

ORIGEN decay calculations basics

SCALE 6.3

ORNL is managed by UT-Battelle, LLC for the US Department of Energy



U.S. DEPARTMENT OF
ENERGY

Learning goals



Gain familiarity with the basic ORIGEN input components and syntax, with a focus on decay calculation input



Gain experience with creating and running inputs for simple ORIGEN decay calculations

ORIGEN input syntax

- Input information is located between **=origen** and **end** in an input file
- Syntax format is SCALE Object Notation (**SON**) and includes
 - **arrays**, denoted by “=[]” (e.g., `t =[1 3 10 30 100]`)
 - **blocks**, denoted by “={ }” (e.g., `lib{file="end7dec}`)
 - **key/value pairs** (e.g., `print_xs=yes`)
- Backwards-compatible, can also read SCALE 6.1 FIDO syntax

```
=origen
0$$ a4 33 a11 71 e t
ce14x14
3$$ 33 a3 1 200 a16 2 a33 47 e t
35$$ 0 t
...
```

Further details:
Section 5.1.5.2 of
the [SCALE manual](#)

Overview

```
'SCALE comment
=origen

    % ORIGIN comment %

solver{ ... }
options{ ... }

case(A) {
    time=[31 365] % days
    ...
}

case(B) {
    ...
}

% more cases?

end
```

- Hierarchical structure
- “Root” level can have
 - **one** of: **solver**, **options**, **bounds**, **build_lib**
 - **sequence** of: **case**
- Main problem is specified within case
 - can have optional id, **case(ID)**
- How to include comments
 - inside ORIGIN, SON uses %
(anywhere)
 - outside ORIGIN, SCALE uses '
(beginning of line only)

Input component: **case**

```
=origen
...
case(ID) {
    title="my title"

    lib{ ... }
    mat{ ... }
    time{ ... }
    flux{ ... } % or power{ ... }

    print{ ... }
    save{ ... }

}
...
end
```

- **setup**
 - **lib** – load library data
 - **mat** – define initial material composition
- **operating history**
 - **time** – define time steps
 - **flux/power** – define flux or power level
- **archival**
 - **print** – results to output file
 - **save** – results to F71 binary file

Input component: **lib**

```
=origen
...
case(initial) {
    lib{ file="my.lib" pos=1 }
    mat{ ... }
    ...
}
case(continuation) {
    lib{ pos=2 }
    ...
}
...
end
```

- Library files can contain multiple (usually burnup-dependent) “positions”
- Position/data set is fixed for each case in input
- Default is **pos=1** (will always exist on any library)
- `file="end7dec"` is a decay only library

Input component: **mat**

```
% from iso
mat{
    iso=[ u235=1.0 u238=9.0 ] %id(sublib)=amount
    units=GRAMS               %units in iso array
}

% from iso with number density input
mat{
    iso=[ u235=1e-2 u238=1e-1 ] %id(sublib)=amount
    units=ATOMS-PER-BARN-CM      %units in iso array
    volume=200                  %cm^3
}

% from position on f71 file
mat{
    load{ file="origen.f71" pos=11 }
}

% from previous case
mat{
    previous=4 %step index from previous case
}
```

- Specify using **iso** array
 - list of **id=amount**
 - units for iso array:
 - GRAMS
 - MOLES (**default units**)
 - ATOMS-PER-BARN-CM (requires volume)
 - CURIES
- Load from a position on a provided binary concentrations file F71
- Load from a previous case in the input

Nuclide identifiers: format

Identifier Form	Comments	Examples <i>nuclide</i> → <i>input id</i>	
IZZZAAA I – <i>isomeric state</i> ZZZ – <i>atomic number</i> AAA – <i>mass number</i>	Standard numeric identifier with one optional digit of isomeric state, three digits of atomic number, three digit of mass number. Elements have mass number of 000.	²³⁵ U	→ 92235
		^{235m} U	→ 1092235
		¹³⁵ Xe	→ 54135
		¹ H	→ 1001
		¹⁰ B	→ 5010
		Fe	→ 54000
EAm E – <i>element symbol</i> A – <i>mass number</i> m – <i>metastable indicator</i>	Standard symbolic identifier with element symbol followed by mass number, followed by optional metastable indicator. Can include a dash between E and A (E-Am). Case insensitive.	²³⁵ U	→ u235
		^{235m} U	→ u235m
		¹³⁵ Xe	→ xe135
		¹ H	→ h1
		¹⁰ B	→ b10
		Fe	→ fe

