EQL0D Manual

Last updated : 13/07/2016

# Introduction

EQL0D [1] is a MATLAB-based wrapper for the Serpent 2 Monte Carlo neutronics code [2] developed at the Paul Scherrer Institut (PSI) in Switzerland and designed to predict the evolution of the fuel composition in Molten Salt Reactors (MSR) during burn-up. Originally, its aim is to predict the equilibrium fuel composition in the case of a system that is regularly refueled to conserve the total fuel mass. It can however be used to compute the fuel composition over finite time steps, for example in the case of open cycle reactors without any fuel recycling.

MSRs have peculiar specificities regarding their fuel cycle, such as, for example, the constant removal of some fission product (FP) species during burn-up due to their limited solubility in the salt, as is the case for noble gases or noble metals. EQL0D includes several options that allows the user to model most processes that are required for MSR fuel cycle modeling, including continuous or batch-wise removal from or addition to the fuel of specific isotopes or elements, or even transfer of all or part of the fuel to another material.

The routine is tightly bound to the Serpent 2 code and its use requires the user to be knowledgeable of both MATLAB and Serpent 2 as a standalone. Serpent is used to determine isotope-wise reaction rates within the reactor and to produce a burn-up matrix for one or several materials. EQL0D will then modify this matrix according to the user input to simulate the continuous addition/removal/transfer processes. EQL0D then used the Chebyshev Rational Approximation Method (as in Serpent in its standalone version) to predict the new fuel composition after a given time step. Finally, it will apply the batch-wise processing steps included in the user input, if any. In the case of an equilibrium composition search, this will be looped over until the composition(s) converge(s) toward an equilibrium. In the case of a finite number of steps however, the process will be repeated the given number of times.

# Computational Aspects

## Installing and launching the routine

The only requirement for the user to install the routine is that the files of the routine be on the MATLAB path of the user’s MATLAB installation.  
Launching the routine can be done using the shell script provided in the files folder.

## Code structure

The computational scheme briefly described in the previous paragraph is outlined on figure 1.

The code is written in a modular way using functions and some elements of Object-Oriented Programming (OOP). As depicted on the previous figure, the code essentially consists of two imbricated loops, the main or outer loop (representing the so-called **cycles**), and the inner loop (representing the so-called **steps**). Cross-sections are refreshed at each call of the Serpent code, that is, every new cycle. In the case of an equilibrium calculation, the code will iterate over both steps and cycles until a user-input convergence criterion is reached or a user-input maximum number of steps/cycles has been reached. In the case of a finite number of steps/cycles, the code will simply compute the said number of steps/cycles.

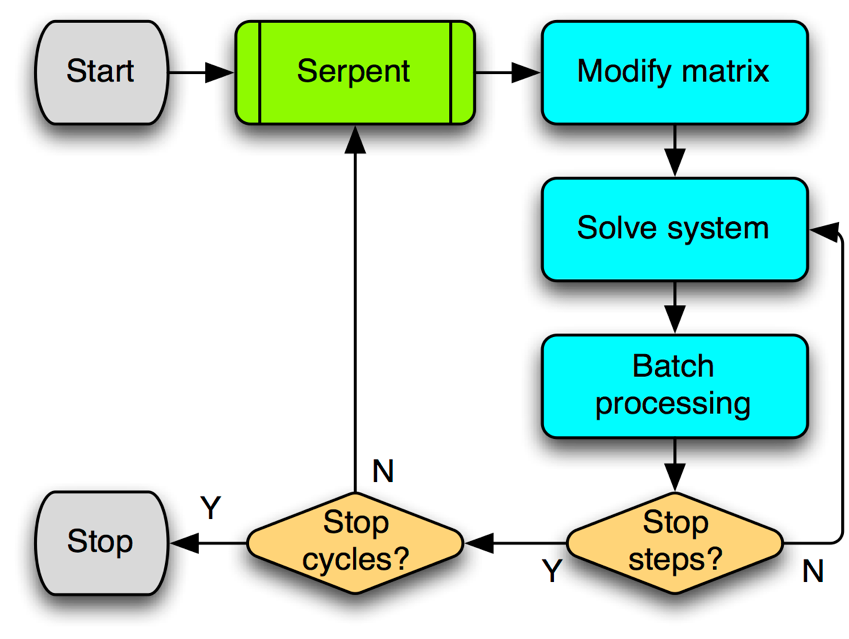


Figure 1: Computational scheme of the EQL0D procedure

The code fundamentally necessitates at least two input files: The Serpent input file to be run, containing geometry description and Serpent-related computational options, and the EQL0D input in a MATLAB .m format containing the EQL0D-related computational options. These files are further described in the next chapter.

