論文 / 著書情報
Article / Book Information

Title	MIK - Multi-region Integral Kinetic code, Version 1.0
Authors	Delgersaikhan Tuya, Hiroki Takezawa, Toru Obara
Citation	MIK-multi-region integral kinetic code, version 1.0
Pub. date	2018, 12

MIK – Multi-region Integral Kinetic code, Version 1.0

Delgersaikhan Tuya, Hiroki Takezawa, Toru Obara

Tokyo Institute of Technology

2018

MIK – Multi-region Integral Kinetic code, Version 1.0

Delgersaikhan Tuya, Hiroki Takezawa, Toru Obara

Tokyo Institute of Technology

2018

Published on December 1, 2018

Correspoinding author:
Toru Obara, Professor
Laboratory for Advanced Nuclear Energy,
Institute of Innovative Research,
Tokyo Institute of Technology
2-12-1-N1-19 Ookayama, Meguro-ku, Tokyo 152-8550, Japan
Phone: +81-3-5734-2380
Fax: +81-3-5734-2959
E-mail: tobara@lane.iir.titech.ac.jp

MIK – Multi-region Integral Kinetic Code, Version 1.0

Delgersaikhan Tuya[a], Hiroki Takezawa[b], Toru Obara[a]

[a] *Laboratory for Advanced Nuclear Energy, Institute of Innovative Research, Tokyo Institute of Technology, Tokyo, Japan*

[b] *Department of Nuclear Safety Engineering, Faculty of Engineering, Tokyo City University, Tokyo, Japan*

Abstract

MIK – Multi-region Integral Kinetic code based on Monte Carlo neutron transport method and integral kinetic model (IKM) has been developed. MIK code calculates a time- and region-dependent fission rate (i.e., power) during a nuclear supercritical transient following a reactivity insertion in a fissile system of arbitrary geometry and composition. A time-dependent feedback capability of MIK code allows a treatment of various system-dependent complicated non-linear feedback phenomena via user-provided governing equations for feedback variables.

This document presents theoretical background and implementation detail of MIK code along with a sample calculation for a Godiva supercritical transient experiment. In addition, as MIK code is an open source code, its source list is provided at the end of this document. In short, this document serves as both a report of MIK code development and a user guide of MIK code.

Keywords: MIK code; integral kinetic model; Monte Carlo neutron transport; criticality safety; nuclear supercritical transient;

CONTENTS

1. INTRODUCTION

MIK – Multi-region Integral Kinetic code can be used to simulate a supercritical transient in an arbitrary fissile system following a reactivity insertion. In specific, it calculates a time- and region-dependent fission rate (i.e., power) during a supercritical transient in a given system considering the given feedback mechanisms. MIK code essentially consists of three parts, namely, the Integral Kinetic Model (IKM), continuous energy Monte Carlo neutron transport code MVP2.0 or MVP3.0 (hereinafter MVP2.0/3.0), and a time-dependent feedback model. In specific, the IKM models a time evolution of fission rate (i.e., power) in considered system in region-wise manner using a neutron transport time- and region-dependent kinetic functions that represent neutronic interaction between the regions; MVP2.0/3.0 code is utilized to calculate the neutron transport time- and region-dependent kinetic functions; and the time-dependent feedback model calculates any change in feedback variables during a transient via user-provided equations.

The current version of MIK code is implemented by externally coupling the IKM with MVP2.0/3.0 in order to show the feasibility of IKM-MVP coupling for a transient calculation. Accordingly, user is required to have MVP2.0/3.0 code installed before being able to use MIK code. As for the time-dependent feedback model, since specific feedback mechanisms are entirely dependent on a considered fissile system and problem, user is required to provide the governing equations for the variables related to the feedback mechanisms in the system.

The current MIK code can run in either single core mode or parallel mode using Open MPI 3.0 or higher. Whenever possible, user is urged to run parallel mode MIK calculation. Especially, as MVP2.0/3.0 calculation takes majority of an overall running time in entire MIK calculation, a parallel mode calculation using as many as reasonably possible cores for MVP2.0/3.0 is recommended. In case of the IKM, a number of parallel cores should not exceed a number of regions.

The rest of this document is organized as follows. In section 2, first the basic theories are given for the IKM, a method to calculate the kinetic functions, and implementation of a feedback. In section 3, MIK code is discussed in detail. A sample calculation example is then worked out to give user a better understanding of how a MIK calculation can be set up and run in section 4. Finally, at the end of this document, the source list of MIK code is given based on the example calculation described in section 4.

Although MIK code is an open source code, MVP2.0 or MVP3.0 is not. Therefore MVP2.0 or MVP3.0 is not included in the MIK code package and needs to be obtained separately from its developers or corresponding organizations.

2. BASIC THEORY

As MIK code is based on the Integral Kinetic Model (IKM), in this section first the IKM is discussed. Then the method to calculate the kinetic functions needed by the IKM is presented in section 2.2. Finally, the implementation of the time-dependent feedback modeling in the MIK code is explained in section 2.3.

2.1. Integral Kinetic Model

MIK code is based on the IKM, which is derived from the Boltzmann neutron transport equation. The IKM and its derivation can be found in the references [1]. The IKM for an arbitrary system of R fissile regions is [2]

$$N_i(t) = \sum_{j=1}^{R} \left(\int_{-\infty}^{t} \alpha_{ij}^{p}(t-t')N_j(t')dt' + \int_{-\infty}^{t} \alpha_{ij}^{d}(t-t')N_j(t')dt' \right), \quad (2.1)$$

where

$N_i(t)$	Total fission rate in region i at time t (fission/second)
$\alpha_{ij}^{p}(t-t')$ and $\alpha_{ij}^{d}(t-t')$	Secondary prompt-fission and delayed-fission probability density functions, respectively, in region i provided by the fission in the source region j with time difference t-t' (fissions in i $\times s^{-1}$/fission in j).

The current version of MIK code treats only supercritical transient, in which the contribution of delayed neutrons can be negligible. For supercritical transient, Eq. (2.1) reduces to

$$N_i(t) = \sum_{j=1}^{R} \int_{t-\tau_c}^{t} \alpha_{ij}(t-t')N_j(t')dt' , \quad (2.2)$$

where τ_c is essentially maximum neutron transport time in the system. Also note that the superscript p for prompt neutrons is omitted in Eq. (2.2).

The neutron transport time between successive fissions across regions is explicitly treated with the $\alpha_{ij}(\tau)$ functions. Using this function, a cumulative distribution function can be defined as

$$C_{ij}(\tau) = \int_0^{\tau} \alpha_{ij}(\tau')\, d\tau'. \quad (2.3)$$

This cumulative distribution function represents the ratio between number of secondary fissions in region i caused by a source fission in region j by time τ and number of source fissions in region j.

Since Eq. (2.2) cannot be solved analytically, a numerical method is employed in MIK code. A discretized IKM for supercritical transient is

$$N_i(k\Delta t) = \sum_{j=1}^{R}\left\{\widetilde{N}_j\left[\tilde{C}_{ij}(\tau', \mathbf{S}_0)\right]_{k\Delta t}^{k_c\Delta t} + \sum_{k'=k-k_c}^{k-1} N_j(k'\Delta t)\left[C_{ij}(\tau', \mathbf{S}_k)\right]_{(k-k'-1)\Delta t}^{(k-k')\Delta t}\right\}, \quad (2.4)$$

where

$\widetilde{N}_j$	Initial steady-state fission rate in region j
$\tilde{C}_{ij}(\tau', \mathbf{S}_0)$	Steady-state cumulative distribution function at initial state $\mathbf{S}_0$
$\mathbf{S}_k$	State of the system at time $k\Delta t$
$\left[C_{ij}(\tau', \mathbf{S}_k)\right]_{(k-k'-1)\Delta t}^{(k-k')\Delta t}$	$C_{ij}\left((k-k')\Delta t, \mathbf{S}_k\right) - C_{ij}\left((k-k'-1)\Delta t, \mathbf{S}_k\right)$.

In MIK code, an initial condition of a system is always a critical steady-state, which means there is already an existing fundamental mode of neutron flux before the reactivity insertion as it can be seen from Eq. (2.4). Mostly, the initial condition is assumed to be low or zero power (e.g., 1 W). Based on these assumptions, the kinetic functions at the initial condition is calculated using the following equation.

$$\tilde{C}_{ij}(\tau', \mathbf{S}_0) = \frac{C_{ij}(\tau', \mathbf{S_1})}{k_p}, \quad (2.5)$$

where $C_{ij}(\tau', \mathbf{S_1})$ is kinetic function at the state $\mathbf{S_1}$ just after the first reactivity insertion and k_p is prompt fission multiplication factor at that state.

From the Eq. (2.4), it can be seen that $C_{ij}(\tau)$ functions are the key kinetic function of the IKM. In MIK code, these functions are calculated using Monte Carlo neutron transport method. In specific, continuous energy Monte Carlo code MVP2.0/3.0 is used for this purpose. In the next section, the method to calculate the kinetic functions will be introduced.

2.2. Method to calculate kinetic functions

Using special tally options and time-dependent option in the eigenvalue calculation in continuous energy Monte Carlo neutron transport code MVP2.0/3.0 [3][4], a neutron transport time- and region-dependent reactions such as fission, neutron production, or loss can be tallied. What follows in this section is essentially based on the method described in the section II.B of the reference [5].

The neutron transport time- and region-dependent fission reaction needed to calculate a secondary fission cumulative distribution function can be defined as follows.

$$f_{ij,m} = \int_{(m-1)\Delta t}^{m\Delta t} d\tau \int_{V_i} d\mathbf{r} \int_0^{E_{max}} dE \int_{V_j} d\mathbf{r}'\, \Sigma_f(\mathbf{r}, E)\Phi(\mathbf{r}' \to \mathbf{r}, \tau, E), \quad (2.6)$$

where

Σ_f	macroscopic fission cross section (cm^{-1})

$\Phi(\mathbf{r}' \to \mathbf{r}, t, E)$	neutron flux coming originally from $\mathbf{r}'$ to $\mathbf{r}$ with energy E after neutron transport time τ (n·cm^{-2}·eV^{-1}·cm^{-3}·s^{-1})
V_i and V_j	volumes of region i and j (cm^3), respectively
m	index of the m-th time bin
Δt	time-step size (s) (same as that in Eq. (2.4))

The neutron transport time- and region-dependent fission reaction defined in Eq. (2.6) can be directly calculated using the special tally options in the eigenvalue calculation of the MVP2.0/3.0. A detailed discussion on how to use the special tally options will be discussed in section 3.3.1.

A cumulative fission by the k-th time bin and over the entire transport time range (i.e., τ_c) can then be calculated respectively as

$$F_{ij}(k\Delta t) = \sum_{m=1}^{k} f_{ij,m} \,, \tag{2.7}$$

$$F_{ij} = F_{ij}(\infty) = F_{ij}(k_c\Delta t) = \sum_{m=1}^{k_c} f_{ij,m} \,. \tag{2.8}$$

Using the cumulative fission over the entire time range defined by Eq. (2.8), the total number of fissions in source region j at the end of generation (i.e., tally batch in MVP2.0/3.0) is

$$F_j = \sum_{r=1}^{R} F_{jr} \,, \tag{2.9}$$

where R is the total number of fissile regions in a system. As just mentioned above, the total number of source fissions defined by Eq. (2.9) is at the end of the generation, meaning that it represents total source fissions in region j after the neutrons are already multiplied in the system. And in order to calculate the secondary fission cumulative distribution function, the total number of source fissions at the beginning of the generation is needed. In order to calculate that, F_j is needed to be divided by fission multiplication factor as

$$F_j' = \frac{F_j}{k_f} = \frac{\sum_{r=1}^{R} F_{jr}}{k_f} \,, \tag{2.10}$$

where k_f is formally the effective prompt fission multiplication factor and is the same with the effective prompt multiplication factor k_p, which is directly obtained by the eigenvalue calculation itself. Finally using Eqs. (2.7) through (2.10), the secondary fission cumulative distribution function is then calculated as

$$C_{ij}(k\Delta t) = \frac{F_{ij}(k\Delta t)}{F_j'} = k_p \frac{F_{ij}(k\Delta t)}{F_j} \,. \tag{2.11}$$

Since the neutron transport time- and region-dependent fission reaction defined by Eq. (2.6) is obtained with the Monte Carlo method, it has statistical uncertainty $\sigma_{ij,m}$ that propagates through Eqs. (2.6 – 2.11). The uncertainty propagation ignoring any correlation is calculated as

$$\sigma\left(C_{ij}(k\Delta t)\right) = \sqrt{k_p^2\left[\left(\frac{\sigma\left(F_{ij}(k\Delta t)\right)}{F_j}\right)^2 + \left(\frac{F_{ij}(k\Delta t)\sigma(F_j)}{F_j^2}\right)^2\right] + \left(\frac{F_{ij}(k\Delta t)}{F_j}\right)^2 \sigma^2(k_p)}\,, \tag{2.12}$$

where

$$\sigma(F_j) = \sqrt{\sum_{r=1}^{R} \sigma^2(F_{jr})}\,,$$

$$\sigma(F_{ij}) = \sqrt{\sum_{m=1}^{k_c} \sigma_{ij,m}^2}\,,$$

$$\sigma\left(F_{ij}(k\Delta t)\right) = \sqrt{\sum_{m=1}^{k} \sigma_{ij,m}^2}\,.$$

The statistical uncertainty defined by Eq. (2.12) further propagates through the IKM and the calculation of such uncertainty is given in the reference [2].

2.3. Time-dependent feedback model

A feedback is extremely important part of a supercritical transient phenomenon. It controls the course of a transient and is highly system-dependent. In this section, an implementation of time-dependent feedback in MIK code is discussed.

During a transient, physical characteristics of a system such as temperature, density, and geometry etc change due to fission energy deposition and feedback phenomena. As a result, cumulative distribution function $C_{ij}(k\Delta t, \mathbf{S_k})$ in Eq. (2.4) changes time-dependently. MIK code allows a treatment of time-dependent and complex feedback mechanisms through the time-dependent feedback modeling capability.

In MIK code, the considered system is represented by its state $\mathbf{S}(t)$ (i.e., $\mathbf{S_k}$) and the state can be composed of various physical variables denoted as *update variables* depending on the system. The update variables are time-dependently calculated during a transient and used to update the kinetic functions at the latest state of the system. For example, in most cases a state of the considered system would be composed of the update variables of at least region-wise temperature and density. In other words, the state can be written as $\mathbf{S}\left(\mathbf{U}(t)\right)$, where $\mathbf{U}(t)$ is a matrix of the update variables at time *t*. The matrix of update variables itself is $\mathbf{U}(\mathbf{T}^T, \mathbf{D}^T)$, where $\mathbf{T}\left(T_1(t), T_2(t), \dots, T_R(t)\right)$ is a vector of region-wise temperatures, $\mathbf{D}\left(D_1(t), D_2(t), \dots, D_R(t)\right)$ is a vector of region-wise densities, and the superscript *T* denotes a transpose operator. Accordingly, the dimension of update variables matrix is $R \times M$, where M is a number of update variables.

In MIK code, update variables are calculated at each time-step of the IKM using user-provided governing equations. Ideally, the neutron transport time- and region-dependent kinetic functions

$C_{ij}(k\Delta t, \mathbf{S}_k)$ must be calculated at each time-step of the IKM reflecting any change in the update variables (thus change in the state of the considered system) using the Monte Carlo method introduced in section 2.2, however, doing so is impractical from the viewpoint of computation cost and time. Furthermore, the time-step size of the IKM is essentially very small, at least hundreds of times shorter than the maximum neutron transport time τ_c in the considered system because at least hundreds of data points are needed to express the kinetic function shape reasonably well (see Eqs. (2.4 & 2.6)). During such a small time period of successive time-steps, the kinetic functions hardly change, thus calculating them at each IKM time-step using the Monte Carlo method is both unnecessary and impractical.

In the current MIK code, the kinetic functions are calculated using the Monte Carlo method routinely during a transient. In other words, the kinetic functions are calculated at certain non-uniform time-steps, not necessarily at each time-step, during the transient. In specific, the kinetic functions are calculated at the latest state of the system when a cumulative change in any update feedback variable since the last update exceeds the user-defined threshold. The condition for calculating/updating the kinetic functions at the latest state of the system is

$$|\mathrm{U}_{rv}(k\Delta t) - \mathrm{U}_{rv}(k'\Delta t)| \geq \delta_{rv} \ , \qquad (2.13)$$

where U_{rv} is an element of **U** matrix; δ_{rv} is an element of a user-defined update criteria matrix $\boldsymbol{\delta}$; $r=1,\ldots,R$ is index for region; $v=1,\ldots,V$ is index for a update variable; k is index for current (latest) time-step; and k' is index for a time-step at which the kinetic functions were last calculated/updated.

This time-dependent feedback model allows a rather accurate treatment of complicated non-linear feedbacks given that, as mentioned before, user provides the governing equations for the update variables. Furthermore, user can manage the accuracy and computation time of the calculation by changing the update criteria matrix $\boldsymbol{\delta}$. If each element of the update criteria matrix is, for example, set to zero, then the kinetic functions will be calculated at each time-step of the IKM via the Monte Carlo method described in section 2.2, however, with hugely increased computation time. On the other hand, by setting the elements of the update criteria matrix larger, the kinetic functions will be calculated less often and the computation time and cost will decrease substantially, however, accuracy of the calculation can degrade. Therefore, it is important for user to find a reasonable trade-off between the cost and accuracy by changing the update criteria matrix that is highly dependent on the considered system and problem.

A discussion on how the time-dependent feedback model can be set up in MIK calculation will be presented in section 3.3.3 and will be clarified more in section 4.4 through an example.

3. MIK CODE

This section introduces MIK code in detail from the technical point of view. As such, this section contains four essential parts: (1) general structure of the MIK code, (2) setup of MIK code on Linux system, (3) MIK calculation setup, and (4) running a MIK calculation on Linux system.

3.1. General structure

A rather simplified general structure of MIK code is shown in Fig. 3.1 and simplified basic calculation flowchart is shown in Fig. 3.2.

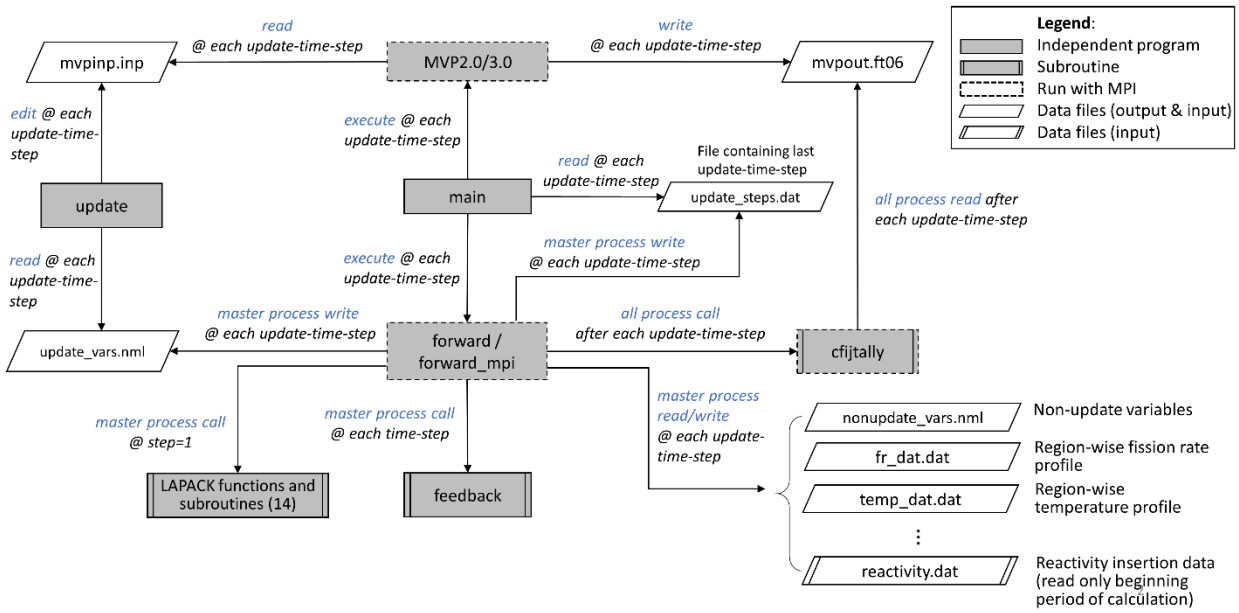

Fig. 3.1. Simplified general structure of MIK code

As shown in the figures, the *main* program executes the *forward*/or *forward_mpi* (hereinafter *forward/forward_mpi*), *update*, and *MVP2.0/3.0* programs successively until the end of a calculation. In brief, at each cycle, the *forward*/*forward_mpi* program advances the solution of the IKM defined by Eq. (2.4) in time using forward Euler method, calculates time-dependent update variables at each time-step via the *feedback* subroutine, and checks whether or not the update condition defined in Eq. (2.13) is satisfied. If the condition is satisfied, then the *forward*/*forward_mpi* stores the necessary data in the related files and returns calculation control back to the *main* program. The *main* program then executes the *update* program, which updates a MVP2.0/3.0 input file with the latest values of the update variables written in `update_vars.nml` file and produces a `mvpinp.inp` input file. The *main* program then executes the *MVP2.0/3.0* program to calculate a time- and region-dependent fission reaction data.

After the *MVP2.0/3.0* program finishes, the *main* program executes the *forward/forward_mpi* program again. The *forward/forward_mpi* reads the necessary data from the related files that it wrote previously, calls the *cfijtally* subroutine to get newly calculated neutron transport time- and region-dependent fission reaction data, calculates the new kinetic functions, and continues advancing the solution of the IKM using the newly calculated/updated kinetic functions and calculating the update variables via the *feedback* subroutine.

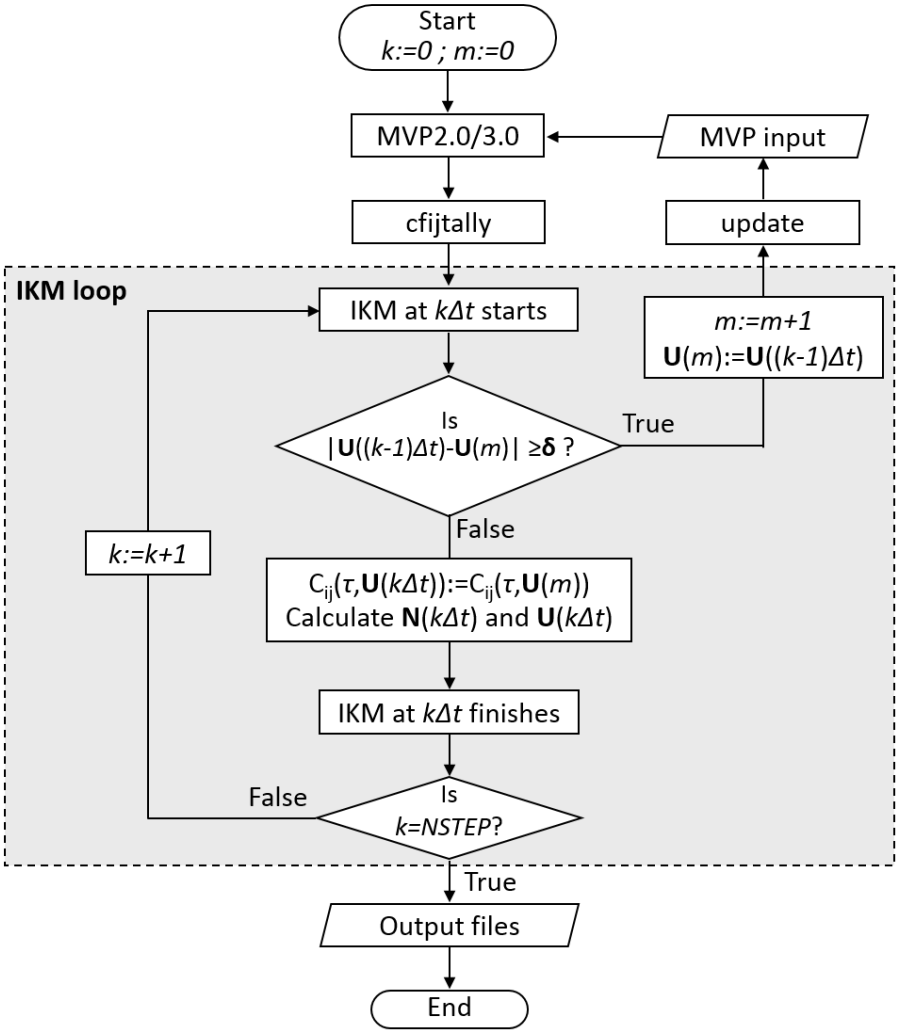

Fig. 3.2. Basic flowchart of MIK code

For more detailed description, Table 3.1 lists the brief definitions of the programs and subroutines.

Table 3.1. Brief definition of programs and subroutines

main	Main program to control the overall calculation flow of MIK code by executing the *forward*/or *forward_mpi*, *update*, and *MVP2.0/3.0* programs successively until the end of calculation.
forward/or *forward_mpi*	Program(s) based on the IKM. It advances a solution of the IKM (see Eq. (2.4)) in time using forward Euler method. In addition, this program stores necessary data to related

	files when it is terminated at each update-time-step by the *main* program and reads necessary data from them again when it is started again after each update-time-step by the *main* program.
MVP2.0/3.0	A continuous energy Monte Carlo neutron transport code used to calculate a necessary data for calculating the kinetic functions at each update-time-step.
cfijtally	Subroutine to read a time- and region-dependent fission reaction data from `mvpout.ft06` file and return a neutron transport time- and region-dependent cumulative fission reaction data to the *forward*/or *forward_mpi*. Called at each update-time-step.
feedback	Subroutine to calculate update and non-update variables at the current time-step using their values at the previous time-step via user-provided governing equations. Called from the *forward*/or *forward_mpi* at each time-step.
14 subroutines from LAPACK library [6]	Subroutines used for the calculation of initial power fraction needed for starting a transient. Called from the *forward*/or *forward_mpi* only at time-step 1 and only if number of regions is more than 2.
update	Program to update the user-prepared input file *MVP2.0/3.0* program with the latest values of update variables at each update-time-step and produce `mvpinp.inp` input file, which is actually used by *MVP2.0/3.0* program.
mod_values	Module containing all necessary declarations of constants, parameters, and variables for the MIK calculation. Read by multiple programs and subroutines.
mod_comp_subs	Module containing all necessary subroutines related to computation such as the *IKM*, *cfijtally*, and *reg_div* subroutines. (See the details from the definition at the beginning of each subroutine)
mod_util_subs	Module containing all necessary utility subroutines for reading and writing data such as the *fr_read*, *fr_write*, *cijp_read*, *cijp_write*, *uvar_read*, *uvar_write*, and *reac_insert* subroutines. (See the details from the definition at the beginning of each subroutine)

3.2. Setting up MIK code on Linux system

The current MIK code runs only on Linux system. Furthermore, the current MIK code was tested only on CentOS 6.2 Linux distribution.

In order to setup MIK code on Linux system, user must first have either MVP2.0 or MVP3.0 code installed with proper setup of its environment. The installation of MVP2.0/3.0 code is straightforward and can be found in its manual [7][4].

Once the MVP2.0/3.0 installation is done, then user can copy the source code of MIK code to any directory where the problem is to be executed, and that directory will be automatically set as `$MIK_DIR`. It is highly recommended that `$MIK_DIR` does not contain any file other than those necessary for the MIK calculation (see Table 3.3). In particular, `$MIK_DIR` must not contain a file named `update_steps.dat` unless the MIK calculation is a restart calculation. Once the source codes files are copied into `$MIK_DIR/src` directory, user needs to setup the MIK calculation for a problem in question by fixing the calculation parameters as will be discussed in section 3.3.

The current MIK code can run in single core mode or parallel mode. In case of the single core mode run, MIK code is compiled with either GNU Fortran (gfortran 5.1 or similar/higher) or

combination of both GNU Fortran (gfortran 5.1 or similar/higher) and Intel Fortran (ifort 12.1 or higher). The reason is that the *main* program is compiled with gfortran, while others can be compiled with either of gfortran or ifort. Furthermore, the reason of using Intel Fortran, despite it being commercial compiler, is that the IKM part of MIK code (i.e., the *IKM* subroutine) runs considerably faster when compiled with Intel Fortran.

In case of the parallel mode run, MIK code is compiled with GNU Fortran (gfortran 5.1 or similar/higher), which is for the *main* program as mentioned above, and Open MPI (v. 3.0.0 or higher) [8] installed with either Intel Fortran (ifort 12.1 or higher) or GNU Fortran (gfortran 5.1 or similar/higher), which is for the other programs. In addition, depending on the single or parallel mode run, user must have either single or parallel mode MVP2.0/3.0 code (see the manual also). It is highly recommended that MVP2.0/3.0 code is compiled with the same compiler that is used for compiling MIK code, particularly in case of the parallel mode run.

