Results using the LISE library with ARIS SpecTcl

Daniel Kaloyanov

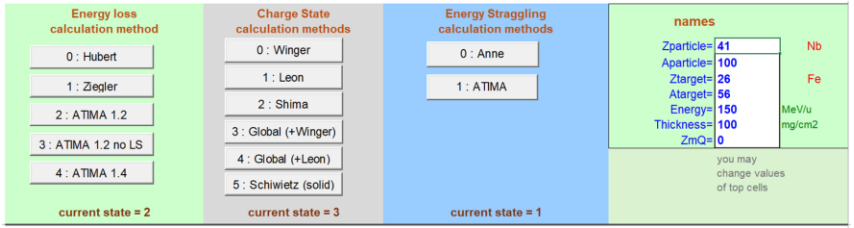
Graduate Student
kaloyano@frib.msu.edu

23 July 2026
updated 31 July 2026

This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Nuclear Physics and used resources of the Facility for Rare Isotope Beams (FRIB) Operations, which is a DOE Office of Science User Facility under Award Number DE-SC0023633, and by the US National Science Foundation under Grants No. PHY-20-12040 and 23-10078 “Windows on the Universe: Open Quantum Systems in Atomic Nuclei at FRIB”.

The LISE Library

- Started as a Windows dynamic link library for LISE for Excel, exposing LISE^{++} routines to an external caller.
- Now built for both Windows and Linux; the Linux build (libLISE_excel64.so) is what the ARIS analysis loads.
- Exports energy loss, range in matter, and energy straggling, with selectable stopping-power, charge-state, and straggling models.
- Calling the library gives full access to LISE^{++} calculations without duplicating or modifying either LISE^{++} or SpecTcl source.



function	return	parameters	example	
Energy After Matter	Residual energy in MeV/u	Zparticle, Aparticle Energy [MeV/u] Ztarget, Thickness [mg/cm2]	**143.507**	MeV/u
Energy After Matter with option	Residual energy in MeV/u	Zparticle, Aparticle Energy [MeV/u], Ztarget Thickness [mg/cm2],option	**143.60**	MeV/u
Range In Matter	Range in mg/cm2	Zparticle, Aparticle Energy [MeV/u] Ztarget	**1460.45**	mg/cm2
Range In Matter with option	Range in mg/cm2	Zparticle, Aparticle Energy [MeV/u] Ztarget, option	**1441.74**	mg/cm2
StragglingEnergy	energy straggling in material [MeV/u]	Zparticle, Aparticle Energy [MeV/u] Ztarget, Thickness [mg/cm2]	**0.043**	MeV/u
StragglingRange	range straggling in material [mg/cm2]	Zparticle, Aparticle Energy [MeV/u] Ztarget	**1.853**	mg/cm2
Stopping power	Hubert - option=0	Zparticle, Aparticle Energy [MeV/u] Ztarget	**6.306**	MeV/(mg/cm^2)

LISE for Excel interface showing some of the functions that can be used in SpecTcl

How to Use the Library

1. **Load the shared object.** Open `libLISE_excel64.so` and declare each function's argument.
2. **Set your calculation options.** `set_loss` (stopping-power), `set_charge_state`, `set_straggling`. Each choice persists until you change it, and the options can be seen in the previous slide.
3. **Call the calculation functions.** Range, energy loss, straggling, and charge state functions all use the models you set in step 2.
4. **Override per call when needed.** `_option` variants let you set what option to use with out the set_ functions.

```
lib.global_code.argtypes = [ctypes.c_int, ctypes.c_double, ctypes.c_int, ctypes.c_int, ctypes.c_double, ctypes.c_double, cty
lib.global_code.restype = ctypes.c_double

Z, M = 41, 101
E = 150.0
Zt =26
thick = 98.0
option = 1
ZminusQ = 0
ZmQ_out, ZmQ_in = 0, 0
Ab, At = 27.0, 20.0
fast = 1
qe = 2

print("global_code:", lib.global_code(ZmQ_out, Ab, Z, ZmQ_in, E, At, Zt, thick, option, fast))
```

Available Library Functions

Energy loss & range:

`trange(z,m,e,zt)` – range in matter (mg/cm²)

`treste(z,m,e,zt,thick)` – residual E after a layer

`stopping_power(z,m,e,zt,opt)`

`straggling_energy / _range`

Isotope masses:

`isotope_mass(z,m)` – nuclear mass

`isotope_mAtom(z,m)` – atomic mass

`isotope_mqe(z,m,qe) / mqq`

Charge states:

`charge_qmean(e,Zb,Zt,opt)` – mean charge

`charge_double` – returns the charge as a double

`charge_dq` – charge change

`chargeSchiwietzGas(...)`

`global_code(...)` – GLOBAL charge-state code

Config functions:

`set_loss(i)` – stopping-power model

`set_charge_state(i)` / `set_straggling(i)`

`show_option(opt)` – read current model

See examples: https://github.com/Kaloyano/LISE-Linux-Examples

Preparing the Environment

- Once the libLISE_excel64.so.3.0.1 file has been built use ldd libLISE… to show all of the libraries needed
- Set up a shell to load the required libraries
  - An example can be seen in orange

busterdaq_LISE — environment setup

```
#!/bin/bash
DAQVERSION=11.3-027

export SINGULARITY_SHELL=/bin/bash

export DAQROOT=/usr/opt/daq/$DAQVERSION
export DAQBIN=$DAQROOT/bin
export DAQLIB=$DAQROOT/lib
export DAQINC=$DAQROOT/include
export PYTHONPATH=$DAQROOT/pythonLibs:$PYTHONPATH
set DAQROOT  $DAQROOT
#echo "daqroot="$DAQROOT

# for aquisition over ssh pipes
export SING_IMAGE=/usr/opt/nscl-buster.img
echo /usr/opt/opt-buster:/usr/opt > ~/.singularity_bindpoints
echo /scratch >> ~/.singularity_bindpoints
echo /mnt/misc/sw/indep >>  ~/.singularity_bindpoints
echo /mnt/misc/sw/x86_64/all >> ~/.singularity_bindpoints
echo /usr/lib/x86_64-linux-gnu/modulecmd.tcl >> ~/.singularity_bindpoints

COMMON_BINDPOINTS="--bind /etc/timezone,/usr/opt/opt-buster:/usr/opt,/scratch,/mnt/misc/sw/indep,/mnt/misc/sw/x86_64/all,/usr/lib/x86_64-linux-gnu/modulecmd.tcl"

case `hostname`
in
spdaq* | ctlrm-daq* | u*pc* | daqcompute*)
    PERNETWORK_BINDPOINTS="--bind /mnt/events/arisdaq"
;;

#expanalysis* )
aris-analysis)
    PERNETWORK_BINDPOINTS="--bind /mnt/rawdata/arisdaq"
;;


*)
   PERNETWORK_BINDPOINTS=""
esac

export SINGULARITYENV_LD_LIBRARY_PATH=/user/arisdaq/PID/liseLib/qtLibs:/user/arisdaq/PID/liseLib:$LD_LIBRARY_PATH

TZ=America/New_York singularity shell $COMMON_BINDPOINTS $PERNETWORK_BINDPOINTS $SING_IMAGE
```

Connecting to SpecTcl

- The busterdaq_LISE shell loads the required libraries.
- CAris.cpp opens the LISE library with dlopen(RTLD_LAZY) and resolves each function with dlsym.
- Calculation options are set once at load.
  - A complete list of calculation options can be seen in image on Slide 2
  - There will be calibration parameters added that allow for swapping between options
- Two functions drive the physics: trange for range in matter and treste for residual energy after a layer.

```
        //============================================================================ liseLib begin
#ifdef _useLiseLib
    const char* libName = "libLISE_excel64.so.3.0.1";
    liseLib = dlopen(libName, RTLD_LAZY);
    if (!liseLib) {
        std::cerr << "Failed to load libLISE_excel64.so.3.0.1: " << dlerror() << std::endl;
    } else {
        std::cout << "Successfully loaded libLISE_excel64.so.3.0.1" << std::endl;

        typedef void (*set_option_func)(int);
        typedef int  (*show_option_func)(int);

        set_option_func set_loss                = (set_option_func)dlsym(liseLib, "set_loss");
        set_option_func set_charge_state = (set_option_func)dlsym(liseLib, "set_charge_state");
        set_option_func set_straggling   = (set_option_func)dlsym(liseLib, "set_straggling");
        show_option_func show_option     = (show_option_func)dlsym(liseLib, "show_option");

        if (set_loss) set_loss(4);
        if (set_charge_state) set_charge_state(3);
        if (set_straggling) set_straggling(1);

        if (show_option) {
            std::cout << "[Options] loss: " << show_option(0)
                      << ", charge: " << show_option(1)
                      << ", straggling: " << show_option(2) << std::endl;
        }

        trange = (trange_func_ptr)dlsym(liseLib, "trange");
        if (!trange) {
            std::cerr << "Failed to load trange function: " << dlerror() << std::endl;
                }

        teloss = (teloss_func_ptr)dlsym(liseLib, "treste");
        if (!teloss) {
            std::cerr << "Failed to load teloss function: " << dlerror() << std::endl;
        }
    }
#endif
        //============================================================================ liseLib end
```