The code additionally contains a default-options file called EQL0D\_config.m that the user can freely modify with his own default preferences. The options set in the EQL0D\_config.m file will in any case be overwritten by the simulation-specific user input.

## Chebyshev Rational Approximation Method

The Chebyshev Rational Approximation Method (CRAM) is used in EQL0D as it is used in Serpent 2 [3]–[5] to solve the Bateman equations set by approximating the exponential matrix. The Bateman equations describe the evolution of the composition of fuel with time under a given neutron flux:

+

Which, for a great number of isotopes, can be simplified in matrix format:

Which can be solved after a time-step T using:

CRAM approximates the exponential of the burn-up matrix in a very accurate way that is more tolerant of eigenvalues of very different orders of magnitude (in the case of burn-up at high flux with very short-lived nuclides in the composition, for example) compared to other methods such as the Padé approximation. Other burn-up codes sometimes use pre-defined burn-up chains that take shortcuts through short-lived nuclides, which is acceptable in the case of solid-fueled reactors but questionable in the case of liquid-fueled reactors, in which even short-lived nuclides can be removed from the fuel composition before decaying, in which case using shortcuts would predict the wrong fuel composition.

## Online processing steps

In the case of online processing steps, such as the removal of volatile nuclides, an additional removal term must be included in the Bateman equation:

+

The last term in the above equation represents proportional removal of the isotope *i*, with being the removal rate (units of s-1) which is also the inverse of the so-called cycle time or the fractional reprocessing rate (fraction of fuel volume reprocessed per second). The variable is the **efficiency**, or **share**, described later in the manual.

In the case of transfer toward another material (such as in the case of tracking of removed isotopes), the equation for the same isotope in the destination material is modified with the opposite proportional rate:

Where the ratio of volumes is added to conserve the total number of nuclides.

## Batch reprocessing steps

The reprocessing steps in batch-wise mode are performed after burning the fuel in a given step. Depending on the mode, the amount refueled can be fixed by other criteria (for example, conserving the total actinide mass) or by the rate and the time step. In the latter case, the amount of isotope I is modified in the source material to:

Which tends toward the proportional removal described above for very small time-steps. In the case of transfer toward other materials, the amount removed at the source is simply added to the destination material.

# Input files and data

As mentioned before the inputs consist of the Serpent input file, called <input>, and the EQL0D input file in MATLAB format, <input>.m. The first one uses standard Serpent syntax to describe the geometry and Serpent options such as number of neutrons per cycles, etc. The latter describes the options to be used by the MATLAB routine, that is, the aforementioned MAT, REP and OPT structures describing the materials, reprocessing streams, and options, respectively. One major advantage of using MATLAB as a platform is the possibility of performing computations directly in the input for the user’s convenience.   
  
In all cases, the <input>.m file should start with the following line:

load(‘<defaultLibary>’)

Which tells the code to load the corresponding nuclear data library (either ‘jeff311’ or ‘endfb7’).

## OPT structure

The OPT structure consists of the fields listed in the following table. Fields marked with (L) in the description are normally set using the EQL0D launcher, and not by user input.

