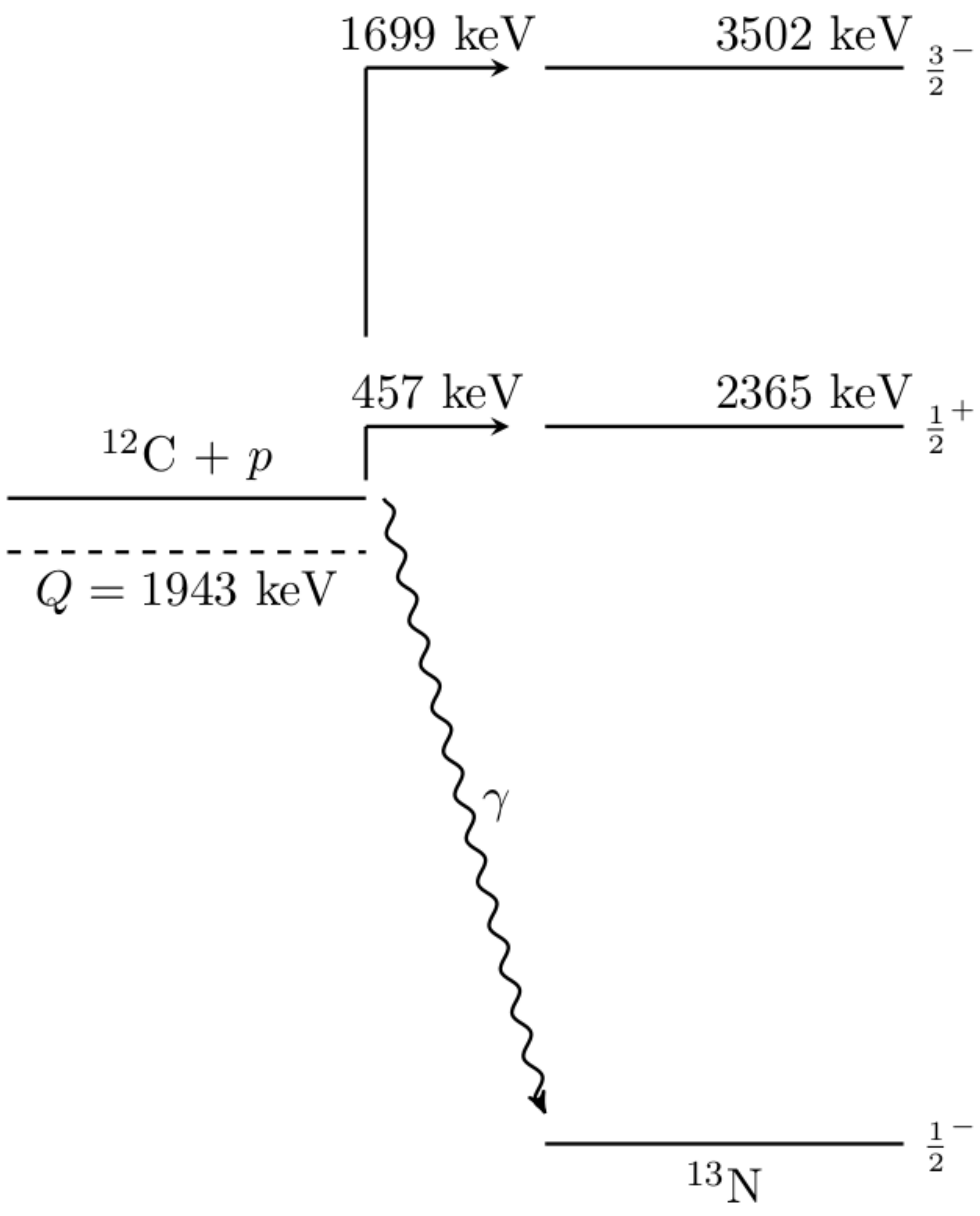
Uncertainty Estimation for R-Matrix Analysis

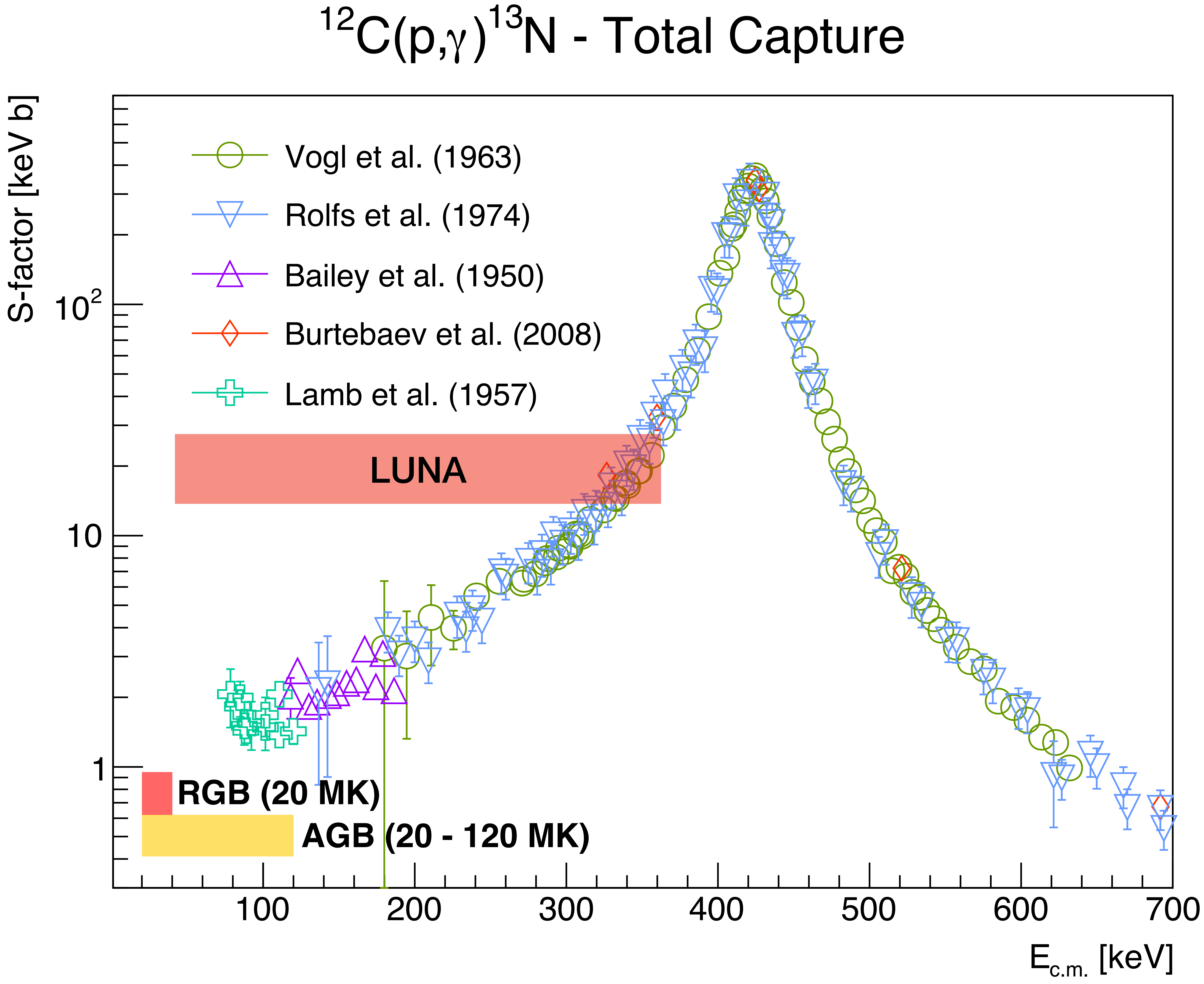
Jakub Skowronski - Università degli Studi di Padova, INFN Padova

INDEN-LE - 18/11/2024

Introduction - $^{12}C(p,\gamma)^{13}N$ Reaction

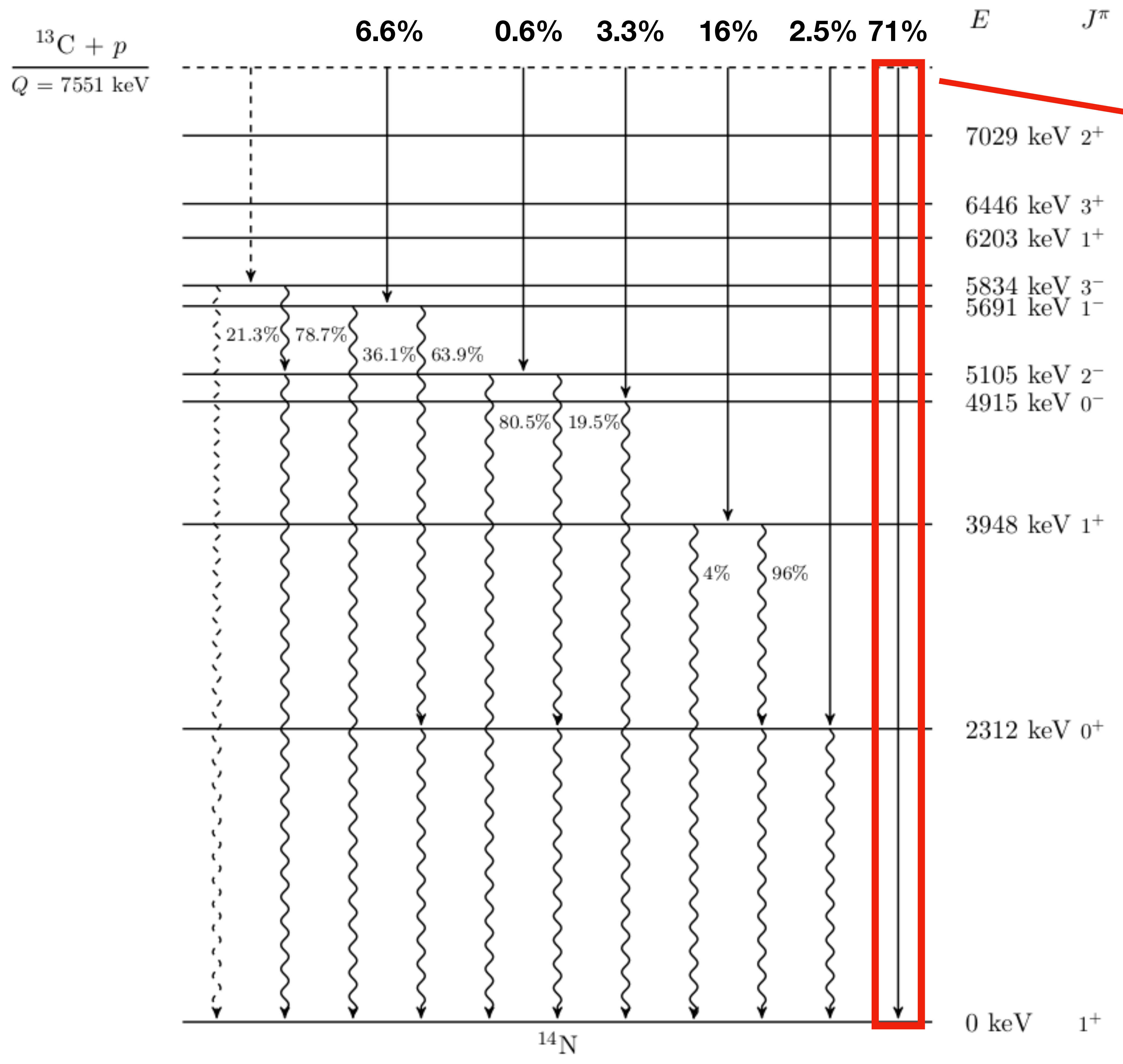
$^{12}C + p \rightarrow {}^{13}N + \gamma$ (Q = 1.943 MeV)

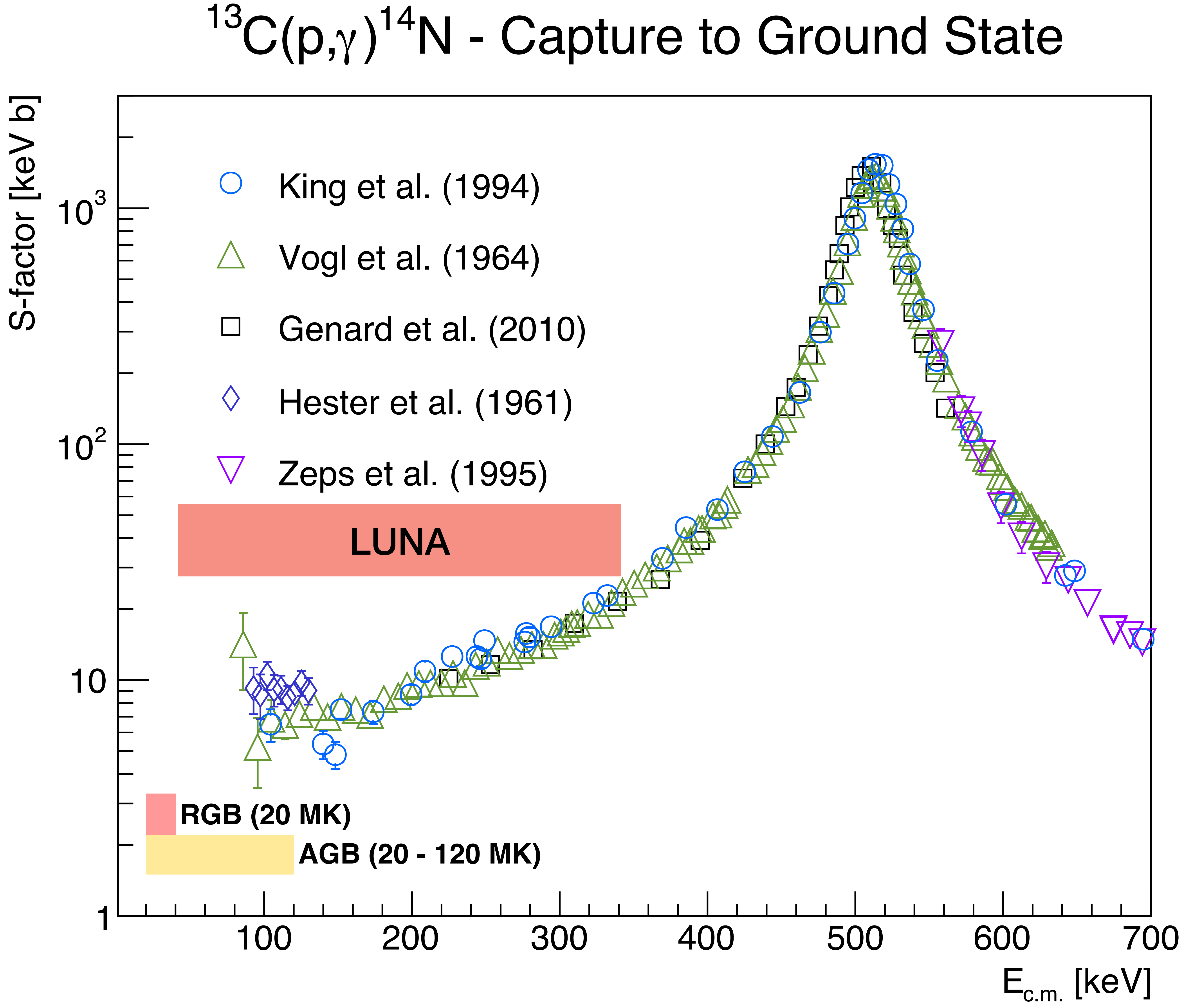




Introduction - $^{13}C(p,\gamma)^{14}N$ Reaction

$^{13}C + p \rightarrow {}^{14}N + \gamma$ (Q = 7.551 MeV)

