A new python-based R-matrix code

Ed Simpson

Department of Nuclear Physics and Accelerator Applications
Research School of Physics
Australia National University

Australian National University

Overview

- Do we really need another R-matrix code?

- Code overview
  - Objectives and design
  - (Mostly, how does the user use it)

- Examples
  - $^{12}C(p,p)^{12}C$
  - $^{15}N(p,\alpha)^{12}C$

This is a work in progress!

Do we need another R-matrix code?

- Human resourcing is a challenge
  - R-matrix evaluations require skill, patience, attention to detail, even before fitting
  - Evaluators are the most important resource

- Cost: 1 person-hour = 3000 CPU hours [Amazon EC2 cloud compute]
  - How can we make use of very large-scale compute to run calculations?
  - How do we design approaches to use big compute to make the human faster? **[∞ iterations]**
  - How do we make the most of the human time (both developer and evaluator)?
  - How fast can we change one parameter AND see the results? **[1 iteration]**

Why python?

Optimise developer and evaluator time

- Fast to write/develop
- Many useful libraries (e.g., Coulomb wave functions, plotting, fitting)
  - Externalise development and maintenance
  - Easy to version manage with tools such as pip
- Interpreted (i.e. not, by default, compiled)
  - Slower for the CPU (but not necessarily a lot slower)
  - Faster for the user - can interact with objects directly (e.g., via jupyter notebooks)
- Scriptable – easily write algorithms to change poles (e.g., add/edit/remove)
- Easy to develop and integrate with machine learning models
- Accessible to students – increase human resource availability

Current Code

- Class-oriented approach
  - User interacts with objects (e.g., particle pairs, poles, sets of observables)
- Standard parameterization [Lane and Thomas]
  - Variable (user adjustable) boundary condition [Barker]
  - Originally intended to edit observed widths
- Allows any entrance/exit channel combination
- Angle-integrated or differential cross sections, but easily extensible
- Interactive and scriptable

- … As of last Friday, also can use alternative parameterization [Brune]

Brune Parameterization

- Brune's alternative parameterization is implemented as daughter class of the standard parameters R-matrix class
- Manages a list of observed widths
- Appropriately updates the formal widths
  - When an observed width is changed, it automatically updates the internal set of formal R-matrix parameters
  - Still has a boundary condition parameter which can still be freely changed, but obviously this only changes the formal widths (not observed)
  - This could be used to input a set of observed parameters and then export the corresponding formal parameters for an arbitrary B_c convention.
  - (And can also be used to demonstrate self consistency between the Brune parameters and the on-resonance formal parameters)

External libraries

Angular momentum

- Use sympy for angular momentum and calculate CG coefficients
- Stores angular momentum as rational numbers
- Also slow – buffer and store in file

SymPy Development Team.
https://www.sympy.org/en/index.html

Coulomb wave functions

- Use mpmath
- Arbitrary precision arithmetic – accurate but (very) slow
- Most Coulomb wave functions (i.e. those associated with calculating cross sections at fixed energies) need only be calculated once – store in file
- I need to do independent benchmarking…

mpmath: a Python library for arbitrary-precision floating-point arithmetic (version 1.3.0) http://mpmath.org/.

I.J. Thompson & A.R. Barnett, J. Comp. Phys., vol 64, no. 2, June 1986.
N. Michel, “Precise Coulomb wave functions for a wide range of complex *l*, *η* and *z*“, http://arxiv.org/abs/physics/0702051v1

Nucleus class

Nucleus class

```
n15 = Nucleus(a=15,z=7,spin=S(1)/2,parity=-1);
```

Describes a nucleus. Ground state masses taken from Atomic Mass Evaluation 2020 – not currently user adjustable but could be changed.

R-matrix class

BruneRmatrix class

```
cn = Nucleus(a=13,z=7);
rm = BruneRmatrix(cn=cn, cn_ex_bc=2.364, l_max=3);
```

Compound nucleus

Excitation energy for boundary conditions

$B_c = S_c(Ex)$

Maximum allowed l-value

Currently global (over all channels, but could be changed)

Particle pairs class

ParticlePair class

```
h1=Nucleus(1,1,spin=S(1)/2);
c12=Nucleus(12,6);
pp=RmatrixParticlePair(nuc1=h1,nuc2=c12,a=3.4);


rm.add_particle_pair(pp);
```

Create projectile and target nuclei

Nucleus 1

Nucleus 2

Radius parameter

Could also specify lmax here?

