

GNA: DATA FLOW APPROACH FOR THE NEUTRINO OSCILLATION EXPERIMENTS

Maxim Gonchar for the GNA group

DLNP, Joint Institute for Nuclear Research

Innovative Workflows in Astro- & Particle Physics
March 10, 2021



Russian
Science
Foundation

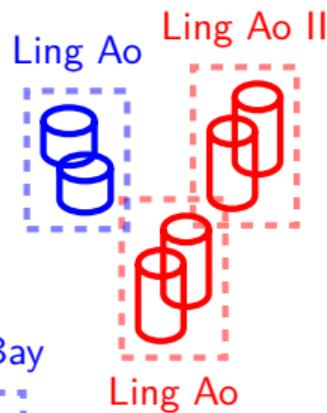
The work is supported by RSF #21-42-00023

v1.1.1

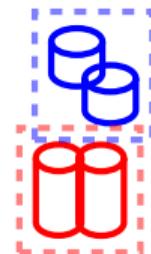


WHY GNA: DAY BAY EXPERIMENT

Far site



Daya Bay



Daya Bay

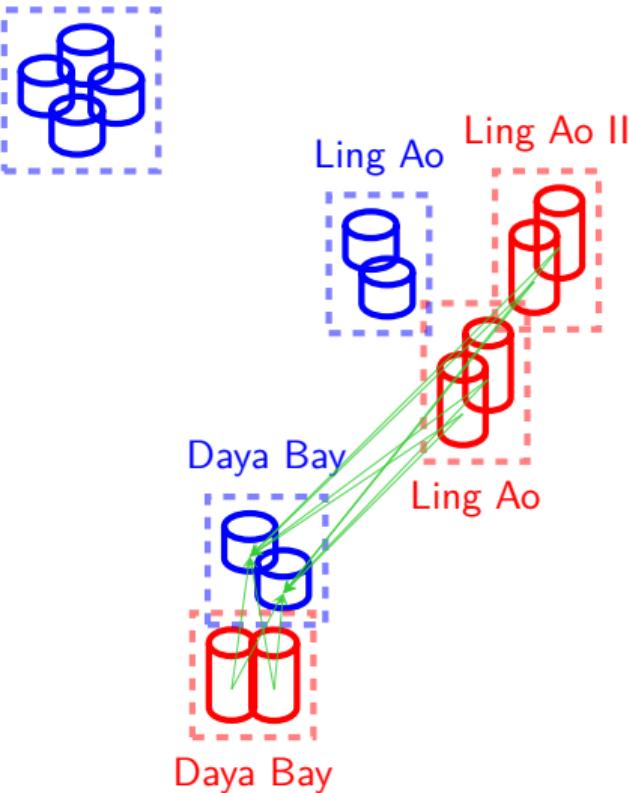
$$\vec{N}^d = \vec{B}^d + \left(\prod_m C_m \right) \times \\ \times \sum_t \dots \sum_r \dots \int \int \dots \sum_i \dots \sum_c P_c(L_r^d, \dots)$$



WHY GNA: DAY BAY EXPERIMENT

Far site

Count: 12



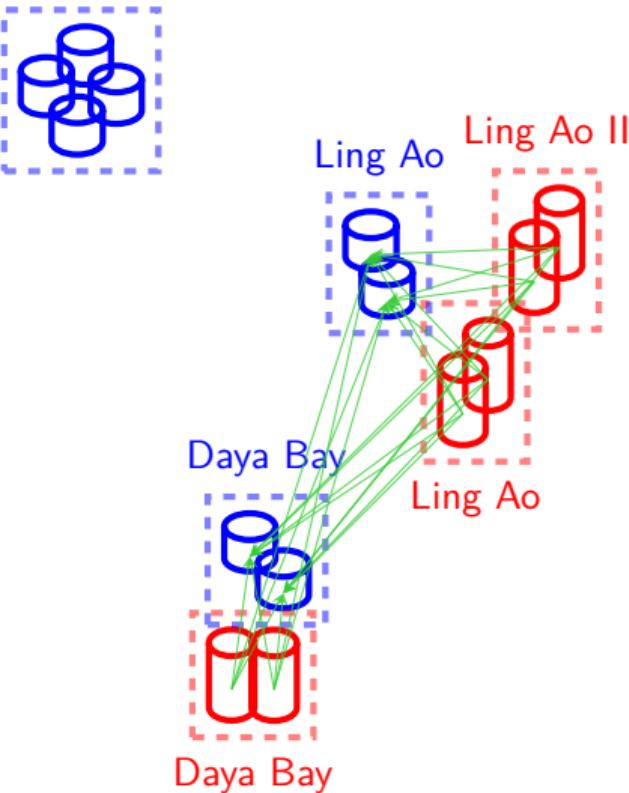
$$\vec{N}^d = \vec{B}^d + \left(\prod_m C_m \right) \times \\ \times \sum_t \dots \sum_r \dots \int \int \dots \sum_i \dots \sum_c P_c(L_r^d, \dots)$$