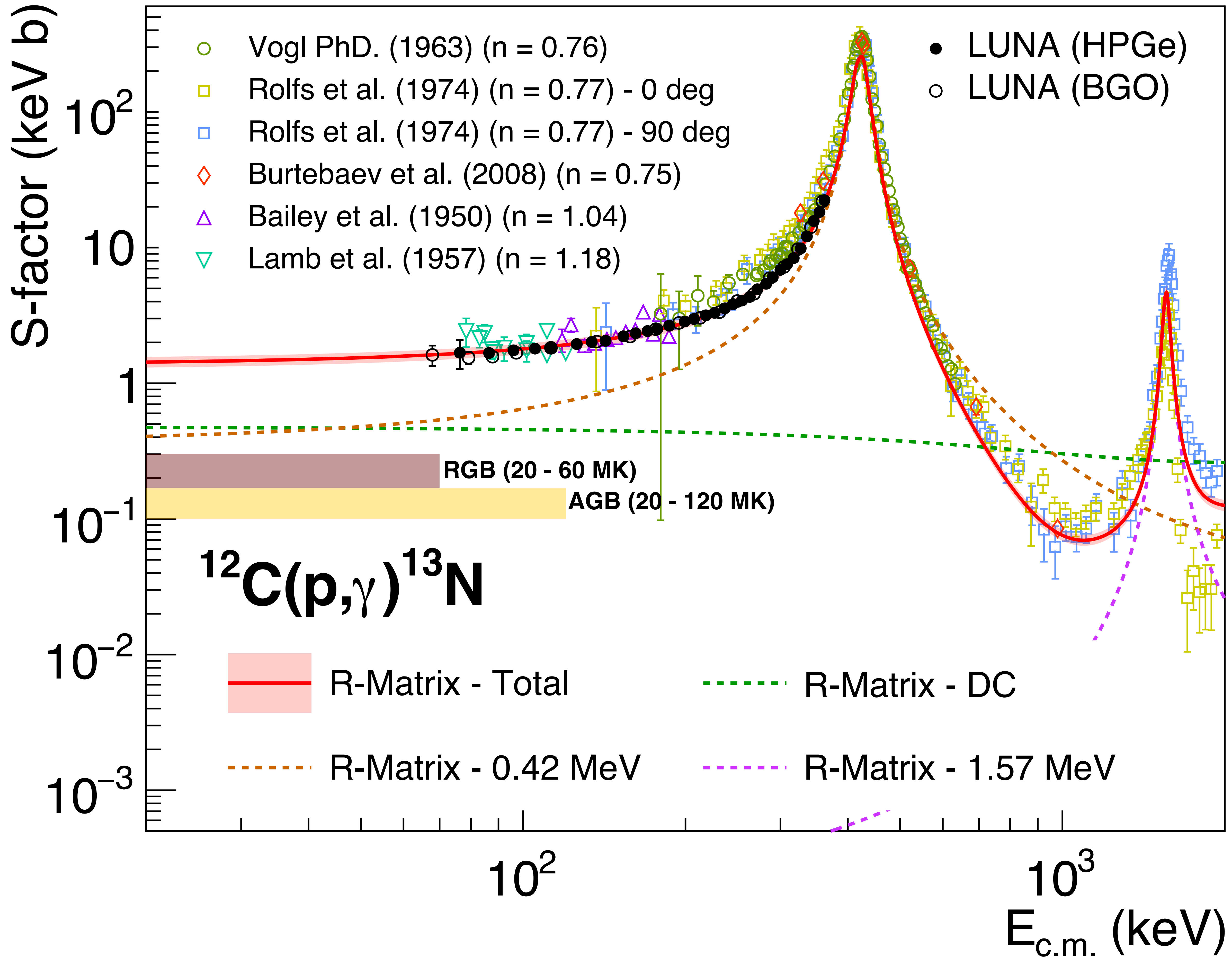




Introduction - Results

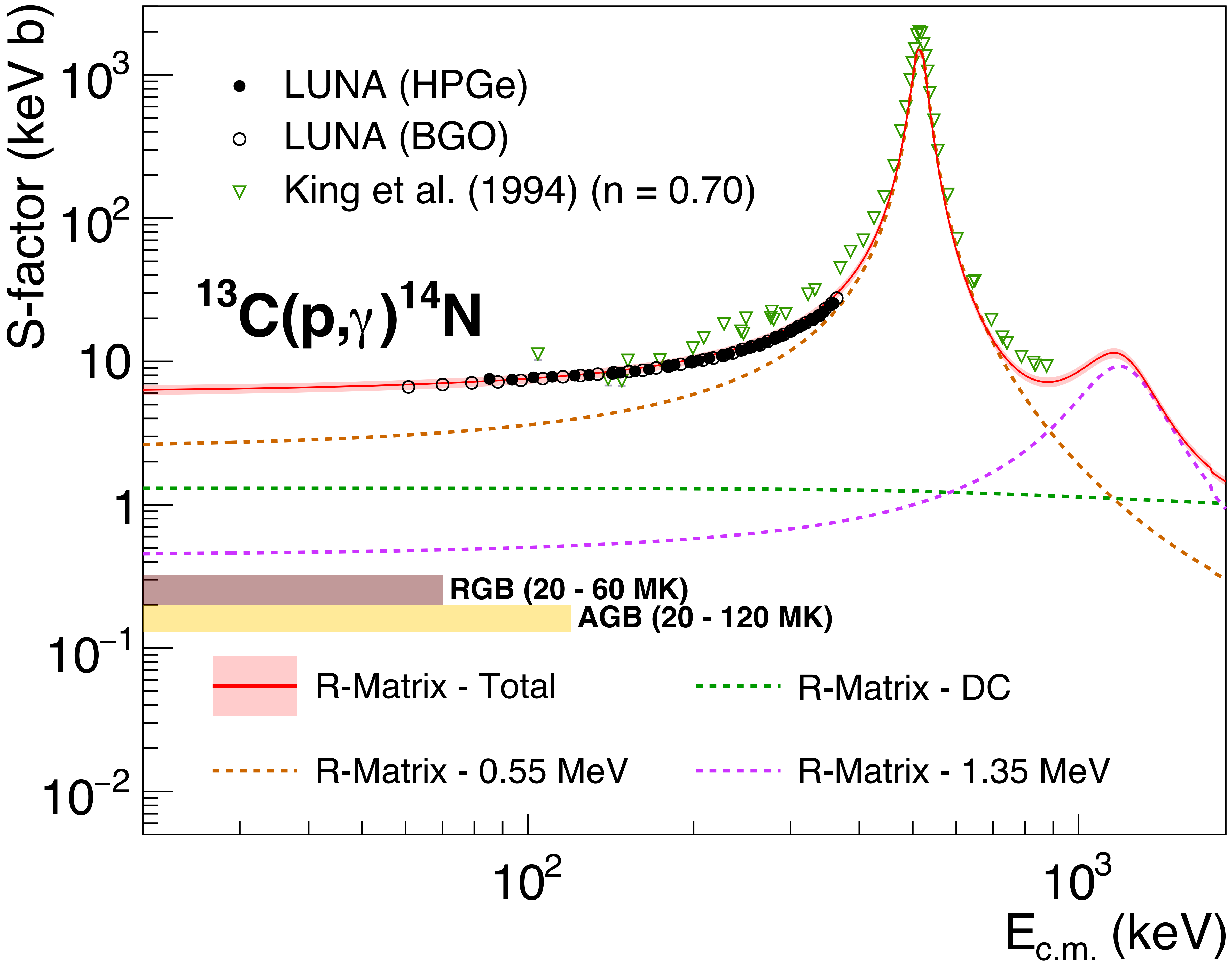
$^{12}C(p,\gamma)^{13}N$

~ 25% lower than literature…



$^{13}C(p,\gamma)^{14}N$

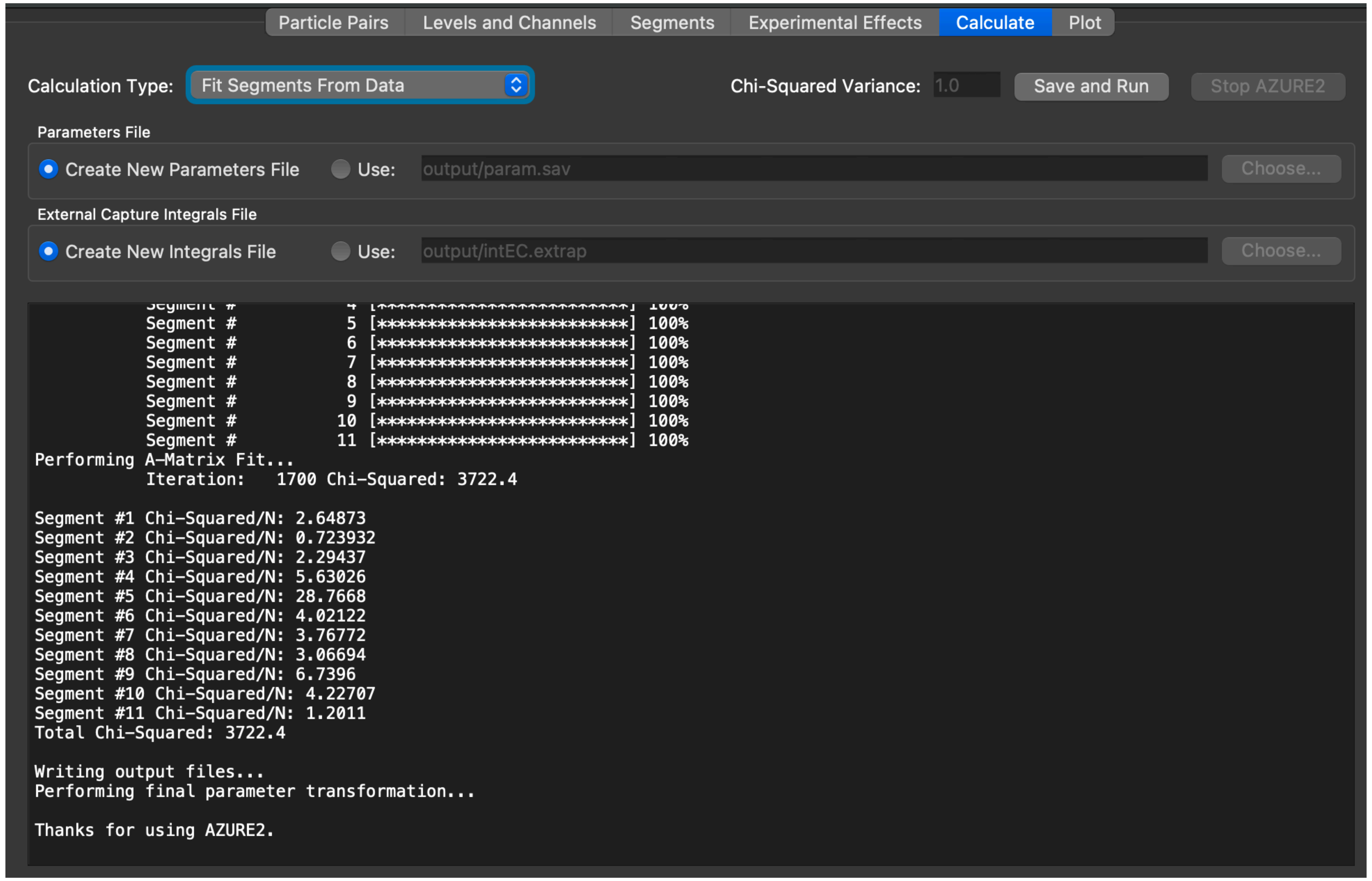
~ 30% lower than literature…



Frequentist Approach

First Attempt with AZURE2

- I started by using blindly the AZURE2 built-in minimiser, ie. MINUIT2
- The best-fit looked good, but…



AZURE2

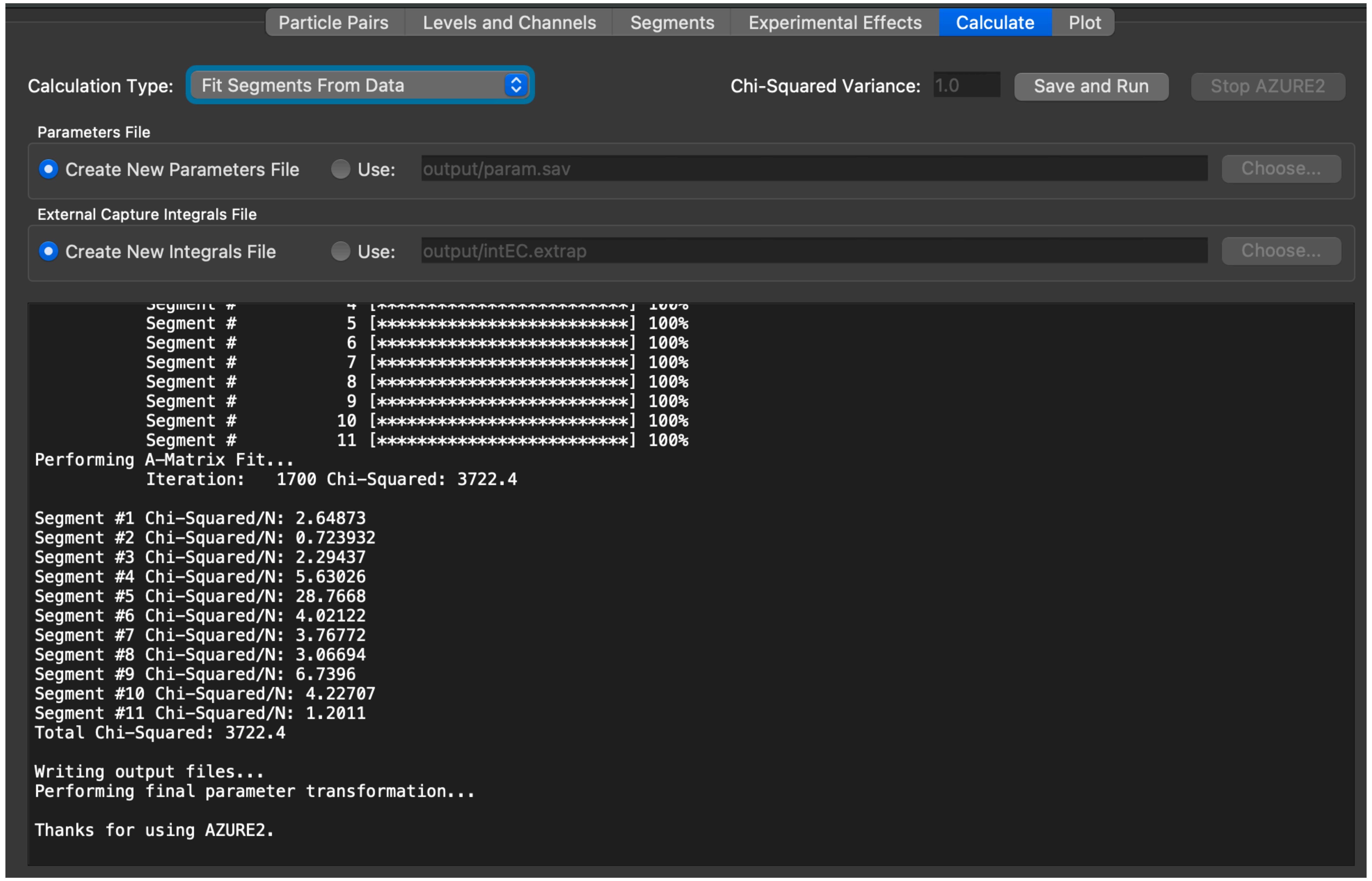
LUNA

First Attempt with AZURE2

- I started by using blindly the AZURE2 built-in minimiser, ie. MINUIT2
- The best-fit looked good, but…
- The uncertainty is provided on the **formal parameters**

