

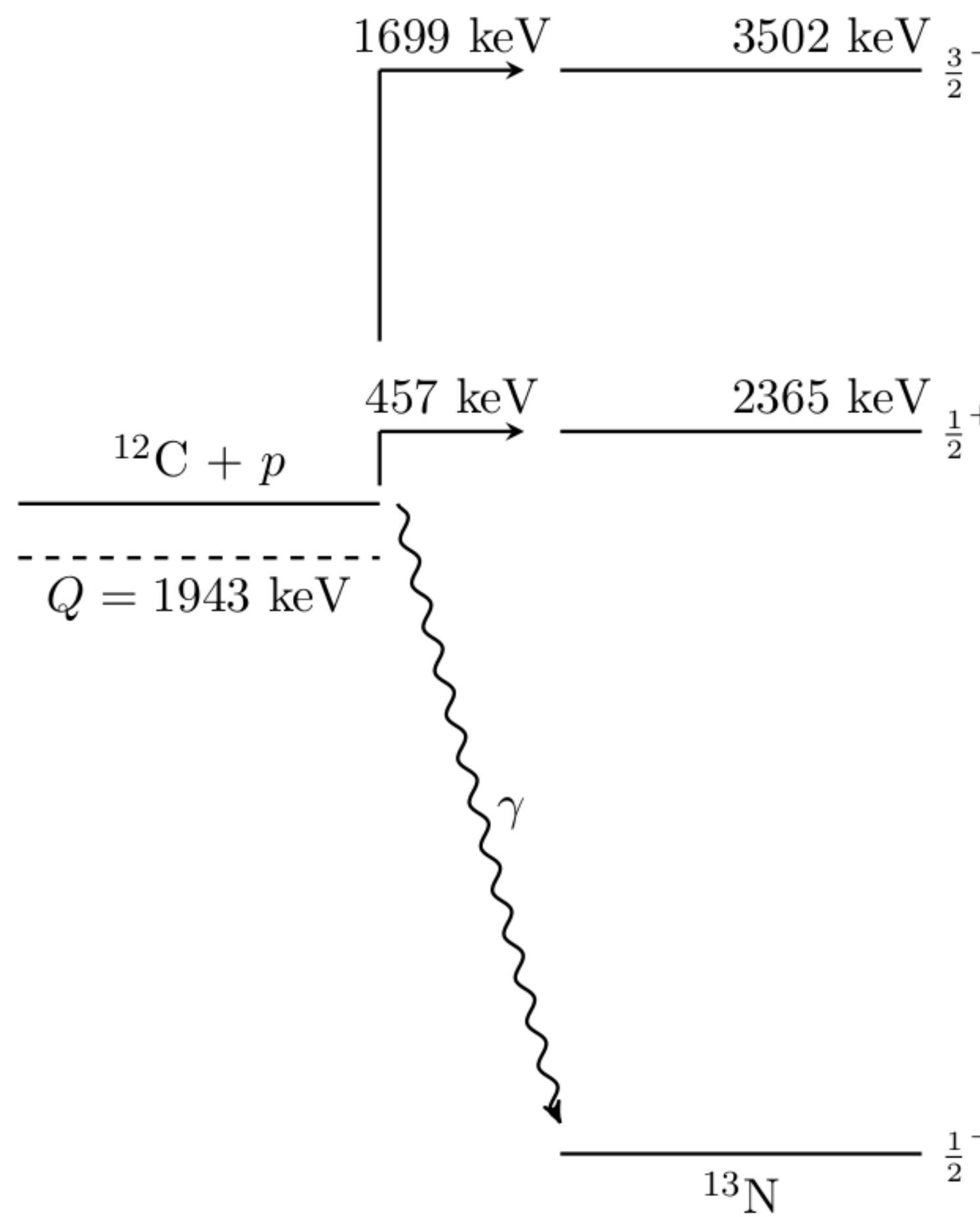
Uncertainty Estimation for R-Matrix Analysis

Jakub Skowronski - Università degli Studi di Padova, INFN Padova

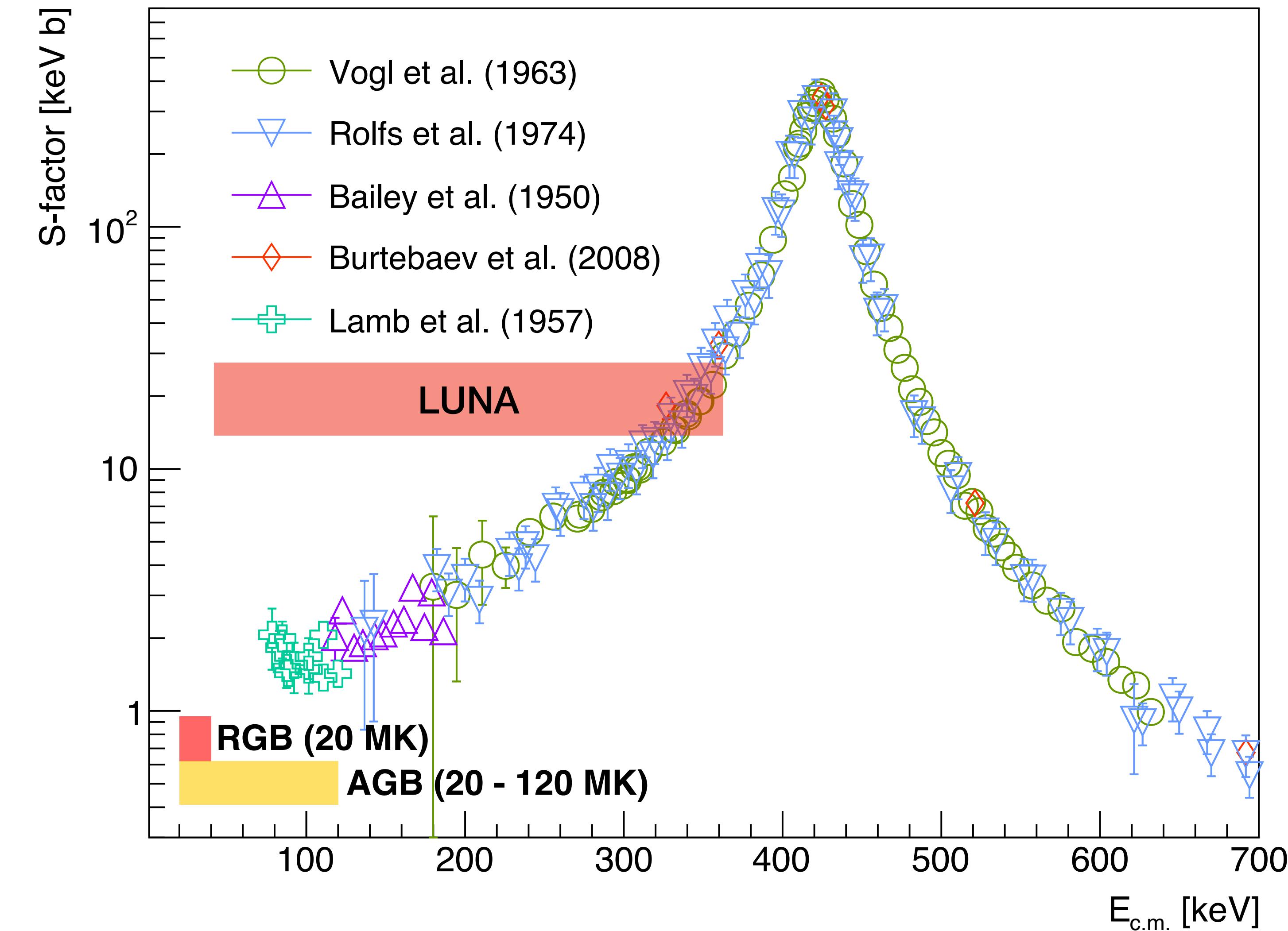
Nuclear Astrophysics PhD. - 08/07/2024



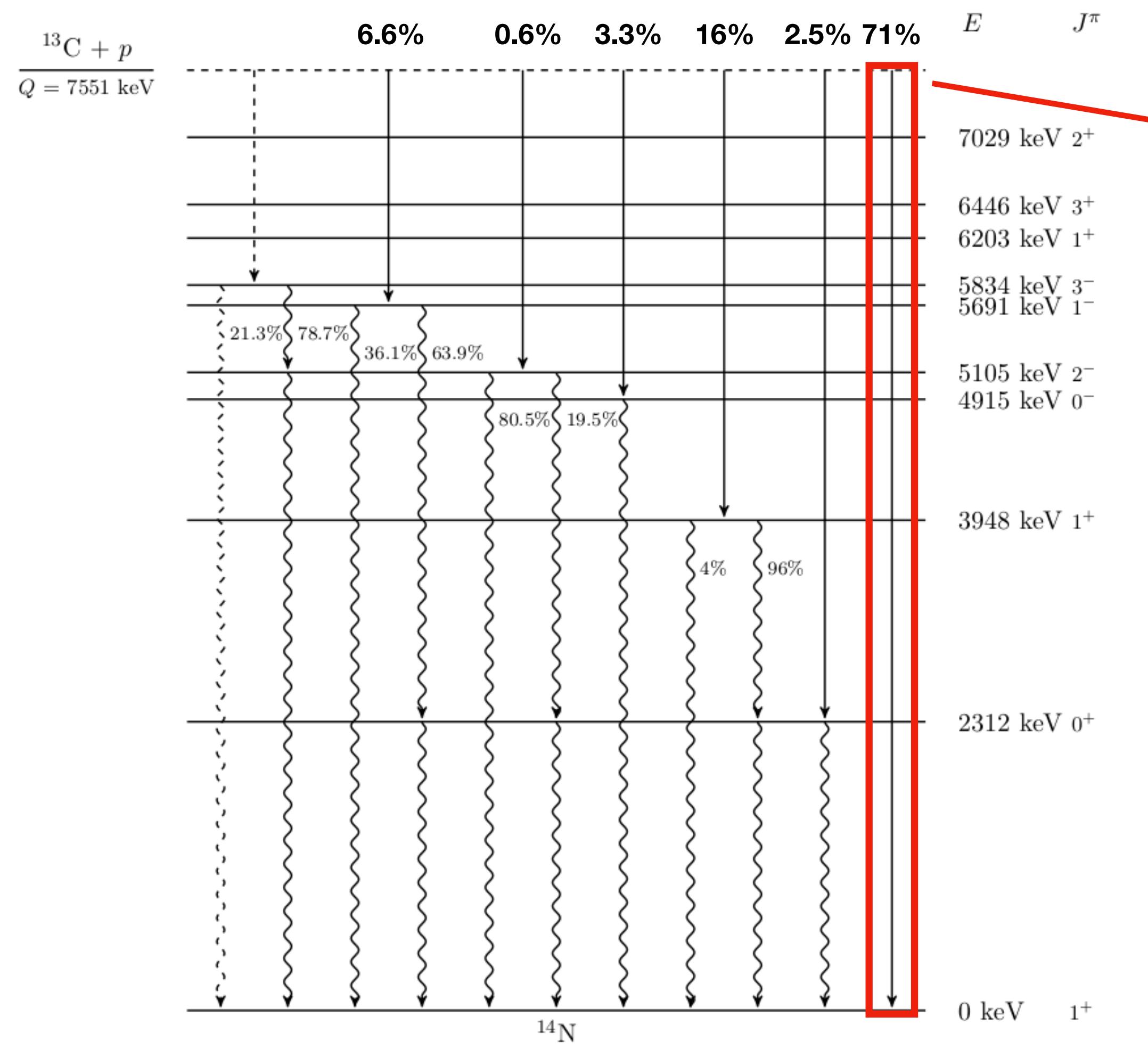
Introduction - $^{12}\text{C}(\text{p},\gamma)^{13}\text{N}$ Reaction



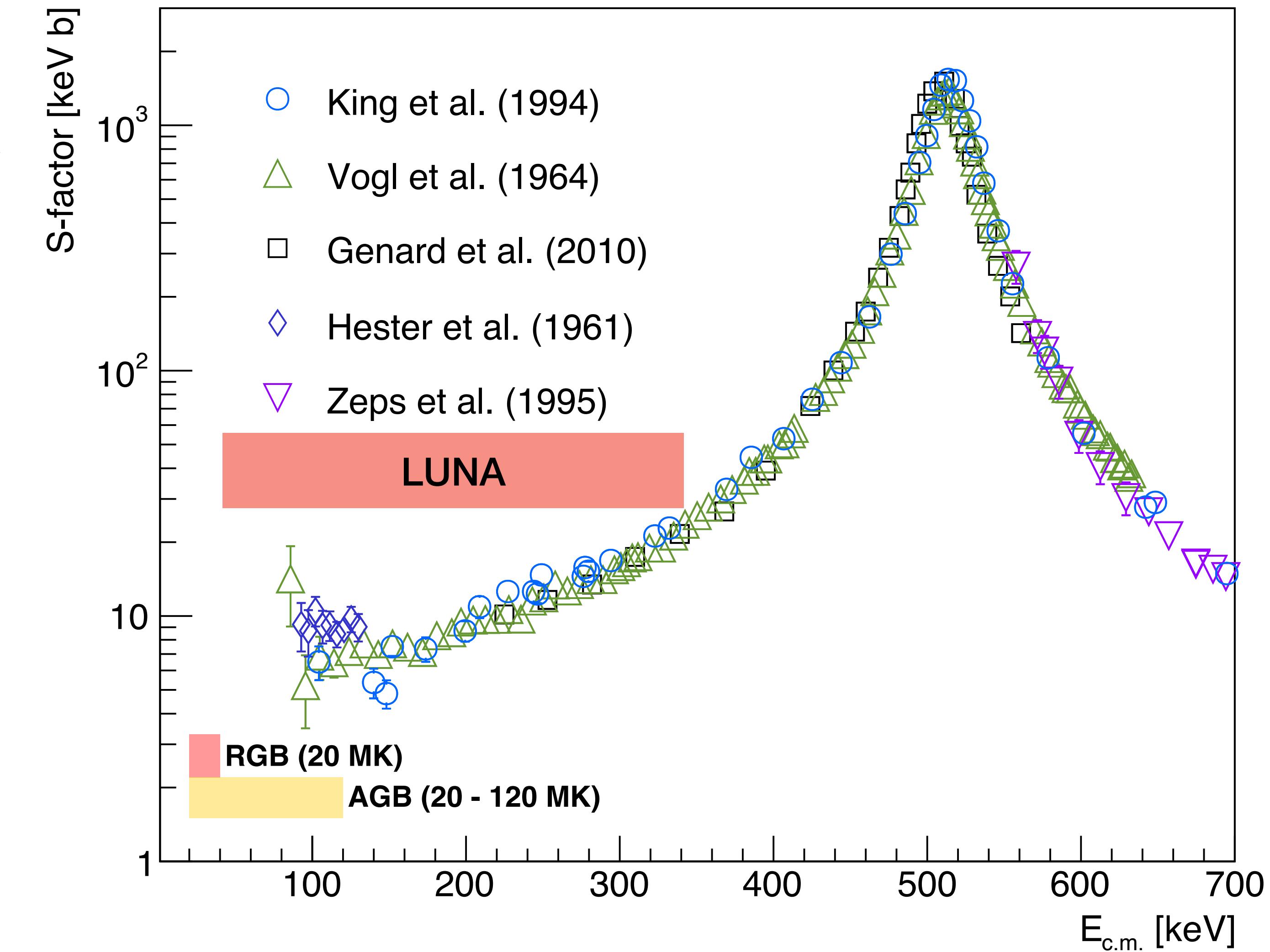
$^{12}\text{C}(\text{p},\gamma)^{13}\text{N}$ - Total Capture



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



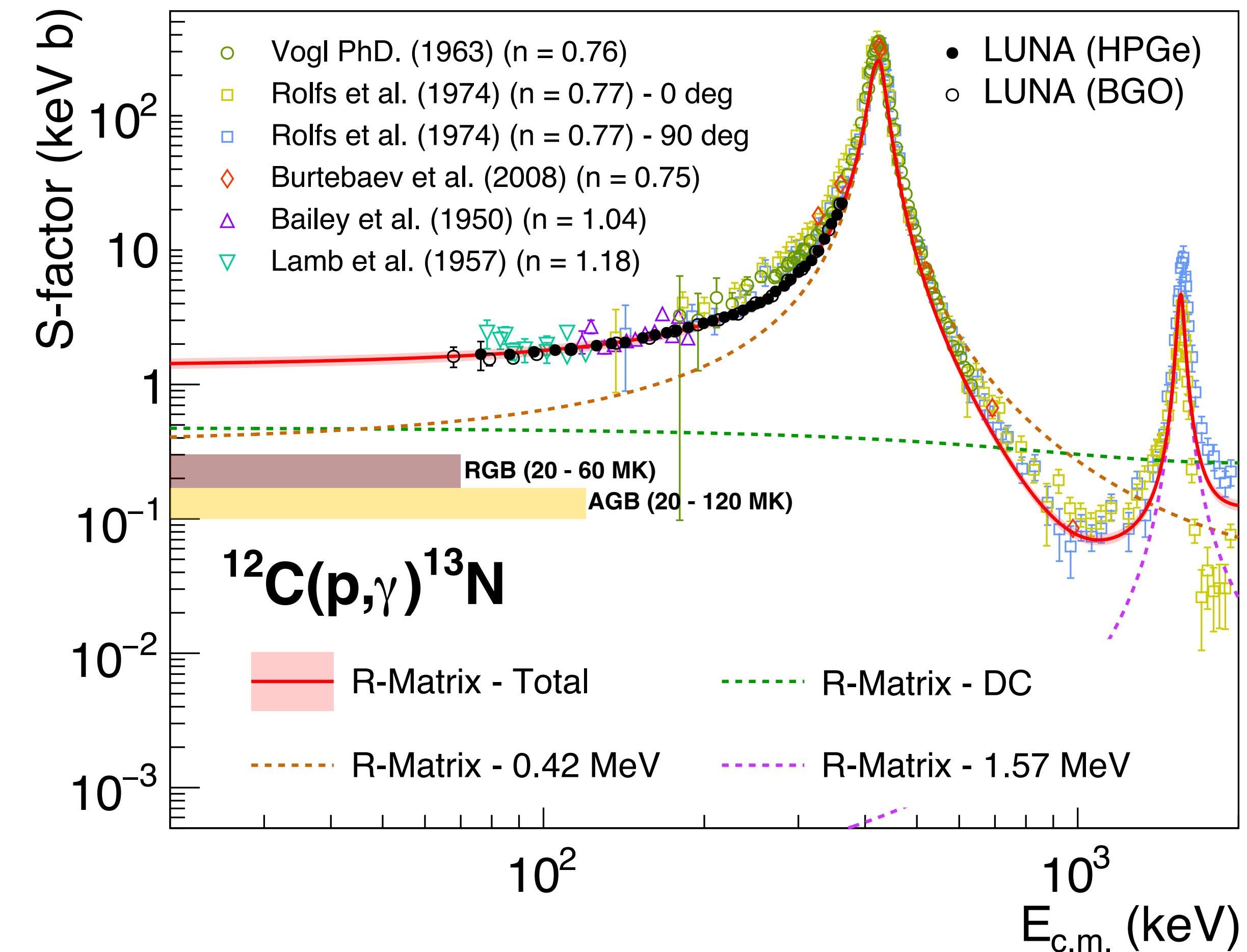
$^{13}\text{C}(\text{p},\gamma)^{14}\text{N}$ - Capture to Ground State



