

pynu: neutrino analysis software, global fits and parameter inference in the HyperK era

Pablo F. – DIPC

HK-ES meeting, 2024/10/01, IFAE

pabloferm readme update 75c3aaf · 2 weeks ago		
docs	untracked files	10 months ago
examples/AnalysisFiles	untracked files	10 months ago
pynu	minor update to handle no gradie...	5 months ago
resources	SuperK detector, more physics tun...	10 months ago
test	untracked files	10 months ago
utils	SuperK detector, more physics tun...	10 months ago
.gitignore	SuperK detector, more physics tun...	10 months ago
README.md	readme update	2 weeks ago
analysis_main.py	minor update to handle no gradie...	5 months ago
plot_main.py	SuperK detector, more physics tun...	10 months ago
report_main.py	SuperK detector, more physics tun...	10 months ago
requirements.txt	bug fixes and updated requireme...	7 months ago
setup.py	SuperK detector, more physics tun...	10 months ago

About

No description, website, or topics provided.

Readme Activity 2 stars 3 watching 0 forks

Releases 1

0.1.0 Latest on Feb 23

Packages

No packages published [Publish your first package](#)

Languages

Python 100.0%

Since last year

- First “stable” release on February 23
- Four branches
 - main (stable)
 - devel (rather reliable new things)
 - HMC (novel inference methods, this talk)
 - pheno (random sensitivity studies, briefly in this talk)

Goal for next year:

- Tag new version, 1.0.0, including everything above and
 - New oscillator (next talk)
 - SK official atm.
 - HK official atm.
- Your ideas come here, get in touch



Introduction/reminder

- **Neutrino analysis software (in Python) for neutrino oscillations, flux and cross-sections**
- **Focus on flexibility: easy to include any neutrino source, cross-section model and detector**
Same framework, different analyses and any combination of them.
- **Made to accommodate any number of experiments accounting for their correlations implicitly**
Joint analyses can be performed out of the box or with very little modifications



Why python

- Flexible
- Makes it easier to expand – quickly implementing and testing new ideas and developments
- Much easier to install and better/easier handling of dependencies (we are trying to remove most of tricky dependencies)
- Lots of developments in many areas, having access to state-of-the-art software
- Easy to include cross-platform tools

... but it's slow(er)ish

- Overcome the slowest parts using small packages inherited from C++ (numpy, boost_histogram, nusquids, etc.)
- New implementations

Summary of Pynu philosophy

Generalized and abstract analysis software focused on performance and flexibility

Pynu is not focused to any particular (neutrino) analysis, but implements a core structure on which different analyses can be implemented

→ All items defining an analysis are defined via an xml file

- **Experiment:** detector + source
- **Analysis is made of:**
 - Detectors – including simulations and data
 - Neutrino sources
 - Cross-sections
 - Oscillations
- **Types of parameters:**
 - Fixed: does not change in the analysis but allows to reweight the simulations to test different models
 - Nuisance: systematic parameters
 - Physics: free parameters to be fitted with the provided data or simulation



PyNu Framework

The aim of this software is to perform neutrino analysis in the most general and flexible way. There are three modules:

- **PyNuFit:** It is the core of the package handling simulations, data and fitting.
- **Plot:** A plotting toolkit to extract all the information from the analysis (UNDER CONSTRUCTION).
- **Report:** Automated module for preparing a report containing the detailed information of the analysis and the results (UNDER CONSTRUCTION).

For a more complete documentation, please open the [docs/pynu.html](#) folder with your browser.



Summary of Pynu philosophy

Performance, CPU time and handling large datasets are the main concerns towards future neutrino analyses.

We aim to open this field to develop novel solutions within HK and adapt others.

In this presentation, after a bit more introduction, we will explain the main recent developments, the next steps and also the physics motivation for all of it.

(The goal of physics data analyses)

Extract the underlying physical parameters explaining a given observed data

For that, we need a theoretical model containing those parameters

Fit: compare the family of predictions made by the model against the data, and extract the values of the physical parameters that **better** reproduce the data

What is better?

The usual approach is to build a likelihood functional with data and model inputs such that it returns how likely it is for a particular prediction to explain the observed data

- + The best fit parameters are those that maximize the likelihood
- + We want more, we want the precision (error size) of the measurement

To construct this likelihood we need to measure how close data and model are, but taking into account the uncertainties of our experiment and model

- + statistics
- + systematics/nuisance

Physics Tunes

Given a model, physics tunes is the part of pynu that applies the different parameter values

Systematics, physics parameters are treated in the same way and generally called Physics Tunes

These are functions for each parameter of the flux, cross section, detector or oscillations

Therefore, each of them can be treated as systematics (nuisance), physics or fixed in an analysis; and any dependence is allowed (no need for linearization, splines or interpolation)

All (nuisance) parameters are linked to a function ($\vec{w}_\eta^\rightarrow(\eta)$) which computes the weights associated to that parameter for each event in the Monte Carlo simulation used to fit the data.

$$\vec{w}_\eta^\rightarrow: \mathbb{R} \longrightarrow \mathbb{R}^{m_\eta} \quad (11)$$

$$\eta \longmapsto \vec{w}_\eta^\rightarrow(\eta) = (E(\eta), ip(\eta), \dots) \quad (12)$$

In this approach, correlations between analysis-relevant variables are implemented implicitly and allow for any parametrization.

Binning and number of expected events is computed at runtime after all the weights are calculated. In turn, it means that there is no linearization in terms of the event rate fraction.

$$E'_i = \sum_{ev \in bin} \prod_{\eta \in nuis} w_\eta^{ev}(\eta) \prod_{\theta \in phys} w_\theta^{ev}(\theta) \quad (13)$$

Vector Functions from a single Parameter to compute non-linear Weights at Runtime

$$\vec{w}_\eta^\rightarrow: \mathbb{R}^{i_\eta} \longrightarrow \mathbb{R}^{m_\eta} \quad (14)$$

$$\vec{\eta} \longmapsto \vec{w}_\eta^\rightarrow(\vec{\eta}) = (E(\vec{\eta}), ip(\vec{\eta}), \dots) \quad (15)$$

Derivatives of Physics Tunes

Having explicitly defined physics tunes as functions, it is rather straight forward to implement their derivatives too

It's not only love (and mathness) for differentiating, their use will also become handy later.

A word on oscillations as physics tunes

Oscillation probabilities tend to have a rather complicated long forms with dependency on several parameters, so they are not easy to differentiate in a general way. Especially if some kind of interpolation or approximation is used.

