https://arxiv.org/abs/2204.13621

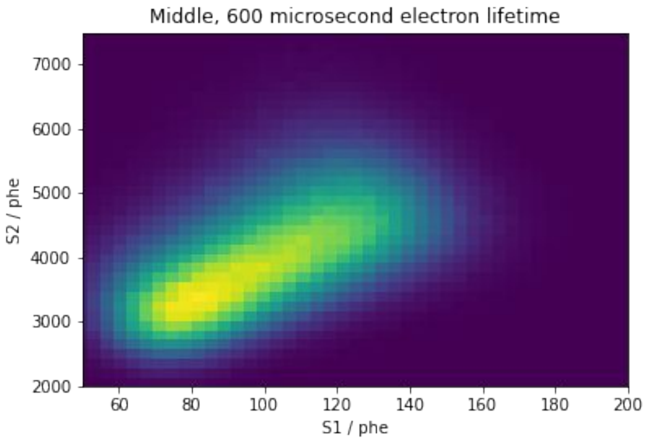
FlameNEST: Explicit Profile Likelihoods with the Noble Element Simulation Technique

DMUK 2022

Robert James

In collaboration with:
Jordan Palmer, Asher Kaboth, Chamkaur Ghag, Jelle Aalbers

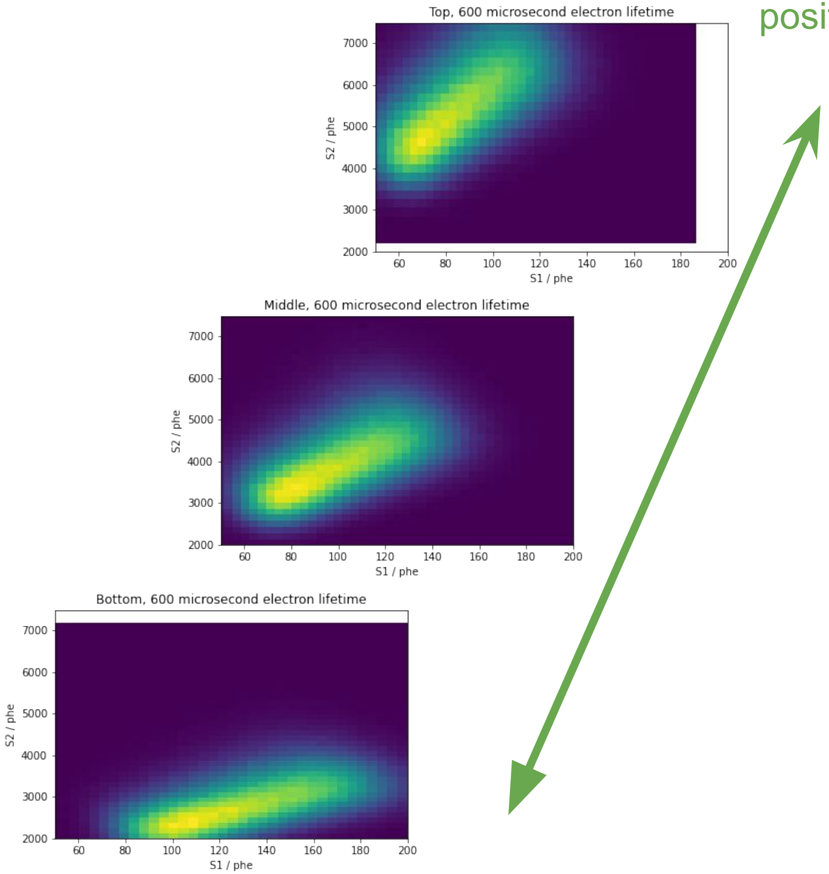
Templates: a problem



- To do statistical inference with noble element detectors, we want to evaluate the **likelihood**
- Build detector response model to signal/background sources to do this
- Traditionally, likelihood evaluation done by approximating event probabilities with Monte Carlo **templates** in observable space
- This is okay if done per source in the space of **2 observables** and with **all nuisance parameters fixed**