|  |  |  |  |
| --- | --- | --- | --- |
| **Field** | **Description** | **Type** | **Example** |
| nCores | Run Serpent using nCores (L) | integer | 6 |
| restartCalc | Restart calculation (L) | logical | true |
| debugMode | Runs without recalling Serpent (L) | logical | true |
| resetCounters | Restarts & resets counters (L) | logical | true |
| iterMode | ‘steps’ or ‘equilibrium’ | string | ’steps’ |
| nSteps | Number of Steps per Cycle | integer | 12 |
| nCycles | Number of Cycles | integer | 10 |
| cycleLength | Cycle length (Serpent units) | double | 365 |
| printSteps | Print composition each step | logical | false |
| printCycles | Print composition each cycle | logical | true |
| computeKeff | Compute k-eff between cycles | logical | true |
| printKeff | Print k-effective values in file | logical | true |
| serpentPath | Path to the Serpent executable | string | ‘sss2’ |
| CONV | Convergence criteria | struct | See below |
| t12Limit | Ignore iso. w/ half-life lower than… | double | experimental |
| defaultDataLibrary | Default data library | string | ‘endfb7’ |
| nbZAI | ZAI of isotopes used in the neutron balance calculation | integer | 922330 |
| writeMail | Writes an email to $LOGNAME at the end of the calculation | logical | true |

Values like nSteps and cycleLength can be given as a vector, each value corresponding to a different cycle.

### The CONV structure

The CONV sub-structure is used to specify the convergence criteria used for the equilibrium mode of iteration. It contains two sub-fields, inner and outer, which separately specific criteria for the inner and outer (where Serpent is run) iterations. The specifications are given on the following table

|  |  |  |  |
| --- | --- | --- | --- |
| **Field** | **Description** | **Type** | **Example** |
| criterion | Name of the convergence criterion | string | ‘maxActRelDiff’ |
| value | Value of convergence criterion | double | 0.0001 |
| maxIter | Maximum number of iterations | double | 10 |
| cutoff | Cutoff concentration below which nuclides are ignored when evaluating the criterion | double | 1e-10 |

The two criteria defined in the current EQL0D version are ‘maxActRelDiff’ and ‘maxRelDiff’, and refer to the maximum of the relative difference in concentration of either actinides or all nuclides (with concentration above the cutoff value) between two steps. The value specified refers to the maximum tolerable value. When below the maximum, the corresponding iteration will stop. If not reached within the maximum number of iterations, the iteration will stop anyway to avoid runaway calculations (for example when the amplitude of the oscillations due to the stochastic nature of the Serpent calculation are greater than the criterion).

## MAT vector

MAT is a vector variable containing materials, a class created specifically for EQL0D. Materials are defined using the constructor method of the class which returns a material, that is:

Mat(<name>,<burnt>,<dens>,<volume>,<temperature>,<elements>,<densities>);

Where:

* <name> is the name of the material
* <burnt> is true if the material is burnt (in the neutron flux)
* <dens> is the density of the material (Serpent syntax, except that the Serpent keyword ‘sum’ is replaced by 0)
* <volume> is the fuel volume
* <temperature> is the temperature for the cross-sections in Serpent; for example, 900 for the 900K XS in Serpent
* <elements> lists the ZAI of the isotopes included in the material
* <densities> lists the densities of the isotopes included

The Serpent syntax for material inputs has been kept, excepted the keyword ‘sum’ used in Serpent to specific absolute mass or atomic densities in the material definition is replaced by the numeric value 0. Both the total density and the individual fractions can be input as a mass using negative numbers and as atomic using positive numbers.

Materials used to track removed isotopes from the fuel composition are created by the routine and are not to be input by the user.

Which describes a material named fuel, burnt in the neutron flux, having a density of 3.6 g/cm3, a volume of 1000 cm3, and a XS temperature of 900 K. It contains Na-23, Cl-37 and U-238 in the proportions given.

## REP structure

The REP vector is a vector of structures describing the processes to be applied to the fuel, such as reprocessing, insoluble FP removal, refueling, etc. Each structure of the vector describes one process. The fields used to describe the processes are given in the following table.

|  |  |  |  |
| --- | --- | --- | --- |
| **Field** | **Description** | **Type** | **Example** |
| name | Name of the stream | string | ‘volatile’ |
| source | Source material (where the isotopes are taken from) | string | ‘fuel’ |
| destination | Destination material (where the isotopes are put) | string | ‘fuel’ |
| elements | Isotopes or elements moved | double | 922350 or [10 11] |
| share | Shares of isotopes moved | double | 1 |
| rate | Reprocessing rate (inverse of the cycle time) | double | 1.3e-10 |
| type | Continuous or batch | string | ‘continuous’ |
| mode | Process applied | string | ‘refillTot’ |
| frequency | Step frequency (experimental) | double | 2 |

The share and frequency fields are not mandatory, but if not specified, the value will be set to 1 for all nuclides/steps.

