{f,sat}}{dP}\right)\frac{dP}{dz}\right] \tag{51}$$

4.2 Fluid Temperature

We know the equilibrium quality is given as:

$$\chi_e = \frac{h - h_{f,sat}}{h_{fg,sat}}, \tag{52}$$

which relates the enthalpy of the system to the saturation enthalpies of the gas and liquid phases. If χ_e is negative, the enthalpy of the system is lower than the saturation enthalpy of the fluid and the system is a sub-cooled liquid. If χ_e is higher than one, the enthalpy of the system is higher than the saturation enthalpy of the gas and the system is a super-heated gas.

Additionally, Eq. 52 can be represented in a different manner assuming the contribution from the product of pressure and volume are negligible:

$$h = u + PV \simeq u = c_pT \tag{53}$$

Which gives:

$$\chi_e = \frac{c_pT - c_pT_{sat}}{h_{fg,sat}} \tag{54a}$$

$$T - T_{sat} = \chi_e \frac{h_{fg,sat}}{c_p} \tag{54b}$$

$$T = \chi_e \frac{h_{fg,sat}}{c_p} + T_{sat} \tag{54c}$$

Now, the temperature of the fluid can be obtained using the equilibrium quality.

5 Numerical Discretization

Using a forward, finite differencing scheme, the derivative of a function x with respect to some variable y that is equal to a quantity f can be numerically discretized as follows:

$$\frac{dx}{dy} = f = \frac{\Delta x}{\Delta y} = \frac{x_i - x^{i-1}}{\Delta y} \tag{55a}$$

$$f\Delta y = x^i - x^{i-1} \tag{55b}$$

$$x^i = x^{i-1} + f\Delta y \tag{55c}$$

With this numerical integration scheme, any continuous first order derivative can be discretized. This equation calculates the next step in y for the function x – this allows the forward progression through a parameter space without the need for the explicit value of $\frac{dx}{dy}$.

This same logic can be applied to the equilibrium quality χ_e and the pressure P in the a fluid channel to solve for the distribution of these quantities numerically. This discitization applied to the pressure yields:

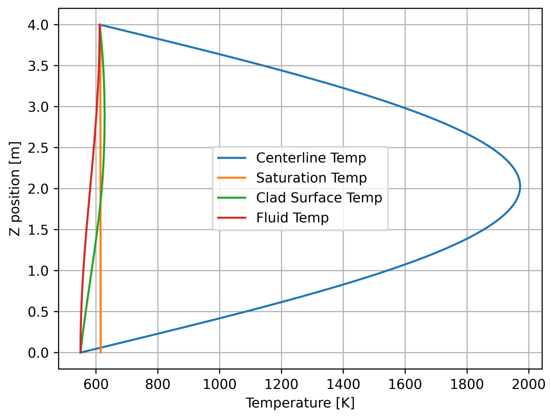
$$P^{i+1} = P^i \Delta z \left\{ \frac{\frac{\xi_h f G^2}{2A_f \rho_m} + \rho_m g + \frac{G\nu_{fg} q''(z)\xi_h}{A_f h_{fg}}}{1 - \frac{G^2 * \nu_{fg}}{h_{fg}} \left[X_e^i \frac{\partial h_g}{\partial P} + (1 - \chi_e^i) \frac{\partial h_f}{\partial P} \right]} \right\} \tag{56}$$

And the equilibrium quality:

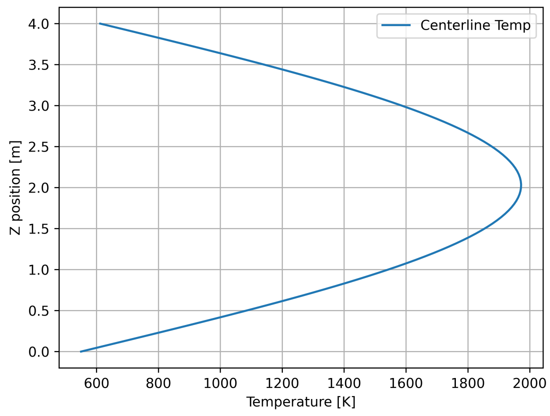
$$\chi_e^{i+1} = \chi_e^i + \Delta z \left\{ \frac{q''(z)\xi_h}{A_f G h_{fg}} - \frac{1}{h_{fg}} \left[X_e^i \frac{\partial h_g}{\partial P} \frac{\partial P}{\partial z} + \left(1 - \chi_e^i\right) \frac{\partial h_f}{\partial P} \frac{\partial P}{\partial z} \right] \right\} \tag{57}$$

