FISPACT-II Advanced exercises

Thomas Stainer, Mark Gilbert, Greg Bailey,
Olga Vilkhivskaya, Andrew Davis
UKAEA

November 23, 2020

1 Exercise readgg

FISPACT-II can also produce a gamma spectrum at each time step. This is done by default with the 24 gamma group, alternatively this can be done with the 22 group using the keyword *GROUP* with option 1.

In the working directory for this example, *'advanced/readgg'*, an input file exists *inventory.i*. Can you run this example and examine the *inventory.out* output file and find the gamma spectrum in the file?

In most practical cases, these group structures can be limiting, and it is therefore useful to define our own bespoke gamma energy boundaries. This can be done with the *READGG* keyword. This should be placed in the control phase of the input file. If you examine the input file, this should be currently commented out. If you uncomment it and run again, the bin boundaries should have changed. Are you able to figure out what the new bin boundaries are?

Note: We have used equal bin sizes, but this is not required. However, the first bin boundary must be non-zero.

Whilst this information is not yet in the JSON output file, we can use the *TAB4* keyword to output it directly to a simpler output file. Are you able to parse this data and make a plot as shown in figure 1. This shows the spectrum at the first cooling time (36 seconds) after irradiation.

2 Exercise multiflux

This exercise examines the ability of FISPACT-II to perform sequential irradiations under different particle spectra, using a series of chained simulations. Thus, the output from one simulation will be used as the input for the next simulation. For the sake of simplicity, we will only use two different energy

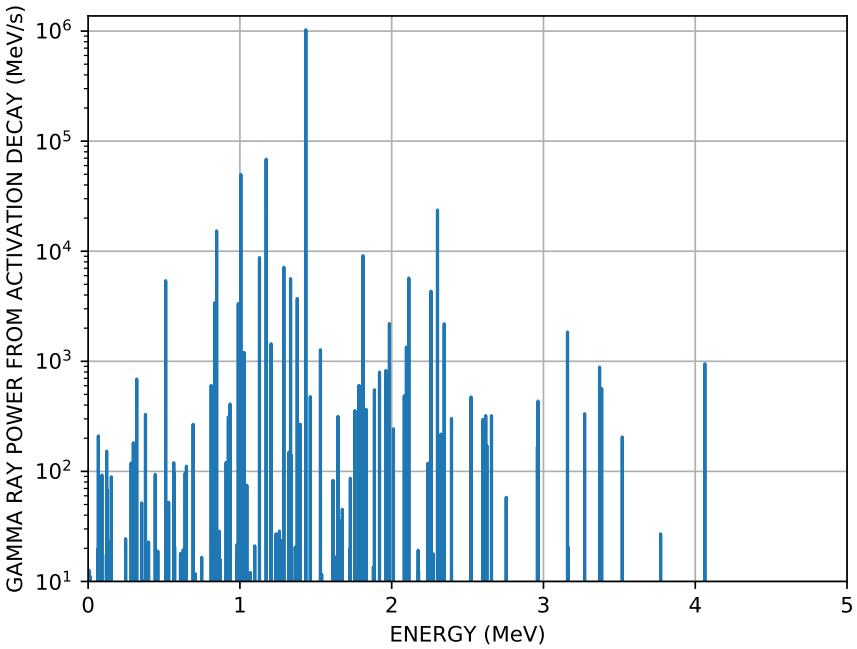

Figure 1: The 20,000 bin gamma spectrum at the first cooling time.

spectra, but in reality this principle can be applied to any number of iterations, whereby the spectra is changing.

In the working directory for this example, *'advanced/mutliflux'*, two fluxes files can be found, *fluxes.1* and *fluxes.2*, representing the spectra at two different times during an irradiation. The spectra are based on a deuterium-tritium (D-T) reaction inside a tokamak - notice the 14 MeV peak. We first plot the two spectra for comparison, as shown in figure 2. Are you able to reproduce this plot?