- **EAm style** is preferred:
u235m instead of 1092235
- Is case-insensitive:
Pu239, pu239, PU239 are all the same
- can include a hyphen:
u-235, xe-135

NOTE: older modules (COUPLE,OPUS) without SON syntax option still use legacy id for backwards compatibility (ZZAAAI)
u235→922350, u235m → 922351

Nuclide identifiers: **sublibraries**

- Historically, ORIGEN has tracked nuclides based on their creation mechanism
 - e.g., Gd-155 as a fission product is different than Gd-155 in Gd_2O_3 fuel dopant
- ORIGEN library files (F33) contain this representation
 - 1/LT – light nuclide (activation) sublibrary
 - 2/AC – actinide sublibrary
 - 3/FP – fission product sublibrary
- To specify explicitly the sublib of a nuclide, use
`iso[ id(sublib)=amount ... ]`
example: `iso=[gd155(LT)=1 gd155(FP)=1]`
- NOTE:** 1/2/3 can be used instead of LT/AC/FP

1 gram of Gd-155 as a fission product +
1 gram of Gd-155 as a light nuclide
= 2 grams total!



Nuclide identifiers: sublibrary search

- When you do not specify a sublib, e.g.,

```
iso = [ gd155=1.0 ]
```

where does it go?

- If the *sublib* for a nuclide/element is not provided, then it is guessed in the following manner:

- if the nuclide is in fact an **element**, then it is placed in **sublib = 1/LT**
- if the atomic number $Z < 26$, an *attempt* is made to place it in **sublib=1/LT**
- otherwise ($Z \geq 26$ or attempt fails), **sublibs are searched in reverse order**, from 3/FP, to 2/AC, then 1/LT

- sublib* is used extensively throughout the ORIGEN family for **output**, e.g., actinides only (AC)



iso=[gd=1] → 1/LT
(and split into Gd isotopes based on natural abundances)



iso=[o16=1] → 1/LT (Z=8)
iso=[gd155=1] → 3/FP (Z=64)

Input component: **mat** revisited

```
mat{
  iso=[
    o16(LT)=1.34432e5
    u234=445
    u235=50000
    u236=230
    u238(AC)=949325
    pb208(LT)=.01
    xe135(FP)=1e-5
    am242m=1.0
    h(LT)=2e3
    2003(LT)=1.55
    94239=700
    1054135=6.0
  ]
  units=GRAMS
}
```

- Which sublibs are necessary?
 - must know which versions of nuclides exist
 - ORIGEN library (F33) dependent!
 - experience
- In the input to the left
 - o16(LT), 2003(LT)
not necessary to specify LT, $Z < 26$
 - **pb208(LT) is necessary**
default would be AC
 - xe135(FP)
not necessary to specify LT, $Z \geq 26$
 - h(LT)
not necessary to specify LT, is element

Input component: case

```
=origen
...
case(ID) {
    title="my title"

    lib{ ... }
    mat{ ... }
    time{ ... }
    flux{ ... } % or power{ ... }

    print{ ... }
    save{ ... }

}
...
end
```

- setup
 - lib – load transition matrix data from library
 - mat – define initial composition
- **Operating history**
 - **time** – define time steps
 - **flux/power** – define flux or power level
- Output
 - print – results to .out file
 - save – results to .f71 file

Input component: **time** 1/2

```
% simple decay case (two steps 0→31 and 31→365 days)
time = [ 31 365 ]

% flux irradiation (decay if flux=0)
time = [ 31 365 396 ]
flux = [ 2e14 1e14 0 ]

% power irradiation (decay if power=0)
time = [ 31 365 396 ]
power= [ 50 45 0 ] %50 MW, 45 MW, then decay

% changing units using time block
time{
    t = [ 5 15 300 ]
    units = HOURS
    % available units:
    % SECONDS, MINUTES, HOURS, DAYS, YEARS, CUSTOM
}

% custom units
time{
    t = [ 1 2 3 ]
    units = CUSTOM
    custom_name = "MONTH"
    custom_length = 2678400 %seconds per "MONTH"
}
```

- Operating history
 - **time** – define time steps
 - flux/power – define flux or power level
- Time can be defined via
 - a simple array []** of time steps with default units of days
 - OR**
 - a block** with a **t=[]** array and **units=**
 - SECONDS
 - MINUTES
 - HOURS
 - DAYS (**default units**)
 - YEARS
 - CUSTOM

