

FISSIONCORE Brief Users Guide

October 2013

Cameron Reed (reed@alma.edu); Klaus Rohe (klaus-rohe@t-online.de)

<https://mail.alma.edu/home/reed@alma.edu/Briefcase/FISSIONCORE>

First, read the published paper describing FISSIONCORE and FISSIONRAND in order to get a general sense of the programs and the units and parameters involved.

PROGRAM VERSIONS

<i>FORTTRAN version</i>	
FISSIONCORE.f	<p>FORTTRAN version that uses compiler's built-in random-number generator. Expects VALUES file of format</p> <p>$d, R, \sigma_f, \sigma_s, v_{neut}$ $v, v_o, \Delta t, t_{max}$</p>
FISSIONRAND.f	<p>FORTTRAN version with RAN2 random-number subroutine. Expects VALUES file with IDUM seed:</p> <p>$d, R, \sigma_f, \sigma_s, v_{neut}$ $v, v_o, \Delta t, t_{max}, IDUM$</p>

The C version is a rewrite of the FORTRAN version in standard C. The semantics and the names of the major variables used in the FORTRAN version have been retained in the C code. The C code also contains comment lines which give hints to the related source code line number in the FORTRAN code.

<i>C Version</i>	
fissioncore.c	<p>This is the C module which contains the main function.</p> <p><i>fissioncore.exe</i> is a windows executable. On Windows (and analogue on LINUX, UNIX or Mac) you can call it from the command prompt: <i>fissioncore values.dat fisres [seed]</i> Expects <i>values.dat</i> file of format</p>

	<p>$d, R, \sigma_f, \sigma_s, v_{neut}$ $v, v_o, \Delta t, t_{max}$</p> <p>The first command line parameter is the name of the simulation parameter file the second parameter is the first part of the name of the result files. There are two result files generated one with name <i>fisres.txt</i> which is an easily readable text file and the other with the name <i>fisres.csv</i> in a ';' separated format which can be processed by spread sheet programs. The third parameter which is optional is the seed for the random generator. If the third parameter is not given <i>seed=1000</i> is used as a default. In '<i>fissioncore.c</i>' the functions defined in '<i>fisutility.h</i>' and '<i>core.h</i>' are called.</p>
core.h, core.c	This module is the work horse of the simulation it initializes all the variables in the function ' <i>intializeSimulation(...)</i> ' and then starts the simulation in the function ' <i>doSimulation(...)</i> '.
neutron.h	In this header file the C structure 'Neutron' is defined as well as an array 'Neutrons' of pointers to type 'Neutron' structures which. The array has a capacity of ' <i>MAX_NUT</i> '. This constant is also defined in this header file. The 'Neutron' structure represents a single neutron during the simulation it contains xyz-coordinates of its position, the components of the velocity vector as well as the state of the neutron. The state of a neutron can be 'A' alive, 'C' consumed in fission, 'E' escaped, 'U' currently unused.
fissutility.h, fissutility.c	The module ' <i>fissutility.c</i> ' contains utility functions for random velocity generation, reading the file which contains the simulation parameters and formatted writing to result files used in ' <i>fissioncore.c</i> '.
makefile	Makefile to generate the executable with the make utility. The windows executable has been created with <i>GNU gcc 4.8.1</i> and <i>GNU Make 3.82</i> as available from http://www.equation.com/servlet/equation.cmd?fa=fortran . There are no Windows specific functions used in the code and therefor it should be no problem to generate an executable for all operating systems for which <i>GNU gcc</i> or other standard C compilers and the <i>Make</i> utility are available (LINIUX, UNIX, Mac, ...). The current ' <i>makefile</i> ' is for Microsoft Windows it has to be adapted accordingly if used on other operating systems.

The Java version is a redesign of the C version. The structure of this version has similarity to the C version but one has to keep in mind that Java is a pure object oriented programming language but C is a procedural language. The architecture of the Java version is a compromise on performance issues, good object oriented design and similarity (comprehensibility) across the different programming language versions of this simulation.