6 Results

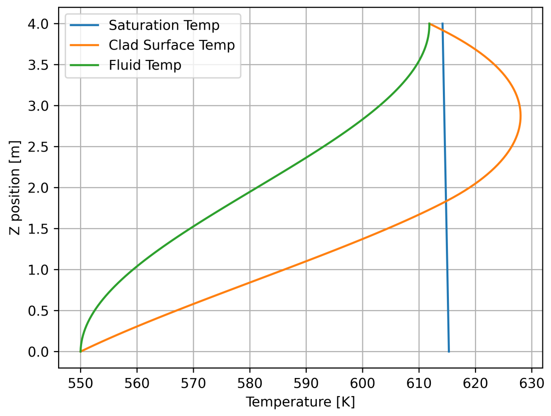
From the equations above, certain scalar fields for the pin cell were calculated and plotted as follows.



(a) All Temperatures

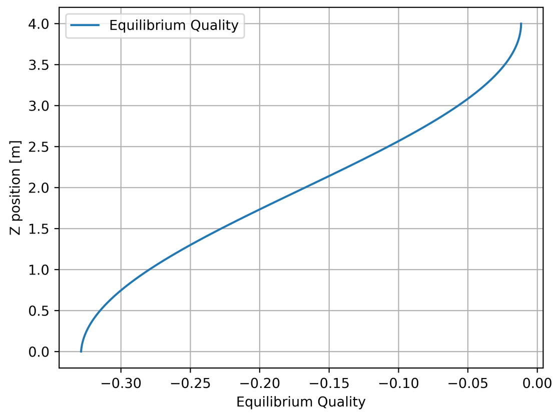


(b) Center-line Temperature

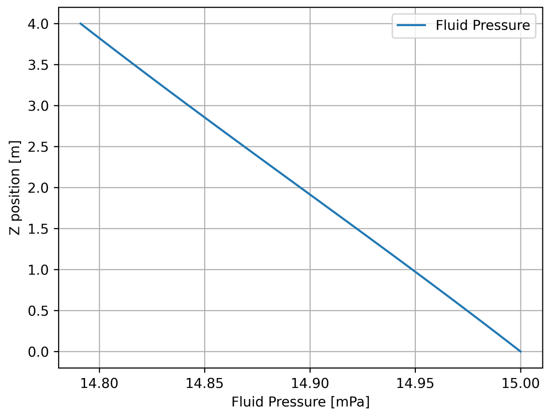


(c) Fluid and Wall Temperatures

Figure 1: Temperature Distributions



(a) Equilibrium Quality



(b) Pressure

Figure 2: Pressure and Equilibrium Quality

7 Appendix

```python
import numpy as np
import numpy.linalg as la
import scipy as sci
from scipy.optimize import fsolve
import matplotlib.pyplot as plt
from pyXSteam.XSteam import XSteam as xs

class InitialConditions:
    def __init__(self, reactor_type):
        """Defines the initial conditions for the reactor.

        Parameters
        ----------
        reactor_type: [1,2]
            defines the reactor type, 1 corresponds to a PWR and 2 corresponds to a BWR
        """

        if reactor_type == 1:
            self.__dict__ = {
                "reactor": "PWR",

                "h": 4, # m
                "d_rod": 0.95 / 100, # cm / 100 = m
                "pitch": 1.26 / 100, # cm / 100 = m
                "d_fuel": 0.82 / 100, # cm / 100 = m
                "t_gap": 0.006 / 100, # cm / 100 = m

                "k_gap": 0.25, # W / mK
                "k_fuel": 3.6, # W / mK
                "k_clad": 21.5, # W / mk

                "g": 4000, # kg / m^2 s
                "qp_0": 430 * 100, # W / cm * 100 = W / m
                "p_z0": 15, # MPa
                "tf_z0": 277 + 273, # *C => K
            }

        if reactor_type == 2:
            self.__dict__ = {
                "reactor": "BWR",

                "h": 4.1, # m
                "d_rod": 1.227 / 100, # cm / 100 = m
                "pitch": 1.62 / 100, # cm / 100 = m
                "d_fuel": 1.04 / 100, # cm / 100 = m
                "t_gap": 0.010 / 100, # cm / 100 = m

                "k_gap": 0.25, # W / mK
                "k_fuel": 3.6, # W / mK
                "k_clad": 21.5, # W / mk

                "g": 2350, # kg / m^2 s
                "qp_0": 605 * 100, # W / cm * 100 = W / m
                "p_z0": 7.5, # MPa
                "tf_z0": 272 + 273, # *C => K
            }

        if reactor_type not in [1,2]:
            raise AttributeError("----- !!! Unsupported Reactor Type !!! -----\n" +
                                "The supported reactor types are:\n" +
                                "For a PWR, reactor_type == 1\n" +
                                "For a BWR, reactor_type == 2")

class EnthalpyFunctions:
    def __init__(self):
        """You shouldn't be here"""

        # enthalphy dictionary
        enthalpies = {
            "hg": [],
            "hf": [],
            "hfg": [],
            "pressures": np.linspace(0.001,22.06,10000)
        }

        # steam lookup table
        steamtable = xs(xs.UNIT_SYSTEM_BARE) # m/kg/s/K/MPa/W

        # obtaining the enthalpies as a function of pressure for liquid, gas, and 2-phase
        for pressure in enthalpies["pressures"]:
            hg = steamtable.hV_p(pressure)
            hf = steamtable.hL_p(pressure)
            hfg = hg - hf

            enthalpies["hg"].append(hg)
            enthalpies["hf"].append(hf)
            enthalpies["hfg"].append(hfg)