Input component: **time** 2/2

```
% first case
case{
    mat{ ... }
    time=[ 1 10 25 50]
    flux=[ 4r1e14 ]
}
% continuation case
case{
    time{
        units=YEARS
        %default start=50 DAYS, how many years?
        %t=[50/365.+0.1 50/365.+0.3 ... ]
        %with start=0, timeline reset
        start=0
        t=[0.1 0.3 0.9 2.7]
    }
}
```

- Operating history
 - **time** – define time steps
 - flux/power – define flux or power level
- Time moves on from case to case
- Can be tedious when switching units
 - e.g., depletion in days to decay in years
- **start=** provides a way to set initial time

NOTE: ORIGEN's internal units are

- **seconds** for time
- **moles** for nuclide amounts

Input component: flux/power

```
% simple decay case (two steps 0→31 and 31→365 days)
time = [ 31 365 ]

% flux irradiation (decay if flux=0)
time = [ 31 365 396 ]
flux = [ 2e14 1e14 0 ]

% power irradiation (decay if power=0)
time = [ 31 365 396 ]
power = [ 50 45 0 ] %50 MW, 45 MW, then decay

% changing units using time block
time{
    t = [ 5 15 300 ]
    units = HOURS
    % available units:
    % SECONDS, MINUTES, HOURS, DAYS, YEARS, CUSTOM
}

% custom units
time{
    t = [ 1 2 3 ]
    units = CUSTOM
    custom_name = "MONTH"
    custom_length = 2678400 %seconds per "MONTH"
}
```

- Operating history
 - time – define time steps
 - **flux/power – define flux or power level**
- Either flux or power (not both)
- Total flux (neutrons/cm²/s) is required to solve the depletion equations
- Power (MW) can be provided
 - converted internally to flux
- Must have the same number of time and flux/power entries

NOTE: for fuel, it is convenient to specify 1E6 total grams of initial heavy metal, so power is MW/MTIHM

Input component: **case**

```
=origen
...
case(ID) {
    title="my title"

    lib{ ... }
    mat{ ... }
    time{ ... }
    flux{ ... } % or power{ ... }

    print{ ... }
    save{ ... }

}
...
end
```

- Setup
 - lib – load data from library
 - mat – define initial composition
- Operating history
 - time – define time steps
 - flux/power – define flux or power level
- **Output**
 - **print** – results to .out file
 - **save** – results to .f71 file

Input component: **print** 1/4

```
=origen
case{
  lib{ file="end7dec" }
  mat{ iso=[pu238=1.0] }
  time=[1 3 10]
}
end
```

```
-----
### Concentrations output options
.
      High-precision output:      no
      Relative cutoff:            yes
      Step for applying cutoff:    integral
.
      light els.  actinides  fis.prods.  total
      nuc   el   nuc   el   nuc   el   nuc   el
gram-atoms
grams
curies
watts
gamma watts
m3 air
m3 water
grams ppm
atoms ppm
atoms/barn-cm
-----
cutoff
1.000E-06
**
1.000E-06
1.000E-06
1.000E-06
1.000E-06
1.000E-06
1.000E-06
1.000E-06
1.000E-06
```

- **lib, mat, time** are all you need to perform a decay calculation
 - add flux/power for irradiation
 - need proper reaction library
- **print** gives you control over what goes to the output file
 - default (nuclide, grams, 1E-6 cutoff)
 - variations
 - nuclide vs. element
 - units
 - sublibs vs. total
 - gram atoms
 - minimum relative cutoffs

Input component: **print** 2/4

```
=origen
case{
  lib{ file="end7dec" }
  mat{ iso=[pu238=1.0] }
  time=[1 3 10]
  print{
    nuc{
      units=[WATTS GRAMS]
      sublibs=[LT FP]
      total=no
    }
    cutoffs=[ WATTS=1e-3 GRAMS=1e-2 ]
  }
}
end
```

Concentrations output options

```
.
                                High-precision output:      no
                                Relative cutoff:             yes
                                Step for applying cutoff:     integral
.
                                light els.   actinides   fis.prods.   total
                                nuc   el     nuc   el     nuc   el     nuc   el
gram-atoms                      1.000E-06
grams                           **                **                1.000E-02
curies                          1.000E-06
watts                           **                **                1.000E-03
gamma watts                     1.000E-06
m3 air                          1.000E-06
m3 water                        1.000E-06
grams ppm                       1.000E-06
atoms ppm                       1.000E-06
atoms/barn-cm                   1.000E-06
```

- **print** contains **nuc**, **ele**, **cutoffs**
- **nuc/ele** blocks
 - control nuclide and element printout
 - same inside
 - **units**=[] list
 - **sublibs**=[] list
 - **total**=yes/no
- can have multiple **nuc{ }**, **ele{ }**
- **cutoffs**
 - has special **ALL** to change all units
cutoffs=[**ALL**=1.0]