Java Version	
FissionCore.java	<p>This is the java class which contains the main method in which all methods of the other objects which provide functionality for the simulation are executed. To execute the java version you have to go to the directory where the java source files reside and compile them with the command 'javac *.java' (you can alternatively make a project with Eclipse or Netbeans if you like to use an IDE). If compilation is finished you can start the simulation with the following command:</p> <p><i>java -Xmx1024m FissionCore values.dat valuesres [seed]</i></p> <p>It is important to remark that java has to be called with the parameter '-Xmx1024m' (this means 1 GB of heap memory) this determines how much heap memory the JVM can use. The heap memory required by the simulation is determined mainly by the maximum number of neutrons which can be processed ('MAX_NUT') defined in the source code and the number of nuclei in the three dimensional arrays 'NUCSTATUS' and 'LAST' computed dynamically at run time from the Parameters 'D' and 'RC'. In the current java source MAX_NUT = 800000 and simulations with 'D=180' and 'RC= 22000' were feasible. If these values are increased the -X parameter has to be adopted accordingly. (For details of the meaning of '-Xmx1024m' see for example: http://stackoverflow.com/questions/1098488/jvm-heap-parameters) or http://stackoverflow.com/questions/5374455/what-does-java-option-xxmx-stand-for</p> <p>The seed parameter is optional is the seed for the random generator. If the fifth parameter is not given seed=1000 is used as a default.</p>
Core.java	<p>This is the work horse for the simulation. It provides the two static methods 'intializeSimulation(...)' and 'doSimulation(...)'. They provide the same functionality as the respective functions in the module 'core.c' of th C version described above.</p>
SimulationParameters.java	<p>An instance of this java class has all the simulation parameters as getters and setters which have to be</p>

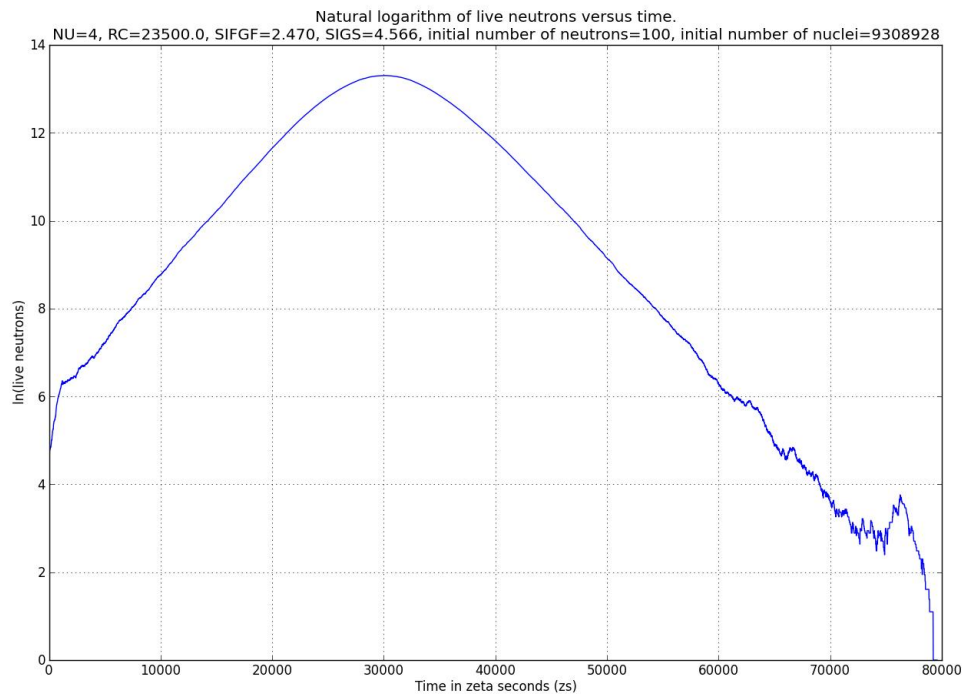
	available before the simulation starts. An instance of this class is generated by calling its constructor with the name of the simulation parameter file (<i>values.dat</i>) as a parameter.
Neutron.java	An instance of this class represents a single neutron during a simulation run. See description of 'neutron.h' for the C version. The class has an analogous function as the C structure defined in 'neutron.h' (see above).
FissUtility.java	Utility class which contains static methods analogous to that in the C module ' <i>fissutility.c</i> ' (see above).
FissionCoreException.java	Application specific exception class for error handling.

