Calibrating the Liquid Drop Model Challenge

https://github.com/ascsn/theory-challenges

The purpose of this challenge is for you to calibrate the Liquid Drop Model https://en.wikipedia.org/wiki/Semi-empirical_mass_formula.

If you have never done anything with Python, we suggest you take a look at this: https://www.youtube.com/watch?v=AJFen_Z3mWM&t=1524s. Also, ChatGPT can be of much help to start learning how to code well: https://chatgpt.com/

Your task are to:

*Non-Bayesian way:

- Import the data from the AME 2016 table (included in the github). We are only using nuclei above A=16 to avoid light nuclei where the LDM fails particularly. Perform a curve fit using the built in functions from python (https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.curve_fit.html) and take note of the reported uncertainties in the parameters.
- Construct a cost function as the sum of the squares of the residuals between your model predictions and the experimental data.
- Numerically minimize this cost function as a function of the four Liquid Drop Model parameters (search on google for scipy minimize function). The optimal parameters will come out of the minimization.
- Make a plot of the residual of your calibrated model and the experimental data. Notice anything interesting pattern?

*Bayesian way:

- Make a model calibration using the Bayesian formalism that is defined in the acompaning file "# Guided Example Bayesian calibration". For the error, use your estimation from the previous point (the model error in this case is much smaller than the actual experimental uncertainties).
- Plot the corner plot posterior as well as the model values on the Binding Energy per nucleon for the Calcium chain up to 60Ca including the available experimental data.
- What would be the results if you have used in the calibration the Binding Energy per nucleon instead of the total Binding Energy?
- Bonus: Find the experimental error in the masses and repeat the calibration using only

experimental errors. This should give a good demonstration on the dangers of not taking into account model errors.

In [2]:
```
import numpy as np
```

In [3]:
```
data = np.loadtxt('Masses2016.txt', skiprows=1)

def LDM(params,x):
    #x = (n,z)
    #params= parameters (volume, surface, asymmetry, Coulomb)

    n=x[0]
    z=x[1]

    return params[0]*(n+z) - params[1]*(n+z)**(2/3) - params[2]*((n-z)**2/(n+z)) -
```