↓

Not trivial to get cross section uncertainty out of it

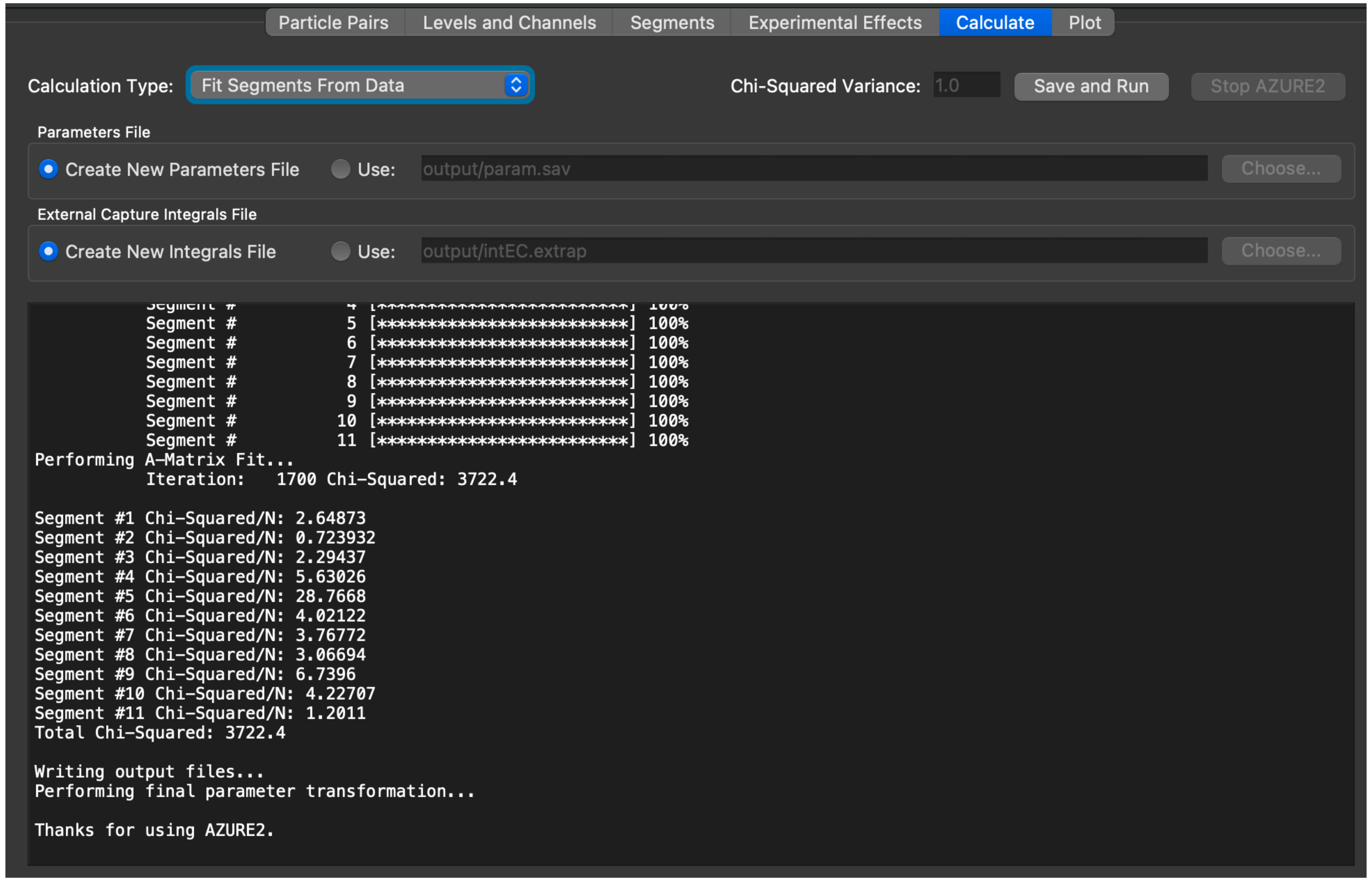


AZURE2

LUNA

First Attempt with AZURE2

- I started by using blindly the AZURE2 built-in minimiser, ie. MINUIT2
- The best-fit looked good, but…
- The uncertainty is provided on the **formal parameters**
- No possibility of **changing the minimiser**
- No possibility of **using bayesian methods**



AZURE2

LUNA

BRICK

Bayesian R-matrix Inference Code Kit (**BRICK**) published in **2021**

https://pypi.org/project/brick-james   10.3389/fphy.2022.888476

What it is: **python3** package to **run AZURE2** in headless mode

BRICK

Bayesian R-matrix Inference Code Kit (**BRICK**) published in 2021

https://pypi.org/project/brick-james 10.3389/fphy.2022.888476

What it is: **python3** package to **run AZURE2** in headless mode

How does it work?

1. Copies your .azr, data and output files
2. Modifies the .azr file accordingly to your calculation
3. Launches the AZURE2 calculation (without GUI)
4. Reads the output files
5. Removes the copied files

LUNA

BRICK

Bayesian R-matrix Inference Code Kit (**BRICK**) published in 2021

https://pypi.org/project/brick-james 10.3389/fphy.2022.888476

Advantages

- You can use any minimiser you want

- **method** (*str*, *optional*) –
  Name of the fitting method to use. Valid values are:
  - *'leastsq'*: Levenberg-Marquardt (default)
  - *'least_squares'*: Least-Squares minimization, using Trust Region Reflective method
  - *'differential_evolution'*: differential evolution
  - *'brute'*: brute force method
  - *'basinhopping'*: basinhopping
  - *'ampgo'*: Adaptive Memory Programming for Global Optimization
  - *'nelder'*: Nelder-Mead
  - *'lbfgsb'*: L-BFGS-B
  - *'powell'*: Powell
  - *'cg'*: Conjugate-Gradient
  - *'newton'*: Newton-CG
  - *'cobyla'*: Cobyla
  - *'bfgs'*: BFGS
  - *'tnc'*: Truncated Newton
  - *'trust-ncg'*: Newton-CG trust-region
  - *'trust-exact'*: nearly exact trust-region
  - *'trust-krylov'*: Newton GLTR trust-region
  - *'trust-constr'*: trust-region for constrained optimization
  - *'dogleg'*: Dog-leg trust-region
  - *'slsqp'*: Sequential Linear Squares Programming
  - *'emcee'*: Maximum likelihood via Monte-Carlo Markov Chain
  - *'shgo'*: Simplicial Homology Global Optimization
  - *'dual_annealing'*: Dual Annealing optimization

LUNA

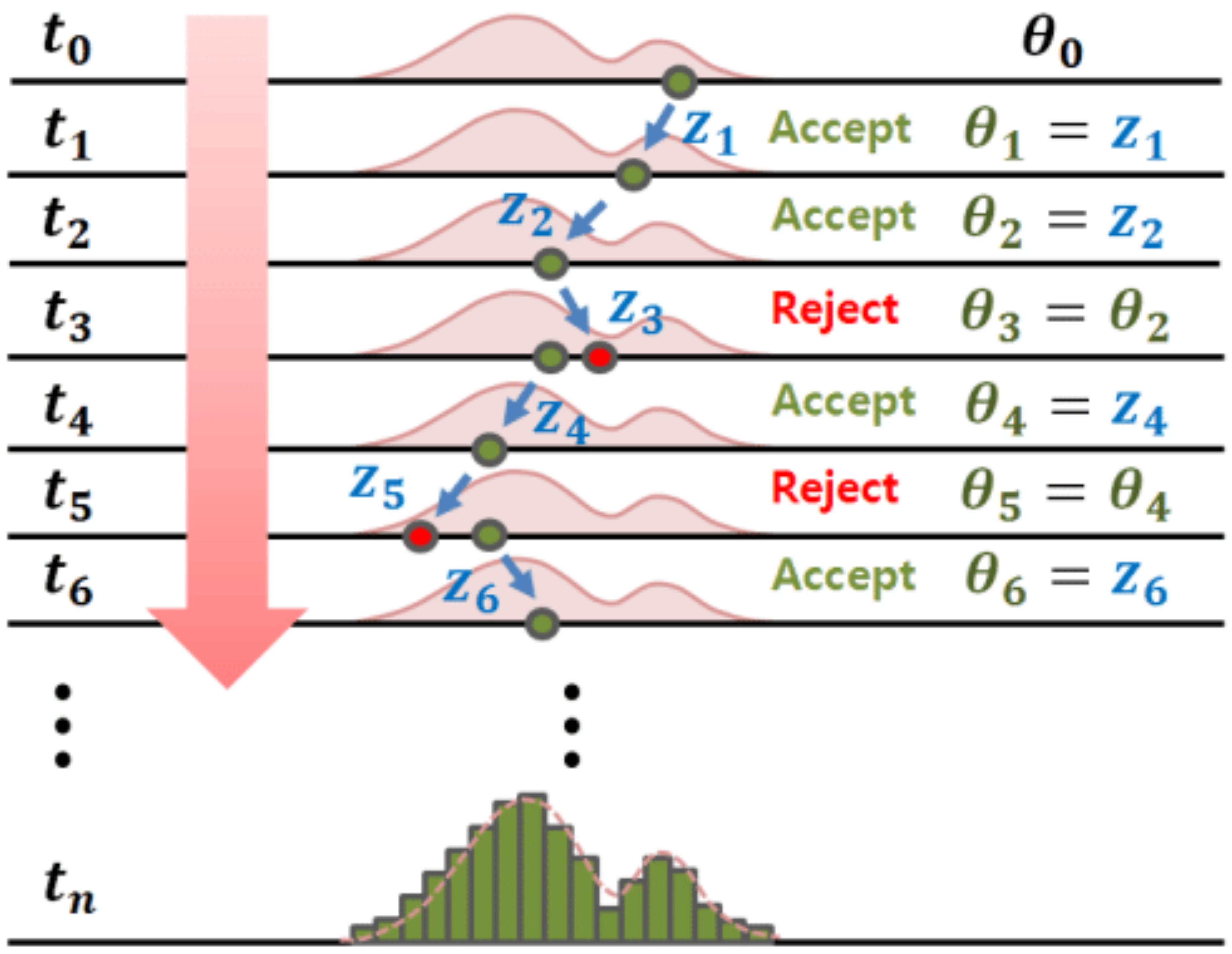
BRICK

Bayesian R-matrix Inference Code Kit (**BRICK**) published in 2021

https://pypi.org/project/brick-james 10.3389/fphy.2022.888476

Advantages

- You can use any minimiser you want
- You can use bayesian approaches



BRICK

Bayesian R-matrix Inference Code Kit (**BRICK**) published in 2021

https://pypi.org/project/brick-james   10.3389/fphy.2022.888476

Advantages

- You can use any minimiser you want
- You can use bayesian approaches
- You can easily manipulate the data

1. **Ask AZURE2 to calculate your cross section**
2. **Do your tricks: apply convolution, distributions, corrections etc.**
3. **Confront it with measured data**

BRICK

Bayesian R-matrix Inference Code Kit (**BRICK**) published in 2021

https://pypi.org/project/brick-james 10.3389/fphy.2022.888476

Advantages

- You can use any minimiser you want
- You can use bayesian approaches
- You can easily manipulate the date
- You can define your minimising functions

$$\chi^2 = \sum_{i,j} \left(\frac{f_j y_{\mathrm{obs,i}} - y_{\mathrm{theo}}}{f_j \sigma_{\mathrm{stat},i}} \right)^2 + \left(\frac{1 - f_j}{\sigma_{\mathrm{sist}}} \right)^2$$

$$\log \mathcal{L} = \sum_i -\frac{1}{2} \log \left(2\pi\sigma^2_{\mathrm{stat,i}} \right) - \frac{1}{2} \left(\frac{f_j y_{\mathrm{obs,i}} - y_{\mathrm{theo}}}{f_j \sigma_{\mathrm{stat},i}} \right)$$

LUNA

BRICK

Bayesian R-matrix Inference Code Kit (**BRICK**) published in 2021

https://pypi.org/project/brick-james   10.3389/fphy.2022.888476

Advantages

- You can use any minimiser you want
- You can use bayesian approaches
- You can easily manipulate the date
- You can define your minimising functions
- You can easy sample your parameters

ANC ~ Gaussian(μ, σ)

Get **1000 random numbers** and see **how it affects** your cross section

BRICK

Bayesian R-matrix Inference Code Kit (**BRICK**) published in 2021

https://pypi.org/project/brick-james   10.3389/fphy.2022.888476

Advantages

- You can use any minimiser you want
- You can use bayesian approaches
- You can easily manipulate the date
- You can define your minimising functions
- You can easy sample your parameters

AZURE2 = R-Matrix Calculator

BRICK

Bayesian R-matrix Inference Code Kit (**BRICK**) published in 2021

https://pypi.org/project/brick-james 10.3389/fphy.2022.888476

Advantages

- You can use any minimiser you want
- You can use bayesian approaches
- You can easily manipulate the date
- You can define your minimising functions
- You can easy sample your parameters

AZURE2 = R-Matrix Calculator

Disadvantages

- Heavy relies on I/O operations
- Slower than using raw AZURE2

LUNA

Frequentist Attempt

We can define our function to minimise as follows:

$$\chi^2 = \sum_{i,j} \left(\frac{f_j y_{\mathrm{obs,i}} - y_{\mathrm{theo}}}{f_j \sigma_{\mathrm{stat},i}} \right)^2 + \left(\frac{1 - f_j}{\sigma_{\mathrm{sist}}} \right)^2$$

