DRAGON Code Sample and Tutorial

Here we present a sample of the DRAGON code used in the reactor simulations carried out for this thesis. The code shown below is applicable for DRAGON version 3.06H and can be obtained from the following site: `http://www.polymtl.ca/nucleaire/DRAGON/en/download/index.php`. Several of the modules listed below have been deprecated for DRAGON version 4, the experimental version of the code [**?**]. DRAGON is freely available and can be obtained from [**?**]. It can run on any platform that supports a Fortran compiler. The code below was successfully run with gfortran [**?**]. This code is the first stage of the Takahama-3 assembly simulation. A DRAGON input file, henceforth referred to as an *input deck*, is a collection of CLE-2000 commands[**?**]. CLE-2000 is a system of Fortran-like procedures and commands that support the modular nature of DRAGON by providing it with a unified and uniform interface. Here, we see an example of variable declaration and assignment:

```
REAL Power := 5.00 ;
```

The := operator represents assignment, and all CLE-2000 statements must terminate with a semicolon. White space is insignificant in DRAGON; however, input in column 72 or greater will not be parsed correctly as it can represent a comment, and tab characters will always parse incorrectly and lead to an error. Variables in DRAGON can be of the typical variety found in most programming languages (integers, characters, double precision, etc.).

0.1 Linked List Directives

An input deck typically begins with a declaration of user-defined names to be introduced into the program. The LINKED_LIST directive begins a line that contains the user-defined variables. CLE-2000 can support output into a variety of formats, including ASCII and binary among others. In the following code segment, NewBurnFile, a user-defined variable, will contain plain text about the fuel evolution, and Track will be in a binary format which is more efficient for use by other DRAGON modules:

```
LINKED_LIST
   ASSMB DISCR LIBRARY CP CALC OUT BURNUP
   EDITION ;
SEQ_ASCII NewBurnFile ;
SEQ_BINARY Track ;
```

0.2 Importing Modules

The MODULE keyword informs DRAGON which elements of the code are required for the simulation. The modules here are CLE-2000 keywords; thus LIB: represents the activation of the cross section library interpolation module. The meaning of the following modules will be elaborated upon below. Failure to include the module name in this list will cause an error.

```
MODULE
   LIB: GEO: EXCELT: SHI: ASM: FLU: EVO: EDI:
   DELETE: END: ;
```

0.3 The LIB: module

The following segment imports the cross section libraries into the input deck.

```
LIBRARY := LIB: ::
  NMIX 6 CTRA WIMS
  DEPL LIB: WIMSD4 FIL: jendl3gx
  MIXS LIB: WIMSD4 FIL: jendl3gx
```

Now we enter the LIB: module. In the first line, we are assigning the linked list variable LIBRARY to the output of the LIB: module. The ...indicate that what follows will be a list of keyword-parameter sublists. For example, the NMIX keyword represents the number of material mixtures, such as fuel or moderator, to be treated in the simulation. In this case, there are 6 mixtures. The CTRA keyword refers to the transport correction to be applied to the cross sections, and in this case, a WIMS-style correction will be applied since the libraries are WIMS-D4 format[**?**]. DEPL here expects the cross section library format, which here is WIMS-D4. DRAGON can interpret several different library formats, including its own native DRAGLIB format[**?**]. The user is required to obtain or generate these libraries elsewhere. The name of the library is user-defined. In this simulation, the 172-group JENDL 3.2 library was used[**?**]. Note that none of the above code ends in a semicolon. This is because the entire LIB: module specification is a statement.

0.4 Mixtures

```
* Water / moderator
  MIX 1 600.0 0.7200768
     H1H2O    = '3001'   11.188
     O16H2O   = '6016'   88.749
     BNat     = '1011'   0.0630
```

Any line that begins with a * is a comment in CLE-2000, and thus DRAGON. The input deck will crash if the asterisk is not in the first column. The comment informs us that we are about to define the moderator for the Takahama fuel assembly. The MIX keyword labels the mixture with an arbitrary integer which must be less than 6,

since NMIX was defined to be 6. The next value is the mixture temperature; thus the temperature of the water will be 600 K. The final value is the density of the mixture in g/cc. We would expect the density of water at 600 K to be less than its room temperature value, and so the value for sample SF97-4 here is about 0.72 g/cc.