For computing oscillations, the current version of pynu uses nuSQuIDS, but it has been shown to be a bottle-neck in the performance of the analysis (~11 times slower than the rest of physics tunes)

→ Jeremy to the rescue (next talk)

USC UNIVERSIDAD DE SANTIAGO DE COMPOSTELA **Alternative neutrino oscillation software**

Speed up calculation of neutrino oscillations model
nuSQuIDS + SQuIDS + gsl → Eigen

Performance comparison
11th Gen Intel(R) Core(TM) i7-11800H @ 2.30GHz
Laptop-grade CPU from 2021
8 cores, 16 threads

1M points
Evaluate in single-thread mode

<u>nuSQuIDS</u> 11503 ms	<u>Eigen</u> 676 ms
-----------------------------	------------------------

Eigen is faster by a factor of ~20
Each point/event is independent, turn on multi-threading
Eigen improves to just 100 ms

Binned negative Log-Likelihood

Negative log-likelihood

$$\pi(\text{Exp}(\vec{\theta})|\text{Obs}) = \underbrace{\frac{1}{2} \sum_{\text{Expmnt.}} \sum_{i \in \text{Bins}} \left(E'_i - O_i + O_i \cdot \log \left(\frac{O_i}{E'_i} \right) \right)}_{\text{Poisson statistics}} - \underbrace{\sum_{j \in \text{params.}} \log \mathcal{P}(\vec{\theta}_j)}_{\text{Nuisance and priors}}$$

O_i : Observed number of events in i th bin (data or simulation with assumed true values)

E_i : Expected number of events in i th bin at a give physics point and with nominal nuisance

E'_i : Expected number of events in i th bin modified by the values of nuisance parameters

θ_j : current value of j^{th} nuisance parameter

\mathcal{P} : prior for nuisance

The usual covariance matrix (everythin-is-gaussian case) is generalized to a block diagonal matrix, emphasizing:

- Parameter correlations (or their absence)
- Different parameters may be described by different distributions, i.e. we are no longer restricted to assume gaussianity of this term

$$\begin{pmatrix} \log \mathcal{P}(\vec{\theta}_0) & & & & \\ & \log \mathcal{P}(\vec{\theta}_1) & & & \\ & & \ddots & & \\ & & & \ddots & \\ & & & & \log \mathcal{P}(\vec{\theta}_m) \end{pmatrix}$$

Negative Log Likelihood

Having implemented the derivatives of physics tunes means that we can compute the **gradient of the log-likelihood** w.r.t. any subset of parameters

$$\nabla_k \pi(\theta) = \sum \left(1 - \frac{O_i}{E'_i} \right) \frac{\partial E'_i}{\partial \theta_k} - \sum_{j \in \text{params.}} \frac{\nabla_k \mathcal{P}(\vec{\theta}_j)}{\mathcal{P}(\vec{\theta}_j)}$$

and,

$$\partial_k E'_i(\vec{\theta}) = \sum_{ev \in bin} \left(\left(\prod_{\eta \in nuis} w_{\eta}^{ev}(\eta) \prod_{\theta \in phys} w_{\theta}^{ev}(\theta) \right) \frac{\partial_k w_{\theta_j}^{ev}}{w_{\theta_j}^{ev}} \right) \text{ where } k \text{ is a subindex of } j.$$

Binning happens during runtime.



Treatment of systematic uncertainties

Bottom-up and top-down systematics

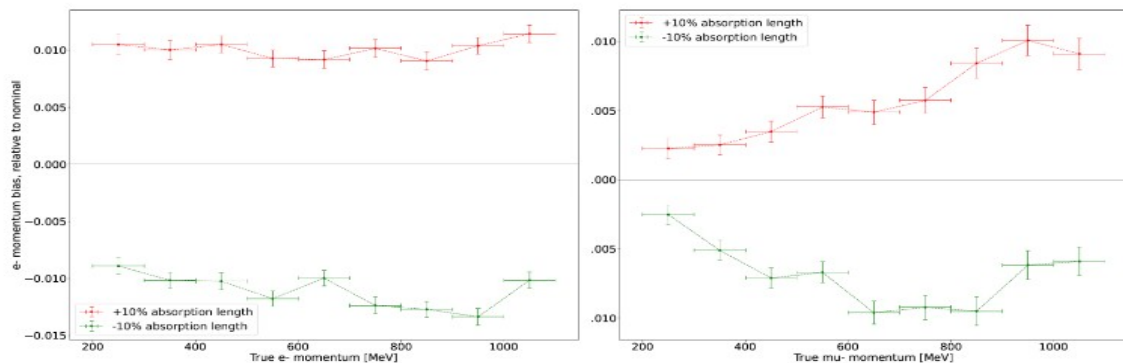
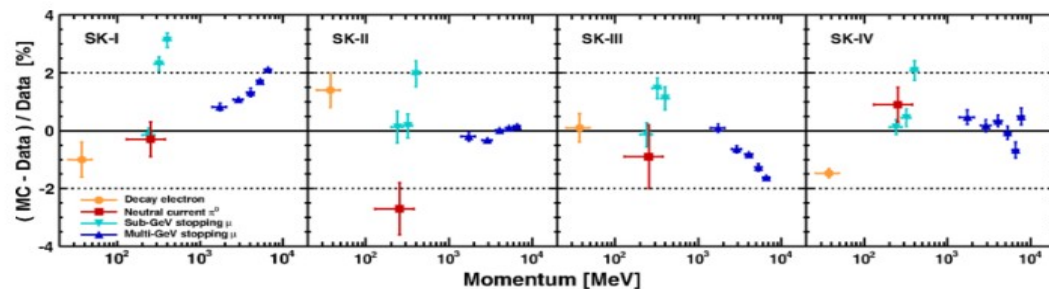
Sample dependence intrinsic uncertainties in the physics and limited statistics

• Top Down

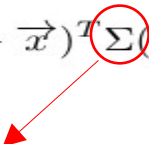
- Estimate systematics based on control samples
- Compare data and simulation
- Perfect when control sample matches signal
- Otherwise requires extrapolation and/or interpolation

• Bottom Up

- Determine uncertainty in detector response from uncertainty in input parameters
- Requires knowledge of all parameters and uncertainties
- No extrapolation and correlations can be understood



Top-down: analysis example

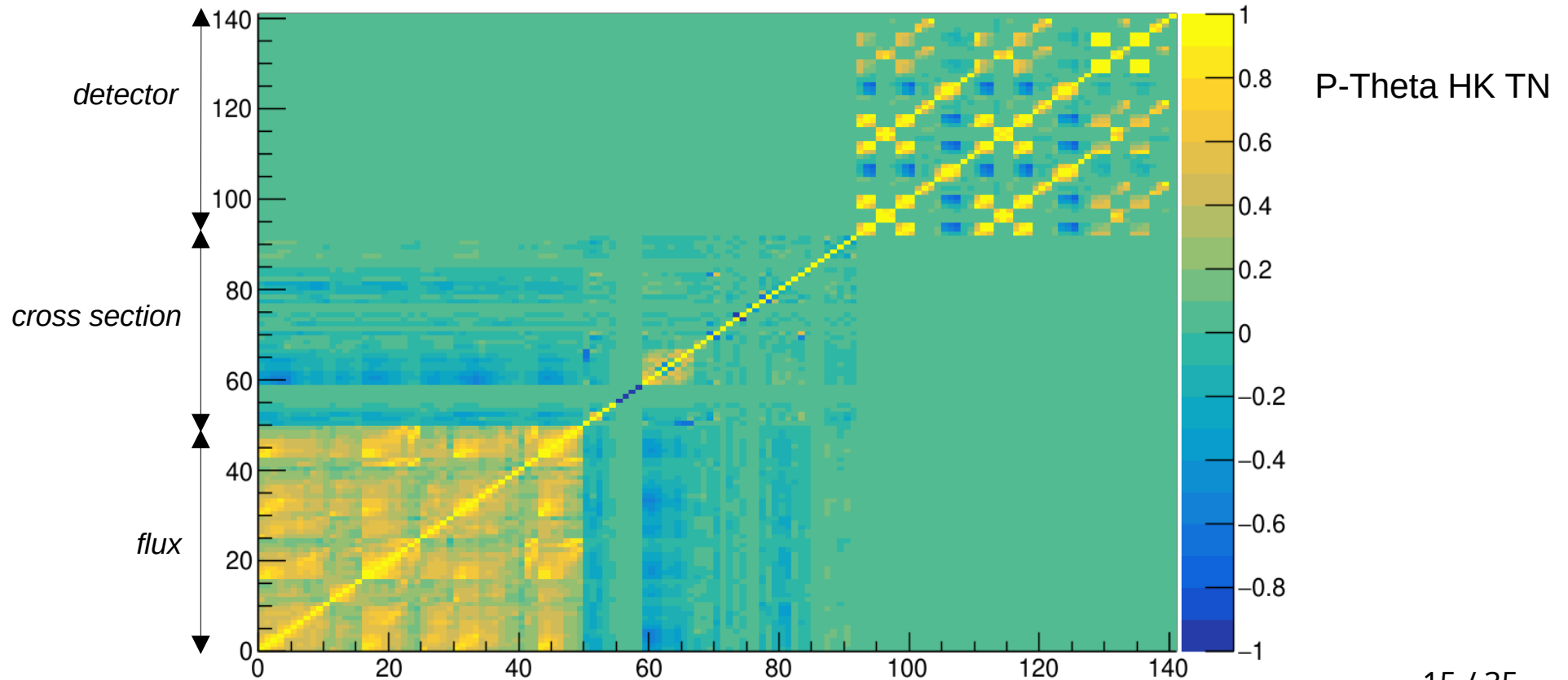
$$\chi^2 = \sum_{\text{Expmnt.}} \sum_{i \in \text{Bins}} \left(E'_i - O_i + O_i \cdot \log \left(\frac{O_i}{E'_i} \right) \right) + (\vec{\mu} - \vec{x})^T \Sigma (\vec{\mu} - \vec{x})$$