WHY GNA: DAY BAY EXPERIMENT

Far site

Count: 24



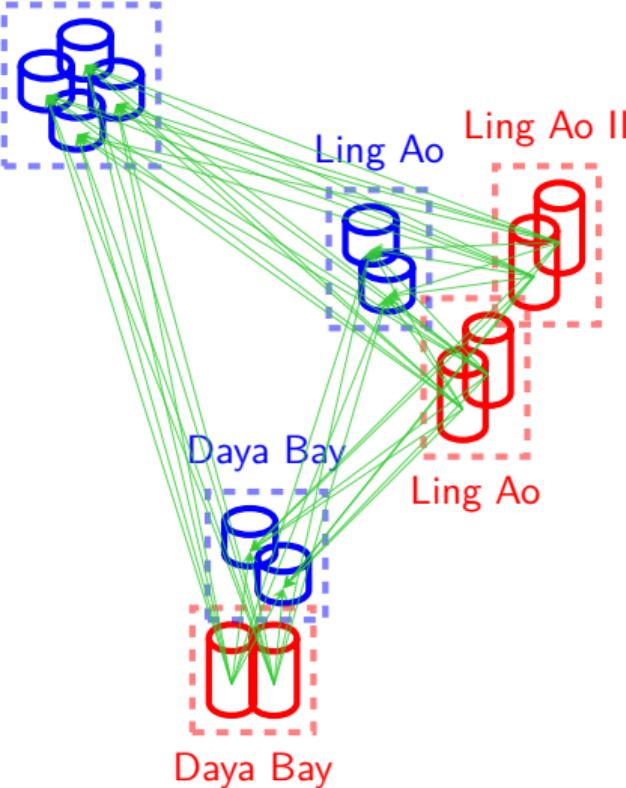
$$\vec{N}^d = \vec{B}^d + \left(\prod_m C_m \right) \times \\ \times \sum_t \dots \sum_r \dots \int \int \dots \sum_i \dots \sum_c P_c(L_r^d, \dots)$$



WHY GNA: DAY BAY EXPERIMENT

Far site

Count: 48



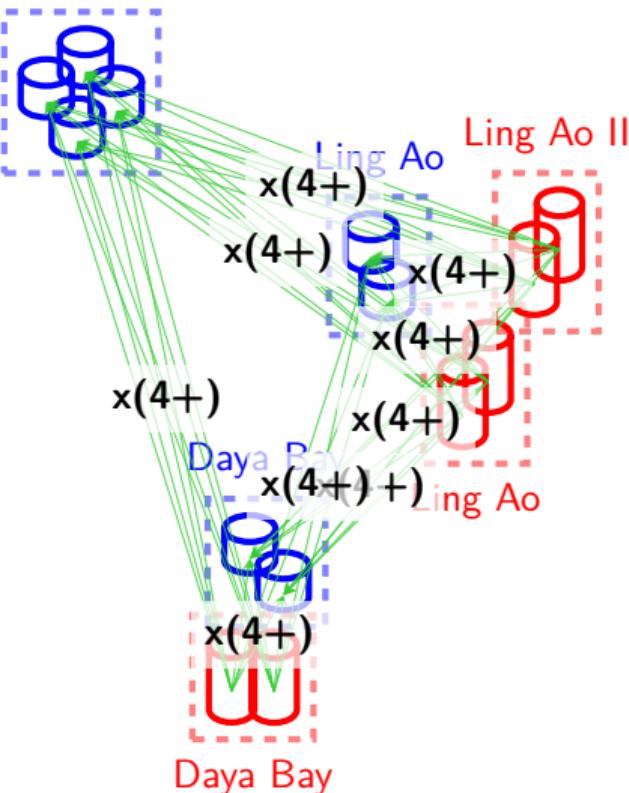
$$\vec{N}^d = \vec{B}^d + \left(\prod_m C_m \right) \times \\ \times \sum_t \dots \sum_r \dots \int \int \dots \sum_i \dots \sum_c P_c(L_r^d, \dots)$$



