FORTRAN 77 Accompanying Report
Ocean Wong (Hoi Yeung Wong)
MSc Physics and Technology of Nuclear Reactors
2018-12-25

1 Formulation of the problem

A flux profile of constant spacing is provided by the user. This represents the variation of neutron flux across the entire height of the reactor.
The power (P) produced per unit volume (V) of fuel is then given as

$$\frac{P}{V} = QF \tag{1}$$

$$\frac{P}{V} = q''' = Q\Sigma_f^F \phi(z) \tag{2}$$

$$= QN_d\sigma_f\phi(z) \tag{3}$$

$$= QE\frac{N_A\,\rho}{A}\sigma_f\phi(z) \tag{4}$$

where the symbols have their usual meaning in nuclear engineering, but can also be found in Table 1. All of the above constants are computed in SI units.

At thermal equilibrium, the temperature of the coolant must monotonically increase as it travels up the cooling channels; and the rate of increase of temperature must be proportional to the power produced by that particular section fuel pin between z and $z + \Delta z$, which in turn is proportional to the flux at z. This is given by equation 5.

$$T_{cool}(z) = \int_0^z \frac{A_f q'''(z')}{\dot{m}C_p} dz' \tag{5}$$

The temperature profile inside the fuel pin can then be inferred from the coolant temperature using equation 9 onwards.

A program needs to be written; the aim of this program is to find out this vertical temperature profile for the coolant, exterior of the cladding, and the maximum temperature of the fuel (closest to the centre point), print them to screen and save to file.

1.1 simplifications and justification

The material characteristics (dimension of the reactor, Macroscopic cross-section of the fuel and other heat transfer constants) is assumed to remain constant with respect to temperature variation.

This the error contribution from this assumption is expected to be relatively small, and and using this assumption reduces the complexity of the program significantly so that the source code is more easily understood by other programmers, and it is less likely to make a mistake.

The temperature variation is found using summation instead of integrating over the interpolated flux profile. (See Figure 1 and equation 5)

Finally, as disambiguation, the flux profile provided by the user is assumed to span the entire height of the reactor, 0 to z_{max}, inclusive of the boundary, so that the size must be And therefore it must have the o

The error contribution from this simplification decreases as the number of flux values provided by the user increase (the spacing decreases).

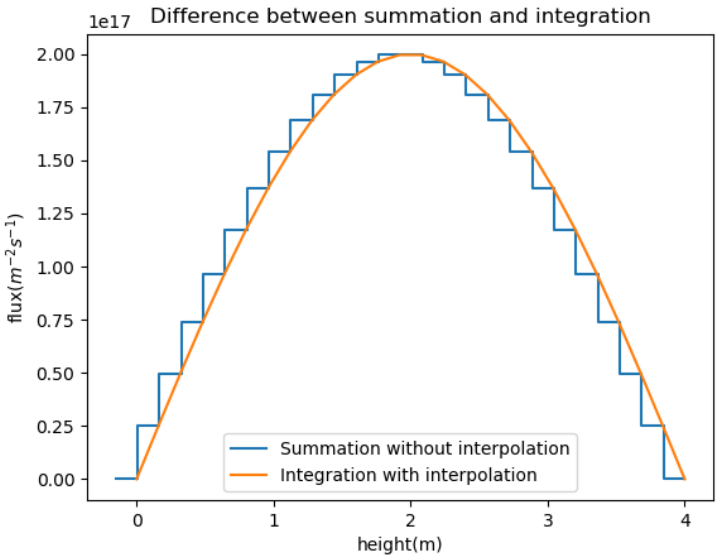

Figure 1: The summation approach given by equation 7 is identical to dividing the flux profile into steps before integrating (equation 5 to find its effect on the temperature profile.

2 Approach

The program is modularized using SUBROUTINEs and FUNCTIONs. The layout is as follows:

After printing some preliminary information about the program (authorship and purpose),

- `PrintProgramHeader()`

it prompts for an input of the file name (which has to be in the same directory as the executable .out file) from the user.