We will irradiate 1 gram of Tungsten following an irradiation of 1.1×10^{14} n cm^{-2} s^{-1} for 5 minutes (time 1) with the first spectrum, followed by an irradiation of 1.3×10^{14} n cm^{-2} s^{-1} for 10 minutes (time 2), followed by a period of cooling for 10 years in 1 year increments. In order to do this we need to run two separate simulations, the first input file is given in the exercise directory, aptly named *first.i*. The second input file does not yet exist as this depends on the output of the first run. There are two approaches to doing this, by hand, or automate it using a script. We will first do the former to show how

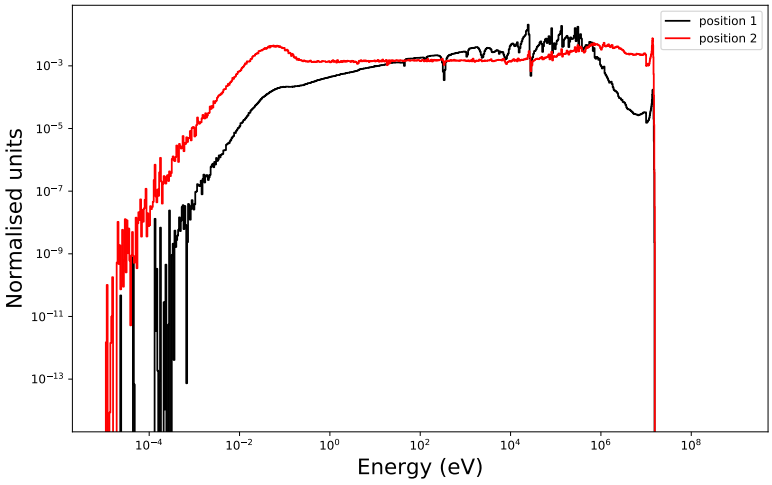

Figure 2: The incident particle spectra as a function of energy in 709 group, for D-T reaction inside a tokamak at time 1 (black) and time 2 (red). Spectra are normalised to unity.

inconvenient it is, then automate this process using a python script.

2.1 By hand approach

We will leverage the *TAB1* keyword to enable an additional output file to be written in tabular format. This keyword should be placed in the initialisation phase of the input file to enable all outputs to be written to the file. The first input file *first.i* already has this keyword enabled. After running this simulation a *first.tab1* file should be produced, which should look like figure 3. We should now be able to construct a second input file, *second.i*, which uses the nuclides in the final inventory from the output of the first, but now using the second spectra as our fluxes source. Are you able to do this?

Hint: The third column in the atoms inventory, which can be directly plugged into the input file via use of the *FUEL* keyword. Note that we must remove the whitespace in nuclide names in order for FISPACT-II to correctly process them.

If you did manage to do this by hand, did you find this an easy process? We will try the automated approach now and compare the results.

```
  TIME  0.000ps       INTERVAL   1  INTERVAL TIME      0.000E+00 SECS
  W 180     741800   3.93086E+21   1.17458E+00
  W 182     741820   8.68065E+23   2.62270E+02
  W 183     741830   4.68755E+23   1.42406E+02
  W 184     741840   1.00368E+24   3.06582E+02
  W 186     741860   9.31287E+23   2.87567E+02
  TIME  5.000 m       INTERVAL   2  INTERVAL TIME      5.000E+00 MINS
  H   1      10010   2.61857E+11   4.38226E-13
  H   2      10020   8.28090E+09   2.76954E-14
  H   3      10030   4.67742E+08   2.34258E-15
  He  3      20030   1.24984E+02   6.25951E-22
  He  4      20040   1.30286E+11   8.65941E-13
  Hf176     721760   1.64155E+07   4.79591E-15
....
```

Figure 3: The expected output TAB1 file produced after running the first example.

2.2 Scripted approach

Whilst we could again use the TAB1 and create a parser for this data, we instead make use of the *pypact* package, which has a validated parser built into it, making it easy to get the values from the standard output file. Additionally, *pypact*, includes functionality to make input files. A script is attached *makesecond.py*, which produces the required input file for the second run. To run this simply do the following.

```
python3 makesecond.py
```

Upon running this, you should have a second input file, *second.i*, which if we use as we did with the hand made approach we should get exactly the same output. Did you find this a simpler process?

Note that while the hand made approach worked, you can quickly see how difficult this would become for more complex scenarios and large inventories. Therefore, for all multi-flux cases we recommend using a scripting approach. In fact, we could script the whole process. Do you think you could write such a script based on what we've just done?

