

LA-UR-26-27244

Approved for public release; distribution is unlimited.

Title: A New MCTAL Python Package in π mcnp
Author(s): Lively, Michael Aaron
Intended for: Meeting: 2026 MCNP User Symposium, 2026-08-24 - 2026-08-26 (Los Alamos, New Mexico, UNITED STATES)
Issued: 2026-08-12



Los Alamos National Laboratory, an affirmative action/equal opportunity employer, is operated by Triad National Security, LLC for the National Nuclear Security Administration of U.S. Department of Energy under contract 89233218CNA000001. By approving this article, the publisher recognizes that the U.S. Government retains nonexclusive, royalty-free license to publish or reproduce the published form of this contribution, or to allow others to do so, for U.S. Government purposes. Los Alamos National Laboratory requests that the publisher identify this article as work performed under the auspices of the U.S. Department of Energy. Los Alamos National Laboratory strongly supports academic freedom and a researcher's right to publish; as an institution, however, the Laboratory does not endorse the viewpoint of a publication or guarantee its technical correctness.



A New MCTAL Python Package in πmcnp

Michael A. Lively
Monte Carlo Codes Group (XCP-3), LANL

2026 MCNP User Symposium
24-26 August 2026

LA-UR-26-xxxxx
Approved for public release; distribution is unlimited.

Overview

- A new Python package has been developed for reading and analyzing data contained in MCTAL files written by the MCNP code.
- Notable improvements include:
 - All-Python design (no difficulties compiling and building wheels from C++).
 - Support for TMESH data.
 - Support for GENXS data.
 - Access to all metadata rather than a subset.
 - Direct access to underlying data arrays.
 - Error messages that are occasionally helpful.
- We will look at these improvements and some usage examples.
 - This talk is not documentation or a user guide, only a preview of new capabilities.
- Suggestions for further development are welcomed.

All-Python code design

- This is the general design principle of πmcnp .
- What does this mean?
 - Easy to develop.
 - Our team only needs to develop in one language, not two.
 - Easy to install.
 - If you have working Python, you can have working `mcnp.mctal`.
 - Easy to use.
 - `from mcnp import mctal`
 - All classes have comprehensive docstrings accessible in the interpreter with `help()`.

```
>>> from mcnp.mctal import Mctal
>>> help(Mctal.from_file)
Help on method from_file in module mcnp.mctal.mctal:

from_file(file_name: pathlib.Path | str) -> Self
class method of mcnp.mctal.mctal.Mctal
    Construct a MCTAL instance from a file.

Parameters
-----
file_name: Path | str
    The name of or path to the file to read from.

Returns
-----
Mctal
    The constructed MCTAL instance.
```

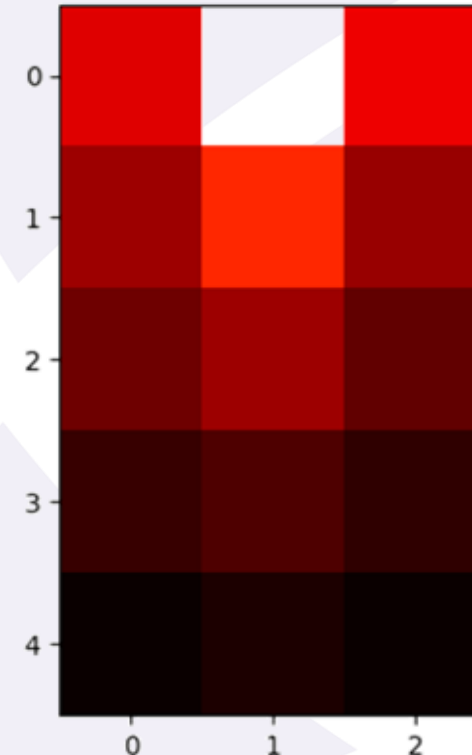
Support for TMESH data

```
# Trying to read TMESH data with MCNPTools:
from mcnp.tools import Mctal
m = Mctal("tmesh_data.mct1")
t = m.GetTally(1)
print(t.GetFBins())
(-25.0, 0.0, 25.0, -15.0, -5.0, 5.0, 15.0, 0.0, 5.0, 10.0,
15.0, 20.0, 25.0)
```