The next line shows the elements and submixtures that comprise the full mixture. The mixtures contained in the LIB: module specify the initial state of the reactor. The first term in this chain of expressions (e.g., H1H2O) is a user-defined synonym for an ENDF/6-formatted[?] integer that represents the properties of the element in question. The last value represents the weight fraction in percent of the element in the mixture. Thus, we can see that oxygen-16 has a label O16H2O, is represented in ENDF/6 by the string '6016', and has a weight fraction of 11.188% in water. It is worth noting here that code '1011' represents non-soluble natural boron, which means that its value is constant throughout the course of the fuel evolution. In this case, the value is set to 630 ppm.

The user can omit the density of the material, but if this is done, instead of inputting the weight fraction of the element, the user inputs the number density of the mixture. This gives added flexibility in cases where the reactor description is incomplete and the mixture densities are unknown.

```
* Fuel cladding
  MIX 2 600.0 5.821341
     CrNat     = '52' 0.0010033
     FeNat     = '2056' 0.0021067
     ZrNat     = '91' 99.689
```

In the above, we can see the mixture specification for the Zircaloy fuel cladding, and that it is almost all natural zirconium by weight.

If the sum of all of the weight percentage do not equal 100%, DRAGON will simply renormalize the inputs by the sum.

```
* UO2 fuel mixture
  MIX 3 900.0 10.0701
```

```
O16      = '6016'  11.852
U234     = '234'   0.03526 1
U235     = '2235'  3.622944 1
U238     = '8238'  84.49 1
Pu238    = '948'    0.0 1
Pu239     = '6239'   0.0 1
Pu240     = '1240'   0.0 1
Pu241     = '1241'   0.0 1
Pu242     = '242'   0.0 1
Pu242h     = '1242'   0.0 1
U232     = '232' 0.0 1
U232ps     = '4232' 0.0 1
U233      = '9233' 0.0 1
U236     = '236'   0.0 1
U237     = '927'   0.0 1
U237ps     = '4927'   0.0 1
Np237     = '937'  0.0 1
Np239     = '1939'   0.0 1
Am241     = '951'  0.0 1
Am242     = '1952'   0.0 1
Am242m     = '952'   0.0 1
Am243     = '953'   0.0 1
Cm242     = '962'   0.0 1
Cm243     = '963'   0.0 1
```

Above, we see the specification for the uranium dioxide fuel mixture at 900 K and with a density of 10.07 g/cc. Of course, during the evolution of a reactor or assembly, fuel is consumed and other actinides, such as americium and curium, are produced. DRAGON will evolve these elements in time, regardless of whether they are specified as above. We explicitly list them above for 2 reasons. Firstly, notice that for all the elements save for oxygen, the last value is a "1". This is a *self-shielding index*,

and it informs DRAGON that the elements sharing the same index must undergo the self-shielding process together. Since the plutonium isotopes and other higher actinides are potentially useful for important cross-checks of proper evolution, their self-shielding must be labeled in the initial state. Oxygen, not being fissile, does not undergo self-shielding and thus does not require an index (indeed, adding one will cause an error). The second reason is that it facilitates parsing of the output by the author's processing programs.

```
* UO2-Gd2O3 fuel mixture
  MIX 4 900.0 10.0701
    U234G = '234'   0.016572 2
    U235G = '2235'  2.1793 2
    U238G = '8238'  80.665 2
    Pu238G    = '948'    0.0 2
    Pu239G    = '6239'   0.0 2
    Pu240G    = '1240'   0.0 2
    Pu241G    = '1241'   0.0 2
    Pu242G    = '242'    0.0 2
    Pu242hG   = '1242'   0.0 2
    U232G     = '232' 0.0 2
    U232psG   = '4232' 0.0 2
    U233G     = '9233' 0.0 2
    U236G     = '236'    0.0 2
    U237G     = '927'    0.0 2
    U237psG   = '4927'   0.0 2
    Np237G    = '937'  0.0 2
    Np239G    = '1939'   0.0 2
    Am241G    = '951'  0.0 2
    Am242G    = '1952'   0.0 2
    Am242mG   = '952'    0.0 2
    Am243G    = '953'    0.0 2
```

```
Cm242G      = '962'    0.0  2
Cm243G      = '963'    0.0  2
Gd154G = '2154' 0.11720  2
Gd155G = '2155' 0.759  2
Gd156G = '2156' 1.0633  2
Gd157G = '2157' 0.81559  2
Gd158G = '2158' 2.4507  2
O16G   = '6016' 11.933
```