It should be noted here that there might be a problem when compiling MVP2.0 code with GNU Fortran (gfortran 5.1 or similar/higher), while MVP2.0 can be compiled with Intel Fortran (ifort 12.1 or higher) without any problem. Thus, user is encouraged to use MVP3.0 if user wants to compile MIK code using only GNU Fortran 5.1 or similar/higher (or Open MPI 3.0 installed with GNU Fortran 5.1 or similar/higher). Otherwise, user can use either MVP2.0 or MVP3.0 when compiling MIK code using a combination of GNU Fortran (5.1 or similar/higher) and Intel Fortran (12.1 or higher) (or Open MPI 3.0 or higher installed with Intel Fortran 12.1 or higher). Table 3.2 summarizes the above notes about the recommended compiler(s) for different cases depending on the choice of the single/parallel and MVP2.0/MVP3.0.

Table 3.2. Recommended compiler(s) for MIK code

Single or parallel	**MVP version**	**Recommended compiler(s)**
Single	MVP2.0	GNU Fortran GCC (5.1 or similar/higher) for *main* program; Intel Fortran (12.1 or higher) for other programs including MVP2.0
	MVP3.0	Option 1) GNU Fortran GCC (5.1 or similar/higher) for all programs Option 2) GNU Fortran GCC (5.1 or similar/higher) for *main* program; Intel Fortran (12.1 or higher) for other programs including MVP3.0
Parallel	MVP2.0	GNU Fortran GCC (5.1 or similar/higher) for *main* program; Open MPI 3.0 or higher installed with Intel Fortran (12.1 or higher) for other programs including MVP2.0
	MVP3.0	Option 1) GNU Fortran GCC (5.1 or similar/higher) for *main* program; Open MPI 3.0 or higher installed with GNU Fortran GCC (5.1 or similar/higher) for other programs including MVP3.0 Option 2) GNU Fortran GCC (5.1 or similar/higher) for *main* program; Open MPI 3.0 or higher installed with Intel Fortran (12.1 or higher) for other programs including MVP3.0

Once all of the above procedures are done and the requirements are met, then MIK code is ready to be compiled and run, which will be discussed in section 3.4. Before that, section 3.3 will discuss how to set up a MIK calculation for a given problem.

3.3. MIK calculation setup

In this section, setting up calculation parameters for a MIK calculation is discussed. Essentially, user needs to create an input file for MVP2.0/3.0 eigenvalue calculation with special time-dependent tally option, define calculation parameters for the IKM, and provide the governing equations for the update and non-update variables.

In general, user must make/or modify the files in `$MIK_DIR`, listed in Table 3.3.

Table 3.3. Files needed to be created/prepared/modified by user in `$MIK_DIR` directory

`XXXXX.inp`	MVP2.0/3.0 input file to be prepared by user. It is defined by `Inpname` parameter (see Table 3.4). The *update* program updates this file with the values of update variables at each update-time-step and produces `mvpinp.inp` file, which is actually used by MVP2.0/3.0 program.
`src/mod_values.f90`	Module containing data for MIK calculation. See more in section 3.3.2.
`src/main.f90`	Main program, in which few parameters related to number of parallel cores etc should be set. Modify only in case of parallel mode calculation. See more in section 3.3.2.
`src/mod_feedback.f90`	Module containing a feedback subroutine, in which the governing equations for the update and non-update variables must be defined. See more in section 3.3.3.
`reactivity.dat`	File containing reactivity insertion data. See more in section 3.3.2.

3.3.1. MVP2.0/3.0 eigenvalue calculation input data

The continuous energy Monte Carlo neutron transport code MVP2.0/3.0 can be used for basically two types of problems – an eigenvalue problem, in which a multiplication factor of a system is estimated, and a fixed source problem, in which essentially a shielding calculation is performed. In any case, an input file containing information about the given system such as geometry, composition, and temperature etc, and control parameters for neutron transport calculation must be prepared. An instruction for making a typical input file for MVP2.0/3.0 can be found in its manual [7].

As for MIK code, eigenvalue calculation of the MVP2.0/3.0 is needed. Using special-tally option in eigenvalue calculation by MVP2.0/3.0 code, it is possible to calculate a neutron transport time- and region-dependent fission reaction defined by Eq. (2.6). In order to do that, it is just necessary to add several options and data as instructed below into a typical MVP2.0/3.0 input file.

i. Add TIME-DEPENDENT option in the option block in the input file. This allows a time-dependent calculation. (see more at p.93 of the MVP2.0 manual, and p.113 of the MVP3.0 manual)

ii. Add time bin structure DT, NTIME, and TIMEB in the beginning of the TALLY data block defined between $TALLY and $END TALLY. This time bin structure is common to all special tally options (see more at p.205 of the MVP2.0 manual, and p.224 of the MVP3.0 manual). Note that the width of time bin DT must be chosen based on the considered system and it is also better to keep NTIME<10000, preferably NTIME=9999.

```
$TALLY
…
…
% DT=1.0D-6
 NWORK( 21000 )
 NTIME(9999)
 TIMEB( 0.0 <%NTIME>:<DT> )
```

iii. Add special tallies for a neutron transport time-and region-dependent fission reaction after the time bin structure in TALLY data block. For example, the following special tally will output fission reaction in Reg-1 by neutrons born in Reg-2 per time bin. These region names are the ones that are defined in the input data for zones in the MVP2.0/3.0 input file (see more in section 10.3 of the MVP2.0/3.0 manual).

```
&
 ID(1000)
 LABEL('Fission rate in Reg-1 by n from Reg-2')
 EVENT( COLLISION )
 PARTICLE( NEUTRON )
 DIMENSION( REGION SOURCE-REGION TIME )
 REGION( Reg-1 )
 SOURCE-REGION( Reg-2 )
 MACRO( FISSION )
 IENERGY( 70(1) )
 ITIME( 1 <%NTIME-1>:1 )
```

Total of R^2 special tally inputs are needed to represent full coupling between R fissile regions. More special tally inputs can be added by changing ID, LABEL, REGION, and SOURCE-REGION. Do not change the others. For instance, if user changes the order of DIMENSION, then *cijtally* subroutine of MIK code cannot read the MVP2.0/3.0 output file. More on these options can be found in the MVP2.0/3.0 manual.

Also, in order to reflect any changes in the characteristics of a given system to the kinetic functions during a transient, the MVP2.0/3.0 input file must contain a definition of update variables (see section 3.3.3.1) because, as described in section 3.1, the MVP2.0/3.0 input file is updated at each update-time-step with the latest values of the update variables. In specific, full names (i.e., base name (see Table 3.4) concatenated with region #) of all update variables for each region must be included in the input file. In a single row of the input file, only one update variable must be defined

i.e., no multiple update variables can be defined in a single row. In addition, each update variable must be prefixed by the characters defined by `Sign` parameter (see Table 3.4) from the start of the row. More on this topic will be provided in section 4.2 with example.

Finally, it should be noted here that MVP2.0 and MVP3.0 might produce slightly different results (e.g., k_{eff} etc) when run with the same input file. This difference can lead to a rather noticiable difference in the result of a transient calculation by MIK code. Thus, user should pay attention to this fact when reproducing the same MIK calculation elsewhere using different version of MVP code. Furthermore, even the same version of MVP code can give slightly different results depending on the single or parallel mode run, and in case of the parallel mode run the results can change slightly depending on a number of parallel cores too.

3.3.2. MIK calculation data

Essentially all data for a MIK calculation must be set in the `mod_values.f90` module, while only few data must be set in the `main.f90` program.

Table 3.4 lists all major parameters to be set in the `mod_values.f90` module with their brief descriptions, however, note that there are other auxiliary data, which are not listed in the table, related to the ones listed in the table. All of the data listed in the table are Fortran *parameter*, which means they are constant during the calculation. Also given in the square parentheses in the table is the type of data.

Table 3.4. List of major data to be set in `mod_values.f90` module

Basic IKM parameters	
`Nreg` *[Integer]*	Number of regions for IKM (R in Eq. (2.4))
`Nstep` *[Integer]*	Number of time-steps for IKM
`Dt` *[Double-precision]*	Time-step size for IKM [s]
`Kcutp` *[Integer]*	k_c for prompt neutron (see Eq. (2.4)). It must be the same with `NTIME` parameter in MVP2.0/3.0 input file (see section 3.3.1).
`Ipwr` *[Double-precision]*	Initial steady-state power [W] (Usually low or zero power, e.g., 1 W)
`Fise` *[Double-precision]*	Energy release per fission [MeV/fission]
Parameters for MVP special tally and couplings	
`Inpname` *[Character]*	Name of the input file for MVP2.0/3.0 code. An update program updates this file with values of update variables and produces `mvpinp.inp` file, which is actually used by MVP2.0/3.0 code.
`Ncoup` *[Integer]*	Number of full coupling (currently $\texttt{Nreg}^2$)
`Tid_list(Ncoup)` *[Character]*	Array containing special tally ID for each coupling. (refer to section 3.3.1). Length of ID should not exceed four digits.

`Coup_list(Ncoup)` *[Character]*	Array containing `REGION` & `SOURCE-REGION` coupling. Each element must correspond the each element of `Tid_list` array. (refer to section 3.3.1). Recommended length for `REGION` & `SOURCE-REGION` is three digits e.g., 001, 002, and 010 etc.
Parameters related to feedback, reactivity insertion, and *update* program	
`Nuvar` *[Integer]*	Number of update variables
`Uvarn(Nuvar)` *[Character]*	Array containing base names for all update variables. All base names have the same length. The full name of region-dependent update variable is made automatically via concatenating the base name and region number. (reminder: the full names of update variables also must be included in the MVP2.0/3.0 input file)
`Iuvar(Nreg, Nuvar)` *[Double-precision]*	Array containing the initial values of update variables in each region.
`Uvarcrit(Nreg,Nuvar)` *[Double-precision]*	Array containing update criterion for each region-dependent update variables. (see Eq. (2.13))
`Nnvar` *[Integer]*	Number of non-update variables. Non-update variables are useful, especially in the *feedback* subroutine.
`Nvarn(Nnvar)` *[Character]*	Array containing base names of all non-update variables. The full name of each non-update variable is made automatically via concatenating the base name and region number.
`Invar(Nreg,Nnvar)` *[Double-precision]*	Array containing the initial values of non-update variables.
`Nrvar` *[Integer]*	Number of reactivity insertion variables.
`Rvarn(Nrvar)` *[Character]*	Array containing full names of reactivity variables. All reactivity variables must be part of update variables too.
`Riend` *[Integer]*	Time-step corresponding to the end of reactivity insertion.
`Rfile` *[Character]*	Name of a file containing reactivity insertion data.
`Sign` *[Character]*	A sign indicating the input parameter in MVP2.0/3.0 input file. It is used to locate and update the update variables in the MVP2.0/3.0 input file. Essentially set it as `Sign="% "` (Note that there is a space after % symbol)
Parameters related to slightly adjusting k_{eff} and C_{ij} functions	
`K_adjust` *[Double-precision]*	A k_{eff} to which the initial k_{eff} obtained with MVP2.0/3.0 is to be adjusted. Useful when trying to make the initial reactivity exactly the same with experiment/ or other calculation.
`K_init` *[Double-precision]*	Actual k_{eff} at initial condition obtained by MVP2.0/3.0 calculation. This k_{eff} will be adjusted to the value defined by `K_adjust`.
Parameters related to outputting C_{ij} functions	
`Nout` *[Integer]*	Number of time-steps at which C_{ij} functions are to be outputted during a transient.
`Out_list(Nout)` *[Integer]*	Array containing time-steps at which C_{ij} functions are to be outputted.

In addition to the `mod_values.f90` module, as mentioned in section 3.3, user needs to prepare the `reactivity.dat` file that contains reactivity insertion data. In specific, this file contains reactivity insertion time and corresponding values of the reactivity variables. The following table shows an example of reactivity insertion data in the `reactivity.dat` file in case of a system with two regions and single reactivity variable for each region and with two reactivity insertions.

```
  Time, s    Value
```
← Header

```
0.00000D0   5.0000D0
0.00000D0   6.0000D0
0.00010D0   5.0400D0
0.00010D0   6.0200D0
```

← Value of the reactivity variable in region 1 at 0.0 sec
← Value of the reactivity variable in region 2 at 0.0 sec
← Value of the reactivity variable in region 1 at 0.0001 sec
← Value of the reactivity variable in region 2 at 0.0001 sec

In this example, reactivity is inserted twice, first at 0.0 sec and then at 1.0×10^{-4} sec by changing the values of the reactivity variable defined via `Rvarn(Nrvar)` parameter array (see Table 3.4). If there are more than one reactivity variable (i.e., `Nrvar > Nreg`), then data in the `reactivity.dat` file must follow specific order. Time must be in ascending order as the row goes down. At each time value, values of all reactivity variables must be given in the same order as in `Rvarn(Nrvar)` parameter array.

Finally, user needs to set just few parameters in the `main.f90` source file in case of parallel mode calculation. Table 3.5 shows the parameters needed to be set/modified, which are highlighted in **bold**. Essentially, user just needs to set `Nstep` again and a number of parallel cores for the IKM and MVP2.0/3.0 code. In the following example, `Nstep` is 1 million, which must be the same with that in the `mod_values.f90` module, and the IKM is run with 2 parallel cores (remember that number of parallel cores for the IKM should not exceed the number of regions `Nreg`) while MVP2.0/3.0 runs with 10 parallel cores.

Table 3.5. Data to be set in the `main.f90`

```
1  INTEGER, PARAMETER :: Nstep=1000000 ! Number of time steps
2  !
3  #ifdef MIK_MPI! -- For parallel calculation
4   INTEGER, PARAMETER :: Flen = 26    ! Command line length for FORWARD
   program run
5   INTEGER, PARAMETER :: Mlen = 167   ! Command line length for MVP
   program run
6   CHARACTER(len=Flen), PARAMETER ::  Fwd_cmd = "mpirun -np
   2 ./forward_mpi"  ! Command line for FORWARD run
7   CHARACTER(len=Mlen), PARAMETER ::  Mvp_cmd = "mpirun -np 10
   $MVP_DIR/bin/LINUXGLIBC/mvp.mpi /5rf:mvpinp.inp "// &
8   "/24rf:$MVPLIB_DIR/neutron.art.index /25rf:$MVPLIB_DIR/neutron.index
   /15f /16 /55f /75f /66f > mvpout.ft06"  ! Command line for MVP run
```

3.3.3. Time-dependent feedback model data

As discussed in section 2.3 before, user must provide the governing equations for the update and non-update variables. In specific, user needs to define the governing equations together with all necessary local variables and parameters in the *feedback* subroutine in the `mod_feedback.f90` module. In other words, user needs to write a Fortran subroutine.

The *feedback* subroutine is called at each time-step from the *forward/forward_mpi* program and receives a region-wise fission rate, region-wise update variables, and region-wise non-update

variables at the previous time-step. Using these data, then the *feedback* subroutine calculates the region-wise update variables and non-update variables at the current time-step based on the user-provided equations and returns them to the *forward/forward_mpi* program.

Since the choice of the update and non-update variables and their governing equations depend completely on the considered system and problem, they will not be discussed here. Instead, general discussions on the update variables and non-update variables are provided in this section. And a concrete example on choosing the update and non-update variables for a problem in question and defining the governing equations for them will be provided in section 4.4.

3.3.3.1. Update variables

The update variable is a variable that defines partly or wholly the state of the system (refer to section 2.3). A choice of update variables strongly depends on the considered system and problem, however, a region-wise temperature is the most fundamental update variable and must be always included in all kind of systems and problems because many physical characteristics are dependent on a temperature. In addition, in many problems a region-wise physical density is also necessary. For example, in supercritical transients, the fundamental feedback mechanisms are Doppler broadening, which is temperature-dependent, and thermal expansion, which is primarily density-dependent. If user wants to consider more feedback mechanisms, then user needs to include the corresponding variables as update variables.

The update variables are used to update the input file for MVP2.0/3.0 code to prepare the kinetic functions at the latest state of the system. More about the update variables and related governing equations will be discussed using an example in section 4.4.

3.3.3.2. Non-update variables

Non-update variable is a variable that does not define the state of the system or is not used to update the input file for MVP2.0/3.0 code, nevertheless might be used for calculating the update variable(s) and other important quantities. Perhaps most fundamental non-update variable could be region-wise energy release, beginning from the start of transient till the current time-step. Region-wise energy release obviously does not define the state of the system, but it might be required to calculate an update variable, which defines the state of the system. Even if it is not used to calculate the update variables or other important quantities, energy release itself is an important quantity to be obtained during a transient. Total energy release during a transient is very useful quantity, for example, for estimating radiation dose. A choice of the non-update variables depends heavily on the specific system and problem. More on the non-update variables will be discussed using an example in section 4.4.

3.4. Running MIK calculation on Linux system

Once all the data and equations are prepared in the corresponding modules and programs in `$MIK_DIR` and `$MIK_DIR/src` directories as briefly discussed in section 3.3, a MIK calculation is then ready to run.

User needs to execute the `mikrun.sh` shell script file in `$MIK_DIR` directory in command line to compile and run the MIK calculation using the following command in C shell (csh and tcsh).

```
chmod +x mikrun.sh
```

```
./mikrun.sh
```

The `mikrun.sh` shell script then prompts user to choose either single mode run or parallel mode run and a compiler. Upon choosing an appropriate option for running, then the `mikrun.sh` script compiles the source codes in `$MIK_DIR/src` directory and starts the calculation once the compilation finishes successfully by printing the following message to standard output stream (i.e., command line).

```
Calculation has started & is running... Please wait till it ends!
```

During the running of the calculation, MIK code outputs some important calculation logs into `log.txt` file and standard output stream. User can monitor the progress of the calculation via standard output stream and `log.txt` file.

If any runtime error that can be caught by MIK code occurs, it then prints the error messages to `log.txt` file and standard output stream, and stops. User can thus know the cause of the runtime error, however, note that not all runtime errors can be caught by MIK code itself, thus sometimes the calculation stops without an error message from MIK code. In that case, user should investigate an error message produced by the operation system on standard output stream.

Upon successful completion of the calculation, MIK code prints the following message indicating the calculation finished successfully to `log.txt` file and standard output stream.

```
The whole calculation finished successfully!
```

MIK code produces a number of files containing the calculation results in `$MIK_DIR` directory. Table 3.6 lists the files and their brief descriptions.

Table 3.6. Files produced by MIK code

File name	Description
`fr_dat.dat`	File containing region-wise fission rate profile data.
`XXXX_dat.dat`	Files containing region-wise update variables' profile data. XXXX is the name of update variable defined in `Uvarn(Nuvar)` parameter array (see Table 3.4).
`update_vars.nml`	Fortran namelist file containing latest values of the update variables.
`nonupdate_vars.nml`	Fortran namelist file containing latest values of the non-update variables.
`keff.dat`	File containing values of k_{eff}'s at the update-steps.

`update_steps.dat`	File containing update steps.
`cij_xxxxxxxxxx.dat`	File containing C_{ij} data at user specified step xxxxxxxxxx via `Out_list(Nout)` parameter array (see Table 3.4).
`log.txt`	Calculation log file.

3.5. Some post processing tools

MIK code outputs a time-dependent region-wise fission rate and update variables' data in their corresponding files. In general, user can make and use any post processing tools to visualize/plot and do some calculation using the calculation data from MIK code. However, user should note that since the number of time-steps is usually on the order of 10^5 - 10^6 in typical supercritical transient calculation, plotting time-dependent calculation output data using spreadsheet software such as MS Excel or Origin is not a good choice.

To provide user with simple and reasonable tools for post processing, MIK code included an auxiliary simple programs `fr_plotter.py`, `uvar_plotter.py`, and `peak_info.py` written in Python language. In order to use these auxiliary simple programs, user just needs to have Python (version 3) installed with necessary numeric libraries Numpy and matplotlib. Instructions on installing these libraries can be found online from their official documents. The `fr_plotter.py` plots region-wise fission rate profile, `uvar_plotter.py` plots region-wise update variable profile, and `peak_info.py` calculates the quantities such as initial reactor period, FWHM, peak time, total energy release etc. User needs to set some parameters in each program, however, they are minor and descriptions of the parameters are given inside the program.

4. SAMPLE CALCULATION

In this section, a sample calculation of a supercritical transient in simple Godiva core is provided to illustrate more about setting up MIK calculation through an example.

4.1. Sample problem

Godiva core is a nearly spherical, highly-enriched, fast spectrum, metal uranium core that was assembled at Los Alamos Scientific Laboratory in the 1950s. A series of supercritical transient experiments with step-wise reactivity insertion had been performed at the Godiva facility [9].

The supercritical transient experiment with the highest reactivity insertion of about 8.12¢ performed at the Godiva facility is chosen as a sample problem for the sample calculation of MIK code. In Godiva core, a major feedback mechanism is a thermal expansion due to fission energy deposit, while Doppler broadening feedback is almost negligible because of its fast spectrum. Furthermore, there is delay in thermal expansion due to shock wave in case of the supercritical experiments with high reactivity insertion.

In the simulation, Godiva core is divided into inner sphere and outer concentric sphere as illustrated in Fig. 4.1. A step-wise reactivity is inserted via sudden increase in outer radius. The step-wise reactivity insertion of 8.12¢ corresponds to the outer radius of about 8.85 cm. As discussed above, essentially there are three feedback mechanisms in the selected Godiva supercritical transient. In the simulation, however, only the thermal expansion and Doppler broadening feedbacks are considered, albeit Doppler broadening being almost negligible.

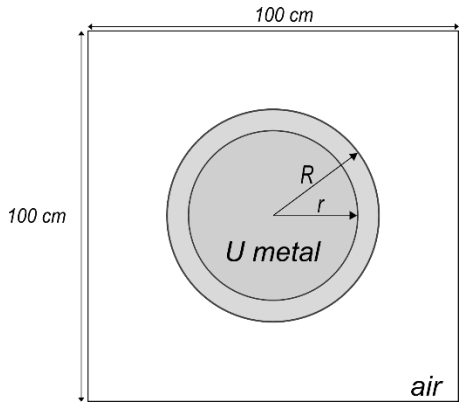

Fig. 4.1. Illustration of Godiva core in MIK calculation

4.2. MVP2.0/3.0 eigenvalue calculation input

Table 4.1 gives an input for Godiva core for MVP2.0/.0 code. This input was made according to the MVP2.0/3.0 input instructions. As described in section 3.3.1, MVP2.0/3.0 input file must

contain the definitions of the update variables and special tally options. In Table 4.1, the update variables are highlighted in **bold**. In specific, for the Godiva supercritical simulation, a region-wise temperature, density, and radius are the update variables. Notice that all update variables are set arbitrarily to zero, which is fine because the MVP2.0/3.0 input file is updated with the values of the update variables at each update-time-step.

Furthermore, also as described in section 3.3.1, the MVP2.0/3.0 input file must contain the definitions of special tally options. As for Godiva supercritical simulation with two regions, total of 2^2 special tally inputs are needed and given at the end of input in Table 4.1, labeled with IDs 1000 through 1003.

Table 4.1. `Godiva.inp` input file

```
SIMPLE GODIVA CORE
SUPERCRITICAL TRANSIENT
************************************************************************
* Options
************************************************************************
  NEUTRON
  DYNAMIC-MEMORY(530000000)
  FISSION
  DELAYED-NEUTRON
  NO-FIXED-SOURCE
  EIGEN-VALUE
  FISSION-MULTIPLICITY( PROMPT )
  RUSSIAN-ROULETTE
  NO-WEIGHT-WINDOW
  NO-EDIT-MICROSCOPIC-DATA(40400000)
  EDIT-MACROSCOPIC-DATA(40400000)
  TIME-DEPENDENT

************************************************************************
* Control data for MVP calculation
************************************************************************
% NHIST  = 2500000           /* number of neutrons in a batch
% KBATCH = 50                /* number of batches for tallies
% NSKIP  = 50                /* number of batches before tallies
% TCPU   = 0
% IRAND  = 62          /* initial random number
% NBATCH = KBATCH + NSKIP    /* total number of batches
% NG = 70


  NGROUP( <NG> )
  NMEMO( 5 )
  TCPU( <TCPU> )
  NPART( <NBATCH*NHIST> )
  NHIST( <NHIST> )
  NSKIP( <NSKIP> )
  NBANK( <INT(1.250*NHIST)> )
  NFBANK( <NHIST> )
  ETOP( 2.0E+7 )
  EBOT( 1.0E-5 )
```

```
  EWCUT( 275.0 )
  ETHMAX( 4.5 )
  AMLIM( 300.0 )
  IRAND(<IRAND>)

*************************************************************************
* Cross section or material composition data
*************************************************************************
$XSEC

% TEMP1 = 0.0           /* temp of Inner
% TEMP2 = 0.0           /* temp of Outer
% TEMP3 = 298.15           /* temp of Air

% NA      = 6.022E+23                    /* Avogadro number
% ENR35   = 0.9371                        /* U235 enrichment in fraction
% ENR34   = 0.01
% DENS1 = 0.0                        /* density of Inner
% DENS2 = 0.0                        /* density of Outer
% AMASS34  =234.0410
% AMASS35  =235.0439
% AMASS38  =238.0508
% N341= ENR34*DENS1*NA/AMASS34*1.0E-24   /* Atom density of U234 [1.0E24
atom/cm3]
% N351 =ENR35*DENS1*NA/AMASS35*1.0E-24   /* Atom density of U235
% N381 =(1.0-ENR35-ENR34)*DENS1*NA/AMASS38*1.0E-24  /* Atom density of U238
% N342= ENR34*DENS2*NA/AMASS34*1.0E-24        /* Atom density of U234
% N352 =ENR35*DENS2*NA/AMASS35*1.0E-24        /* Atom density of U235
% N382 =(1.0-ENR35-ENR34)*DENS2*NA/AMASS38*1.0E-24  /* Atom density of U238

*---------------------MATERIAL ID-----------------------*

& IDMAT(1)                        /* solid
  TEMPMT( <TEMP1> )              /* Kelvin
  U02340J40( <N341>    <TEMP1> )  /* U234
  U02350J40( <N351>    <TEMP1> )  /* U235
  U02380J40( <N381>    <TEMP1> )  /* U238

& IDMAT(2)                        /* solid
  TEMPMT( <TEMP2> )              /* Kelvin
  U02340J40( <N342>    <TEMP2> )  /* U234
  U02350J40( <N352>    <TEMP2> )  /* U235
  U02380J40( <N382>    <TEMP2> )  /* U238

& IDMAT(3)      /* air
  TEMPMT( <TEMP3> )
  N00140J40( 3.9016E-5 <TEMP3>)
  O00160J40( 1.0409E-5 <TEMP3>)

$END XSEC

*************************************************************************
* Geometry data
*************************************************************************
$GEOMETRY

% RADI1 = 0.0       /* inner radius
```

```
% RADI2 = 0.0       /* outer radius

*** BODIES ***

 RPP ( 15  -50.0 50.0 -50.0 50.0 -50.0 50.0 )

 SPH ( 10    0.0       0.0    0.0  <RADI2>)          /* Outer radius sphere

 SPH ( 21    0.0       0.0    0.0  <RADI1>)          /* Inner radius sphere

END

**** ZONE ****

 Vac   :         : -1000 : -15
 Air   : AIR     : 3   : 15 -10
 Inner : INNER  : 1   : +21
 Outer : OUTER  : 2   : 10 -21

$END GEOMETRY

*** INITIAL SOURCE ***

$SOURCE
&
  NEUTRON
  RATIO( 1.0 )
  @E = #FISSION(U0235* 1.0E+5);
  @(X Y Z) = #SPHERE( 0.0 <RADI2> );
$END SOURCE

*** VARIANCE REDUCTION PARAMETERS ***

  WKIL( !INNER( <NG>(0.001) ) )
  WKIL( !OUTER( <NG>(0.001)))
  WSRV( !INNER( <NG>(0.1) ) )
  WSRV( !OUTER( <NG>(0.1) ) )

*** RESPONCE PARAMETERS ***
*  NRESP(1)
*  RESP( <NG>(1.0) )

*** FISSION NEUTRON GENERATION ***
  WGTF( !INNER(0.8)  )
  WGTF( !OUTER(0.8)  )

$TALLY
*** TALLY ENERGY GROUP ***
 ENGYB(
  2.000E+7  7.788E+6  6.065E+6  4.723E+6  3.678E+6  2.865E+6  2.231E+6
  1.737E+6  1.353E+6  1.054E+6  8.208E+5  6.392E+5  4.978E+5  3.877E+5
  3.019E+5  2.351E+5  1.831E+5  1.426E+5  1.110E+5  8.651E+4  6.737E+4
  5.247E+4  4.086E+4  3.182E+4  2.478E+4  1.930E+4  1.503E+4  1.170E+4
  9.118E+3  7.101E+3  5.530E+3  4.307E+3  3.354E+3  2.612E+3  2.034E+3
  1.584E+3  1.234E+3  9.611E+2  7.485E+2  5.829E+2  4.540E+2  3.535E+2
  2.753E+2  2.144E+2  1.670E+2  1.300E+2  1.013E+2  7.889E+1  6.144E+1
  4.785E+1  3.726E+1  2.902E+1  2.260E+1  1.760E+1  1.371E+1  1.067E+1
```

```
  8.315E+0  6.476E+0  5.043E+0  3.927E+0  3.059E+0  2.382E+0  1.855E+0
  1.445E+0  1.125E+0  8.764E-1  6.825E-1  5.315E-1  4.139E-1  3.224E-1
  1.000E-5 )
% NR = %NREG
% NRG = NR*NG
% DT=1.0D-10
 NWORK( 21000 )
 NTIME(10000)
 TIMEB( 0.0 <%NTIME-1>:<DT> 1.0E-3 )
&
 ID(1000)
 LABEL('Fission n in INNER by n from INNER')
 EVENT( COLLISION )
 PARTICLE( NEUTRON )
 DIMENSION( REGION SOURCE-REGION TIME )
 REGION( INNER )
 SOURCE-REGION( INNER )
 MACRO( FISSION )
 IENERGY( 70(1) )
 ITIME( 1 <%NTIME-1>:1 )
&
 ID(1001)
 LABEL('Fission n in INNER by n from OUTER')
 EVENT( COLLISION )
 PARTICLE( NEUTRON )
 DIMENSION( REGION SOURCE-REGION TIME )
 REGION( INNER )
 SOURCE-REGION( OUTER )
 MACRO( FISSION )
 IENERGY( 70(1) )
 ITIME( 1 <%NTIME-1>:1 )
&
 ID(1002)
 LABEL('Fission n in OUTER by n from INNER')
 EVENT( COLLISION )
 PARTICLE( NEUTRON )
 DIMENSION( REGION SOURCE-REGION TIME )
 REGION( OUTER )
 SOURCE-REGION( INNER )
 MACRO( FISSION )
 IENERGY( 70(1) )
 ITIME( 1 <%NTIME-1>:1 )
&
 ID(1003)
 LABEL('Fission n in OUTER by n from OUTER')
 EVENT( COLLISION )
 PARTICLE( NEUTRON )
 DIMENSION( REGION SOURCE-REGION TIME )
 REGION( OUTER )
 SOURCE-REGION( OUTER )
 MACRO( FISSION )
 IENERGY( 70(1) )
 ITIME( 1 <%NTIME-1>:1 )

$END TALLY
/
```

4.3. MIK calculation data

As described in section 3.3.2, a basic transient data for MIK calculation should be set in the `mod_values.f90`, while very few data should be set in the `main.f90` program. In addition to that, the governing equations must be provided in *feedback* subroutine in `mod_feedback.f90` module as will be shown in section 4.4.