How SpecTcl Calculated Z, A, and q

- First the velocity is calculated by using the reconstructed flight path length and the calibrated ToF to get both β and γ
- A/q comes from rigidity and velocity
  - $A/q = B\rho / (3.1071 \cdot \beta \cdot \gamma)$
- Z is obtained from the energy loss measured in the PIN and silicon detectors
- q is calculated using the calibrated TKE and from there we can calculate A
  - $q = TKE / [(\gamma - 1) \cdot u \cdot (A/q)]$
- These calculations are all needed before LISE range and energy loss functions can be called

Calculations Part #1

- To do the calculations we need energy per nucleon
- Brho2Energy is used and converts the magnetic rigidity to MeV/u
- Energy is calculated and the integer form of A and Z are used to calculate the Range

```
//WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW

double Brho2Energy(double Brho, double M, double Q) {
    if (Q == 0) return 0.0;
    double Gamma_Beta = Brho / 3.107129578 / M * Q;
    double gamma = sqrt(1 + Gamma_Beta * Gamma_Beta);

    return (gamma - 1.0) * 931.494013;
}
//WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW

void ARISparticleID::CalcRange(int Zt) {
#ifdef _useLiseLib
    int targetZ = (Zt > 0) ? Zt : 14;

    if (ARIS::trange && Q.isValid() && Am3Z.isValid() && ZmQ.isValid() && brho.isValid()) {

        int Zint = int(Z + 0.5);
        double aCalc = 3.0*Z + Am3Z;
        int Aint = int(aCalc + 0.5);

        if (Zint > 120) {
            range.setInvalid();
            return;
        }

        if (Zint <= 0 || Aint <= 0 || targetZ <= 0) {
            range.setInvalid();
            return;
        }

        double energy = Brho2Energy(brho, aCalc, Q);
        double result = ARIS::trange(Zint, Aint, energy, targetZ);
        range = result;
    } else {
        range.setInvalid();
    }
#endif
}
```

Calculations Part #2

- The initial Setup for the energy loss is identical except for the thickness variable
  - Currently hard coded, but will eventually be a tunable parameter
- *teloss* returned the remaining energy so it is subtracted from the initial energy

```
void ARISparticleID::CalcEnergyLoss(int Zt) {
#ifdef _useLiseLib

    if (!ARIS::teloss) {
        std::cerr << "Error: teloss function not loaded" << std::endl;
        dE_Calc.setInvalid();
        dE_per_u_Calc.setInvalid();
        return;
    }

    int targetZ = (Zt > 0) ? Zt : 14;
    double thickness = 69.636;

    if (Z.isValid() && Am3Z.isValid() && ZmQ.isValid() && brho.isValid() && Q.isValid()) {

        int Zint = int(Z + 0.5);

        if (Zint > 120) {
            dE_Calc.setInvalid();
            dE_per_u_Calc.setInvalid();
            return;
        }

        double aCalc = 3.0*Z + Am3Z;
        int Aint = int(aCalc + 0.5);
        int Qint = int(Q + 0.5);
        double qCalc = Z - ZmQ;
        double energyPerU = Brho2Energy(brho, aCalc, Q);
```