Templates: a problem

template for each vertical TPC position

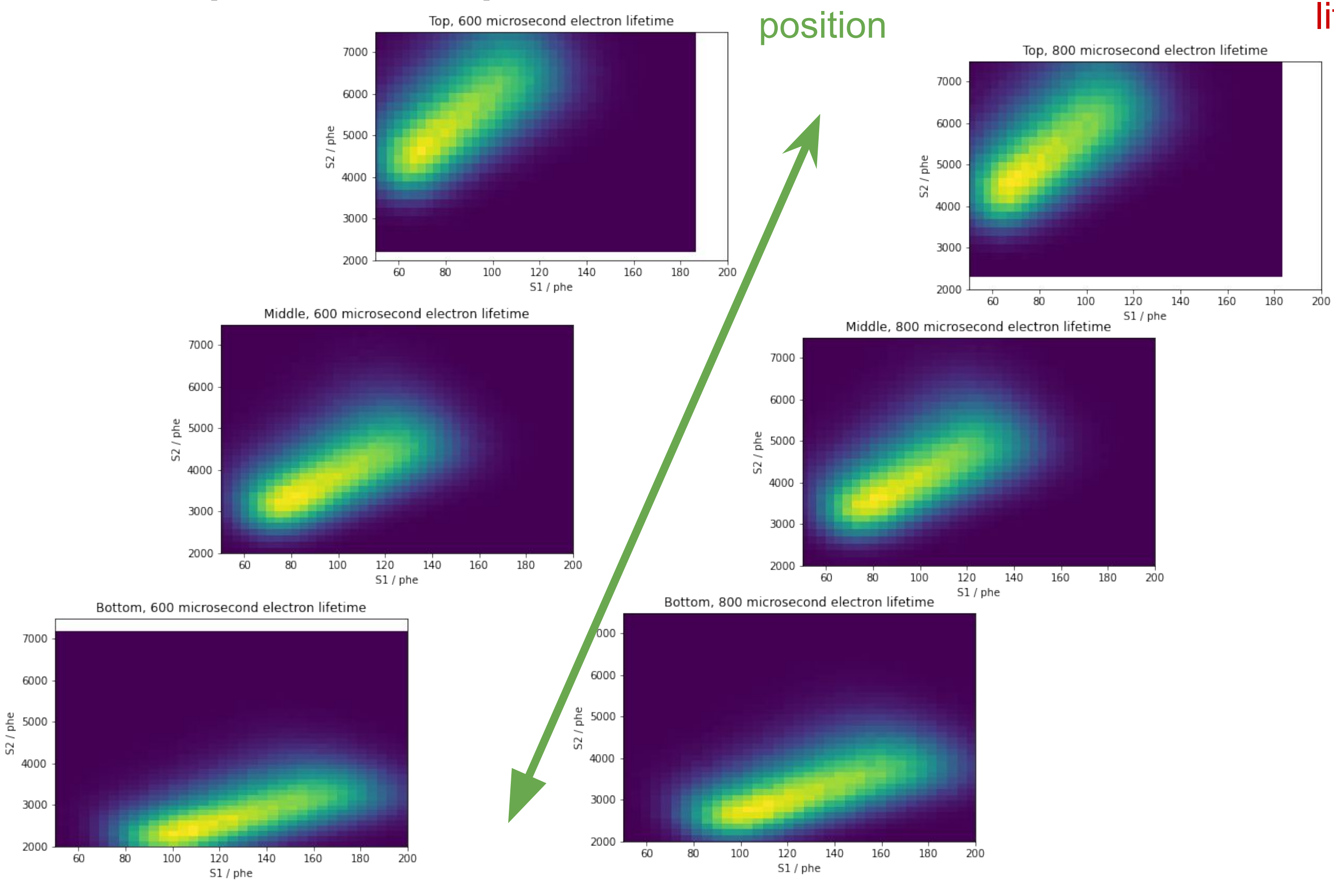


- Signal/background discrimination better at the top of the detector
- So rather than normalising signals to some fixed vertical position, better to include vertical position as an **additional observable**
- This means generating templates finely binned in this new coordinate

Templates: a problem

template for each vertical TPC position

template stack for each electron lifetime



- What if we have some uncertainty in the electron lifetime
- We should include it as a **nuisance parameter**
- Now we are generating a stack of templates for a large number of electron lifetimes

Template construction scales exponentially

z-dependence of (S1,S2)

z-dependence of electric field

...

r-dependence of energy deposition

r-dependence of electric field

...