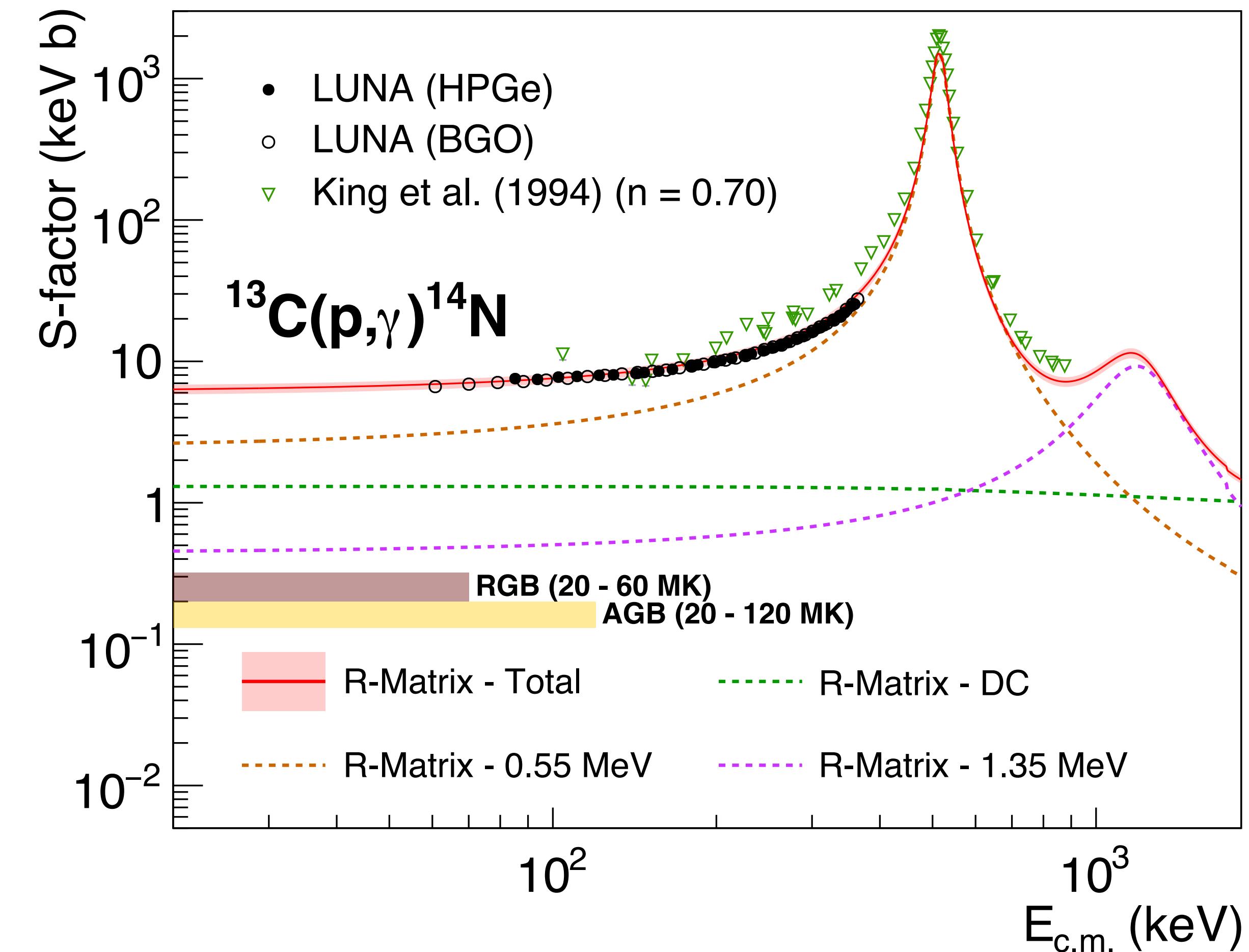
Introduction - Results



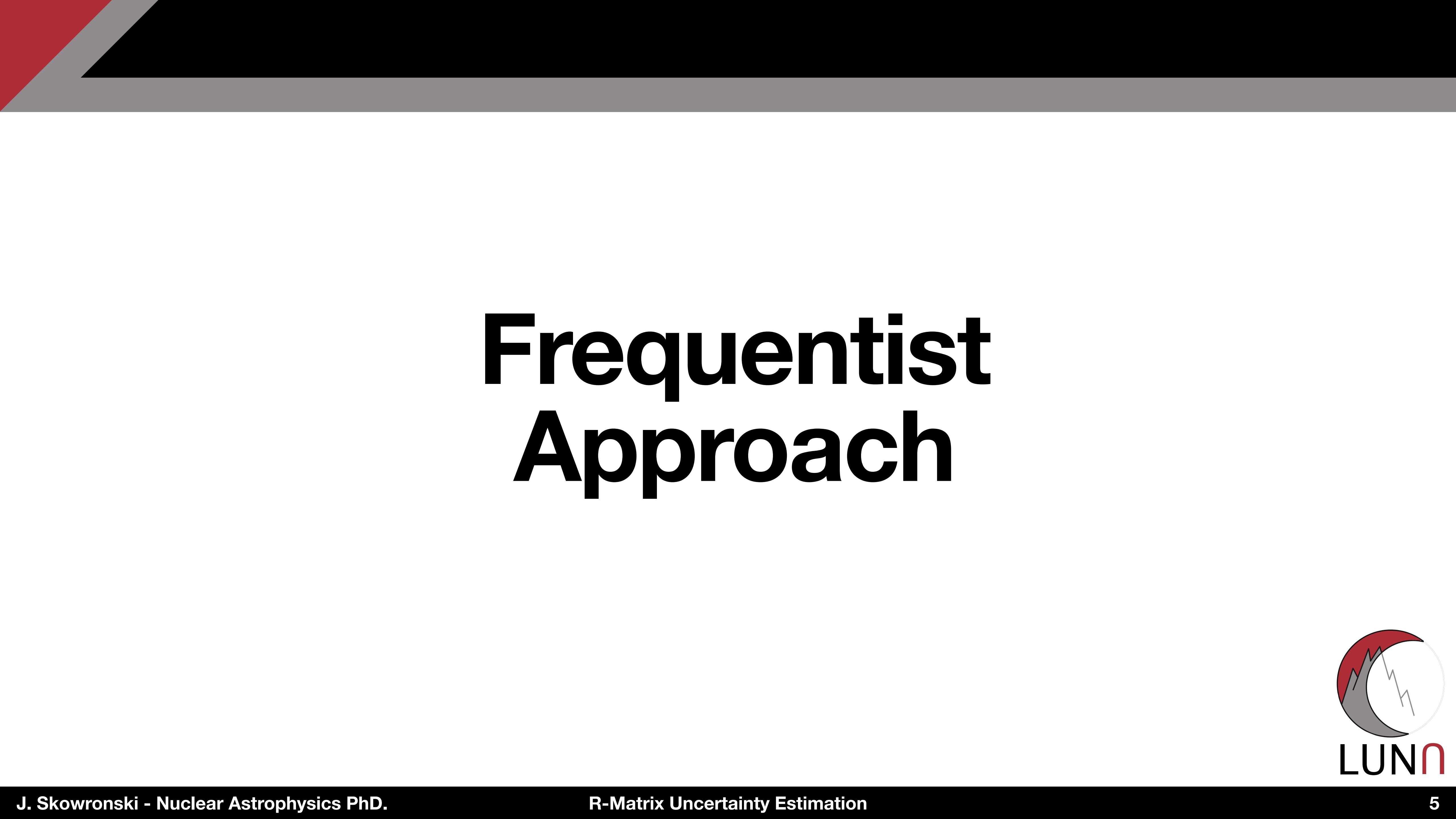
~ 25% lower than literature...



~ 30% lower than literature...

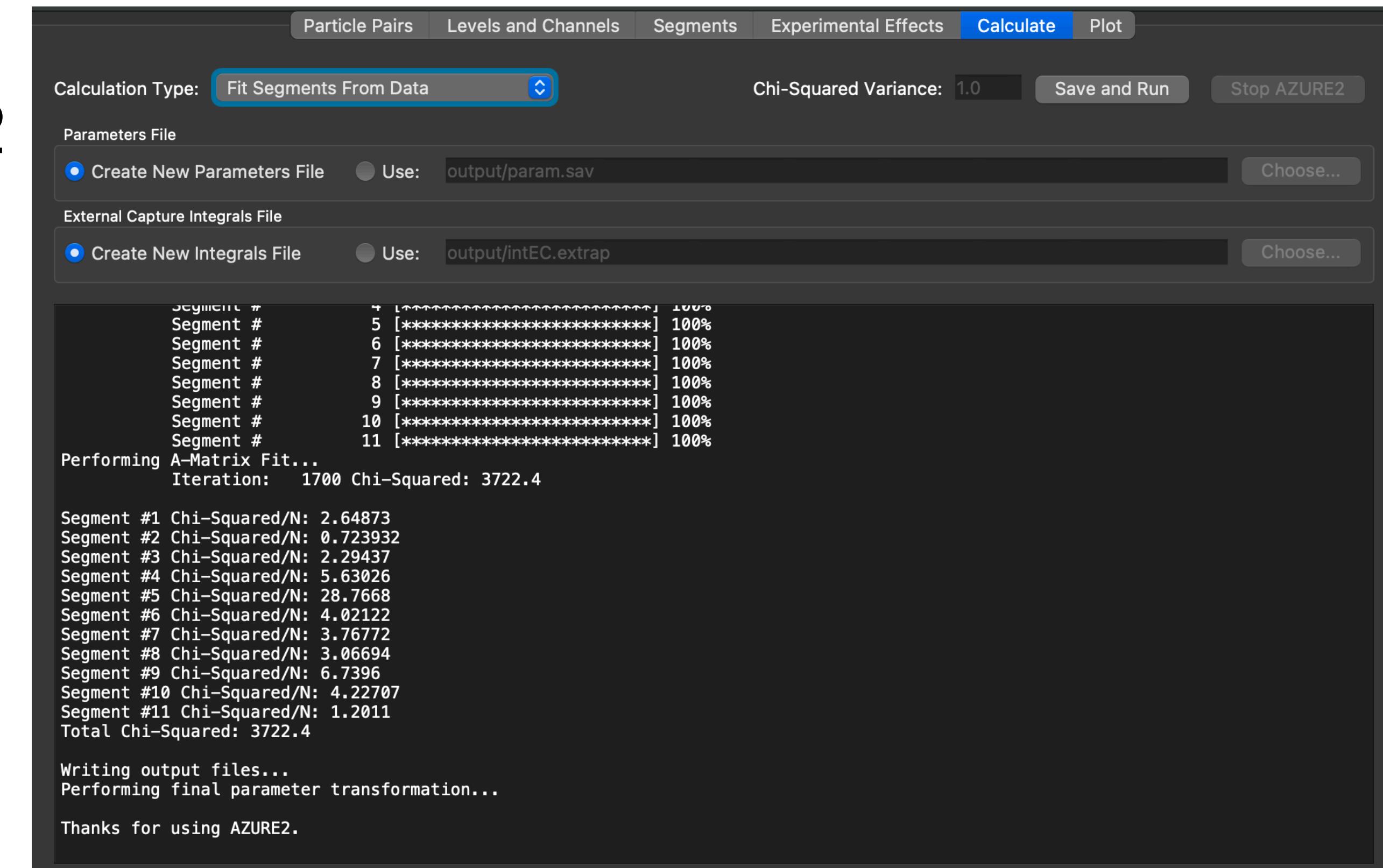


Frequentist Approach



First Attempt with AZURE2

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



The screenshot shows the AZURE2 software interface with a "Fit Segments From Data" calculation type selected. The parameters file is set to "Create New Parameters File" and the external capture integrals file is set to "Create New Integrals File". The output window displays the following text:

```
Segment #      5 [*****] 100%
Segment #      6 [*****] 100%
Segment #      7 [*****] 100%
Segment #      8 [*****] 100%
Segment #      9 [*****] 100%
Segment #     10 [*****] 100%
Segment #     11 [*****] 100%
Performing A-Matrix Fit...
Iteration: 1700 Chi-Squared: 3722.4
Segment #1 Chi-Squared/N: 2.64873
Segment #2 Chi-Squared/N: 0.723932
Segment #3 Chi-Squared/N: 2.29437
Segment #4 Chi-Squared/N: 5.63026
Segment #5 Chi-Squared/N: 28.7668
Segment #6 Chi-Squared/N: 4.02122
Segment #7 Chi-Squared/N: 3.76772
Segment #8 Chi-Squared/N: 3.06694
Segment #9 Chi-Squared/N: 6.7396
Segment #10 Chi-Squared/N: 4.22707
Segment #11 Chi-Squared/N: 1.2011
Total Chi-Squared: 3722.4
Writing output files...
Performing final parameter transformation...
Thanks for using AZURE2.
```