- `GetFileName(fileName)`
- `ScanFlux(fileName, length, flux, arysize)`

(The program will exit if the file can be opened but cannot be read as floating point values, such that the user can change the content of the input file and re-run the program.)
The values read from the file are stored in an array (of size 1023) in the main program.
This sets an upper limit of 1023 to the number of values that can be provided by the input file (and therefore limits the spatial resolution in vertical flux profile).

After ensuring that the file exists and is read without error, the function `Init_params` prompts for the reactor type from the user, such that the proper parameters for the reactor can be chosen.

- `Init_params(length)`
  - `GetReactorType()`
  - `AGRparameters`($z_{max}, \dot{m}, C_p, h_c, T_{in}, E, d_f, d_h, T_c, k_c, h_g$)
  - `PWRparameters`($z_{max}, \dot{m}, C_p, h_c, T_{in}, E, d_f, d_h, T_c, k_c, h_g$)
  - `UO2MaterialConstants(k_f,Macroscopic,Q,E)`

The parameters are then saved in COMMON blocks to be shared with other subroutines below. The number of flux values read from the input file (which is stored as the variable `length`) is needed to determine how many steps to divide the height of the reactor `z` into; and level of enrichment is required to calculate the Macroscopic cross-section for fission of the fuel.

The program then appends `"_T_output"` after the input file's name as the output file.

- `OutFileName(fileName)`

(i.e. if the input file is `flux.txt`, then the output file will be named `flux_T_output.txt`.)
If a file of the same name as the output file already exists, an option is given to the user to either proceed to overwrite the file or stop the program, to prevent silent overwriting.

The temperature at each height is then calculated using a `DO` loop, looping over all actively used values of flux.

  - `IncrementHeight(z,z1)`

If there are only 101 values in the input file `flux.txt`, then it will only loop over the first 101 elements in each array, with the exception of the two arrays `z` and `Tcool`, which will loop over the first 102 values of the arrays.
This is because the subroutines for calculating the values of each element in these two arrays propagates the value upwards(calculates `var1(i+1)` from `var1(i)`) instead of laterally (calculates `var2(i)` from `var1(i)`).
Therefore the sizes of these two arrays are of `arysize+1`=1024 to account for the edge case where `length`==`arysize`

The corresponding temperature changes can then be calculated using this information. Since the term $\frac{A_f q'''(z)}{\pi}$ (where q"' is a z-dependent variable given by equation 4) appears in every one of the equations,

$$\frac{A_f q'''}{\pi} = (d_f^2 - d_h^2) Q E \frac{N_A \rho}{A} \sigma_f \phi(z) \tag{6}$$

an array `q_lin_over_pi` is allocated to store its values. It is named as such because $q''' A_f = q'$ which is the linear heat generation rate, i.e. heat produced per unit length of fuel pin:

  - `LinearHeatGenOverPi(flux(i), q_lin_over_pi(i))`

This, and the following subroutines, retrieve parameters for the reactor via common blocks. The definitions of these parameters are listed in Table 2.

The coolant temperature is then

$$T_{cool}(z + \Delta z) = T_{cool}(z) + \left(\frac{A_f q'''(z)}{\pi} \right) \pi \frac{\Delta z}{\dot{m} C_p} \tag{7}$$

which is calculated by the following subroutine using `COMMON /Tcool/ dz, mdot, Cp`
A better approach is to take the average of the flux at z and $z + \Delta z$

$$T_{cool}(z + \Delta z) = T_{cool}(z) + \left(\frac{\left(\frac{A_f q'''(z+\Delta z)}{\pi} \right) + \left(\frac{A_f q'''(z)}{\pi} \right)}{2} \right) \pi \frac{\Delta z}{\dot{m} C_p} \tag{8}$$