- Σ is the covariance matrix with all systematics and accounting for their correlation (off-diagonal terms)
- f_{ij} take into account the effect of nuisance parameters in terms of the event rates, s.t.

$$E'_i = E_i \left(1 + \sum_{j \in \text{Syst.}} f_{ij} \cdot x_j \right)$$

f_{ij} are the linearized (1st order) fractional change of the event rate w.r.t. the nuisance parameter j at bin i

Top-down: analysis example





Bottom-up: the physics

By definition, in bottom-up approach, there will be fewer correlations between nuisance parameters; correlations are removed by

- Independent calibration sources and detector features
- Redundant calibration to decouple effects

Enables

- More precise implementation of physics analysis
- Better understanding of impact of nuisance parameters, calibration and detector performance

Requires

- Dedicated simulations to compute parametrizations and weight functions
- More work and communication between calibration and physics

Going from top-down approach to bottom-up



Change of parametrization s.t. it makes “more diagonal” the covariance matrix and removes most of the correlations

In the case of top-down approach, the covariance matrix is defined for top-level variables which depend on several common parameters, thus exhibiting significant correlations. Another feature of this approach is that computing the f_{ij} using top-level variables is quite natural and straight-forward.

In the bottom-up approach, variables carrying the uncertainties are more fundamental, therefore their correlations are much smaller in the covariance matrix. On the other hand, the computation of the f_{ij} becomes a bit more complex in the sense that a single given nuisance parameter will affect more bins and in different ways (affects to several top-level variables).

The covariance matrix Σ is a positive, symmetry square-matrix, so \exists a matrix P s.t. $P^T \Sigma_D P = \Sigma$, where Σ_D is a diagonal matrix.

The nuisance term is invariant under a change of variables in which the covariance matrix is diagonal.

$$\vec{\eta} = P \vec{x} \quad (5)$$

$$\vec{x}^T \Sigma \vec{x} \longrightarrow \vec{\eta}^T \Sigma_D \vec{\eta} \quad (6)$$

$$\Sigma_{BU} = \begin{pmatrix} \blacksquare & & & & & \\ & \blacksquare & & & & \\ & & \dots & & & \\ & & & \blacksquare & & \\ & & & & \blacksquare & \\ & & & & & \blacksquare \end{pmatrix}$$

Bottom-up: pynu implementation

All (nuisance) parameters are linked to a function ($\vec{w}_\eta(\eta)$) which computes the weights associated to that parameter for each event in the Monte Carlo simulation used to fit the data.

$$\vec{w}_\eta: \mathbb{R} \longrightarrow \mathbb{R}^{m_\eta} \quad (11)$$


$$\eta \longmapsto \vec{w}_\eta(\eta) = (E(\eta), ip(\eta), \dots) \quad (12)$$

In this approach, correlations between analysis-relevant variables are implemented implicitly and allow for any parametrization.

Binning and number of expected events is computed at runtime after all the weights are calculated. In turn, it means that there is no linearization in terms of the event rate fraction.

$$E'_i = \sum_{ev \in bin} \prod_{\eta \in nuis} w_\eta^{ev}(\eta) \prod_{\theta \in phys} w_\theta^{ev}(\theta)$$

Of course, explicit correlation of nuisance parameters can be implemented as well simply generalizing the previous, that is each diagonal block has its own vector multivariate function

$$\vec{w}_\eta: \mathbb{R}^{i_\eta} \longrightarrow \mathbb{R}^{m_\eta}$$
$$\vec{\eta} \longmapsto \vec{w}_\eta(\vec{\eta}) = (E(\vec{\eta}), ip(\vec{\eta}), \dots)$$


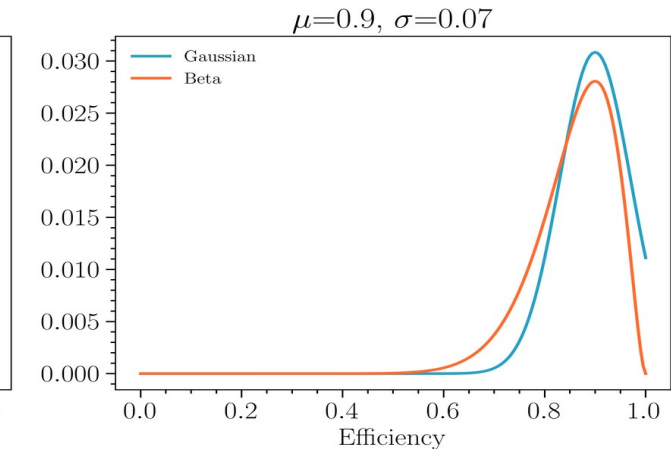
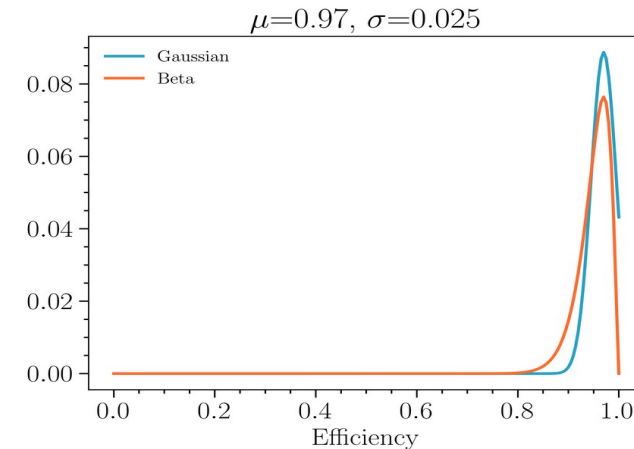
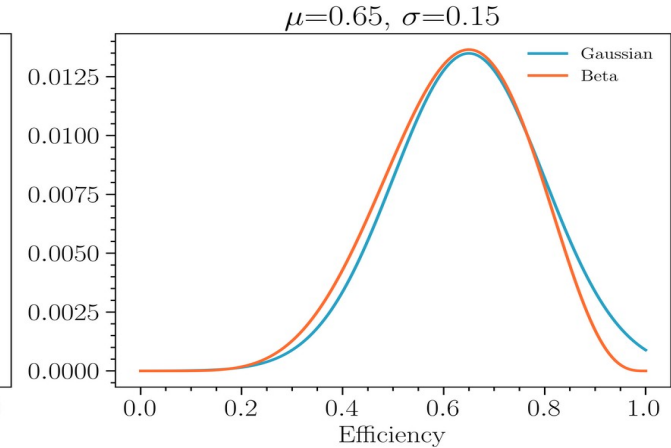
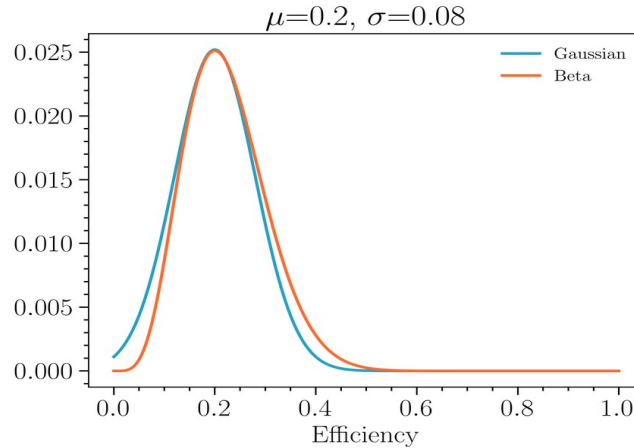
These functions are exactly the same as those obtained from calibration analyses