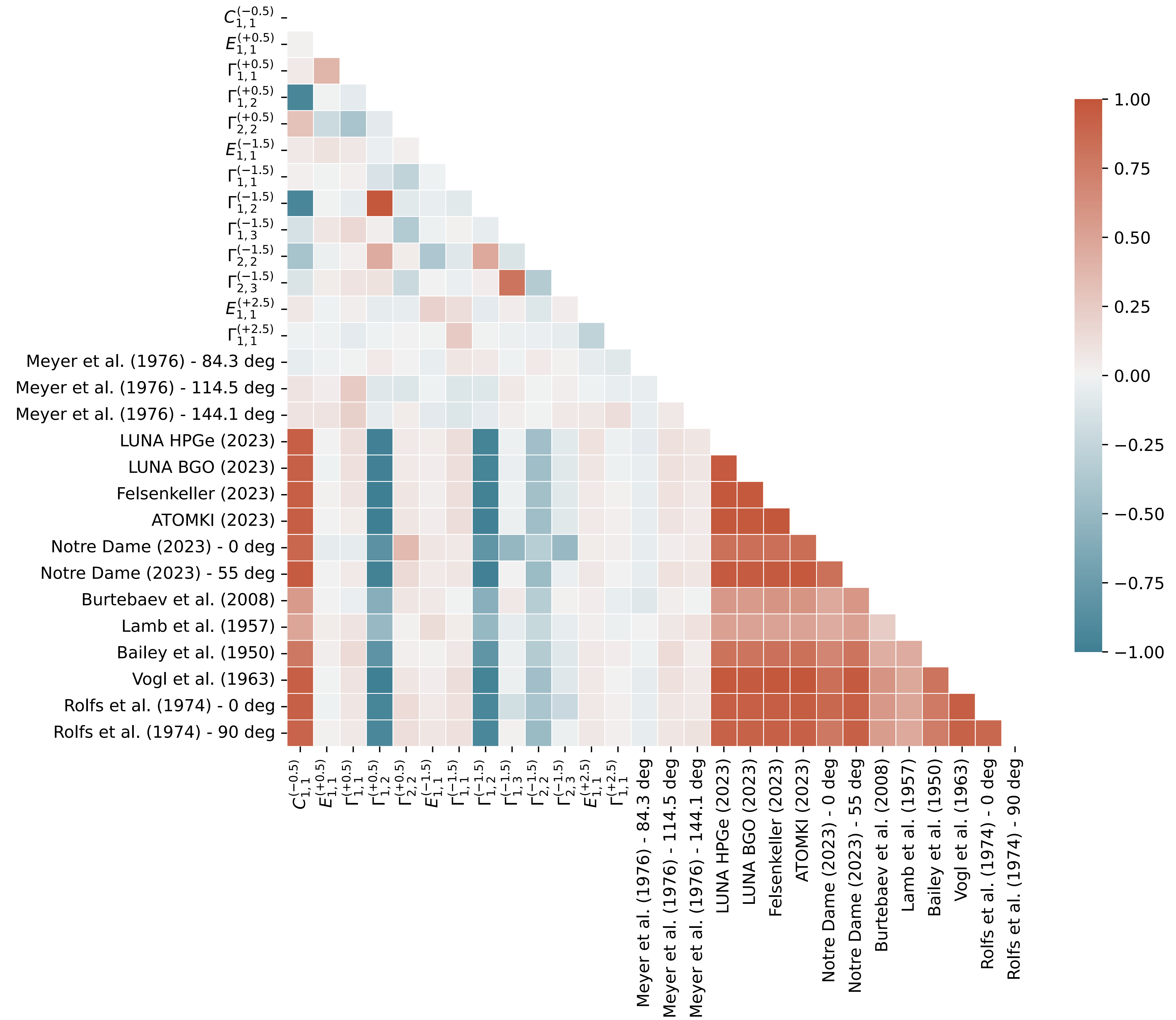
doi: **10.1016/0168-9002(94)90719-6**

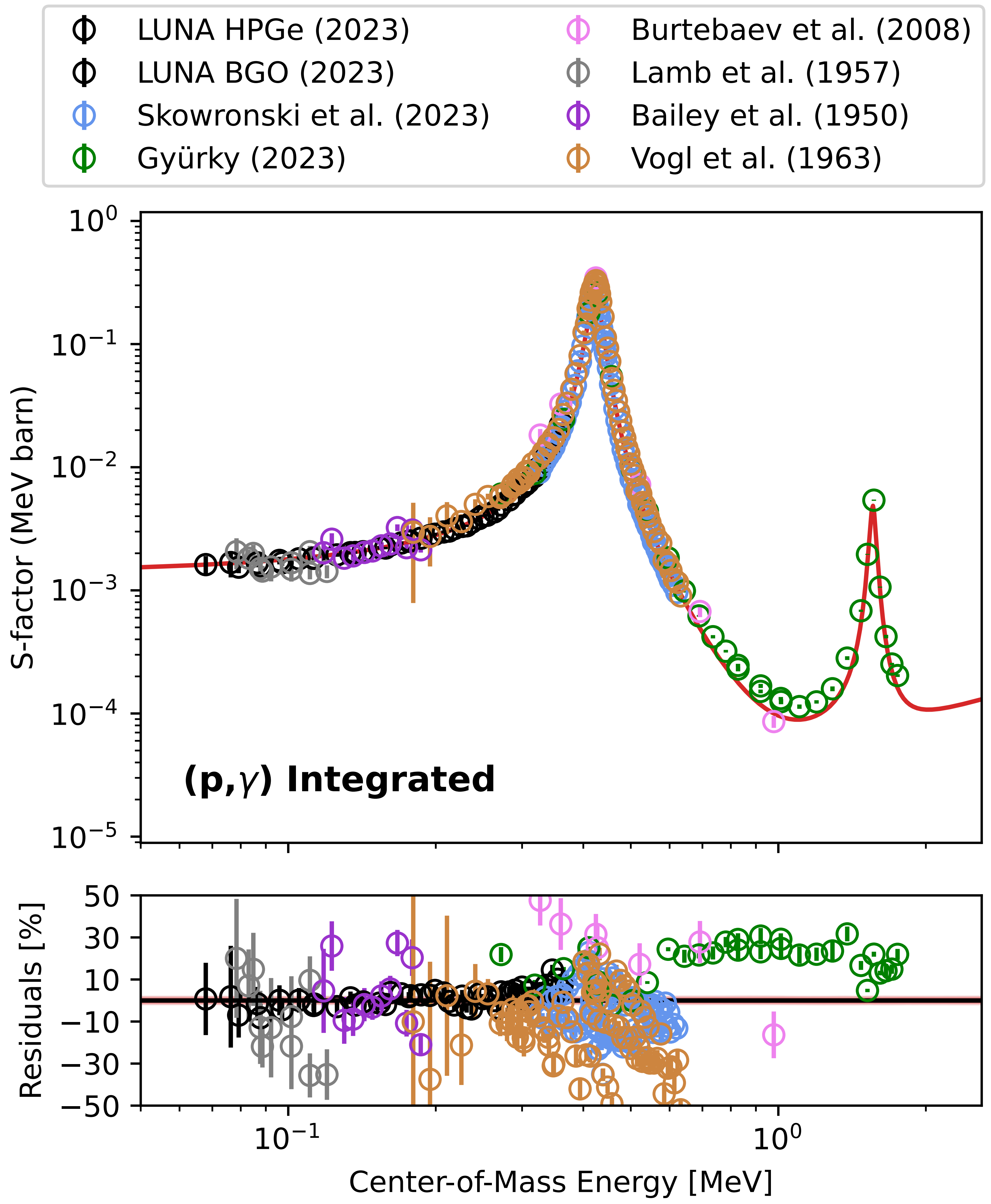
Then use the MINUIT2 wrapper for python3, *iminuit*

LUNA

Frequentist Fit - $^{12}C(p,\gamma)^{13}N$ (1)

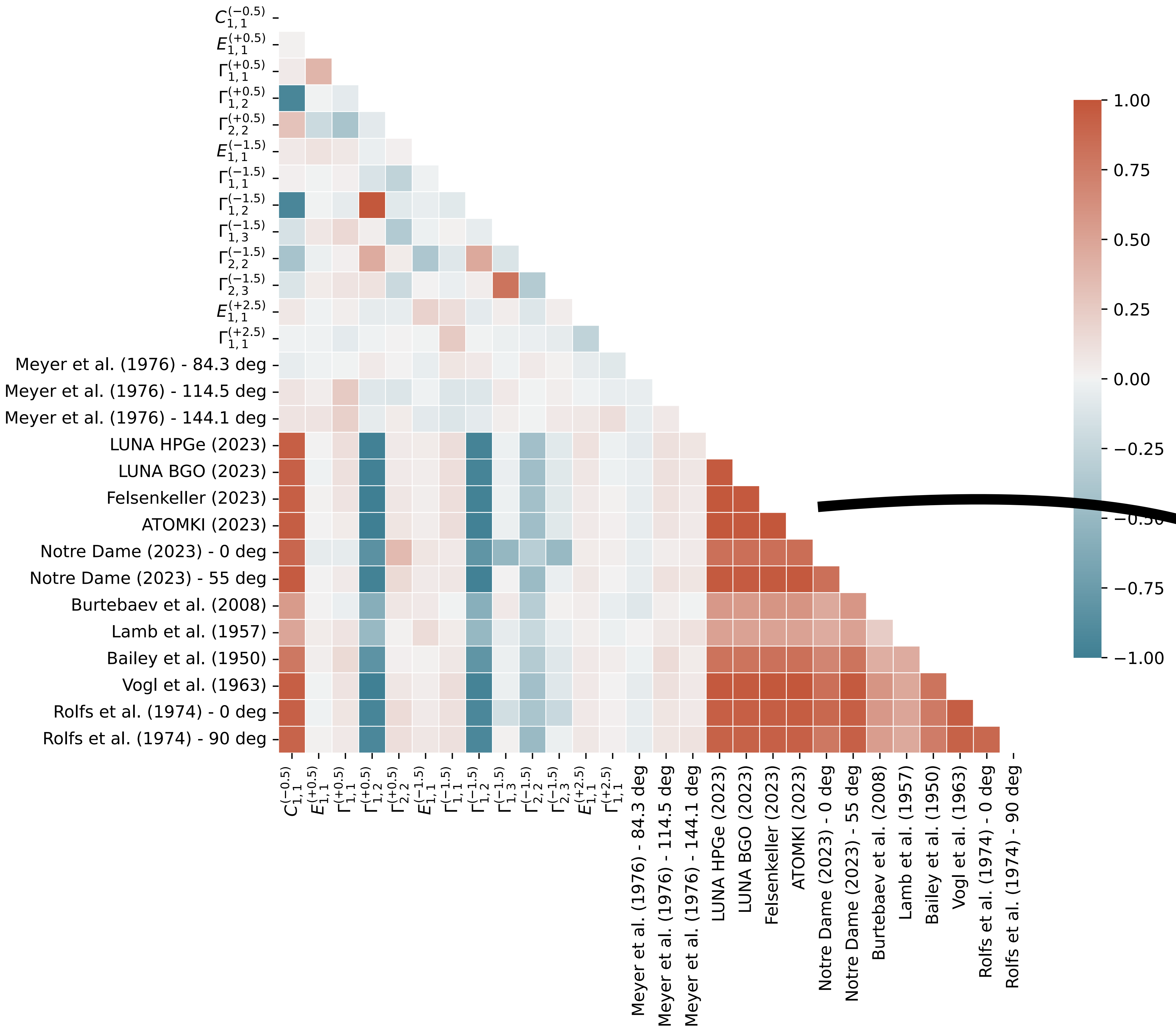
What happened when I applied MINUIT2?





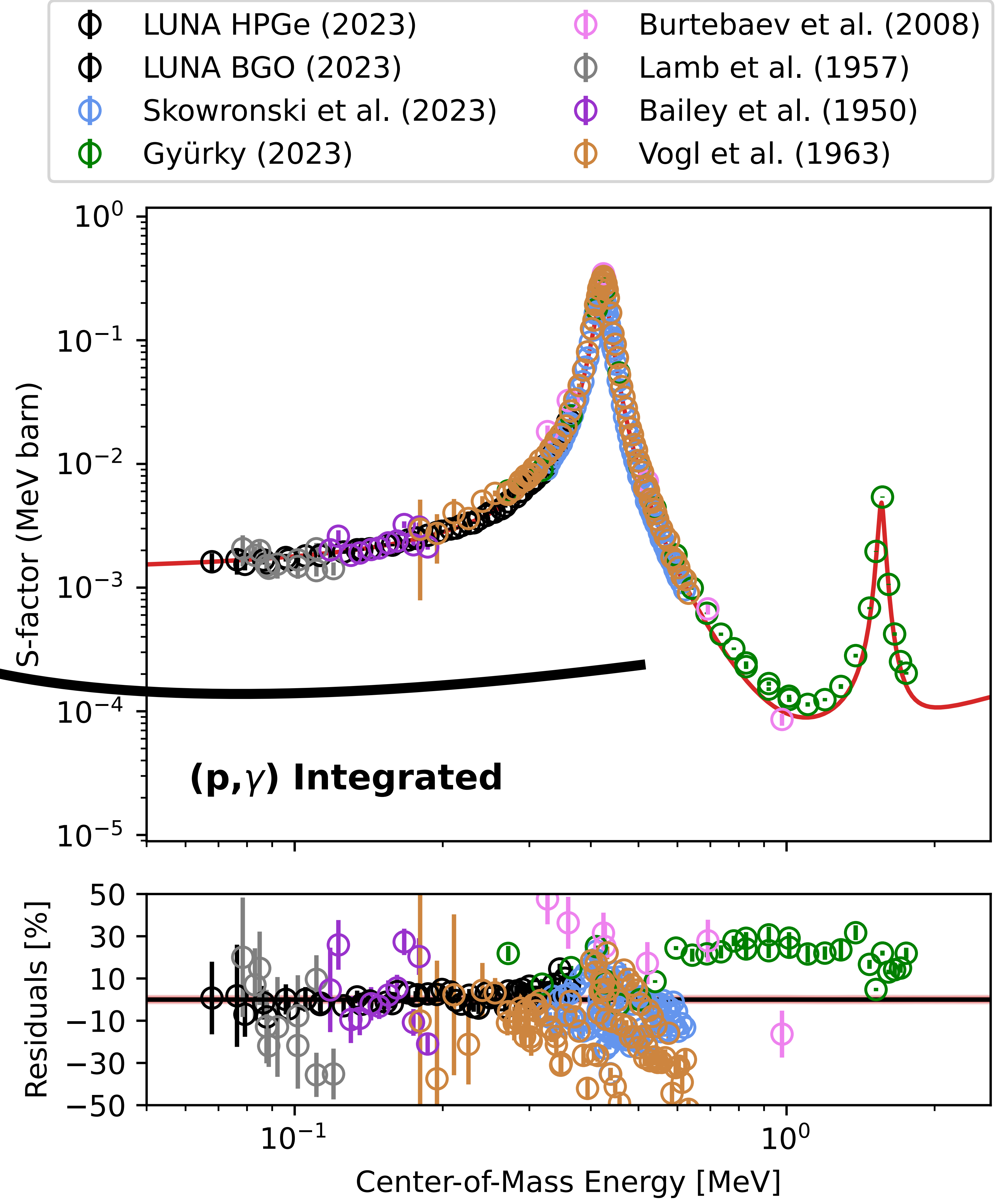
Frequentist Fit - $^{12}C(p,\gamma)^{13}N$ (1)

What happened when I applied MINUIT2?



Small uncertainty

Huge Correlation



Frequentist Uncertainty (2)

How is the covariance matrix, ie. uncertainty, estimated in MINUIT2?

MINUIT2 does get the **relative errors**, ie. covariance, correctly
but the **absolute scale** depends on

$$\chi^2_{\mathrm{dof}}\left(p_{\mathrm{best}} + \sigma_p\right) = \chi^2_{\mathrm{dof}}\left(p_{\mathrm{best}}\right) + 1$$

Particle Data Group, Prog. Theor. Exp. Phys. 2022, 083C01

Frequentist Uncertainty (2)

How is the covariance matrix, ie. uncertainty, estimated in MINUIT2?

MINUIT2 does get the **relative errors**, ie. covariance, correctly but the **absolute scale** depends on

$$\chi^2_{\mathrm{dof}}\left(p_{\mathrm{best}} + \sigma_p\right) = \chi^2_{\mathrm{dof}}\left(p_{\mathrm{best}}\right) + 1$$

Particle Data Group, Prog. Theor. Exp. Phys. 2022, 083C01

Discrepant datasets → High χ^2 → Small fit uncertainty

Frequentist Uncertainty (2)

How is the covariance matrix, ie. uncertainty, estimated in MINUIT2?

MINUIT2 does get the **relative errors**, ie. covariance, correctly
but the **absolute scale** depends on

$$\chi^2_{\mathrm{dof}}\left(p_{\mathrm{best}} + \sigma_p\right) = \chi^2_{\mathrm{dof}}\left(p_{\mathrm{best}}\right) + 1$$

Particle Data Group, Prog. Theor. Exp. Phys. 2022, 083C01

Discrepant datasets → High χ^2 → Small fit uncertainty

… and indeed the previous fit had $\chi^2_{dof} = 4.5$

Frequentist Uncertainty (2)

How is the covariance matrix, ie. uncertainty, estimated in MINUIT2?

MINUIT2 does get the **relative errors**, ie. covariance, correctly
but the **absolute scale** depends on

$$\chi^2_{\mathrm{dof}}\left(p_{\mathrm{best}} + \sigma_p\right) = \chi^2_{\mathrm{dof}}\left(p_{\mathrm{best}}\right) + 1$$

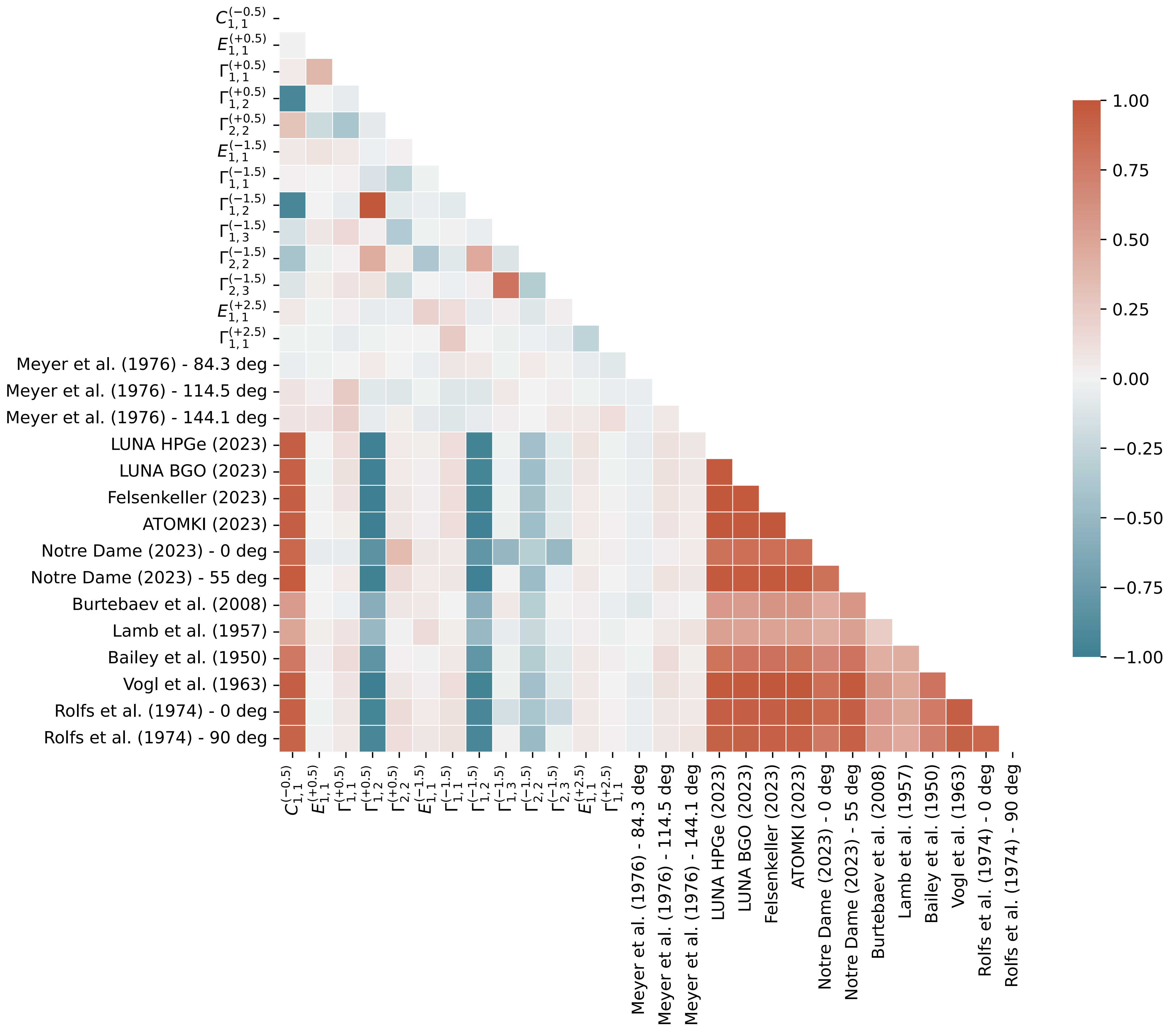
Particle Data Group, Prog. Theor. Exp. Phys. 2022, 083C01