Table 4.2 shows the contents of the `mod_values.f90`. For the definitions of the data in Table 4.2, refer to section 3.3.2.

Table 4.2. Data in the `mod_values.f90` module

```
!----------------- BEGINNING OF USER-DEFINED PARAMETERS----------------
-
! -- Basic IKM parameters --
  INTEGER(KIND=4), PARAMETER :: Nreg=2
  INTEGER(KIND=4), PARAMETER :: Nstep=6000000
  REAL(KIND=KIND(1.D0)), PARAMETER ::  Dt=1.0D-10
  INTEGER(KIND=4), PARAMETER :: Kcutp=9999
  INTEGER(KIND=4), PARAMETER :: Ipwr=1.0D0
  REAL(KIND=KIND(1.D0)), PARAMETER :: Mevtoj=1.6021D-13
  REAL(KIND=KIND(1.D0)), PARAMETER :: Fise=1.78D2
!
! -- Parameters for MVP special tally and couplings --
  CHARACTER(len=10), PARAMETER :: Inpname='Godiva.inp'
  INTEGER(KIND=4), PARAMETER :: Ncoup=4
  INTEGER(KIND=4), PARAMETER :: Ilen=4
  INTEGER(KIND=4), PARAMETER :: Plen=4
  CHARACTER(len=PLEN), PARAMETER :: Prefix="ID= "
  INTEGER(KIND=4), PARAMETER :: Klen=46
  CHARACTER(len=KLEN), PARAMETER ::  Cond1="RESULTS BY THE PRINCIPLE OF
MAXIMUM LIKELIHOOD"
  CHARACTER(len=ILEN), PARAMETER :: Tid_list(Ncoup) =
(/"1000","1001","1002","1003"/)
  INTEGER(KIND=4), PARAMETER :: Clen=7
  CHARACTER(len=CLEN), PARAMETER :: Coup_list(Ncoup) =
(/"001,001","001,002","002,001","002,002"/)
!
! -- Parameters related to feedback, reactivity insertion, and UPDATE
program --
!  - For update variables
  REAL(KIND=KIND(1.D0)), PARAMETER ::
Mass(Nreg)=(/2.716755D1,2.716755D1/)
  INTEGER(KIND=4), PARAMETER :: Nuvar=3
  INTEGER(KIND=4), PARAMETER :: Uvarnl = 4
  CHARACTER(len=Uvarnl), PARAMETER ::
Uvarn(Nuvar)=(/"TEMP","DENS","RADI"/)
  INTEGER(KIND=4), PARAMETER :: Uvar_step_read = 1
  REAL(KIND=KIND(1.D0)), PARAMETER ::
Iuvar(Nreg,Nuvar)=(/2.98D2,2.98D2,1.871D1,1.871D1,7.0247D0,8.8506D0/)
  REAL(KIND=KIND(1.D0)), PARAMETER ::
Uvarcrit(Nreg,Nuvar)=(/3.0D0,3.0D0,5.0D-2,5.0D-2,5.0D-3,5.0D-3/)
! -- For non-update variables
  INTEGER(KIND=4), PARAMETER :: Nnvar = 1
```

```
  INTEGER(KIND=4), PARAMETER :: Nvarnl = 4
  CHARACTER(len=Nvarnl), PARAMETER :: Nvarn(Nnvar)=(/"ENER"/)
  REAL(KIND=KIND(1.D0)), PARAMETER :: Invar(Nreg,Nnvar)=(/0.D0,0.D0/)
!
!  - For reactivity inserion
  INTEGER(KIND=4), PARAMETER :: Nrvar=2
  INTEGER(KIND=4), PARAMETER :: Rvarnl = 5
  CHARACTER(len=Rvarnl), PARAMETER :: Rvarn(Nrvar)=(/"RADI1","RADI2"/)
  INTEGER(KIND=4), PARAMETER :: Riend = 5
  INTEGER, PARAMETER :: Lrfile = 14
  CHARACTER(len=Lrfile), PARAMETER :: Rfile = "reactivity.dat"
!  - For UPDATE program
  INTEGER(KIND=4), PARAMETER :: Dlen=5
  CHARACTER(len=Dlen) :: num
  CHARACTER(len=MAX(Uvarnl,Rvarnl)+Dlen) :: para
  INTEGER(KIND=4), PARAMETER :: Llen=73
  CHARACTER(len=2), PARAMETER :: Sign='% '
  INTEGER(KIND=4), PARAMETER :: Vlen=11
  CHARACTER(len=8), PARAMETER :: Val_fmt="(ES11.4)"
!
! -- Parameters used for slightly adjusting [keff] and C_ij functions
(used to make exact comparison with other calculation) --
  REAL(KIND=KIND(1.D0)), PARAMETER :: K_adjust=1.000546D0
  REAL(KIND=KIND(1.D0)), PARAMETER :: K_init=1.00055D0
!
! -- Parameters related to outputting C_ij functions --
  INTEGER(KIND=4), PARAMETER :: Nout=1
  INTEGER(KIND=4), PARAMETER :: Out_list(Nout)=(/2500000/)
  INTEGER(KIND=4), PARAMETER :: Nlen=10
  CHARACTER(len=Nlen) :: fnum
  CHARACTER(len=8), PARAMETER :: Cij_fmt="(I10.10)"
!
!------------------ END OF USER-DEFINED PARAMETERS ------------------
```

It should be noted here that this sample calculation was performed using MVP2.0 code in parallel mode. It was noted before in section 3.3.1 that MVP2.0 and MVP3.0 codes might give slightly different results even if the input file is exactly the same. Thus, if user wants to use MVP3.0 code, user needs to change at least the `K_init` parameter (see Table 4.2) in order to obtain the same reactivity with the actual experiment.

Table 4.3 presents the parameters related to number of parallel cores for the IKM and MVP2.0/3.0 calculations set in the `main.f90` program.

Table 4.3. Data in the `main.f90` program

```
!--------------- BEGINNING OF USER-DEFINED PARAMETERS --------------
!
 INTEGER, PARAMETER :: Nstep=6000000 ! Number of time steps
!
#ifdef MIK_MPI
! -- For parallel calculation
 INTEGER, PARAMETER :: Flen = 26    ! Command line length for FORWARD
program run
```

```
 8  INTEGER, PARAMETER :: Mlen = 167   ! Command line length for MVP
    program run
 9  CHARACTER(len=Flen), PARAMETER ::  Fwd_cmd = "mpirun -np
    2 ./forward_mpi"  ! Command line for FORWARD run
10  CHARACTER(len=Mlen), PARAMETER ::  Mvp_cmd = "mpirun -np 10
    $MVP_DIR/bin/LINUXGLIBC/mvp.mpi /5rf:mvpinp.inp "// &
11  "/24rf:$MVPLIB_DIR/neutron.art.index /25rf:$MVPLIB_DIR/neutron.index
    /15f /16 /55f /75f /66f > mvpout.ft06"  ! Command line
12  ! for MVP run
```

4.4. Time-dependent feedback model data

The governing equations for the update and non-update variables for the time-dependent feedback model are set in the *feedback* subroutine in the `mod_feedback.f90` module. The content of the *feedback* subroutine for the Godiva supercritical transient is given in Table 4.4.

Table 4.4. Content of the *feedback* subroutine

```
 1  SUBROUTINE FEEDBACK(frprev,uvarprev,nvarprev,uvarcur,nvarcur,estat)
 2  !=====================================================================
 3  ! << Subroutine to calculate update variables at each time step >>
 4  !
 5  ! CALLED FROM: FORWARD
 6  !
 7  ! This subroutine takes the values of update & non-update variables at
 8  previous time step [step-1], calculates
 9  ! their step-wise changes, and returns the values of update & non-
    update variables at current time step [step].
10  !
11  ! Written in original form by: Delgersaikhan Tuya (Feb 2018)
12  ! Updated on: Aug 2018 by Delgersaikhan TUYA
13  !=====================================================================
14  !
15  USE mod_values, ONLY: Nreg, Nuvar, Nnvar, Dt, Capa, Mass, Iuvar,
    Mevtoj, Fise
16  !
17  REAL(KIND=KIND(1.D0)), INTENT(IN), DIMENSION(Nreg) :: frprev ! Region-
    wise fission rate at previous step [step-1]
18  REAL(KIND=KIND(1.D0)), INTENT(IN), DIMENSION(Nreg,Nuvar) :: uvarprev !
    Update variable at previous step [step-1]
19  REAL(KIND=KIND(1.D0)), INTENT(IN), DIMENSION(Nreg,Nnvar) :: nvarprev !
    Non-update variable at previous step [step-1]
20  REAL(KIND=KIND(1.D0)), INTENT(OUT),DIMENSION(Nreg,Nuvar) :: uvarcur  !
    Update variable at current step [step]
21  REAL(KIND=KIND(1.D0)), INTENT(OUT),DIMENSION(Nreg,Nnvar) :: nvarcur  !
    Non-update variable at current step [step]
22  INTEGER, INTENT(OUT) :: estat   ! Error status (0 means no error)
23  !
24  ! -- Local variables
25  INTEGER :: i, v
26  REAL(KIND=KIND(1.D0)), DIMENSION(nreg) :: vrat, vol, vol0
27  REAL(KIND=KIND(1.D0)), PARAMETER :: Pi=3.1415D0
```

```
 28   REAL(KIND=KIND(1.D0)), PARAMETER :: Au=235.201D0
 29   REAL(KIND=KIND(1.D0)), PARAMETER :: Ac=28.4264D0, Bc=-6.9587D-3,
      Cc=2.98744D-5, Ec=-1.1888D5
 30   REAL(KIND=KIND(1.D0)), PARAMETER :: Av=3.203D-5, Bv=3.287D-8,
      Cv=1.625D-11
 31   !
 32   ! -- Set estat = 0 for now --
 33   estat = 0
 34   !
 35   vol0(1) = 4.0/3.0*Pi*Iuvar(1,3)**3
 36   vol0(2) = 4.0/3.0*Pi*Iuvar(2,3)**3 - vol0(1)
 37   DO I=1, Nreg
 38    vrat(i) = 1.0D0 + Av*(uvarprev(i,1)-2.98D2) + Bv*(uvarprev(i,1)-
      2.98D2)**2 &
 39            + Cv*(uvarprev(i,1)-2.98D2)**3
 40    vol(i) = vrat(I)*vol0(i)
 41   END DO
 42   !
 43   DO i=1, Nreg
 44    nvarcur(i,1) = nvarprev(i,1) + Mevtoj*Fise*frprev(i)*Dt
 45   END DO
 46   !
 47   DO i=1, Nreg
 48    Capa(i) = Mass(i)*(1.0/(Au*1.0D-3))* &
 49                  ( Ac + Bc*uvarprev(i,1) + Cc*uvarprev(i,1)**2 &
 50                  + Ec*uvarprev(i,1)**(-2) )
 51    uvarcur(i,1) = uvarprev(i,1) + Mevtoj*Fise*frprev(i)*Dt/Capa(i)   !
      Temp
 52    uvarcur(i,2) = (1.0D3)*Mass(i) / vol(i)
 53   END DO
 54   uvarcur(1,3) = ( 3.0/4.0/Pi*vol(1) )**(1.0/3)
 55   uvarcur(2,3) = ( 3.0/4.0/Pi*( vol(1)+vol(2) ) )**(1.0/3)
 56   !
 57   END SUBROUTINE
```

In the *feedback* subroutine above, the non-update variable nvar(i,1) is region-wise energy release, while the update variables uvar(i,1), uvar(i,2), and uvar(i,3) are region-wise temperature, density, and radius, respectively. The region-wise energy release is calculated simply from

$$\frac{dE_i}{dt} = \varepsilon\omega N_i(t) \ , \tag{4.1}$$

where E_i is energy release in region i (J), ε is an energy release per fission (178 MeV/f), and ω is an energy conversion factor (1.6×10^{-13} J/MeV).

Since heat loss from the system is ignored, the region-wise temperature is kept increasing as the transient progresses and is calculated from

$$\frac{dT_i}{dt} = \frac{\varepsilon\omega N_i(t)}{C_i(T_i)} \ , \tag{4.2}$$

where T_i is the temperature in region i (K) and C_i is the temperature-dependent heat capacity in region i (J/K). The specific heat of uranium metal can be found in the reference [10].

The region-wise density (g/cm^3) is calculated simply as

$$\rho_i = \frac{m_i}{V_i} 10^3 \quad , \tag{4.3}$$

where m_i is the region-wise mass (kg) and is constant. On the other hand, the region-wise volume changes (i.e., volumetric thermal expansion) due to a change in the temperature and is given in the reference [11] as

$$V_i(T_i) = V_i(25)(1 + 32.03 \times 10^{-6} T_i + 32.87 \times 10^{-9} T_i^2 + 16.25 \times 10^{-12} T_i^3) \ , \tag{4.4}$$

where $V_i(25)$ is the volume (cm^3) at the initial condition of 25°C in region *i* and the unit of temperature is Celsius degrees.

Finally, the region-wise radius (cm) is calculated from

$$r_i = \sqrt[3]{\frac{3 \sum_{r=1}^{i} V_r}{4\pi}} \ . \tag{4.5}$$

4.5. Running the calculation and post processing the results

After setting all necessary data described in the previous sections, user can run the MIK calculation for Godiva supercritical transient by executing `mikrun.sh` shell script in `$MIK_DIR` directory from the command line as described in section 3.4. Before running the MIK calculation, user should make sure there is enough available disk space as the calculation results of the MIK calculation requires disk space on the order of 10^0-10^1 GB.

```
chmod +x mikrun.sh
```

```
./mikrun.sh
```

At the start of the execution, it prompts user to choose single or parallel mode calculation and a compiler. In this example, we used parallel mode run using Intel Fortran (ifort 12.1.5) by choosing an option 4 (in this case we compiled MVP2.0 parallel code using Open MPI 3.0 installed with ifort 12.1.5 as instructed in section 3.2). If the compilation is successful, then the MIK calculation starts running by notifying/printing the following message to standard output stream as described in section 3.4.

```
$ Calculation has started & is running... Please wait till it ends!
```

This MIK calculation for the Godiva transient takes about 3 hours on a system with Intel(R) Xeon(R) CPU E5-2690 with 2.8 GHz and 20MB cache memory. If the MIK calculation finishes successfully, it prints the following message to standard output and `log.txt` file.

```
The whole calculation finished successfully!
```

At this time, the calculation must have produced calculation result files such as `fr_dat.dat`, `TEMP_dat.dat`, `DENS_dat.dat`, `RADI_dat.dat`, `update_vars.nml`, and `nonupdate_vars.nml` etc as discussed in section 3.4.

Using the some post processing tools provided, user can obtain the following figures and results. Fig. 4.2 shows the region-wise fission rate profile produced via the `fr_plotter.py` program.

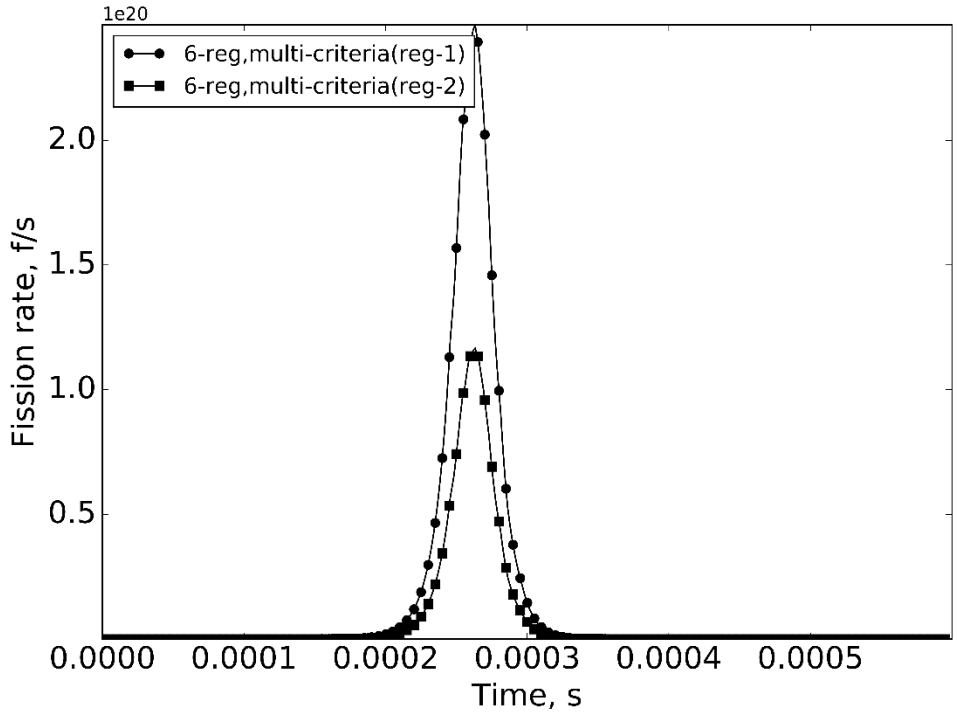

Fig. 4.2. Region-wise fission rate profile

Table 4.5 presents the content of the `result.dat` file produced by the `peak_info.py` program.

Table 4.5. Contents of `result.dat` file

```
For 2-reg Godiva file:
Peak is (f/s):3.63036928e+20
Peak power is (W):10352862029.8
FWHM is (s):3.09929e-05
Inverse period is (s-1):91082.3025186
Total fission is (f):1.29928468418e+16
Total energy release is (J):370521.950669
Peak sigma (%):0.886164227098
Peak_p sigma (%):0.886164227098
Total fission sigma percent(%):0.00119895390066
Total energy release sigma (%):0.00119895390066
```

Table 4.6 gives the values of the update and non-update variables at the end of calculation, given in the `update_vars.nml` and `nonupdate_vars.nml` files.

Table 4.6. Update and non-update variable values

Update/non-update variables	Reg-1	Reg-2
Temperature [K]	373.0	334.0
Density [g/cm^3]	18.66	18.69
Radius [cm]	7.031	8.856
Energy [J]	2.51466×10^5	1.19056×10^5

User can also plot the region-dependent update variables' profiles using the `uvar_plotter.py` program. Fig. 4.3 and 4.4 show the region-wise temperature and density profiles, respectively.

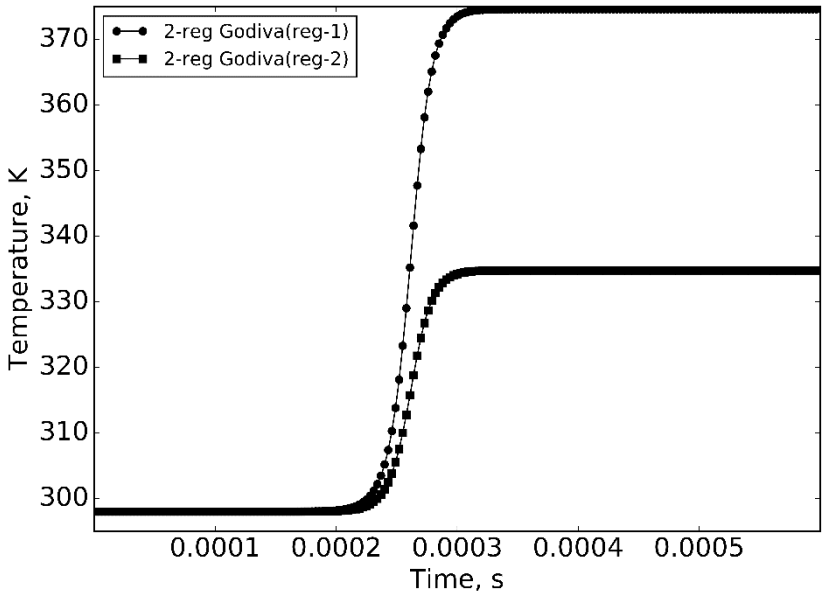

Fig. 4.3. Region-wise temperature profile

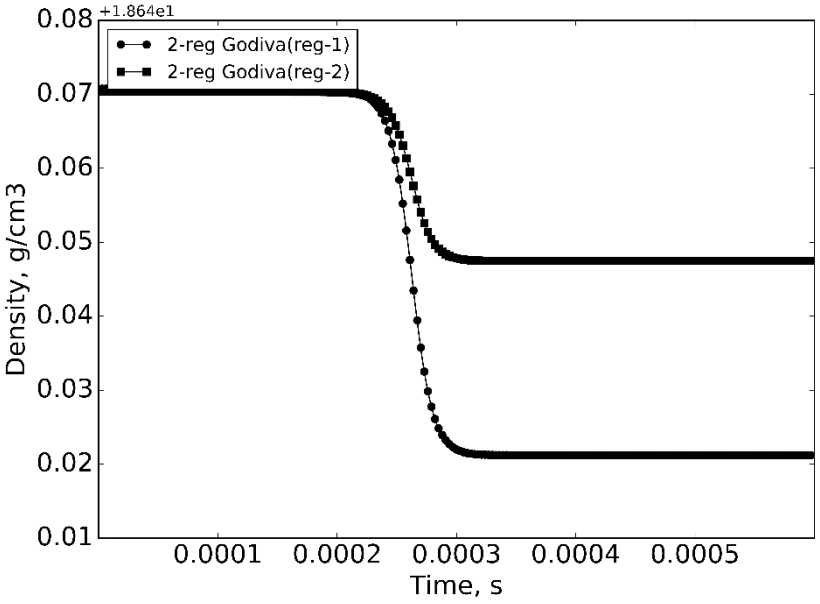

Fig. 4.4. Region-wise density profile

REFERENCES

1. Gulevich A V., Kukharchuk OF. Methods for calculating coupled reactor systems. At. Energy [Internet]. 2004 Dec;97:803–811. Available from: http://link.springer.com/10.1007/s10512-005-0066-0.

2. Takezawa H, Obara T. New Approach to Space-Dependent Kinetic Analysis by the Integral Kinetic Model. Nucl. Sci. Eng. [Internet]. 2012 May;171:1–12. Available from: https://www.tandfonline.com/doi/full/10.13182/NSE09-59.

3. Nagaya Y, Okumura K, Mori T, Nakagawa M. MVP/GMVP II: General Purpose Monte Carlo Codes for Neutron and Photon Transport Calculations Based on Continuous Energy and Multigroup Method. JAERI--1348. Japan At. Energy Res. Inst. Japan Atomic Energy Research Institute; 2005. p. 412. .

4. Nagaya Y, Okumura K, Sakurai T, Mori T. MVP/GMVP Version 3 : General Purpose Monte Carlo Codes for Neutron and Photon Transport Calculations Based on Continuous Energy and Multigroup Methods. Report No. JAEA-Data/Code 2016-018. 2016. .

5. Tuya D, Takezawa H, Obara T. Improved approach to multiregion supercritical transient analysis based on the integral kinetic model and Monte Carlo method. Nucl. Sci. Eng. 2017;188:33–42. .

6. Anderson E, Bai Z, Bischof C, Blackford S, Demmel J, Dongarra J, Du Croz J, Greenbaum A, Hammarling S, McKenney A, Sorensen D. LAPACK Users' Guide, 3rd edition. Society for Industrial and Applied Mathematics; 1999. .

7. Nagaya Y, Okumura K, Mori T, Nakagawa M. MVP/GMVP II: General Purpose Monte Carlo Codes for Neutron and Photon Transport Calculations Based on Continuous Enery and Multigroup Method. Report No. JAERI-1348. 2005. .

8. Gabriel E, Fagg GE, Bosilca G, Angskun T, Dongarra JJ, Squyres JM, Sahay V, Kambadur P, Barrett B, Lumsdaine A, Castain RH, Daniel DJ, Graham RL, Woodall TS. Open MPI: Goals, concept, and design of a next generation MPI implementation. 11th Eur. PVN/MPI Users' Gr. Meet. Budapest, Hungary; 2004. .

9. Wimett TF, Engle LB, Graves, Jr. GR, Orndoff JD. Time behavior of Godiva through prompt critical, Report No. LA-2029. 1956. .

10. Konings JM, Benes O, Griveau J-C. The actinides elements: Properties and characteristics. Compr. Nucl. Mater. Vol. 2. Elsevier Ltd.; 2012. .