AZURE2

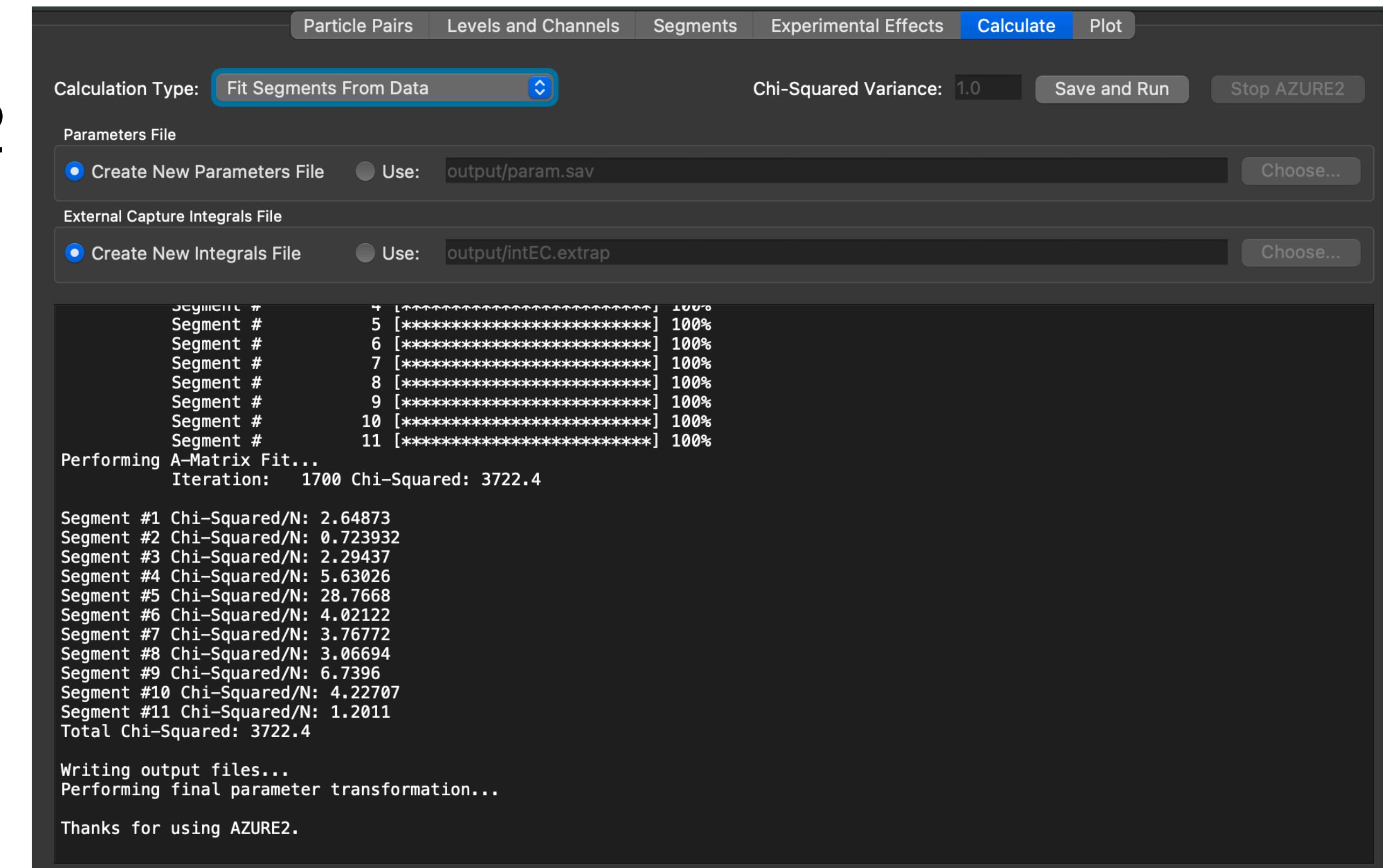


First Attempt with AZURE2

- 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

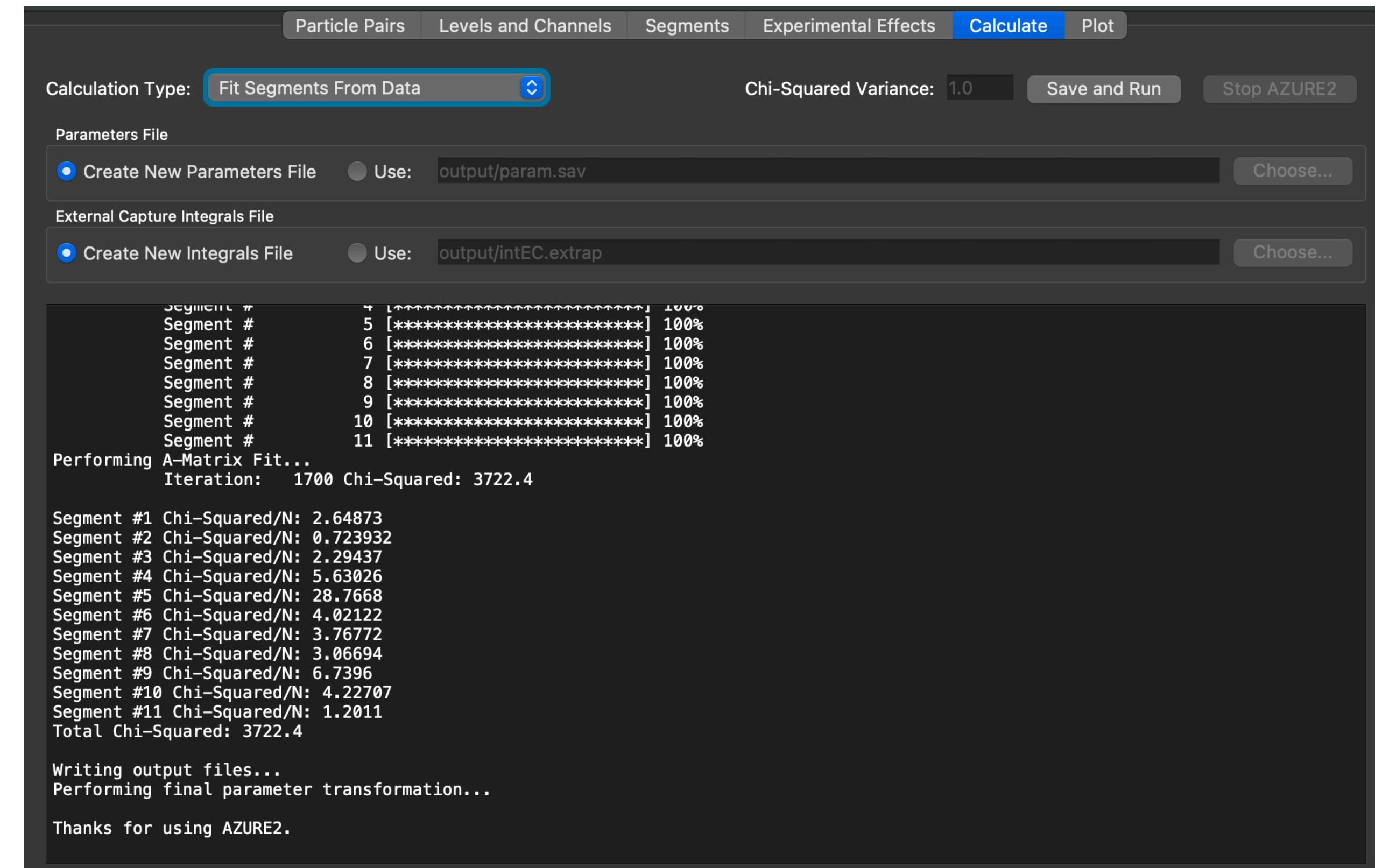


AZURE2



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



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



LUNN

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



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

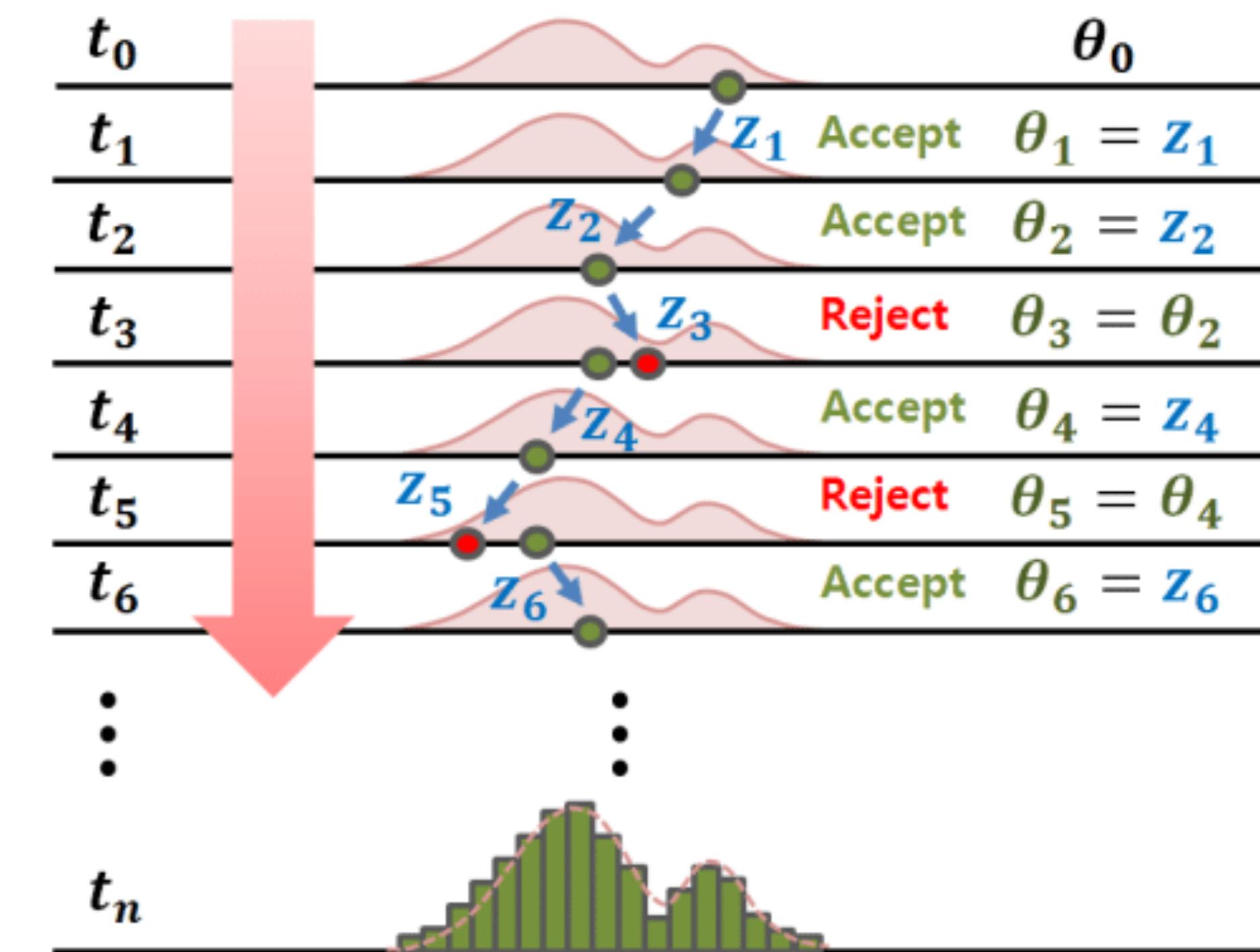


Bayesian R-matrix Inference Code Kit (**BRICK**) published in 2021

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

Advantages

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



LUNN

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**



LUNN

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

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

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



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 easily sample your parameters

ANC ~ Gaussian(μ , σ)



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



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



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

Disadvantages

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

AZURE2 = R-Matrix Calculator



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

<https://github.com/skowrons94/r-matrix-school-2024>

Exercise n. 0

- 1) Try to run BRICK and check if everything is working.
- 2) Play with the interactive plotting.



Frequentist Attempt

We can define our function to minimise as follows:

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

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



LUNN

Frequentist Attempt

Let's define our function to minimise as follows:

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

doi: [10.1016/0168-9002\(94\)90719-6](https://doi.org/10.1016/0168-9002(94)90719-6)

Then can try to minimise it with **iminuit** package (<https://scikit-hep.org/iminuit/>):

- MINUIT2 implementation for python3
- We can define **limits for parameters** (not possible in AZURE2)
- We can launch **many minimisation in parallel** (not possible in AZURE2)



Frequentist Uncertainty (1)

MINUIT2 will give you the **parameter uncertainties** from the **covariance matrix** but...

How to get cross section uncertainty?



Frequentist Uncertainty (1)

MINUIT2 will give you the **parameter uncertainties** from the **covariance matrix** but...

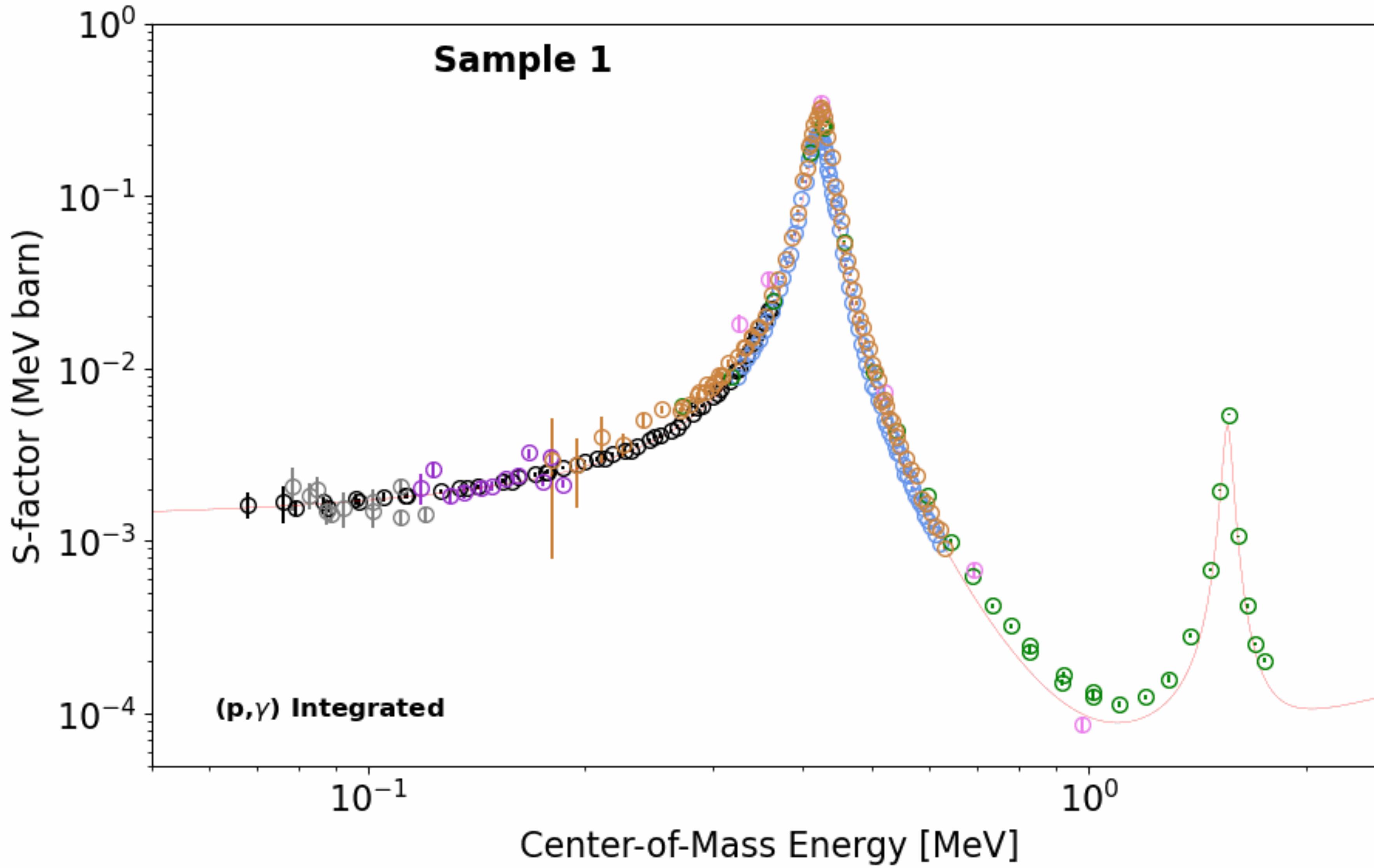
How to get cross section uncertainty?

1. Get the **covariance matrix** of the parameters
2. Construct the **multivariate normal distribution** from it
3. Take **10k samples** of your parameters
4. Calculate the cross section for each sample
5. Calculate the errors as the **percentiles** of your cross sections



LUNN

Frequentist Uncertainty (1)



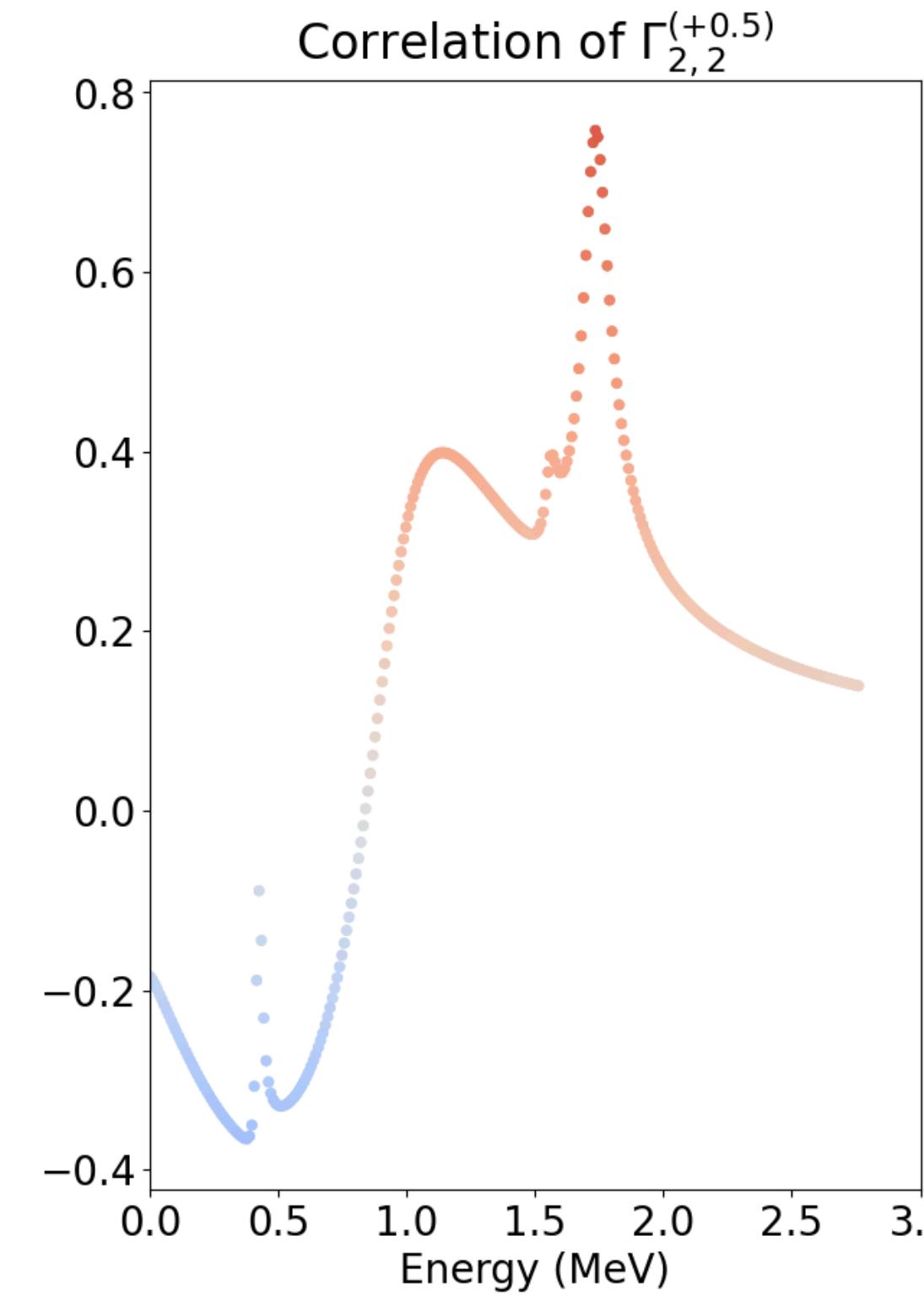
Monte Carlo gives you power!

- Study the extrapolation distribution

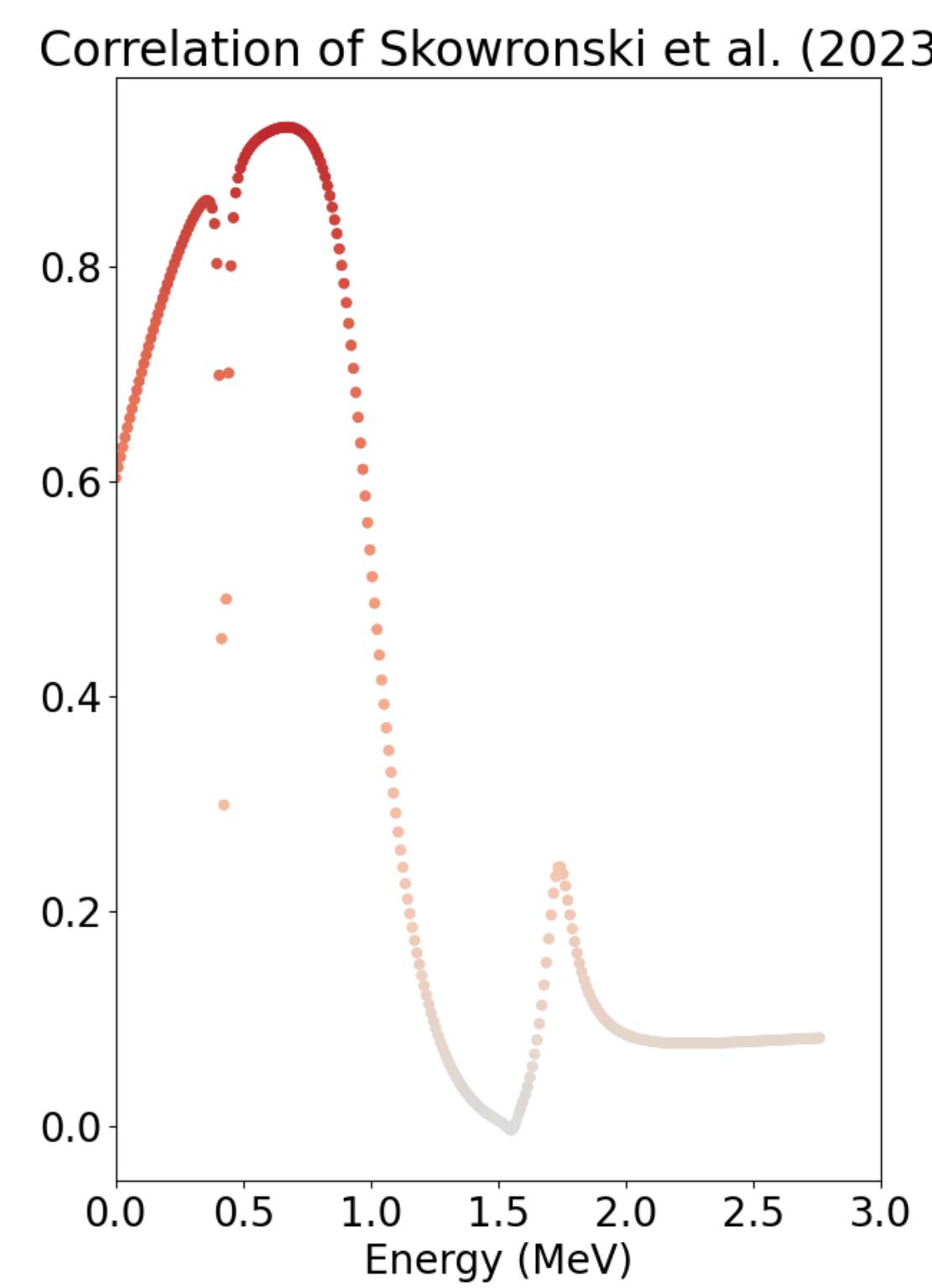


Frequentist Uncertainty (1)