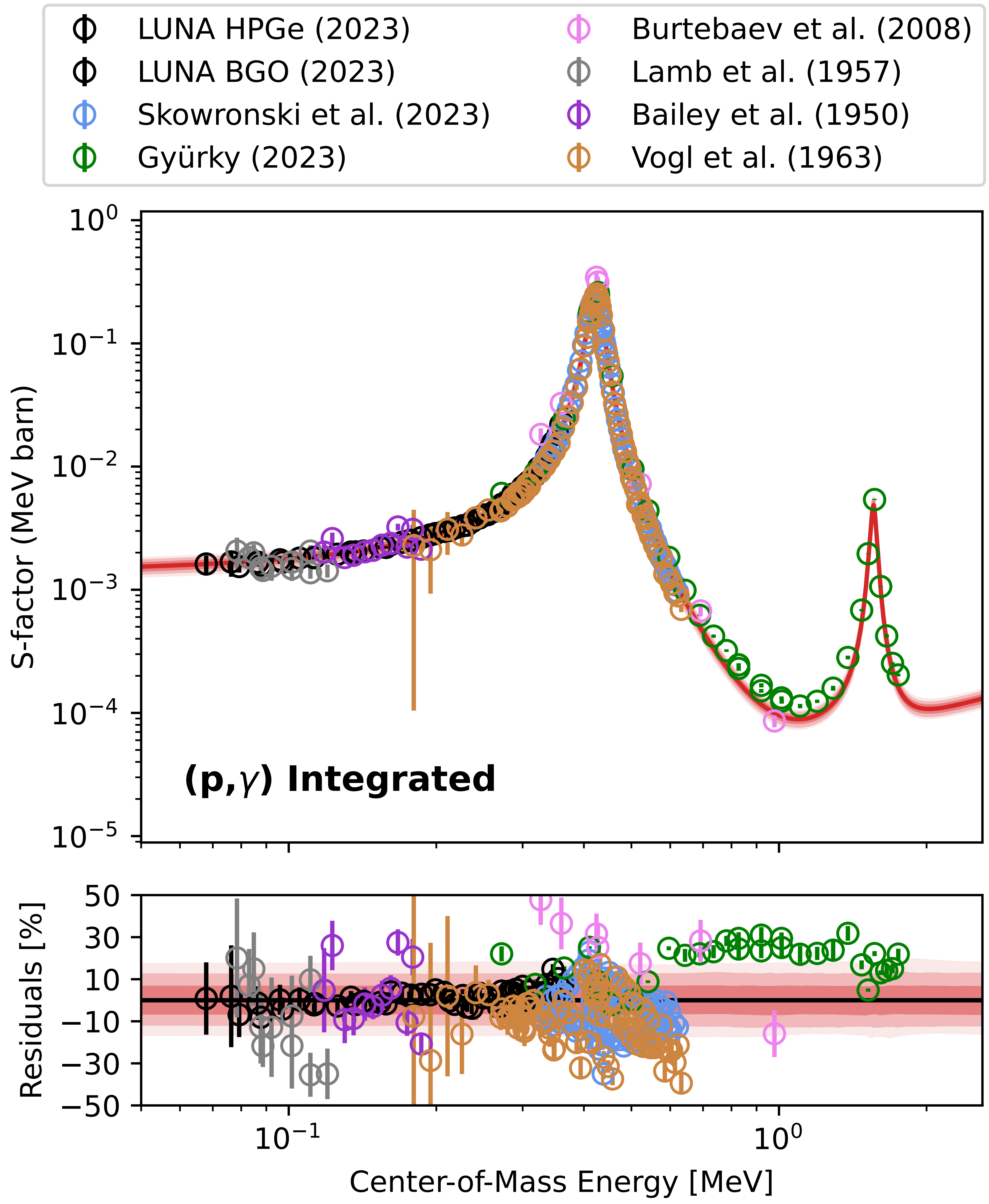
Discrepant datasets → High χ^2 → Small fit uncertainty

Solution: scale the data uncertainty to get $\chi^2_{dof} = 1$

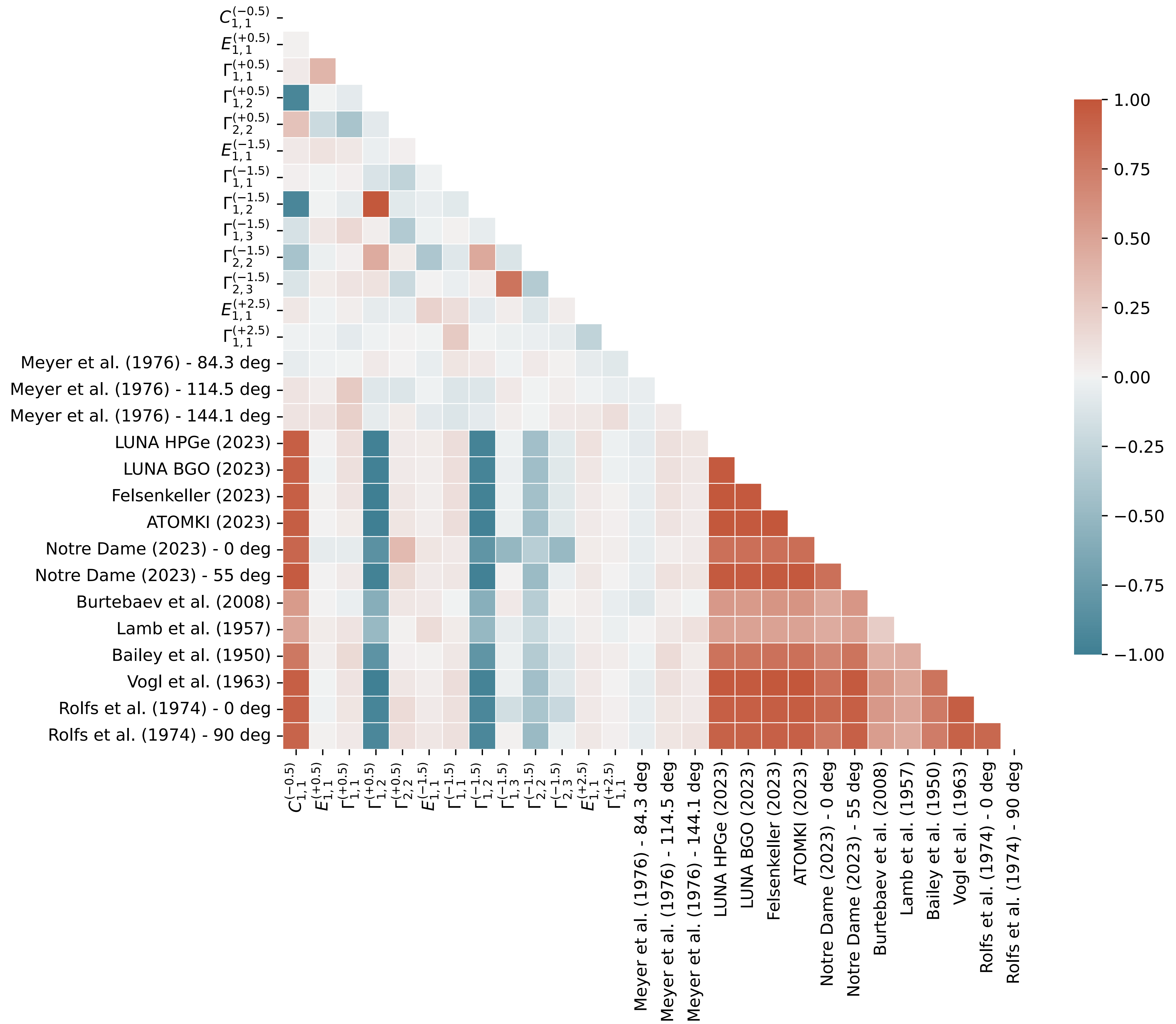
LUNA

Frequentist Fit - $^{12}C(p,\gamma)^{13}N$ (2)



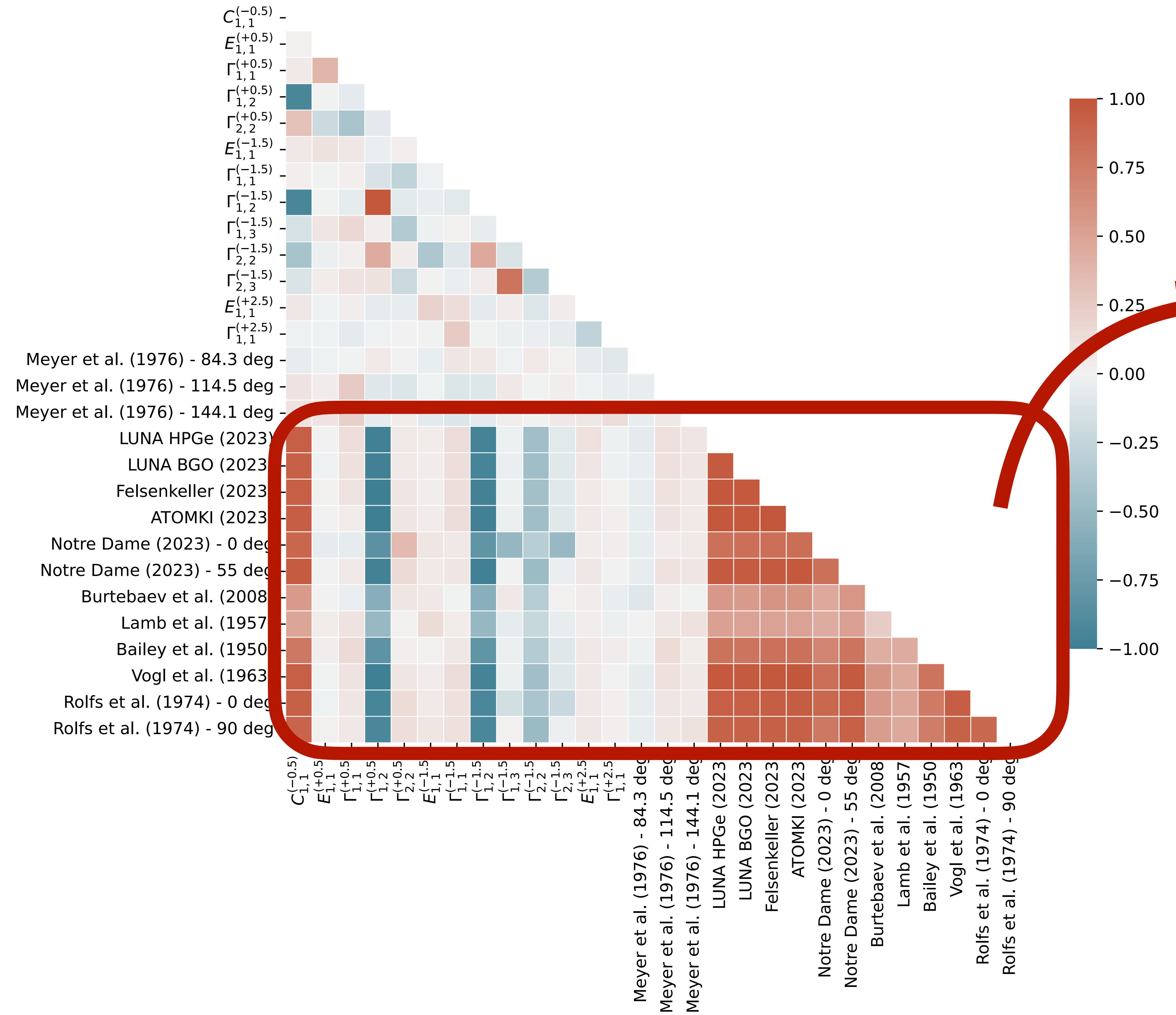


Correlation Matrix



The **correlation matrix** points out an another issue

Correlation Matrix



The **correlation matrix** points out an another issue

↓

The **normalizations** drives all the other parameters

– Correlation coefficients very close to one (greater than 0.99). This indicates both an exceptionally difficult problem, and one which has been badly parametrized so that individual errors are not very meaningful because they are so highly correlated.

Taken from MINUIT manual

LUNA

Correlation Problem

Why huge correlations?

$$\chi^2 = \sum_{i,j} \left(\frac{f_j y_{\text{obs,i}} - y_{\text{theo}}}{f_j \sigma_{\text{stat},i}} \right)^2 + \left(\frac{1 - f_j}{\sigma_{\text{sist}}} \right)^2$$

~ number of data points

~ number of dataset

Correlation Problem

Why huge correlations?

$$\chi^2 = \sum_{i,j} \left(\frac{f_j y_{\mathrm{obs,i}} - y_{\mathrm{theo}}}{f_j \sigma_{\mathrm{stat},i}} \right)^2 + \left(\frac{1 - f_j}{\sigma_{\mathrm{sist}}} \right)^2$$

~ number of data points

~ number of dataset

There is very little constrain on which is the best data normalization…

→ **Many multiple minima exists!**

LUNA

Correlation Problem

Why huge correlations?

$$\chi^2 = \sum_{i,j} \left(\frac{f_j y_{\mathrm{obs,i}} - y_{\mathrm{theo}}}{f_j \sigma_{\mathrm{stat},i}} \right)^2 + \left(\frac{1 - f_j}{\sigma_{\mathrm{sist}}} \right)^2$$

~ number of data points

~ number of dataset

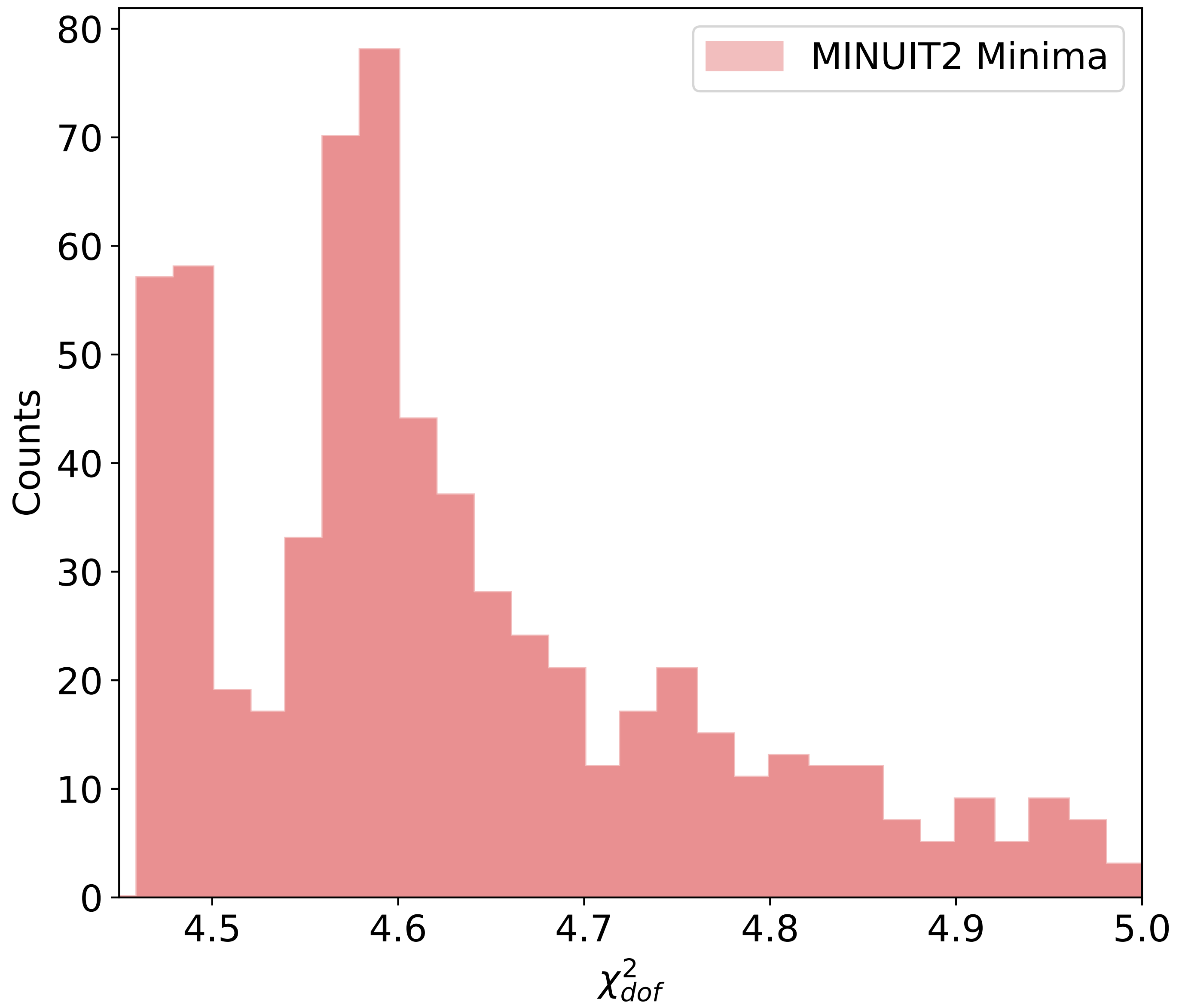
There is very little constrain on which is the best data normalization…