Template space dimensionality

(S1,S2) + z + r + t + n_1 + n_2 + ...

t-dependence of energy deposition

t-dependence of electron lifetime

...

uncertainty in electron lifetime: nuisance parameter n_1

uncertainty in g1: nuisance parameter n_2

...

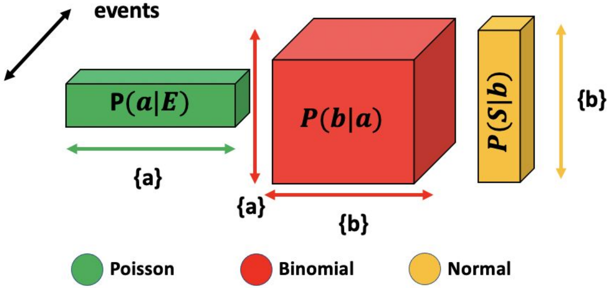
...

Evaluating likelihoods directly

$$P(a|E) \sim Poisson(\mu(E, n_1, ...))$$

$$P(b|a) \sim Binomial(n(a, n_2, ...), p(a, n_3, ...))$$

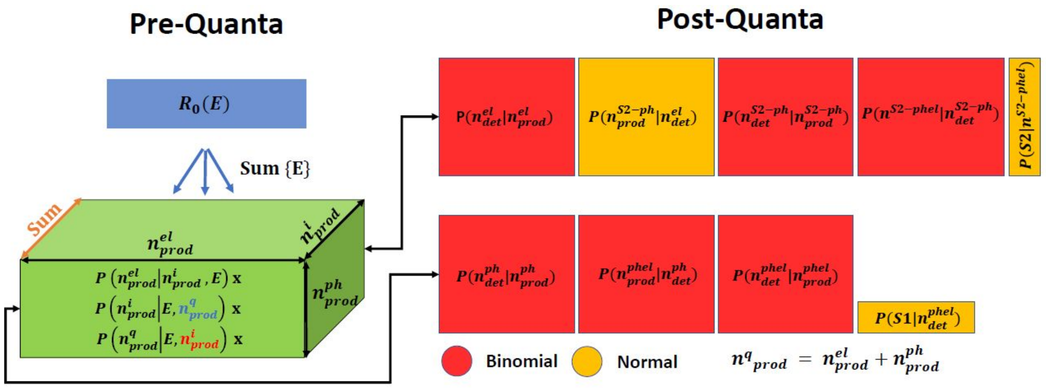
$$P(S|b) \sim Normal(\mu(b, n_4, ...), \sigma(b, n_5, ...))$$



$$P(S|E) = \sum_{a,b} P(a|E)P(b|a)P(S|b)$$

- Consider a **simple model** where some **energy deposition E** leads to some **detected signal S** via these processes - **hidden variables a,b**, **nuisance parameters n_1,n_2,...**
- To evaluate P(S|E) via **template filling**, we would have to do **MC simulation** via these distributions, **repeated over all n_i**
- More direct way: perform the **convolution of probability elements** directly. Can represent this as a **matrix multiplication**
- This means you do a **single calculation** to evaluate the likelihood for some observed S, and given set of n_i