WHY GNA: DAY BAY EXPERIMENT

Far site

Count: 192+

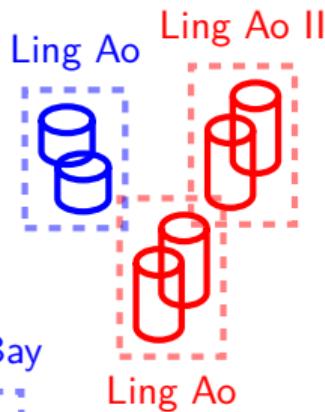


$$\vec{N}^d = \vec{B}^d + \left(\prod_m C_m \right) \times \sum_t \dots \sum_r \dots \int \int \dots \sum_i \dots \sum_c P_c(L_r^d, \dots) \dots$$

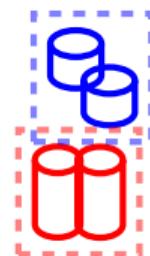


WHY GNA: DAY BAY EXPERIMENT

Far site



Daya Bay



Daya Bay

$$\vec{N}^d = \vec{B}^d + \left(\prod_m C_m \right) \times$$

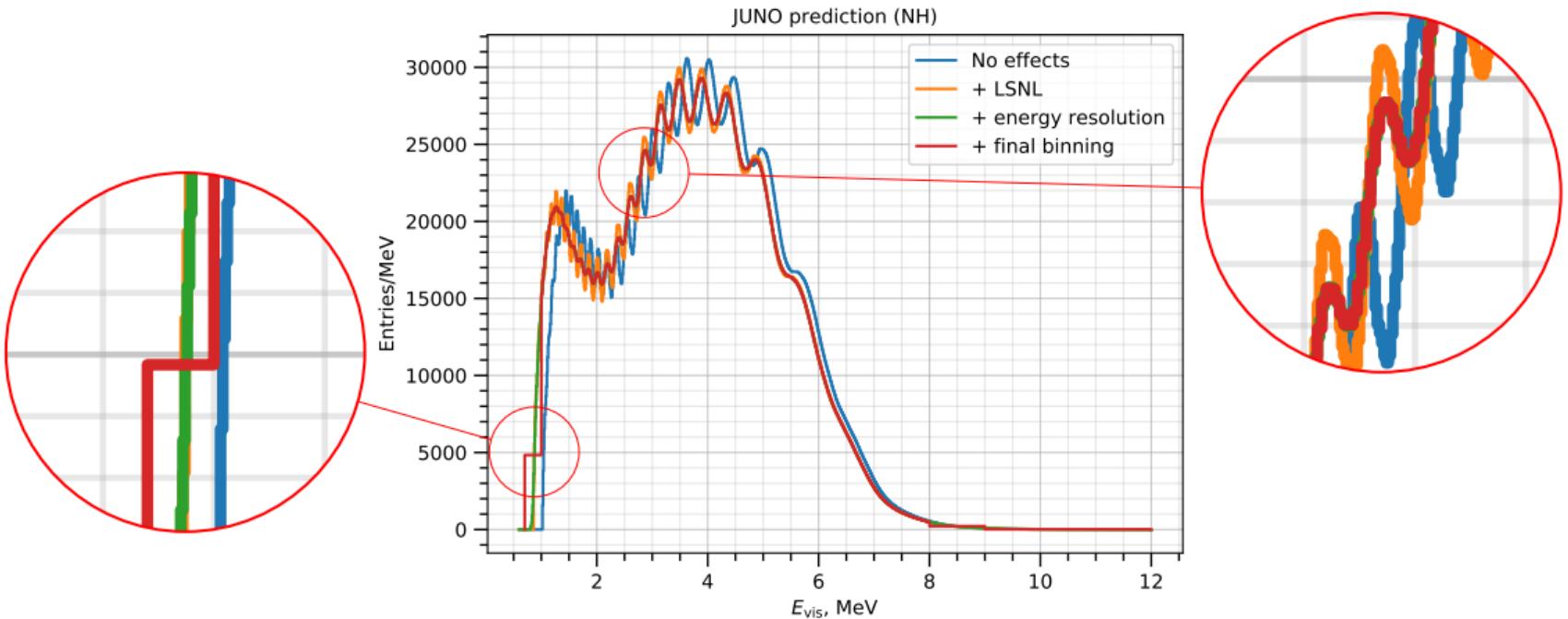
?