Prior distributions

In addition to the physical/philosophical motivation of bottom-up approach, it also brings some computation advantages

Being able to de-correlate most of nuisance parameters and following a bottom-up approach, also enables the capability to plug in more meaningful priors, beyond Gaussian distributions and avoiding nonphysical values

Pynu accomodates any prior dsitribution for any parameter





Fitting strategies

Profiling and marginalizing (inference)

The final result is the same in most cases and their difference relies on how the model parameters are treated during the fitting

Both are very time-consuming processes (weeks to months to run on clusters)

- + Part of the reason are oscillations

- + The rest is due to the complexity of the numerical optimization problem of the likelihood over $O(10^2)$ parameters/dimensions

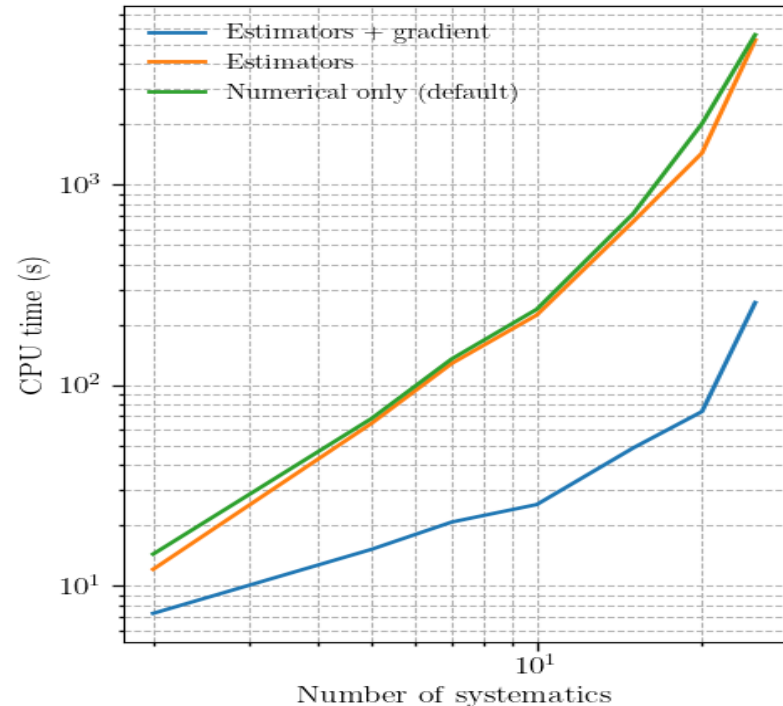
Profiling

In profiling, we assume some parameters are the physics we want to measure and for which we obtain their distribution and best fit values; others are considered nuisance or systematics and for which only the best fit value matters (i.e. the mode of the distribution)

Therefore, for every point of the physics parameter space the systematics must vanish the gradient of the log-likelihood

→ Being able to compute the gradient analytically instead of numerically, makes the minimization much faster.

→ First order estimators and bounds improve the performance by reducing the volume of the parameter space to probe.



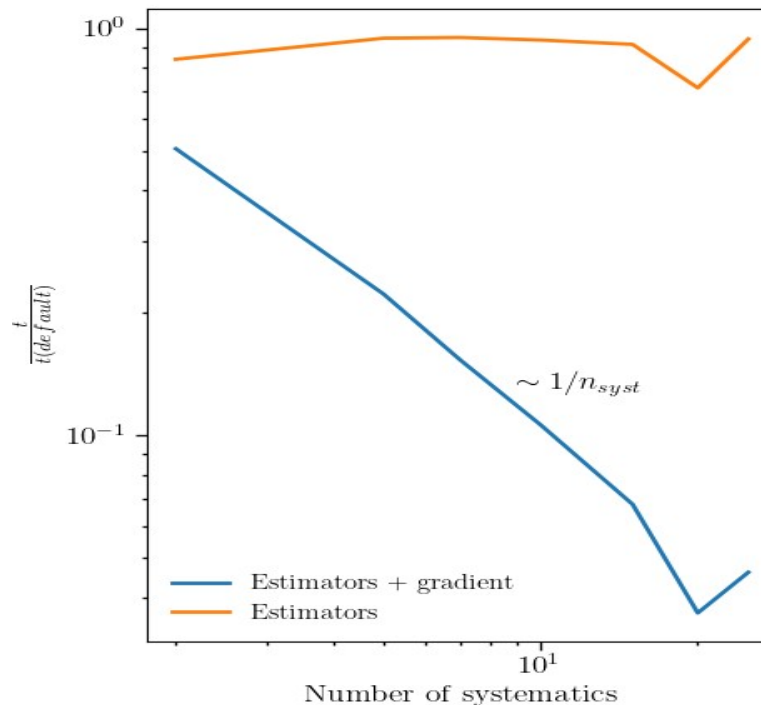
Profiling

In profiling, we assume some parameters are the physics we want to measure and for which we obtain their distribution and best fit values; others are considered nuisance or systematics and for which only the best fit value matters (i.e. the mode of the distribution)

Therefore, for every point of the physics parameter space the systematics must vanish the gradient of the log-likelihood

→ Being able to compute the gradient analytically instead of numerically, makes the minimization much faster.

→ First order estimators and bounds improve the performance by reducing the volume of the parameter space to probe.





Marginalizing

In marginalization, all parameters are treated in the same way in the analysis, we want to obtain their distribution and best fit values. Afterwards, some of these parameters are considered nuisance of the analysis and marginalized/integrated over.

Therefore, we don't minimize the log-likelihood but infer the overall distribution.

Metropolis-Hastings Markov Chain Monte Carlo (Mach3) or numerical integration using Choleski decomposition (P-Theta) is the widely used method for this integration, but there exist methods which scale better with dimensionality (validation ongoing).

We will focus on the implementation of Hamiltonian Monte Carlo. This is an active field mainly in machine learning which originated in lattice field theory and is one of the most effective/efficient MC integration algorithms currently.