11. Gurinsky DH, Dienes GJ. Nuclear Fuels. Geneva Ser. Peac. use At. energy. D. Van Nostrand Inc.; 1956. .

SOURCE LIST OF MIK CODE

1. *main.f90* program

```
PROGRAM MAIN
!
!=====================================================================================================================
!
! Program: MAIN
! Executes: FORWARD_MPI/or FORWARD, UPDATE, and MVP2.0/or MVP3.0
!
! This program executes FORWARD_MPI/or FORWARD, MVP2.0/or MVP3.0, and UPDATE programs successively until the
! end of whole calculation.
!   - This program is compiled with GNU Fortran in general (it was tested with gfortran 5.1.0). The reason is
!     due to lack of EXECUTE_COMMAND_LINE subroutine in Intel Fortran IFORT 12.1 or earlier versions. If you
!     have a version of IFORT that supports that subroutine, then you can use IFORT to compile this program.
!     As this program itself does not perform any heavy calculation, any of these two compilers will work fine
!     as long as they support EXECUTE_COMMAND_LINE subroutine.
!   - In parallel mode, the FORWARD_MPI called from this program currently runs only on shared memory system
!     (i.e., on a single node of cluster), thus number of parallel processes cannot be higher than maximum
!     number of cores per node when executing those programs from here.
!   - The programs called by this program are compiled with either Intel Fortran IFORT (ifort 12.1 or higher)
!     or GNU Fortran GCC (gfortran 5.1 or similar); and with Open MPI 3.0 if they are run with parallel mode.
!     (When compiled with IFORT compiler, FORWARD/FORWARD_MPI can run faster).
!
! --- Modification/change log ---
! - Written in original form by Delgersaikhan Tuya (Sep, 2018)
!
! ---------------------------------------------------------------------------------------------------------
! << MIK - Multi-region Integral Kinetic code, Version 1.0 >> by Delgersaikhan Tuya, Hiroki Takezawa, and
!    Toru Obara.
!    Laboratory for Advanced Nuclear Energy, Institute of Innovative Research, Tokyo Institute of Technology.
!
!    Date of release of code:  December 1, 2018
!
! Written in original form by: Delgersaikhan TUYA (Sep, 2018)
! ---------------------------------------------------------------------------------------------------------
!=====================================================================================================================
!
 IMPLICIT NONE
!---------------------------------- BEGINNING OF USER-DEFINED PARAMETERS ----------------------------------
!
 INTEGER, PARAMETER :: Nstep=6000000 ! Number of time steps
!
#ifdef MIK_MPI
! -- For parallel calculation
 INTEGER, PARAMETER :: Flen = 26    ! Command line length for FORWARD program run
 INTEGER, PARAMETER :: Mlen = 167   ! Command line length for MVP program run
 CHARACTER(len=Flen), PARAMETER ::  Fwd_cmd = "mpirun -np 2 ./forward_mpi"  ! Command line for FORWARD run
 CHARACTER(len=Mlen), PARAMETER ::  Mvp_cmd = "mpirun -np 10 $MVP_DIR/bin/LINUXGLIBC/mvp.mpi /5rf:mvpinp.inp "// &
 "/24rf:$MVPLIB_DIR/neutron.art.index /25rf:$MVPLIB_DIR/neutron.index /15f /16 /55f /75f /66f > mvpout.ft06"  ! Command line
                                                                    ! for MVP run
#else
! -- For single core calculation
 INTEGER, PARAMETER :: Flen = 10    ! Command line length for FORWARD program run
 INTEGER, PARAMETER :: Mlen = 150   ! Command line length for MVP program run
 CHARACTER(len=Flen), PARAMETER ::  Fwd_cmd = "./forward"  ! Command line for FORWARD run
 CHARACTER(len=Mlen), PARAMETER ::  Mvp_cmd = "$MVP_DIR/bin/LINUXGLIBC/mvp /5rf:mvpinp.inp "// &
 "/24rf:$MVPLIB_DIR/neutron.art.index /25rf:$MVPLIB_DIR/neutron.index /15f /16 /55f /75f /66f > mvpout.ft06"  ! Command line
                                                                    ! for MVP run
#endif
!
!------------------------------------ END OF USER-DEFINED PARAMETERS ------------------------------------
!
 INTEGER :: iost, callst
```

```
 INTEGER :: step
 LOGICAL :: lstat
!
! --- Start a main loop, which runs FORWARD/or FORWARD_MPI, UPDATE, and MVP2.0/or MVP3.0, successively ---
! -- First create a 'log.txt' file, which contains major events/logs of the calculation --
 INQUIRE(FILE="log.txt", EXIST=lstat)
 IF( .NOT. lstat )THEN
  OPEN(99, FILE='log.txt',FORM='FORMATTED',STATUS='NEW',ACTION='WRITE',POSITION='REWIND')
  WRITE(99,'(10X,A30)') " --- CALCULATION LOG --- "
  WRITE(99,*)
  CLOSE(99)
 END IF
!
 mainloop: DO
!
!  -- Open update_steps.dat and read the latest update step --
  OPEN(1,FILE='update_steps.dat',FORM='FORMATTED',STATUS='OLD',ACTION='READ',POSITION='APPEND',IOSTAT=iost)
  IF ( iost > 0 )THEN
   step = 0
  ELSEIF( iost == 0 )THEN
   BACKSPACE(1)
   READ(1,'(I10)') step
   step = step + 1
  END IF
  CLOSE(1)
!
  IF ( step < Nstep )THEN
!
!  -- Execute FORWARD program --
   OPEN(99, FILE='log.txt',FORM='FORMATTED',STATUS='OLD',ACTION='WRITE',POSITION='APPEND')
   WRITE(99,'(1X,A33,I10,A1)') "FORWARD program started at step ", step,"!"
   CLOSE(99)
   WRITE(*,'(1X,A33,I10,A1)') "FORWARD program started at step ", step,"!"
   CALL EXECUTE_COMMAND_LINE (Fwd_cmd, EXITSTAT=callst)
   OPEN(99, FILE='log.txt',FORM='FORMATTED',STATUS='OLD',ACTION='WRITE',POSITION='APPEND')
   WRITE(99,'(1X,A23,I10,A18)') "FORWARD program started at step ", step," is now finished!"
   WRITE(*,'(1X,A23,I10,A18)') "FORWARD program started at step ", step," is now finished!"
   CLOSE(99)
!
!  -- Finish calculation if [Nstep] is reached --
  ELSEIF( step == Nstep )THEN
   OPEN(99, FILE='log.txt',FORM='FORMATTED',STATUS='OLD',ACTION='WRITE',POSITION='APPEND')
   WRITE(99,*) "-----------------------------------------"
   WRITE(99,*) "The whole calculation finished successfully!"
   WRITE(*,*) "The whole calculation finished successfully!"
   CLOSE(99)
   EXIT mainloop
  END IF
!
! -- Stop if error was produced --
  INQUIRE(FILE="error.txt", EXIST=lstat)
  IF( lstat )THEN
   STOP
  END IF
!
!  -- Open update_steps.dat again and read the latest update step --
  OPEN(1,FILE='update_steps.dat',FORM='FORMATTED',STATUS='OLD',ACTION='READ',POSITION='APPEND',IOSTAT=iost)
  BACKSPACE(1)
  READ(1,'(I10)') step
  CLOSE(1)
!
  IF (step < Nstep)THEN
!
! -- Execute UPDATE program --
   OPEN(99, FILE='log.txt',FORM='FORMATTED',STATUS='OLD',ACTION='WRITE',POSITION='APPEND')
   WRITE(99,'(1X,A32,I10,A1)') "UPDATE program started at step ", step,"!"
```

```
  WRITE(*,'(1X,A32,I10,A1)') "UPDATE program started at step ", step,"!"
  CLOSE(99)
  CALL EXECUTE_COMMAND_LINE ("./update", EXITSTAT=callst)
  OPEN(99, FILE='log.txt',FORM='FORMATTED',STATUS='OLD',ACTION='WRITE',POSITION='APPEND')
  WRITE(99,'(1X,A32,I10,A17)') "UPDATE program started at step ", step," is finished now!"
  WRITE(*,'(1X,A32,I10,A17)') "UPDATE program started at step ", step," is finished now!"
  CLOSE(99)
!
! -- Stop if error was produced --
  INQUIRE(FILE="error.txt", EXIST=lstat)
  IF( lstat )THEN
   STOP
  END IF
!
! -- Execute MVP2.0/ or MVP3.0 program --
  OPEN(99, FILE='log.txt',FORM='FORMATTED',STATUS='OLD',ACTION='WRITE',POSITION='APPEND')
  WRITE(99,'(1X,A32,I10,A1)') "MVP program started at step ", step,"!"
  WRITE(*,'(1X,A32,I10,A1)') "MVP program started at step ", step,"!"
  CALL EXECUTE_COMMAND_LINE (Mvp_cmd, EXITSTAT=callst)
  CLOSE(99)
!
! -- Finish calculation if [Nstep] is reached --
 ELSEIF( step == Nstep )THEN
  OPEN(99, FILE='log.txt',FORM='FORMATTED',STATUS='OLD',ACTION='WRITE',POSITION='APPEND')
  WRITE(99,*) "-------------------------------------------"
  WRITE(99,*) "The whole calculation finished successfully!"
  WRITE(*,*) "The whole calculation finished successfully!"
  CLOSE(99)
  EXIT mainloop
 END IF
END DO mainloop
!
 END PROGRAM
```

2. *update.f90* program

```
PROGRAM UPDATE
!
!=============================================================================================================================
!
! Program: UPDATE ( Program to update a MVP2.0/or MVP3.0 input file at each update time step )
! Executed by: MAIN
!
! This program reads original MVP input file, updates it with update variable values stored in "update_vars.nml"
! namelist file, and creates "mvpinp.inp" file, which is then used by MVP2.0/or MVP3.0 code.
!
! --- Modification/change log ---
! - Written in original form by Delgersaikhan TUYA (Aug, 2018)
!
! ----------------------------------------------------------------------------------------------------
! << MIK - Multi-region Integral Kinetic code, Version 1.0 >> by Delgersaikhan Tuya, Hiroki Takezawa, and
!    Toru Obara.
!    Laboratory for Advanced Nuclear Energy, Institute of Innovative Research, Tokyo Institute of Technology.
!
!    Date of release of code:  December 1, 2018
!
! Written in original form by: Delgersaikhan TUYA (Sep, 2018)
! ----------------------------------------------------------------------------------------------------
!=============================================================================================================================
!
 USE MOD_VALUES, ONLY: Nreg,Nuvar,Nrvar,Uvarn,Rvarn,para,num,uvar,rvar,uvarnml,rvarnml,Inpname,Dlen,Llen,Sign,Vlen,Val_fmt
!
 CHARACTER(len=Llen), DIMENSION(:), ALLOCATABLE :: line ! Array to store lines in [Inpname] file
```

```
 CHARACTER(len=Llen), DIMENSION(:), ALLOCATABLE :: line_ns  ! line with no space
 CHARACTER(len=Llen) :: line_var ! Variable to hold one line of [Inpname] file
 CHARACTER(len=Llen) :: line_var_ns ! line_var with no space
 CHARACTER(len=Vlen) :: value
 CHARACTER(len=4) :: reg_fmt, dmynum
 INTEGER :: nline, i, j, k, m, iost, l, count
!
! --- Open 'update_vars.nml' and read values to be updated ---
!
 OPEN(1,FILE='update_vars.nml',STATUS='OLD', DELIM='APOSTROPHE')
 READ(1,nml=uvarnml)
 READ(1,nml=rvarnml)
 CLOSE(1)
!
! --- Open Inpname file and copy each line ---
!  -- First count a number of lines in the input file --
 OPEN(2,FILE=Inpname,FORM='FORMATTED',STATUS='OLD',ACTION='READ',POSITION='REWIND')
 nline=0
 DO
  READ(2,'(A73)',IOSTAT=iost)
  IF( iost == -1)THEN
   EXIT
  END IF
  nline = nline + 1
 END DO
 REWIND(2)
! WRITE(*,*) 'Line count:', nline
!
!  -- Allocate 'line' and [line_ns] arrays --
 ALLOCATE(line(nline))
 ALLOCATE(line_ns(nline))
!
 DO k=1, nline
  READ(2,'(A73)') line(k)
 END DO
 CLOSE(2)
!
! --- Eliminate white spaces ---
 lineloop: DO k=1, nline
  line_var_ns=" "
  recordloop: DO j=1, Llen
   line_var = line(k)
   IF(line_var(j:j) == " " )THEN
    CYCLE recordloop
   ELSE
    line_var_ns = TRIM(line_var_ns)//line_var(j:j)
   END IF
  END DO recordloop
  line_ns(k) = line_var_ns
 END DO lineloop
!
! --- Update parameters ---
!  -- First, update variables --
 WRITE(dmynum,'(I1)') Dlen
 reg_fmt = "(I"//TRIM(dmynum)//")"
!
 OPEN(99,FILE='log.txt',FORM='FORMATTED',STATUS='OLD',ACTION='WRITE',POSITION='APPEND')
 count = 0
 DO k=1, nline
  line_var_ns = TRIM(ADJUSTL(line_ns(k)))
  DO j=1, Nuvar
   DO i=1, Nreg
    WRITE(num,reg_fmt) i
    para = Uvarn(j)//TRIM(ADJUSTL(num))
    L = LEN_TRIM(para)
    IF( TRIM(Sign)//TRIM(para)//'=' == line_var_ns(1:L+2) )THEN
```

```
    WRITE(value,Val_fmt) uvar(i,j)
    line(k) = TRIM(Sign//TRIM(para)//' ='//value)
    WRITE(99,*) "UPDATE program: Update variable ",para," is updated!"
    WRITE(*,*) "UPDATE program: Update variable ",para," is updated!"
    count = count + 1
   END IF
  END DO
 END DO
END DO
!
! -- Check if all update variables are updated --
 IF( count /= Nuvar*Nreg )THEN
  WRITE(99,*) "UPDATE program: Not all update variables are updated. Error, stop!"
  WRITE(*,*) "UPDATE program: Not all update variables are updated. Error, stop!"
  CLOSE(99)
  OPEN(100,FILE="error.txt",FORM='FORMATTED',STATUS='REPLACE')
  CLOSE(100)
  STOP
 END IF
!
!  -- Next, reactivity variables --
 count = 0
 DO k=1, nline
  line_var_ns = TRIM(ADJUSTL(line_ns(k)))
  DO j=1, Nrvar
   para = Rvarn(j)
   L = LEN_TRIM(para)
   IF( TRIM(Sign)//TRIM(para)//'=' == line_var_ns(1:L+2) )THEN
    WRITE(value,Val_fmt) rvar(j)
    WRITE(99,*) "UPDATE program: Reactivity variable ", para," is updated!"
    WRITE(*,*) "UPDATE program: Reactivity variable ", para," is updated!"
    line(k) = TRIM(Sign//TRIM(para)//' ='//value)
    count = count + 1
   END IF
  END DO
 END DO
!
! -- Check if all reactivity variables are updated --
 IF( count /= Nrvar )THEN
  WRITE(99,*) "UPDATE program: Not all reactivity variables are updated. Error, stop!"
  WRITE(*,*) "UPDATE program: Not all reactivity variables are updated. Error, stop!"
  CLOSE(99)
  OPEN(100,FILE="error.txt",FORM='FORMATTED',STATUS='REPLACE')
  CLOSE(100)
  STOP
 END IF
!
 CLOSE(99)
!
! --- Save the updated input as 'mvpinp.inp' file ---
 OPEN(2,FILE='mvpinp.inp',FORM='FORMATTED',STATUS='REPLACE',ACTION='WRITE')
!
 DO k=1, nline
  line_var = line(k)
  IF(line_var(1:1) == "  ")THEN
   WRITE(2,'(2X,A)') TRIM(ADJUSTL(line(k)))
  ELSEIF(line_var(1:1) == " ")THEN
   WRITE(2,'(X,A)') TRIM(ADJUSTL(line(k)))
  ELSE
   WRITE(2,'(A)') TRIM(ADJUSTL(line(k)))
  END IF
 END DO
 CLOSE(2)
!
 DEALLOCATE(line)
 DEALLOCATE(line_ns)
```

```
!
 END PROGRAM
```

3. *forward.90* program

```
PROGRAM FORWARD
!
!=============================================================================================================================
!
! Program: FORWARD ( for single mode calculation )
! Executed by: MAIN
! Calls: CFIJTALLY, IKM, FEEDBACK, & utility subroutines for reading and writing.
!       And SGETRF and SGETRS subroutines from LAPACK library (SGETRF and SGETRS also call
!       other functions and subroutines from LAPACK. In total 14 subroutines/functions are
!       used from LAPACK)
!
! --- Modification/change log ---
! - The IKM subroutine called by this program was initially developed by Hiroki TAKEZAWA (Since Aug, 2008))
! - New IKM-MVP coupling was developed by Delgersaikhan TUYA  (2017 - 2018)
!     -- It is compiled with either Intel Fortran IFORT (ifort 12.1 or similar/higher)
!       or GNU Fortran GCC (gfortran 5.1 or similar). (When compiled with IFORT compiler, it can run faster).
!
! ----------------------------------------------------------------------------------------------------------
! << MIK - Multi-region Integral Kinetic code, Version 1.0 >> by Delgersaikhan Tuya, Hiroki Takezawa, and
!    Toru Obara.
!    Laboratory for Advanced Nuclear Energy, Institute of Innovative Research, Tokyo Institute of Technology.
!
!    Date of release of code:  December 1, 2018
!
! Written in original form by: Delgersaikhan TUYA (Sep, 2018)
! ----------------------------------------------------------------------------------------------------------
!=============================================================================================================================
!
! Use 'mod_values.f90' module which defines essentially all array & parameter declarations used for FORWARD
 USE MOD_VALUES
 ! Use 'mod_comp_subs.f90' module which contains subroutines used for computations
 USE MOD_COMP_SUBS
 ! Use 'mod_util_subs.f90' module which contains utility subroutines used for reading or writing data
 USE MOD_UTIL_SUBS
 ! Use 'mod_feedback.f90' module which contains FEEDBACK subroutine
 USE MOD_FEEDBACK
!
! IMPLICIT NONE
!
! -- Read [step_beg] from "update_steps.dat" --
 INQUIRE(FILE='update_steps.dat', EXIST=lstat)
 IF( .NOT. lstat )THEN
! it means calculation is starting
  step_beg = 0
! -- Create 'update_steps.dat' file and write step_beg into it--
  OPEN(1,FILE='update_steps.dat',FORM='FORMATTED',STATUS='NEW',ACTION='WRITE')
  WRITE(1,'(A12)') 'Update steps'
  WRITE(1,'(I10)') step_beg
  CLOSE(1)
 ELSE
  OPEN(1,FILE='update_steps.dat',FORM='FORMATTED',STATUS='OLD',ACTION='READ',POSITION='APPEND')
  BACKSPACE(UNIT=1, IOSTAT=estat)
  READ(1,'(I10)') step_end
  step_beg = step_end + 1
  CLOSE(1)
!
! -- Start writing logs to output screen and "log.txt" --
  OPEN(99,FILE='log.txt',FORM='FORMATTED',STATUS='OLD',ACTION='WRITE',POSITION='APPEND')
  WRITE(99,100) "FORWARD: step_beg is ", step_beg
  WRITE(*,100) "FORWARD: step_beg is ", step_beg
  100 FORMAT (1X,A22,I10)
```

```
 CLOSE(99)
!
 END IF
!
! -- Allocate and initialize necessary arrays --
 count_a = 0
! -- Allocate --
 IF( step_beg < Kcutp )THEN
  ALLOCATE(fr(0:Nstep, Nreg), STAT=astat)
  IF( astat /= 0 )THEN
   count_a = count_a + 1
  END IF
  ALLOCATE(sigfr(0:Nstep, Nreg), STAT=astat)
  IF( astat /= 0 )THEN
   count_a = count_a + 1
  END IF
  ALLOCATE(uvara(0:Nstep, Nreg, Nuvar), STAT=astat)
  IF( astat /= 0 )THEN
   count_a = count_a + 1
  END IF
 ELSEIF( step_beg >= Kcutp )THEN
  ALLOCATE(fr(step_beg-Kcutp:Nstep, Nreg), STAT=astat)
  IF( astat /= 0 )THEN
   count_a = count_a + 1
  END IF
  ALLOCATE(sigfr(step_beg-Kcutp:Nstep, Nreg), STAT=astat)
  IF( astat /= 0 )THEN
   count_a = count_a + 1
  END IF
  ALLOCATE(uvara(step_beg-Uvar_step_read:Nstep, Nreg, Nuvar), STAT=astat)
  IF( astat /= 0 )THEN
   count_a = count_a + 1
  END IF
 END IF
! -- Stop if error occurred during allocation --
 IF( count_a /= 0 )THEN
  OPEN(99,FILE='log.txt',FORM='FORMATTED',STATUS='OLD',ACTION='WRITE',POSITION='APPEND')
  WRITE(99,*) "FORWARD: Allocation is unsuccessful. Error, stop!"
  WRITE(*,*) "FORWARD: Allocation is unsuccessful. Error, stop!"
  CLOSE(99)
  OPEN(100,FILE="error.txt",FORM='FORMATTED',STATUS='REPLACE')
  CLOSE(100)
  STOP
 END IF
! -- Initialize necessary arrays --
 IF( step_beg <= 1 )THEN
  uvara(0,:,:) = Iuvar(:,:)
  nvar(:,:) = Invar(:,:)
 END IF
!
! -- Read [fr(k_beg:k_end)] data from "fr_dat.dat" --
!  -- First, read [fr] data
 IF( 1 < step_beg .AND. step_beg < Kcutp )THEN
  k_beg = 0
  k_end = step_beg - 1
  CALL FR_READ(k_beg, k_end, 0, 10, fr(k_beg:k_end,:), sigfr(k_beg:k_end,:), estat)
!
 ELSEIF( step_beg >= Kcutp )THEN
  skip_num = (step_beg - Kcutp)*Nreg
  k_beg = step_beg - Kcutp
  k_end = step_beg - 1
  CALL FR_READ(k_beg, k_end, skip_num, 10, fr(k_beg:k_end,:), sigfr(k_beg:k_end,:), estat)
 END IF
! -- Stop if there is an error and create "error.txt" file --
 IF( estat /= 0 )THEN
  OPEN(100,FILE="error.txt",FORM='FORMATTED',STATUS='REPLACE')
```

```
 CLOSE(100)
 STOP
 END IF
!
! -- Read [uvara(k_beg:k_end,:,:)] from a given file --
 IF( step_beg > 1 )THEN
  skip_num = (step_beg - Uvar_step_read)*Nreg
  k_beg = step_beg - Uvar_step_read
  k_end = step_beg - 1
  DO v=1, Nuvar
   u = v + 20  ! Input/output unit number
   CALL UVAR_READ(k_beg, k_end, skip_num, Uvarn(v), u, uvara(k_beg:k_end,:,v), estat)
! -- Stop if there is an error and create "error.txt" file --
   IF( estat /= 0 )THEN
    OPEN(100,FILE="error.txt",FORM='FORMATTED',STATUS='REPLACE')
    CLOSE(100)
    STOP
   END IF
  END DO
 END IF
!
! -- Read [uvara(last update step,:,:)] from 'update_vars.nml' file as [uvar_last] variable. This was arised particularly
!    due to restart calculation because in case of restart calculation the [uvara(step_beg-1,:,:)], which is
!    read above, is not necessarily equal to [uvara(last update step,:,:)]. The [uvara(last update step,:,:)]
!    is needed for checking whether or not to update kinetic functions! --
!
 IF( step_beg > 1 )THEN
  OPEN(40,FILE='update_vars.nml',STATUS='OLD', DELIM='APOSTROPHE')
  READ(40,nml=uvarnml)
  uvar_last(:,:) = uvar(:,:)
  CLOSE(40)
 ELSEIF( step_beg <= 1)THEN
  uvar_last = uvara(0,:,:)
 END IF
!
! -- Read [nvar] at the last update time step --
 IF( step_beg > 1 )THEN
  OPEN(40,FILE='nonupdate_vars.nml',STATUS='OLD', DELIM='APOSTROPHE')
  READ(40,nml=nvarnml)
  CLOSE(40)
 END IF
!
! -- Call CFIJTALLY subroutine --
 IF( step_beg > 0 )THEN
  CALL CFIJTALLY(1, Nreg, keff, cfijp, sigcfijp, estat)
! -- Stop if there is an error in CFIJTALLY call and create "error.txt" file --
  IF( estat /= 0 )THEN
   OPEN(100,FILE="error.txt",FORM='FORMATTED',STATUS='REPLACE')
   CLOSE(100)
   STOP
  END IF
 END IF
!
! -- Calculate [cijp] function using cumulative fission [cfijp] --
!   -- First, calculate total fission source in each region --
!
 IF( step_beg > 0 )THEN
  tcfp = 0.0D0
  sigtcfp = 0.0D0
  DO i=1, Nreg
   DO j=1, Nreg
    tcfp(i) = tcfp(i) + cfijp(Kcutp,i,j)
    sigtcfp(i) = SQRT((sigtcfp(i))**2 + (sigcfijp(Kcutp,i,j))**2)
   END DO
  END DO
!
```

```
!   -- Calculating [cijp] functions --
!
  DO i=1, Nreg
   DO j=1, Nreg
    DO k=0, Kcutp
     cijp(k,i,j) = ( keff * cfijp(k,i,j) / tcfp(j) )*K_adjust/K_init
!
     sigcijp(k,i,j) = ( keff * SQRT((sigcfijp(k,i,j)/tcfp(j))**2 + (cfijp(k,i,j)*sigtcfp(j)/ &
                      (tcfp(J))**2)**2) )*K_adjust/K_init
!
     sigcijp(k,i,j) = keff * sigcijp(k,i,j)
    END DO
   END DO
  END DO
 END IF
!
! -- Save [keff] at update time step --
 IF( step_beg == 1)THEN
  OPEN(5,FILE='keff.dat',FORM='FORMATTED',STATUS='NEW',ACTION='WRITE',POSITION='REWIND')
  WRITE(5,'(2X,A10,3X,A5)') 'STEP','K-EFF'
  WRITE(5,'(I10,ES12.5)') step_beg-1 ,keff*K_adjust/K_init
  CLOSE(5)
 ELSEIF( step_beg > 1 )THEN
  OPEN(5,FILE='keff.dat',FORM='FORMATTED',STATUS='OLD',ACTION='WRITE',POSITION='APPEND')
  WRITE(5,'(I10,ES12.5)') step_beg-1 ,keff*K_adjust/K_init
  CLOSE(5)
 END IF
!
! -- First save [cijp] at initial condition --
 IF( step_beg == 1 )THEN
  CALL CIJP_WRITE(keff, step_beg-1, cijp, sigcijp, 30)
!
! -- Calculate initial region-wise fission rate --
  IF( Nreg == 1 )THEN
   fr(0,1) = Ipwr/Fise/Mevtoj
  ELSEIF( Nreg == 2 )THEN
   frac(1) = 1/( 1+( 1-cijp(Kcutp,1,1)/(keff*K_adjust/K_init))/cijp(Kcutp,1,2)/(keff*K_adjust/K_init) )
   fr(0,1) = frac(1)*Ipwr/Fise/Mevtoj
   frac(2) = 1.0-frac(1)
   fr(0,2) = frac(2)*Ipwr/Fise/Mevtoj
!    WRITE(*,*) "fr(0,1) is ", fr(0,1)
!    WRITE(*,*) "fr(0,2) is ", fr(0,2)
  ELSE
!
   DO i=1, Nreg
    DO j=1, Nreg
     ccij0(i,j) = cijp(Kcutp,i,j)/(keff*K_adjust/K_init)
     IF ( i == j) THEN
      ccij0(i,j) = ccij0(i,j) - 1.D0
     END IF
    END DO
   END DO
!
!
   DO i=1, Nreg-1
    rfrac(i) = - ccij0(i,Nreg)
    DO j=1, Nreg-1
     cijmat(i,j) = ccij0(i,j) - ccij0(i,Nreg)
    END DO
   END DO
!
   CALL SGETRF(Nreg-1,Nreg-1,cijmat,Nreg-1,piv,info1)
!
   CALL SGETRS('N',Nreg-1,1,cijmat,Nreg-1,piv,rfrac,Nreg-1,info2)
!
   DO I=1, Nreg-1
```

```
    frac(i) = rfrac(i)
    fr(0,i) = frac(i)*Ipwr/Fise/Mevtoj
   END DO
   frac(Nreg) = 1.D0 - SUM(rfrac)
   fr(0,Nreg) = frac(Nreg)*Ipwr/Fise/Mevtoj
!
  END IF
  sigfr(0,:) = 0.0D0
!
 END IF
!
! -- Read [cijp0] if step_beg < Kcutp --
 IF( 0 < step_beg .AND. step_beg < Kcutp )THEN
  CALL CIJP_READ(30, 0, cijp0, sigcijp0, estat)
! -- Stop if there was error in calling CIJP_READ --
  IF( estat /= 0 )THEN
   OPEN(100,FILE="error.txt",FORM='FORMATTED',STATUS='REPLACE')
   CLOSE(100)
   STOP
  END IF
 END IF
!
! --- Start of FORWARD calculation ---
!
 FORWARDloop: DO step=step_beg, Nstep
!
  stopstep = step
!
! -- First, read 'reactivity.dat' and insert reactivity at step=0 --
  IF( step == 0 )THEN
   CALL REAC_INSERT(step, 2, Lrfile, Rfile, rvar, estat, rins)
! -- Stop if there was an error in calling REAC_INSERT --
   IF( estat /= 0 )THEN
    OPEN(100,FILE="error.txt",FORM='FORMATTED',STATUS='REPLACE')
    CLOSE(100)
    STOP
   END IF
!
   IF ( rins == "Y" )THEN
    uvar(:,:) = uvara(0,:,:)
    OPEN(40, FILE='update_vars.nml', STATUS='NEW', DELIM='APOSTROPHE')
    WRITE(40, NML = uvarnml)
    WRITE(40, NML = rvarnml)
    CLOSE(40)
!
    OPEN(40, FILE='nonupdate_vars.nml', STATUS='NEW', DELIM='APOSTROPHE')
    WRITE(40, NML = nvarnml)
    CLOSE(40)
    EXIT FORWARDloop
   END IF
  END IF
!
! -- Calculate [fr] & [sigfr] at the current time step --
  IF( step < Kcutp )THEN
   k_beg = 0
   k_end = step - 1
  ELSEIF( step >= Kcutp )THEN
   k_beg = step - Kcutp
   k_end = step - 1
! -- Set their [cijp0] and [sigcijp0] to 0.D0 because they are not used in the calculation
!    when step >= Kcutp, but are argument for calling IKM subroutine below--
   cijp0(:,:,:) = 0.D0
   sigcijp0(:,:,:) = 0.D0
  END IF
!
! -- Call IKM subroutine to calculate [curfr]
```

```
  CALL IKM(1, Nreg, k_beg, k_end, cijp, sigcijp, cijp0, sigcijp0, fr(k_beg:k_end,:), &
        sigfr(k_beg:k_end,:), curfr(:,1), curfr(:,2), estat)
! -- Stop if error occurred in calling IKM subroutine --
  IF( estat /= 0 )THEN
   OPEN(100,FILE="error.txt",FORM='FORMATTED',STATUS='REPLACE')
   CLOSE(100)
   STOP
  END IF
!
! -- Call feedback.f90 to calculate [uvara(step,:,:)] and [nvar]--
  fr(step,:) = curfr(:,1)
  sigfr(step,:) = curfr(:,2)
  uvarprev(:,:) = uvara(step-1,:,:)
  nvarprev(:,:) = nvar(:,:)
  frprev(:) = fr(step-1,:)
  CALL FEEDBACK(frprev, uvarprev, nvarprev, uvarcur, nvar, estat)
! -- Stop if error occurred in calling FEEDBACK --
  IF( estat /= 0 )THEN
   OPEN(100,FILE="error.txt",FORM='FORMATTED',STATUS='REPLACE')
   CLOSE(100)
   STOP
  END IF
  uvara(step,:,:) = uvarcur(:,:)
  duvar(:,:) = uvara(step,:,:) - uvar_last(:,:)
!
! -- Output [cijp] function at user-specified steps in [Out_list] if there is any --
  IF( Nout > 0 )THEN
   outloop: DO m=1, Nout
    IF( Out_list(m) == step )THEN
     CALL CIJP_WRITE(0.D0, step, cijp, sigcijp, 30)
     EXIT outloop
    END IF
   END DO outloop
  END IF
!
! -- Check the followings: --
!   -  Check-1: check if C_ij functions are to be updated or not based on update criteria --
!   -  Check-2: check if reactivity is to be inserted or not at the next step, i.e., at step+1 --
!
! -- Check-1
  updcheck = 0
  checkloop: DO v=1, Nuvar
   DO i=1, Nreg
    IF( duvar(i,v) >= Uvarcrit(i,v) )THEN
     updcheck = 1
     EXIT checkloop
    END IF
   END DO
  END DO checkloop
!
  IF( updcheck == 1  )THEN
! -- First, write update & reactivity variables
   uvar(:,:) = uvara(step,:,:)
   OPEN(40, FILE='update_vars.nml', STATUS='REPLACE', DELIM='APOSTROPHE')
   WRITE(40, NML = uvarnml)
   DO u=1, Nuvar
    DO i=1, Nreg
     DO v=1, Nrvar
      WRITE(num,'(I3)') i
      para = Uvarn(u)//TRIM(ADJUSTL(num))
      IF( TRIM(para) == rvarn(v) )THEN
       rvar(v) = uvar(i,u)
      END IF
     END DO
    END DO
   END DO
```

```
  WRITE(40, NML = rvarnml)
  CLOSE(40)
!
! -- Then, write non-update variables
  OPEN(40, FILE='nonupdate_vars.nml', STATUS='REPLACE', DELIM='APOSTROPHE')
  WRITE(40, NML = nvarnml)
  CLOSE(40)
  EXIT FORWARDloop
 END IF
!
! -- Check-2:
 IF( step+1 <= Riend )THEN
! -- Open [Rfile] file and read reactivity insertion data --
  CALL REAC_INSERT(step+1, 2, Lrfile, Rfile, rvar, estat, rins)
! -- Stop if there was an error in calling REAC_INSERT --
  IF( estat /= 0 )THEN
   OPEN(100,FILE="error.txt",FORM='FORMATTED',STATUS='REPLACE')
   CLOSE(100)
   STOP
  END IF
! -- Insert reactivity if necessary --
  IF( rins == "Y" )THEN
! -- First, write update & reactivity variables
   uvar(:,:) = uvara(step,:,:)
   OPEN(40, FILE='update_vars.nml', STATUS='REPLACE', DELIM='APOSTROPHE')
   WRITE(40, NML = uvarnml)
   WRITE(40, NML = rvarnml)
   CLOSE(40)
!
! -- Then, write non-update variables
   OPEN(40, FILE='nonupdate_vars.nml', STATUS='REPLACE', DELIM='APOSTROPHE')
   WRITE(40, NML = nvarnml)
   CLOSE(40)
   EXIT FORWARDloop
  END IF
 END IF
!
! -- End of FORWARD calculation --
 END DO FORWARDloop
!
! -- Write [fr(step_beg:stopstep,:)] and [uvara(step_beg:stopstep,:,:)]
!    to "fr_dat.dat" and "xxxx_dat.dat" respectively --

 IF( step_beg == 1 )THEN
  CALL FR_WRITE(0, stopstep, fr(0:stopstep,:), sigfr(0:stopstep,:), 10, "NEW", "REWIND", estat)
! -- Stop if there was error writing --
  IF( estat /= 0 )THEN
   OPEN(100,FILE="error.txt",FORM='FORMATTED',STATUS='REPLACE')
   CLOSE(100)
   STOP
  ENDIF
!
 ELSEIF( step_beg > 1 )THEN
  CALL FR_WRITE(step_beg, stopstep, fr(step_beg:stopstep,:), sigfr(step_beg:stopstep,:), 10, "OLD", "APPEND", estat)
! -- Stop if there was error writing --
  IF( estat /= 0 )THEN
   OPEN(100,FILE="error.txt",FORM='FORMATTED',STATUS='REPLACE')
   CLOSE(100)
   STOP
  ENDIF
 END IF
!
 IF( step_beg == 1 )THEN
  DO v=1, Nuvar
   u = v + 20
   CALL UVAR_WRITE(0, stopstep, uvara(0:stopstep,:,v), Uvarn(v), u, "NEW", "REWIND", estat)
```

```
! -- Stop if there was error writing --
  IF( estat /= 0 )THEN
   OPEN(100,FILE="error.txt",FORM='FORMATTED',STATUS='REPLACE')
   CLOSE(100)
   STOP
  ENDIF
 END DO
 ELSEIF( step_beg > 1 )THEN
  DO v=1, Nuvar
  u = v + 20
  CALL UVAR_WRITE(step_beg, stopstep, uvara(step_beg:stopstep,:,v), Uvarn(v), u, "OLD", "APPEND", estat)
! -- Stop if there was error writing --
  IF( estat /= 0 )THEN
   OPEN(100,FILE="error.txt",FORM='FORMATTED',STATUS='REPLACE')
   CLOSE(100)
   STOP
  ENDIF
 END DO
 END IF
!
! -- Write [nvarnml] --
 OPEN(40, FILE='nonupdate_vars.nml', STATUS='REPLACE', DELIM='APOSTROPHE')
 WRITE(40, NML = nvarnml)
 CLOSE(40)
! -- Write [stopstep] to "update_steps.dat" --
 OPEN(1,FILE='update_steps.dat',FORM='FORMATTED',STATUS='OLD',ACTION='WRITE',POSITION='APPEND')
 WRITE(1,'(I10)') stopstep
 CLOSE(1)
!
 STOP
!
! 1010 FORMAT(I5, ES16.8, ES16.8, ES16.8)  ! For UNIT=10 or fr_dat.dat
! 1020 FORMAT(I5, ES16.8, ES16.8)    ! For UNIT=U or xxxx_dat.dat
! 1030 FORMAT(2I5, ES16.8, ES16.8, ES16.8) ! For UNIT=30 or cij_xxxxxxxxxx.dat
 1002 FORMAT(ES16.8, ES16.8) ! For UNIT=2 or reactivity.dat
!
 END PROGRAM
```