        # getting polynomial fits for the enthalpy functions
        poly_deg = 25
        self.steamtable = steamtable

        ## h(P)
        hg_vs_p = np.polynomial.Polynomial.fit(
            enthalpies["pressures"], enthalpies["hg"], deg=poly_deg)
        hf_vs_p = np.polynomial.Polynomial.fit(
            enthalpies["pressures"], enthalpies["hf"], deg=poly_deg)
        hfg_vs_p = np.polynomial.Polynomial.fit(
            enthalpies["pressures"], enthalpies["hfg"], deg=poly_deg)

        ## dh/dP
        dhg_dp = hg_vs_p.deriv()
        dhf_dp = hf_vs_p.deriv()
        dhfg_dp = hfg_vs_p.deriv()

        # initializing the dict
        self.__dict__ = {
            "hg": hg_vs_p,
            "hf": hf_vs_p,
            "hfg": hfg_vs_p,

            "dhg": dhg_dp,
            "dhf": dhf_dp,
            "dhfg": dhg_dp,
        }

    def plot(self):
        x = np.linspace(0.001, 22.06, 10000)

        hg = self.hg(x)
        hf = self.hf(x)
        hfg = self.hfg(x)

        dhg = self.dhg(x)
        dhf = self.dhf(x)
        dhfg = self.dhfg(x)

        for enthalpy, title in zip([hg, dhg, hf, dhf, hfg, dhfg],
                                   ["h_g", "dh_g/dP", "h_f", "dh_f/dP", "h_fg", "dh_fg/dP"])
    :
            plt.plot(x, enthalpy)
            plt.xlabel("Pressure [MPa]")
            plt.ylabel(title)
            plt.grid('both')
            plt.savefig(f"plots/{title.replace('/', '')}", dpi=600)
            plt.show()

class Solver:
    def __init__(self, reactor_type, mesh_elements = 101):
        self.rector_type = reactor_type

        self.ics = InitialConditions(reactor_type)
        self.enthalpies = EnthalpyFunctions()
        self.st = xs(xs.UNIT_SYSTEM_BARE) # m/kg/s/K/MPa/W

        ics = self.ics
        st = self.st

        self.xi = np.pi * ics.d_rod
        self.a_fluid = ics.pitch**2 - np.pi/4 * ics.d_rod**2
        self.d_h = 4 * self.a_fluid / self.xi

        self.rho0 = st.rhoL_p(ics.p_z0)

        self.mu_f = st.my_pt(ics.p_z0, ics.tf_z0)# * self.rho0 * 1e-3 # mPa*s / 1000 = Pa*s
        self.k_fluid = st.tc_pt(ics.p_z0, ics.tf_z0)
        self.cp = st.Cp_pt(ics.p_z0, ics.tf_z0)
        self.mw = 18

        self.re = ics.g * self.d_h / self.mu_f #6 * self.d_h / self.mu_f
        self.pr = self.cp * self.mu_f / self.k_fluid * 1000
        self.nu = 0.023 * self.re**(0.8) * self.pr**(0.4)
```

```python

        self.h_f0 = self.nu / self.d_h * self.k_fluid
        self.ff = 0.316 * self.re**(-1/4)

        self.pc = 22.06 # MPa

        self.zs = np.linspace(0, self.ics.h, mesh_elements)

    def solve_fluid(self, show):
        # renaming
        ics = self.ics
        enthalpies = self.enthalpies
        st = self.st

        # initial calcs
        h = st.h_pt(ics.p_z0, ics.tf_z0)
        hf = st.hL_p(ics.p_z0)
        hg = st.hV_p(ics.p_z0)

        tf0 = self.ics.tf_z0
        tsat0 = st.tsat_p(self.ics.p_z0)

        p0 = self.ics.p_z0
        chi_e0 = self.cp * (tf0 - tsat0) / enthalpies.hfg(p0)

        # lists
        temp_fluid = [tf0]
        temp_sat = [tsat0]

        pressure = [p0]
        chi_e = [chi_e0]

        # functions
        self.heat_flux = lambda z : 1e-3 * ics.qp_0 / self.xi * np.sin(np.pi * z / ics.h) # W / kW