Marginalizing

Hamiltonian Monte Carlo

Instead of random walks of MCMC, HMC simulates predictable trajectories (of random but constant energy-probability) probing the entire probability distribution and following Hamiltonian-like dynamics

$$H(\theta, p) = U(\theta) + K(p)$$

with

$$U(\theta) = \pi(\theta) \quad \text{Potential energy as the target negative log likelihood}$$

Thetas are the parameters of the analysis and p are the conjugate variables (analog of Hamiltonian formulation momenta)

$$K(p) = \frac{1}{2} p M^{-1} p^T \quad \text{"Gaussian" kinetic energy from which to draw momenta to probe the distribution}$$

Hamilton eqs. are solved easily and efficiently by having the information of the gradient of the log-likelihood and with methods from I-QCD

$$\dot{\theta} = \frac{\partial H}{\partial p}, \dot{p} = -\frac{\partial H}{\partial \theta}$$

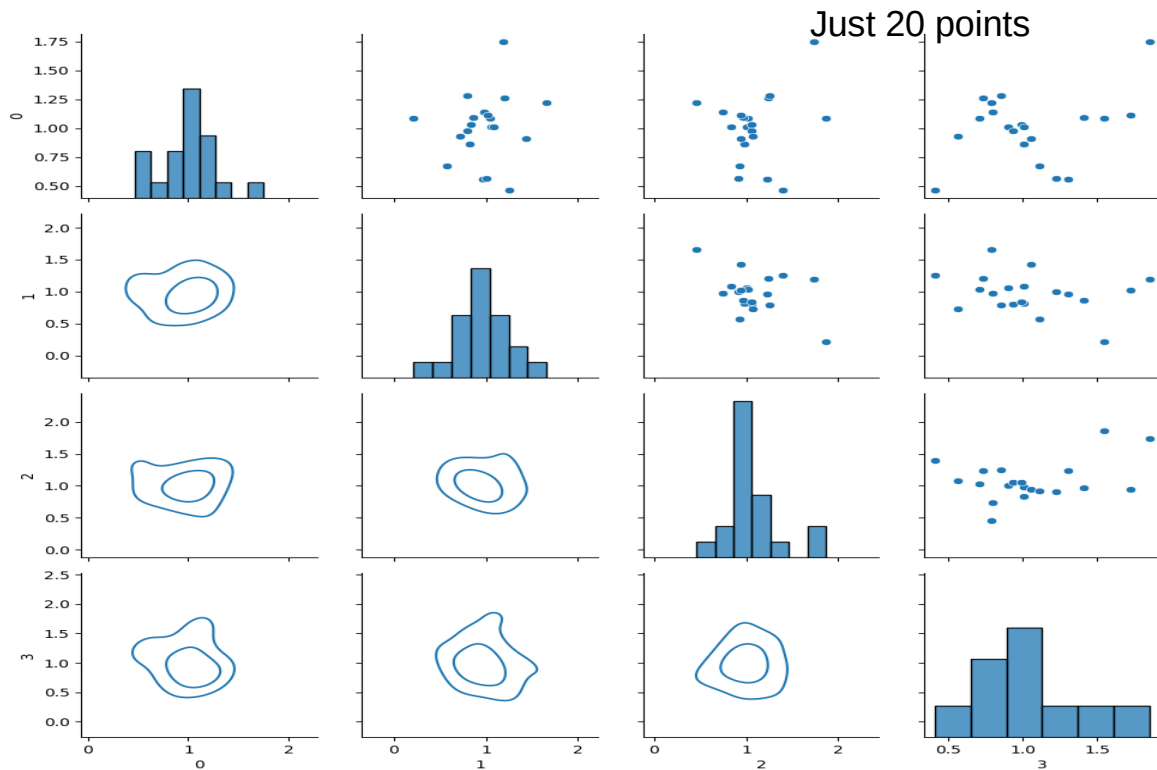
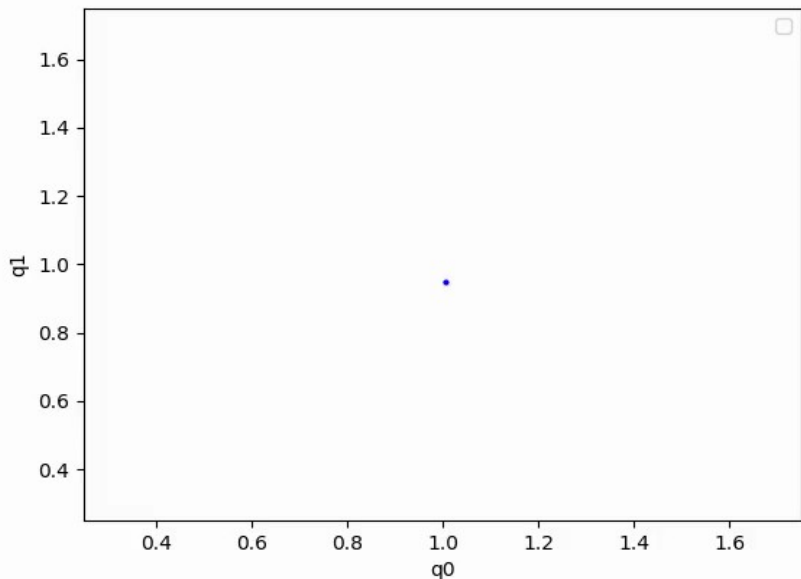
solutions for thetas follow then the canonical distribution, which is our target distribution

$$P(\theta, p) \propto \exp(-H(\theta, p)) = \exp(-\pi(\theta) + K(p))$$

Marginalizing

Hamiltonian Monte Carlo in pynu

Several versions of HMC have already been implemented in pynu and are being further developed and adapted to this problem (*paper in preparation*)



Marginalizing

Hamiltonian Monte Carlo vs Metropolis-Hastings

- Usual MH-MCMC scales with dimension as d^2 whereas HMC goes as $d^{5/4}$
It converges to the target distribution much faster
- Acceptance rate in MH-MCMC is between 20% to 25%, in HMC is 80% to 95% (only due to error in trajectory computation)
- One of the main obstacles in MH-MCMC is that autocorrelation times between points is larger and can only be solved with more points. In HMC, correlation between points decreases much quicker
- Additionally, we make a first order analytical approximation of the best fit value (mode of the distribution). By starting the trajectories near the modes, they can probe efficiently the overall distribution



Other approaches for the future

- Stein Variational gradient descent (partially implemented). A type of variational inference method
(<https://papers.nips.cc/paper/2016/file/b3ba8f1bee1238a2f37603d90b58898d-Paper.pdf>)
- Unbinned likelihood fit (roughly implemented)
- Likelihood-free inference

A word on the motivation of having a flexible framework to make joint fits

Philosophy of combined data analyses with the atmospheric showcase

Different experiments: independent measurements

Joint fit: Enhanced precision systematics cancellation/reduction

In the end we want both, a precise and reliable measurement

Will ordering will be measured and the octant of θ_{23} ?