$$\times \sum_t \dots \sum_r \dots \int_i \dots \sum_c P_c(L_r^d, \dots)$$



EXAMPLE JUNO SPECTRUM

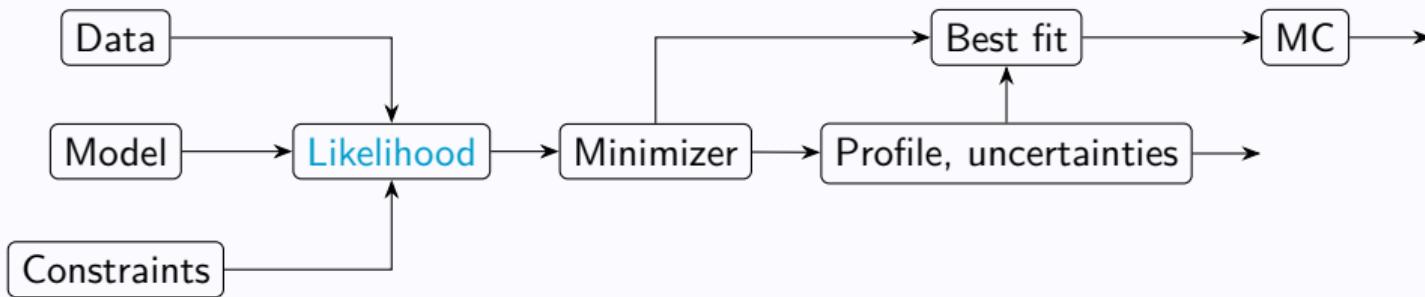
Example JUNO spectrum for LSNL + 200 sub-detectors with own energy resolutions