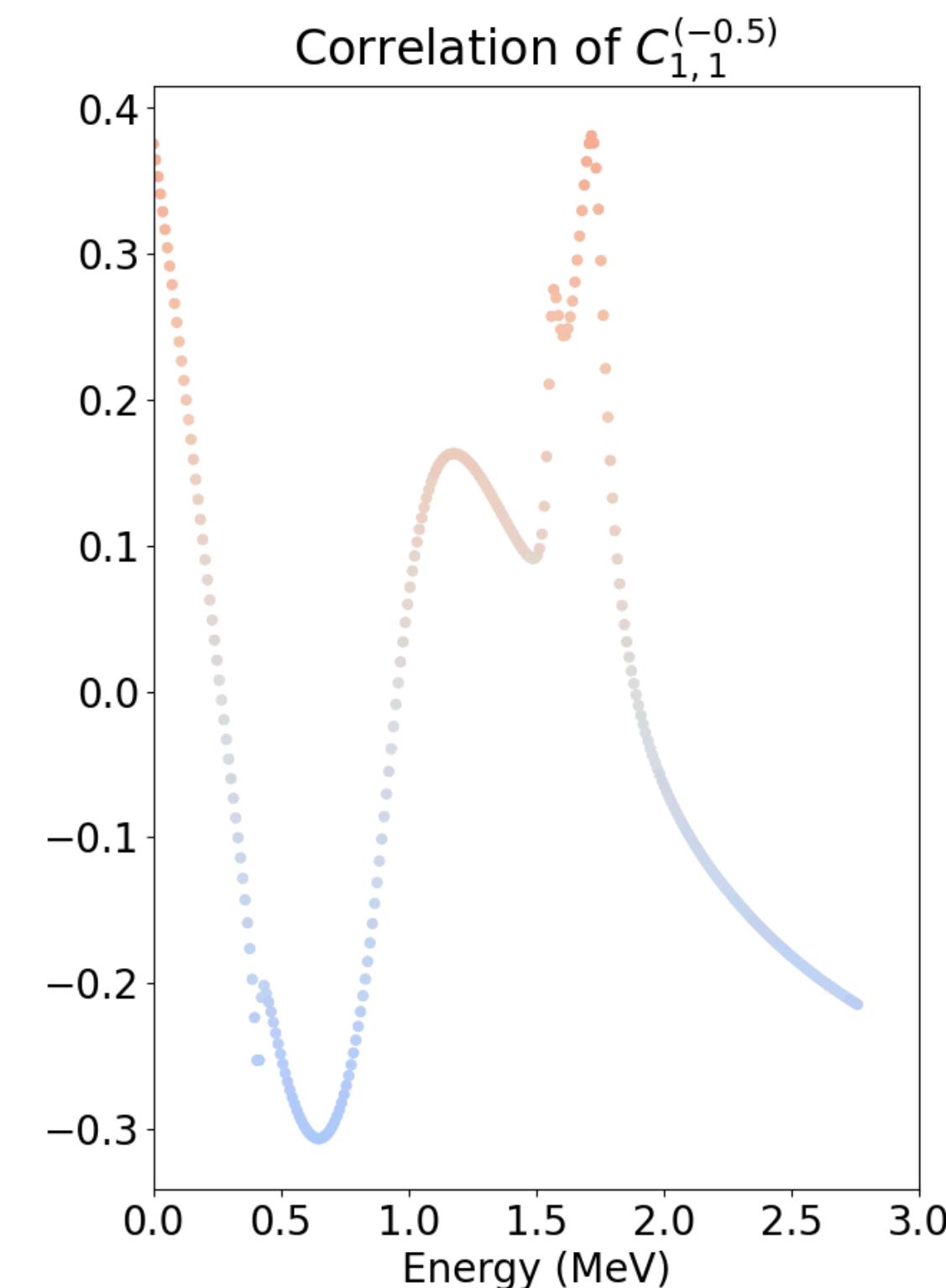
Background Pole



Data Normalization



ANC



Monte Carlo gives you power!

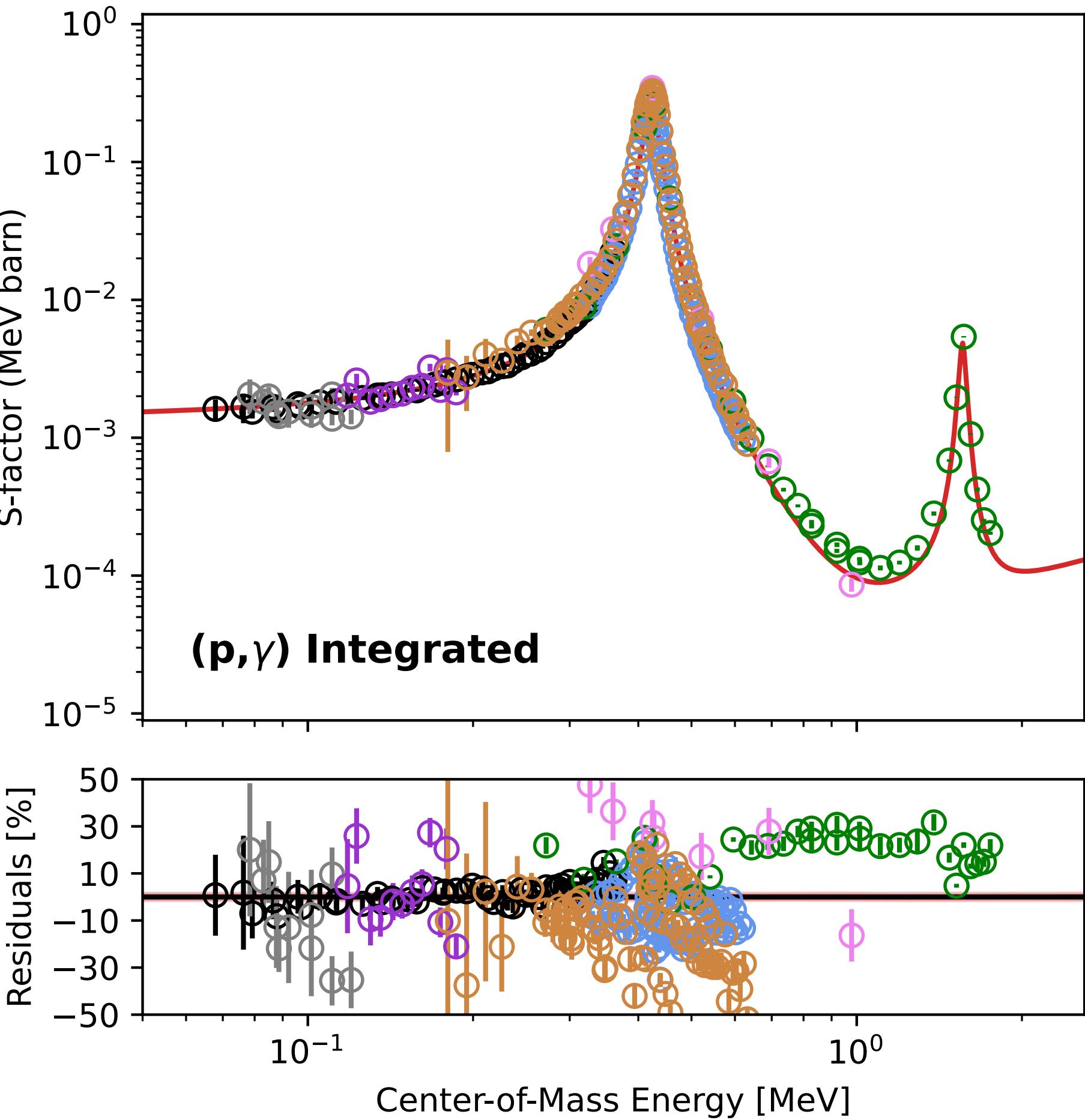
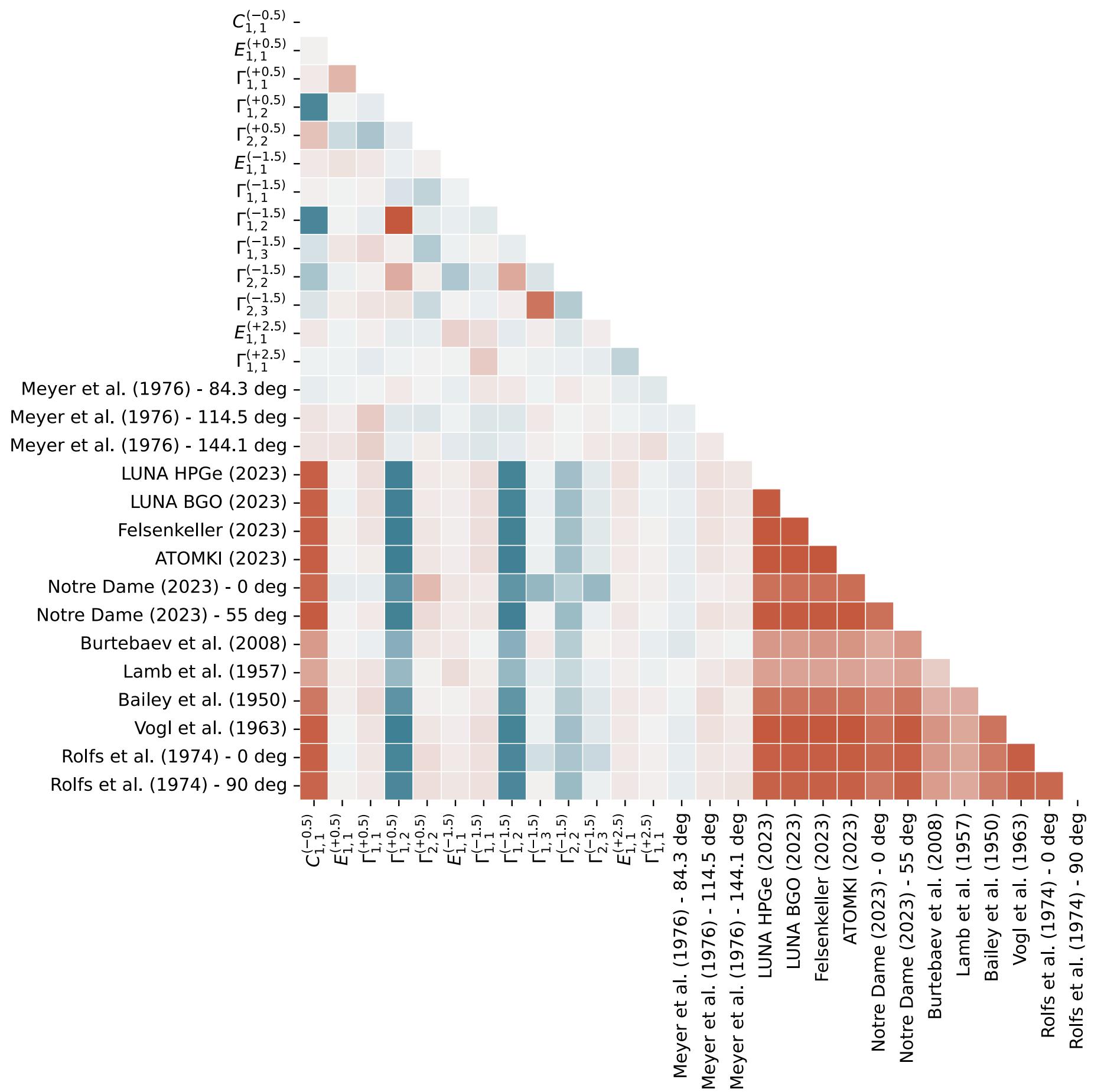
- Study the extrapolation distribution
- Study the correlation between cross section and different parameters



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

What happened when I applied MINUIT2?

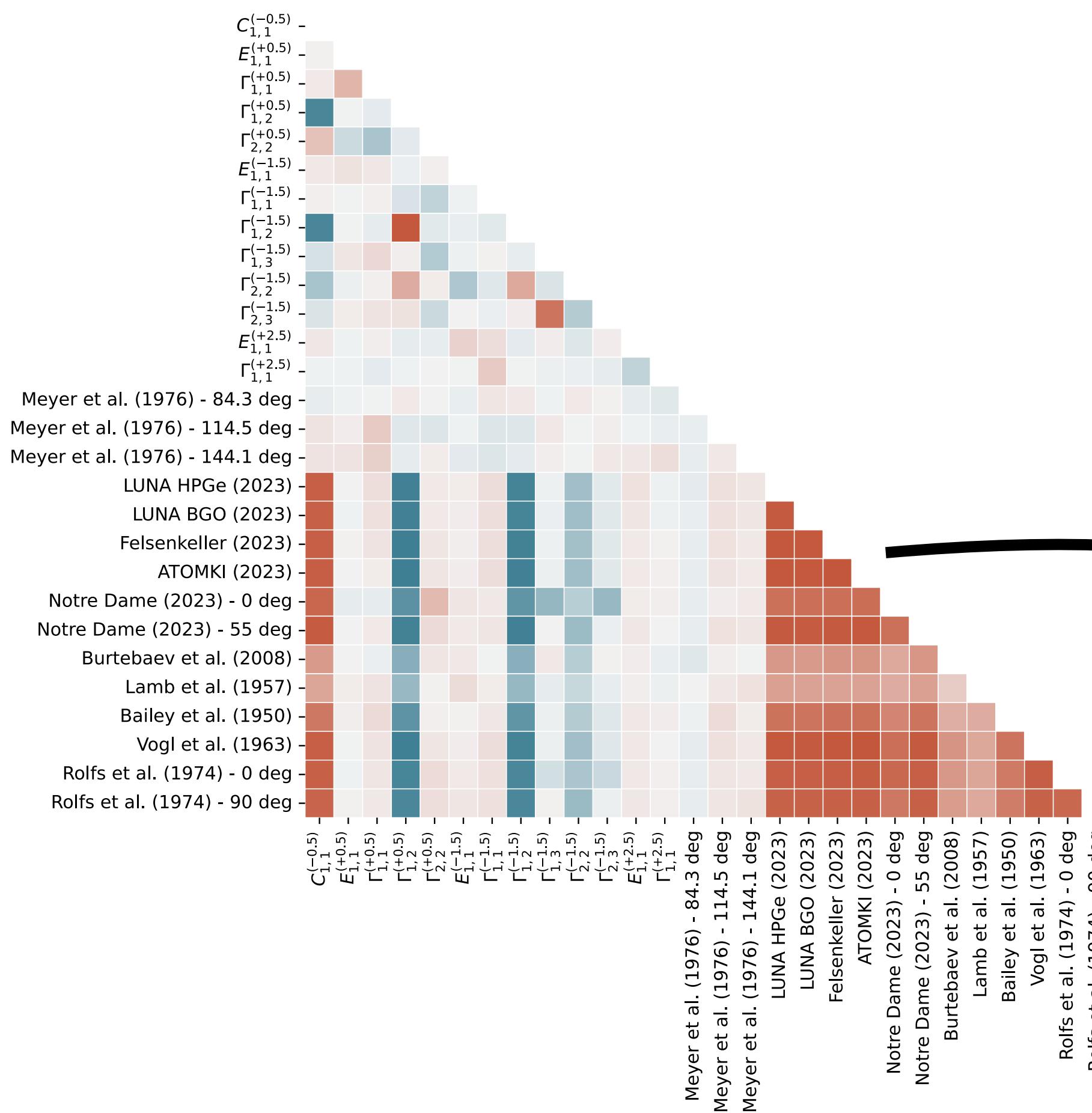
- | | | | |
|---|--------------------------|---|-------------------------|
| ∅ | LUNA HPGe (2023) | ∅ | Burtebaev et al. (2008) |
| ∅ | LUNA BGO (2023) | ∅ | Lamb et al. (1957) |
| ∅ | Skowronski et al. (2023) | ∅ | Bailey et al. (1950) |
| ∅ | Gyürky (2023) | ∅ | Vogl et al. (1963) |



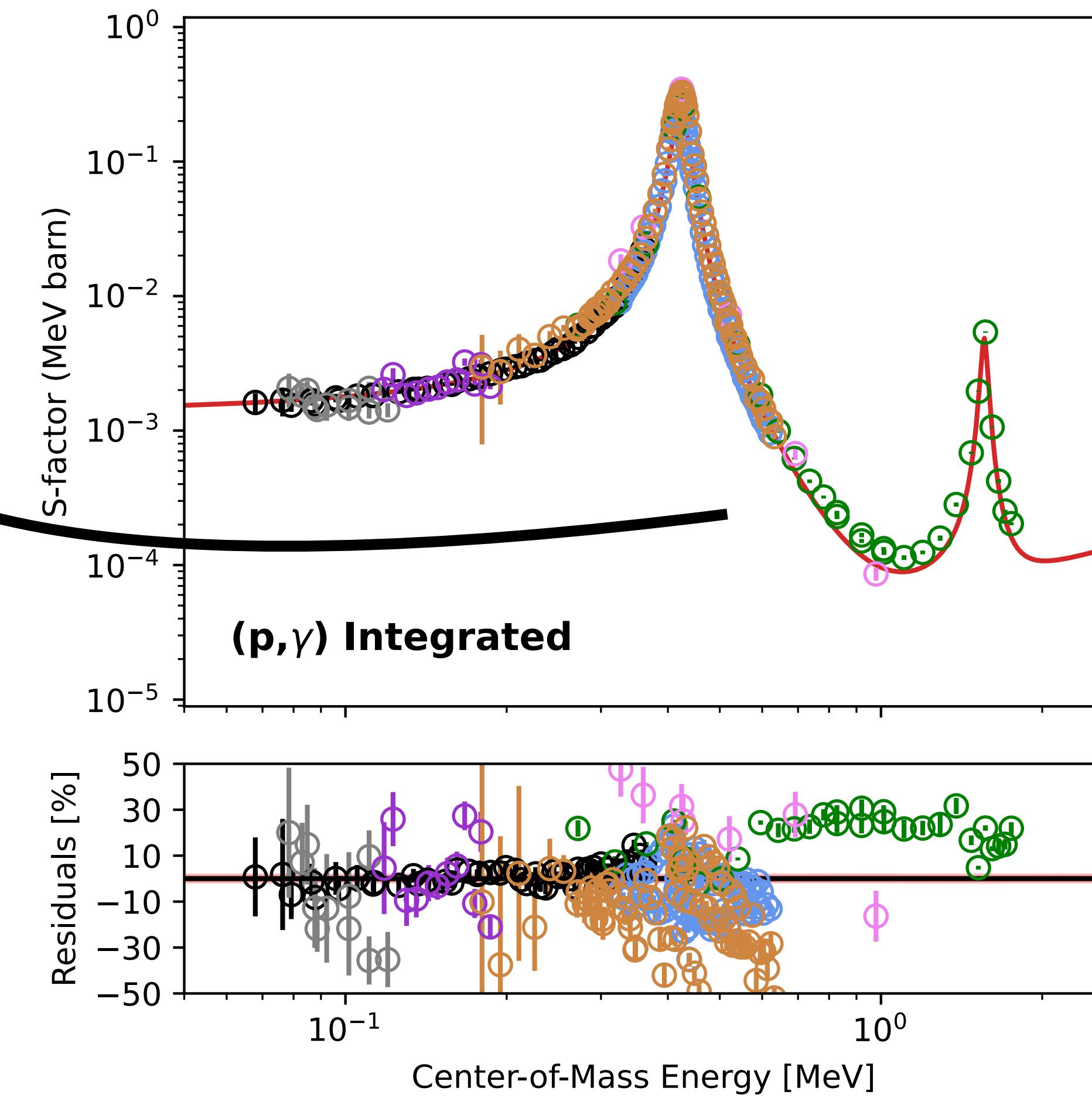
Frequentist Fit - $^{12}\text{C}(\text{p},\gamma)^{13}\text{N}$ (1)

What happened when I applied MINUIT2?