⟶ **Many multiple minima exists!** ⟶ **Sometimes you get a different fit by changing the initial values…**

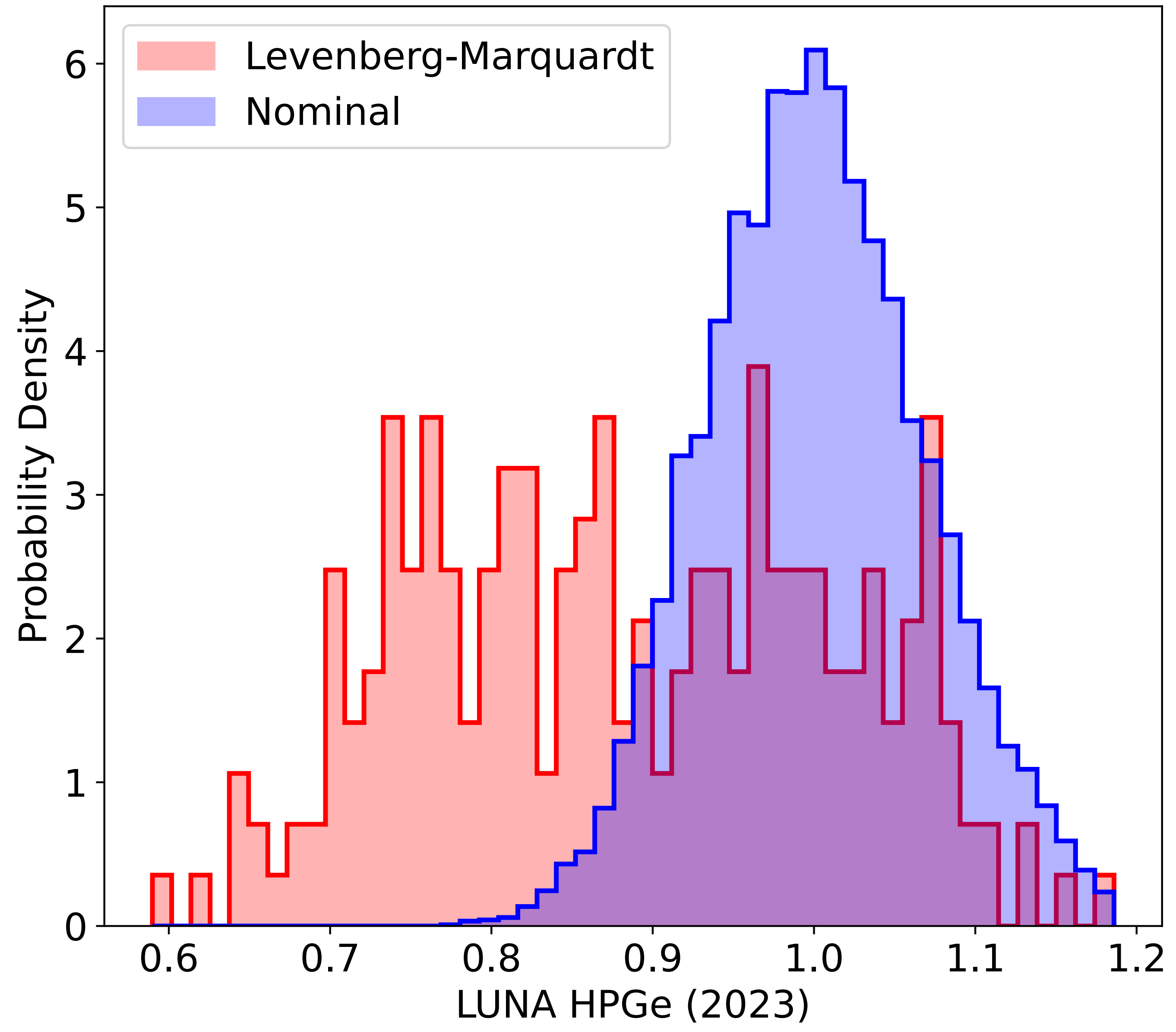
Correlation Problem

All this just by changing the initial parameters of the fit!

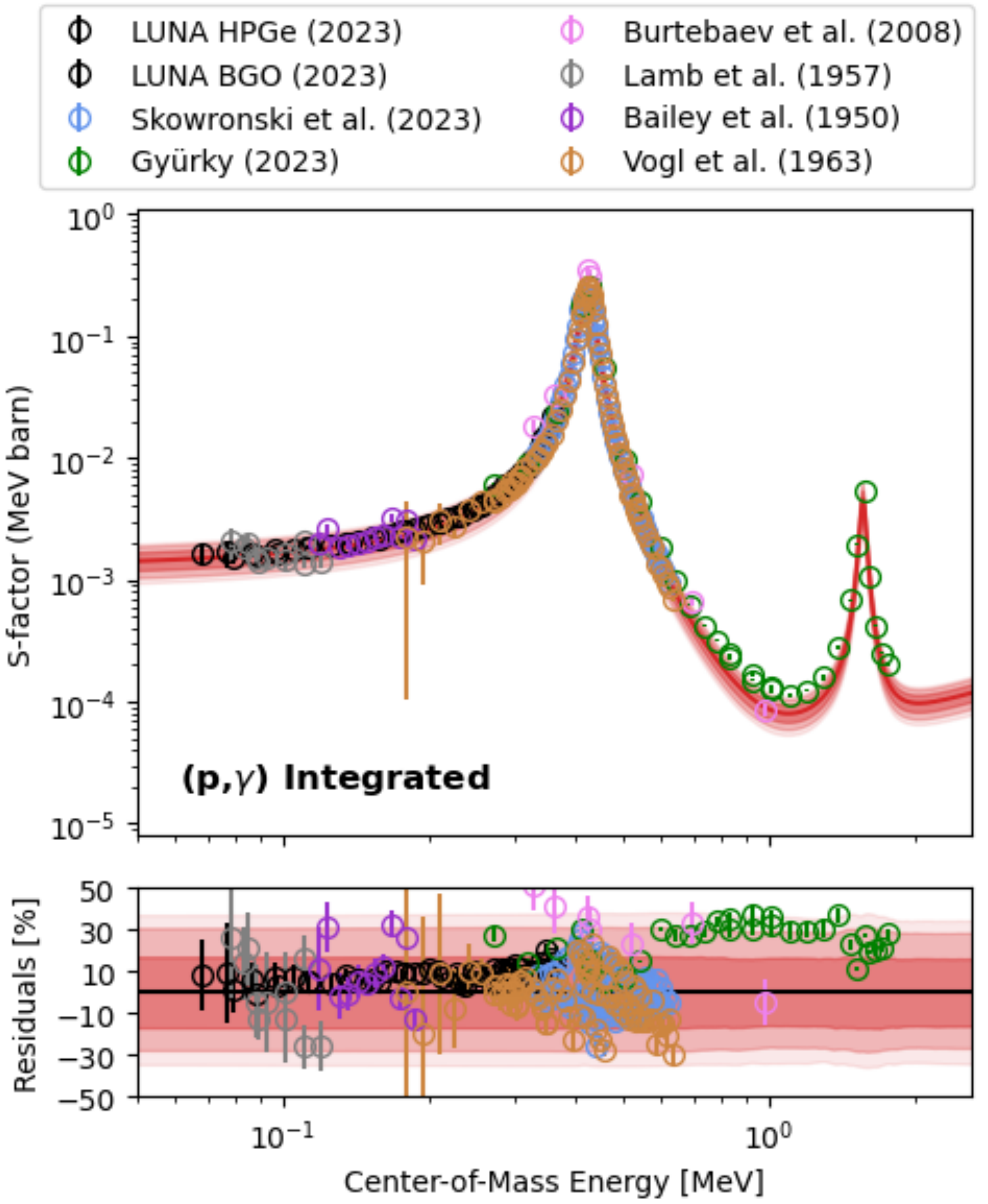
Comparable Minima



Wider Distribution



Huge Uncertainty



Minimisers

MIGRAD can easily stuck in a local minima → **Different initial parameters can give different final result**

Minimisers

MIGRAD can easily stuck in a local minima → **Different initial parameters can give different final result**

However, many other algorithms are available. The **<u>lmfit</u> package** include theme all:

- **method** (*str*, *optional*) –
  Name of the fitting method to use. Valid values are:
  - *'leastsq'*: Levenberg-Marquardt (default)
  - *'least_squares'*: Least-Squares minimization, using Trust Region Reflective method
  - *'differential_evolution'*: differential evolution
  - *'brute'*: brute force method
  - *'basinhopping'*: basinhopping
  - *'ampgo'*: Adaptive Memory Programming for Global Optimization
  - *'nelder'*: Nelder-Mead
  - *'lbfgsb'*: L-BFGS-B
  - *'powell'*: Powell
  - *'cg'*: Conjugate-Gradient
  - *'newton'*: Newton-CG
  - *'cobyla'*: Cobyla
  - *'bfgs'*: BFGS
  - *'tnc'*: Truncated Newton
  - *'trust-ncg'*: Newton-CG trust-region
  - *'trust-exact'*: nearly exact trust-region
  - *'trust-krylov'*: Newton GLTR trust-region
  - *'trust-constr'*: trust-region for constrained optimization
  - *'dogleg'*: Dog-leg trust-region
  - *'slsqp'*: Sequential Linear Squares Programming
  - *'emcee'*: Maximum likelihood via Monte-Carlo Markov Chain
  - *'shgo'*: Simplicial Homology Global Optimization
  - *'dual_annealing'*: Dual Annealing optimization

LUNA

Minimisers

MIGRAD can easily stuck in a local minima → **Different initial parameters can give different final result**

However, many other algorithms are available. The **lmfit package** include theme all:

Automatically scales the covariance

- **method** (*str*, *optional*) –
  Name of the fitting method to use. Valid values are:
  - *'leastsq'*: Levenberg-Marquardt (default)
  - *'least_squares'*: Least-Squares minimization, using Trust Region Reflective method
  - *'differential_evolution'*: differential evolution
  - *'brute'*: brute force method
  - *'basinhopping'*: basinhopping
  - *'ampgo'*: Adaptive Memory Programming for Global Optimization
  - *'nelder'*: Nelder-Mead
  - *'lbfgsb'*: L-BFGS-B
  - *'powell'*: Powell
  - *'cg'*: Conjugate-Gradient
  - *'newton'*: Newton-CG
  - *'cobyla'*: Cobyla
  - *'bfgs'*: BFGS
  - *'tnc'*: Truncated Newton
  - *'trust-ncg'*: Newton-CG trust-region
  - *'trust-exact'*: nearly exact trust-region
  - *'trust-krylov'*: Newton GLTR trust-region
  - *'trust-constr'*: trust-region for constrained optimization
  - *'dogleg'*: Dog-leg trust-region
  - *'slsqp'*: Sequential Linear Squares Programming
  - *'emcee'*: Maximum likelihood via Monte-Carlo Markov Chain
  - *'shgo'*: Simplicial Homology Global Optimization
  - *'dual_annealing'*: Dual Annealing optimization

- Not all find the same minimum
- Some are more consistent than others, ie. are able to find the same minimum…

Frequentist Alternative

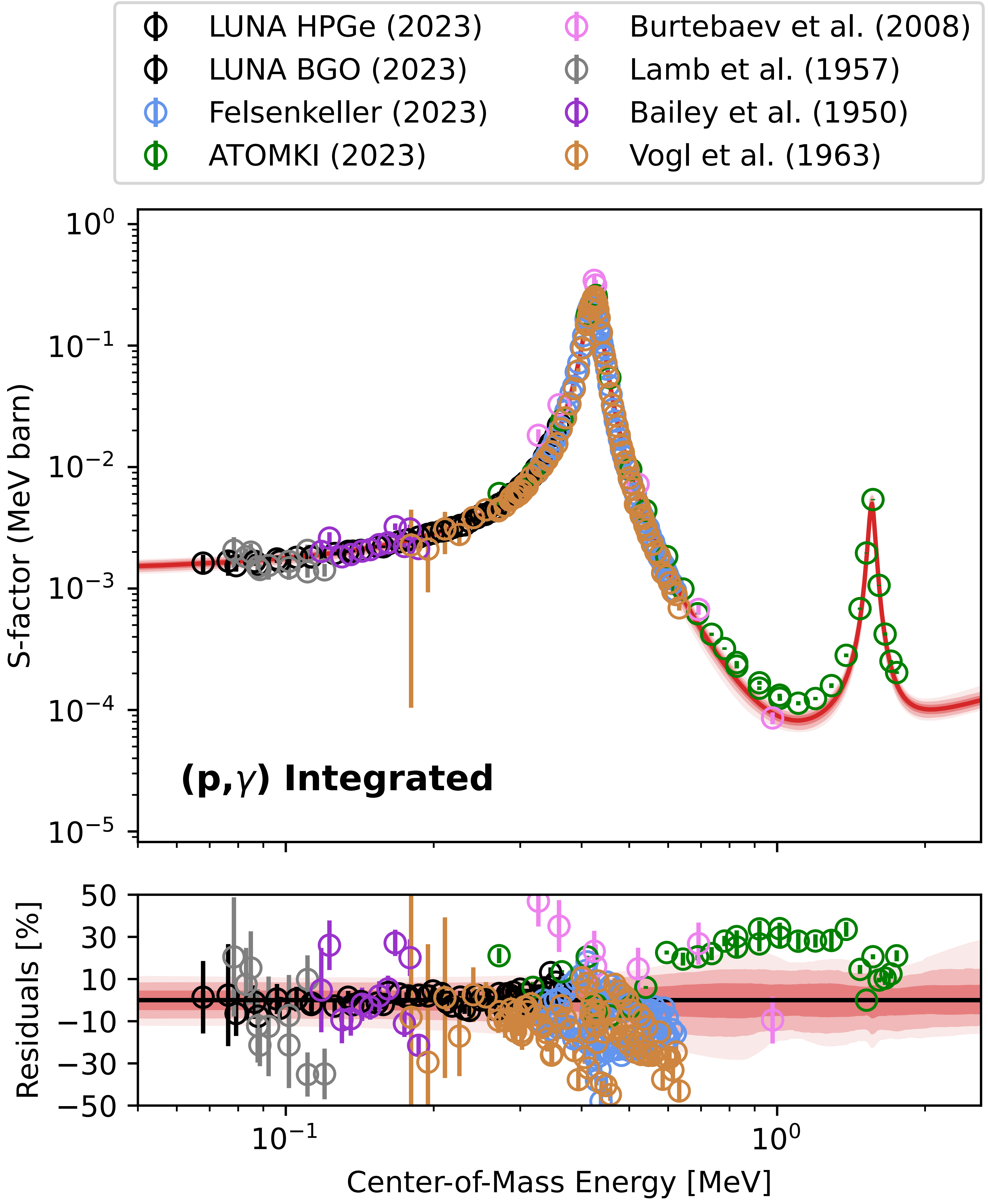
To overcome the previous issues we can:

1. **Sample the data normalizations from systematic error distribution**
2. **Fix the data normalizations to the sampled value**
3. **Run the minimisation**
4. **Repeat 10k times**
5. **Estimate the uncertainty**