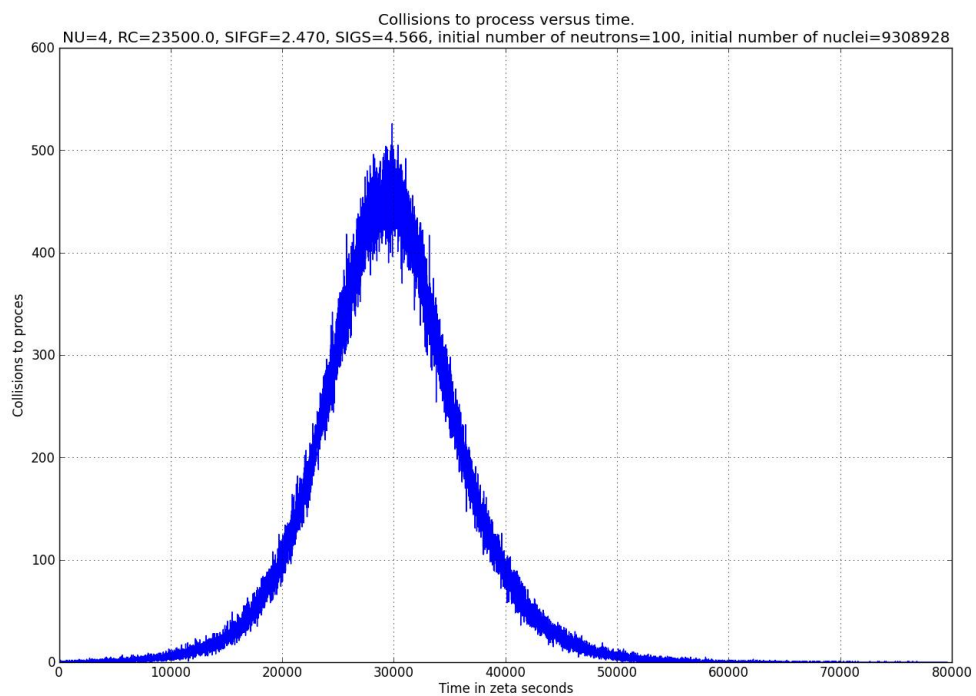
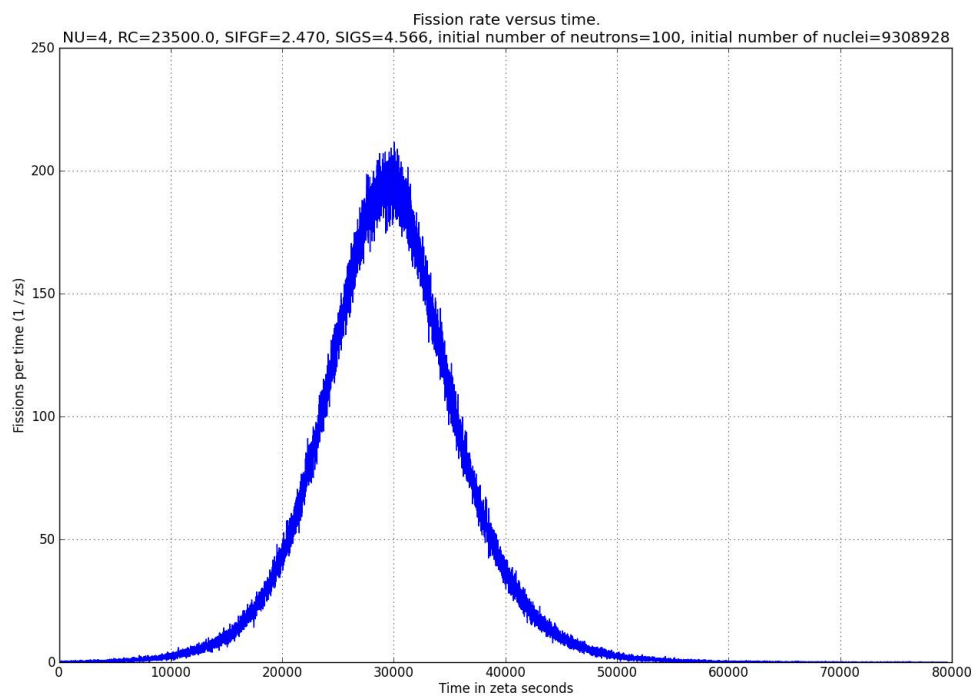
Python utility to evaluate and plot the data in the csv result files.