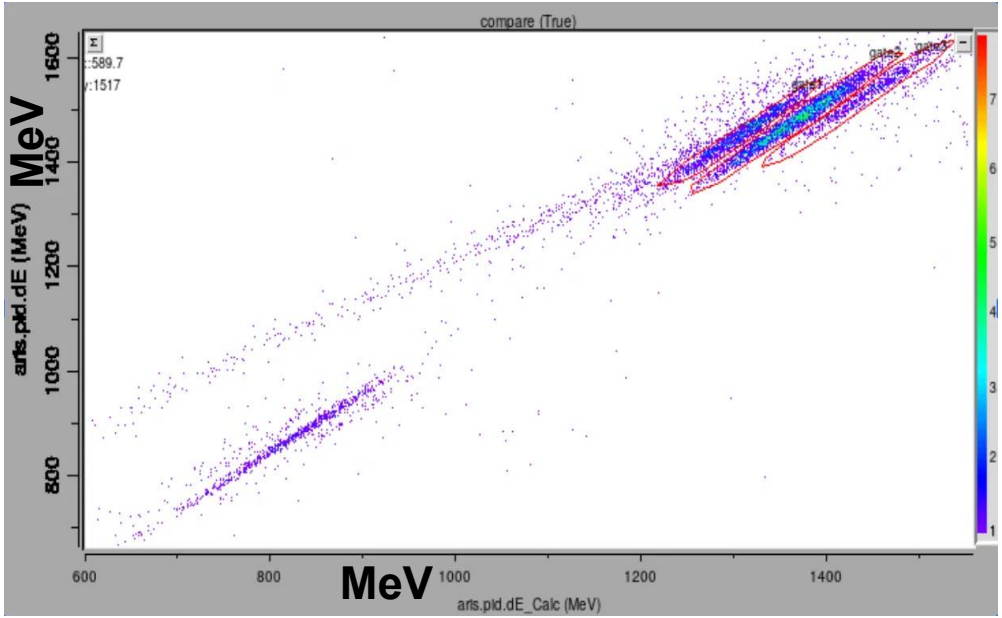
```
        double eResPerU = ARIS::teloss(Zint, Aint, energyPerU, targetZ, thickness);

        std::cerr << "[CalcEnergyLoss]"
                  << " eResPerU=" << eResPerU
                  << " dE_per_u_Calc=" << energyPerU - eResPerU
                  << " dE_Calc=" << energyPerU*aCalc - eResPerU*aCalc
                  << std::endl;

        if (eResPerU > 0 && eResPerU < energyPerU) {
            double eResidual = eResPerU * aCalc;
            dE_Calc = energyPerU*aCalc - eResidual;
            dE_per_u_Calc = energyPerU - eResPerU;
        } else {
            std::cerr << "[CalcEnergyLoss] eResPerU out of range: eResPerU="
                      << eResPerU << " energyPerU=" << energyPerU << std::endl;
            dE_Calc.setInvalid();
            dE_per_u_Calc.setInvalid();
        }
    } else {
        std::cerr << "[CalcEnergyLoss] invalid inputs:"
                  << " Z.isValid()="    << Z.isValid()
                  << " Am3Z.isValid()=" << Am3Z.isValid()
                  << " ZmQ.isValid()="  << ZmQ.isValid()
                  << " brho.isValid()=" << brho.isValid()
                  << " Q.isValid()="    << Q.isValid()
                  << std::endl;
        dE_Calc.setInvalid();
        dE_per_u_Calc.setInvalid();
    }
#endif
}
//WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
```

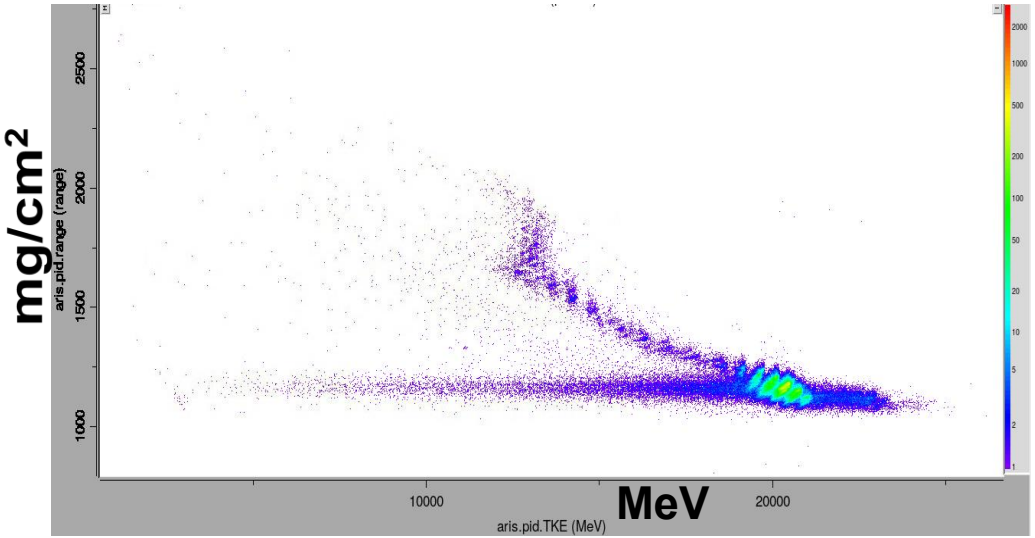
Results #1

- The two energies don't line up meaning a TKE calibration or thickness used is incorrect
- There are also three different lines corresponding to charge states.
- This is because when we do Brho2Energy we use Q meaning the initial energy is different for dE_Calc



