Tutorial of general-purpose reactor physics code system CBZ

Go CHIBA

June 6, 2024

1 Introduction

When neutron collides with nucleus, nuclear fission reaction might occur with given probability and this nucleus is divided into (generally) two fragments. These fragments, known as *fission fragments*, possess a huge amount of kinetic energy of about 167 MeV, and nuclear power utilizes this energy finally as electric power. [1]

Nuclear fuel in which fission reactions occur is generally contained in a *pressure vessel*. In a pressure vessel, various equipments are located, and a small part around nuclear fuels is called a *reactor core*, and a reactor core is a "heart" of a nuclear power plant.

In nuclear power plants, the number of neutrons which cause fission reactions is controlled to maintain fission reactions. That is, to properly control fission reactions, we have to know and understand behavior of neutrons in a reactor core. Neutron behavior in a reactor core is described by the Boltzman-type neutron transport equation, so we have to solve this equation somehow. To solve this equation to know neutron behavior is called *reactor core analysis* or *neutronics analysis*, and these are one of important subjects in the field of nuclear engineering.

Many code systems for reactor core analyses have been developed so far in the world. One of the most famous code systems might be CASMO/SIMULATE developed by STUDVIK. This code system has been widely used for actual light water power reactors analyses in many countries. In Japan, various code systems have been developed, and the SRAC code of Japan Atomic Energy Agency (JAEA) and AEGIS/SCOPE-2 developed by Nuclear Fuel Industry, Nuclear Engineering Ltd. and Nagoya university would representative.

The author worked at JAEA in the past. There he had developed a code system CBG by himself, and had published several papers with it. Based on the development experience of CBG, he began to develop a new code system CBZ after moving to Hokkaido University. This is the reason why both of the names "CBZ" and "CBG" are used through this tutorial. With CBZ, one can perform various kinds of reactor physics calculations such as criticality calculations, kinetic calculations, radiation shielding calculations and nuclear fuel burnup calculations. This tutorial is for users to learn how to use the code system CBZ.

[1] Strictly speaking, fission energy includes energy of γ-ray instantaneously emitted in the fission reaction and energy of β-ray and γ-ray which are emitted from unstable fission fragments nuclides.

2 Install

When one uses CBZ, he can do his calculations by using various modules provided in CBZ. This can be done for users by writing a computer program with C++, and compiling and running it. In this compiling, a library file `libCBG.a`, which can be generated by compiling varous modules of CBZ, should be included. This is shown in **Fig. 1**. In this section, we will compile the CBZ source program. How to compile computer programs written by users will be explained in the following sections.

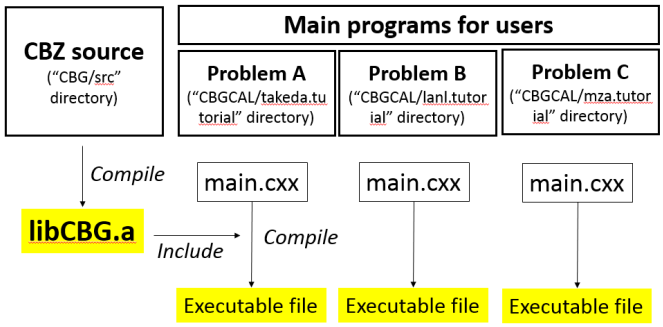

Fig. 1: Relation between the CBZ source files and computer programs prepared by users

After uncompressing and expanding the package named as `cbz.xxxxyyzz.tgz`, [2] you can get a directory `CBZ` which contains the following directories: [3]

- `CBG`
- `CBGCAL`
- `CBGLIB`
- `CBGLIB_BURN`

In the `CBG` directory, you can find a `src` directory. In this directory, source codes of CBZ are located.

The `CBGCAL` directory is used to locate data to perform actual calculations, and in the distribution package you can find several directories such as `Takeda.tutorial`, `lanl.tutorial` and `mza.tutorial`.

The `CBGLIB` directory is used to store nuclear reaction data (nuclear data), CBZLIB [4], and the `CBGLIB_BURN` directory is to store data files required in nuclear fuel burnup calculations.

Now let us compile CBZ. Please move to the `CBG/src` directory. [5] There is possibility that compilation has been already done in the distribution package, so please type `make clean` to delete all the files previously generated. Then to start compiling, please type `make`. After that, it starts compiling and a file `libCBG.a` might be generated after several minutes. By typing `ls *.a`, you can confirm that this file is successfully generated. When the CBZ source programs are revised, users have to compile them again and newly generate `libCBG.a`.