Input component: **print** 3/4

```
% print each nuclide (total across all sublibs) in grams
print{
    nuc{ total=yes units=GRAMS }
}

% print each element (total across all sublibs)
% in moles, grams, and curies with cutoffs of 1%
print{
    ele{ total=yes units=[MOLES GRAMS CURIES] }
    cutoffs[ ALL=1.0 ]
}

% print decay heat and mass (by element)
% of fission products and actinides only
print{
    ele{ sublibs=[AC FP] units=[GRAMS WATTS] }
}

% change cutoff to absolute curies by element,
% in step of interest (7), but print GRAMS
print{
    cutoff_step = 7 % default -1 for average
    rel_cutoff = no % default is yes for cutoff in percent
                    % only print above 1e-3 curies
    cutoffs[ CURIES=1e-3 ] % default is 1e-6 percent
    nuc{
        total=yes
        units=GRAMS
    }
}
```

- **cutoff** control
 - **cutoff_step**
 - apply cutoff criteria to nuclides at a particular step instead of the average over the case
 - **rel_cutoff=no**
 - use absolute values

Input component: **print** 4/4

```
% defaults special printing options
print{
    absfrac_sublib=NONE %print absorption fractions for
                        % a specific sublib (LT,AC,FP)
                        % all sublibs (ALL)
                        % or no sublibs (NONE)

    absfrac_step = 7    %if absfrac active, step to print
                        % default is last step

    fisrate = NONE      %print fission rates (default NONE)
                        %absolute (ABS) or relative (REL)

    kinf = no           %print fission/absorption (yes/no)
}
```

- **special output**

- **absfrac_sublib**=
NONE/LT/AC/FP/ALL
 - print absorption fractions for nuclides in a given sublib, or
 - all sublibs
- **absfrac_step**
 - step index to print absfrac
 - default last step
- **fisrate**=NONE/ABS/REL
 - absolute fission rates (ABS) or
 - relative fission rates (REL)
- **kinf** = no/yes
 - infinite multiplication factor
 - fission rate / absorption rate

Input component: **case**

```
=origen
...
case(ID) {
    title="my title"

    lib{ ... }
    mat{ ... }
    time{ ... }
    flux{ ... } % or power{ ... }

    print{ ... }
    save{ ... }

}
...
end
```

- Setup
 - lib – load data from library
 - mat – define initial composition
- Operating history
 - time – define time steps
 - flux/power – define flux or power level
- **Output**
 - print – results to .out file
 - **save – results to .f71 file**

Input component: **save** 1/2

```
case{
  mat{ ... }
  time=[ 1 10 100 1000 ] % 4 steps
  save{
    file="my.f71"           % file name
    steps=[0 2 4]           % save initial (0) isotopics
                           % end-of-step 2 (10 days)
                           % end-of-step 4 (1000 days)
  }
}
```

```
=origen
case{
  % use ENDF/VII-based decay library
  lib{ file="end7dec" }
  % create a material with 1 gram U-238
  mat{
    units=grams
    iso=[u238=1.0]
  }
  time=[20L 1.0 1e9] %default units are days

  % save to f71
  save{
    file="u238.f71"
    steps=ALL
  } %save=yes is shortcut for ALL, ${OUTPUTBASENAME}.f71
}
end
```

- **save** block requires
 - file name, `file="xyz.f71"`
 - array of step indices, `steps=[]`
- Each value in the time array indicates a "step" from the previous time
 - the first step is from 0 days to 1 day
 - second is from 1 day to 10 days
 - etc.
- Using log interpolation $\mathbb{L}$ and linear interpolation $\mathbb{I}$ is convenient for time and steps, respectively
 - $x\mathbb{L} \ A \ B$ logarithmic spacing between A and B
 - total number of steps is $x+2$

Input component: **save** 2/2

```
=origen
case (decay) {
  time{
    units=YEARS
    start=0
    t=[0.1 0.3 0.9 1 2 3 4 5]
  }
  save{ file="discharge" steps=[0 LAST] }
  %only save begin and end in this case
}
end

'must copy back f71 explicitly because doesn't match
BASENAME.f71
=shell
cp discharge ${INPDIR}/discharge
end
```