- \diamond LUNA HPGe (2023)
- \diamond LUNA BGO (2023)
- \diamond Skowronski et al. (2023)
- \diamond Gyürky (2023)
- \diamond Burtebaev et al. (2008)
- \diamond Lamb et al. (1957)
- \diamond Bailey et al. (1950)
- \diamond Vogl et al. (1963)



Small uncertainty
Huge Correlation



Frequentist Uncertainty (2)

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



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

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

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

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

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

Discrepant datasets \rightarrow High $\chi^2 \rightarrow$ 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

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

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

Discrepant datasets \rightarrow High $\chi^2 \rightarrow$ Small fit uncertainty

... and indeed the previous fit had $\chi^2_{\text{dof}} = 7$



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

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

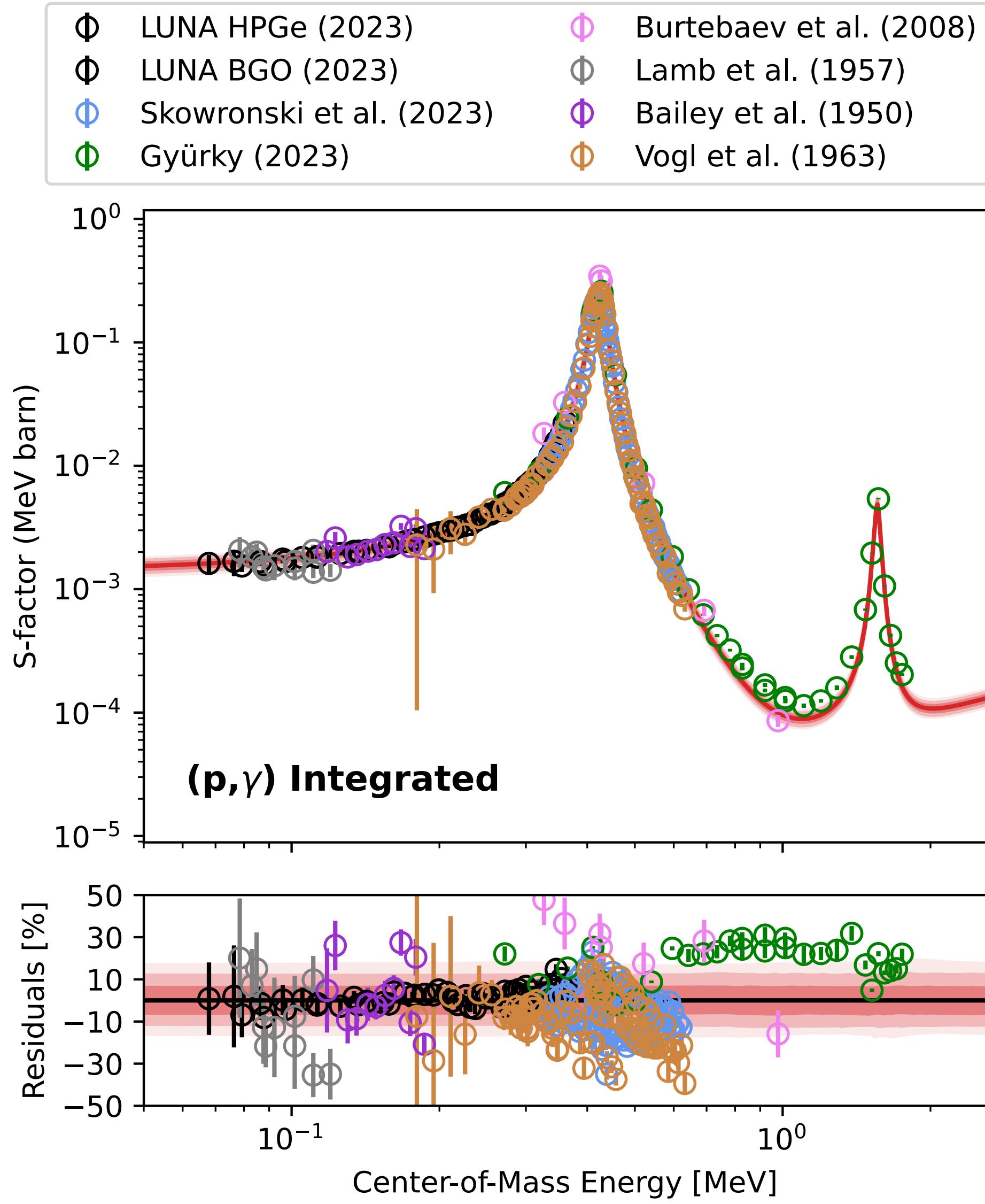
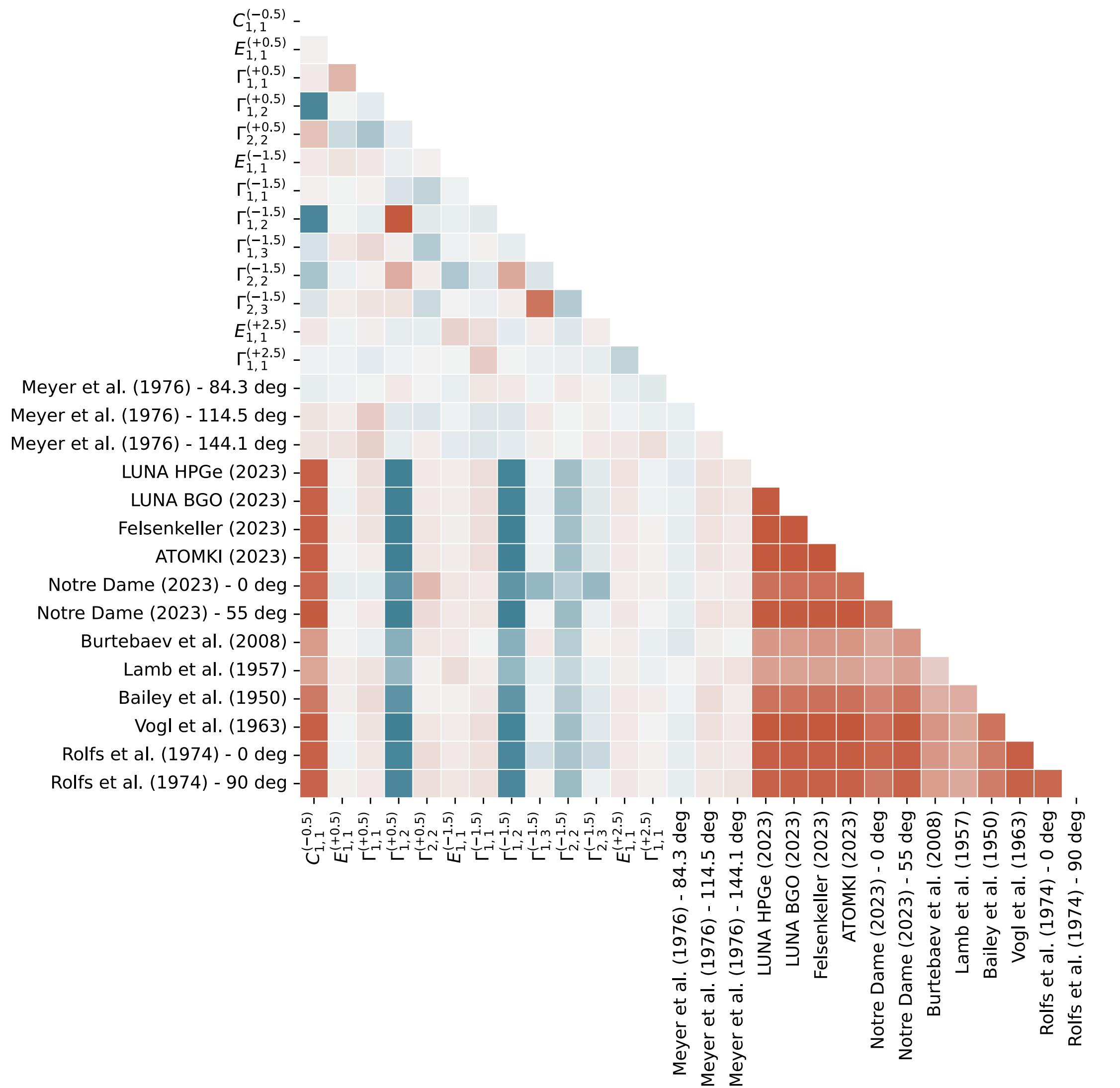
Particle Data Group, Prog. Theor. Exp. Phys. 2022, 083C01

Discrepant datasets \rightarrow High $\chi^2 \rightarrow$ Small fit uncertainty

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

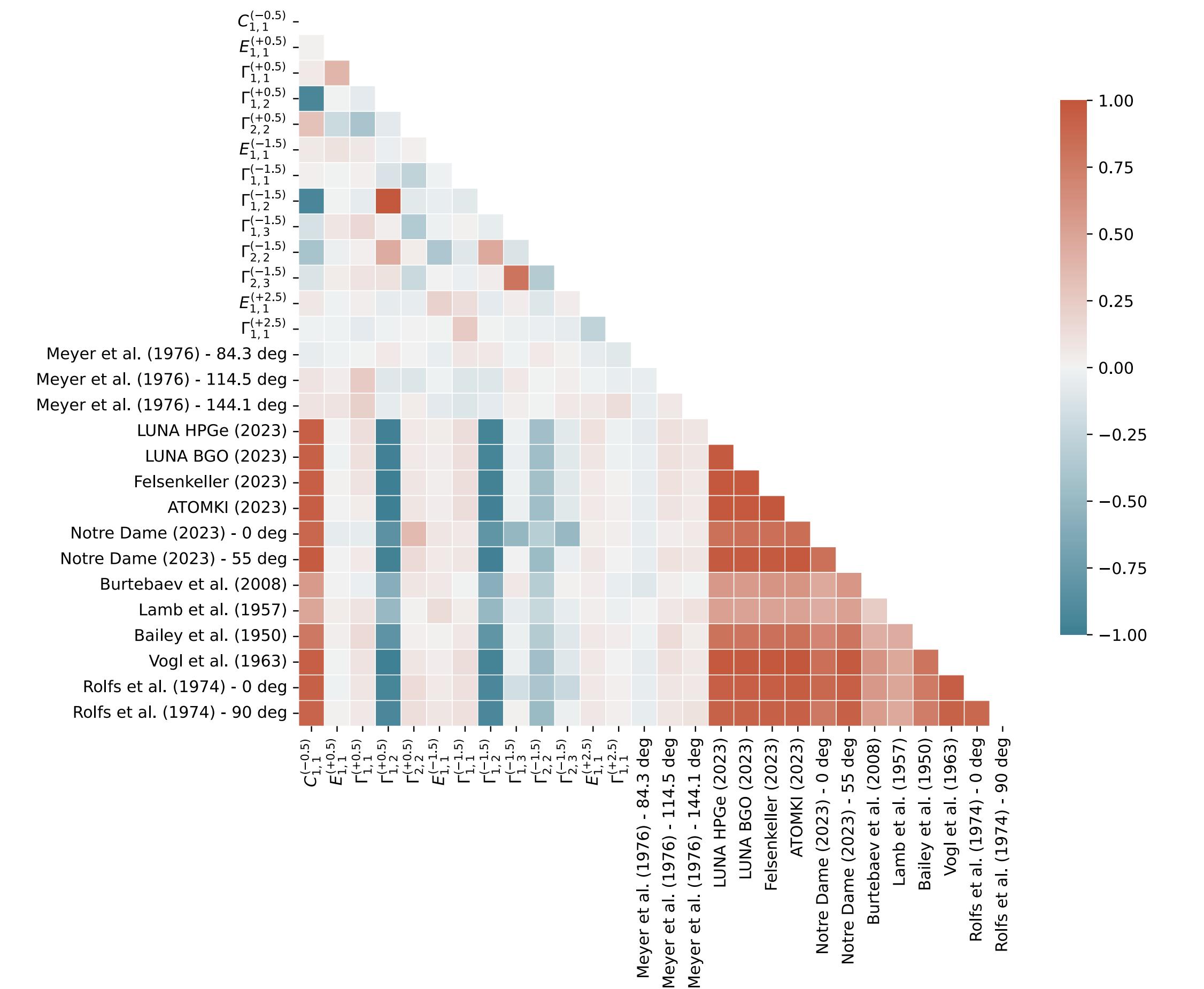


Frequentist Fit - $^{12}\text{C}(\text{p},\gamma)^{13}\text{N}$ (2)



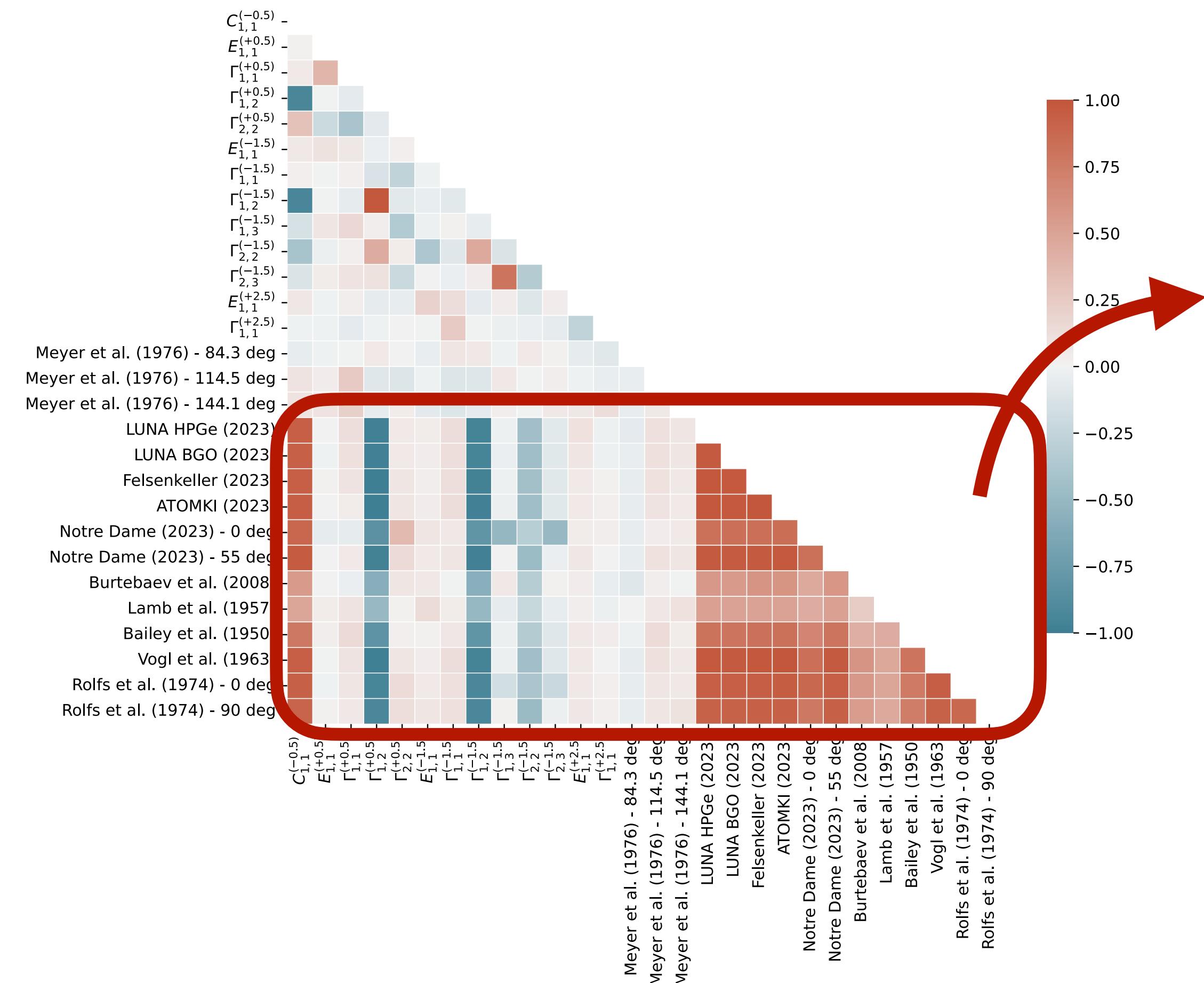
Correlation Matrix

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



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?

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

~ number of data points

~ number of dataset

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

→ Many multiple minima exists!