This is the end of the install. Note that CBGLIB might not be included in the package because of its large size. In such a case, please contact with Chiba or others to get the CBGLIB data.

[2] Please type `tar xvfz cbz.xxxxyyzz.tgz`.

[3] In the `CBZ` directory, many files are located in layer-structured directories. When one wants to move all the data contained in the `CBZ` directory, it is quite convenient if one can create one file including all these data. Doing "tar" is to summarize one directory including the data into one file, and a summarized file is called a *tar file*. For example, when ones create a tar file for the `CBZ` directory, they have to type `tar cvf cbz.tar CBZ`, and then one file `cbz.tar` which contains all the data can be generated. Furthermore, a tar file is generally large-sized, so it is better to compress it. The command for compression is `gzip cbz.tar`, and by doing this we can generate a `cbz.tar.gz` file. The above procedure can be done at once by a command `tar cvfz cbz.tgz CBZ`, and its opposite is `tar xvfz cbz.tgz`.

[4] Nuclear data defines probability of interactions between neutron and nucleus. Detail will be described later.

[5] Please type `cd CBZ/CBG/src`.

3 Calculations of Takeda benchmark

Here let us calculate a neutron transport problem, *Takeda benchmark*, with CBZ. [6] The Takeda benchmark includes four (sub-)problems (or reactor cores), and here we use the simplest one, problem 1. Geometrical specification of the problem 1 is shown in **Fig. 2**.

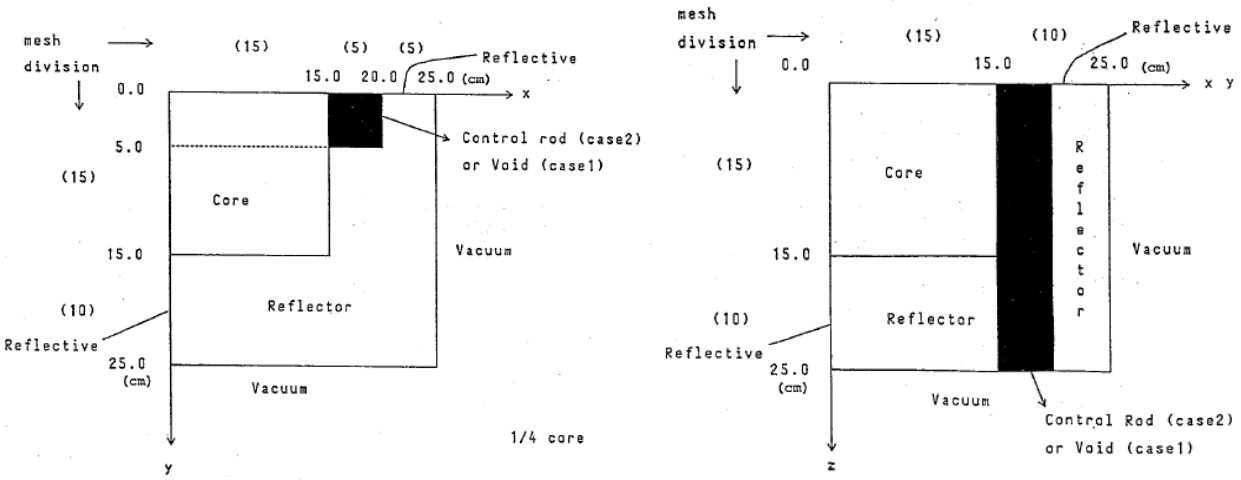

Fig. 2: Problem 1 of Takeda benchmark (left: XY-plane, right:XZ(YZ)-plane)

The left part of Fig. 2 is a top view. In this figure, 1/4 of the whole system is presented. We can find that a core region is located around the center and a reflector region is around the peripheral. Black region in the reflector is a position in which a control rod is inserted. When a control rod is not inserted, this region should be treated as void. The right part of this figure is a side view, and you can find that a control rod (or void) region has a length along the z-axis, and that a core region is surrounded by a reflector region in axial direction also.

Please open a file `main.takeda1.dif_1.cxx` in the directory `CBGCAL/Takeda.tutorial` by an editor.