Frequentist Alternative

To overcome the previous issues we can:

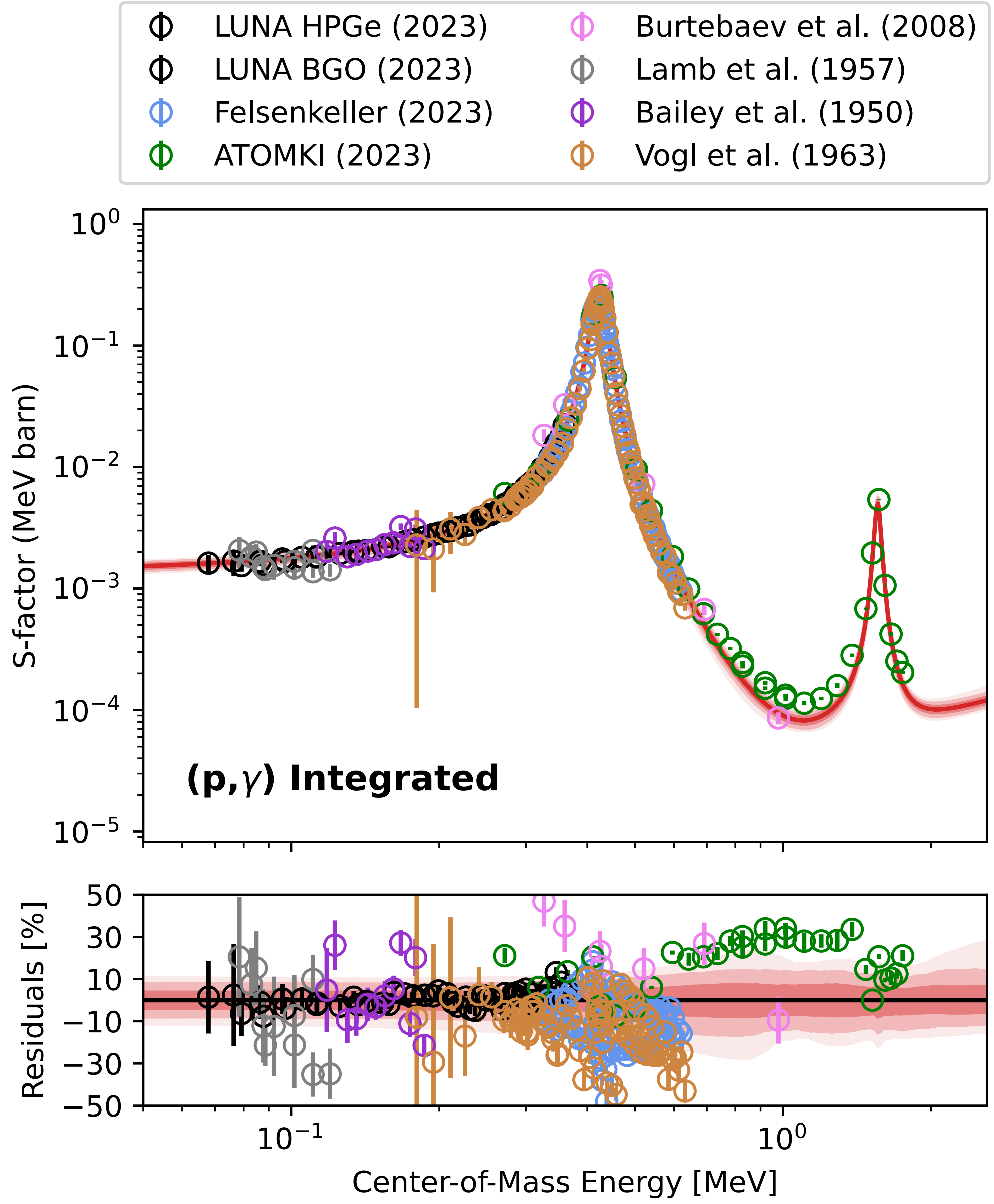
1. **Sample the data normalizations from systematic error distribution**
2. **Fix the data normalizations to the sampled value**
3. **Run the minimisation**
4. **Repeat 10k times**
5. **Estimate the uncertainty**



Frequentist Alternative

To overcome the previous issues we can:

1. **Sample the data normalizations from systematic error distribution**
2. **Fix the data normalizations to the sampled value**
3. **Run the minimisation**
4. **Repeat 10k times**
5. **Estimate the uncertainty**



Advantages	Disadvantages
• No covariance matrix estimation • Unique minimum	• Depends on systematic error evaluation

Bayesian Approach

emcee

<u>emcee</u>: python3 package to perform MCMC (*intended* use of BRICK)

Reference: https://arxiv.org/abs/1202.3665

- Easy to use MCMC implementation
- Widely used in Physics community
- Very efficient, ie. **fast convergence**
- It sets up N **parallel workers** with $N > 2N_p$
- Uses more *complex* algorithm than MH one

Reference: 10.2140/camcos.2010.5.65

emcee

<u>emcee</u>: python3 package to perform MCMC (*intended* use of BRICK)

Reference: https://arxiv.org/abs/1202.3665

In order to use it, we need to change our cost function:

$$\log \mathcal{L} = \sum_i -\frac{1}{2} \log\left(2\pi\sigma^2_{\mathrm{stat,i}}\right) - \frac{1}{2}\left(\frac{f_j y_{\mathrm{obs,i}} - y_{\mathrm{theo}}}{f_j \sigma_{\mathrm{stat},i}}\right)$$

LUNA

emcee

emcee: python3 package to perform MCMC (*intended* use of BRICK)

Reference: https://arxiv.org/abs/1202.3665

In order to use it, we need to change our cost function:

$$\log \mathcal{L} = \sum_i -\frac{1}{2}\log\left(2\pi\sigma^2_{\mathrm{stat,i}}\right) - \frac{1}{2}\left(\frac{f_j y_{\mathrm{obs,i}} - y_{\mathrm{theo}}}{f_j \sigma_{\mathrm{stat},i}}\right)$$

No additional term!

Prior Distributions

Before starting the sampling, the **prior distributions** for our parameters should be selected

Prior Distributions

Before starting the sampling, the **prior distributions** for our parameters should be selected

- Normal/log-normal distributions for **normalization parameters**
- Normal distributions for the **independently measured ANCs**

Informative

- Uniform distributions for **partial widths** of the resonances
- Uniform distributions for **resonance energies**

Uninformative

LUNA

Prior Distributions

Before starting the sampling, the **prior distributions** for our parameters should be selected

- Normal/log-normal distributions for **normalization parameters**
- Normal distributions for the **independently measured ANCs**
- Uniform distributions for **partial widths** of the resonances
- Uniform distributions for **resonance energies**

Informative

Uninformative

$[10^{-12},\ 10^{12}]$ eV

$[E_{res} - 0.1,\ E_{res} + 0.1]$ keV

LUNA

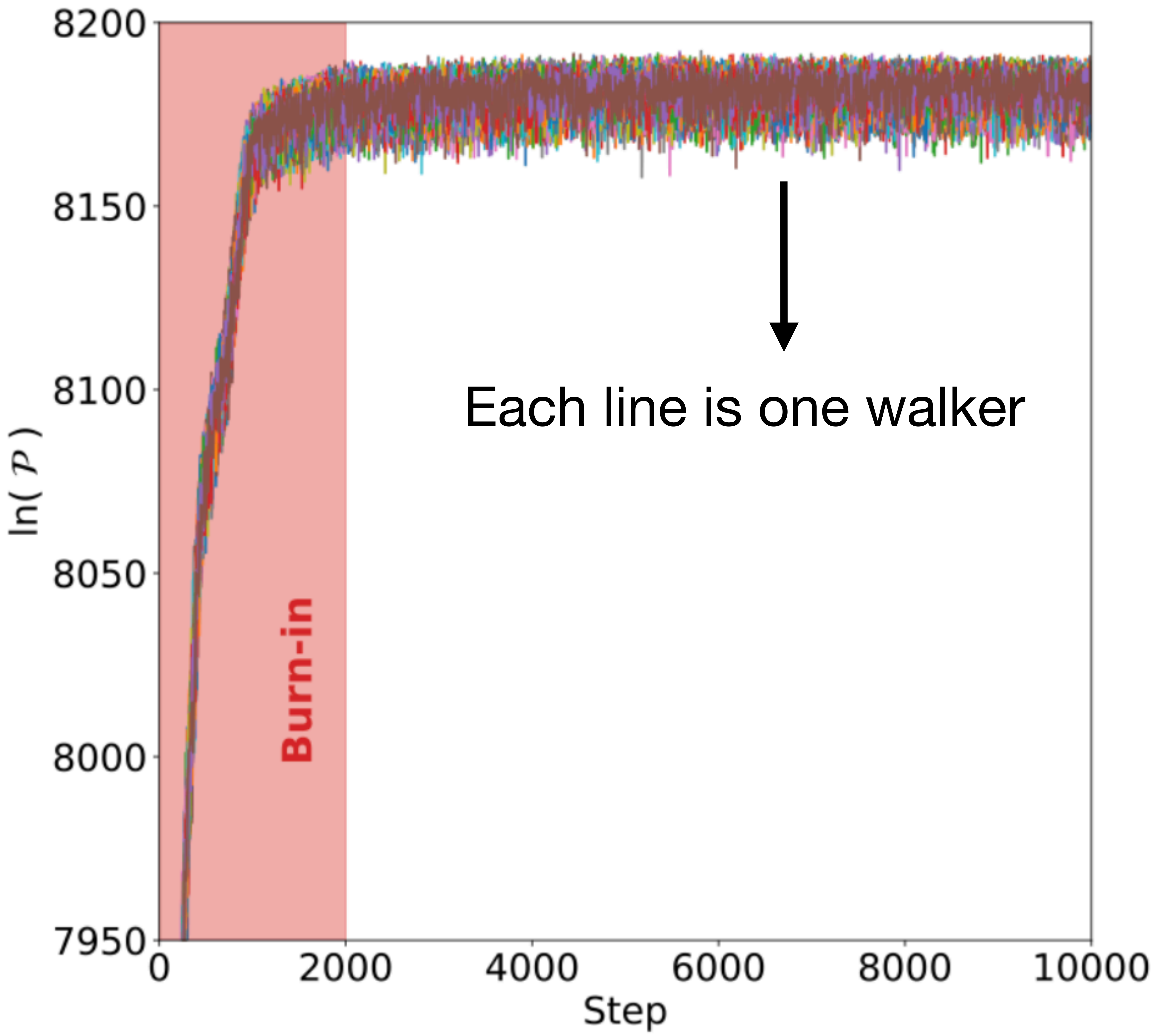
Burn-in

Once the MCMC finishes check the **burn-in** region, ie. the number of samples that the MCMC needs to reach the minimum, and **remove it**

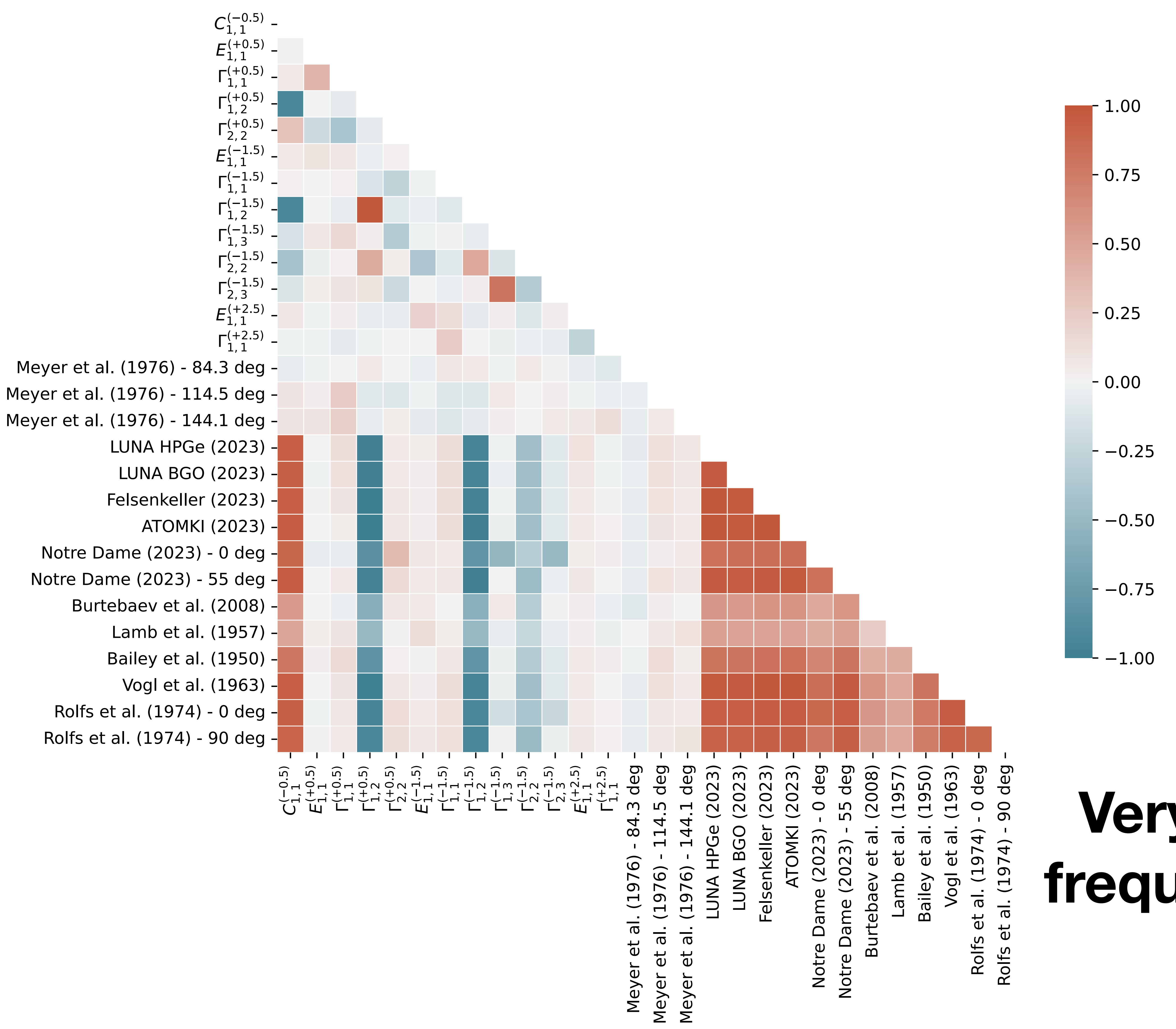
Burn-in

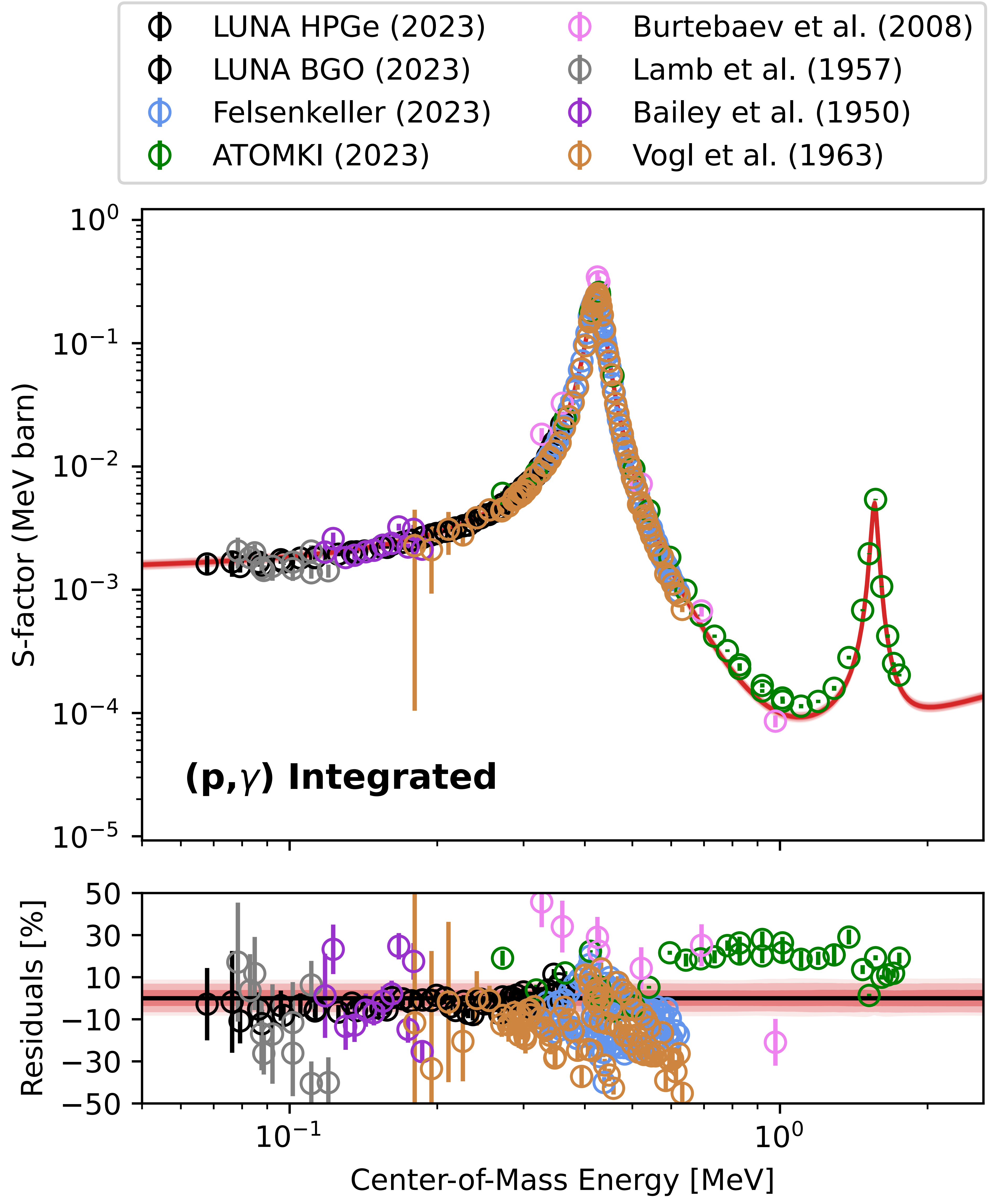
Once the MCMC finishes check the **burn-in** region, ie. the number of samples that the MCMC needs to reach the minimum, and **remove it**