4. *forward_mpi.f90* program

```
PROGRAM FORWARD_MPI
!
!=============================================================================================================
!
! Program: FORWARD_MPI ( for parallel mode calculation using Open MPI )
! Executed by: MAIN
! Calls: CFIJTALLY, IKM, FEEDBACK, & utility subroutines for reading and writing.
!       And SGETRF and SGETRS subroutines from LAPACK library (SGETRF and SGETRS also call
!       other functions and subroutines from LAPACK. In total 14 subroutines/functions are
!       used from LAPACK)
!
! --- Modification/change log ---
! - The IKM subroutine called by this program was initially developed by Hiroki TAKEZAWA (Since Aug, 2008))
! - New IKM-MVP coupling and parallel mode calculation (MPI) were developed by Delgersaikhan TUYA  (2017 - 2018)
!     -- It is compiled with Open MPI 3.0, which supports shared-memory window feature, installed with either
!       Intel Fortran IFORT (ifort 12.1 or higher/similar) or GNU Fortran GCC (gfortran 5.1 or similar).
!       (When compiled with Open MPI installed with IFORT compiler, it can run faster).
!     -- It currently runs only on shared memory system (i.e., on a single node, which has shared memory
!       architecture, of cluster), thus number of parallel processes cannot be higher than maximum number
!       of cores per node.
!
! ------------------------------------------------------------------------------------------------------
! << MIK - Multi-region Integral Kinetic code, Version 1.0 >> by Delgersaikhan Tuya, Hiroki Takezawa, and
```

```
!    Toru Obara.
!    Laboratory for Advanced Nuclear Energy, Institute of Innovative Research, Tokyo Institute of Technology.
!
!    Date of release of code:  December 1, 2018
!
! Written in original form by: Delgersaikhan TUYA (Sep, 2018)
! ------------------------------------------------------------------------------------------------------------
!============================================================================================================
!
! Use 'mod_values.f90' module which defines essentially all array & parameter declarations used for FORWARD
 USE MOD_VALUES
 ! Use 'mod_comp_subs.f90' module which contains subroutines used for computations
 USE MOD_COMP_SUBS
 ! Use 'mod_util_subs.f90' module which contains utility subroutines used for reading or writing data
 USE MOD_UTIL_SUBS
 ! Use 'mod_feedback.f90' module which contains FEEDBACK subroutine
 USE MOD_FEEDBACK
!
! IMPLICIT NONE
!
! -- All processes initialize MPI --
 CALL MPI_Init(ierr)
 CALL MPI_COMM_RANK(MPI_COMM_WORLD, idtsk, ierr)
!
 CALL MPI_Comm_Split_Type(MPI_COMM_WORLD, MPI_COMM_TYPE_SHARED, 0, MPI_INFO_NULL, nodecomm,ierr)
 CALL MPI_Comm_Rank(nodecomm, noderank, ierr)
 CALL MPI_Comm_Size(nodecomm, n_procs, ierr)
!
! -- All processes read [step_beg] from "update_steps.dat" --
 INQUIRE(FILE='update_steps.dat', EXIST=lstat)
 IF( .NOT. lstat )THEN
! it means calculation is starting
  step_beg = 0
! -- Master process creates 'update_steps.dat' file and writes step_beg into it--
  IF( noderank == 0 )THEN
   OPEN(1,FILE='update_steps.dat',FORM='FORMATTED',STATUS='NEW',ACTION='WRITE')
   WRITE(1,'(A12)') 'Update steps'
   WRITE(1,'(I10)') step_beg
   CLOSE(1)
  END IF
 ELSE
  OPEN(1,FILE='update_steps.dat',FORM='FORMATTED',STATUS='OLD',ACTION='READ',POSITION='APPEND')
  BACKSPACE(UNIT=1, IOSTAT=estat)
  READ(1,'(I10)') step_end
  step_beg = step_end + 1
  CLOSE(1)
!
! -- All processes start writing logs to output screen and "log.txt" --
  OPEN(99,FILE='log.txt',FORM='FORMATTED',STATUS='OLD',ACTION='WRITE',POSITION='APPEND')
  WRITE(99,100) 'Process ',noderank,' reporting: step_beg is ', step_beg
  WRITE(*,100) 'Process ',noderank,' reporting: step_beg is ', step_beg
  100 FORMAT (1X,A8,I3,A24,I10)
  CLOSE(99)
!
 END IF
!
! -- Master process allocates and initializes necessary arrays --
 IF( noderank == 0 )THEN
  count_a = 0
! -- Allocate --
  IF( step_beg < Kcutp )THEN
   ALLOCATE(fr(0:Nstep, Nreg), STAT=astat)
   IF( astat /= 0 )THEN
    count_a = count_a + 1
   END IF
   ALLOCATE(sigfr(0:Nstep, Nreg), STAT=astat)
```

```
  IF( astat /= 0 )THEN
   count_a = count_a + 1
  END IF
  ALLOCATE(uvara(0:Nstep, Nreg, Nuvar), STAT=astat)
  IF( astat /= 0 )THEN
   count_a = count_a + 1
  END IF
 ELSEIF( step_beg >= Kcutp )THEN
  ALLOCATE(fr(step_beg-Kcutp:Nstep, Nreg), STAT=astat)
  IF( astat /= 0 )THEN
   count_a = count_a + 1
  END IF
  ALLOCATE(sigfr(step_beg-Kcutp:Nstep, Nreg), STAT=astat)
  IF( astat /= 0 )THEN
   count_a = count_a + 1
  END IF
  ALLOCATE(uvara(step_beg-Uvar_step_read:Nstep, Nreg, Nuvar), STAT=astat)
  IF( astat /= 0 )THEN
   count_a = count_a + 1
  END IF
 END IF
! -- Stop if error occurred during allocation --
 IF( count_a /= 0 )THEN
  OPEN(99,FILE='log.txt',FORM='FORMATTED',STATUS='OLD',ACTION='WRITE',POSITION='APPEND')
  WRITE(99,*) "FORWARD_MPI: Allocation is unsuccessful. Error, stop!"
  WRITE(*,*) "FORWARD_MPI: Allocation is unsuccessful. Error, stop!"
  CLOSE(99)
  OPEN(100,FILE="error.txt",FORM='FORMATTED',STATUS='REPLACE')
  CLOSE(100)
  STOP
 END IF
! -- Initialize necessary arrays --
 IF( step_beg <= 1 )THEN
  uvara(0,:,:) = Iuvar(:,:)
  nvar(:,:) = Invar(:,:)
 END IF
END IF
!
! -- Master process reads [fr(k_beg:k_end)] data from "fr_dat.dat" --
!  -- First, read [fr] data
IF( noderank == 0 )THEN
 IF( 1 < step_beg .AND. step_beg < Kcutp )THEN
  k_beg = 0
  k_end = step_beg - 1
  CALL FR_READ(k_beg, k_end, 0, 10, fr(k_beg:k_end,:), sigfr(k_beg:k_end,:), estat)
!
 ELSEIF( step_beg >= Kcutp )THEN
  skip_num = (step_beg - Kcutp)*Nreg
  k_beg = step_beg - Kcutp
  k_end = step_beg - 1
  CALL FR_READ(k_beg, k_end, skip_num, 10, fr(k_beg:k_end,:), sigfr(k_beg:k_end,:), estat)
 END IF
! -- Stop if there is an error and create "error.txt" file --
 IF( estat /= 0 )THEN
  OPEN(100,FILE="error.txt",FORM='FORMATTED',STATUS='REPLACE')
  CLOSE(100)
  STOP
 END IF
!
! -- Master process reads [uvara(k_beg:k_end,:,:)] from a given file --
 IF( step_beg > 1 )THEN
  skip_num = (step_beg - Uvar_step_read)*Nreg
  k_beg = step_beg - Uvar_step_read
  k_end = step_beg - 1
  DO v=1, Nuvar
   u = v + 20  ! Input/output unit number
```

```
   CALL UVAR_READ(k_beg, k_end, skip_num, Uvarn(v), u, uvara(k_beg:k_end,:,v), estat)
! -- Stop if there is an error and create "error.txt" file --
   IF( estat /= 0 )THEN
    OPEN(100,FILE="error.txt",FORM='FORMATTED',STATUS='REPLACE')
    CLOSE(100)
    STOP
   END IF
  END DO
 END IF
!
! -- Master process reads [uvara(last update step,:,:)] from 'update_vars.nml' file as [uvar_last] variable. This was arised
!   particularly due to restart calculation because in case of restart calculation the [uvara(step_beg-1,:,:)], which is
!   read above, is not necessarily equal to [uvara(last update step,:,:)]. The [uvara(last update step,:,:)] is needed for
!   checking whether or not to update kinetic functions! --
!
 IF( step_beg > 1 )THEN
  OPEN(40,FILE='update_vars.nml',STATUS='OLD', DELIM='APOSTROPHE')
  READ(40,nml=uvarnml)
  uvar_last(:,:) = uvar(:,:)
  CLOSE(40)
 ELSEIF( step_beg <= 1)THEN
  uvar_last = uvara(0,:,:)
 END IF
!
! -- Master process reads [nvar] at the last update time step --
 IF( step_beg > 1 )THEN
  OPEN(40,FILE='nonupdate_vars.nml',STATUS='OLD', DELIM='APOSTROPHE')
  READ(40,nml=nvarnml)
  CLOSE(40)
 END IF
!
! -- Set shared window sizes for master process --
 winsize1 = Nreg*Nreg*(Kcutp+1)*8_MPI_ADDRESS_KIND! Byte
 winsize2 = winsize1
 winsize3 = Nreg*Kcutp*8_MPI_ADDRESS_KIND ! Byte
 winsize4 = 2*Nreg*8_MPI_ADDRESS_KIND ! Byte ( Contains both [fr] and [sigfr] at current time step, hence the factor 2)
 winsize5 = Nreg*Nuvar*8_MPI_ADDRESS_KIND ! Byte
 winsize6 = winsize3 ! Byte
!
 ELSEIF( noderank > 0)THEN
! -- Set shared window sizes for other processes --
 winsize1 = 0_MPI_ADDRESS_KIND
 winsize2 = 0_MPI_ADDRESS_KIND
 winsize3 = 0_MPI_ADDRESS_KIND
 winsize4 = 0_MPI_ADDRESS_KIND
 winsize5 = 0_MPI_ADDRESS_KIND
 winsize6 = 0_MPI_ADDRESS_KIND
 END IF
!
! -- Synchronize MPI processes --
 CALL MPI_Barrier(nodecomm,ierr)
!
! -- All processes create shared window with given size for data exchange --
 disp_unit = 8 ! Byte
 CALL MPI_Win_Allocate_Shared(winsize1, disp_unit, MPI_INFO_NULL, nodecomm, baseptr1, win1, ierr)
 CALL MPI_Win_Allocate_Shared(winsize2, disp_unit, MPI_INFO_NULL, nodecomm, baseptr2, win2, ierr)
 CALL MPI_Win_Allocate_Shared(winsize3, disp_unit, MPI_INFO_NULL, nodecomm, baseptr3, win3, ierr)
 CALL MPI_Win_Allocate_Shared(winsize4, disp_unit, MPI_INFO_NULL, nodecomm, baseptr4, win4, ierr)
 CALL MPI_Win_Allocate_Shared(winsize5, disp_unit, MPI_INFO_NULL, nodecomm, baseptr5, win5, ierr)
 CALL MPI_Win_Allocate_Shared(winsize6, disp_unit, MPI_INFO_NULL, nodecomm, baseptr6, win6, ierr)
!
! -- All processes query and get information about shared windows created by the master process --
 CALL MPI_Win_Shared_Query(win1, 0, winsize1, disp_unit, baseptr1, ierr)
 CALL MPI_Win_Shared_Query(win2, 0, winsize2, disp_unit, baseptr2, ierr)
 CALL MPI_Win_Shared_Query(win3, 0, winsize3, disp_unit, baseptr3, ierr)
 CALL MPI_Win_Shared_Query(win4, 0, winsize4, disp_unit, baseptr4, ierr)
```

```
 CALL MPI_Win_Shared_Query(win5, 0, winsize5, disp_unit, baseptr5, ierr)
 CALL MPI_Win_Shared_Query(win6, 0, winsize6, disp_unit, baseptr6, ierr)
!
! -- Convert C pointer to Fortran pointer --
 CALL C_F_POINTER(baseptr1, cfijpPTR, (/Kcutp+1, Nreg, Nreg/))
 CALL C_F_POINTER(baseptr2, sigcfijpPTR, (/Kcutp+1, Nreg, Nreg/))
 CALL C_F_POINTER(baseptr3, frPTR, (/Kcutp, Nreg/))
 CALL C_F_POINTER(baseptr4, curPTR, (/Nreg,2/))
 CALL C_F_POINTER(baseptr5, duvarPTR, (/Nreg, Nuvar/))
 CALL C_F_POINTER(baseptr6, sigfrPTR, (/Kcutp, Nreg/))
!
! -- All processes call CFIJTALLY subroutine and put their part to the shared window --
 IF( step_beg > 0 )THEN
!
! -- First, calling REG_DIV to get regions corresponding to particular process/or task
  CALL REG_DIV(Nreg, n_procs, noderank, reg_beg, reg_end, n_regtask, estat)
! -- Stop if there is an error in REG_DIV call and create "error.txt" file --
  IF( estat /= 0 )THEN
   OPEN(100,FILE="error.txt",FORM='FORMATTED',STATUS='REPLACE')
   CLOSE(100)
   STOP
  END IF
!
! -- Then, calling CFIJTALLY to get [cfijpPTR] for corresponding regions
  CALL CFIJTALLY(reg_beg, reg_end, keff, cfijpPTR(:,reg_beg:reg_end,:), sigcfijpPTR(:,reg_beg:reg_end,:), estat)
! -- Stop if there is an error in CFIJTALLY call and create "error.txt" file --
  IF( estat /= 0 )THEN
   OPEN(100,FILE="error.txt",FORM='FORMATTED',STATUS='REPLACE')
   CLOSE(100)
   STOP
  END IF
 END IF
!
! -- Synchronize MPI processes --
 CALL MPI_Barrier(nodecomm,ierr)
!
! -- All processes calculate [cijp] function using cumulative fission [cfijpPTR] --
!   -- First, calculate total fission source in each region --
!
 IF( step_beg > 0 )THEN
  tcfp = 0.0D0
  sigtcfp = 0.0D0
  DO i=1, Nreg
   DO j=1, Nreg
    tcfp(i) = tcfp(i) + cfijpPTR(Kcutp+1,i,j)
    sigtcfp(i) = SQRT((sigtcfp(i))**2 + (sigcfijpPTR(Kcutp+1,i,j))**2)
   END DO
  END DO
!
! -- Synchronize MPI processes --
  call MPI_Barrier(nodecomm,ierr)
!
!   -- Calculating [cijp] functions --
!
  DO i=1, Nreg
   DO j=1, Nreg
    DO k=0, Kcutp
     cijp(k,i,j) = ( keff * cfijpPTR(k+1,i,j) / tcfp(j) )*K_adjust/K_init
!
     sigcijp(k,i,j) = ( keff * SQRT((sigcfijpPTR(k+1,i,j)/tcfp(j))**2 + (cfijpPTR(k+1,i,j)*sigtcfp(j)/ &
                      (tcfp(J))**2)**2) )*K_adjust/K_init
!
     sigcijp(k,i,j) = keff * sigcijp(k,i,j)
    END DO
   END DO
  END DO
```

```
 END IF
!
! -- Synchronize MPI processes --
 CALL MPI_Barrier(nodecomm,ierr)
!
! -- Master process saves [keff] at update time step --
 IF( noderank == 0 )THEN
  IF( step_beg == 1)THEN
   OPEN(5,FILE='keff.dat',FORM='FORMATTED',STATUS='NEW',ACTION='WRITE',POSITION='REWIND')
   WRITE(5,'(2X,A10,3X,A5)') 'STEP','K-EFF'
   WRITE(5,'(I10,ES12.5)') step_beg-1 ,keff*K_adjust/K_init
   CLOSE(5)
  ELSEIF( step_beg > 1 )THEN
   OPEN(5,FILE='keff.dat',FORM='FORMATTED',STATUS='OLD',ACTION='WRITE',POSITION='APPEND')
   WRITE(5,'(I10,ES12.5)') step_beg-1 ,keff*K_adjust/K_init
   CLOSE(5)
  END IF
 END IF
!
! -- Master process first saves [cijp] at initial condition --
 IF( noderank == 0 )THEN
  IF( step_beg == 1 )THEN
   CALL CIJP_WRITE(keff, step_beg-1, cijp, sigcijp, 30)
!
! -- Master process calculates initial region-wise fission rate --
   IF( Nreg == 1 )THEN
    fr(0,1) = Ipwr/Fise/Mevtoj
   ELSEIF( Nreg == 2 )THEN
    frac(1) = 1/( 1+( 1-cijp(Kcutp,1,1)/(keff*K_adjust/K_init))/cijp(Kcutp,1,2)/(keff*K_adjust/K_init) )
    fr(0,1) = frac(1)*Ipwr/Fise/Mevtoj
    frac(2) = 1.0-frac(1)
    fr(0,2) = frac(2)*Ipwr/Fise/Mevtoj
!    WRITE(*,*) "fr(0,1) is ", fr(0,1)
!    WRITE(*,*) "fr(0,2) is ", fr(0,2)
   ELSE
!
    DO i=1, Nreg
     DO j=1, Nreg
      ccij0(i,j) = cijp(Kcutp,i,j)/(keff*K_adjust/K_init)
      IF ( i == j) THEN
       ccij0(i,j) = ccij0(i,j) - 1.D0
      END IF
     END DO
    END DO
!
!
    DO i=1, Nreg-1
     rfrac(i) = - ccij0(i,Nreg)
     DO j=1, Nreg-1
      cijmat(i,j) = ccij0(i,j) - ccij0(i,Nreg)
     END DO
    END DO
!
    CALL SGETRF(Nreg-1,Nreg-1,cijmat,Nreg-1,piv,info1)
!
    CALL SGETRS('N',Nreg-1,1,cijmat,Nreg-1,piv,rfrac,Nreg-1,info2)
!
    DO I=1, Nreg-1
     frac(i) = rfrac(i)
     fr(0,i) = frac(i)*Ipwr/Fise/Mevtoj
    END DO
    frac(Nreg) = 1.D0 - SUM(rfrac)
    fr(0,Nreg) = frac(Nreg)*Ipwr/Fise/Mevtoj
!
   END IF
   sigfr(0,:) = 0.0D0
```

```
!
 END IF
 END IF
!
! -- Synchronize MPI processes --
 CALL MPI_Barrier(nodecomm,ierr)
!
! -- All processes read [cijp0] if step_beg < Kcutp --
 IF( 0 < step_beg .AND. step_beg < Kcutp )THEN
  CALL CIJP_READ(30, 0, cijp0, sigcijp0, estat)
! -- Stop if there was error in calling CIJP_READ --
  IF( estat /= 0 )THEN
   OPEN(100,FILE="error.txt",FORM='FORMATTED',STATUS='REPLACE')
   CLOSE(100)
   STOP
  END IF
 END IF
!
! -- Synchronize MPI processes --
 CALL MPI_Barrier(nodecomm,ierr)
!
! --- Start of FORWARD calculation ---
!
 FORWARDloop: DO step=step_beg, Nstep
!
  stopstep = step
!
! -- First, master process reads 'reactivity.dat' and inserts reactivity at step=0 --
  IF( step == 0 )THEN
   CALL REAC_INSERT(step, 2, Lrfile, Rfile, rvar, estat, rins)
! -- Stop if there was an error in calling REAC_INSERT --
   IF( estat /= 0 )THEN
    OPEN(100,FILE="error.txt",FORM='FORMATTED',STATUS='REPLACE')
    CLOSE(100)
    STOP
   END IF
!
   IF ( rins == "Y" )THEN
    IF( noderank > 0 )THEN
     EXIT FORWARDloop
    ELSEIF( noderank == 0 )THEN
     uvar(:,:) = uvara(0,:,:)
     OPEN(40, FILE='update_vars.nml', STATUS='NEW', DELIM='APOSTROPHE')
     WRITE(40, NML = uvarnml)
     WRITE(40, NML = rvarnml)
     CLOSE(40)
!
     OPEN(40, FILE='nonupdate_vars.nml', STATUS='NEW', DELIM='APOSTROPHE')
     WRITE(40, NML = nvarnml)
     CLOSE(40)
     EXIT FORWARDloop
    END IF
   END IF
  END IF
!
! -- Calculate [fr] & [sigfr] at the current time step --
! -- First, master process prepares [frPTR] & [sigfrPTR] --
  IF( step < Kcutp )THEN
   k_beg = 0
   k_end = step - 1
   IF( noderank == 0 )THEN
    frPTR(1:step,:) = fr(k_beg:k_end,:)
    frPTR(step+1:Kcutp,:) = 0.D0
    sigfrPTR(1:step,:) = sigfr(k_beg:k_end,:)
    sigfrPTR(step+1:Kcutp,:) = 0.D0
   END IF
```

```
 ELSEIF( step >= Kcutp )THEN
  k_beg = step - Kcutp
  k_end = step - 1
  IF ( noderank == 0 )THEN
   frPTR(1:Kcutp,:) = fr(k_beg:k_end,:)
   sigfrPTR(1:Kcutp,:) = sigfr(k_beg:k_end,:)
  END IF
! -- All processes set their part of [cijp0] and [sigcijp0] to 0.D0 because they are not used in the calculation
!    when step >= Kcutp, but are argument for calling IKM subroutine below--
  cijp0(:,reg_beg:reg_end,:) = 0.D0
  sigcijp0(:,reg_beg:reg_end,:) = 0.D0
 END IF
!
! -- Synchronize MPI processes --
 CALL MPI_Barrier(nodecomm,ierr)
!
! -- All process call IKM subroutine to calculate their part of [curPTR]
 IF( step < Kcutp )THEN
  CALL IKM(reg_beg,reg_end,k_beg,k_end,cijp(:,reg_beg:reg_end,:),sigcijp(:,reg_beg:reg_end,:), &
       cijp0(:,reg_beg:reg_end,:), sigcijp0(:,reg_beg:reg_end,:),frPTR(1:step,:), &
       sigfrPTR(1:step,:),curPTR(reg_beg:reg_end,1), curPTR(reg_beg:reg_end,2), estat)
 ELSEIF( step >= Kcutp )THEN
  CALL IKM(reg_beg,reg_end,k_beg,k_end,cijp(:,reg_beg:reg_end,:),sigcijp(:,reg_beg:reg_end,:), &
       cijp0(:,reg_beg:reg_end,:), sigcijp0(:,reg_beg:reg_end,:),frPTR(:,:), &
       sigfrPTR(:,:),curPTR(reg_beg:reg_end,1), curPTR(reg_beg:reg_end,2), estat)
 END IF
! -- Stop if error occurred in calling IKM subroutine --
 IF( estat /= 0 )THEN
  OPEN(100,FILE="error.txt",FORM='FORMATTED',STATUS='REPLACE')
  CLOSE(100)
  STOP
 END IF
!
! -- Synchronize MPI processes --
 CALL MPI_Barrier(nodecomm,ierr)
!
! -- Master process calls feedback.f90 to calculate [uvara(step,:,:)] and [nvar]--
 IF( noderank == 0 )THEN
  fr(step,:) = curPTR(:,1)
  sigfr(step,:) = curPTR(:,2)
  uvarprev(:,:) = uvara(step-1,:,:)
  nvarprev(:,:) = nvar(:,:)
  frprev(:) = fr(step-1,:)
  CALL FEEDBACK(frprev, uvarprev, nvarprev, uvarcur, nvar, estat)
! -- Stop if error occurred in calling FEEDBACK --
  IF( estat /= 0 )THEN
   OPEN(100,FILE="error.txt",FORM='FORMATTED',STATUS='REPLACE')
   CLOSE(100)
   STOP
  END IF
  uvara(step,:,:) = uvarcur(:,:)
  duvar(:,:) = uvara(step,:,:) - uvar_last(:,:)
  duvarPTR(:,:) = duvar(:,:)
 END IF
!
! -- Master process outputs [cijp] function at user-specified steps in [Out_list] if there is any --
 IF( noderank == 0 )THEN
  IF( Nout > 0 )THEN
   outloop: DO m=1, Nout
    IF( Out_list(m) == step )THEN
     CALL CIJP_WRITE(0.D0, step, cijp, sigcijp, 30)
     EXIT outloop
    END IF
   END DO outloop
  END IF
 END IF
```

```
!
! -- Master process checks the followings: --
!   -  Check-1: check if C_ij functions are to be updated or not based on update criteria --
!   -  Check-2: check if reactivity is to be inserted or not at the next step, i.e., at step+1 --
!
! -- Synchronize MPI processes --
  CALL MPI_Barrier(nodecomm,ierr)
!
! -- Check-1
  updcheck = 0
  checkloop: DO v=1, Nuvar
   DO i=1, Nreg
    IF( duvarPTR(i,v) >= Uvarcrit(i,v) )THEN
     updcheck = 1
     EXIT checkloop
    END IF
   END DO
  END DO checkloop
!
  IF( updcheck == 1  )THEN
   IF( noderank > 0 )THEN
    EXIT FORWARDloop
   ELSEIF( noderank == 0 )THEN
! -- First, write update & reactivity variables
    uvar(:,:) = uvara(step,:,:)
    OPEN(40, FILE='update_vars.nml', STATUS='REPLACE', DELIM='APOSTROPHE')
    WRITE(40, NML = uvarnml)
    DO u=1, Nuvar
     DO i=1, Nreg
      DO v=1, Nrvar
       WRITE(num,'(I3)') i
       para = Uvarn(u)//TRIM(ADJUSTL(num))
       IF( TRIM(para) == rvarn(v) )THEN
        rvar(v) = uvar(i,u)
       END IF
      END DO
     END DO
    END DO
    WRITE(40, NML = rvarnml)
    CLOSE(40)
!
! -- Then, write non-update variables
    OPEN(40, FILE='nonupdate_vars.nml', STATUS='REPLACE', DELIM='APOSTROPHE')
    WRITE(40, NML = nvarnml)
    CLOSE(40)
    EXIT FORWARDloop
   END IF
  END IF
!
! -- Check-2:
  IF( step+1 <= Riend )THEN
! -- All processes open [Rfile] file and reads reactivity insertion data --
   CALL REAC_INSERT(step+1, 2, Lrfile, Rfile, rvar, estat, rins)
! -- Stop if there was an error in calling REAC_INSERT --
   IF( estat /= 0 )THEN
    OPEN(100,FILE="error.txt",FORM='FORMATTED',STATUS='REPLACE')
    CLOSE(100)
    STOP
   END IF
! -- Master process inserts reactivity if necessary --
   IF( rins == "Y" )THEN
    IF( noderank > 0)THEN
     EXIT FORWARDloop
    ELSEIF( noderank == 0 )THEN
! -- First, write update & reactivity variables
     uvar(:,:) = uvara(step,:,:)
```

```
    OPEN(40, FILE='update_vars.nml', STATUS='REPLACE', DELIM='APOSTROPHE')
    WRITE(40, NML = uvarnml)
    WRITE(40, NML = rvarnml)
    CLOSE(40)
!
! -- Then, write non-update variables
    OPEN(40, FILE='nonupdate_vars.nml', STATUS='REPLACE', DELIM='APOSTROPHE')
    WRITE(40, NML = nvarnml)
    CLOSE(40)
    EXIT FORWARDloop
   END IF
  END IF
 END IF
!
! -- Synchronize MPI processes --
 CALL MPI_Barrier(nodecomm,ierr)
!
! -- End of FORWARD calculation --
 END DO FORWARDloop
!
! -- Master process writes [fr(step_beg:stopstep,:)] and [uvara(step_beg:stopstep,:,:)]
!    to "fr_dat.dat" and "xxxx_dat.dat" respectively --
 IF( noderank == 0 )THEN
  IF( step_beg == 1 )THEN
   CALL FR_WRITE(0, stopstep, fr(0:stopstep,:), sigfr(0:stopstep,:), 10, "NEW", "REWIND", estat)
! -- Stop if there was error writing --
   IF( estat /= 0 )THEN
    OPEN(100,FILE="error.txt",FORM='FORMATTED',STATUS='REPLACE')
    CLOSE(100)
    STOP
   ENDIF
!
  ELSEIF( step_beg > 1 )THEN
   CALL FR_WRITE(step_beg, stopstep, fr(step_beg:stopstep,:), sigfr(step_beg:stopstep,:), 10, "OLD", "APPEND", estat)
! -- Stop if there was error writing --
   IF( estat /= 0 )THEN
    OPEN(100,FILE="error.txt",FORM='FORMATTED',STATUS='REPLACE')
    CLOSE(100)
    STOP
   ENDIF
  END IF
!
  IF( step_beg == 1 )THEN
   DO v=1, Nuvar
    u = v + 20
    CALL UVAR_WRITE(0, stopstep, uvara(0:stopstep,:,v), Uvarn(v), u, "NEW", "REWIND", estat)
! -- Stop if there was error writing --
    IF( estat /= 0 )THEN
     OPEN(100,FILE="error.txt",FORM='FORMATTED',STATUS='REPLACE')
     CLOSE(100)
     STOP
    ENDIF
   END DO
  ELSEIF( step_beg > 1 )THEN
   DO v=1, Nuvar
    u = v + 20
    CALL UVAR_WRITE(step_beg, stopstep, uvara(step_beg:stopstep,:,v), Uvarn(v), u, "OLD", "APPEND", estat)
! -- Stop if there was error writing --
    IF( estat /= 0 )THEN
     OPEN(100,FILE="error.txt",FORM='FORMATTED',STATUS='REPLACE')
     CLOSE(100)
     STOP
    ENDIF
   END DO
  END IF
!
```

```
! -- Master process writes [nvarnml] --
 OPEN(40, FILE='nonupdate_vars.nml', STATUS='REPLACE', DELIM='APOSTROPHE')
 WRITE(40, NML = nvarnml)
 CLOSE(40)
! -- Master process writes [stopstep] to "update_steps.dat" --
 OPEN(1,FILE='update_steps.dat',FORM='FORMATTED',STATUS='OLD',ACTION='WRITE',POSITION='APPEND')
 WRITE(1,'(I10)') stopstep
 CLOSE(1)
 END IF
!
! -- Synchronize MPI processes --
 CALL MPI_Barrier(nodecomm,ierr)
!
 CALL MPI_Win_free(win1,ierr)
 CALL MPI_Win_free(win2,ierr)
 CALL MPI_Win_free(win3,ierr)
 CALL MPI_Win_free(win4,ierr)
 CALL MPI_Win_free(win5,ierr)
!
! -- All processes finalize MPI --
 CALL MPI_Finalize(ierr)
 STOP
!
! 1010 FORMAT(I5, ES16.8, ES16.8, ES16.8)  ! For UNIT=10 or fr_dat.dat
! 1020 FORMAT(I5, ES16.8, ES16.8)    ! For UNIT=U or xxxx_dat.dat
! 1030 FORMAT(2I5, ES16.8, ES16.8, ES16.8) ! For UNIT=30 or cij_xxxxxxxxxx.dat
 1002 FORMAT(ES16.8, ES16.8) ! For UNIT=2 or reactivity.dat
!
 END PROGRAM
```