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

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

→ Many multiple minima exists! →

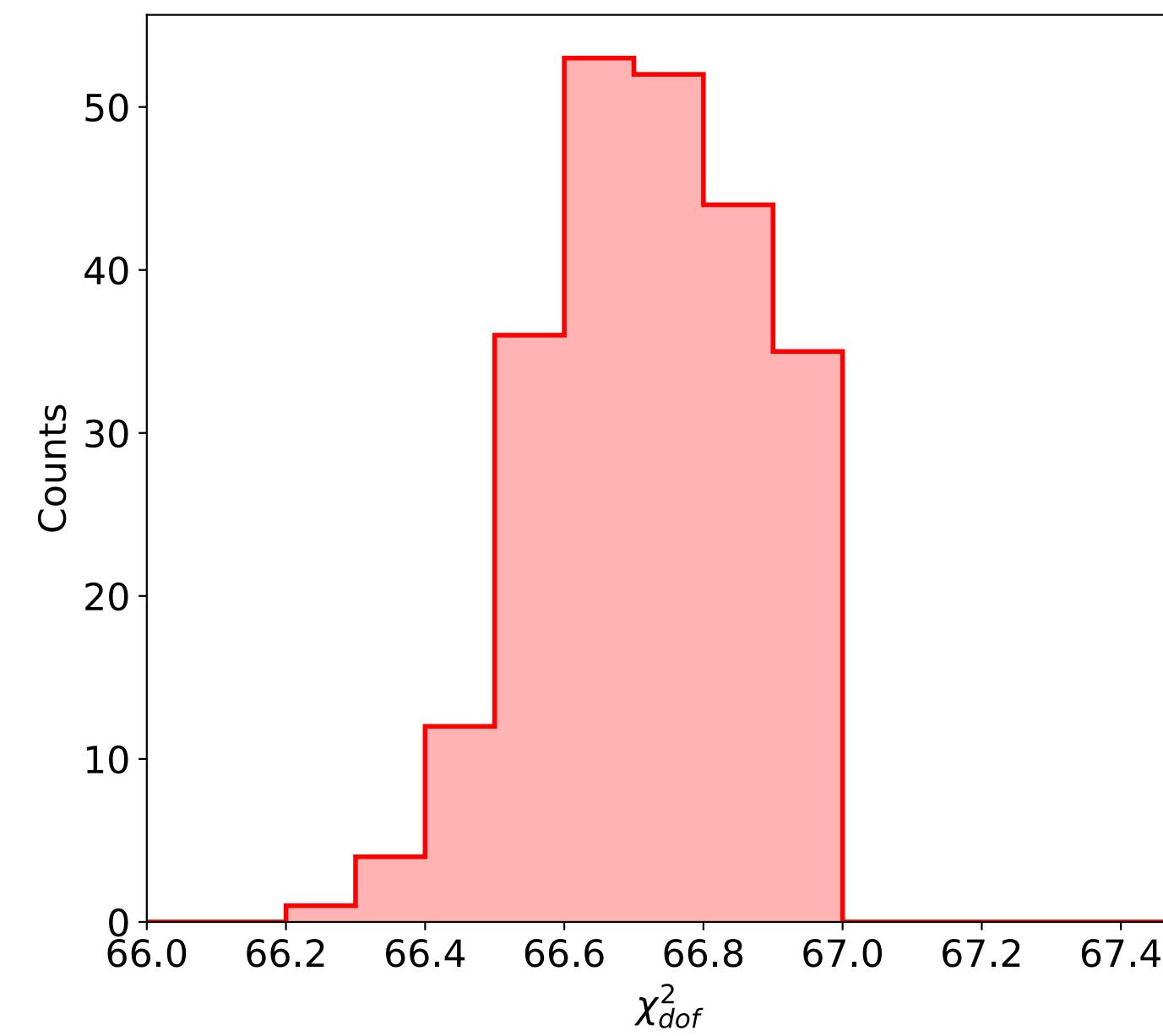
Sometimes you get a different fit
by changing the initial values...



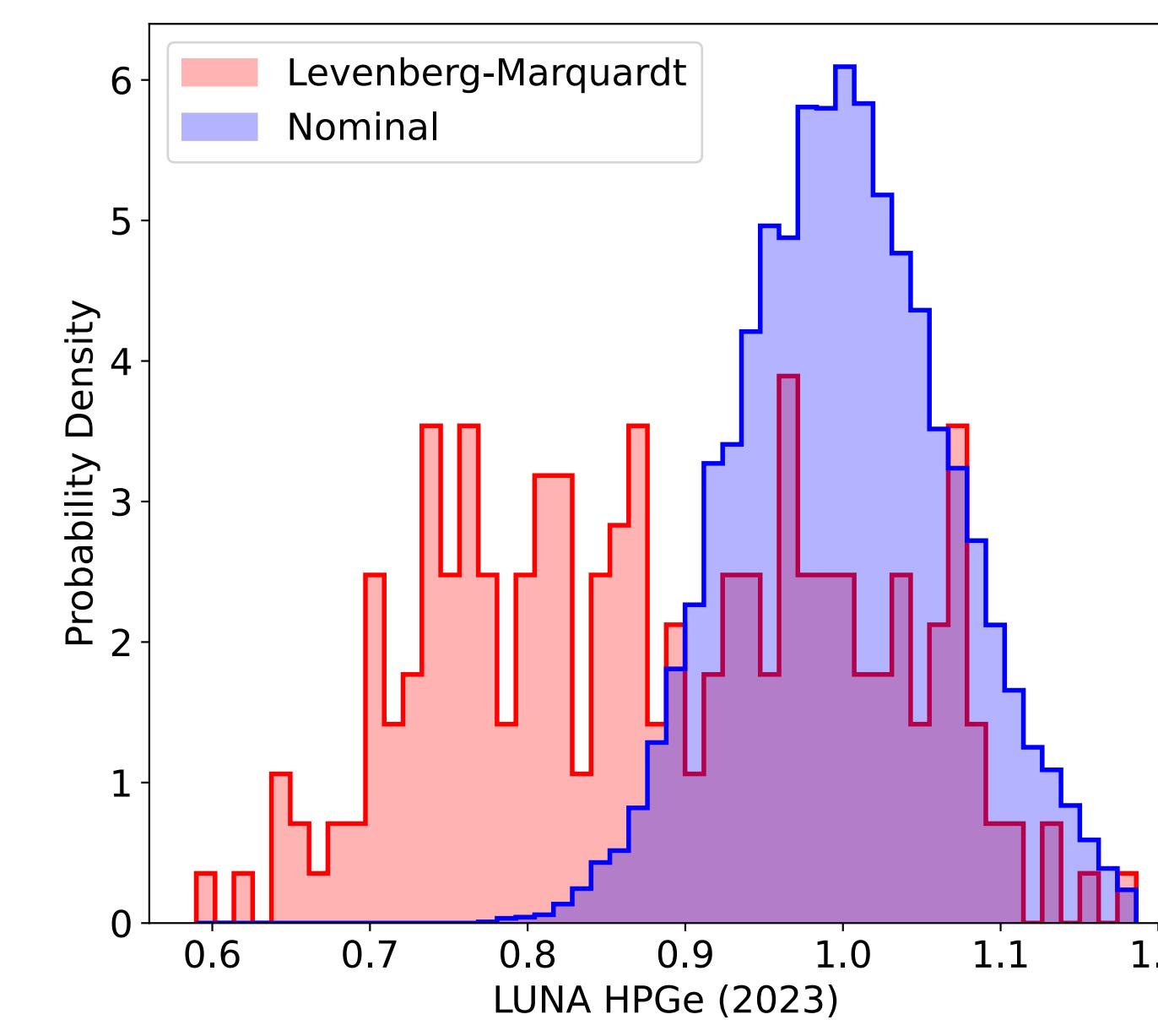
Correlation Problem

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

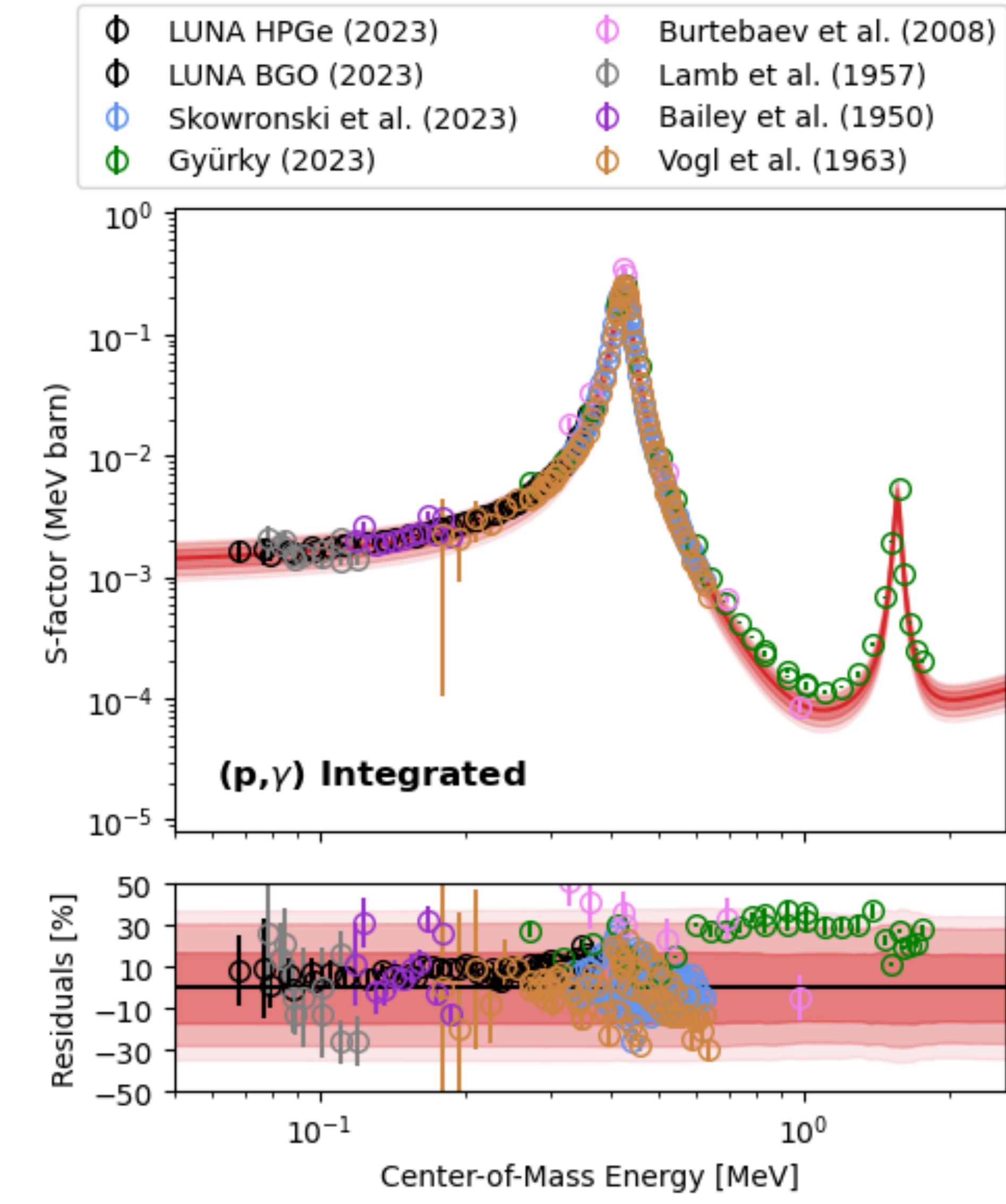
Comparable Minima



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 Imfit 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



Minimisers

MIGRAD can easily stuck in a local minima



Different initial parameters can give different final result

However, many other algorithms are available. The Imfit 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

↳ Automatically scales the covariance

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



Minimisers

MIGRAD can easily stuck in a local minima



Different initial parameters can give different final result

However, many other algorithms are available. The Imfit package include them 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

↳ Automatically scales the covariance

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



<https://github.com/skowrons94/r-matrix-school-2024>

Exercise n. 1

- 1) Try to use at least 3 different algorithms and compare the result.
- 2) Select one algorithm but change the initial parameters and compare the result.



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**



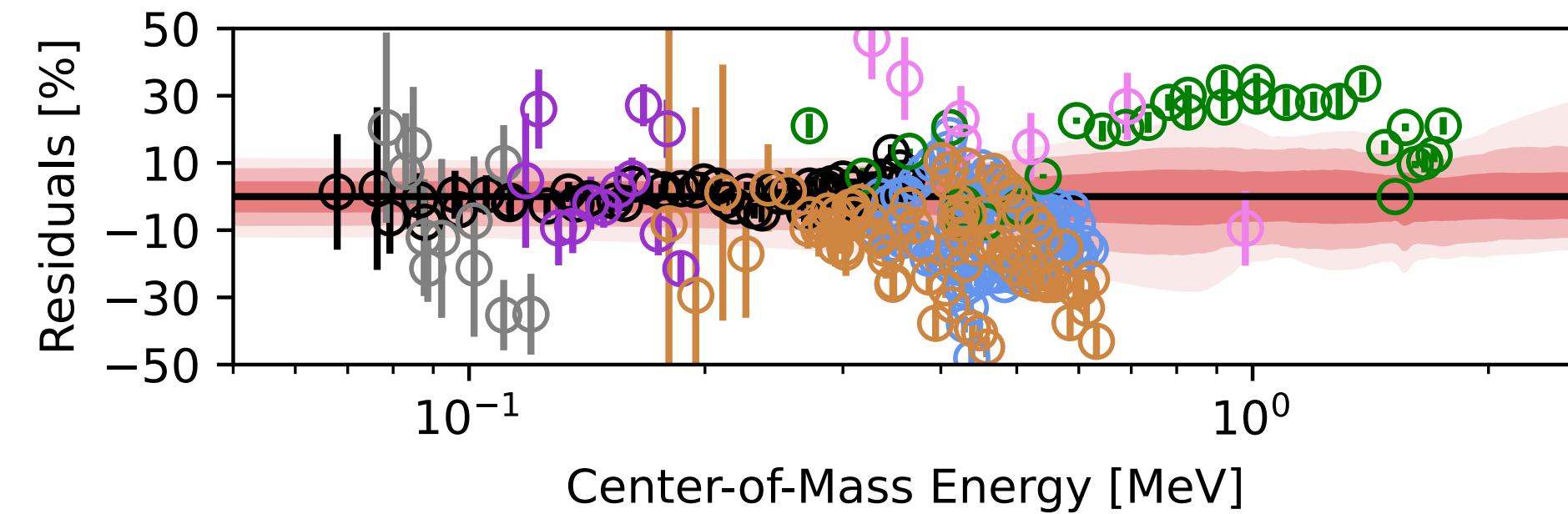
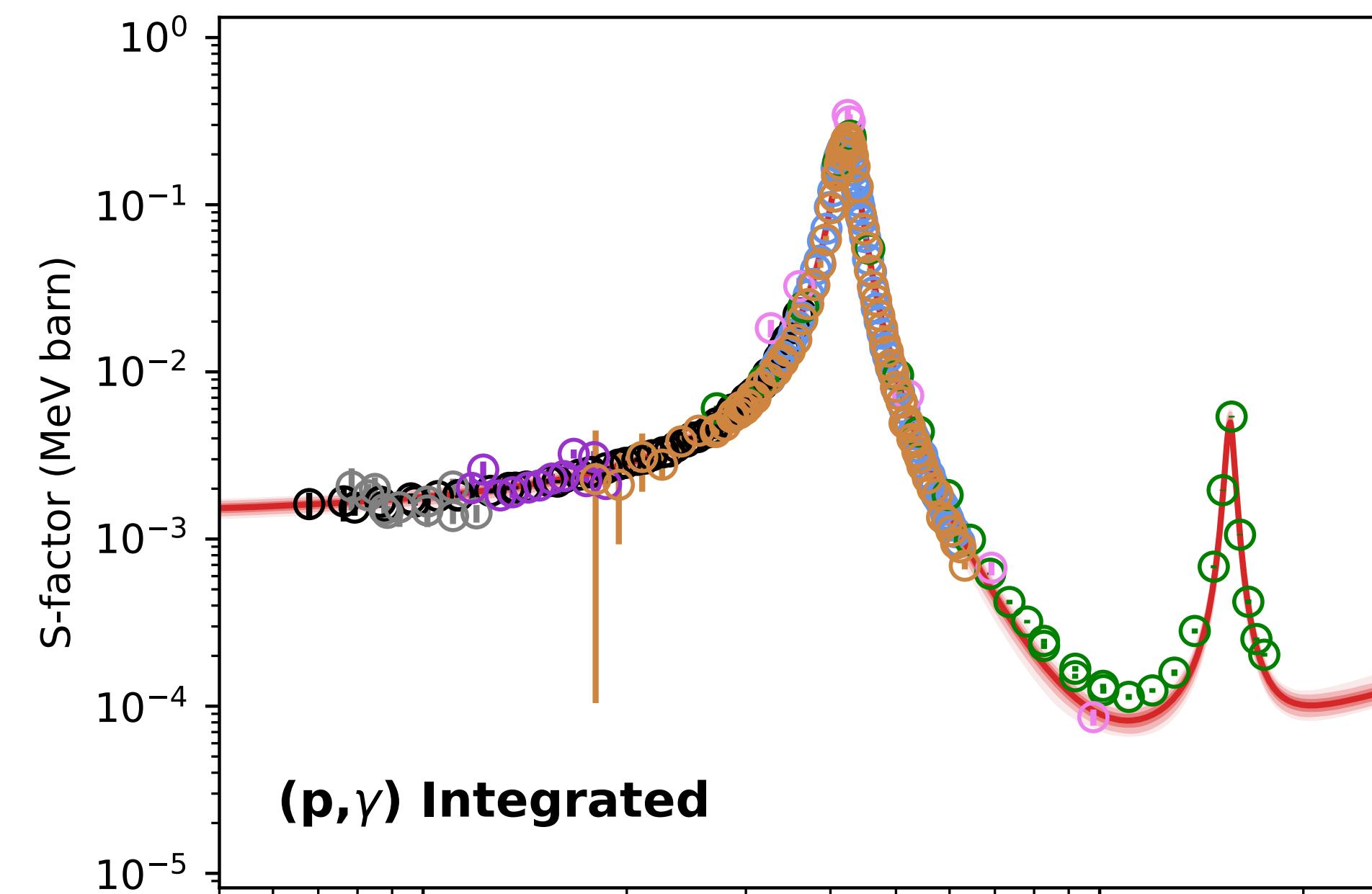
LUNN

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**

∅	LUNA HPGe (2023)	∅	Burtebaev et al. (2008)
∅	LUNA BGO (2023)	∅	Lamb et al. (1957)
∅	Felsenkeller (2023)	∅	Bailey et al. (1950)
∅	ATOMKI (2023)	∅	Vogl et al. (1963)