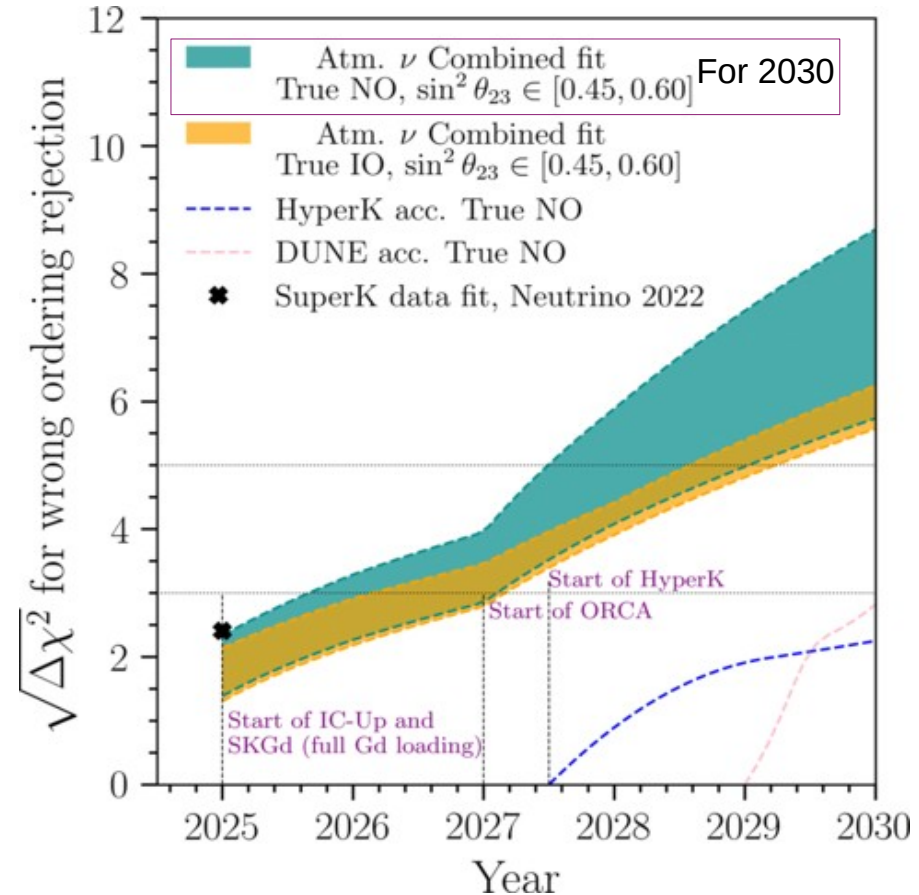
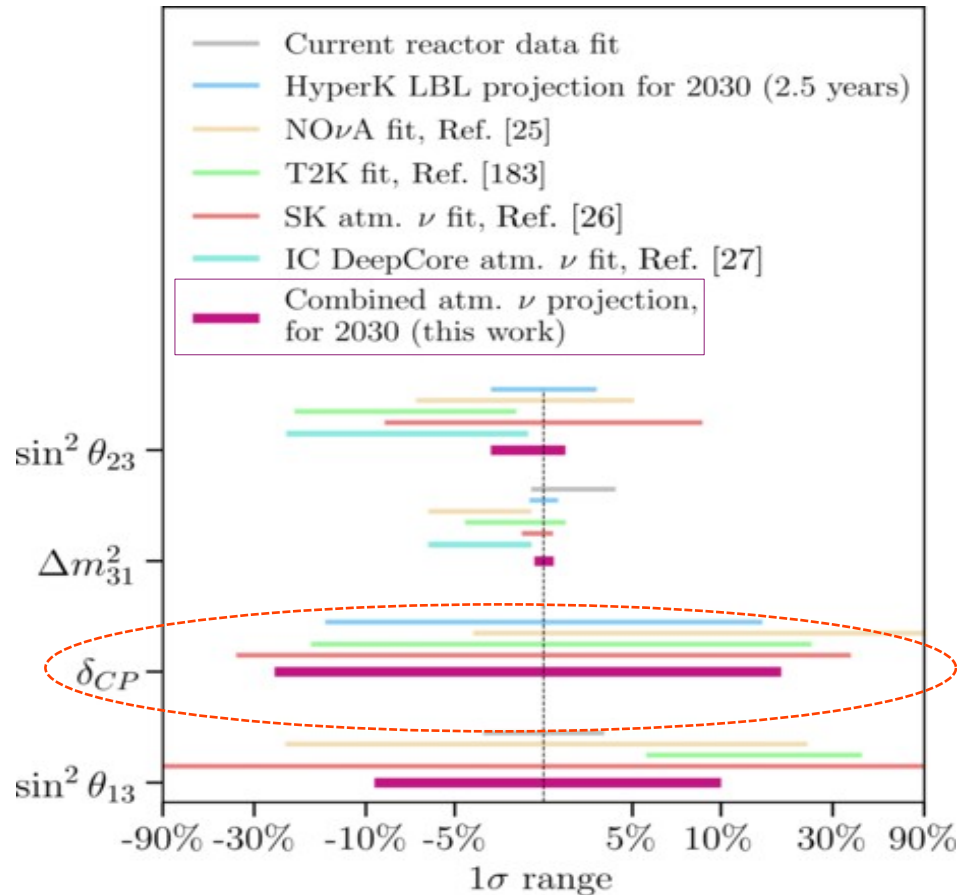
Learn more about low energy uncertainties

Inputs from previous LBL experiments, NovA and T2K (and MINOS)

Joint analysis of accelerator and atmospheric, the official SK+T2K joint fit will be out soon
T2K+NOvA official joint fit is ongoing

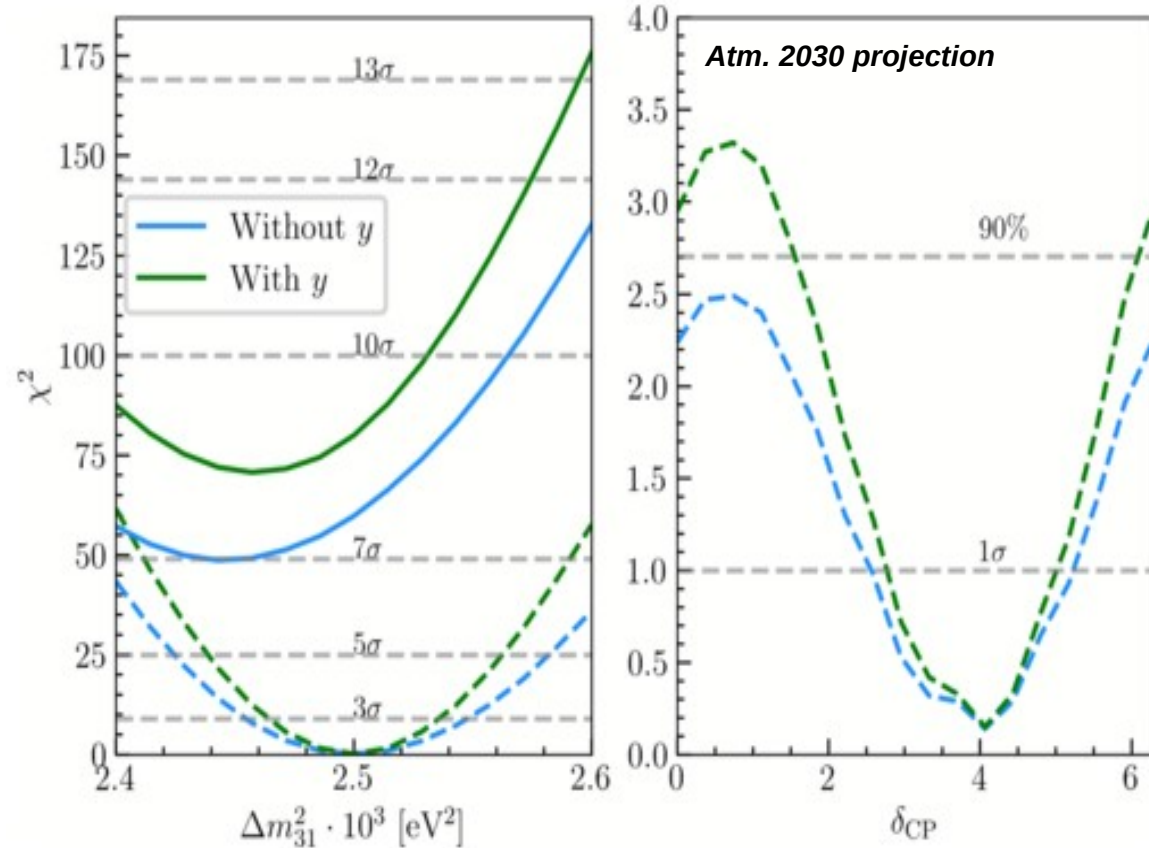
Combined fits and the start of HyperK -- atmospheric