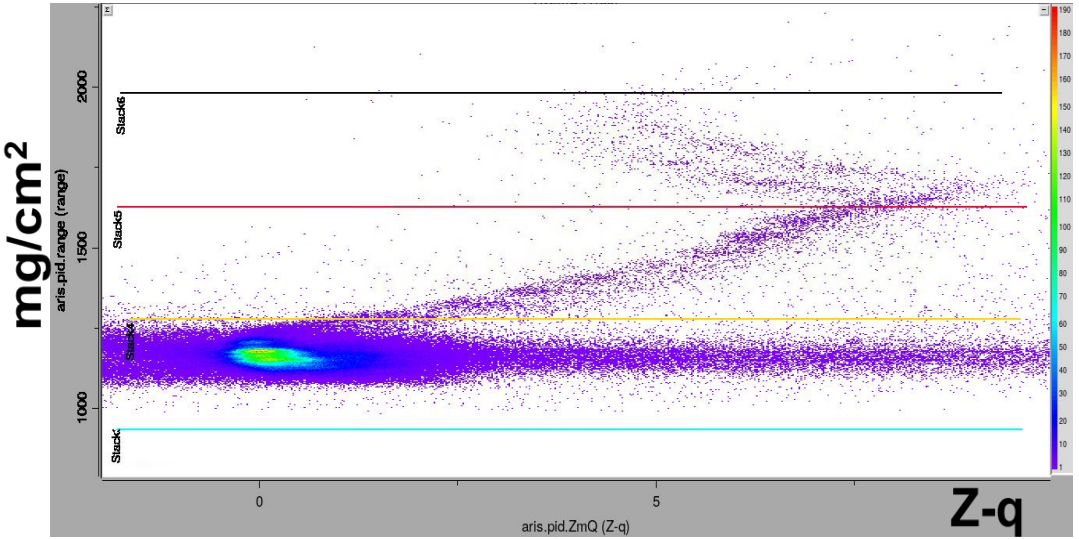
Results #2

- Noticeable noise around the 1200 mg/cm^2 area
- Steady downward trend with some breakdown near the 1750-2000 range



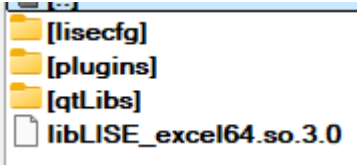
Results #3

- The horizontal lines mark the end of each detector in the stack. Where the distribution crosses them tells you which ions stop in which detector.
- Same noise as previously seen around the 1200 (mg/cm^2) area
- Same break down as the previous result
- Most likely punch through causing incomplete TKE



Access to LISE Library for Linux

http://lise.frib.msu.edu/work/SpecTcl/liseLib-3-0-1.zip



Next Steps

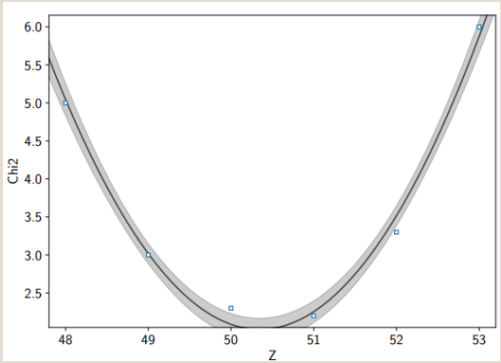
Near term

- Use the dE vs dE_Calc plots to evaluate and then refine the TKE calibration.
- Fit the offset from the diagonal and use the range calculations to extract an effective silicon thickness.
- Apply the predicted range as a cut to separate stopped events from punch-through and reaction events.