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**

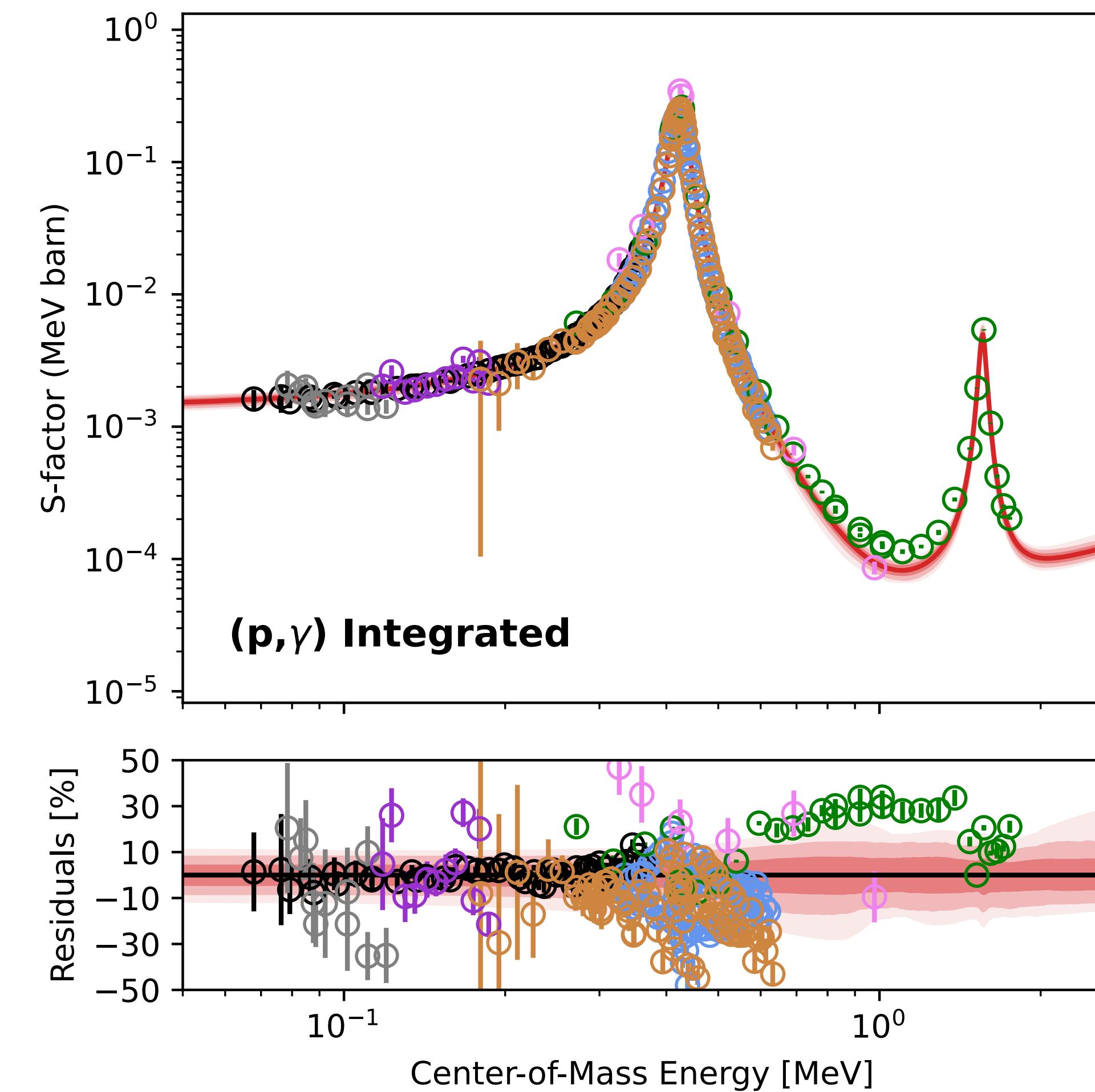
Advantages

- No covariance matrix estimation**
- Unique minimum**

Disadvantages

- Depends on systematic error evaluation**

∅	LUNA HPGe (2023)	∅	Burtebaev et al. (2008)
∅	LUNA BGO (2023)	∅	Lamb et al. (1957)
∅	Felsenkeller (2023)	∅	Bailey et al. (1950)
∅	ATOMKI (2023)	∅	Vogl et al. (1963)



Bayesian Approach



Bayes Theorem

$$P(p|x) = \frac{P(x|p) P(p)}{P(x)}$$

where x are the data
and p are the parameters



LUNN

Bayes Theorem

$$P(p|x) = \frac{P(x|p) P(p)}{P(x)}$$

where x are the data
and p are the parameters

- $P(p)$: prior distribution, ie. the prior knowledge on the parameter



Bayes Theorem

$$P(p|x) = \frac{P(x|p) P(p)}{P(x)}$$

where x are the data
and p are the parameters

- $P(p)$: prior distribution, ie. the prior knowledge on the parameter
- $P(x|p)$: likelihood function, ie. the probability of the data x given the parameters p



Bayes Theorem

$$P(p | x) = \frac{P(x | p) P(p)}{P(x)}$$

where x are the data
and p are the parameters

- $P(p)$: prior distribution, ie. the prior knowledge on the parameter
- $P(x | p)$: likelihood function, ie. the probability of the data x given the parameters p
- $P(p | x)$: posterior distribution, ie. the probability of the parameters given the data



LUNN

Bayes Theorem

$$P(p|x) = \frac{P(x|p) P(p)}{P(x)}$$

where x are the data
and p are the parameters

- $P(p)$: prior distribution, ie. the prior knowledge on the parameter
- $P(x|p)$: likelihood function, ie. the probability of the data x given the parameters p
- $P(p|x)$: posterior distribution, ie. the probability of the parameters given the data
- $P(x)$: probability of the data, ie. normalization factor



Bayesian Method

Bayes Theorem

$$P(p|x) = \frac{P(x|p) P(p)}{P(x)}$$

where x are the data
and p are the parameters

- $P(p)$: prior distribution, ie. the prior knowledge on the parameter
- $P(x|p)$: likelihood function, ie. the probability of the data x given the parameters p
- $P(p|x)$: posterior distribution, ie. the probability of the parameters given the data
- $P(x)$: probability of the data, ie. normalization factor

Problem

$$P(x) = \int P(x|p)P(p)dp \rightarrow \text{quite intractable to compute if } p \text{ dimension is high!}$$



Markov Chain Monte Carlo

Markov Chain Monte Carlo (MCMC) is a class of algorithms used to draw samples from a probability distribution. One can construct a Markov Chain which **equilibrium distribution** matches the **target distribution**.

$$P(p | x) = \frac{P(x | p) P(p)}{P(x)}$$



Markov Chain Monte Carlo

Markov Chain Monte Carlo (MCMC) is a class of algorithms used to draw samples from a probability distribution. One can construct a Markov Chain which **equilibrium distribution** matches the **target distribution**.

$$P(p | x) = \frac{P(x | p) P(p)}{P(x)}$$

Markov Chain is a stochastic model describing a sequence of possible events in which the **probability of each event** (t_i) depends **only on the state attained in the previous event** (t_{i-1}).



Metropolis Algorithm

Start: select an initial position p_0 and a proposal distribution $Q(p_i)$

→ $Q(p_i)$ is usually a multivariate normal distribution



Metropolis Algorithm

Start: select an initial position p_0 and a proposal distribution $Q(p_i)$

→ $Q(p_i)$ is usually a multivariate normal distribution

1. Sample a candidate p' from a proposal distribution $Q(p_i)$

2. Calculate the acceptance ratio: $\alpha = \frac{P(p' | x)}{P(p_i | x)}$



Metropolis Algorithm

Start: select an initial position p_0 and a proposal distribution $Q(p_i)$

→ $Q(p_i)$ is usually a multivariate normal distribution

1. Sample a candidate p' from a proposal distribution $Q(p_i)$

2. Calculate the acceptance ratio: $\alpha = \frac{P(p' | x)}{P(p_i | x)}$

$$\alpha < 1$$

$$\alpha \geq 1$$

- Sample a random number u from $[0, 1]$
- If $u < \alpha$ accept the candidate: $p_{i+1} = p'$
- If $u > \alpha$ reject the candidate: $p_{i+1} = p_i$



emcee: 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](https://doi.org/10.2140/camcos.2010.5.65)



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:

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



LUNN

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:

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

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

Informative

Uninformative



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

Informative
Uninformative

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

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



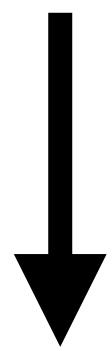
Burn-in

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**

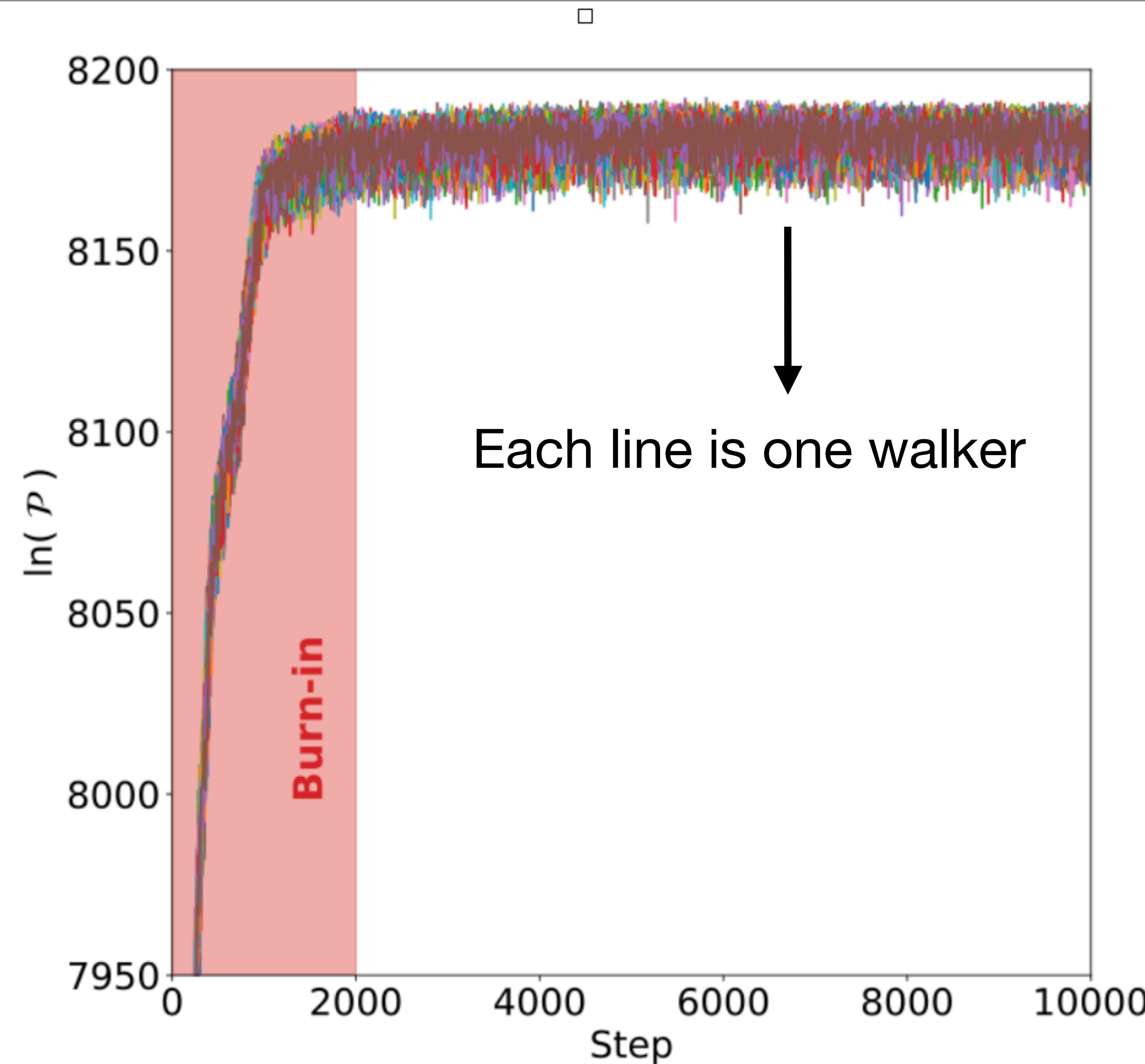


Burn-in

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



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**

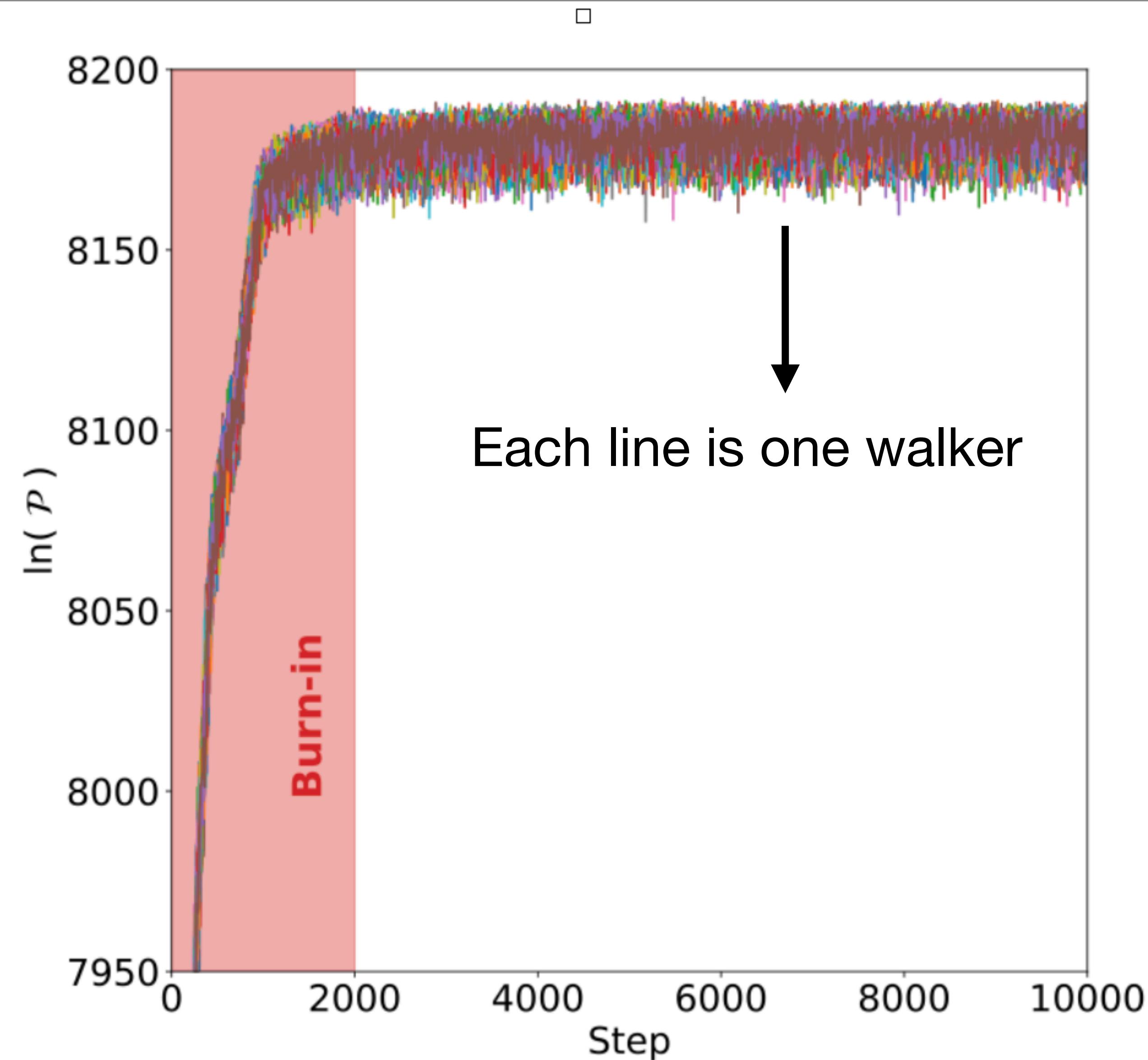


Use all the other ones to estimate the **mean value** and the **uncertainty** of your extrapolation

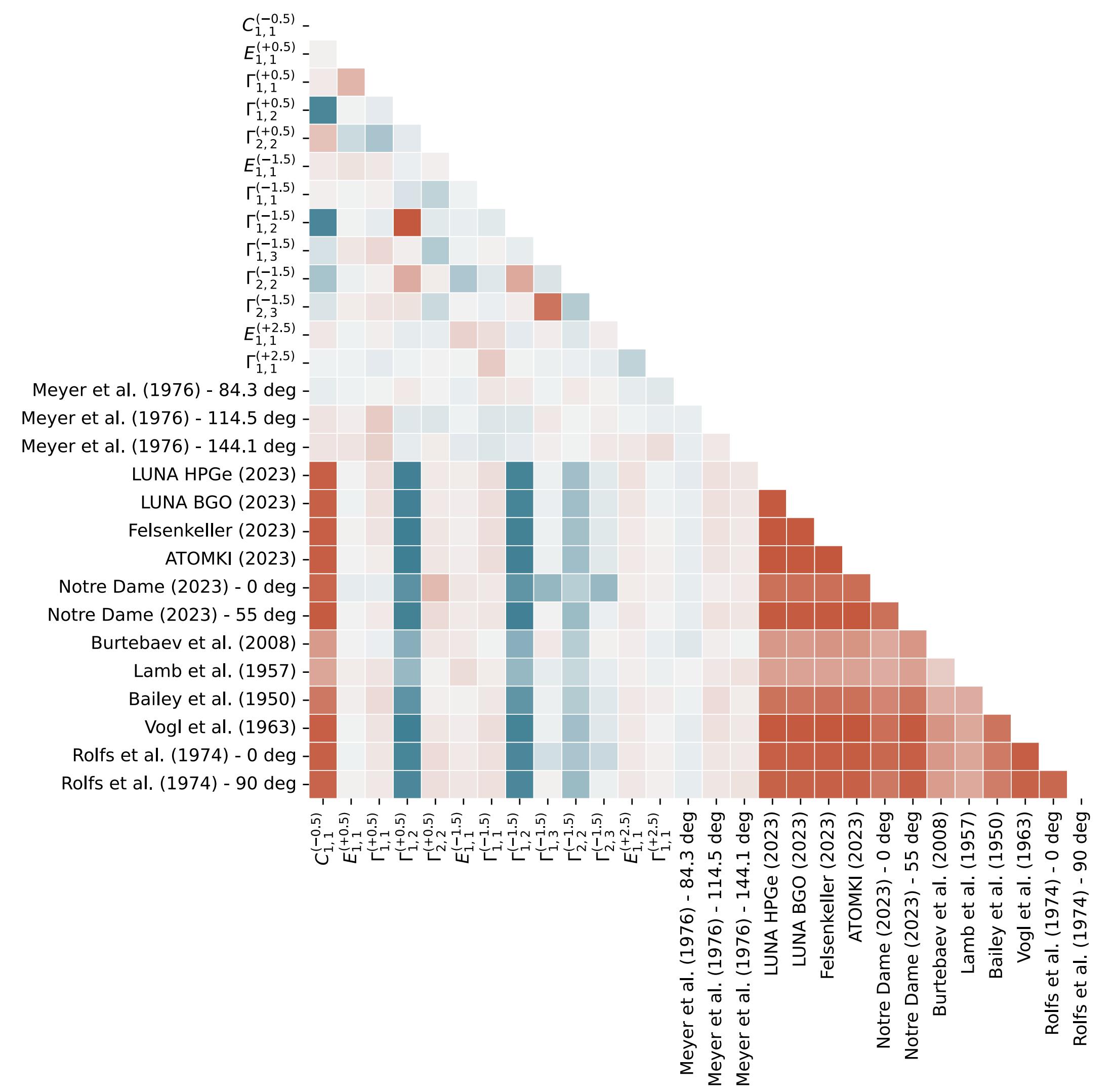
<https://github.com/skowrons94/r-matrix-school-2024>