        self.vol_heat_gen = lambda z: 4 * ics.qp_0 / np.pi / ics.d_fuel**2 * np.sin(np.pi * z / ics.h) / 1000 # W / kW

        if show: print(f"z: {self.zs[0]} \ttf: {round(temp_fluid[0],2)}        \tp: {round(pressure[0],3)}        \tchi_e: {round(chi_e[0],3)}\tt_sat: {round(temp_sat[0],3)}")
        #z: 0.4   tf: 551.62069   p: 14.9797266593   chi_e: -0.32   t_sat: 615.2
        for i in range(1, len(self.zs)):
            # step vars
            z = self.zs[i]
            dz = z - self.zs[i-1]

            chi_e0 = chi_e[i - 1]
            tf0 = temp_fluid[i - 1]
            p0 = pressure[i - 1]

            # calcs
            qpp = self.heat_flux(z)

            # vapor quality
            if chi_e0 <= 0:
                chi = 0
                dchi = 0
            if chi_e0 > 0 and chi_e0 <= 1:
                chi = chi_e0
                chi_prev = chi_e[i - 1]
                if chi_prev < 0: chi_prev = 0
                dchi = chi - chi_prev
            if chi_e0 > 1:
                print("WARNING!!! DRYOUT HAS OCCURED") # yeay yeayesye ayea


            # densities
            nu_f = st.vL_p(p0) # maybe an issue here
            nu_g = st.vV_p(p0)
            nu_fg = nu_g - nu_f

            nu_m = nu_f + chi * nu_fg

            rho_l = st.rhoL_p(pressure[0])
            rho_g = st.rhoV_p(pressure[0])
            rho_fg = rho_l - rho_g
            rho_m = (1 / rho_l + 1/(rho_g - rho_l)*chi)**(-1)

            # enthalpies
            hf0 = st.hL_p(p0)
            hg0 = st.hV_p(p0)
            hfg0 = hg0 - hf0

            # calculating p1
            pc1 = self.xi * self.ff * ics.g**2 / 2 / self.a_fluid / rho_m #topP1
            pc2 = rho_m * 9.81 # topP2
            pc3 = ics.g * qpp * self.xi / rho_fg / self.a_fluid / hfg0
            pc4 = chi * enthalpies.dhg(p0)
            pc5 = (1 - chi) * enthalpies.dhf(p0)
            pc6 = ics.g**2 / rho_fg / hfg0 * (pc4 + pc5)
#            pc6 = 1 - ics.g**2 / (rho_fg * enthalpies.hfg(p0)) * (chi * enthalpies.dhg(p0) + (1 - chi) * enthalpies.dhf(p0))

#            botp1 = chi * enthalpies.dhg(p0) + (1 - chi) * enthalpies.dhf(p0)
#            botp2 = 1 - ics.g**2 / (rho_fg * enthalpies.hfg(p0 * 1e-6)) * botp1
#            1 - g**2 / (rhofg * hfg) * (chi_e * dhgdp + (1 - chi_e) * dhfdp)

#            if i == 1: print(f"pc1: {pc1}\npc2: {pc2}\npc3: {pc3}\npc4: {pc4}\npc5: {pc5}\npc6: {pc6}")
#            if i == 1: print(chi_e0, enthalpies.dhg(p0), enthalpies.dhf(p0), ics.g, rho_fg, enthalpies.hfg(p0))

            dp = ((pc1 + pc2 + pc3) / (1 - pc6)) * dz * 1e-3
#            print(dp * 1e3, p0 * 1e6)
            p1 = p0 + dp

            # enthalpies 2
            hf1 = st.hL_p(p1)
            hg1 = st.hV_p(p1)
            hfg1 = hg1 - hf1

            dhf = hf1 - hf0
            dhg = hg1 - hg0
            dhfg = hfg1 - hfg0

            # calculating chi_e1
            chic1 = qpp * self.xi / self.a_fluid / ics.g / hfg0
            chic2 = chi * enthalpies.dhg(p1) * dp/dz
            chic3 = (1 - chi) * enthalpies.dhf(p1) * dp/dz
            chic4 = 1 / hfg0 * (chic2 + chic3)

            dchi_e = dz * (chic1 - chic4)
            chi_e1 = chi_e0 + dchi_e

            # calculating tf1
            t_sat = st.tsat_p(p1)

            tf1 = enthalpies.hfg(p1) * chi_e1 / self.cp + st.tsat_p(p1)

            # output