First you can find a comment line with `Geometry data` around line 20. Below this line, geometry data of problem is defined. In this problem, Cartesian coordinate is used, and each direction (X, Y and Z) is divided into five blocks. As defined by arrays `xl`, `yl` and `zl`, length of each block is set 5 cm, and in calculations, each block is further divided into five as defined by arrays `xm`, `ym` and `zm`. Each axis is divided to 25, so the total number of spatial meshes becomes $25 \times 25 \times 25 = 15,625$ in this example. Material assignment to $5 \times 5 \times 5 = 125$ blocks is defined by an array `mat`. In this example, "0" is fuel, "1" is reflector, "2" is control rod and "3" is void. In the problem 1 of the Takeda benchmark, two calculations are expected: a control rod insertion case and a control rod withdrawn case. In this example, material "3" is assigned to control rod position, so this example corresponds to the control rod withdrawn case.

Next you can find other comment line with the description "`Material data`" around line 70. Below this line, material data, that are neutron-nucleus reaction cross sections, are defined. In this problem, neutron energy is discretized to two. [7]

Finally, using the geometry data and material data defined above, the neutron diffusion equation is numerically solved. As mentioned above, neutron behavior is described by the neutron transport equation. The diffusion equation can be derived by introducing several assumptions to the transport equation. In the neutron transport equation, position, energy and direction dependence of neutrons should be considered, but angular dependence can be neglected in the diffusion equation. This makes diffusion equation easier to be solved. Here we use a solver PLOS which can numerically solve the diffusion equation.

First, please type `cp main.takeda1.dif_1.cxx main.cxx` to copy a file `main.takeda1.dif_1.cxx` to `main.cxx`. Then type `make` to compile it. [8]

As a result, you can generate a load module file named `a.out`. Please confirm this. After the `a.out` file is generated, please type `./a.out` and perform calculations. A result similar to the following would be printed on your screen.

[6] The Takeda benchmark is a benchmark problem prepared by Prof. Toshikazu Takeda of Osaka University to verify computer programs for neutron transport. In 1991, experts on neutron transport calculations were invited by OCED/NEA and their own computer codes were compared with each other using this benchmark problem.

[7] Divided energy region is called *group*, and we can say that the number of energy groups is two in this problem.

[8] `make` compiles a file `main.cxx` located in the same directory. Details of procedures are defined in a file `makefile`. So when you have a file to be calculated, you have to copy (or rename) it to `main.cxx` in CBZ.

Listing 1: Example of Takeda benchmark calculation

```
#*** Total Mesh : 15625
#***********************************************
#* System CBG
#*   Power iteration
#*   Solver : PLOS
#*   Acceleration : none
#*   Forward calculation
#*   Convergence condition (k_eff) : 1e-05
#*                         (flux)  : 0.0001
#*                         (source): 0.0001
#***********************************************
#*It:   K_eff    : Err in k : Err in    : Err in
#*er:           :          : flux      : source
#***********************************************
#    0:   0.906483  9.352e-02  2.233e+00  3.700e-01
#    1:   0.917541  1.220e-02  1.018e-01  7.723e-02
#    2:   0.925198  8.344e-03  6.630e-02  5.346e-02
#    3:   0.927640  2.640e-03  1.552e-02  1.376e-02
#    4:   0.927664  2.565e-05  4.699e-02  2.787e-02
#    5:   0.928331  7.198e-04  4.428e-03  2.952e-03
#    6:   0.927417  9.849e-04  1.424e-03  2.563e-02
#    7:   0.927214  2.192e-04  4.204e-03  1.606e-02
#    8:   0.927261  5.146e-05  2.991e-03  7.066e-03
#    9:   0.927328  7.128e-05  3.155e-03  2.690e-03
#   10:   0.927390  6.744e-05  6.710e-04  4.409e-04
#   11:   0.927305  9.221e-05  2.339e-04  2.580e-03
#   12:   0.927291  1.492e-05  4.324e-04  1.440e-03
#   13:   0.927291  6.520e-07  8.558e-04  7.044e-04
#   14:   0.927304  1.414e-05  1.460e-04  1.022e-04
#   15:   0.927295  9.924e-06  4.249e-04  3.309e-04
#   16:   0.927300  5.689e-06  5.646e-05  3.009e-05
```

A solver PLOS solves the neutron diffusion equation with an iterative procedure, so neutron effective multiplication factor k_{eff} and (spatial and energy distributions of) neutron flux are revised at each iteration. By confirming those quantities are converged, these can be final results. More detail, difference between a result of the present iteration and that of the preceding iteration is compared, and convergence is confirmed if this difference is smaller than a predetermined criteria. The above example is a process during this iterative procedure, and relative differences of k_{eff} and neutron flux and fission source are shown. Also you can get the final result of k_{eff} from this.