- F71 file is ORIGIN's "results" file
 - binary ☹️
 - can be post-processed with OPUS ☺️
 - can initialize ORIGIN material with it ☺️
 - can be viewed in Fulcrum ☺️☺️
 - also called “concentrations” file
 - can contain concentrations *and* spectra
- **SCALE rule:** all files with extension “.f71” or “.f33” will be copied from the TMPDIR back to the user directory
- To avoid automatic copy of either type of file, file names ft71f001 and ft33f001 can be used (legacy SCALE names)
- Shortcuts
 - **LAST** for last step index
 - **steps=ALL** for everything

Input component: **solver** (1/3)

```
=origen
solver{
  type=MATREX
  opt{
    %number of expansion terms
    terms=21
    %maximum number of short-lived
    %precursors in long-lived chain
    maxp=100
    %number of time step divisions
    substeps=1
  }
}
case{
  ...
}
end
```

- **MATREX**

- default solver
- in ORIGEN since the beginning (50 years)
- uses a hybrid approach to solve systems with large number of nuclides
 - matrix exponential for “long-lived”
 - linear chains for “short-lived”
 - definition of “long-lived” vs. “short-lived” depends on time step length

Input component: solver (2/3)

```
=origen
solver{
  type=CRAM
  opt{
    %solver order
    %(runtime/accuracy tradeoff)
    order=16
    %number of time step divisions
    substeps=1
  }
}
case{
  ...
}
end
```

- CRAM
 - **C**hebyshev **R**ational **A**pproximation **M**ethod
 - faster, more accurate, higher stability
 - why not default?
 - backwards compatibility with SCALE 6.1
 - no way to specify solver in FIDO input
 - did not want solver switch between FIDO vs. SON formats
 - will be default for SCALE 7

Input component: **solver** (3/3)

```
=origen
solver{
  ...
  opt{
    substeps=4
  }
}
case{
  time=[1 10 20 50 100]
  ...
}
end
```

cost for N substeps

MATREX: factor N runtime increase

CRAM: <factor N runtime increase

*because of reuse of a matrix
factorization*

- **substeps=N** divides each step into N equal substeps
- Does not introduce additional output
- Easy time grid refinement
 - applies to every step in every case
 - e.g., substeps=4
 - **default:** 0→1 day in a single step
 - substeps=4: 0→0.25→0.5→0.75→1 day in 4 steps
- Results with MATREX vs. CRAM will differ, e.g., in 3rd or 4th digit
 - increasing substeps will eventually lead to solvers giving the *exact* same answer

Input component: options

```
=origen
options{
    %print cross sections
    print_xs=no
    %digits=6 is high-precision
    digits=4
    %yes is 200 MeV per fission
    fixed_fission_energy=no
}
case{...}
end
```

- Detailed cross section printout for each case
`print_xs=yes`
- Increase number of digits shown in output
`digits=6`
- Switch to 200 MeV energy release per fission (i.e., 0 MeV per capture)
 - convenient for comparisons to other codes
 - useful when evaluating the effect of captures and nuclide-dependent energy releases

Summary of simple decay input

- Hierarchical input based on SON syntax
- Main input is a sequence of **case** blocks
- In each **case**
 - setup
 - **lib** – load data from library
 - **mat** – define initial composition
 - decay history
 - **time** – define time steps
 - archival
 - **print** – results to output file
 - **save** – results to F71 binary file
- Additional options and solver blocks available

Further details:

Section 5.1.5.2 of
the [SCALE manual](#)

ORIGEN decay with F71 input



Overview

- **Goal:** perform ORIGIN decay calculations using as input existing F71 file
 - also called ORIGIN *restart* calculations
- The “F71” file is the key to transferring nuclide inventories from one part of SCALE to another
- What is needed
 - name and location of the F71 file
 - position (number) on the F71 file to use as input