            if i % 10 == 0 and show: print(f"z: {round(z,5)} \ttf: {round(tf1,5)} \tp: {
    round(p1,5)} \tchi_e: {round(chi_e1,3)} \tt_sat: {round(t_sat,3)}")
            temp_fluid.append(tf1)
            pressure.append(p1)
            chi_e.append(chi_e1)
            temp_sat.append(t_sat)

        self.temp_fluid = temp_fluid
        self.temp_sat = temp_sat

        self.pressure = pressure
        self.chi_e = chi_e
        return

    def fluid_plotter(self):
        plt.plot(self.temp_fluid, self.zs)
        plt.plot(self.temp_sat, self.zs)
        plt.title("temps"), plt.grid('both')
        plt.show()

        plt.plot(self.pressure, self.zs)
        plt.title("pressure"), plt.grid('both')
        plt.show()
        plt.title("chi"), plt.grid('both')
        plt.plot(self.chi_e, self.zs)
        plt.show()

    def fluid_interpolaters(self, show):
        deg = 5

        self.func_temp_fluid = np.polynomial.Polynomial.fit(self.zs, self.temp_fluid, deg)
        self.func_temp_sat = np.polynomial.Polynomial.fit(self.zs, self.temp_sat, deg)
        self.func_pressure = np.polynomial.Polynomial.fit(self.zs, self.pressure, deg)
        self.func_chi_e = np.polynomial.Polynomial.fit(self.zs, self.chi_e, deg)

        if show:
            functions = [self.func_temp_fluid, self.func_temp_sat, self.func_pressure, self.func_chi_e]
            discrete = [self.temp_fluid, self.temp_sat, self.pressure, self.chi_e]
            titles = ["Fluid Temp [K]", "Saturation Temp [K]", "Pressure [MPa]", "Equilibrium Quality"]

            for func, disc, title in zip(functions, discrete, titles):
                plt.plot(func(self.zs), self.zs, label='Function', c='r')
                plt.plot(disc, self.zs, label = 'Discrete Points', ls=(0,(5,5)), c='b')
                plt.xlabel(title), plt.ylabel('Z position [m]')
                plt.legend()
                plt.show()

    def solve_clad(self):
        st = self.st
        ics = self.ics

        press = lambda z: self.func_pressure(z)
        F = lambda chi_e, p: 1 if chi_e < 0 else (1 + chi_e * self.pr * (st.rhoV_p(p) / st.rhoL_p(p) - 1))**(0.35)
        S = lambda chi_e, z: (1 + 0.055 * F(chi_e, press(z))**(0.1) * (ics.g * self.d_h / self.mu_f)**(0.16))**(-1)
        hnb = lambda z: 55 * (st.rhoV_p(press(z)) / st.rhoL_p(press(z)))**(0.12) * (1000 * self.heat_flux(z))**(2/3) * (-np.log(press(z) / self.pc)) * self.mw**(-0.5)

        def one_phase(z): return 1000 * self.heat_flux(z) / self.h_f0 + self.func_temp_fluid(z)

        def two_phase(tw, z, chi_e):
            f_ = (F(chi_e, press(z)) * self.h_f0 * (tw - self.func_temp_fluid(z)))**2
            s_ = (S(chi_e,z) * hnb(z) * (tw - st.tsat_p(press(z))))**2

#             print(f_, s_)

            return 1000 * self.heat_flux(z) - (f_ + s_)**(0.5)

        tcs = []
        guess = st.tsat_p(ics.p_z0)

        for i,z in enumerate(self.zs):
            chi_e = self.func_chi_e(z)

            t_guess = one_phase(z)
            tsat = self.func_temp_sat(z)

            if t_guess < tsat:
                tcs.append(t_guess)

            if t_guess >= tsat:
                try: _ = self.onb * 2
                except: self.onb = z

                func = lambda tw: two_phase(tw, z, chi_e)
                root = sci.optimize.root(func, guess).x[0]
                tcs.append(root)