Atmospherics joint fit for 2030: IC-UP (5 years), ORCA (3 years), SK (current+5 years) and HK (2.5 years)



Combined fits and the start of HyperK -- atmospherics

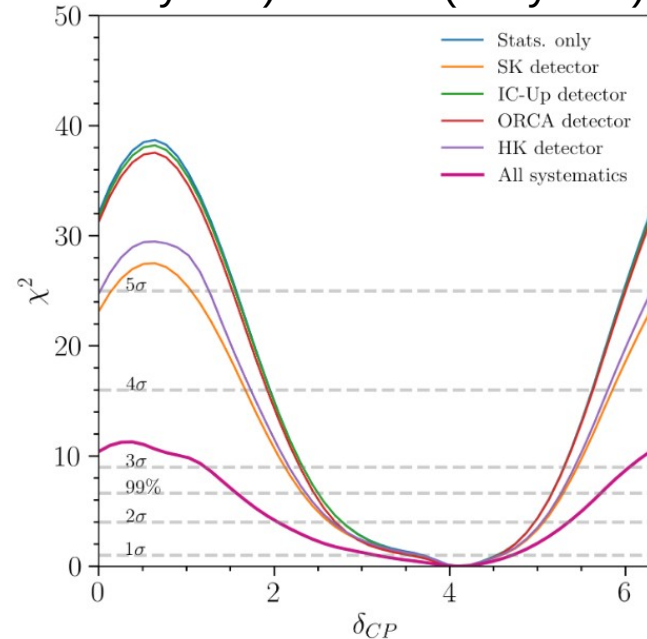
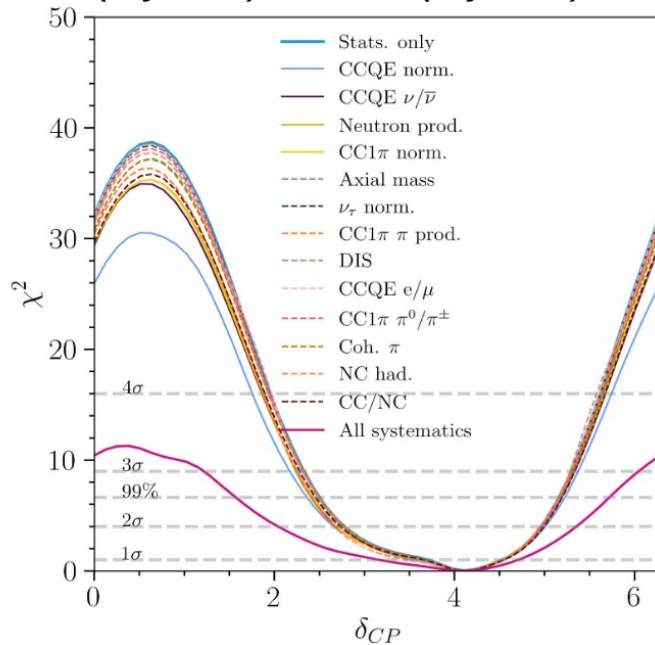
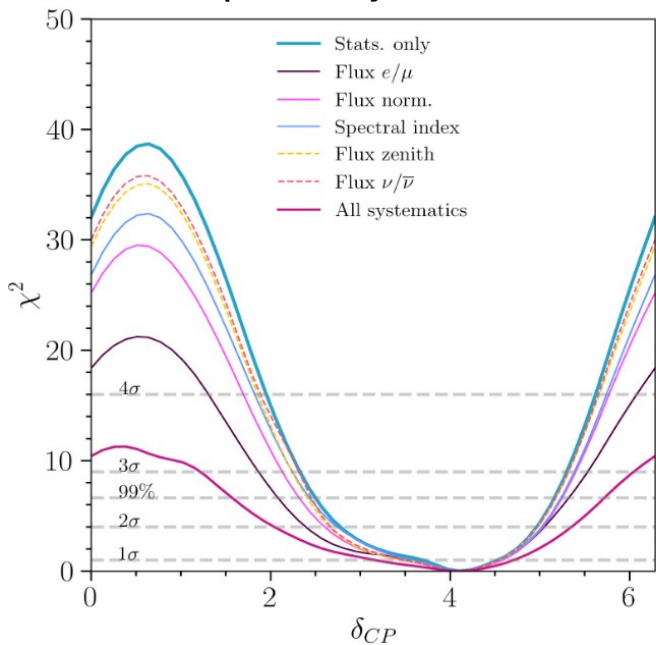
Actually, **IceCube-Upgrade (5 years)**
and ORCA (3 years) alone will “easily”
determine the neutrino mass ordering
around the relevant time for the first data
of HK



Combined fits and the start of HyperK -- atmospheric

Atmospherics have a huge sensitivity to the CP phase, if it weren't for the systematics...

Atmospherics joint fit for 2030: IC-UP (5 years), ORCA (3 years), SK (current+5 years) and HK (2.5 years)

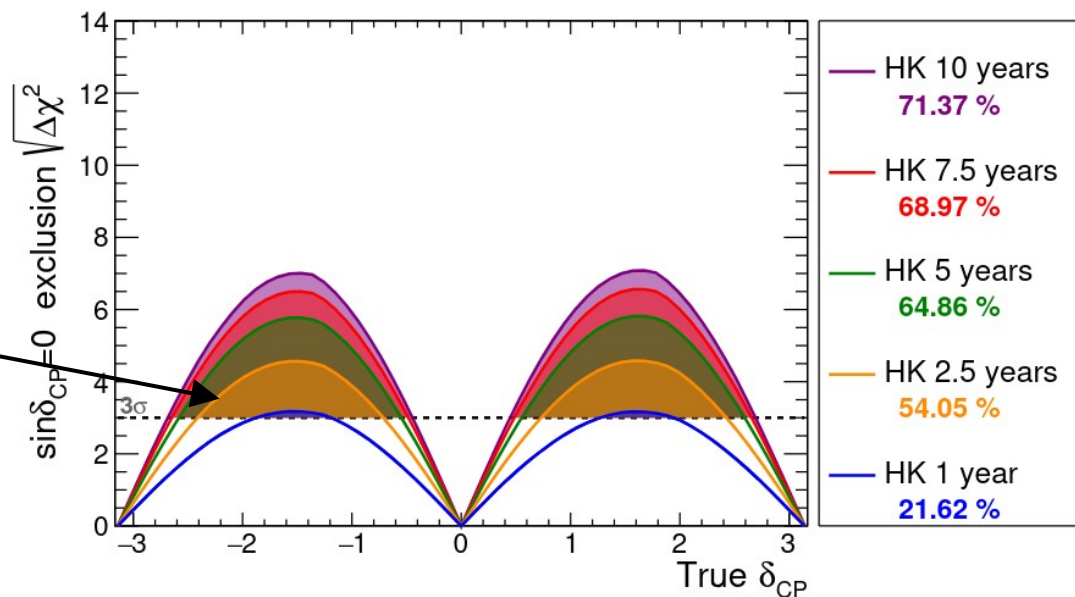
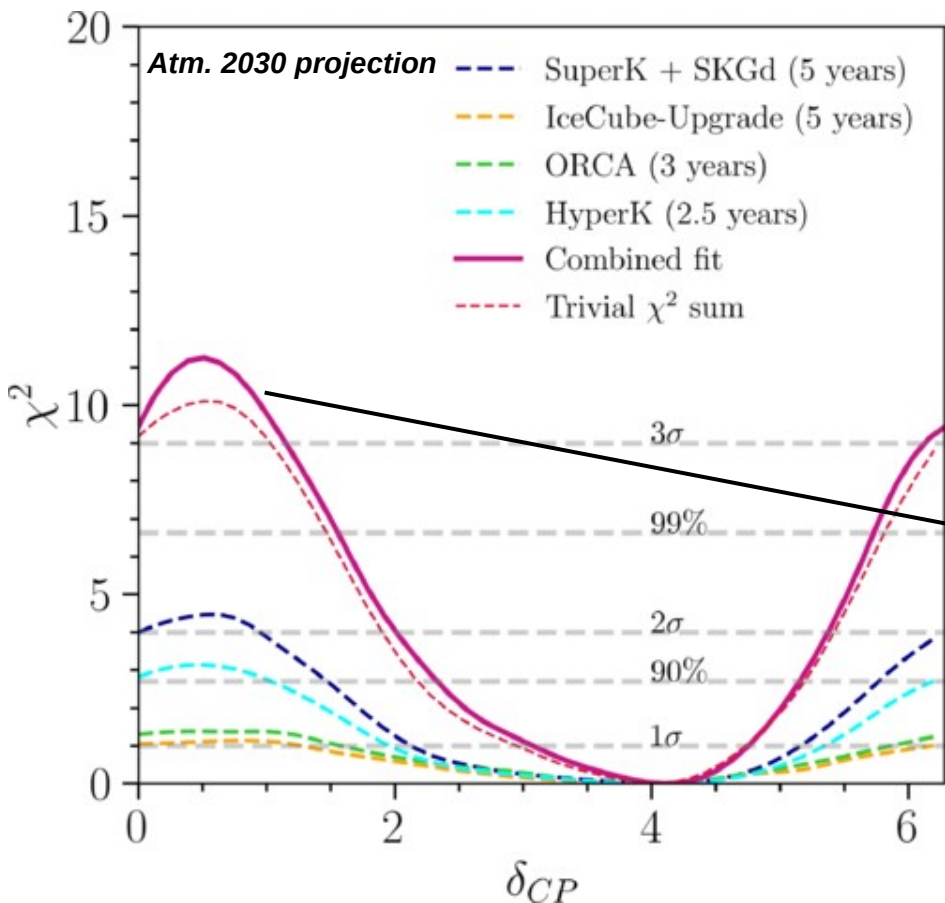