Of the 289 ($= 17 \times 17$) fuel rods in the assembly, 25 of them contain borated water, 16 contain a UO_2-$Gd_2\ O_3$ mixture, and the rest are UO_2 rods. The above mixture represents the gadolinium-fuel mixture. It has a lower enrichment (2.63%), which translates to a weight percentage of about 2.18% as shown above. This illustrates that the user must compute the weight percentages of each element separately in a mixture and not merely inputting the fuel enrichment directly. The Gd rods also have a separate self-shielding index. This has been applied due to the large neutron capture cross section of Gd, and so the self-shielding procedure employed here requires its own correction which will be more detailed than that of the regular fuel rods. Again, the other element are listed despite their null values for ease in future parsing. The suffix "G" refers to "gadolinium" so that the uranium-235 in the regular fuel rods ("U235" above) can be extracted separately from that in the gadolinium-fuel mixture ("U235G").

```
* Guide tube material
  MIX 5 COMB 2  1.0
* SF97-4
  MIX 6  900.0  10.0701
    O16T      = '6016'   11.852
    U234T     = '234'    0.03526  1
    U235T     = '2235'   3.622944  1
    U238T     = '8238'   84.49  1
```

```
   Pu238T     = '948'     0.0  1
   Pu239T      = '6239'    0.0  1
   Pu240T      = '1240'    0.0  1
   Pu241T      = '1241'    0.0  1
   Pu242T      = '242'    0.0  1
   Pu242hT      = '1242'    0.0  1
   U232T      = '232' 0.0  1
   U232psT      = '4232' 0.0  1
   U233T       = '9233' 0.0  1
   U236T      = '236'    0.0  1
   U237T      = '927'    0.0  1
   U237psT      = '4927'    0.0  1
   Np237T      = '937'   0.0  1
   Np239T      = '1939'    0.0  1
   Am241T      = '951'   0.0  1
   Am242T      = '1952'    0.0  1
   Am242mT      = '952'    0.0  1
   Am243T      = '953'    0.0  1
   Cm242T      = '962'    0.0  1
   Cm243T      = '963'    0.0  1
;
```

Mixture 5 is, like mixture 2, composed of Zircaloy. Because it is not interesting to study the behavior of the components of this material, we can merely copy the composition from mixture 2 with the COMB (for *comb*ine) keyword. The value of 1.0 means that mixture 5 is made 100% of mixture 2.

Mixture 6 represents the fuel rod of interest; in this case, it is rod SF97-4. The suffix here is "T" for "target rod". It is simple for DRAGON to simulate a fuel rod, or to simulate a full assembly. However, to extract details about a particular rod within a fuel assembly required the specification of a mixture representing the rod of interest. Then, as will be shown below, this target mixture will be assigned the required

location in the reactor geometry. Since this mixture is otherwise indistinguishable from mixture 3, it is unnecessary to use a new self-shielding index. Thus to extract details for rod SF97-4, in the DRAGON output, we can search for inventory records labeled with "T".

In a real fuel evolution/depletion calculation, it may be necessary to change the values of various mixtures as a function of time or burnup. For example, the thermal power and boron will fluctuate over the course of a fuel cycle. The LIB: module contains a special mode to enable this behavior. See the DRAGON User Manual, section 3.2.7[**?**].

Finally, note that the semicolon is only written at the very end of the LIB: module.

0.5 Assembly Geometry

We now illustrate the specification of the assembly geometry in DRAGON.

```
*----
* C1: Fuel Cell (UO2)
* C2: Guide Tube (Zircaloy)
* C3: Water bath (0.0538 cm thick)
* C4: Corner water bath
* C5: Fuel Cell (UO2-Gd2O3)
* C6: Water bath (0.0538 cm thick)
* CX: SF-97 (Target for benchmark)
* CY: SF-97 (Target for benchmark)
* CZ: SF-97 (Target for benchmark)
*----
ASSMB := GEO: ::
CAR2D 10 10
  X- DIAG X+ REFL Y- SYME Y+ DIAG
CELL     C2 C1 C1 C2 C1 C5 C2 C1 CX C3
            C1 C1 C1 C1 C1 C1 C1 C1 C3
               C5 C1 C1 C1 C1 C1 C1 C3
```

```
C2 C1 C1 C2 C1 C1 C3
   C1 C1 C5 C1 C1 C3
      C2 C1 C1 C1 C3
         C1 C1 C1 C3
            C1 C1 C3
               C1 C3
                  C4
```

Now that we've specified the fuel mixtures, we can set the physical location of these mixtures in our assembly. For this, we use the GEO: module, which we assign to the variable ASSMB. We begin by labeling the shape of our assembly; in this case, it is a CAR2D, for 2-dimensional Cartesian shape (i.e., rectangular grid of rods). The directive above means that we are defining a 10×10 arrangement of rods. The items labeled CN are the individual fuel rods, which will themselves be specified next.