Numerical calculations should include various errors and one of the most important ones is discretization error on space. In the present example, one block with 5 cm length is divided into $5 \times 5 \times 5$ meshes. We should know that a result obtained with this model is just an approximated solution, and rigorous solution which is free from the discretization error can be obtained when we use infinite number of spatial meshes.

Problem 3.1: Perform calculations with fine mesh division: the number of meshes per each plane should be increased by two or three times. Compare result (k_{eff}) and computation time. You can measure computation time by typing `time ./a.out`. Change in the number of spatial meshes can be done by changing values of `xm`, `ym` and `zm`.

This benchmark problem has rigorous numerical solution, which was obtained by solving the neutron transport equation with the Monte Carlo method, and the rigorous solution is 0.9780 in the control rod withdrawn case. What about your result with PLOS? There might be large difference between your result and the rigorous one. As already mentioned, PLOS yields an approximated solution to the diffusion equation, which is approximated form of the transport equation, so the PLOS solution is just an approximation.

Next let's use another program to solve the neutron transport equation. Please run `main.takeda1.sn.cxx`. Note that this calculations uses $10 \times 10 \times 10$ meshes for each block with 5 cm plane. What about your result?

Here we use a solver SNT, which solves the neutron transport equation numerically. By using SNT, you might get solution close to the rigorous one (0.9780), but you require longer computation time than PLOS. In the field of numerical calculations, method development to reduce computation time to attain convergence is one of the most important subjects. The same thing can be said in the reactor core analysis field, and various methods have been proposed. Let us use here the diffusion synthetic acceleration (DSA) method. This method uses solution of diffusion equation to guess converged solution to transport equation. You can see a command `sys.NoDSAAcceleration()` around line 5 from the end in `main.cxx`, so please remove this line or comment it out by adding `//`. By doing this, you can get converged solution with short computation time.

In addition to DSA, we can further accelerate calculation by using the coarse-mesh finite difference (CMFD) acceleration. You can use this by changing `CalIgen(‘‘none’’);` around the end of the program to `CalIgen(‘‘cmfd’’);`. Please confirm that computation time is further reduced.

Problem 3.2: Next let us calculate the control rod insertion case. It can be easily done by changing material assignment. Control rod reactivity worth can be calculated from two k_{eff}s of the control rod insertion and withdrawn cases, so obtain these reactivity worth by both the diffusion and transport calculations. Control rod reactivity worth can be defined as $(1/k_1 - 1/k_2)$ if neutron multiplication factors before and after the control rod insertion are k_1 and k_2. In the reference Monte Carlo calculations, a control rod reactivity worth of -0.0166$\Delta k/kk'$ was obtained, so please compare your results with this reference.

Next let's observe spatial distribution of neutron flux.

Here we use a method `ShowNeutronFluxAlongXAxis` which outputs neutron flux spatial distribution along the X axis. You can use this by commenting out a corresponding line in the file `main.takeda1.dif_1.cxx`. Arguments of this method are as follows: The first and second ones are positions in Y and Z directions for which neutron flux spatial distribution is printed out. If both of them are zero, neutron flux spatial distribution along the X axis about the first meshes on the Y and Z directions. The third argument is an energy group of which neutron flux spatial distribution is printed out. If you choose `-1` for this argument, neutron fluxes of all energy groups are printed. Please be careful that the first position/energy corresponds to `0` in a C++ array.

Problem 3.3: Please plot spatial distribution of neutron flux of groups 1 and 2 along the X-axis by Gnuplot or Excel. Control rod is inserted and distribution should be chosen over control rod position. In addition, the same plot should be prepared for the control rod withdrawn case, and compare the distribution before and after the control rod insertion.

Calculation results are printed out on your screen, so you have to copy and paste it. This should be cumbersome, so you can do calculations by typing `./a.out > output`. If you do so, all these results are written in a file `output`, and you can edit this file to manipulate your results.