5. *mod_values.f90* module

```
MODULE MOD_VALUES
!
!=============================================================================================================
!
! Module: MOD_VALUES ( Module containing declaration of constants, parameters, and arrays )
! Used wholly or partly by: FORWARD or FORWARD_MPI, FEEDBACK, UPDATE
!
! -------------------------------------------------------------------------------------------------------
! << MIK - Multi-region Integral Kinetic code, Version 1.0 >> by Delgersaikhan Tuya, Hiroki Takezawa, and
!    Toru Obara.
!    Laboratory for Advanced Nuclear Energy, Institute of Innovative Research, Tokyo Institute of Technology.
!
!    Date of release of code:  December 1, 2018
!
! Written in original form by: Delgersaikhan TUYA (Sep, 2018)
! -------------------------------------------------------------------------------------------------------
!=============================================================================================================
!
#ifdef MIK_MPI
! -- For MPI run --
  USE mpi
  USE, INTRINSIC :: ISO_C_BINDING
#endif
  IMPLICIT NONE
!
!--------------------------------- BEGINNING OF USER-DEFINED PARAMETERS ---------------------------------
! -- Basic IKM parameters --
  INTEGER(KIND=4), PARAMETER :: Nreg=2  ! Number of fissile regions
  INTEGER(KIND=4), PARAMETER :: Nstep=6000000     ! Total number of time steps (calculation runs till NSTEP)
  REAL(KIND=KIND(1.D0)), PARAMETER ::  Dt=1.0D-10  ! Size of time step (s)
  INTEGER(KIND=4), PARAMETER :: Kcutp=9999      ! Number of data points for C_ij function for prompt neutron
```

```
  INTEGER(KIND=4), PARAMETER :: Ipwr=1.0D0  ! Initial steady-state power (W)
  REAL(KIND=KIND(1.D0)), PARAMETER :: Mevtoj=1.6021D-13 ! MeV to Joule
  REAL(KIND=KIND(1.D0)), PARAMETER :: Fise=1.78D2   ! Energy release per fission (MeV/f)
!
! -- Parameters for MVP special tally and couplings --
  CHARACTER(len=10), PARAMETER :: Inpname='Godiva.inp' ! Name of the initial MVP input file
  INTEGER(KIND=4), PARAMETER :: Ncoup=4  !  Total number of region couplings
  INTEGER(KIND=4), PARAMETER :: Ilen=4  !  Length of label IDs for region couplings (Must be 5>=Ilen)
  INTEGER(KIND=4), PARAMETER :: Plen=4  !  Length of prefix for coupling (Change it depending on ilen. Must be Ilen+Plen=8)
  CHARACTER(len=PLEN), PARAMETER :: Prefix="ID= "   ! Prefix for label ID number. For example, "ID= 1000"
                                    ! (Add necessary plen-3 spaces after "ID=")
  INTEGER(KIND=4), PARAMETER :: Klen=46   ! Keyword1 & cond1 length
  CHARACTER(len=KLEN), PARAMETER ::  Cond1="RESULTS BY THE PRINCIPLE OF MAXIMUM LIKELIHOOD" ! Keyword to find [keff] in
! mvpout.ft06 file
  CHARACTER(len=ILEN), PARAMETER :: Tid_list(Ncoup) = (/"1000","1001","1002","1003"/) ! Special tally ID list for region couplings
  INTEGER(KIND=4), PARAMETER :: Clen=7  !  Length of coupling (below) for coupled regions
  CHARACTER(len=CLEN), PARAMETER :: Coup_list(Ncoup) = (/"001,001","001,002","002,001","002,002"/) ! List for region couplings. Order
! must correspond to that of Tid_list. Format is "iii,jjj", where iii is target region # and jjj is source region #.
!
! -- Parameters related to feedback, reactivity insertion, and UPDATE program --
!  - For update variables
  REAL(KIND=KIND(1.D0)), PARAMETER ::  Mass(Nreg)=(/2.716755D1,2.716755D1/)  ! Region-wise initial masses (kg)
! ( Note: Other parameters related to feedback, such as region-wise specific heat etc, are declared in FEEDBACK subroutine )
  INTEGER(KIND=4), PARAMETER :: Nuvar=3        ! Number of region-wise update (i.e., feedback) variables to be updated
  INTEGER(KIND=4), PARAMETER :: Uvarnl = 4     ! Update variable base name length excluding region number (All update variable names have
! the same length)
  CHARACTER(len=Uvarnl), PARAMETER :: Uvarn(Nuvar)=(/"TEMP","DENS","RADI"/)    ! List of update variable base names without region
! number. Region numbers will be concatenated automatically e.g., if update variable base name is TEMP and region number is 3, then it will
! be TEMP3
  INTEGER(KIND=4), PARAMETER :: Uvar_step_read = 1     ! Number of necessary time steps to read for [uvara] when starting FORWARD
! program. Basically set it to 1
  REAL(KIND=KIND(1.D0)), PARAMETER :: Iuvar(Nreg,Nuvar)=reshape( (/2.98D2,2.98D2,1.871D1,1.871D1,7.0247D0,8.8506D0/), &
                                      (/Nreg,Nuvar/) )  ! Initial values of region-wise update variables
  REAL(KIND=KIND(1.D0)), PARAMETER :: Uvarcrit(Nreg,Nuvar)=reshape( (/3.0D0,3.0D0,5.0D-2,5.0D-2,5.0D-3,5.0D-3/), &
                                    (/Nreg,Nuvar/) )  ! Update criteria for corresponding region-wise update variable
! -- For non-update variables
  INTEGER(KIND=4), PARAMETER :: Nnvar = 1    ! Number of region-wise non-update variables
  INTEGER(KIND=4), PARAMETER :: Nvarnl = 4   ! Non-update variable base name length excluding region number
  CHARACTER(len=Nvarnl), PARAMETER :: Nvarn(Nnvar)=(/"ENER"/)   ! List of non-update variable base names
  REAL(KIND=KIND(1.D0)), PARAMETER :: Invar(Nreg,Nnvar)=reshape( (/0.D0,0.D0/), (/Nreg, Nnvar/) )   ! Initial values of region-wise non-update
! variables
!
!  - For reactivity inserion
  INTEGER(KIND=4), PARAMETER :: Nrvar=2       ! Number of variables for reactivity insertion
  INTEGER(KIND=4), PARAMETER :: Rvarnl = 5     ! Reactivity variable full name length (All reactivity variable names have the same length)
  CHARACTER(len=Rvarnl), PARAMETER :: Rvarn(Nrvar)=(/"RADI1","RADI2"/)    ! List of reactivity variable full names
  INTEGER(KIND=4), PARAMETER :: Riend = 5   ! Time step corresponding to the reactivity insertion end
  INTEGER, PARAMETER :: Lrfile = 14      ! Character length of reactivity insertion file name [Rfile]
  CHARACTER(len=Lrfile), PARAMETER :: Rfile = "reactivity.dat"    ! Reactivity insertion file
!  - For UPDATE program
  INTEGER(KIND=4), PARAMETER :: Dlen=5   ! Maximum number of digit for region number representation (Just set Dlen=5 if Nreg<100000)
  CHARACTER(len=Dlen) :: num    ! Placeholder variable for region number representation
  CHARACTER(len=MAX(Uvarnl,Rvarnl)+Dlen) :: para   ! Placeholder variable for update variable name
  INTEGER(KIND=4), PARAMETER :: Llen=73  ! Length of each line in Inpname file for reading (Must be Llen>72)
  CHARACTER(len=2), PARAMETER :: Sign='% '  ! Sign indicating input parameter in Inpname file
  INTEGER(KIND=4), PARAMETER :: Vlen=11  ! Field width (i.e., number of characters) of update variable value
                        ! (E.g., for 2.3300E-04, Vlen=11 including sign of positive or negative)
  CHARACTER(len=8), PARAMETER :: Val_fmt="(ES11.4)" ! Format to write update variable value ( Set a field width
                                  ! same as Vlen )
!
! -- Parameters used for slightly adjusting [keff] and C_ij functions (used to make exact comparison with other calculation) --
!    [ If no adjustment is needed, set both K_adjust and K_init to 1.0D0 (i.e., K_adjust=1.0D0, K_init=1.0D0 ) ]
  REAL(KIND=KIND(1.D0)), PARAMETER :: K_adjust=1.000546D0   ! K-eff, to which others are adjusted. /used for little adjustment/
  REAL(KIND=KIND(1.D0)), PARAMETER :: K_init=1.00055D0   ! Actual (non-adjusted) K-eff at the condition just after first reactivity insertion.
!
! -- Parameters related to outputting C_ij functions --
```

```
  INTEGER(KIND=4), PARAMETER :: Nout=1  ! Number of time steps at which C_ij is to be outputted
  INTEGER(KIND=4), PARAMETER :: Out_list(Nout)=(/2500000/) ! List of time steps at which C_ij to be outputted
  INTEGER(KIND=4), PARAMETER :: Nlen=10   ! Max length of time step part of C_ij file name
  CHARACTER(len=Nlen) :: fnum
  CHARACTER(len=8), PARAMETER :: Cij_fmt="(I10.10)"  ! Format for C_ij file name. Should be (I<Nlen>.<Nlen>)
!
!------------------------------------------- END OF USER-DEFINED PARAMETERS -------------------------------------------
!
! -- Variables for IKM --
  INTEGER(KIND=4) :: step, stopstep, freg, sreg, i, j, k, l, m, n, o, updcheck
  INTEGER(KIND=4) :: u, v
  INTEGER(KIND=4) :: skip_num, k_beg, k_end
  INTEGER(KIND=4) :: step_end, step_beg, freg_read, n_react, k_react
  REAL(KIND=KIND(1.D0)) :: keff, time_read, val_read, err_read
  CHARACTER(len=10) :: dummychar
!
! -- Arrays for IKM --
  REAL(KIND=KIND(1.D0)), DIMENSION(0:Kcutp,Nreg,Nreg) :: cijp, sigcijp  ! Array for C_ij and its 1sigma data for prompt neutron
#ifndef MIK_MPI
  REAL(KIND=KIND(1.D0)), DIMENSION(0:Kcutp,Nreg,Nreg) :: cfijp, sigcfijp  ! Array for cumulative fission and its 1sigma data for prompt neutron
  REAL(KIND=KIND(1.D0)), DIMENSION(Nreg,2) :: curfr      ! Fission rate at current time step (used for single mode FORWARD)
#endif
  REAL(KIND=KIND(1.D0)), DIMENSION(Nreg) :: tcfp, sigtcfp  ! Total cumulative fission in each region
  REAL(KIND=KIND(1.D0)), DIMENSION(:,:), ALLOCATABLE :: fr, sigfr ! Region-wise fission rate [f/s]
  REAL(KIND=KIND(1.D0)), DIMENSION(0:Kcutp,Nreg,Nreg) :: cijp0, sigcijp0  ! C_ij at initial condition for prompt neutron
  REAL(KIND=KIND(1.D0)), DIMENSION(Nreg-1,Nreg-1) :: cijmat
  REAL(KIND=KIND(1.D0)), DIMENSION(Nreg) :: frac  ! Region-wise fission rate (or power) fraction
  REAL(KIND=KIND(1.D0)), DIMENSION(Nreg-1) :: rfrac
  REAL(KIND=KIND(1.D0)), DIMENSION(Nreg, Nreg) :: ccij0
  INTEGER, DIMENSION(1:Nreg-1) :: piv
  INTEGER :: info1, info2
  INTEGER :: estat, astat, count_a
  CHARACTER(len=1) :: rins
  LOGICAL :: lstat
!
#ifdef MIK_MPI
! -- Variables and pointers related to MPI --
  INTEGER :: ierr, idtsk, n_procs, n_regtask
  INTEGER(KIND=4) :: reg_beg, reg_end   ! Region ranges for given process/task
  INTEGER :: STATUS(mpi_status_size)
  INTEGER :: win1, win2, win3, win4, win5, win6  ! Name of shared-memory windows
  INTEGER :: nodecomm, noderank, disp_unit  ! MPI communication on a node with shared-memory architecture
  INTEGER(KIND=MPI_ADDRESS_KIND) :: winsize1, winsize2, winsize3, winsize4, winsize5, winsize6 ! Size of shared-memory windows
  TYPE(C_PTR) :: baseptr1, baseptr2, baseptr3, baseptr4, baseptr5, baseptr6 ! Base pointers for shared-memory windows
  REAL(KIND=KIND(1.D0)), POINTER, DIMENSION(:,:,:) :: cfijpPTR, sigcfijpPTR ! Pointer for cumulative fission for prompt neutron
  REAL(KIND=KIND(1.D0)), POINTER, DIMENSION(:,:) :: frPTR, sigfrPTR ! Pointer for fission rate
  REAL(KIND=KIND(1.D0)), POINTER, DIMENSION(:,:) :: curPTR ! Pointer for fission rate [fr] and its sigma [sigfr] at current time step
  REAL(KIND=KIND(1.D0)), POINTER, DIMENSION(:,:) :: duvarPTR ! Pointer for [duvar]
#endif
!
! -- Variables for feedback --
  REAL(KIND=KIND(1.D0)), DIMENSION(:,:,:), ALLOCATABLE :: uvara ! A whole array for update variables
  REAL(KIND=KIND(1.D0)), DIMENSION(Nreg,Nuvar) :: uvar_last ! Update variables at last step
  REAL(KIND=KIND(1.D0)), DIMENSION(Nreg,Nuvar) :: duvar ! Difference of update variable values at last update step and current time step
  REAL(KIND=KIND(1.D0)), DIMENSION(Nreg) :: frprev  !  Region-wise fission rate at [step-1]
  REAL(KIND=KIND(1.D0)), DIMENSION(Nreg,Nuvar) :: uvarprev ! Update variable values at [step-1]
  REAL(KIND=KIND(1.D0)), DIMENSION(Nreg,Nuvar) :: uvarcur  !  Update variable values at [step]
  REAL(KIND=KIND(1.D0)), DIMENSION(Nreg,Nuvar) :: uvar   ! Update variables at update step
  REAL(KIND=KIND(1.D0)), DIMENSION(Nrvar) :: rvar   ! Reactivity insertion variables at update/or insertion step
  REAL(KIND=KIND(1.D0)), DIMENSION(Nreg) :: CAPA  ! Region-wise heat capacity [J/K]
!
  REAL(KIND=KIND(1.D0)), DIMENSION(Nreg,Nnvar) :: nvar      !  Non-update variables at [step]
  REAL(KIND=KIND(1.D0)), DIMENSION(Nreg,Nnvar) :: nvarprev  !  Non-update variables at [step-1]
  NAMELIST / uvarnml / uvar  ! Namelist for update variables
  NAMELIST / rvarnml / rvar  ! Namelist for reactivity variables
  NAMELIST / nvarnml / nvar  ! Namelist for non-update variables
```

```
!
 END MODULE MOD_VALUES
```

6. *mod_feedback.f90* module

```
MODULE MOD_FEEDBACK
!
!=============================================================================================================
!
! Module: MOD_FEEDBACK ( Module containing FEEDBACK subroutine )
! Used by: FORWARD or FORWARD_MPI program
!
! Definition of each subroutine is included at the beginning of the subroutine.
!
! -------------------------------------------------------------------------------------------------
! << MIK - Multi-region Integral Kinetic code, Version 1.0 >> by Delgersaikhan Tuya, Hiroki Takezawa, and
!    Toru Obara.
!    Laboratory for Advanced Nuclear Energy, Institute of Innovative Research, Tokyo Institute of Technology.
!
!    Date of release of code:  December 1, 2018
!
! Written in original form by: Delgersaikhan TUYA (Sep, 2018)
! -------------------------------------------------------------------------------------------------
!=============================================================================================================
!
 IMPLICIT NONE
!
 CONTAINS
!
!*************************************************************************************************************
!
 SUBROUTINE FEEDBACK(frprev,uvarprev,nvarprev,uvarcur,nvarcur,estat)
!=============================================================================================================
!
! Subroutine: FEEDBACK ( Subroutine to calculate update variables at each time step )
! Called from: FORWARD or FORWARD_MPI program
!
! This subroutine takes the values of update & non-update variables at previous time step [step-1], calculates
! their step-wise changes, and returns the values of update & non-update variables at current time step [step].
!
! --- Modification/change log ---
! - Written in original form by Delgersaikhan Tuya (Aug, 2018)
!
!=============================================================================================================
!
 USE mod_values, ONLY: Nreg, Nuvar, Nnvar, Dt, Capa, Mass, Iuvar, Mevtoj, Fise
!
 REAL(KIND=KIND(1.D0)), INTENT(IN), DIMENSION(Nreg) :: frprev ! Region-wise fission rate at previous step [step-1]
 REAL(KIND=KIND(1.D0)), INTENT(IN), DIMENSION(Nreg,Nuvar) :: uvarprev ! Update variable at previous step [step-1]
 REAL(KIND=KIND(1.D0)), INTENT(IN), DIMENSION(Nreg,Nnvar) :: nvarprev ! Non-update variable at previous step [step-1]
 REAL(KIND=KIND(1.D0)), INTENT(OUT),DIMENSION(Nreg,Nuvar) :: uvarcur  ! Update variable at current step [step]
 REAL(KIND=KIND(1.D0)), INTENT(OUT),DIMENSION(Nreg,Nnvar) :: nvarcur  ! Non-update variable at current step [step]
 INTEGER, INTENT(OUT) :: estat   ! Error status (0 means no error)
!
! -- Local variables
 INTEGER :: i, v
 REAL(KIND=KIND(1.D0)), DIMENSION(nreg) :: vrat, vol, vol0
 REAL(KIND=KIND(1.D0)), PARAMETER :: Pi=3.1415D0
 REAL(KIND=KIND(1.D0)), PARAMETER :: Au=235.201D0
 REAL(KIND=KIND(1.D0)), PARAMETER :: Ac=28.4264D0, Bc=-6.9587D-3, Cc=2.98744D-5, Ec=-1.1888D5
 REAL(KIND=KIND(1.D0)), PARAMETER :: Av=3.203D-5, Bv=3.287D-8, Cv=1.625D-11
!
! -- Set estat = 0 for now --
```

```
 estat = 0
!
 vol0(1) = 4.0/3.0*Pi*Iuvar(1,3)**3
 vol0(2) = 4.0/3.0*Pi*Iuvar(2,3)**3 - vol0(1)
 DO I=1, Nreg
  vrat(i) = 1.0D0 + Av*(uvarprev(i,1)-2.98D2) + Bv*(uvarprev(i,1)-2.98D2)**2 &
       + Cv*(uvarprev(i,1)-2.98D2)**3
  vol(i) = vrat(I)*vol0(i)
 END DO
!
 DO i=1, Nreg
  nvarcur(i,1) = nvarprev(i,1) + Mevtoj*Fise*frprev(i)*Dt
 END DO
!
 DO i=1, Nreg
  Capa(i) = Mass(i)*(1.0/(Au*1.0D-3))* &
            ( Ac + Bc*uvarprev(i,1) + Cc*uvarprev(i,1)**2 &
            + Ec*uvarprev(i,1)**(-2) )
  uvarcur(i,1) = uvarprev(i,1) + Mevtoj*Fise*frprev(i)*Dt/Capa(i)   ! Temp
  uvarcur(i,2) = (1.0D3)*Mass(i) / vol(i)
 END DO
 uvarcur(1,3) = ( 3.0/4.0/Pi*vol(1) )**(1.0/3)
 uvarcur(2,3) = ( 3.0/4.0/Pi*( vol(1)+vol(2) ) )**(1.0/3)
!
 END SUBROUTINE
!
!*************************************************************************************************************
!
 END MODULE MOD_FEEDBACK
```