Exercise n. 2

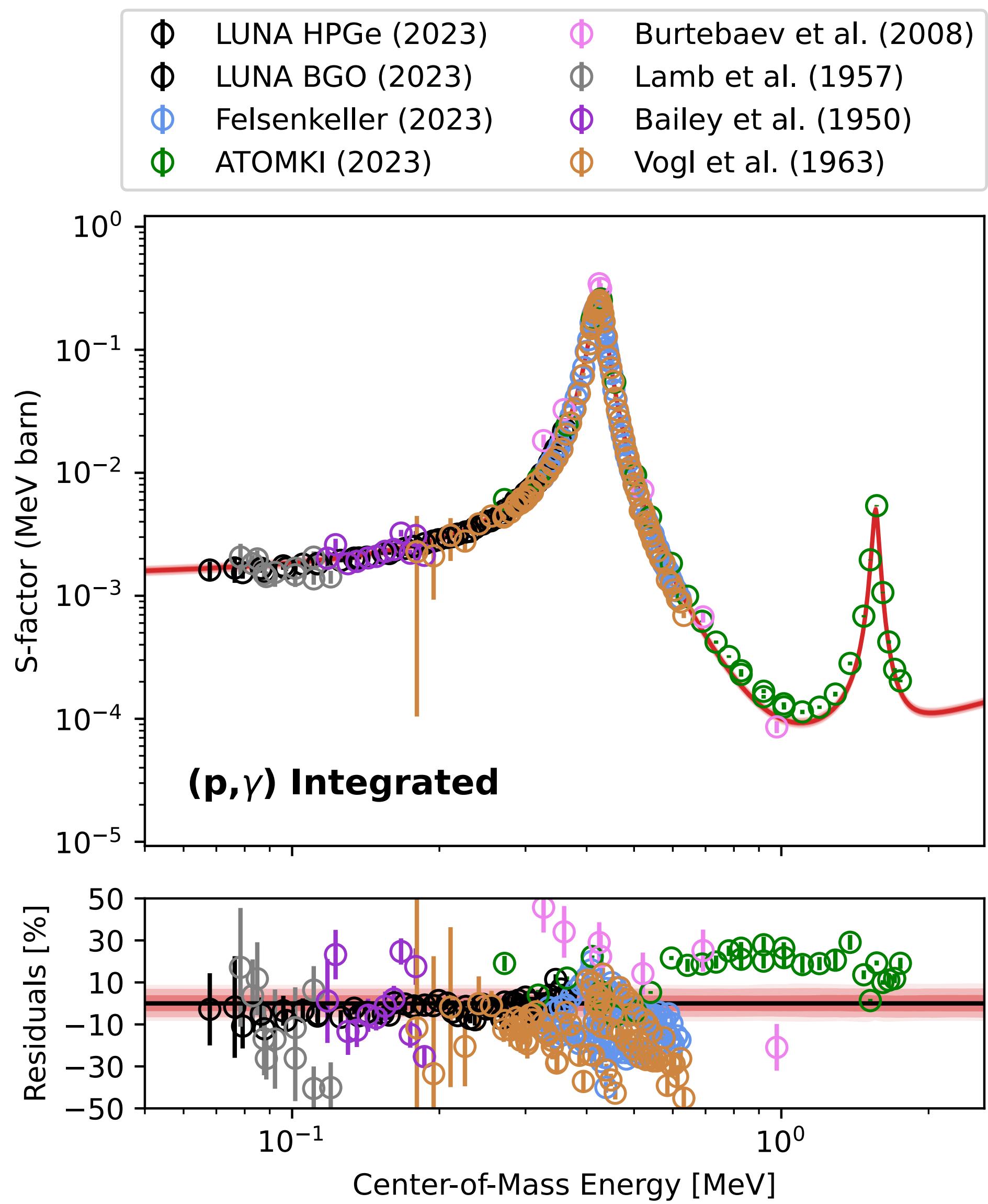
- 1) Try to run your first MCMC and compare it to previous result.
- 2) If you have time, change informative priors and see how it affects the result.



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



Very similar to frequentist case!

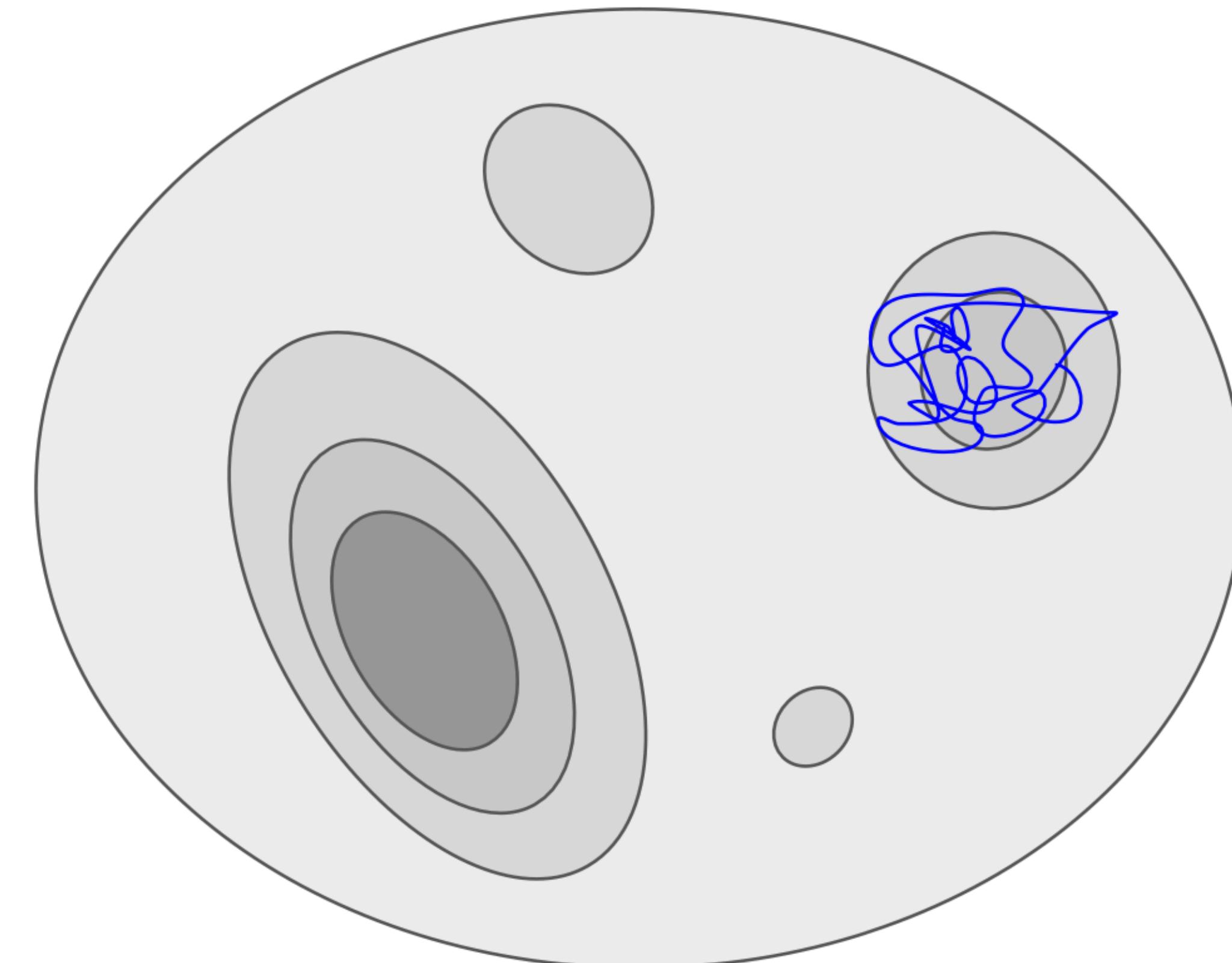


MCMC and Multimodality

**MCMC is inherently bad at
sampling multimodal
distributions**

[Interactive Example](#)

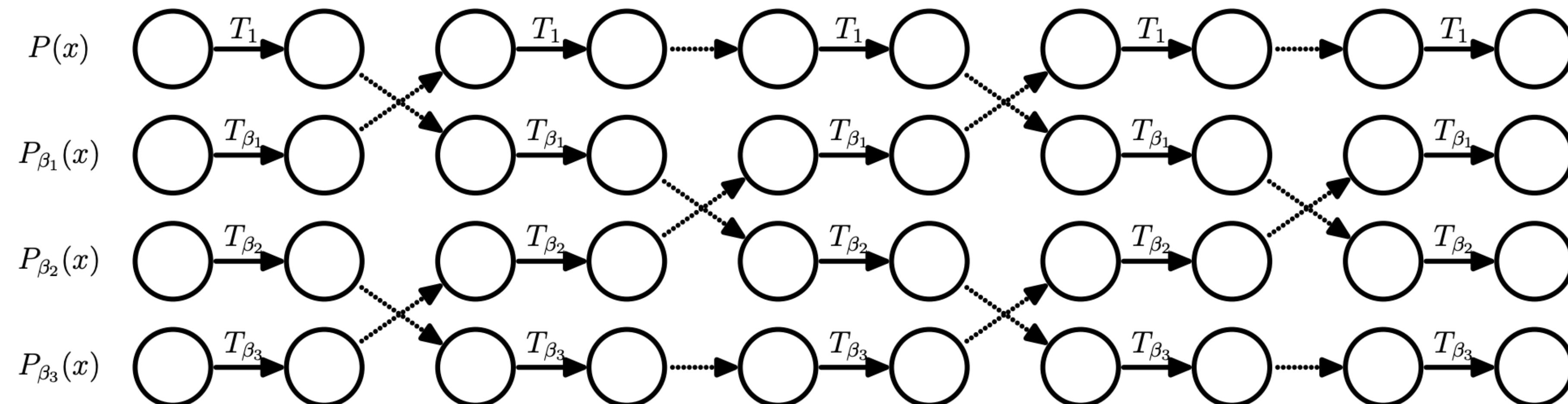
Multimodality



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

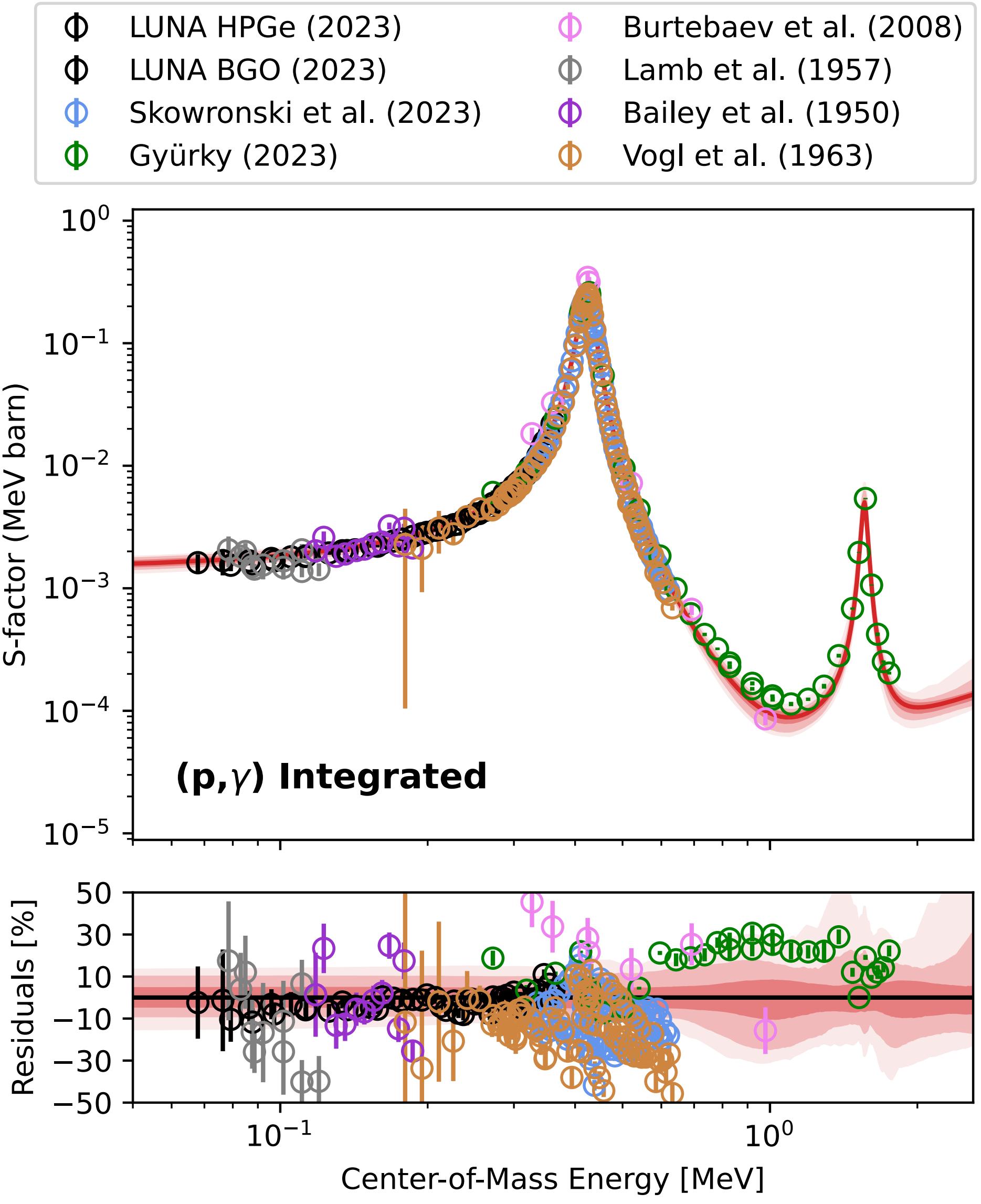
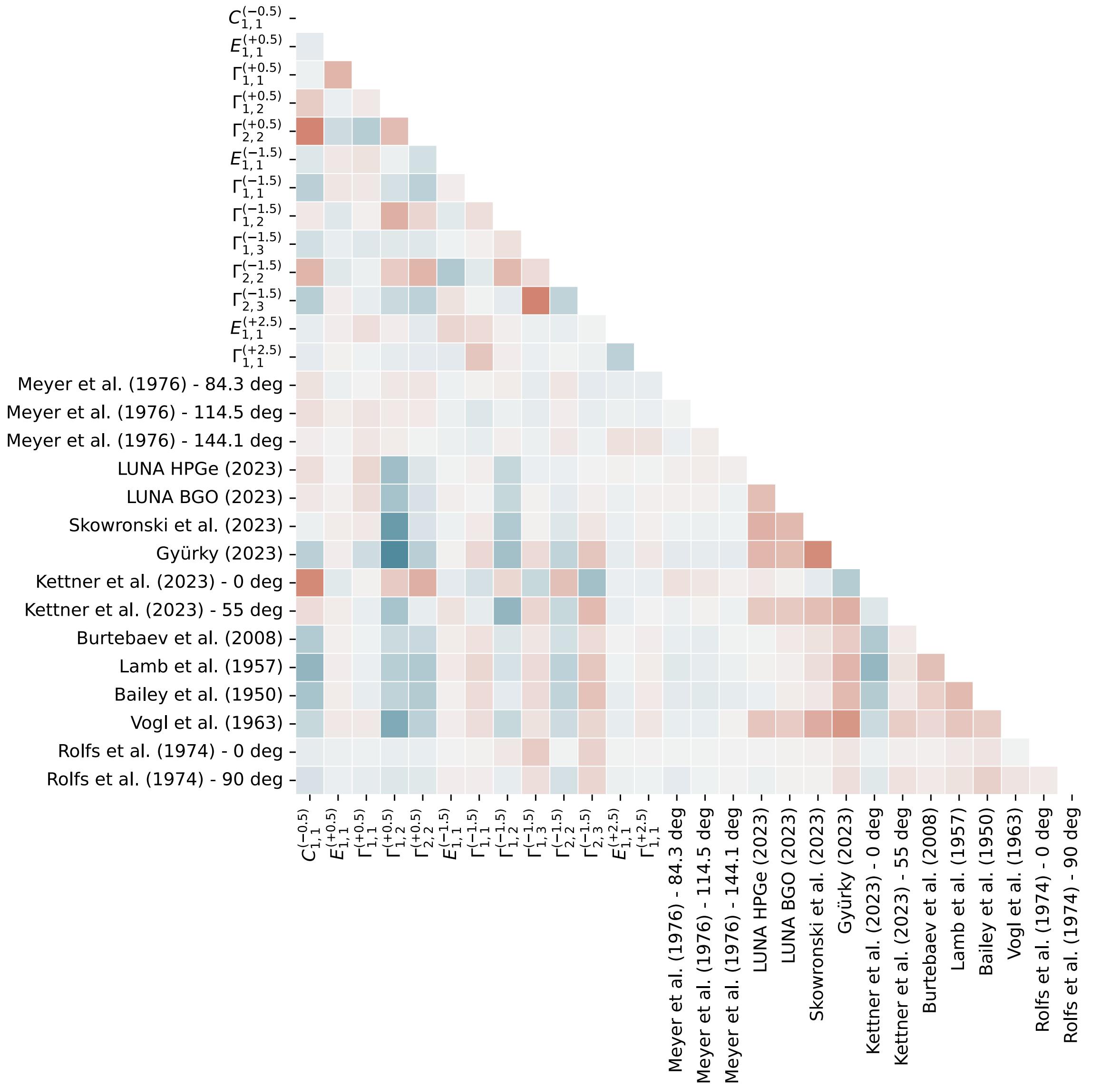


Code: <https://github.com/willvousden/ptemcee>

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



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



Conclusions

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

Thank you for attention!



Useful References

- MINUIT Manual
- MacKay, “Information Theory, Inference and Learning Algorithms”
- Bohm and Zech, “Introduction to Statistics and Data Analysis for Physicists”