3 Exercise JSON

Whilst we have seen the power of the *GRAPH* and *NUCGRAPH*, this can sometimes be limiting, particularly if you want to create bespoke plots. Additionally, they only work with GNUPLOT. This exercise examines the how to *JSON* keyword to produce a json file output to enable you to easily parse the

output data for custom plotting. JSON is a file format well supported by many programming languages, in this exercise we use python to explore how to use this, but it should be transferable to other programming languages.

In the working directory for this example, *'advanced/json'*, an input file exists *inventory.i*, notice that it should have the *JSON* keyword in the control phase of the file. Can you run this example and examine the *inventory.json* output file?
Three python scripts exist which consume this output file and produce three plots.

```
plotpieinitial.py       plotpiefinal.py         plotnuclideactivity.py
```

Can you run them all and examine the associated plots produced (pdfs)? Are you able to understand what they represent?
They should look like figure 4. Are you able to understand how these scripts work?

Can you change the *plotnuclideactivity.py* script to plot the heat instead of activity?

4 Exercise pypact

This exercise introduces the user to the open source Python3 tool, known as *pypact*. In addition to the JSON output, *pypact* can parse the existing standard FISPACT-II output file (*.out) and will therefore work for versions prior to version 4.0. It is a Python3 tool, but is easy to install if you are familiar with the python package manager, pip. If you are not Python literate, you are welcome to ignore this exercise.

This exercise illustrates that the same analysis can be performed using either the JSON output or the standard output and parsing it with *pypact*. Both output files will be used in this exercise and will produce the same chart as that which was produced in JSON exercise, which was made with gnuplot. It shows an alternative and more bespoke approach to performing analysis.
This exercise should contain one input file along with the files file and the fluxes file from the JSON exercise. In addition to these files there should exist one python file, *inventory.py*. The input file should be the same as the previous input file from the previous exercise.

```
[1206][user1@fispact:~/exercises/advanced/pypact] ls
       inventory.i        inventory.py       fluxes        files
```

Again run the inventory input file and check that a valid JSON file and out file is written, namely *inventory.json* and *inventory.out*. We will now use this to perform our analysis. Run the python3 script *inventory.py*.

```
[1206][user1@fispact:~/exercises/advanced/pypact] python3 inventory.py
```

After running the script, two identical plots should be dumped to screen. If not it may be that *matplotlib* was not installed. The two plots should match that of figure 6, one was produced using the JSON output data, while the other was produced from the standard .out file.

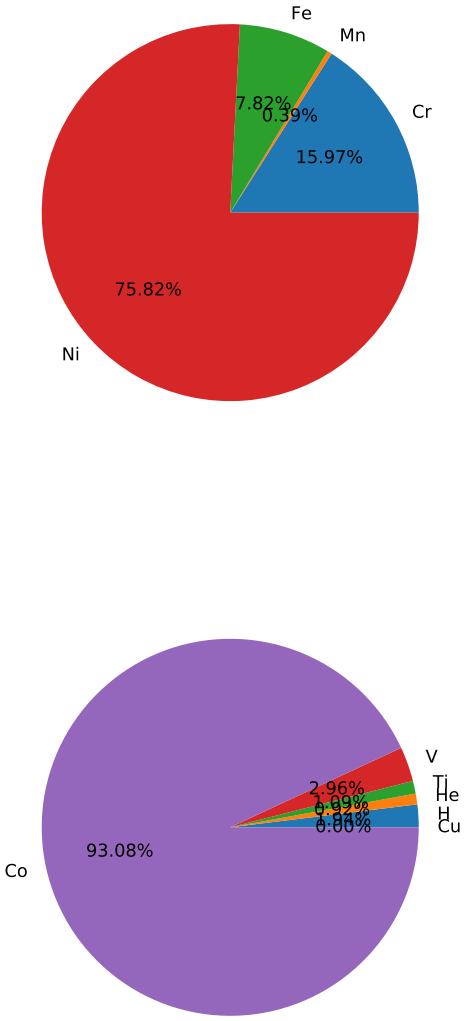

Figure 4: A pie chart showing the elemental composition specified in the initial (top) and final (bottom) inventories.