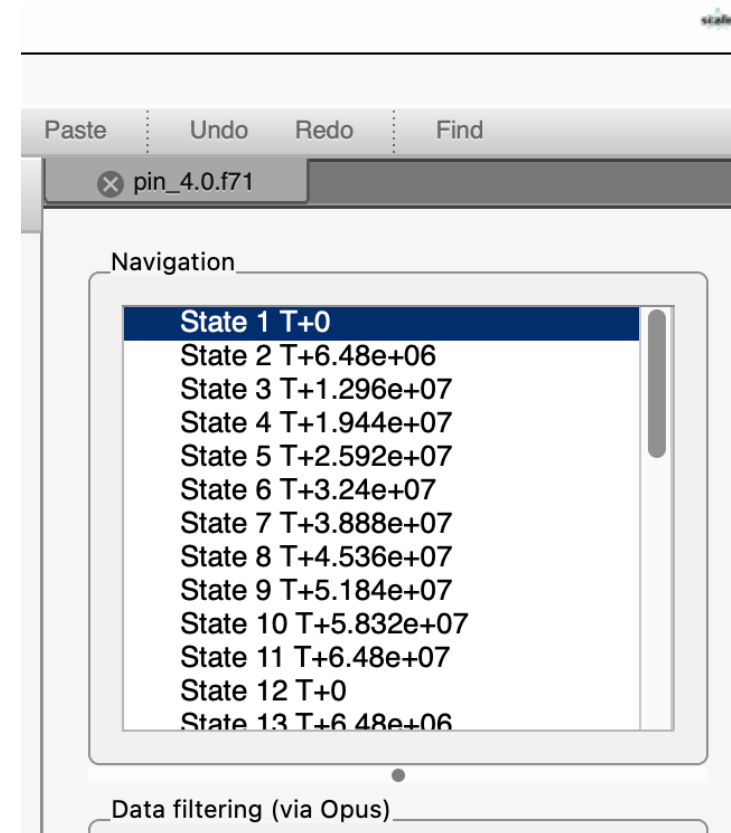
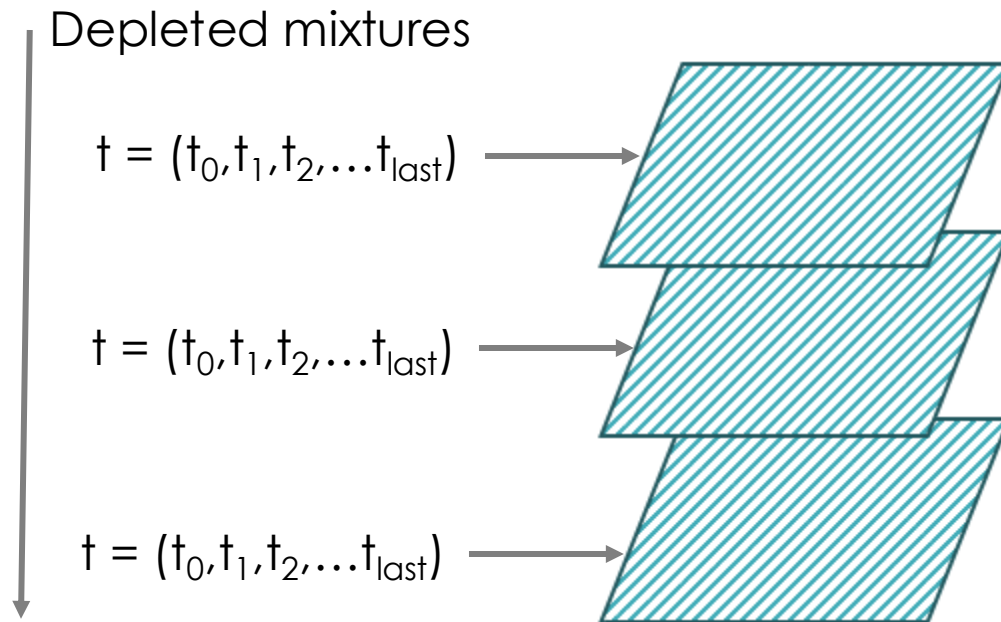
```
mat{ load{ file="mydata.f71" pos=13 }}
```
- The F71 file can originate from
 - depletion calculations with TRITON
 - depletion calculations with Polaris
 - depletion calculations with ORIGAMI
 - other ORIGIN decay calculations

Why use ORIGEN restart calculations?

- ORIGEN can use as input for the **mat** card the nuclide inventory (all nuclides in the ORIGEN library) corresponding to a given position in an available F71 file
- Potential applications
 - calculate nuclide inventory, decay heat, neutron emission, and gamma emission at desired cooling times for a desired assembly burnup (burnup indicated by position on F71 file) for a fuel assembly already simulated with TRITON or Polaris over a burnup range from 0 (fresh fuel) to 80 GWd/t
 - calculate nuclide inventory, decay heat, neutron emission, and gamma emission at desired cooling times for a specific axial plane in a fuel assembly already simulated with ORIGAMI
- You can always “convert” the nuclide inventory at the end of irradiation to decay heat, for example, with OPUS or F71 Fulcrum viewer, but without an ORIGEN decay calculation, it must be interpreted as decay heat at cooling time=0

F71 structure for files generated with TRITON

- Stores nuclide concentration for **every depleted mixture** (1,2,...,N) in the problem and for **each timestep**, plus a final set for the sum of all depletion materials
- Time corresponding to each time dump can be identified if opening the F71 file with Fulcrum



Example of F71 file information (from TRITON .out file)

File ft71f001 contains origen/opus-formatted binary data for

- each of the 1 depletion materials,
- a set for the sum of all depletion materials,
- a set for the sum of selected materials (from optional opus block),
- a set for the sum of all fuel materials

for all time steps e.g. 21 always.