Use all the other ones to estimate the **mean value** and the **uncertainty** of your extrapolation



Bayesian Fit - $^{12}C(p,\gamma)^{13}N$





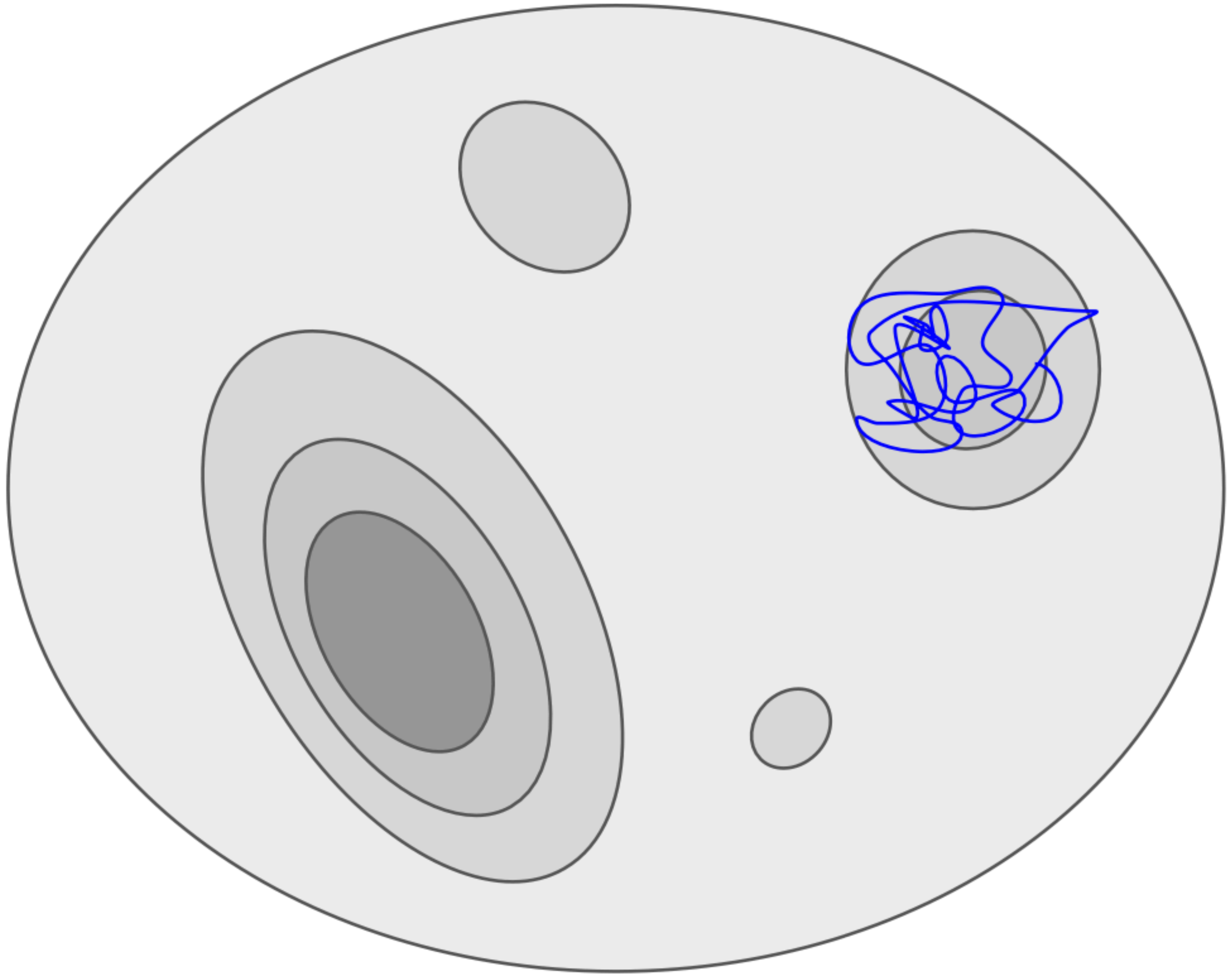
Very similar to frequentist case!

MCMC and Multimodality

MCMC is inherently bad at sampling multimodal distributions

[Interactive Example]

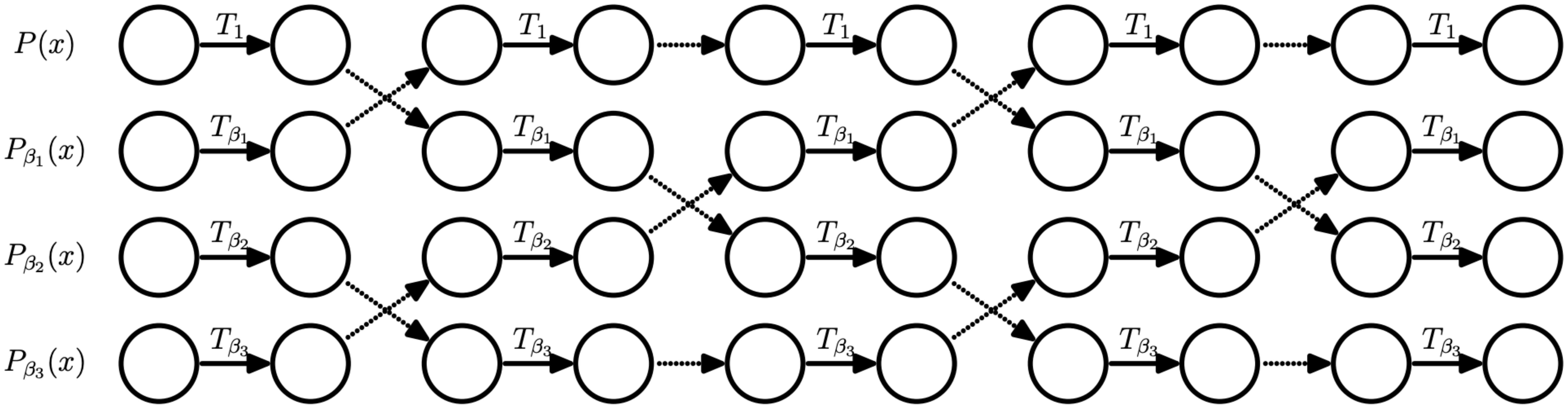
Multimodality



Tempered MCMC

Idea: introduce the concept of **temperature**, T_i, that **broadens** the acceptance distribution

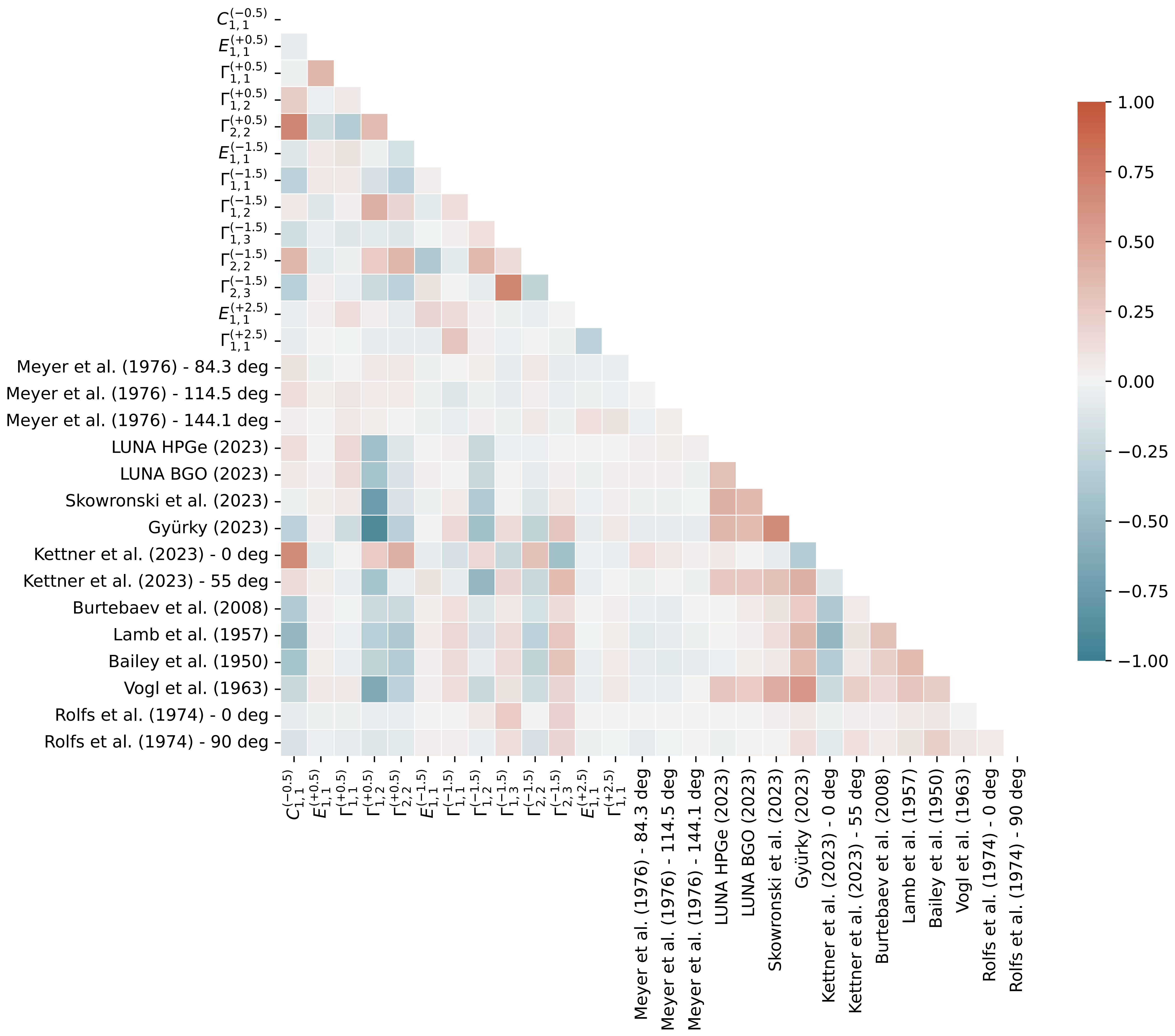
- Start multiple MCMC chains at **different temperatures**
- The cold chains will **explore the minima**, the hot chains will **find the minima**
- During the sampling, **exchange the temperatures** of the chains

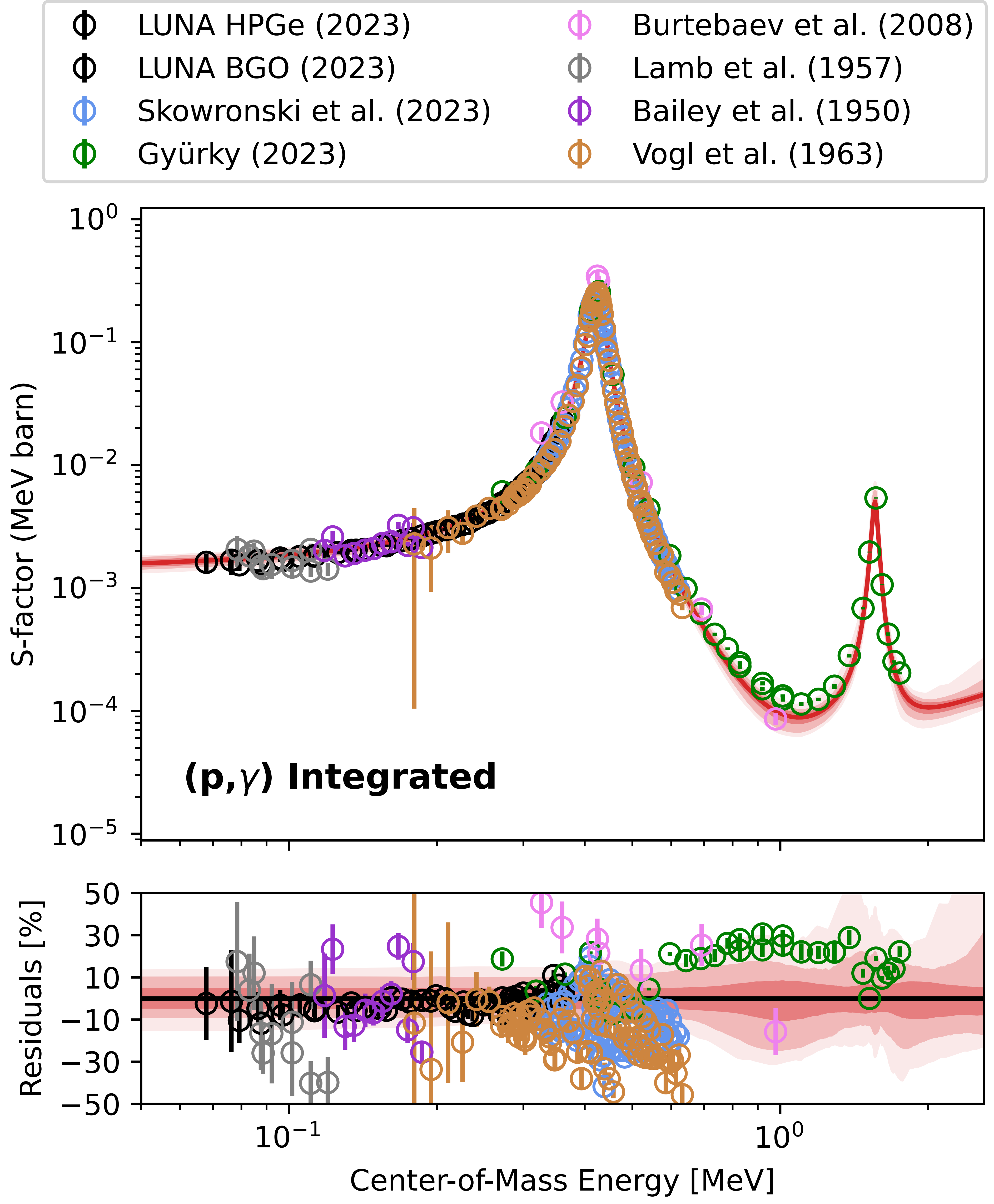


Code: https://github.com/willvousden/ptemcee

Reference: https://arxiv.org/abs/1501.05823

Tempered Bayesian Fit - $^{12}C(p,\gamma)^{13}N$





Conclusions

- Do not take blindly whatever the minimiser gives you
- There is not much difference between the bayesian and the frequentist (as long as we know what we are doing!)
- Multimodality is our worst enemy…
- …but Monte Carlo is our best friend
- python3 gives us all the tools to nicely analyze your data

Thank you for attention!