Add to the R-matrix object.
Adds channels given pp and lmax

Channels

There is an internal Channels class that defines the allowed channels. The user does not need to edit, but useful to know:

rm.list_channels(); ← List all channels

```
J= 1/2  Parity= -1
   0  pp= RmatrixParticlePair H1(Ex=0.0,Jpi=1/2+)+C12(Ex=0.0,Jpi=0+)   s= 1/2  l= 1
J= 1/2  Parity= 1
   0  pp= RmatrixParticlePair H1(Ex=0.0,Jpi=1/2+)+C12(Ex=0.0,Jpi=0+)   s= 1/2  l= 0
J= 3/2  Parity= -1
   0  pp= RmatrixParticlePair H1(Ex=0.0,Jpi=1/2+)+C12(Ex=0.0,Jpi=0+)   s= 1/2  l= 1
```

… (Some omitted for brevity)

```
J= 7/2  Parity= -1
   0  pp= RmatrixParticlePair H1(Ex=0.0,Jpi=1/2+)+C12(Ex=0.0,Jpi=0+)   s= 1/2  l= 3
```


Observable Sets

Created in python:

Class name for type of observable:
CrossSection, CrossSectionDifferential

Label for observable set

Experimental value and uncertainty

```
ObservableSet(name, classname,
              cn, pp_in, pp_out, coords, val_exp, dval_exp);
```

Compound nucleus and particle pairs

List of coordinates

Example ObservableSet

Example above shows an ObservableSet without experimental data.

```
energies=[0.1, 0.2, 0.3, 0.4];
angles=[89.1, 146.9]
coords=[ { "ke_cm_in": ke_cm_in, "theta_3_cm": theta_3_cm }
               for ke_cm_in in energies
               for theta_3_cm in angles ];

os=ObservableSet("My 12C(p,p)12C data","CrossSectionDifferential",cn,pp,pp,coords);

rm.add_observable_set(os);
```

Example energies and angles

Construct coords

List of dictionaries

Make the ObservableSet object and add to R-matrix object

Data Filtering

Allows user to select a subset of the data to run calculations for. Equivalent of “segments” in AZURE2, but more flexible.

```
pp_c12p = RmatrixParticlePair(nuc1=h1,nuc2=c12,a=3.4);

f = Filter(name="12C(p,p)12C filter");
f.add( ("pp_in","=",pp_c12p) );
f.add( ("pp_out","=",pp_c12p) );
f.add( ("theta_3_cm","near",150,2) );

rm.select([f]);
```

Create a Filter object with a name

Add conditions

General structure is:
“thing to filter”, “operator”, “value”

Operator “near” allows you to specify a range +/- about value

Can also filter ObservableSet name

Apply the filter

All can be done programmatically

Json file format

The calculation objects are serialized into a json-format file. Default option is to compress the json file with gz compression but is human readable if uncompressed.

But not designed to be edited directly!

We don't use python pickle for serialization – sensitive to python version changes and (in principle) can contain arbitrary executable code.

```
{
  "cn": {
    "a": 16,
    "z": 8,
    "spin": 0.0,
    "parity": 1.0,
    "ex": 0.0
  },
  "cn_ex_bc": 12.0,
  "l_max": 5,
  "particle_pairs": [
    {
      "nuc1": {
        "a": 4,
        "z": 2,
        "spin": 0.0,
        "parity": 1.0,
        "ex": 0.0
      },
      "nuc2": ...
```


Example: $^{12}C(p,p)^{12}C$

```
cn=Nucleus(13,7);
rm=BruneRmatrix(cn=cn, cn_ex_bc=2.364, l_max=3);

# Create nuclei and the respective particle pairs
h1=Nucleus(1,1,spin=S(1)/2);
c12=Nucleus(12,6);
pp=RmatrixParticlePair(nuc1=h1,nuc2=c12,a=3.4);
rm.add_particle_pair(pp);
```

Create compound nucleus and R-matrix object

Create particle pair

Add levels

Can add a level object as:

By default, asks user to confirm they want to add the level.

```
level=rm.add_level(j=0.5,parity=1,energy=2.364,skip_check=True);
```

```
print(level)
>>> E=2.364 MeV Jpi=1/2+ obs_widths=[0.]
```

Top-level information about level

Widths are a list which corresponds to the allowed channels

```
print(level.channels)
>>> [CHANNEL Jpi=1/2+ pp=H1(Ex=0.0,Jpi=1/2+)+C12(Ex=0.0,Jpi=0+)
             s=1/2 l=0 i1=1/2 parity1=+ i2=0 parity2=+]
```