Isotopic data locations are listed according to the following table.

Position	Time Step	Cycle Time (d)	Cumulative Time (y)	
1	0	0.0000e+00	0.0000e+00	Depletion material no. 2
2	1	2.6042e+00	7.1298e-03	
3	2	1.0417e+01	3.5649e-02	
4	3	1.3021e+01	7.1298e-02	
5	4	2.6042e+01	1.4260e-01	
6	5	5.2083e+01	2.8519e-01	
7	6	5.2083e+01	4.2779e-01	
...				
22	0	0.0000e+00	0.0000e+00	Weighted sum of concentrations of all depleted materials
...				
43	0	0.0000e+00	0.0000e+00	Weighted sum of concentrations for selected materials

Example of ORIGEN decay with F71 input

```
=shell
  cp "${INPDIR}/my_restart.f71" "${TMPDIR}/"
end

end
=origen
case{
  lib{file="end7dec"}
  mat{load {file=my_restart.f71 pos=7}}
  time{ t=[ 8L 1 100] units=years}
  print{ nuc{ units=[grams watts] } }
  save{ file="decay_result.f71" steps=all }
}
end
```

- Use the **shell** sequence to copy the available FT71 file from a specific location (here, user directory) to the SCALE temporary working directory
- ORIGEN can then find this file to load

Workshop:

ORIGEN simple decay calculations



Problem 1: Activity of potassium in a banana

What is the activity of the potassium (K) in a banana?

- Input
 - assume 440 mg K in a banana
 - consider 1 day decay time
- Required results
 - activity in Curies
- Strategy
 - use ORIGEN with the “end7dec” decay library

Problem 2: Nuclide inventory and decay heat for PWR spent fuel

Calculate nuclide concentrations and decay heat for a generic PWR assembly as function of burnup (30 and 60 GWd/t) and cooling time (0 and 100 years)

- Input
 - F71 file provided from previous TRITON depletion calculation (filename “w17_triton.f71”)
 - the positions on the F71 file that correspond to the two desired burnups are 14 and 20
- Required results
 - U-235 and Pu-239 (grams per tU) at 100 years cooling time and for 30 GWd/t and 60 GWd/t burnups
 - total decay heat (Watts per tU) at 0 and 100 years cooling time and for 30 GWd/t and 60 GWd/t burnups
 - plot comparison for total decay heat for the two burnups, as function of cooling time from 0 to 100 years
- Strategy:
 - use ORIGEN restart calculation with “end7dec” library and F71 “mat” input
 - use logarithmic time steps for ORIGEN decay
 - save f71 file for ORIGEN decay and use it with OPUS to plot neutron spectra and gamma spectra

Problem 3: Decay heat for nuclear calorimetry

How much heat per gram of material is generated as a function of cooling time (0 to 20 years) for typical commercial-grade fuel?

- Input
 - F71 file provided from previous TRITON depletion calculation (filename “w17_bu45.f71”)
 - select position 15 on the F71 file (corresponds to 45 GWd/t burnup)
 - note that the mass basis for data in provided F71 file is initial MTU
- Required results
 - total decay heat (W/gU initial) at cooling times of 0, 10, and 20 years
- Strategy
 - use ORIGEN restart calculation with “end7dec” library and F71 “mat” input
 - calculate total heat, not just gamma heat (total heat is “watts”, gamma heat only is “g-watts”)
- **Relevance** for safeguards applications: nuclear calorimetry decay corrections
 - Pu-240 fraction is measured by nuclear calorimetry

Solution



Problem 1: Activity of potassium in a banana

ANSWER: 0.4 nCi (or 15 Bq)

```
=origen
case{
  lib{ file="end7dec" }
  mat{
    units=grams
    iso=[K=0.440]
  }
  time=[1]
  print{
    ele{ units=[curies] }
  }
}
end
```