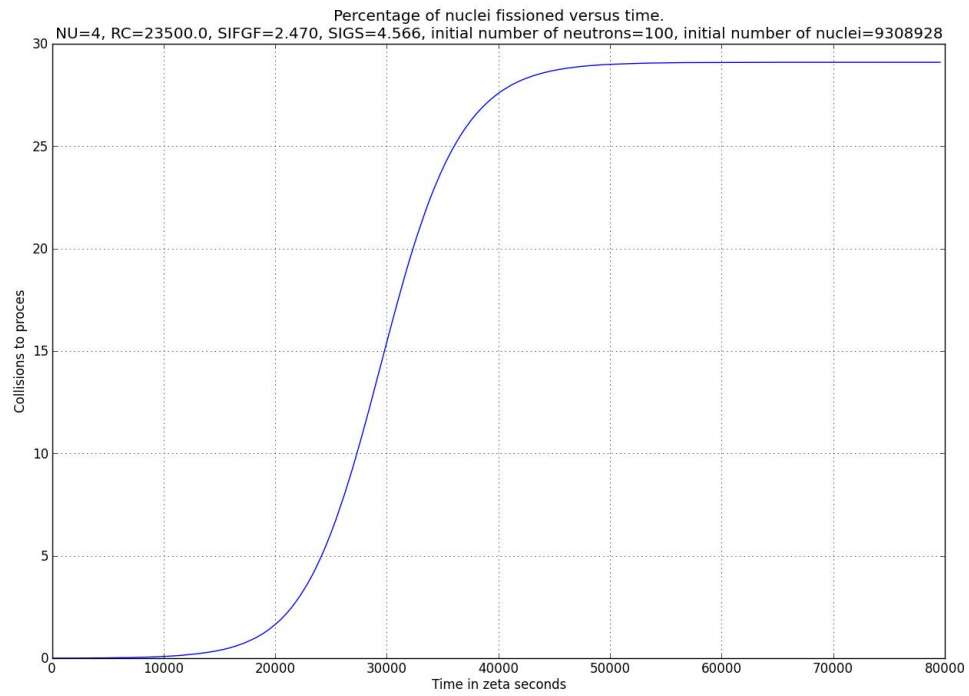
The python script '***evalfisres.py***' takes *.csv files generated by the simulations programs and generates graphical plots which are stored in jpeg files. These files can be easily included in other documents. '***evalfisres.py***' is based on ***matplotlib*** (see <http://matplotlib.org/>). It is called from the command line as follows:

python evalfisres.py fisres.csv

It reads the simulation result file '***fisres.csv***' and generates the following plots stored in the files '***LogLiveNeutrons_fisres.jpeg***', '***FissionsPerTime_fisres.jpeg***', '***Collisions2Process_fisres.jpeg***' and '***PercentageFissioned_fisres.jpeg***'. Examples are shown below:







This python script can be easily extended to generate plots for other kinds of evaluations of 'FISSIOBCORE' simulation results.

Detailed comments on the FORTRAN versions of FISSIONCORE:

The user need only supply a brief two-line file, VALUES, which gives input parameters. These are entered in two lines:

**$d, R, \sigma_f, \sigma_s, v_{neut}$
 $\nu, \nu_0, \Delta t, t_{max}$**

For example, if the contents of VALUES is

300.0,39000.0,1.235,4.566,19.56
10,100,0.5,25000.

Then we have

$d = 300$ fm
 $R = 39000$ fm
 $\sigma_f, \sigma_s = 1.235, 4.566$ barns
 $v_{neut} = 19.56$ fm/zs
 $\nu = 10$ neutrons per fission
 $\nu_0 = 100$ initial neutrons
 $\Delta t = 0.5$ zs
 $t_{max} = 25,000$ zs

The program creates two output files: CORE-RESULTS, which gives a detailed listing of neutron numbers, number of fissions processed, number of escapees, etc., at every 10th timestep, and CORE-PLOT, a file which lists only time, number of neutrons, and natural logarithm of number of neutrons every 10th timestep; this latter file is to help in producing quick EXCEL plots of the number of live neutrons as a function of time. If 105,000 live neutrons is reached before t_{max} , the program stops automatically. This odd-looking value was chosen in order to get to 100,000 neutrons recorded in the output files; some “hot” combinations of parameters can lead to addition of hundreds of neutrons per timestep.

The next page gives a listing of some of the more important program variables. This is followed by some sample output.

Lattice-Core Simulation Variables (not all “utility” variables are listed – just important ones)

General variables

D	Lattice side length
RC	Core radius (fm)
ETA	Number of nuclei across diameter of core = $2*RC/D + 1$
DT	Δt , in units of 10^{-21} sec
FISS	Number of fissions processed
NU	ν
PI	π
RATIO	σ_f/σ_t
RNUC	Nuclear radius (fm)
SIGF, SIGS, SIGT	Cross-sections (bn)
TIME	Current elapsed time, 10^{-21} sec
TMAX	Maximum simulation time
VEL	Neutron speed, fm per 10^{-21} sec

Collision-Processing Variables

NCOLL	Number of collisions to be processed currently
PAIR1(10000)	List of neutrons in collisions. Up to 10000 collisions/timestep
PAIR2(10000,3)	(i, j, k) lattice location of nucleus involved in a collision
ID1	ID number of neutrons in collision to be processed
R	Separation of neutron/nucleus in possible collision currently being examined
DIST	Array of neutron/nucleus distances in putative collision
STATUS	Status of a putative collision (1, 0, -1, -2) = (fission, scatter, not used, pending) set to -1 if another neutron closer
REJECT	Collisions rejected due to 2 or more neutrs striking a given nucleus
SCATT	Number of scatterings processed