DEEP LEARNING VS. MODEL FITTING

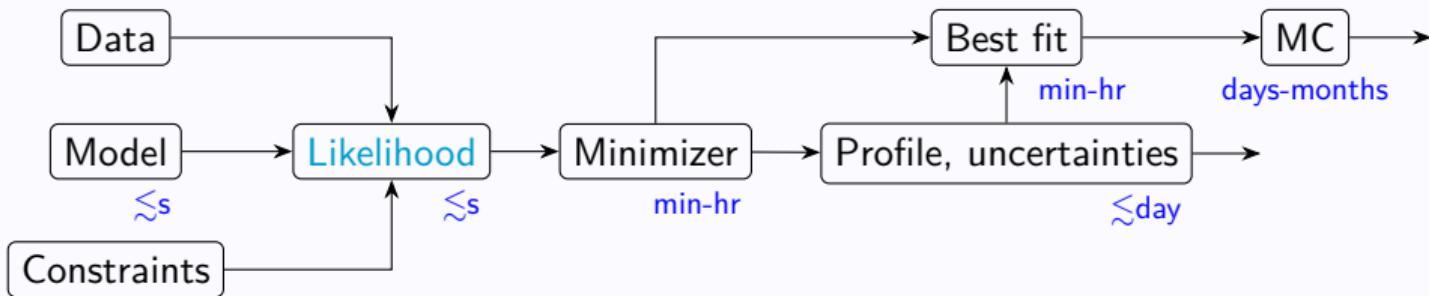
Typical workflow





DEEP LEARNING VS. MODEL FITTING

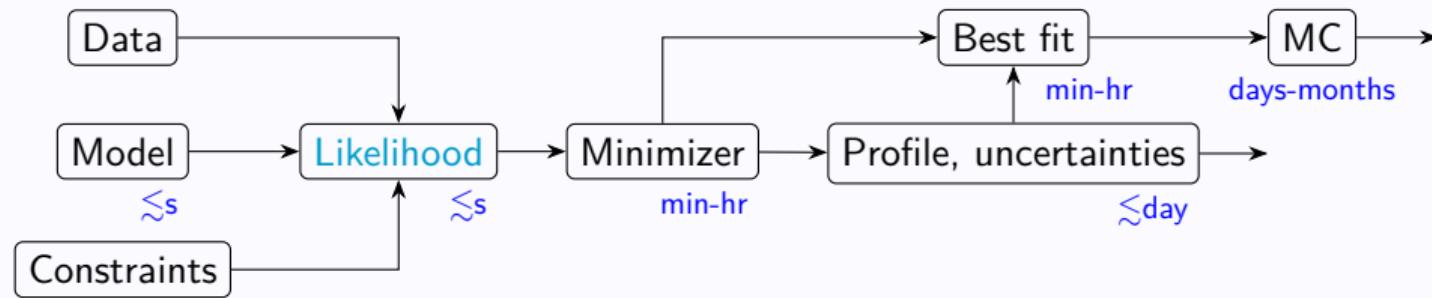
Typical workflow





DEEP LEARNING VS. MODEL FITTING

Typical workflow



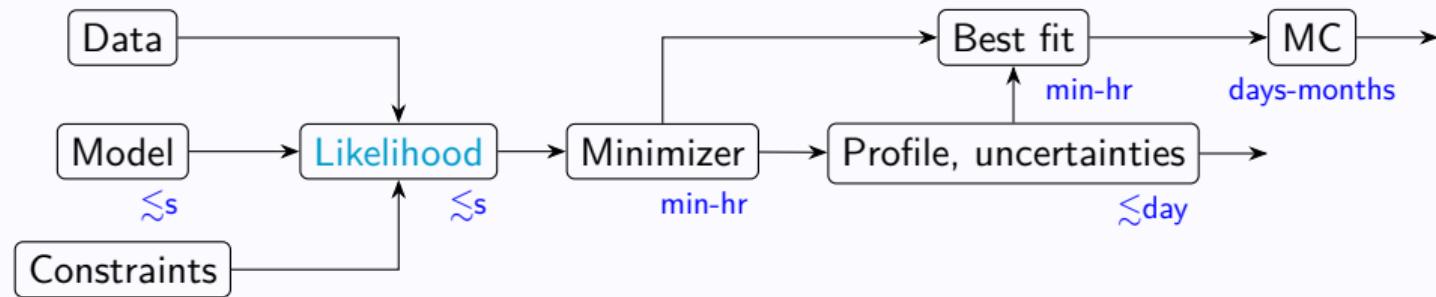
Elements

DL	Trivial, homogeneous
HEP	Non-trivial, heterogeneous



DEEP LEARNING VS. MODEL FITTING

Typical workflow

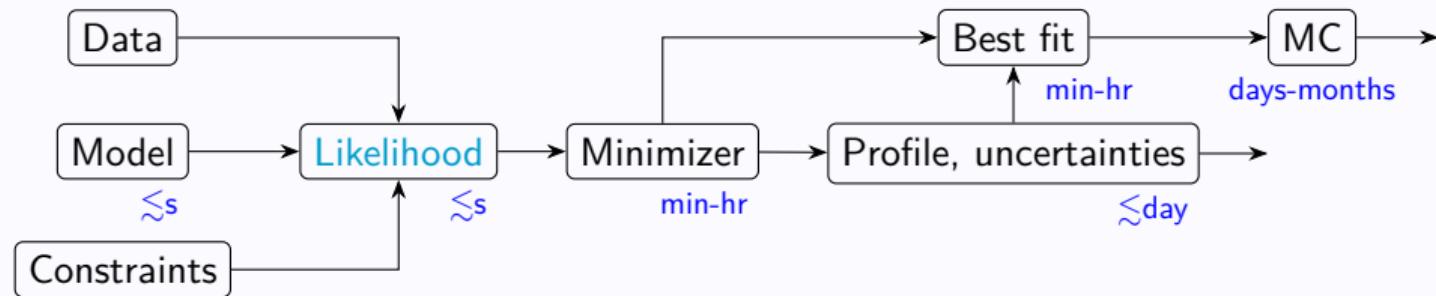


	Elements	Data scale
DL	Trivial, homogeneous	Batches
HEP	Non-trivial, heterogeneous	Whole dataset



DEEP LEARNING VS. MODEL FITTING

Typical workflow

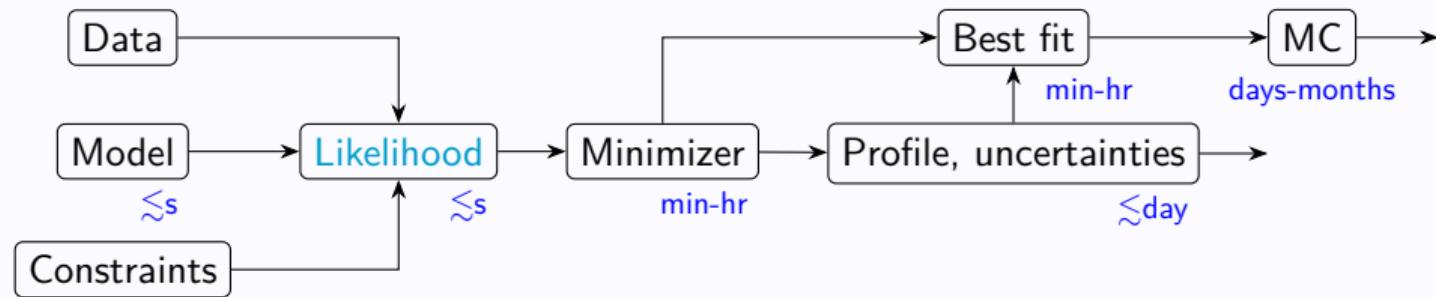


	Elements	Data scale	Composed loss
DL	Trivial, homogeneous	Batches	Any
HEP	Non-trivial, heterogeneous	Whole dataset	Simultaneous + constraints