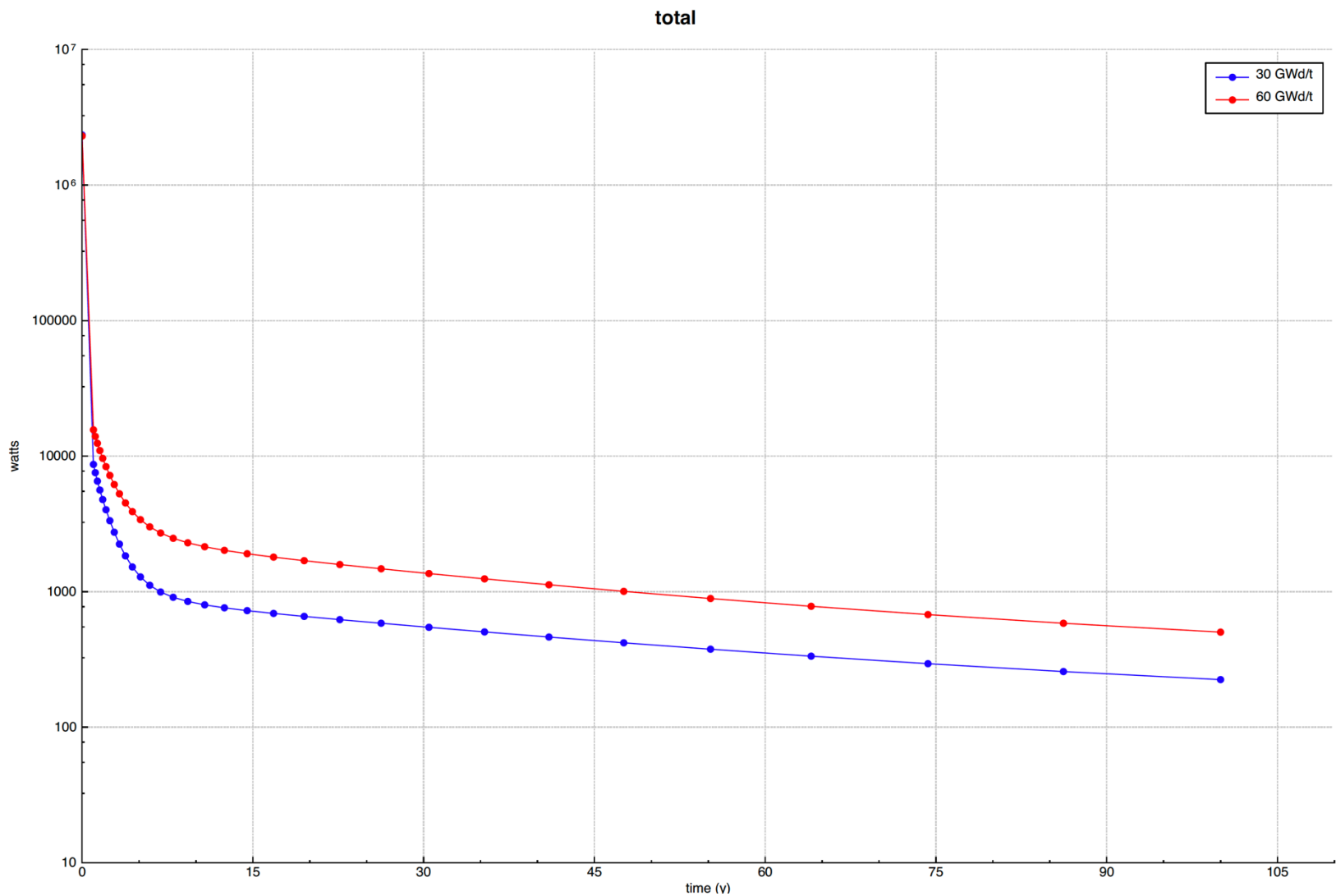
Note: K-40 is a radioactive isotope with half-life of 1.25 billion years; ~120 ppm of natural K

Problem 2: Nuclide inventory and decay heat for PWR spent fuel (1/2)

at 100 yr cooling	30 GWd/t burnup	60 GWd/t burnup
U-235 (g/MTIHM)	1.151e+04	2.605e+03
Pu-239 (g/MTIHM)	5.293e+03	5.486e+03

Decay heat (W/MTIHM)	30 GWd/t burnup	60 GWd/t burnup
discharge	2.342e+06	2.300e+06
100 yr cooling	2.244e+02	5.028e+02

Problem 2: Nuclide inventory and decay heat for PWR spent fuel (1/2)



Decay heat plot comparison

Problem 3: Decay heat for nuclear calorimetry

Decay heat per gram of initial U

ANSWER:

1.840 W/g at $t=0$

0.002 W/g at $t=10y$

0.001 W/g at $t=20y$