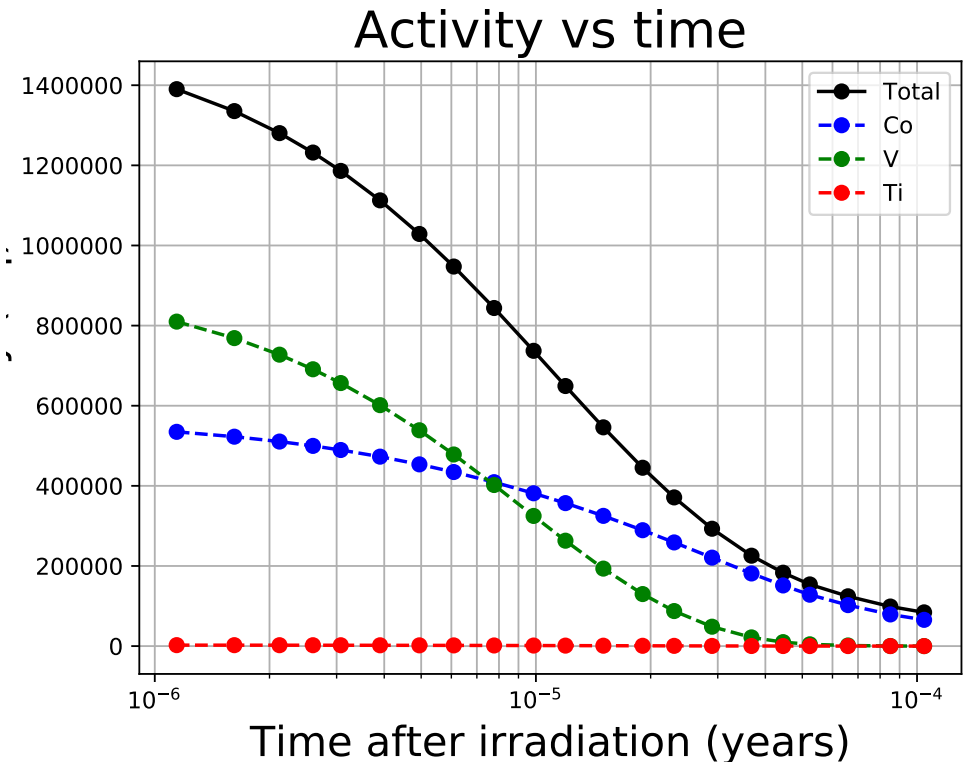

Figure 5: The activity of Co (blue), V (green), and Ti (red) as a function of time after irradiation. The total activity is shown in black.

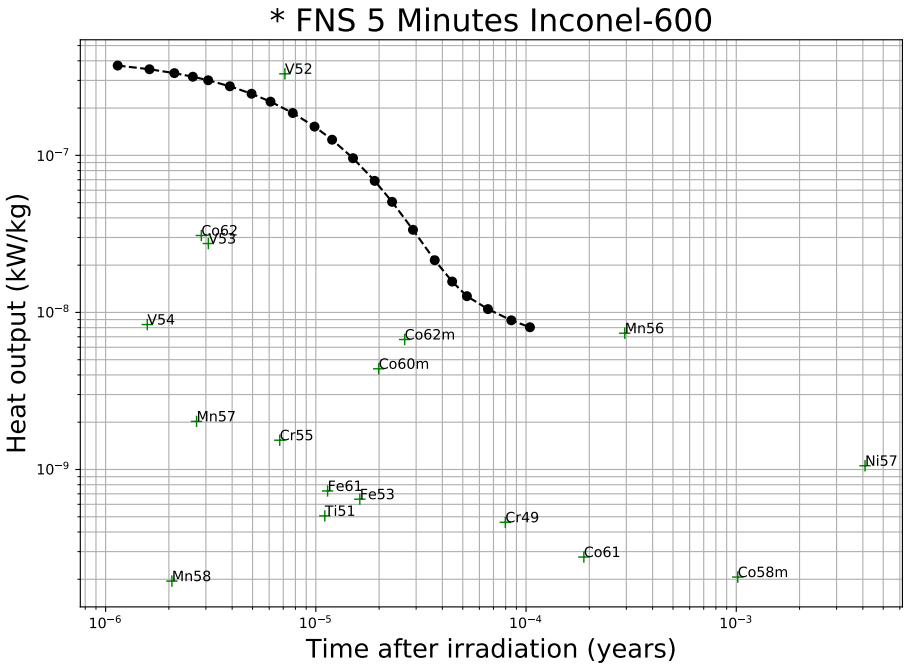

Figure 6: The total heat output for FNS Inconel 600 run as a function of cooling time. Dominate nuclide half lives are plotted with green crosses.