Main showstopper are the flux uncertainties but it partially benefits from cross section measurements and detector control.

Ancillary measurements for flux much like for the accelerator neutrino beams are much needed

Combined fits and the start of HyperK -- atmospheric

Nevertheless, in 2030, HK LBL and combined atmospheric will have comparable sensitivities to δ_{CP}



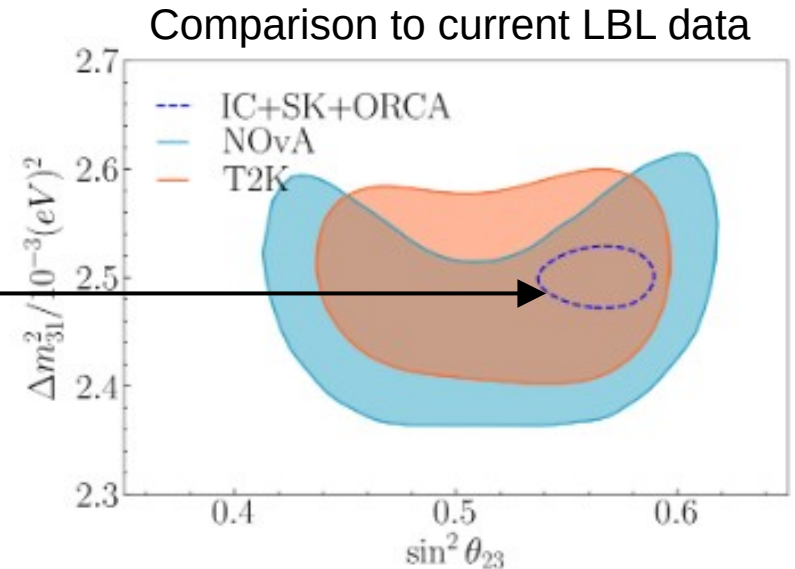
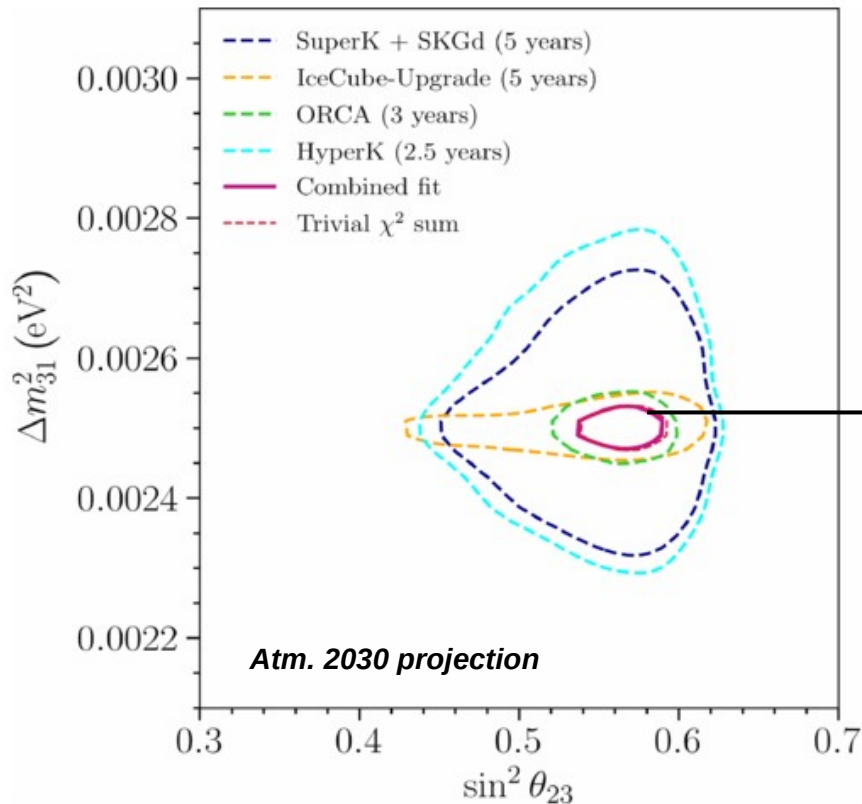
Hyper-K preliminary

True normal ordering (known), T2K 2020 systematics

$$\sin^2\theta_{13}=0.0218\pm 0.0007, \sin^2\theta_{23}=0.528, \Delta m_{32}^2=2.509\times 10^{-3}\text{eV}^2/c^4$$

Combined fits and the start of HyperK -- atmospheric

And very similar for θ_{23} . Example of how a combined fit can provide very valuable input to next-generation experiments to be optimized and their physics programs accelerated



Combined fits and the start of HyperK – take out message

- Experiments providing independent measurements is crucial
- Combined fits check for consistency and provide potentially more precise measurements
 - Ideally, look for complementarities and sharing/cancellation of systematics
- They guide your next experiment and extract the most out of the current ones. Around HyperK:
 - Global fits (as official/realistic) by the start of HK
 - SK+T2K
 - ND280+IWCD+HK(atm+LBL)
 - ...
 - Not only “high-energy”, HK (solars+reactors?, DSNB)
 - Not only 3-flavor scenario
- Pynu provides a framework to do it, the only remaining thing would be to implement your experiment(s)
 - Same framework for phenomenology studies, single and combined data fits