Nuclei-Related

NUCSTATUS(600,600,600)	Array indicating status of individual nuclei at lattice corner (i, j, k) (1, 0, -1, -2) = (live, fissioned, dead due to overlap, not yet assigned)
NUCALIVE	Number of live nuclei at any time
NUCS	Number of nuclei initially read in
LAST(600,600,600)	Array indicating last neutron that nucleus at lattice corner (i, j, k) collided with

Neutron Related

ESC	Number of escaped neutrons at any time
LIVENUT(400000)	Array indicating status of individual nuclei (1, 0, -1, -2) = (live, consumed in fission, escaped, not yet assigned)
NEUTS	Total number of neutrons created, including initial ones
NUTINIT	Initial number of neutrons
NUTALIVE	Number of live neutrons at any time
NUTX, NUTY, NUTZ	Neutron (x, y, z) locations, each an array of size 400000
VNUTX, VNUTY, VNUTZ	Neutron speed components, each an array of size 400000

Here is an abbreviated copy of the output in the case of the VALUES results above.

LATTICE-CORE SIMULATION RESULTS

COPYRIGHT BRUCE CAMERON REED & KLAUS ROHE 2013

CORE RADIUS (FM) 39000.
 LATTICE SPACING (FM), ETA 300.0 261
 FISS & SCATT SIGMAS (BN) 1.235 4.566
 NUCLEI ALIVE INITIALLY 9201234
 NUCLEAR RADIUS (FM) 13.589
 NEUTRONS PER FISSION 10
 NEUTRON VELOCITY (FM/ZS) 19.560
 INITIAL NUMBER OF NEUTRONS 100
 TIMESTEP (ZS) 0.500

TIME	LIVE NUCLEI	LIVE NEUTRONS	NEUTS CREATED	FISSIONS	ESC NEUTS	COLLISIONS TO PROCESS	REJECTED COLLISIONS	SCATT NEUTS
0.0	9201234	100	100	0	0	0	0	0
5.0	9201234	100	100	0	0	0	0	0
10.0	9201234	100	100	0	0	0	0	0
15.0	9201234	100	100	0	0	0	0	0
20.0	9201234	100	100	0	0	0	0	0
25.0	9201234	100	100	0	0	0	0	1
30.0	9201234	100	100	0	0	0	0	1
35.0	9201234	100	100	0	0	0	0	2
40.0	9201234	100	100	0	0	0	0	4
45.0	9201234	100	100	0	0	0	0	4
50.0	9201234	100	100	0	0	0	0	4
55.0	9201234	100	100	0	0	0	0	4
60.0	9201233	109	110	1	0	0	0	4
65.0	9201233	109	110	1	0	0	0	4
70.0	9201233	109	110	1	0	0	0	4
75.0	9201233	109	110	1	0	0	0	4
80.0	9201233	109	110	1	0	0	0	4
85.0	9201233	109	110	1	0	0	0	5
90.0	9201233	109	110	1	0	0	0	5
95.0	9201233	109	110	1	0	0	0	5
100.0	9201233	109	110	1	0	0	0	5

(lines deleted)

24930.0	9182971	61543	182730	18263	102924	10	3	67195
24935.0	9182949	61597	182950	18285	103068	17	3	67287
24940.0	9182927	61667	183170	18307	103196	13	3	67377
24945.0	9182901	61783	183430	18333	103314	8	3	67471
24950.0	9182877	61863	183670	18357	103450	14	3	67552
24955.0	9182840	62060	184040	18394	103586	11	3	67634
24960.0	9182811	62181	184330	18423	103726	10	3	67722
24965.0	9182781	62318	184630	18453	103859	15	3	67810
24970.0	9182762	62335	184820	18472	104013	10	3	67912
24975.0	9182746	62341	184980	18488	104151	10	3	67996
24980.0	9182719	62436	185250	18515	104299	13	3	68088
24985.0	9182689	62569	185550	18545	104436	11	3	68188
24990.0	9182656	62755	185880	18578	104547	9	3	68267
24995.0	9182628	62864	186160	18606	104690	13	3	68359
25000.0	9182600	62967	186440	18634	104839	14	3	68465

FISSIONRAND

Operates exactly as FISSIIONCORE but with the addition of parameter IDUM in the VALUES file:

$d, R, \sigma_f, \sigma_s, v_{neut}$
 $v, v_o, \Delta t, t_{max}, IDUM$

IDUM seeds the Press et. al. RAN2 random number generator, and should be set to a negative integer; a different value will lead to a different sequence of random numbers.