The next line is the all-important declaration of boundary conditions (BCs). To understand them, first note that in fact we have only shown an *eighth* of the assembly above. We must describe to DRAGON how we would like to "unfold" this assembly until we reach its full size. It is obvious that we would like to exploit the symmetry along the diagonal. To do this, we use the expression "X- DIAG Y+ DIAG". Here, "X-" means "incoming from the negative-X surface", which in this case is the left side of the above arrangement. Likewise, "Y+" means "incoming from the positive-Y surface". The combination of these two commands then instructs DRAGON to reflect the arrangement about a line passing through the upper left. It also means that DRAGON should reflect the assembly through the positive-Y and negative-X surfaces. Thus the assembly begins to unfold "upwards" and "towards the left". A natural question then is: do we copy the diagonal itself in this reflection, or do we merely copy around the diagonal? If we want the former, we would use the SSYM keyword, but we do not wish to duplicate the diagonal, so we use the SYME keyword. The "Y-" here means to reflect though a surface coming from the negative-Y direction (i.e., the bottom of the arrangement).

How do we get a 17×17 assembly from these BCs? To see, for a moment let us ignore the C3 and C4 cells above and pretend that we had instead specified a 9×9 Cartesian shape (CAR2D 9 9). Upon reflection through the diagonal using the SYME keyword would give us an assembly of dimensions $2 \cdot 9 - 1 \times 2 \cdot 9 - 1$ (we subtracted since the diagonal was not duplicated), or a 17×17 assembly. Finally, "X+" means "place a reflecting surface on the positive-X surface". So the assembly begins to unfold upwards, downwards, and to the left, and on each surface, there is a reflecting BC. Cells C3 and C4 are very small cells meant to represent the very thin water bath surrounding the assembly, and so the assembly maintains its 17×17 shape even though technically it is 18×18.

```
::: C1 := GEO: CARCEL 2
  MESHX 0.0 1.265  MESHY 0.0 1.265
  RADIUS 0.0 0.4025 0.475 MIX 3 2 1 SPLITR 2 1 ;
```

We now enumerate the properties of the individual cells making up our assembly. The ":::" operator tells DRAGON that this object is within the scope of the overall GEO: declaration. C1 is defined to be a CARCEL, meaning Cartesian cell. This is a square cell containing a series of cylindrical pins. MESHX/Y give the dimensions of the surrounding square cell in centimeters. RADIUS expects an array of numbers delineating the annular boundaries between regions in the cell. Thus we can see that this cell will be split into 3 regions, with the radius varying from [0..0.4025], [0.4025, 0.475], and [0.475, edge of cell]. Now we must assign a mixture to each region, and we do this in the same order that the regions were defined. Recall that mixture 3 was the UO_2 fuel, mixture 2 was the cladding, and mixture 1 was the moderator. We can see that the cell now consists of an inner fuel region surrounded with cladding, which is in turn surrounded by water. Finally, SPLITR will split the cell radially in the the specified number of subregions. This is done to increase discretization which is important in mixtures that require extensive self-shielding. SPLITR 2 1 means to split the innermost (fuel) region into 2 regions, and to split the cladding into 1 region (i.e., not to split it at all). The default value for splitting is 1, which is why the

outermost region did not have to be specified. We can see that C1 is our fuel cell in the configuration above.

```
   ::: C2 := GEO: CARCEL 2
          MESHX 0.0 1.265 MESHY 0.0 1.265
          RADIUS 0.0 0.573 0.613
          MIX 1 5 1 SPLITR 2 1 ;
::: C3 := GEO: CAR2D 1 1
     MESHX 1.265 1.3188 MESHY 0.0 1.265
     MIX 1 ;
   ::: C4 := GEO: CAR2D 1 1
     MESHX 1.265 1.3188 MESHY 1.265 1.3188
     MIX 1 ;
```

As mentioned previously, C3 and C4 are the "water bath" cells with a thickness of 0.0538 cm. We can see in the specification of MESHX/Y that only the δx of the mesh (size) of the cell is important to specify. Also, we can see that geometries can be defined recursively; C3 and C4 are of type CAR2D, which are both themselves embedded in a larger CAR2D structure. This allows for a variety of complicated reactor geometries.

```
   ::: C5 := GEO: CARCEL 2
     MESHX 0.0 1.265 MESHY 0.0 1.265
     RADIUS 0.0 0.4025 0.475 MIX 4 2 1 SPLITR 2 1 ;
   ::: CX := GEO: CARCEL 2
     MESHX 0.0 1.265 MESHY 0.0 1.265 SPLITR 8 1
     RADIUS 0.0 0.4025 0.475 MIX 6 2 1 ;
   ;
```

Cell CX is the fuel rod we wish to model, SF97-4. Recall that we have to define a unique mixture (here, 6) as well as a unique cell to extract useful data from a particular rod. We have chosen a high radial splitting to ensure that self-shielding effects are taken into account.