This more accurately represents the temperature change of the coolant after travelling from z to Δz; but to keep the program simple the former option was adopted, leading to the result:

– `HeatCoolant(Ti,Ti1,qi)`

In a similar vein, the equation for calculating the temperature of various components and the associated subroutines to do the computation are listed below:

Exterior of cladding:

$$T_{oc}(z) = T_{cool}(z) + \left(\frac{A_f q'''(z)}{\pi}\right)\frac{1}{h_c d_c} \tag{9}$$

calculated by the following subroutine with the aid of `COMMON /Toc/ hc, dc`

– `CoolCladding( Tcool(i),Tclado(i),q_lin_over_pi(i))`

Interior of cladding:

$$T_{ic}(z) = T_{oc}(z) + \left(\frac{A_f q'''(z)}{\pi}\right)\frac{1}{2k_c} ln\left(\frac{d_c}{d_f}\right) \tag{10}$$

calculated by the following subroutine with the aid of `COMMON /Tic/ kc, ln_d_c_over_d_f`

– `CoolCladdingIn(Tclado(i),Tcladi(i),q_lin_over_pi(i))`

Surface of fuel pellets:

$$T_{fo}(z) = T_{ic}(z) + \left(\frac{A_f q'''(z)}{\pi}\right)\frac{1}{h_g d_f} \tag{11}$$

calculated by the following subroutine with the aid of `COMMON /Tof/ hg, df`

– `CoolFuel( Tcladi(i),Tfuelo(i),q_lin_over_pi(i))`

Maximum temperature inside fuel (nearest to the centre):

$$T_{fmax}(z) = T_{of}(z) + \left(\frac{A_f q'''(z)}{\pi}\right)\frac{1}{4k_f} - \left(\frac{A_f q'''(z)}{\pi}\right)\frac{1}{2k_f}\frac{d_h^2}{d_f^2 - d_h^2} ln\left(\frac{d_f}{d_h}\right) \tag{12}$$

where the last term $\to 0$ as $d_h \to 0$, calculated by the following subroutine with the aid of `COMMON /Tfmax/ kf, dh2_df2_dh2_ln_df_dh`

– `FindHottest( Tfuelo(i), Tfmax(i),q_lin_over_pi(i))`

The values of interest are printed to the screen as well as into the file as an extra line:

– `SaveTemps(z(i), Tcool(i),Tclado(i),Tfmax(i), o )`

After looping through all values, the file is closed, and the end time of the program is printed,

- `finished()`

and the program is exited.

3 Error handling features

The program may encounter user induced error in 4 locations:

1. Prompting the user to input the file name (user input)
2. Reading the file
   (a) If the file contains unreadable lines
   (b) If the file size is too big
3. Prompting the user to input the reactor type.
4. Saving the output file

Which are handled as follows:

1. Check that file exist; if not, prompt user to type in the file name again (loop infinitely until a correct file name is found)
2. Exit the main program after:
   (a) printing the line number where there is an unrecognized character.
   (b) Print the size of the allocated memory, and printing the line number where memory overflow will occur.
3. Check that the input is correct, if not ask the user to try again (loop infinitely)
4. If file exists, warn the user and ask the user whether to overwrite or not
   (a) if 'y', resume the program to overwrite the old file with the new outputs.
   (b) if 'n', exit the program

4 Portability of source code onto other machines

This program was written on a Linux (Ubuntu 18.04.1 LTS) machine compiled by GFortran. It has been subsequently been tested again on a Scientific Linux 7.3 (Nitrogen).

This source is expected to work across multiple platforms since:

- Despite the use of many subroutines and functions, they are all kept in the same file, so the source code can be compiled on its own without dependence on other files/libraries.
- No local file paths were specified
- No special line ending (e.g. \r\t) were used.
- The source code is written in FORTRAN 77 syntax, so it should be compilable on most platforms by any currently actively maintained FORTRAN compilers. (e.g. GFortran, which is available on macOS and Windows as well)

5 Potential future improvements

Currently, the program takes *any* flux values, and calculate an output temperature, without checking whether these flux values are reasonable or not.