Unless the user already knows the tally shape, it is not obvious how to access the mesh tally data.

```
# Reading TMESH data with mcnp.mctal:
from mcnp.mctal import Mctal
from mcnp.mctal.enums import TallyMeshBinType
m = Mctal.from_file("tmesh_data.mct1")
t = m.tallies[1]
print(t.bins[TallyMeshBinType.IBINS].values) # 2 'I' bins
[-25.  0. 25.]
print(t.bins[TallyMeshBinType.JBINS].values) # 3 'J' bins
[-15. -5.  5. 15.]
print(t.bins[TallyMeshBinType.KBINS].values) # 5 'K' bins
[ 0.  5. 10. 15. 20. 25.]
print(t.get_value(i=1, j=2, k=4))
0.000261971
```

```
# Plot 'K' and 'J' bins from last 'I' bin.
from matplotlib import pyplot as plt
data = t.values[:, :, 1, 0]
plt.imshow(data, cmap='hot', interpolation='nearest')
plt.show()
```



Support for GENXS data

```
# Trying to read GENXS data with MCNPTools:
```

```
from mcnpTools import Mctal
m = Mctal("genxs_data.mctal")
t = m.GetTally(3240) # Alpha production
print(t.GetValue(0, 0, 0, 0, 0, 0, 0, 0))
```

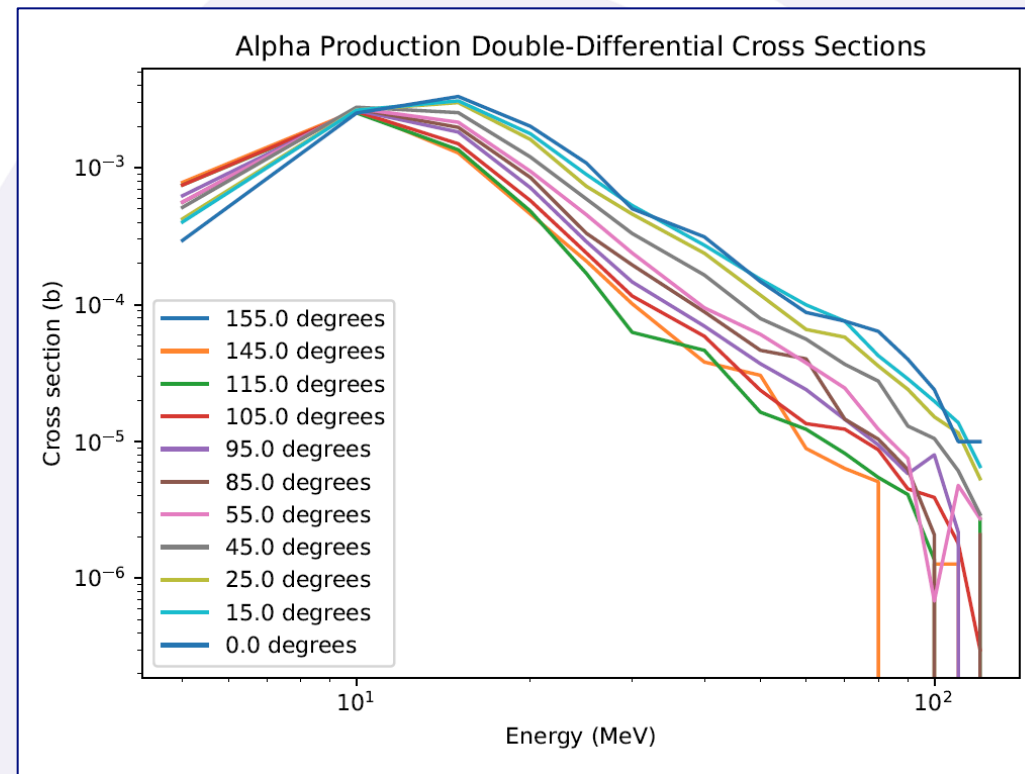
(Python hangs, then crashes) # :-(

Users can access bins information, but not the data values.

```
# Reading GENXS data with mcnp.mctal:
```

```
from mcnp.mctal import Mctal
m = Mctal.from_file("genxs_data.mctal")
t = m.tallies[3240] # Alpha production
print(t.get_value())
0.50533
```

```
from mcnp.mctal.enums import TallyBinType
for c in range(len(t.bins[TallyBinType.CBINS].values)):
    for e in range(len(t.bins[TallyBinType.EBINS].values)):
        print(t.get_value(c=c, e=e))
0.50533
0.0011408
0.000121362
...
```



Example: $^{56}\text{Fe}(n,\alpha)$ double-differential cross sections obtained using the MCNP6.3.1 code with GENXS option.