Notice that each cell definition as well as the overall GEO: declaration must end with a semicolon.

0.6 Tracking

We now specify the tracking parameters.

```
DISCR Track := EXCELT: ASSMB ::
TRAK TISO 10 12.0 ;
```

The variable DISCR will contain the output of the EXCELT: module, and the file Track will contain a record of Tracking parameters for later debugging. It can be omitted if desired. The tracking module requires at the very minimum one argument, which is a GEO: object. The TRAK TISO command tells DRAGON to use isotropic reflection at any boundaries in the assembly. The first value after TISO is the angular quadrature parameter, and the final value is the integration line density. To understand these, we can imagine that the tracking module replaces our reactor geometry with sets of lines. These lines are all parallel with one another, and have a density specified in inverse centimeters. These lines represent the paths of neutrons through the assembly, and using the mixture information from the geometry definition, DRAGON can determine which materials a given neutron on a certain path will interact with. We can also take our set of lines and rotate through some fixed angle and repeat this process. In this way, we can cover the entire assembly with neutron paths which will later be used in calculating collision probabilities and neutron fluxes. The fixed angle is the angular quadrature parameter, and means that in this case, the tracking module must generate 10 sets of lines (each undergoing successive rotations of 360 / 10 = 36°), and their density is 12 lines per cm. Increasing either of these parameters will enable a more detailed and accurate simulation, at the cost of longer running times.

We can see here an instance of DRAGON's modularity in action. Had we decided to use one of the many other tracking modules in DRAGON, only the above code would have to change.

0.7 Self-shielding, Collision Probabilities, and Flux

SHI: is the self-shielding module used in DRAGON version 3.

```
LIBRARY := SHI: LIBRARY DISCR Track :: LEVE 1 ;
```

DRAGON version 4 has a more sophisticated module for self-shielding, USS: (for universal self-shielding), however, for the Takahama-3 simulation of SF97, the SHI: module was found to be sufficient. It modifies the cross section library; this is represented in DRAGON by assigning the results of applying SHI: to LIBRARY and storing the result back into LIBRARY. The LEVE 1 option allows the different self-shielding indices from the LIB: module.

```
CP := ASM: LIBRARY DISCR Track ;
```

The ASM: module calculates collision probabilities, and takes a cross section library and tracking parameters as arguments.

```
CALC := FLU: CP LIBRARY DISCR :: TYPE K ;
```

The FLU: module takes the collision probabilities from the previous step and uses them, along with the tracking and libraries, to compute the neutron flux. The TYPE keyword here takes the value of K, implying that K_{eff} is the eigenvalue to be solved for in the multigroup neutron transport equation. The evolution of K_{eff} is a useful cross-check in cases where this information is available.

0.8 Fuel Depletion

The EVO: module evolves the fuel concentrations in time.

```
BURNUP LIBRARY := EVO: LIBRARY CALC DISCR ::
   EDIT -2 DEPL 7.17118E-07 DAY
   POWR <<Power>> ;
```

The last module that we will consider is the EVO: module, which evolves the fuel inventories in time. It requires the current value of the neutron flux (CALC), the tracking (DISCR) and the current values of the component inventories (LIBRARY).

The LIBRARY variable, and all of the mixtures that it contains, is evolved for a certain length of time. The POWR keyword indicates that the fuel evolution will be done at a constant power. The brackets are CLE-2000 syntax for interpolating a variable's value. The value of the POWR keyword must be the *specific power*, which is given in units of power per initial mass of heavy metal. The DEPL keyword requires the length of time required for evolution, in this case specified in days. The EDIT -2 directive is unique to the author's code; it informs DRAGON that fission rate calculations are to be performed and output.

```
NewBurnFile := BURNUP ;
END: ;
QUIT "LIST" .
```

After the fuel evolution, the details of the evolution can be output in a file labeled NewBurnFile, a user-declared variable. This can then be passed onto subsequent calculations, and illustrates how to control output from DRAGON.