Add levels and set widths

```
# 1/2+
level = rm.add_level(j=0.5,parity=1,energy=2.364,skip_check=True);
level.set_observed_width(level.channels[0],0.0341);
```

Set width using valid channel object
To set width need channel and value

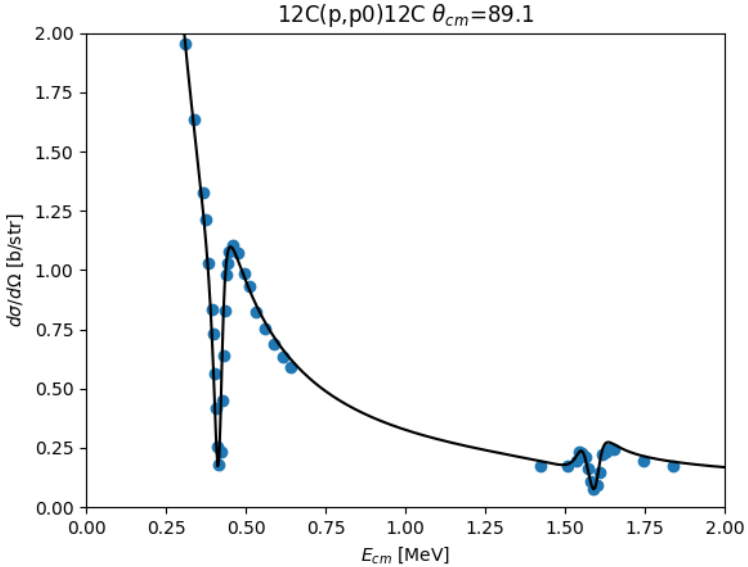
Widths set according to Azuma 2010 (azure paper)

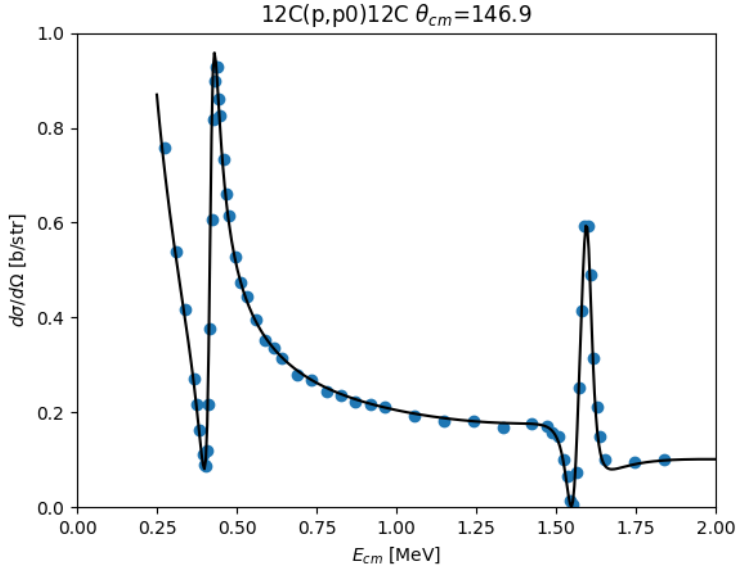
```
# 3/2-
level = rm.add_level(j=1.5,parity=-1,energy=3.500,skip_check=True);
level.set_observed_width(level.channels[0],0.0579);

# 5/2+
level = rm.add_level(j=2.5,parity=1,energy=3.545,skip_check=True);
level.set_observed_width(level.channels[0],0.0483);

rm.update();
```

Update the calculation

$^{12}C(p,p)^{12}C$ differential cross sections

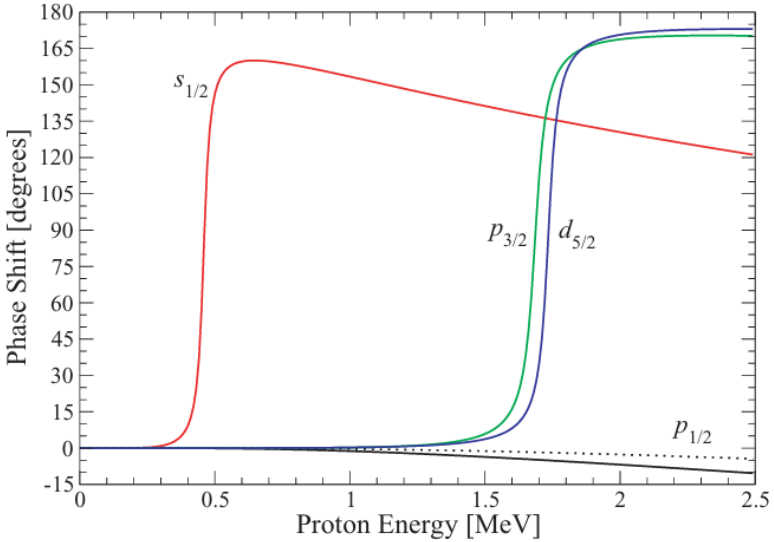
Using literature widths, with no adjustments/fitting:



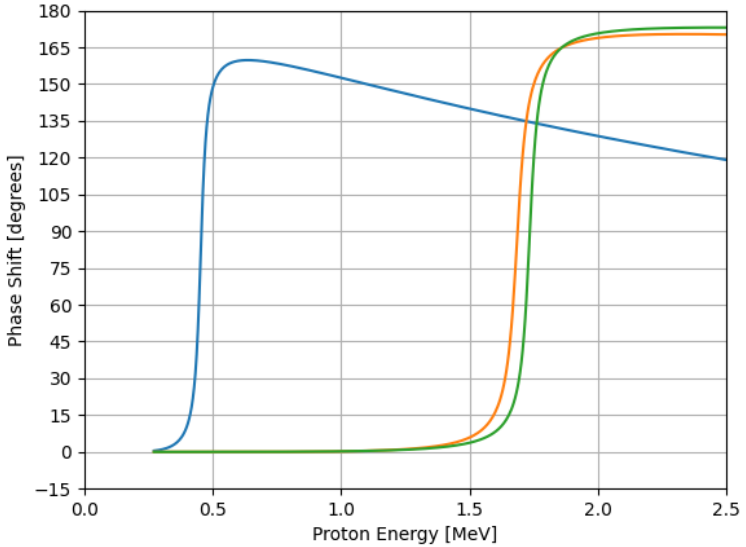


✓ Good agreement with data [Meyer 1976]

$^{12}C(p,p)^{12}C$ phase shifts



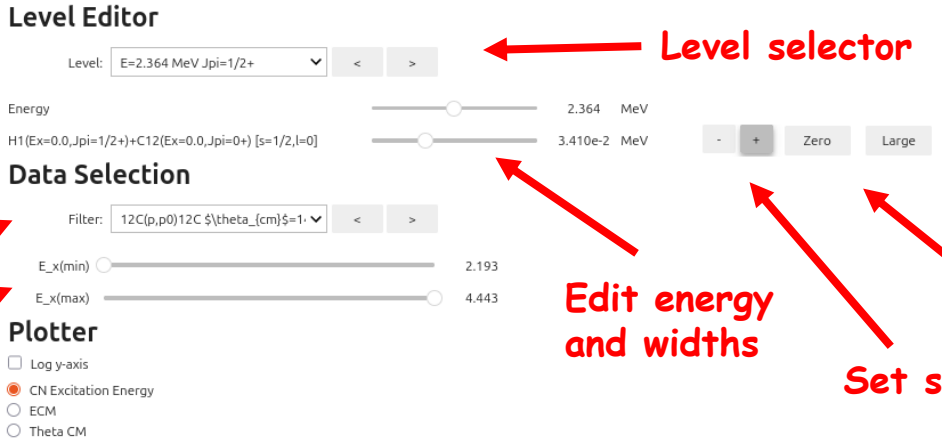
[Azuma et al, Physical Review C 81, 045805 (2010)]