7. *mod_comp_subs.f90* module

```
MODULE MOD_COMP_SUBS
!
!=============================================================================================================
!
! Module: MOD_COMP_SUBS ( module containing computational subroutines )
! Used by: FORWARD or FORWARD_MPI program
!
! Definition of each subroutine is included at the beginning of the subroutine.
!
! -------------------------------------------------------------------------------------------------------------
! << MIK - Multi-region Integral Kinetic code, Version 1.0 >> by Delgersaikhan Tuya, Hiroki Takezawa, and
!    Toru Obara.
!    Laboratory for Advanced Nuclear Energy, Institute of Innovative Research, Tokyo Institute of Technology.
!
!    Date of release of code:  December 1, 2018
!
! Written in original form by: Delgersaikhan TUYA (Sep, 2018)
! -------------------------------------------------------------------------------------------------------------
!=============================================================================================================
!
 IMPLICIT NONE
!
 CONTAINS
!
!*************************************************************************************************************
!
 SUBROUTINE IKM(reg_beg, reg_end, k_beg, k_end, cijp, sigcijp, cijp0, sigcijp0, fr, sigfr, curfr, cursigfr, estat)
!=============================================================================================================
!
! Subroutine: IKM ( Subroutine to compute [fr] and [sigfr] at the current time step k_end+1 )
! Called from: FORWARD or FORWARD_MPI
```

```
!
! --- Modification/change log ---
! - The content of this subroutine was initially developed by Hiroki TAKEZAWA (2008)
! - The modifications regarding parallel calculation was made by Delgersaikhan Tuya (2018)
!   -- A given task/core is responsible for calculating [fr] & [sigfr] for only the range [reg_beg, reg_end].
!
!=============================================================================================================
!
 USE mod_values, ONLY: Nreg, Kcutp
!
 INTEGER(KIND=4), INTENT(IN) :: reg_beg, reg_end    ! Beginning and ending regions, respectively
 INTEGER(KIND=4), INTENT(IN) :: k_beg, k_end       ! Beginning and ending indices for [fr], k_end=step-1
 REAL(KIND=KIND(1.D0)), INTENT(IN), DIMENSION(0:Kcutp,reg_beg:reg_end,Nreg) :: cijp0, sigcijp0  ! C_ij function for prompt neutron at initial
condition
 REAL(KIND=KIND(1.D0)), INTENT(IN), DIMENSION(0:Kcutp,reg_beg:reg_end,Nreg) :: cijp, sigcijp    ! C_ij function for prompt neutron
 REAL(KIND=KIND(1.D0)), INTENT(IN), DIMENSION(k_beg:k_end, Nreg) :: fr     ! Past fission rates
 REAL(KIND=KIND(1.D0)), INTENT(IN), DIMENSION(k_beg:k_end, Nreg) :: sigfr  ! Past fission rate uncertainty
 REAL(KIND=KIND(1.D0)), INTENT(OUT), DIMENSION(reg_beg:reg_end) :: curfr     ! Fission rate at current step
 REAL(KIND=KIND(1.D0)), INTENT(OUT), DIMENSION(reg_beg:reg_end) :: cursigfr  ! Fission rate uncertainty at current step
 INTEGER, INTENT(OUT) :: estat    ! Error status (0 means no error )
!
! -- Local variables
  INTEGER(KIND=4) :: step, m, j , i, k
  REAL(KIND=KIND(1.D0)), DIMENSION(reg_beg:reg_end,Nreg) :: pfr    ! Partial fission rate
  REAL(KIND=KIND(1.D0)), DIMENSION(reg_beg:reg_end) :: vnb, vna    ! Variance of fission rate (before and after initial reactivity insertion)
!
! -- Set estat = 0 for now --
  estat = 0
!
  step = k_end + 1
  pfr(reg_beg:reg_end,:)= 0.D0
  curfr(reg_beg:reg_end) = 0.D0
  vnb(reg_beg:reg_end) = 0.D0
  vna(reg_beg:reg_end) = 0.D0
!
! -- Calculate [curfr] inlcuding the contribution from initial steady-state condition --
  IF( step < Kcutp )THEN
   DO j=1, Nreg
    DO i=reg_beg, reg_end
     pfr(i,j) = pfr(i,j) + ( cijp0(Kcutp,i,j) - cijp0(step,i,j) )*fr(0,j)
     vnb(i) = ( sigcijp0(Kcutp,i,j)**2 + sigcijp0(step,i,j)**2 )*fr(0,j)**2
     DO k=k_beg, k_end
      m = k_end + 1 - k
      pfr(i,j) = pfr(i,j) + ( cijp(m,i,j) - cijp(m-1,i,j) )*fr(k,j)
      vna(i) = vna(i) + (sigfr(k,j)**2)*( cijp(m,i,j) - cijp(m-1,i,j) )**2 &
           + ( sigcijp(m,i,j)**2 + sigcijp(m-1,i,j)**2 )*fr(k,j)**2
     END DO
     curfr(i)=curfr(i) + pfr(i,j) ! Fission rate
    END DO
   END DO
   DO i=reg_beg, reg_end
    cursigfr(i)= SQRT( vnb(i) + vna(i) ) ! Sigma of fission rate
   END DO
!
! -- Calculate [curfr] not inlcuding the contribution from initial steady-state condition --
  ELSEIF( step >= Kcutp )THEN
!
   DO j=1, Nreg
    DO i=reg_beg, reg_end
     DO k=k_beg, k_end
      m = k_end + 1 - k
      pfr(i,j) = pfr(i,j) + ( cijp(m,i,j) - cijp(m-1,i,j) )*fr(k,j)
      vna(i) = vna(i) + (sigfr(k,j)**2)*( cijp(m,i,j)-cijp(m-1,i,j) )**2 &
           + ( sigcijp(m,i,j)**2 + sigcijp(m-1,i,j)**2 )*fr(k,j)**2
     END DO
     curfr(i) = curfr(i) + pfr(i,j)
```

```
   END DO
  END DO
  DO i=reg_beg, reg_end
   cursigfr(i)= SQRT( vnb(i) + vna(i) ) ! Sigma of fission rate
  END DO
 END IF
END SUBROUTINE
!
!*****************************************************************************************************************
!
 SUBROUTINE CFIJTALLY(reg_beg, reg_end, keff, cfijp, sigcfijp, estat)
!=================================================================================================================
!
! Subroutine: CFIJTALLY ( Subroutine to tally a cumulative fission data from MVP2.0/or MVP3.0 output file )
! Called from: FORWARD or FORWARD_MPI
!
! The following file must be input to this subroutine:
! mvpout.ft06 : MVP2.0/or MVP3.0 output file containing special tallies for time- and region-dependent fission
!             reaction
!
! --- Modification/change log ---
! - Written in original form by Delgersaikhan Tuya (Aug, 2018)
!
!=================================================================================================================
!
 USE mod_values, ONLY: Ncoup, Tid_list, Coup_list, Ilen, Clen, Plen, Klen, Prefix, Cond1, Nreg, Kcutp
!
 IMPLICIT NONE
!
! Dummy and  local variables & parameters
 INTEGER(KIND=4), INTENT(IN) :: reg_beg, reg_end
 REAL(KIND=KIND(1.D0)), INTENT(OUT) :: keff    ! k-eff
 REAL(KIND=KIND(1.D0)), INTENT(OUT), DIMENSION(0:Kcutp, reg_beg:reg_end, Nreg)  :: cfijp, sigcfijp ! Cumulative fission for prompt neutron
 INTEGER, INTENT(OUT) :: estat   ! Error status (0 means no error)
!
 CHARACTER(len=Ilen), ALLOCATABLE, DIMENSION(:) :: ptid_list   ! Special tally ID partial-list for corresponding process/task
 CHARACTER(len=Clen), ALLOCATABLE, DIMENSION(:) :: pcoup_list  ! Partial-list for couplings for corresponding process/task
 CHARACTER(len=Ilen) :: cha   ! Character for label ID number
 CHARACTER(len=Klen) :: keyword1
 CHARACTER(len=Ilen+Plen) :: keyword2, idc
 CHARACTER(len=Clen) :: coup
 CHARACTER(len=Clen/2) :: charfr
!
 INTEGER :: stat, info1, info2
 INTEGER :: step, h, i, j, k, hl, m, n, u
 INTEGER :: freg, sreg, count
!
 REAL(KIND=KIND(1.D0)) :: time1, time2, fdat, err
 REAL(KIND=KIND(1.D0)), DIMENSION(0:Kcutp, reg_beg:reg_end, Nreg) :: fijp, sigfijp, sigfijpp
 REAL(KIND=KIND(1.D0)), DIMENSION(0:Kcutp) :: timeb
 CHARACTER(len=Clen) :: tempFRSR
 CHARACTER(len=1) :: dmynum
 CHARACTER(len=4) :: cha_fmt
! ------------------------------------------------------------------------------------
!
! === Initialize arrays & variables ===
!
 estat = 0
 keff = 0.D0
 fijp(:,reg_beg:reg_end,:) = 0.D0
 sigfijp(:,reg_beg:reg_end,:) = 0.D0
 sigfijpp(:,reg_beg:reg_end,:) = 0.D0
 cfijp(:,reg_beg:reg_end,:) = 0.D0
 sigcfijp(:,reg_beg:reg_end,:) = 0.D0
 timeb = 0.D0
!
```

```
! --- Collect needed IDs from [Tid_list] and make partial list [ptid_list] ---
 count = 0
 hl = (Clen - 1)/2
 WRITE(dmynum,"(I1)") hl
 cha_fmt = "(I"//dmynum//")"
 DO i=reg_beg, reg_end
  DO n=1, Ncoup
   coup = Coup_list(n)
   READ(coup(1:hl),cha_fmt) m
   IF( m == i )THEN
    count = count + 1
   END IF
  END DO
 END DO
!
 ALLOCATE(ptid_list(count))
 ALLOCATE(pcoup_list(count))
!
 u = 1
 DO i=reg_beg, reg_end
  DO n=1, Ncoup
   coup = Coup_list(n)
   READ(coup(1:hl),cha_fmt) m
   IF( m == i )THEN
    ptid_list(u) = Tid_list(n)
    pcoup_list(u) = Coup_list(n)
    u = u + 1
   END IF
  END DO
 END DO
!
 OPEN(99,FILE='log.txt',FORM='FORMATTED',STATUS='OLD',ACTION='WRITE',POSITION='APPEND')
 DO i=1, count
  WRITE(99,'(1X,A44,I5,A3,A10)') "CFIJTALLY subroutine: IDs for process/task ",reg_beg-1,"is", ptid_list(i)
  WRITE(99,'(1X,A50,I5,A3,A10)') "CFIJTALLY subroutine: Couplings for process/task ",reg_beg-1,"is", pcoup_list(i)
  WRITE(*,'(1X,A44,I5,A3,A10)') "CFIJTALLY subroutine: IDs for process/task ",reg_beg-1,"is", ptid_list(i)
  WRITE(*,'(1X,A50,I5,A3,A10)') "CFIJTALLY subroutine: Couplings for process/task ",reg_beg-1,"is", pcoup_list(i)
 END DO
 CLOSE(99)
!
! === Open MVP output file "mvpout.ft06" and find [keff] ===
!
 OPEN(10,FILE="mvpout.ft06",FORM='FORMATTED',STATUS='OLD',ACTION='READ',IOSTAT=stat)
! -- Print out error message if any and return --
 IF ( stat > 0 ) THEN
  estat = 1
  OPEN(99,FILE='log.txt',FORM='FORMATTED',STATUS='OLD',ACTION='WRITE',POSITION='APPEND')
  WRITE(99,'(1X,A80)') "CFIJTALLY subroutine: ERROR MESSAGE: mvpout.f06 file does not exist! STOP"
  WRITE(*,'(1X,A80)') "CFIJTALLY subroutine: ERROR MESSAGE: mvpout.f06 file does not exist! STOP"
  CLOSE(99)
  RETURN
 END IF
!
! --- Read the file ---
!
 REWIND(10)
 searchloop1: DO
  READ(10,1100,IOSTAT=stat) keyword1
! *** Print out error message if any and return ***
  IF ( stat == -1 ) THEN
   estat = 1
   OPEN(99,FILE='log.txt',FORM='FORMATTED',STATUS='OLD',ACTION='WRITE',POSITION='APPEND')
   WRITE(99,*) "CFIJTALLY subroutine: Cannot find the keyword "//Cond1//" in the MVP output file. END OF FILE."
   WRITE(99,*) "Check if the keyword and its length are correctly defined. Also check 1001 FORMAT at the end of "// &
            " this subroutine and modify it according to the MVP output files."
   WRITE(*,*) "CFIJTALLY subroutine: Cannot find the keyword "//Cond1//" in the MVP output file. END OF FILE."
```

```
  WRITE(*,*) "Check if the keyword and its length are correctly defined. Also check 1001 FORMAT at the end of "// &
        " this subroutine and modify it according to the MVP output files."
  CLOSE(99)
  RETURN
 END IF
! --- Continue ---
 IF ( keyword1 == Cond1 ) THEN
  EXIT searchloop1
 ELSE
  CYCLE searchloop1
 END IF
END DO searchloop1
! --- Skip next 7 lines ---
DO j=1, 7
 READ(10,1100) keyword1
END DO
READ(10,1101,IOSTAT=stat) keff
! --- Print out read error message if any and return ---
IF ( stat > 0 ) THEN
 estat = 1
 OPEN(99,FILE='log.txt',FORM='FORMATTED',STATUS='OLD',ACTION='WRITE',POSITION='APPEND')
 WRITE(99,*) "CFIJTALLY subroutine: Cannot read the k-eff. Check and modify 1101 FORMAT at the end of this subroutine"// &
        " according to the MVP output files. BE CAREFUL with the FORMAT!!!"
 WRITE(*,*) "CFIJTALLY subroutine: Cannot read the k-eff. Check and modify 1101 FORMAT at the end of this subroutine"// &
        " according to the MVP output files. BE CAREFUL with the FORMAT!!!"
 CLOSE(99)
 RETURN
END IF
CLOSE(10)
!
! Print out [keff] for checking --
OPEN(99,FILE='log.txt',FORM='FORMATTED',STATUS='OLD',ACTION='WRITE',POSITION='APPEND')
WRITE(99,'(1X,A32,ES12.5)') "CFIJTALLY subroutine: k-eff is ", keff
WRITE(*,'(1X,A32,ES12.5)') "CFIJTALLY subroutine: k-eff is ", keff
CLOSE(99)
!
! === Open MVP output file again for [cfij] tally ===
!
OPEN(10,FILE="mvpout.ft06",FORM='FORMATTED',STATUS='OLD',ACTION='READ',POSITION='REWIND')
!
! --- Find the special tally part of the MVP output file by their ID's ---
!
WRITE(dmynum,"(I1)") Ilen
cha_fmt = "(A"//dmynum//")"
idloop: DO h=1, count
 WRITE(cha,cha_fmt) ptid_list(h)
 idc = Prefix//cha
 searchloop2: DO
  READ(10,1102,IOSTAT=stat) keyword2
! --- print out reading error message if any and return ---
  IF ( stat == -1 ) THEN
   estat = 1
   OPEN(99,FILE='log.txt',FORM='FORMATTED',STATUS='OLD',ACTION='WRITE',POSITION='APPEND')
   WRITE(99,*) "CFIJTALLY subroutine: Cannot find the special tally keyword "//idc//" in the MVP output files. END OF FILE."
   WRITE(99,*) "Check if the Prefix and Tid_list are correctly defined. Also check FORMAT 1000 at the end"// &
        " this subroutine if it is correctly defined. BE CAREFUL with the FORMAT!!!"
   WRITE(*,*) "CFIJTALLY subroutine: Cannot find the special tally keyword "//idc//" in the MVP output files. END OF FILE."
   WRITE(*,*) "Check if the Prefix and Tid_list are correctly defined. Also check FORMAT 1000 at the end"// &
        " this subroutine if it is correctly defined. BE CAREFUL with the FORMAT!!!"
   CLOSE(99)
   RETURN
  END IF
! --- continue if no error ---
  IF ( keyword2 == idc ) THEN
   EXIT searchloop2
  ELSE
```

```
   CYCLE searchloop2
  END IF
 END DO searchloop2
! --- Skip next 14 lines ---
 DO m=1, 14
  READ(10,1102) keyword2
 END DO
!
! --- Read data ---
 tempFRSR = pcoup_list(H)
 IF ( tempFRSR(hl+1:hl+1) == "," ) THEN
  READ(tempFRSR(1:hl),"(I3)") freg
  READ(tempFRSR(hl+2:Clen),"(I3)") sreg
 ELSE
  estat = 1
  OPEN(99,FILE='log.txt',FORM='FORMATTED',STATUS='OLD',ACTION='WRITE',POSITION='APPEND')
  WRITE(99,*) "CFIJTALLY subroutine: Wrong definition of region coupling pcoup_list= "//pcoup_list(H)// &
           ". ',' is expected. Please check and correct."
  WRITE(*,*) "CFIJTALLY subroutine: Wrong definition of region coupling pcoup_list= "//pcoup_list(H)// &
           ". ',' is expected. Please check and correct."
  CLOSE(99)
  RETURN
 END IF
!
 DO k=1, Kcutp
!
  IF( k <= 9999 )THEN
   READ(10,*,IOSTAT=stat) Step, time1, fdat
   IF ( stat > 0 ) THEN
    estat = 1
    OPEN(99,FILE='log.txt',FORM='FORMATTED',STATUS='OLD',ACTION='WRITE',POSITION='APPEND')
    WRITE(99,*) "CFIJTALLY subroutine: Cannot read the special tally data when k<10000. Check if the parameter Kcutp=" &
     ,Kcutp," is correctly defined. Also check if special tally data lines in the MVP output files include step, time,"// &
     " value in this order."
    WRITE(*,*) "CFIJTALLY subroutine: Cannot read the special tally data when k<10000. Check if the parameter Kcutp=" &
     ,Kcutp," is correctly defined. Also check if special tally data lines in the MVP output files include step, time,"// &
     " value in this order."
    CLOSE(99)
    RETURN
   END IF
  ELSE
   READ(10,1103,IOSTAT=stat) time1, fdat
   IF ( stat > 0 ) THEN
    estat = 1
    OPEN(99,FILE='log.txt',FORM='FORMATTED',STATUS='OLD',ACTION='WRITE',POSITION='APPEND')
    WRITE(99,*) "CFIJTALLY subroutine: Cannot read the special tally data when k>=10000. Check if the parameter Kcutp=" &
    ,Kcutp," is correctly defined. Also check if special tally data lines in the MVP output files include time, value in"// &
    " this order. IF NECESSARY, CHANGE FORMAT 1102. PLEASE BE CAREFUL WHEN CHANGING!"
    WRITE(*,*) "CFIJTALLY subroutine: Cannot read the special tally data when k>=10000. Check if the parameter Kcutp=" &
    ,Kcutp," is correctly defined. Also check if special tally data lines in the MVP output files include time, value in"// &
    " this order. IF NECESSARY, CHANGE FORMAT 1102. PLEASE BE CAREFUL WHEN CHANGING!"
    CLOSE(99)
    RETURN
   END IF
  END IF
!
! --- Continue if no error ---
!
  fijp(k,freg,sreg) = fdat
!
  READ(10,1104,IOSTAT=stat) time2, err
!
! *** Print out read error message if any and return ***
!
  IF ( stat > 0 ) THEN
   estat = 1
```

```
   OPEN(99,FILE='log.txt',FORM='FORMATTED',STATUS='OLD',ACTION='WRITE',POSITION='APPEND')
   WRITE(99,*) "CFIJTALLY subroutine: Cannot read the special tally data. Check if 1103 FORMAT at the end of"// &
   " this subroutine is correctly defined according to the MVP output files. BE CAREFUL with the FORMAT!!!"
   WRITE(*,*) "CFIJTALLY subroutine: Cannot read the special tally data. Check if 1103 FORMAT at the end of"// &
   " this subroutine is correctly defined according to the MVP output files. BE CAREFUL with the FORMAT!!!"
   CLOSE(99)
   RETURN
  END IF
!
! --- Continue if no error ---
!
  timeb(k) = time2
  sigfijpp(k,freg,sreg) = err
  sigfijp(k,freg,sreg) = sigfijpp(k,freg,sreg)/100.0 * fijp(k,freg,sreg)
!
 END DO
 REWIND(10)
END DO idloop
CLOSE(10)
!
!  === Calculating cumulative fission for prompt neutron ===
!
DO freg=reg_beg, reg_end
 DO sreg=1, Nreg
  DO k=1, Kcutp
   cfijp(k,freg,sreg) = cfijp(k-1,freg,sreg) + fijp(k,freg,sreg)
   sigcfijp(k,freg,sreg) = SQRT((sigcfijp(k-1,freg,sreg))**2 + (sigfijp(k,freg,sreg))**2)
  END DO
 END DO
END DO
!
!
1100 FORMAT(12X,A46)
1101 FORMAT(33X,ES12.5)
1102 FORMAT(35X,A8)
1103 FORMAT(7X,ES11.4,2X,ES12.5)
1104 FORMAT(7X,ES11.4,3X,F7.3)
END SUBROUTINE
!
!*****************************************************************************************************************
!
SUBROUTINE REG_DIV(nreg, n_procs, id, reg_beg, reg_end, n_regtask, estat)
!=================================================================================================================
!
! Subroutine: REG_DIV ( Subroutine to devide [nreg] into [n_procs] subregions, and allocate corresponding
!                region range [reg_beg,reg_end] to the corresponding process/task ID )
! Called from: FORWARD_MPI
!
! This subroutine is for n_procs>=1 and nreg>=1. Also, nreg>=n_procs must be held!
!
! --- Modification/change log ---
! - Written in original form by Delgersaikhan Tuya (Aug, 2018)
!
!=================================================================================================================
!
implicit none
!
INTEGER(KIND=4), INTENT(IN) :: nreg, n_procs, id  ! Number of regions, number of processes, and id of the process, respectively
INTEGER(KIND=4), INTENT(OUT) :: reg_beg, reg_end  ! Beginning and ending regions, respectively [reg_beg, reg_end]
INTEGER, INTENT(OUT) :: n_regtask         ! number of regions per process/task
INTEGER, INTENT(OUT) :: estat     ! Error status (0 means no error)
!
INTEGER :: n_regtask1        ! number of regions per process after re-balance (for first "remndr" processes )
INTEGER :: remndr            ! remainder
!
estat = 0
```

```
! --- Check if nreg>=n_procs, and if not, then raise error and stop ---
 IF( nreg < n_procs )THEN
  estat = 1
  OPEN(99,FILE='log.txt',FORM='FORMATTED',STATUS='OLD',ACTION='WRITE',POSITION='APPEND')
  WRITE(99,'(1X,A72)') "REG_DIV: Number of regions Nreg is smaller than number of processes!"
  WRITE(*,'(1X,A72)') "REG_DIV: Number of regions Nreg is smaller than number of processes!"
  CLOSE(99)
  RETURN
 END IF
!
 remndr = MOD(nreg,n_procs) ! Remainder
 n_regtask = (nreg-remndr)/n_procs
!
 IF( remndr > 0 )THEN
  n_regtask1 = n_regtask +1
 ELSE
  n_regtask1 = n_regtask
 END IF
!
! --- Allocate regions to corresponding process/task ---
 IF( id == 0 )THEN
  reg_beg = 1
  reg_end = reg_beg + (n_regtask1 - 1)
 ELSEIF( id < remndr )THEN
  reg_beg = id*n_regtask1 + 1
  reg_end = reg_beg + (n_regtask1 - 1)
 ELSE
  reg_beg = remndr*n_regtask1 + (id-remndr)*n_regtask + 1
  reg_end = reg_beg + (n_regtask - 1)
 END IF
 END SUBROUTINE
!
!*****************************************************************************************************************
!
 END MODULE MOD_COMP_SUBS
```