Improvements to data access

- Provides complete metadata access (user is not required to know everything).
 - Any information written to the MCTAL file has a corresponding property somewhere.
 - Compare with MCNPTools which provides access to only a subset of metadata.
 - Examples: # of perts, radiograph tally type, tally comments, tally TFC data, ...
 - Noteworthy example: ability to interrogate if tally bins are cumulative or include total.
- Provides performant data access for high-throughput workflows.
 - Nearly all array-type data can be accessed as NumPy arrays (versus index-by-index).

```
# Iterating over tally data with MCNPTools:
from mcnpTools import Mctal
m = Mctal("big_data.mctal")
t = m.GetTally(1)
for f in range(len(t.GetFBins())):
    for d in range(len(t.GetFDins())):
        ... ..
        print(t.GetValue(f, d, u, s, m, c, e, t))
```

```
# Iterating over tally data with mcnp.mctal:
import numpy as np
from mcnp.mctal import Mctal
m = Mctal.from_file("big_data.mctal")
t = m.tallies[1]
for val in np.nditer(t.values):
    print(val)
```

Access by index is also available. It is less performant but provides handling for errors and default index values.

Improved error handling and messaging

- Error messages may, on occasion, be helpful.
- Exhibit A: A file that does not exist.

```
>>> from mcnp.tools import Mctal
>>> m = Mctal("invalid_file_name.mctal")
ERROR: file invalid_file_name.mctal does not look like an MCNP mctal file
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "C:\Path\To\Python\site-packages\mcnp\tools\mcnp_tools_wrap.py", line
816, in __init__
    _mcnp_tools_wrap.Mctal_swiginit(self, _mcnp_tools_wrap.new_Mctal(*args))
    ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
RuntimeError: McnpToolsException
```

What is wrong with the file? Is it ...

- *Not existing?*
- *Wrong file format?*
- *Weird typo in header?*
- *A valid MCTAL file that MCNPTools doesn't like for some reason?*

```
>>> from mcnp.mctal import Mctal
>>> m = Mctal.from_file("invalid_file_name.mctal")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "C:\Path\To\Python\site-packages\mcnp\mctal\mctal.py", line 266, in from_file
    with open(file_name) as f:
    ^^^^^^^^^^^^^^^^^^^^^
FileNotFoundError: [Errno 2] No such file or directory: 'invalid_file_name.mctal'
```

The issue is evident.

Improved error handling and messaging

- mcnp.mctal also handles valid MCTAL files that MCNPTools doesn't like.
- Exhibit B: A valid MCTAL file according to the MCNP manual:

```

1 e      7
2      .2   .4   .7   1   3   8  12
3 vals
4      4.00E-5   .022   5.78E-4   .054   3.70E-5   .079   1.22E-5   .122
5      7.60E-6   .187   2.20E-6   .245   9.10E-7   .307

```

[illegible]

```
>>> from mcnp.mctal import Mctal
>>> m = Mctal.from_file("example.mctl")
>>> m.tallies
{0: MctalTally(shape(values)=(1, 1, 1, 1, 1, 1, 1, 7, 1),
shape(errors)=(1, 1, 1, 1, 1, 1, 1, 7, 1)))}
>>> m.tallies[0].values
array([[[[[[[[4.00e-05],
              [5.78e-04],
              [3.70e-05],
              [1.22e-05],
              [7.60e-06],
              [2.20e-06],
              [9.10e-07]]]]]]]])])
```

What about performance?

- Switching from C++/Python to pure Python, we might expect some slowness.
- Did some simple tests on a LANL Windows laptop and on Rocinante scratch.
 - Reading 10 MCTAL files of ~430 MB each (~4.3 GB total).

For all 10 files:	MCNPTools, Windows laptop	πmcnp, Windows laptop	MCNPTools, Rocinante	πmcnp, Rocinante
Total process time	160.0 s	81.3 s	25.4 s	100.0 s
Speedup vs MCNPTools	N/A	+49.2%	N/A	-294%

- What is going on here? Some (educated?) guesses:
 - Not sure why pure Python is faster on a laptop. Possibly due to coding differences.
 - On Rocinante scratch the dominant source of slowdown is the data handling and error checking done by the new Python implementation (~90-95% of time spent).

Afterword

- Questions?
- Comments?
- Concerns?
- Witty insults?

Please reach out to livelym@lanl.gov and/or mcnp_help@lanl.gov