Bragg Curve Z calibration

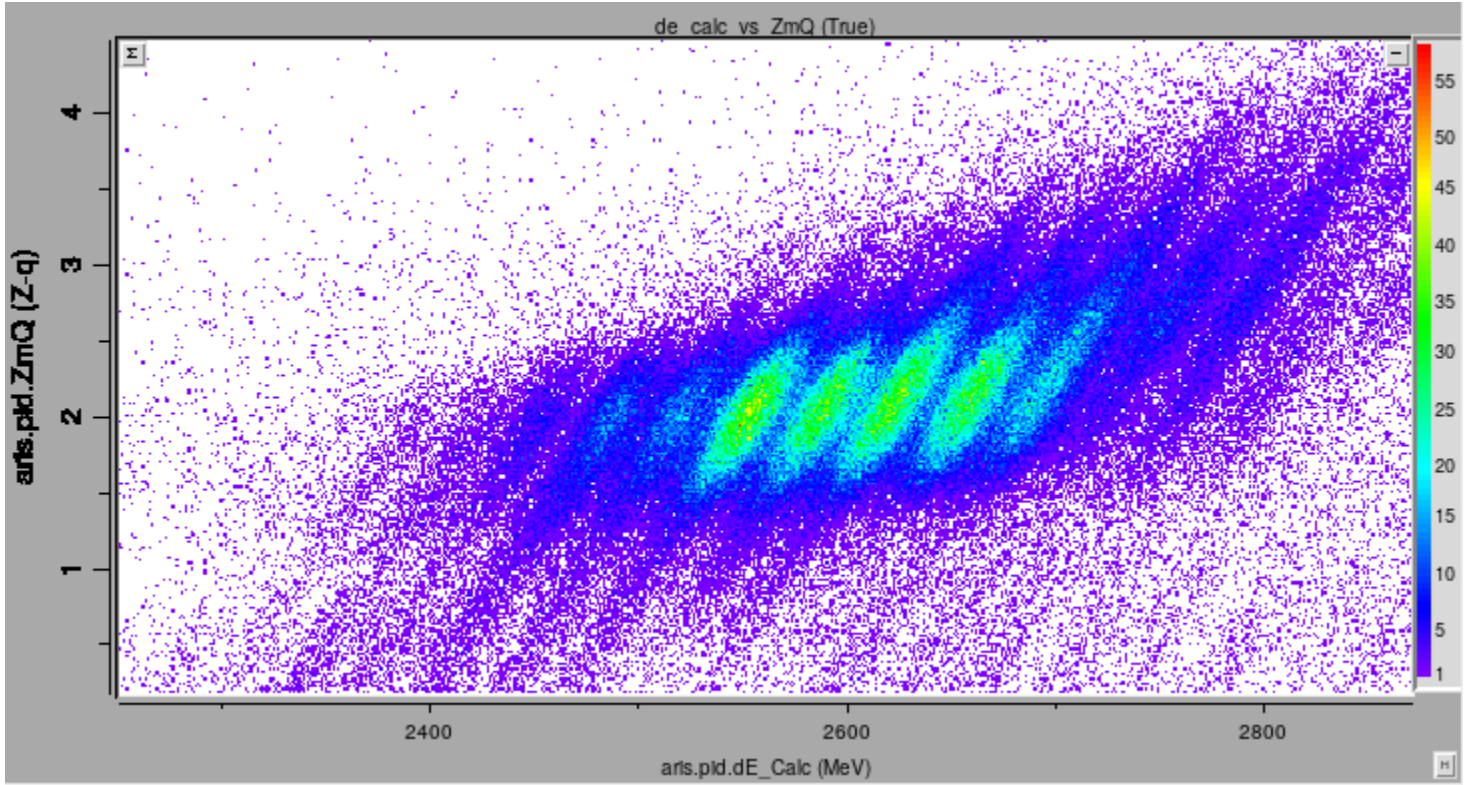
- Use the LISE library to predict dE in each layer of the silicon stack.
- Compare predicted against measured dE.
- Scan Z over a range compute χ^2 for each trial value.
- Take the minimum of the scan as the fitted Z_Bragg.

Obtaining Z_Bragg



- Three scan points: parabolic interpolation through the minimum.
- More than three: quadratic fit to the χ^2 curve.
- The example shown returns Z_Bragg = 50.35.

Supplement


de_calc_vs_ZmQ (True)
arts.pid.ZmQ (Z-q)
1
2
3
4
2400
2600
2800
arts.pid.dE_Calc (MeV)
55
50
45
40
35
30
25
20
15
10
5
1