FlameNEST



- NEST is the state-of-the-art for Monte Carlo noble element yield physics, contains very good models for detector response
- FLAMEDISX is a proof-of-concept framework for evaluating liquid xenon TPC likelihoods in this way, using simplified models
- Uses TensorFlow: benefit from GPU acceleration, automatic differentiation
- FlameNEST is an encapsulation of the full NEST computation in the FLAMEDISX framework, allowing it be be used for a variety of detector conditions and for noble element physics beyond liquid xenon

photon yield - > S1 detector response

energy -> electron/photon yields

electron yield - > S2 detector response

$$\sum_{E,e,\gamma,i,j,k,l,m,n,\ldots} \underbrace{P(S1|i)P(i|j)P(j|\ldots)\ldots P(k|\gamma)}_{\text{photon yield -> S1 detector response}}\underbrace{P(e,\gamma|E)R^j(E)}_{\text{energy -> electron/photon yields}}\underbrace{P(l|e)\ldots P(m|\ldots)P(n|m)P(S2|n)}_{\text{electron yield -> S2 detector response}},$$

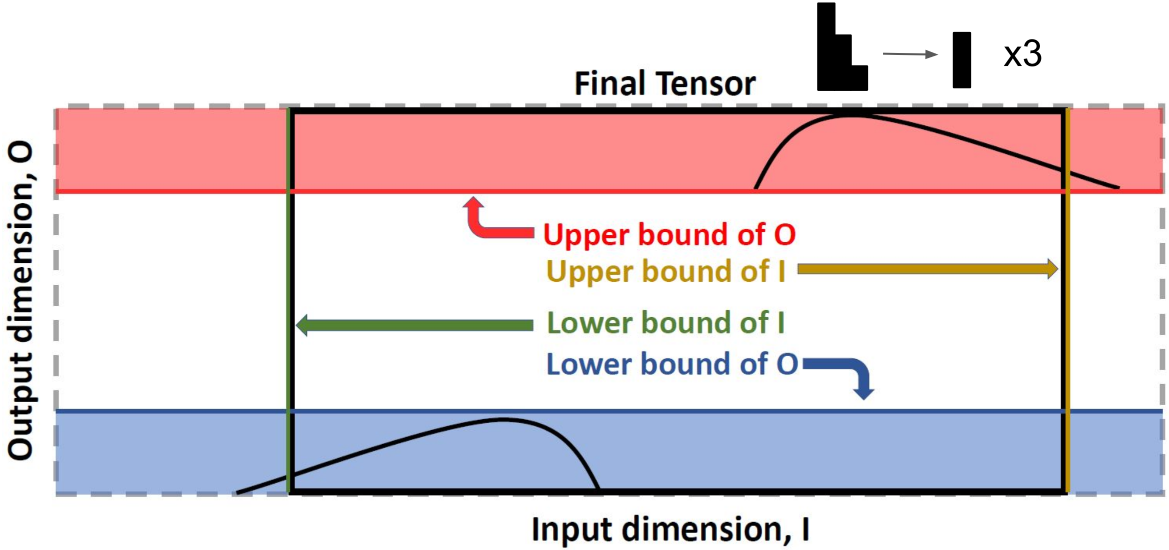
Performance features

Bounds computation

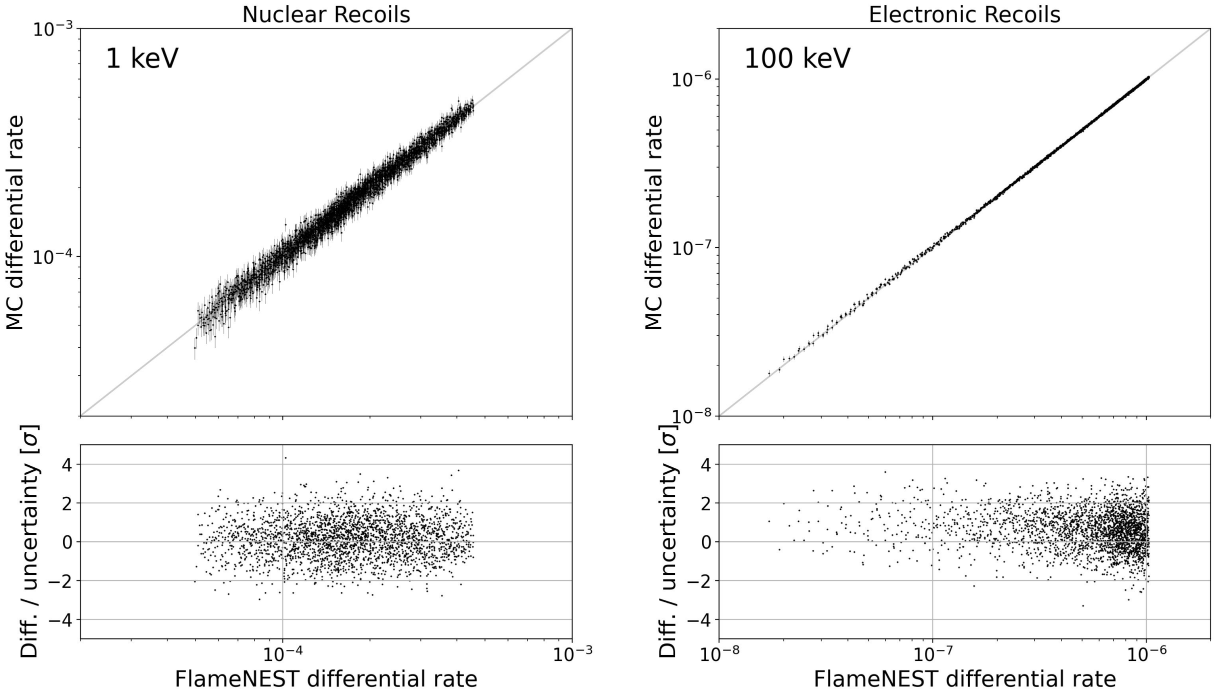