DEEP LEARNING VS. MODEL FITTING

Typical workflow

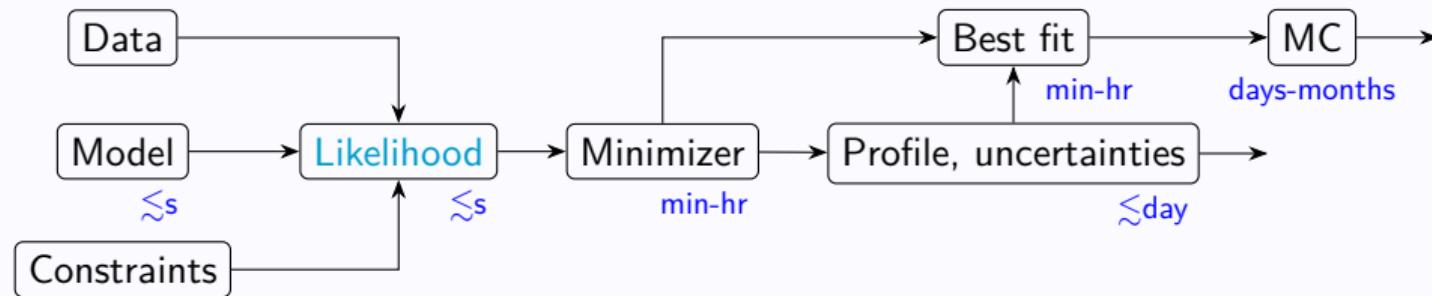


	Elements	Data scale	Composed loss	Minimization
DL	Trivial, homogeneous	Batches	Any	Local, 1 st derivative (gradient)
HEP	Non-trivial, heterogeneous	Whole dataset	Simultaneous + constraints	Global, 2 nd derivative (quasi-Newton)



DEEP LEARNING VS. MODEL FITTING

Typical workflow



	Elements	Data scale	Composed loss	Minimization	Uncertainties
DL	Trivial, homogeneous	Batches	Any	Local, 1 st derivative (gradient)	
HEP	Non-trivial, heterogeneous	Whole dataset	Simultaneous + constraints	Global, 2 nd derivative (quasi-Newton)	Hesse, profiling, MC



GNA GOALS

GNA: GLOBAL NEUTRINO ANALYSIS

Variables

- Easy access to all variables.
- Combination: merge variables for partially dependent models.
- Lazy evaluation.



GNA GOALS

GNA: GLOBAL NEUTRINO ANALYSIS

Variables

- Easy access to all variables.
- Combination: merge variables for partially dependent models.
- Lazy evaluation.

Internal data and output data

- Easy access to all the internals.



GNA GOALS

GNA: GLOBAL NEUTRINO ANALYSIS

Variables

- Easy access to all variables.
- Combination: merge variables for partially dependent models.
- Lazy evaluation.

Structure

- Concise and readable model definition.
- Aggressive refactoring.
- Introspection, labels, etc.

Internal data and output data

- Easy access to all the internals.



GNA GOALS

GNA: GLOBAL NEUTRINO ANALYSIS

Variables

- Easy access to all variables.
- Combination: merge variables for partially dependent models.
- Lazy evaluation.

Internal data and output data

- Easy access to all the internals.

Structure

- Concise and readable model definition.
- Aggressive refactoring.
- Introspection, labels, etc.