8. *mod_util_subs.f90* module

```
MODULE MOD_UTIL_SUBS
!
!=================================================================================================================
!
! Module: MOD_UTIL_SUBS ( module containing utility subroutines )
! Used by: FORWARD or FORWARD_MPI program
!
! Definition of each subroutine is included at the beginning of the subroutine.
!
! -----------------------------------------------------------------------------------------------------------------
! << MIK - Multi-region Integral Kinetic code, Version 1.0 >> by Delgersaikhan Tuya, Hiroki Takezawa, and
!    Toru Obara.
!    Laboratory for Advanced Nuclear Energy, Institute of Innovative Research, Tokyo Institute of Technology.
!
!    Date of release of code:  December 1, 2018
!
! Written in original form by: Delgersaikhan TUYA (Sep, 2018)
! -----------------------------------------------------------------------------------------------------------------
!=================================================================================================================
!
 IMPLICIT NONE
!
 CONTAINS
!
!*****************************************************************************************************************
```

```
!
 SUBROUTINE FR_READ(k_beg, k_end, skip, unit_n, in_arr1, in_arr2, estat)
!=============================================================================================================================
!
! Subroutine: FR_READ ( Subroutine to read [fr] and [sigfr] arrays from a "fr_dat.dat" file )
! Called from: FORWARD or FORWARD_MPI
!
! --- Modification/change log ---
! - Written in original form by Delgersaikhan Tuya (July 2018)
!
!=============================================================================================================================
!
 USE mod_values, ONLY: Nreg, Dt, dummychar
!
 INTEGER(KIND=4), INTENT(IN) :: k_beg, k_end    ! Beginning and ending indices for [in_arr1] & [in_arr2]
 INTEGER(KIND=4), INTENT(IN) :: skip            ! # of lines to be skipped from the beginning (excluding header)
 INTEGER(KIND=4), INTENT(IN) :: unit_n          ! Unit number for opening file
 REAL(KIND=KIND(1.D0)), INTENT(OUT), DIMENSION(k_beg:k_end,Nreg) :: in_arr1, in_arr2       ! Arrays for input to be read
 INTEGER, INTENT(OUT) :: estat   ! error status (0 means no error)
!
 INTEGER(KIND=4) :: i, j, k, freg, step
 INTEGER :: rstat
 REAL(KIND=KIND(1.D0)) :: time, fr, sigfr
 LOGICAL :: lstat
!
! -- Check if the file exists --
 estat = 0
 INQUIRE(FILE="fr_dat.dat", EXIST=lstat)
 IF( .NOT. lstat )THEN
  estat = 1
  OPEN(99,FILE='log.txt',FORM='FORMATTED',STATUS='OLD',ACTION='WRITE',POSITION='APPEND')
  WRITE(99,*) "FR_READ subroutine: Requested file not found. Error, stop!"
  WRITE(*,*) "FR_READ subroutine: Requested file not found. Error, stop!"
  CLOSE(99)
  RETURN
 END IF
!
! -- Open a fname file & read header--
 OPEN(UNIT=unit_n,FILE="fr_dat.dat",FORM='FORMATTED', STATUS='OLD',ACTION='READ',POSITION='REWIND')
 READ(unit_n,'(A10)') dummychar
!
! -- Start reading --
!
 IF( skip == 0 )THEN
  DO k=k_beg, k_end
   DO i=1, Nreg
    READ(unit_n,*,IOSTAT=rstat) freg, time, fr, sigfr
    step = NINT(time/Dt)
    IF( step /= k )THEN
     estat = 1
     OPEN(99,FILE='log.txt',FORM='FORMATTED',STATUS='OLD',ACTION='WRITE',POSITION='APPEND')
     WRITE(99,*) "FR_READ subroutine: Error in reading from fr_dat.dat. Time steps don't match!, Stop"
     WRITE(*,*) "FR_READ subroutine: Error in reading from fr_dat.dat. Time steps don't match!, Stop"
     CLOSE(99)
     RETURN
    END IF
    in_arr1(step,freg) = fr
    in_arr2(step,freg) = sigfr
   END DO
  END DO
!
 ELSEIF( skip > 0 )THEN
  DO k=1, skip
   READ(unit_n,'(A10)') dummychar
  END DO
  DO k=k_beg, k_end
```

```
  DO i=1, Nreg
   READ(unit_n,*,IOSTAT=rstat) freg, time, fr, sigfr
   step = NINT(time/Dt)
   IF( step /= k )THEN
    estat = 1
    OPEN(99,FILE='log.txt',FORM='FORMATTED',STATUS='OLD',ACTION='WRITE',POSITION='APPEND')
    WRITE(99,*) "FR_READ subroutine: Error in reading from fr_dat.dat. Time steps don't match!, Stop"
    WRITE(*,*) "FR_READ subroutine: Error in reading from fr_dat.dat. Time steps don't match!, Stop"
    CLOSE(99)
    RETURN
   END IF
   in_arr1(step,freg) = fr
   in_arr2(step,freg) = sigfr
  END DO
 END DO
END IF
!
CLOSE(unit_n)
END SUBROUTINE
!
!*****************************************************************************************************************
!
SUBROUTINE FR_WRITE(k_beg, k_end, out_arr1, out_arr2, unit_n, ostat, pos, estat)
!=================================================================================================================
!
! Subroutine: FR_WRITE ( Subroutine to write [fr] and [sigfr] arrays to "fr_dat.dat" file )
! Called from: FORWARD or FORWARD_MPI
!
! --- Modification/change log ---
! - Written in original form by Delgersaikhan Tuya (July 2018)
!
!=================================================================================================================
!
USE mod_values, ONLY: Nreg, Dt
!
INTEGER(KIND=4), INTENT(IN) :: k_beg, k_end      ! Beginning and ending indices, respectively
REAL(KIND=KIND(1.D0)), INTENT(IN), DIMENSION(k_beg:k_end,Nreg) :: out_arr1, out_arr2      ! Output arrays to be written
INTEGER, INTENT(IN) :: unit_n                  ! Unit number for opening fname file
CHARACTER(len=3), INTENT(IN) :: ostat      ! Argument for STATUS in OPEN statement.
CHARACTER(len=6), INTENT(IN) :: pos        ! Argument for POSITION in OPEN statement.
INTEGER, INTENT(OUT) :: estat                                  ! Error status (0 means no error)
!
INTEGER(KIND=4) :: i, j, k
!
estat = 0
!
! -- Open a file & write header if necessary--
IF( ostat == "NEW" .AND. pos == "REWIND" )THEN
 OPEN(UNIT=unit_n, FILE="fr_dat.dat", FORM='FORMATTED', STATUS='NEW',ACTION='WRITE', POSITION='REWIND')
 WRITE(unit_n,100) "REG", "TIME, s", "Fis.rate, f/s", "Sigma"
 100 FORMAT(1X,A3,3X,A7,3X,A13,A10)
ELSEIF( ostat == "OLD" .AND. pos == "APPEND" )THEN
 OPEN(UNIT=unit_n, FILE="fr_dat.dat", FORM='FORMATTED', STATUS='OLD',ACTION='WRITE', POSITION='APPEND')
ELSE
 estat = 1
 OPEN(99,FILE='log.txt',FORM='FORMATTED',STATUS='OLD',ACTION='WRITE',POSITION='APPEND')
 WRITE(99,*) "FR_WRITE subroutine: Looks like wrong combination of ostat and pos! STOP"
 WRITE(*,*) "FR_WRITE subroutine: Looks like wrong combination of ostat and pos! STOP"
 CLOSE(99)
 RETURN
END IF
!
! -- Start writing --
!
DO k=k_beg, k_end
 DO i=1, Nreg
```

```
  WRITE(unit_n,110) i, k*Dt, out_arr1(k,i), out_arr2(k,i)
  110 FORMAT(I5, ES16.8, ES16.8, ES16.8)
 END DO
 END DO
!
 CLOSE(unit_n)
 END SUBROUTINE
!
!*****************************************************************************************************************
!
 SUBROUTINE CIJP_READ(unit_n, step, cijp, sigcijp, estat)
!=================================================================================================================
!
! Subroutine: CIJP_READ ( Subroutine to read [cijp] and [sigcijp] arrays written at [step] )
! Called from: FORWARD or FORWARD_MPI
!
! This subroutine reads a [cijp] and [sigcijp] arrays from a given file written at [step].
!
! --- Modification/change log ---
! - Written in original form by Delgersaikhan Tuya (July 2018)
!
!=================================================================================================================
!
 USE mod_values, ONLY: Nreg, Kcutp, Dt, Nlen, fnum, Cij_fmt, dummychar
!
 INTEGER, INTENT(IN) :: unit_n              ! Unit number for a C_ij file
 INTEGER(KIND=4), INTENT(IN) :: step        ! Step at which C_ij function is to be read
 REAL(KIND=KIND(1.D0)),INTENT(OUT),DIMENSION(0:Kcutp, Nreg, Nreg) :: cijp, sigcijp
 INTEGER, INTENT(OUT) :: estat              ! Error status to return (0 means no error)
!
 INTEGER(KIND=4) :: i, j, k, freg, sreg, n
 REAL(KIND=KIND(1.D0)) :: time_read, val_read, err_read
 INTEGER :: rstat
 LOGICAL :: lstat
!
! -- Check if the requested data file exist --
 estat = 0
!
 WRITE(fnum,Cij_fmt) step
 INQUIRE(FILE="cij_"//fnum//".dat", EXIST=lstat)
 IF( .NOT. lstat )THEN
  estat = 1
  OPEN(99,FILE='log.txt',FORM='FORMATTED',STATUS='OLD',ACTION='WRITE',POSITION='APPEND')
  WRITE(99,*) "CIJP_READ subroutine: Requested file written at step ", step, " not found. Error, stop!"
  WRITE(*,*) "CIJP_READ subroutine: Requested file written at step ", step, " not found. Error, stop!"
  CLOSE(99)
  RETURN
 END IF
!
! -- Open a fname file & skip header --

 OPEN(unit_n, FILE="cij_"//fnum//".dat", FORM='FORMATTED', STATUS='OLD', ACTION='READ', POSITION='REWIND')
 READ(unit_n,'(A10)') dummychar
!
! -- Start reading --
!
 n = 0
!
 CIJloop: DO
!!   READ(unit_n,1030, IOSTAT=stat) freg, sreg, time_read, val_read, err_read
  READ(unit_n,*, IOSTAT=rstat) freg, sreg, time_read, val_read, err_read
  IF ( rstat < 0 )THEN
! -- en of file --
   EXIT cijloop
  ELSEIF( rstat > 0 )THEN
   estat = 1
```

```
  OPEN(99,FILE='log.txt',FORM='FORMATTED',STATUS='OLD',ACTION='WRITE',POSITION='APPEND')
  WRITE(99,'(1X,A62,I10,A15)') "CIJP_READ subroutine: Reading error to read data written at ", step, ". Error, stop!"
  WRITE(*,'(1X,A62,I10,A15)') "CIJP_READ subroutine: Reading error to read data written at ", step, ". Error, stop!"
  CLOSE(99)
  RETURN
 END IF
 k = NINT(time_read/Dt)
 cijp(k,freg,sreg) = val_read
 sigcijp(k,freg,sreg) = err_read
 n = n + 1
END DO CIJloop
!
! -- Check if all necessary data have been read --
IF( n == Nreg*Nreg*(Kcutp+1) )THEN
 estat = 0
ELSE
 estat = 1
 OPEN(99,FILE='log.txt',FORM='FORMATTED',STATUS='OLD',ACTION='WRITE',POSITION='APPEND')
 WRITE(99,'(1X,A62,I10,A15)') "CIJP_READ subroutine: Incomplete data have been read at step ", step, ". Error, stop!"
 WRITE(*,'(1X,A62,I10,A15)') "CIJP_READ subroutine: Incomplete data have been read at step ", step, ". Error, stop!"
 CLOSE(99)
 RETURN
END IF
CLOSE(unit_n)
END SUBROUTINE
!
!*****************************************************************************************************************
!
 SUBROUTINE CIJP_WRITE(keff, step, out_arr1, out_arr2, unit_n)
!=================================================================================================================
!
! Subroutine: CIJP_WRITE ( Subroutine to write [cijp] and [sigcijp] arrays to a given output file )
! Called from: FORWARD or FORWARD_MPI
!
! This subroutine writes a rank-2 arrays [out_arr1] and [out_arr2] to a given output file.
! [out_arr1] is an integer array, while [out_arr2] is a double-precision real number array.
!
! --- Modification/change log ---
! - Written in original form by Delgersaikhan Tuya (July 2018)
!
!=================================================================================================================
!
 USE mod_values, ONLY: Nreg, Kcutp, Dt, K_adjust, K_init, fnum, Cij_fmt
!
 REAL(KIND=KIND(1.D0)),INTENT(IN) :: keff    ! k-eff
 INTEGER(KIND=4), INTENT(IN) :: step        ! step at which C_ij function is outputted
 REAL(KIND=KIND(1.D0)),INTENT(IN),DIMENSION(0:kcutp, nreg, nreg) :: out_arr1, out_arr2     ! Rank-3 double precision real arrays for C_ij and
sigC_ij
 INTEGER, INTENT(IN) :: unit_n       ! Unit number
!
 INTEGER(KIND=4) :: i, j, k
!
! -- Open a fname file & write header if necessary--
 WRITE(fnum,Cij_fmt) step
 OPEN(unit_n, FILE="cij_"//fnum//".dat", FORM='FORMATTED', STATUS='NEW',ACTION='WRITE', POSITION='REWIND')
 WRITE(unit_n,100) "FREG", "SREG", "Time, s", "C_ij", "Sigma"
 100 FORMAT(3X,A4,2X,A4,3X,A7,3X,A4,5X,A5)
!
! -- Start writing --
!
 IF( step == 0)THEN
  DO i=1, Nreg
   DO j=1, Nreg
    DO k=0, Kcutp
     WRITE(unit_n,110) i, j, k*Dt, out_arr1(k,i,j)/(keff*K_adjust/K_init), out_arr2(k,i,j)/(keff*K_adjust/K_init)
    END DO
```

```
  END DO
 END DO
ELSEIF( step > 0 )THEN
 DO i=1, Nreg
  DO j=1, Nreg
   DO k=0, Kcutp
    WRITE(unit_n,110) i, j, k*Dt, out_arr1(k,i,j)*K_adjust/K_init, out_arr2(k,i,j)*K_adjust/K_init
   END DO
  END DO
 END DO
END IF
110 FORMAT(2I5, ES16.8, ES16.8, ES16.8)
!
 CLOSE(unit_n)
 END SUBROUTINE
!
!*****************************************************************************************************************
!
 SUBROUTINE UVAR_READ(k_beg, k_end, skip, u_name, unit_n, in_arr1, estat)
!=================================================================================================================
!
! Subroutine: UVAR_READ ( Subroutine to read a part of [uvara] array from a given file )
! Called from: FORWARD or FORWARD_MPI
!
! --- Modification/change log ---
! - Written in original form by Delgersaikhan Tuya (July 2018)
!
!=================================================================================================================
!
 USE mod_values, ONLY: Nreg, Dt, Uvarnl, dummychar
!
 INTEGER(KIND=4), INTENT(IN) :: k_beg, k_end      ! Beginning and ending indices, respectively
 INTEGER(KIND=4), INTENT(IN) :: skip              ! # of lines to be skipped from the beginning (excluding header)
 CHARACTER(len=Uvarnl), INTENT(IN) :: u_name      ! Input update variable name
 INTEGER(KIND=4), INTENT(IN) :: unit_n            ! Unit number for opening file
 REAL(KIND=KIND(1.D0)), INTENT(OUT), DIMENSION(k_beg:k_end,Nreg) :: in_arr1      ! Array for the input to be read
 INTEGER, INTENT(OUT) :: estat    ! Error status to return (0 means no error)
!
 INTEGER(KIND=4) :: i, j, k, freg, step
 INTEGER :: rstat
 REAL(KIND=KIND(1.D0)) :: time, var
 LOGICAL :: lstat
!
! -- Check if file exists --
 estat = 0
!
 INQUIRE(FILE=u_name//"_dat.dat", EXIST=lstat)
 IF( .NOT. lstat )THEN
  estat = 1
  OPEN(99,FILE='log.txt',FORM='FORMATTED',STATUS='OLD',ACTION='WRITE',POSITION='APPEND')
  WRITE(99,*) "UVAR_READ subroutine: Requested file for ", u_name, " data not found. Error, stop!"
  WRITE(*,*) "UVAR_READ subroutine: Requested file for ", u_name, " data not found. Error, stop!"
  CLOSE(99)
  RETURN
 END IF
!
! -- Open a file & skip header --
 OPEN(UNIT=unit_n, FILE=u_name//"_dat.dat", FORM='FORMATTED', STATUS='OLD',ACTION='READ', POSITION='REWIND')
 READ(unit_n,'(A10)') dummychar
!
! -- Start reading --
!
 IF( skip == 0 )THEN
  DO k=k_beg, k_end
   DO i=1, Nreg
    READ(unit_n,*,IOSTAT=rstat) freg, time, var
```

```
   step = NINT(time/Dt)
   IF( step /= k )THEN
    estat = 1
    OPEN(99,FILE='log.txt',FORM='FORMATTED',STATUS='OLD',ACTION='WRITE',POSITION='APPEND')
    WRITE(99,*) "UVAR_READ subroutine: Error in reading ", u_name, " data. Time steps don't match!, Stop"
    WRITE(*,*) "UVAR_READ subroutine: Error in reading ", u_name, " data. Time steps don't match!, Stop"
    CLOSE(99)
    RETURN
   END IF
   in_arr1(step,freg) = var
  END DO
 END DO
!
 ELSEIF( skip > 0 )THEN
  DO k=1, skip
   READ(unit_n,'(A10)') dummychar
  END DO
  DO k=k_beg, k_end
   DO i=1, Nreg
    READ(unit_n,*,IOSTAT=rstat) freg, time, var
!    110 FORMAT(I5, ES16.8, ES16.8)
   step = NINT(time/Dt)
   IF( step /= k )THEN
    estat = 1
    OPEN(99,FILE='log.txt',FORM='FORMATTED',STATUS='OLD',ACTION='WRITE',POSITION='APPEND')
    WRITE(99,*) "UVAR_READ subroutine: Error in reading ", u_name, " data. Time steps don't match!, Stop"
    WRITE(*,*) "UVAR_READ subroutine: Error in reading ", u_name, " data. Time steps don't match!, Stop"
    CLOSE(99)
    RETURN
   END IF
   in_arr1(step,freg) = var
  END DO
 END DO
END IF
!
 CLOSE(unit_n)
 END SUBROUTINE
!
!*****************************************************************************************************************
!
 SUBROUTINE UVAR_WRITE(k_beg, k_end, out_arr1, u_name, unit_n, ostat, pos, estat)
!=================================================================================================================
!
! Subroutine: UVAR_WRITE ( Subroutine to write a part of [uvara] array to a given file )
! Called from: FORWARD or FORWARD_MPI
!
! --- Modification/change log ---
! - Written in original form by Delgersaikhan Tuya (July 2018)
!
!=================================================================================================================
!
 USE mod_values, ONLY: Nreg, Dt, Uvarnl
!
 INTEGER(KIND=4), INTENT(IN) :: k_beg, k_end    ! Beginning and ending indices, respectively
 REAL(KIND=KIND(1.D0)), INTENT(IN), DIMENSION(k_beg:k_end,Nreg) :: out_arr1      ! Output array to be written
 CHARACTER(len=Uvarnl), INTENT(IN) :: u_name      ! Output update variable name
 INTEGER(KIND=4), INTENT(IN) :: unit_n           ! Unit number for opening fname file
 CHARACTER(len=3), INTENT(IN) :: ostat       ! Argument for STATUS clause in OPEN statement.
 CHARACTER(len=6), INTENT(IN) :: pos         ! Argument for POSITION clause in OPEN statement.
 INTEGER, INTENT(OUT) :: estat               ! Error status (0 means no error)
!
 INTEGER(KIND=4) :: i, j, k
!
 estat = 0
!
! -- Open a fname file & write header if necessary--
```

```
 IF( ostat == "NEW" .AND. pos == "REWIND" )THEN
  OPEN(UNIT=unit_n, FILE=u_name//"_dat.dat", FORM='FORMATTED', STATUS='NEW',ACTION='WRITE', POSITION='REWIND')
  WRITE(unit_n,100) "REG","TIME, s", u_name
  100 FORMAT(1X,A3,3X,A7,3X,A20)
 ELSEIF( ostat == "OLD" .AND. pos == "APPEND" )THEN
  OPEN(UNIT=unit_n, FILE=u_name//"_dat.dat", FORM='FORMATTED', STATUS='OLD',ACTION='WRITE', POSITION='APPEND')
 ELSE
  estat = 1
  OPEN(99,FILE='log.txt',FORM='FORMATTED',STATUS='OLD',ACTION='WRITE',POSITION='APPEND')
  WRITE(99,*) "UVAR_WRITE subroutine: Looks like wrong combination of ostat and pos! STOP"
  WRITE(*,*) "UVAR_WRITE subroutine: Looks like wrong combination of ostat and pos! STOP"
  CLOSE(99)
  RETURN
 END IF
!
! -- Start writing --
!
 DO k=k_beg, k_end
  DO i=1, Nreg
   WRITE(unit_n,110) i, k*Dt, out_arr1(k,i)
   110 FORMAT(I5, ES16.8, ES16.8)
  END DO
 END DO
!
 CLOSE(unit_n)
 END SUBROUTINE
!
!*************************************************************************************************************
!
 SUBROUTINE REAC_INSERT(step, unit_n, n_char, fname ,out_arr, estat, rins)
!=============================================================================================================
!
! Subroutine: REAC_INSERT ( Subroutine to read and insert reactivity data at [step] )
! Called from: FORWARD or FORWARD_MPI
!
! --- Modification/change log ---
! - Written in original form by Delgersaikhan Tuya (July 2018)
!
!=============================================================================================================
!
 USE mod_values, ONLY: Nrvar, Dt, dummychar
!
 INTEGER(KIND=4), INTENT(IN) :: step    ! Step at which to check and insert reactivity
 INTEGER(KIND=4), INTENT(IN) :: unit_n  ! Unit number for opening [fname] file
 INTEGER(KIND=4), INTENT(IN) :: n_char  ! Number of characters in [fname]
 CHARACTER(len=n_char), INTENT(IN) :: fname  ! File name containing reactivity data
 REAL(KIND=KIND(1.D0)), INTENT(OUT), DIMENSION(Nrvar) :: out_arr   ! Out array to containing all reactivity data
 INTEGER, INTENT(OUT) :: estat    ! Error status (0 means no error)
 CHARACTER(len=1), INTENT(OUT) :: rins  ! Whether or not to insert reactivity. rins = "Y" means to insert reactivity,
                                ! while rins = "N" means not to insert reactivity at [step]
!
 INTEGER(KIND=4) :: i, j, k, n
 INTEGER :: rstat
 REAL(KIND=KIND(1.D0)) :: time_read, val_read
 LOGICAL :: lstat
!
! -- Check if [fname] file exists --
 estat = 0
 INQUIRE(FILE=fname, EXIST=lstat)
 IF( .NOT. lstat )THEN
  estat = 1
  OPEN(99,FILE='log.txt',FORM='FORMATTED',STATUS='OLD',ACTION='WRITE',POSITION='APPEND')
  WRITE(99,'(1X,A20,A20,A25)') "REAC_INSERT subroutine: File ", fname, " not found. Error, stop!"
  WRITE(*,'(1X,A20,A20,A25)') "REAC_INSERT subroutine: File ", fname, " not found. Error, stop!"
  CLOSE(99)
  RETURN
```

```
 END IF
!
! -- Open a fname file & write header if necessary--
 OPEN(UNIT=unit_n, FILE=fname, FORM='FORMATTED', STATUS='OLD', ACTION='READ')
 REWIND(unit_n)
! -- Skip header --
 READ(unit_n,'(A10)') dummychar
 n = 0
!
! -- Set rins = "Y" if step == 0 because reactvity must be inserted at step 0 --
 IF( step == 0 )THEN
  rins = "Y"
 END IF
!
 reacloop : DO
  READ(unit_n, *, IOSTAT=rstat) time_read, val_read
  IF( rstat < 0 )THEN
! -- end of file
   EXIT reacloop
  ELSEIF( rstat > 0 )THEN
   estat = 1
   OPEN(99,FILE='log.txt',FORM='FORMATTED',STATUS='OLD',ACTION='WRITE',POSITION='APPEND')
   WRITE(99,'(1X,A57,A20,A20)') "REAC_INSERT subroutine: Reading error to read data from ", fname, " file. Error, stop!"
   WRITE(*,'(1X,A57,A20,A20)') "REAC_INSERT subroutine: Reading error to read data from ", fname, " file. Error, stop!"
   CLOSE(99)
   RETURN
  END IF
!
  IF( step == NINT(time_read/Dt) )THEN
   n = n + 1
   out_arr(n) = val_read
  END IF
 END DO reacloop
!
! -- Check if all [Nrvar] reactivity data has been read --
 IF( n == Nrvar )THEN
  estat = 0
  rins = "Y"
 ELSEIF( n == 0 )THEN
! -- This means reactivity is not supposed to be inserted at this [step], so no error --
  estat = 0
  rins = "N"
 ELSEIF( n > 0 .AND. n < Nrvar)THEN
  estat = 1
  OPEN(99,FILE='log.txt',FORM='FORMATTED',STATUS='OLD',ACTION='WRITE',POSITION='APPEND')
  WRITE(99,'(1X,A65,I10,A20)') "REAC_INSERT subroutine: Incomplete reactivity insertion data at ", step, " step. Error, stop!"
  WRITE(99,'(1X,A65,I10,A20)') "REAC_INSERT subroutine: Incomplete reactivity insertion data at ", step, " step. Error, stop!"
  CLOSE(99)
  RETURN
 ELSEIF( n > 0 .AND. n > Nrvar)THEN
  estat = 1
  OPEN(99,FILE='log.txt',FORM='FORMATTED',STATUS='OLD',ACTION='WRITE',POSITION='APPEND')
  WRITE(99,'(1X,A65,I10,A20)') "REAC_INSERT subroutine: Additional reactivity insertion data at ", step, " step. Error, stop!"
  WRITE(99,'(1X,A65,I10,A20)') "REAC_INSERT subroutine: Additional reactivity insertion data at ", step, " step. Error, stop!"
  CLOSE(99)
  RETURN
 END IF
!
 CLOSE(unit_n)
 END SUBROUTINE
!
!*****************************************************************************************************************
!
 END MODULE MOD_UTIL_SUBS
```

9. *mikrun.sh* shell script

```
#!/bin/csh
#================================================================================
# << Shell script to compile and run MIK code >>
#
# - The MAIN program is compiled with GNU Fortran GCC (gfortran 5.1 or similar/higher).
#   Look at the source code for the reason.
# - The other programs are compiled with either Intel Fortran IFORT (12.1 or higher)
#   or GNU Fortran GCC (5.1 or similar/higher) depending on the user choice.
# - In case of parallel mode run, other programs are compiled with Open MPI 3.0
#   (or higher) installed with either Intel Fortran IFORT (12.1 or higher) or
#   GNU Fortran GCC (gfortran 5.1 or similar/higher).
#
# Written in original form by: Delgersaikhan TUYA (Feb 2018)
# Updated on: Sep 2018 by Delgersaikhan TUYA
#================================================================================
 set MIK_DIR = $PWD
#
 set prompt_win = " "
 while( $prompt_win != "q" )
  echo " "
  echo "-----------------------------------------------------------------"
  echo " "
  echo " Options for calculation mode:"
  echo " 1 : Single mode calculation ( using GNU Fortran (gfortran 5.1 or similar/higher) )."
  echo " 2 : Single mode calculation ( using Intel Fortran (ifort 12.1 or similar/higher) )."
  echo " 3 : Parallel mode calculation using OpenMPI 3.0 or higher."
  echo "     ( installed with GNU Fortran (gfortran 5.1 or similar/higher) )"
  echo " 4 : Parallel mode calculation using OpenMPI 3.0 or higher."
  echo "     ( installed with Intel Fortran (ifort 12.1 or similar/higher) )"
  echo " q : Quit."
  echo " "
  echo "-----------------------------------------------------------------"
  echo -n "Enter an appropriate number : "
  set compiler_choice = $<
#
# Set a compiler and its flags
  if ( $compiler_choice == 1 ) then
                    set mik_compiler = "gfortran"
                    set main_compiler = $mik_compiler
                    set mik_compiler_flags = "-O2 -cpp"
                    set main_compiler_flags = "-O2 -cpp"
                    set mik_forward = "forward"
  else if ( $compiler_choice == 2 ) then
                    set mik_compiler = "ifort"
                    set main_compiler = "gfortran"
                    set mik_compiler_flags = "-O2 -fpp"
                    set main_compiler_flags = "-O2 -cpp"
                    set mik_forward = "forward"
  else if ( $compiler_choice == 3 ) then
                    set mik_compiler = "mpifort"
                    set main_compiler = "gfortran"
                    set mik_compiler_flags = "-O2 -cpp -D MIK_MPI"
                    set main_compiler_flags = "-O2 -cpp -D MIK_MPI"
                    set mik_forward = "forward_mpi"
  else if ( $compiler_choice == 4 ) then
                    set mik_compiler = "mpifort"
                    set main_compiler = "gfortran"
                    set mik_compiler_flags = "-O2 -fpp -D MIK_MPI"
                    set main_compiler_flags = "-O2 -cpp -D MIK_MPI"
                    set mik_forward = "forward_mpi"
  else if ( $compiler_choice == "q" ) then
                    echo " Quit option is chosen. Quit!"
                    exit 0
  else
```

```
                echo "Invalid input! Exit!"
                exit 1
 endif
#
# Copy the source files temporarily
 cd src
 cp main.f90 $mik_forward.f90 mod_values.f90 update.f90 mod_comp_subs.f90 mod_util_subs.f90 mod_feedback.f90 sgetrf.f sgetrf2.f \
   sgetrs.f slaswp.f strsm.f ilaenv.f lsame.f ieeeck.f iparmq.f isamax.f sgemm.f slamch.f sscal.f xerbla.f $MIK_DIR
 cd $MIK_DIR
#
# Start compiling to produce object files ...
 $mik_compiler -c mod_values.f90 $mik_compiler_flags
 $mik_compiler -c mod_feedback.f90 $mik_compiler_flags
 $mik_compiler -c mod_comp_subs.f90 $mik_compiler_flags
 $mik_compiler -c mod_util_subs.f90 $mik_compiler_flags
 $mik_compiler -c $mik_forward.f90 sgetrf.f sgetrf2.f sgetrs.f slaswp.f strsm.f \
     ilaenv.f lsame.f ieeeck.f iparmq.f isamax.f sgemm.f slamch.f sscal.f xerbla.f $mik_compiler_flags
 $mik_compiler -c update.f90 $mik_compiler_flags
 $main_compiler -c main.f90 $main_compiler_flags
#
# Start linking to produce executables
 $mik_compiler -o $mik_forward $mik_forward.o mod_comp_subs.o mod_util_subs.o mod_feedback.o sgetrf.o sgetrf2.o sgetrs.o slaswp.o
strsm.o \
     ilaenv.o lsame.o ieeeck.o iparmq.o isamax.o sgemm.o slamch.o sscal.o xerbla.o mod_values.o
 $mik_compiler -o update update.o mod_values.o
 $main_compiler -o main main.o
#
# Remove the copied src files from MIK_DIR
 rm -f  main.f90 $mik_forward.f90 mod_values.f90 update.f90 mod_comp_subs.f90 mod_util_subs.f90 mod_feedback.f90 sgetrf.f sgetrf2.f \
   sgetrs.f slaswp.f strsm.f ilaenv.f lsame.f ieeeck.f iparmq.f isamax.f sgemm.f slamch.f sscal.f xerbla.f
 rm -f  main.o $mik_forward.o mod_values.o update.o mod_comp_subs.o mod_util_subs.o mod_feedback.o sgetrf.o sgetrf2.o \
   sgetrs.o slaswp.o strsm.o ilaenv.o lsame.o ieeeck.o iparmq.o isamax.o sgemm.o slamch.o sscal.o xerbla.o
#
# Set the prompt_win to "q" and quit the while loop.
 set prompt_win = "q"
 end
#
# Print if compilation is successful
 echo "--------------------------------------------------------------------"
 echo "Calculation has started & is running... Please wait till it ends!"
 echo "--------------------------------------------------------------------"
# Run the MIK calculation if compilation is successful
 ./main
```