Obtain tensor bounds for a block's "in" dimension by constructing posterior PDF using bounds for "out" dimension, evaluated over a range of "in" values. Obtain sensible energy bounds for summing over the spectrum, per event.

Variable stepping

Enable extension to higher energy sources by scaling probability elements evaluated at stepped hidden variable values, enabling smaller tensor construction. Do a similar stepped/scaled sum over the energy spectrum.



Validations: mono-energetic



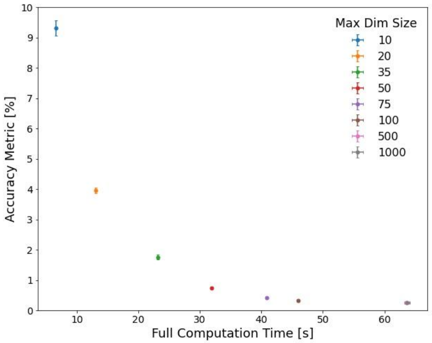
Methodology

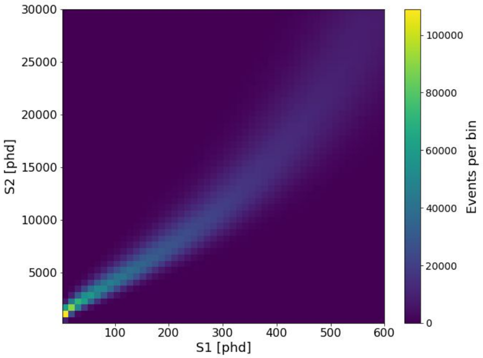
- Fill S1/S2 histograms for sources at fixed (x,y,z,t) using NEST
- Count events in each bin - 'MC differential rate'
- Compute expected events at the bin's central (S1,S2) and the fixed (x,y,z,t) via FlameNEST - 'FlameNEST differential rate'
- Check they agree within statistical + binning errors from the MC

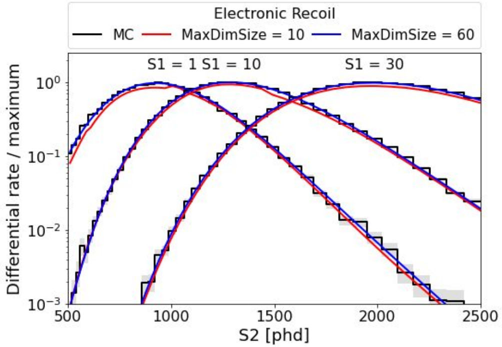
Validations: flat spectra

$$\Delta = \frac{\sum_{S1,S2}(R(S1,S2)_{MC} - R(S1,S2)_{FN})}{\sum_{S1,S2}\frac{1}{2}(R(S1,S2)_{MC} + R(S1,S2)_{FN})} \times 100\%$$