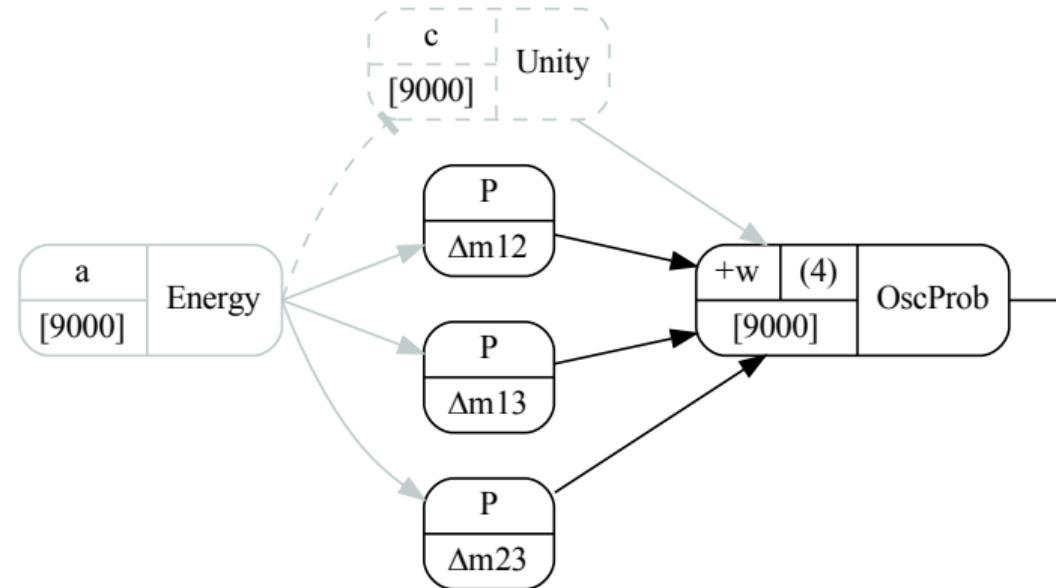
Timeline

- Analysis: 2012–
- Formulation: 2013
- First implementation: 2014
- Current version: 2016–



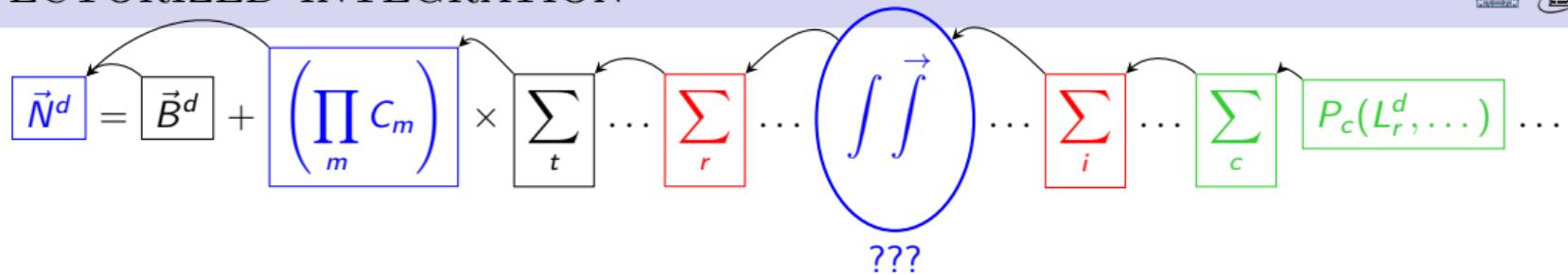
GNA STACK

- C++ Eigen — data representation, linear algebra, vectorized algorithms, etc.
- ROOT — MINUIT, reflection to Python.
- Boost — ptr_container, etc.
- Python 3 and:
 - ▶ numpy,
 - ▶ matplotlib,
 - ▶ pygraphviz,
 - ▶
- <https://git.jinr.ru/gna/gna>