In most cases, either the source or destination material does not exist, in which case the value should be set as ‘void’. In the case of tracking of the fate of removed nuclides (for example to compute the evolution of the composition of nuclides removed from the fuel, or in the case of the transfer of isotopes from one material such as the blanket to another material such as the fuel) the name of a material should be provided. If the material does not exist in the system, it will be created by the routine.

The elements vector can list either specific isotopes by using the ZAI or all isotopes of a given element (for example, 92 for all uranium isotopes).

The share vector has different meanings depending on the type of reprocessing done, but in most cases it reflects an efficiency of the removal, for example to simulate the slower or faster removal of a given element or isotope. In the case of a value higher than 1, the isotope/element will be removed faster than the rate and in the case of a lower value, slower than the rate.

The mode string can either be ‘addVolume’, ‘remove’, ‘refillTot’, or ‘refillAct’. The effect of these modes is described in the following table.

## Example input

The following lines are an example input to illustrate how to specify all the aforementioned options.

load('endfb7')

OPT.nSteps=1;

OPT.cycleLength=365;

OPT.iterMode='equilibrium';

OPT.printCycles=true;

lambda=3.701E-10;

zai=[110230 170370 922380];

dens=[9.671285e-03 2.332485e-02 4.551197e-03];

MAT=Mat('fuel',1,-3.6,20E6,900,matzai,matdens);

REP=struct('name',{'volatile',’reprocessing’,’feed’},...

'source', {'fuel','fuel','void'},...

'destination',{'volatile',’void’,'fuel'},...

'elements', {[1 2 7 8 10 18 36 41:1:47 51 52 54 86],[39 40 48 49 50 51 53 57:1:84],922380},...

'rate', {1/30,lambda,1},...

'type', {'continuous','continuous','batch'},...

'mode', {'remove','remove','refillAct'}...  
);

# Output files

The usual output files produced by Serpent are kept in separate folders for each Cycle and can be used by the user for other purposes such as few-group XS generation. Files generated by EQL0D itself are:

* EQL0D.log, the logfile of the EQL0D computation
* CycleXXX, the folders containing data related to each step, including Serpent files created at each cycle
* <filename>.mat, the MATLAB archive containing all variables of the computation. Used for post-processing or restarting the calculation
* <material>.serp files containing the material compositions for the Serpent computations
* <files>.bak, backup files of all the files modified by the EQL0D routine
* <material>\_cycleXXX(\_stepYYY).txt, the composition printed for all materials in the system at a given cycle or step.
* <parameter>.txt, the output file for a given parameter such as k-effective (if computed) or the volume added in some processing streams.

# References

[1] B. Hombourger, J. Krepel, K. Mikityuk, and A. Pautz, “The EQL0D Procedure for Fuel Cycle Studies in Molten Salt Reactors,” in *Proceedings of the 2016 International Congress on Advances in Nuclear Power Plants (ICAPP ’16)*, San Francisco, CA, USA, 2016.

[2] J. Leppänen, M. Pusa, T. Viitanen, V. Valtavirta, and T. Kaltiaisenaho, “The Serpent Monte Carlo code: Status, development and applications in 2013,” *Ann. Nucl. Energy*, vol. 82, pp. 142–150, Aug. 2015.

[3] M. Pusa and J. Leppänen, “Computing the Matrix Exponential in Burnup Calculations,” *Nucl. Sci. Eng.*, vol. 164, no. 2, pp. 140–150, 2010.

[4] M. Pusa, “Rational Approximations to the Matrix Exponential in Burnup Calculations,” *Nucl. Sci. Eng.*, vol. 169, no. 2, pp. 155–167, Oct. 2011.

[5] M. Pusa, “Numerical methods for nuclear fuel burnup calculations: Maria Pusa.,” Ph.D. Thesis, Aalto University, Espoo, Finland, 2013.