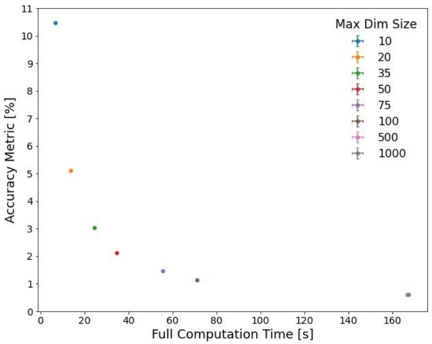
ER

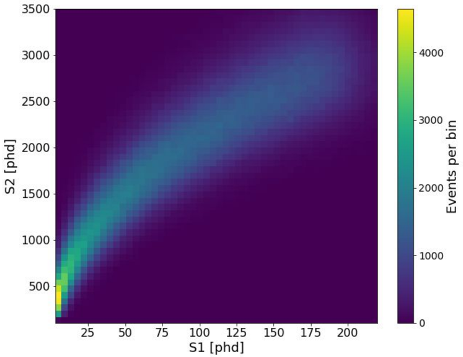


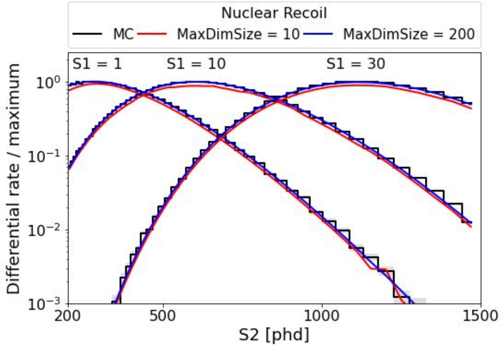




NR







For general energy spectra, the biggest tradeoff between accuracy and performance comes from the stepping done over the source energy spectrum between bounds calculated per event, controlled by ‘max_dim_size.’

Compute the above accuracy metric for different choices of this to select sensible default choices.

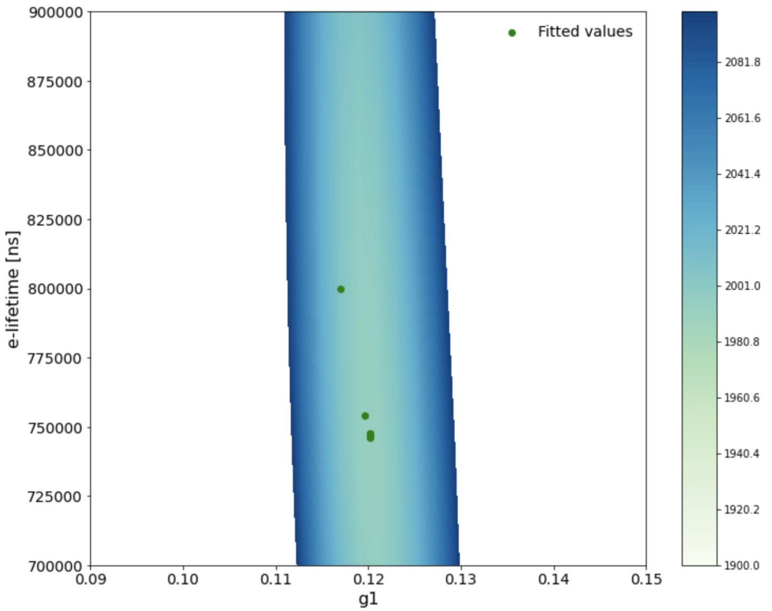
Measuring the time to evaluate differential events of all non-empty bins.