The program makes some assumption about the material, i.e. it has temperature independent density, and the Doppler coefficient =0. In a real reactor, the fuel should expand and be less susceptible to fission by thermal neutrons, so this should be taken into account in a more advanced version of the program.

A more advanced version of this program may check for negative values in the input file's flux values, which is unphysical. And if the appropriate data is found, such as the recrystallization temperature of the cladding, and safe operating temperature of UO2, the safe operating conditions can be found, so that the program can print a warning when the temperature goes over the safe limits.

This program also exits upon encountering any non-floating point value lines. In the future, if one wishes to, this program can be improved upon by adding a feature to skip over lines starting with "#", which typically denotes a header.

It is also restricted to read a file below a certain size, arbitrarily chosen as 1023. This restriction can be lifted using some of the more recent FORTRAN syntax, e.g. with automatic object in later versions of FORTRAN, or with another language (e.g. Python). However, this is very difficult to achieve with FORTRAN 77, as the size of the array needs to be known before any file reading operation begins. So while this limit can be easily edited in the source code (by editing the parameter `arysize`), this value will be fixed once the code is compiled. [1]

Lastly, this program uses simple summation of the given flux value over the intervals of height. In the future, cubic spline can be used to obtain the flux value at any arbitrary height between any two data points; and then an integration over that can be carried out, such that the temperature profile can still be well approximated even when very few data points are provided by the user, increasing the ease of use.

Appendices

symbol	name in program	physical meaning	SI unit	usually given in
F	-	reaction(fission) rate per volume	$m^{-3}s^{-1}$	
Q	`Q`	Q value of the reaction	J(Joule)	eV
$\phi(z)$	`flux(i)`	2200 ms^{-1} flux (thermal neutron flux) as a function of height	$m^{-2}s^{-1}$	$cm^{-2}s^{-1}$
Σ_f^F	`Sigma_mac` /`Macroscopic`	macroscopic fission cross section of the fuel (UO_2)	m^{-1}	
N_d	-	number density of ^{235}U	m^{-3}	
σ_f	`sigma_mic`	microscopic fission cross section of UO_2	m^2	b(barns)
ρ	`rho`	density of UO_2	kgm^{-3}	gcm^{-3}
E	`E`	enrichment (fraction of ^{235}U)	1	%
A	`A`	molar mass of UO_2(dependent on enrichment)	kg	g

Table 1: Definitions of constants used in equation 1 to 4

symbol	name in program	physical meaning	SI unit
z_{max}	`zmax`	Total height of channel	m
$\dot{m}$	`mdot`	Mass flow rate per channel	kgs^{-1}
C_p	`Cp`	Specific heat capacity of coolant	$Jkg^{-1}\ K^{-1}$
h_c	`hc`	Convective heat transfer coefficient	$Wm^{-2}\ K^{-1}$
T_{in}	`Tin`	Inlet coolant temperature	°C
d_f	`df`	Outer diameter of fuel pellet,	m
d_h	`dh`	Inner diameter of fuel pellet,	m
t_c	`Tc`	Thickness of cladding	m
k_c	`kc`	Thermal conductivity of cladding,	$Wm^{-1}\ K^{-1}$
h_g	`hg`	Gap conductance, h g	$Wm^{-2}\ K^{-1}$
d_c	`dc`	Outer diameter of cladding	m

Table 2: Definitions of constants used from equation 6 to 12. Thicknesses/diameters were given in mm and has to be converted to m (SI).

References

[1] Ibm.com. (2018). IBM Knowledge Center. [online] Available at: `https://www.ibm.com/support/knowledgecenter/SSGH4D_13.1.0/com.ibm.xlf131.aix.doc/language_ref/autoobj.html` [Accessed 26 Dec. 2018].