#             print(tcs[i], st.tsat_p(self.func_pressure(z)))

        try: self.sat_boil = sci.optimize.root(self.func_chi_e, self.h_f0 / 3).x[0]
        except ValueError:
            pass
        self.func_tcs = sci.interpolate.interp1d(self.zs, tcs)

        #plt.plot(self.func_tcs(self.zs), self.zs)
        #plt.plot(self.func_temp_sat(self.zs), self.zs)
        #print(np.max(tcs))
        return

    def solve_fuel(self):
        ics = self.ics

        r1 = ics.d_fuel / 2
        r2 = r1 + ics.t_gap
        r3 = ics.d_rod / 2

        fuel_region = np.linspace(0,r1,1000)
        gap_region = np.linspace(r1,r2,1000)
        clad_region =  np.linspace(r3,r3,1000)

        c3 = lambda z: - 1e3 * self.vol_heat_gen(z) * r1**2 / 2 / ics.k_gap
        c5 = lambda z: ics.k_gap / ics.k_clad * c3(z)
        c6 = lambda z: self.func_temp_fluid(z) - c5(z) * np.log(r3)
        c4 = lambda z: c5(z) * np.log(r2) - c3(z) * np.log(r2) + c6(z)
        c2 = lambda z: 1e3 * self.vol_heat_gen(z) * r1**2 / 4 / ics.k_fuel + c3(z) * np.log(
    r1) + c4(z)

        t_fuel = lambda r,z: - 1e3 * self.vol_heat_gen(z) * r**2 / 4 / ics.k_fuel + c2(z)
        t_gap = lambda r,z: c3(z) * np.log(r) + c4(z)
        t_clad = lambda r,z: c5(z) * np.log(r) + c6(z)

        t_cl = t_fuel(0, self.zs)
        self.temp_cl = np.polynomial.Polynomial.fit(self.zs, t_cl, 10)
        return

    def plotter(self, save = False):
        plt.plot(self.temp_cl(self.zs), self.zs, label = "Centerline Temp")
        plt.plot(self.func_temp_sat(self.zs), self.zs, label = "Saturation Temp")
        plt.plot(self.func_tcs(self.zs), self.zs, label = "Clad Surface Temp")
        plt.plot(self.func_temp_fluid(self.zs), self.zs, label = "Fluid Temp")
        plt.grid("both"), plt.legend(), plt.ylabel("Z position [m]")
        plt.xlabel("Temperature [K]")
        if save: plt.savefig("plots/all-temps-vs-z.png", dpi=600)
        plt.show()

        plt.plot(self.func_temp_sat(self.zs), self.zs, label = "Saturation Temp")
        plt.plot(self.func_tcs(self.zs), self.zs, label = "Clad Surface Temp")
        plt.plot(self.func_temp_fluid(self.zs), self.zs, label = "Fluid Temp")
        plt.grid("both"), plt.legend(), plt.ylabel("Z position [m]")
        plt.xlabel("Temperature [K]")
        if save: plt.savefig("plots/temps-vs-z.png", dpi=600)
        plt.show()

        plt.plot(self.temp_cl(self.zs), self.zs, label = "Centerline Temp")
        plt.grid("both"), plt.legend(), plt.ylabel("Z position [m]")
        plt.xlabel("Temperature [K]")
        if save: plt.savefig("plots/tcl-vs-z.png", dpi=600)
        plt.show()

        plt.plot(self.func_chi_e(self.zs), self.zs, label = "Equilibrium Quality")
        plt.grid("both"), plt.legend(), plt.ylabel("Z position [m]")
        plt.xlabel("Equilibrium Quality")
        if save: plt.savefig("plots/chie-vs-z.png", dpi=600)
        plt.show()

        plt.plot(self.func_pressure(self.zs), self.zs, label = "Fluid Pressure")
        plt.grid("both"), plt.legend(), plt.ylabel("Z position [m]")
        plt.xlabel("Fluid Pressure [mPa]")
        if save: plt.savefig("plots/p-vs-z.png", dpi=600)
        plt.show()


a = Solver(1)
a.solve_fluid(0)
#a.fluid_plotter()
a.fluid_interpolaters(False)
a.solve_clad()
a.solve_fuel()
a.plotter(1)
```