A usage example: 2 parameter fit

```
Likelihood = fd.LogLikelihood(sources = dict(er=fd.lux.LUXERSource),
                              batch_size = 20,
                              free_rates = ('er'),
                              arguments = dict(er={'energy_min':50, 'energy_max':50, 'num_energies':1}),
                              elife = (700000, 900000, 2),
                              g1 = (0.09, 0.15, 2))

ERData = ERSource.simulate(n_events)
Likelihood.set_data(ERData)
```

```
fix={'er_rate_multiplier': n_events}
Likelihood_guess = Likelihood.guess() # provide a guess (the true value used for simualtion)
```




```
bestfit = Likelihood.bestfit(guess=Likelihood_guess, optimizer='scipy', use_hessian=True, get_history=True, fix=fix) # fit
bestfit
```

	elife	g1	er_rate_multiplier	y	elife_grad	g1_grad	elife_scaled_grad	g1_scaled_grad
0	800000.000000	0.117000	100	2004.603455	-2.740938e-06	-7686.879883	-0.548188	-461.212793
1	754013.980335	0.119652	100	1992.867615	-2.989179e-06	-1349.363953	-0.597836	-80.961837
2	747643.986797	0.120168	100	1992.499634	1.144532e-06	-106.338623	0.228906	-6.380317
3	747644.512604	0.120206	100	1992.497162	2.368304e-06	-0.153137	0.473661	-0.009188
4	746360.158492	0.120221	100	1992.496155	3.385553e-07	-0.102051	0.067711	-0.006123
5	746185.073679	0.120223	100	1992.495575	6.042956e-08	0.001770	0.012086	0.000106

Outlook

- Currently addressing reviewer comments on our paper, will update the arxiv version accordingly

- Big problem #1: current method for obtaining the poisson μ in an analytic/differentiable form has major flaws, working on an improvement (see backup slide 13)

- Big problem #2: because of the way the computation scales, non-asymptotic p-value evaluation will be tricky. Ideas: higher-order asymptotics, shortcuts to verifying Wilks' theorem holds Bayesian methods

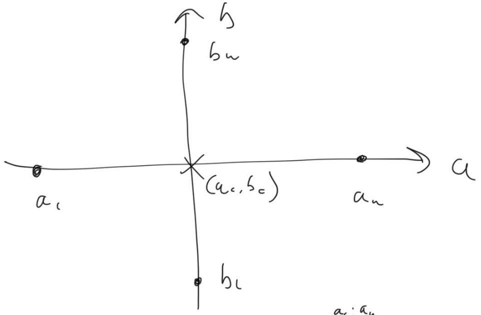
- Code publicly available on the FLAMEDISX GitHub repository: link

Backup: poisson μ interpolation

$$L = \frac{e^{-\mu(\vec{\theta})}\mu^N}{N!}\prod_{e=1}^{N}\sum_s \frac{R^s(d_e|\vec{\theta})}{\mu(\vec{\theta})}$$

$$\Rightarrow \ln L = -\mu(\vec{\theta}) + \sum_{e=1}^{N}\sum_s \ln R^s(d_e|\vec{\theta})$$

Current method: ‘cross interpolation’ between estimated μs. Fails badly with correlated parameters



$$\mu(a,b) \simeq \mu_c \times \frac{\text{interp}^{a_l:a_u}_{a,\text{lin}}[\mu(a,b_c)]}{\mu_c} \times \frac{\text{interp}^{b_l:b_u}_{b,\text{lin}}[\mu(a_c,b)]}{\mu_c}$$

$$\mu_{\text{estimate}} = \overline{N_{\text{events}}} \times \frac{N_{\text{survive}}}{N_{\text{trials}}}$$

New idea: calculate efficiency analytically/differentiably from one set of nuisance parameters, getting proportionality constant via μ estimation. Get 1st and 2nd derivatives, Taylor expand around here to get at other nuisance values

$$1 - \varepsilon = A \sum_{i \in \{O_s\}} R(\{o_i\})$$

$$\mu = \sum \mu^{be}$$

$$= \mu^{be}\Big(1 - A \sum_{i \in \{O_s\}} R(\{o_i\})\Big)$$