✓ **Phase shifts agree very well**

Level editor

```
gui.BruneLevelEditor(rm,fpp);
```



Level selector

Data filter selector

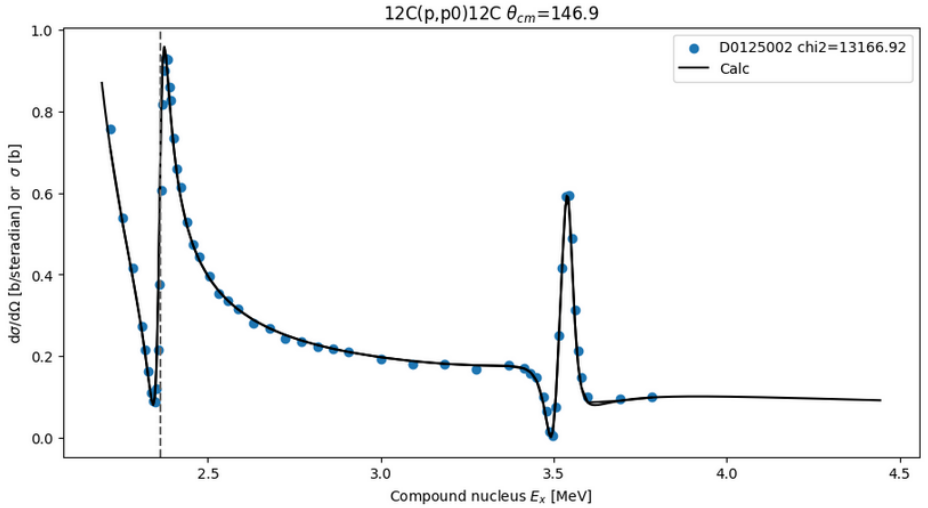
Additional filtering on Ex

Edit energy and widths

Set sign

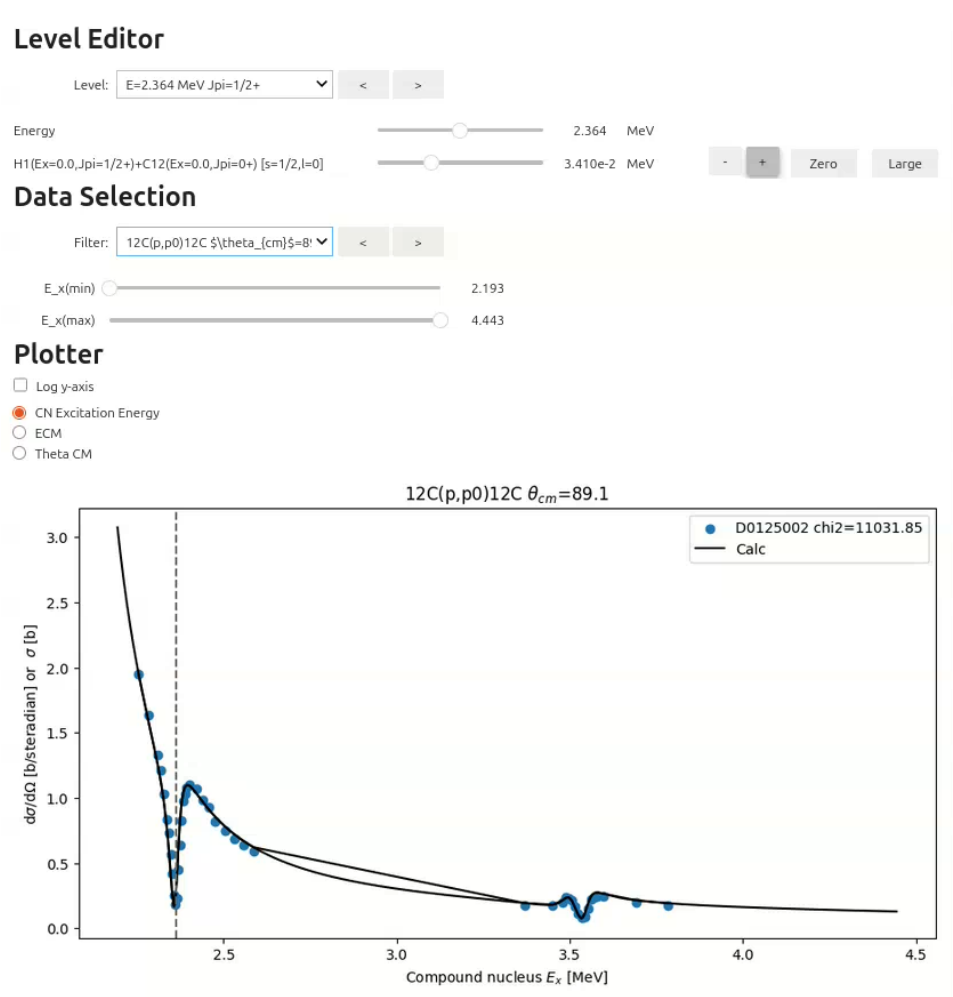
Useful buttons

Changing anything updates the plot for the current data selection in real time



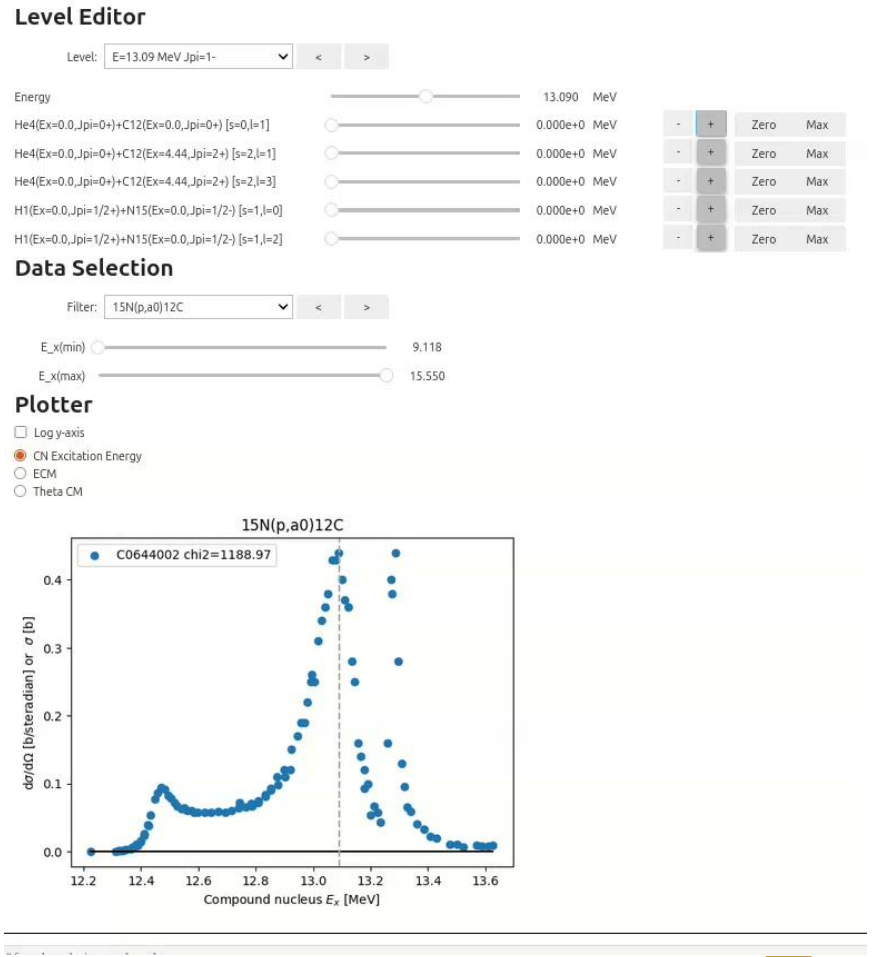
Level editor

- Calculation for $^{12}C(p,p)^{12}C$
- I'll zero widths and demonstrate how to use the editor to adjust them



$^{15}N(p,\alpha_0)^{16}O$

- Two 1^- resonances which interfere
- File is set up with resonances and data for $^{15}N(p,\alpha)^{16}O$, $^{15}N(p,p)^{15}N$ and $^{12}C(\alpha,\alpha)^{12}C$
- But this just looks at $^{15}N(p,\alpha)^{16}O$ and cross sections automatically calculated for only the selected data



Conclusions

New R-matrix code

- Initial focus: designing for the user
- How fast can we do a single calculation and see the results? Real time.
- Goal: a platform for experimenting with R-matrix analysis techniques

Finding starting parameters

- Easy to use, fast experimentation
- Can export to for different B_c - simple way to convert parameters?

Teaching and training

- Easy to install, easily portable (OS independent)
- Uses tools students may be familiar with (python, jupyter, matplotlib)
- Can distribution pre-built compound nuclei with data (and calculated Coulomb functions)

Future work…

Detailed benchmarking

- Initial tests appear to be working well but more required (e.g., check subthreshold states, Coulomb precision)
- Documentation! Should be in-code, docs generated with doxygen

Need some capacity for fitting

- Are there approaches that can separate the width magnitude from the sign?
- Master student working on convolutional neural networks
- Philosophy – small, modular and to accelerate the evaluator rather than do the evaluation

Observables

- Experimental effects (target thickness)
- Observables: Polarization data, relative yields, total cross sections
- Gamma-ray channels – though I'm hesitant to do anything without advice

What should the priorities be?

Useful stuff

- Because the whole object is serialized, could easily include time stamped evaluator log/notes
- Can include doi of data source, link to EXFOR entry
- Export formatted/unformatted table of all required information for publications

Ed Simpson

Department of Nuclear Physics and Accelerator Applications
Australian National University

edward.simpson@anu.edu.au

Australian National University