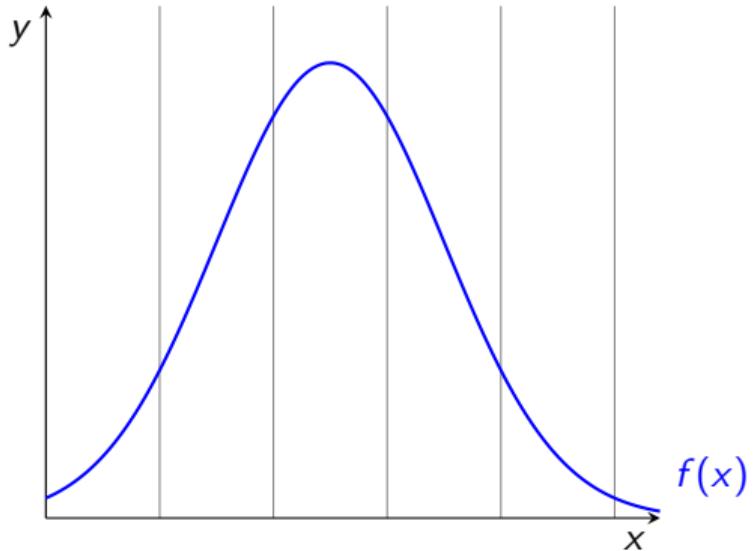
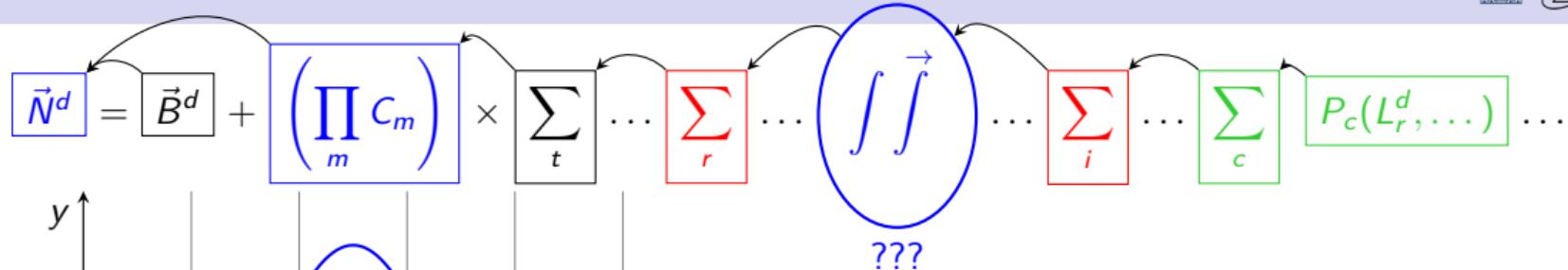


VECTORIZED INTEGRATION



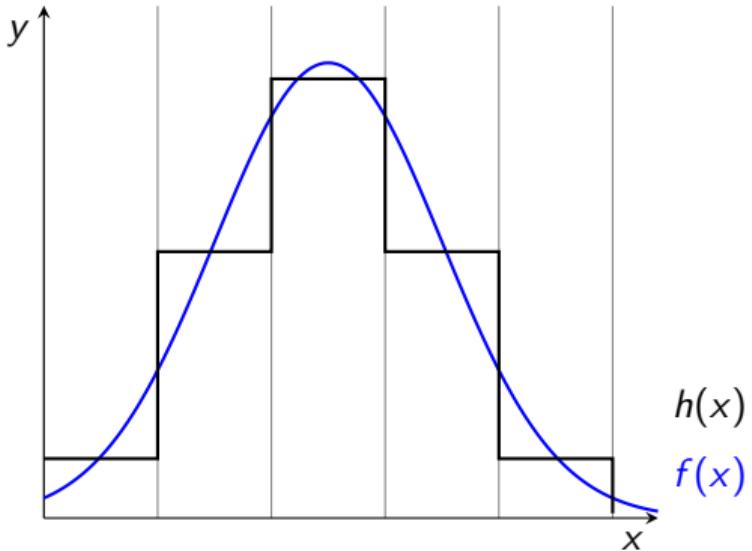
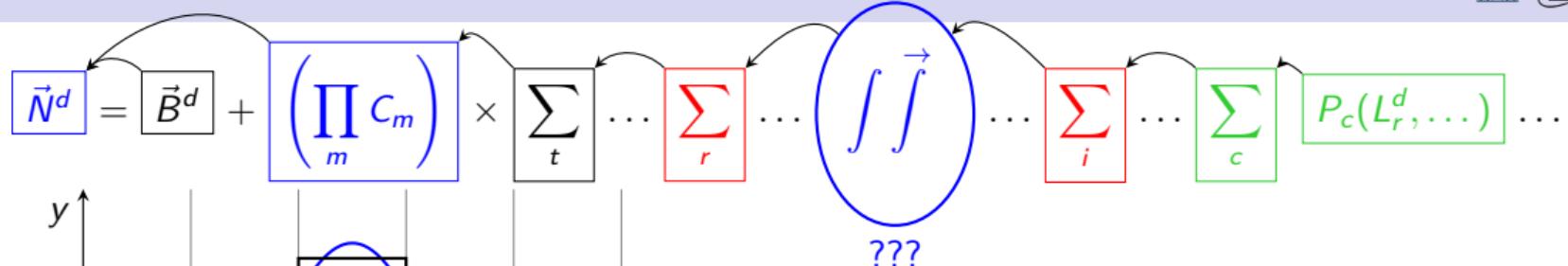


VECTORIZED INTEGRATION



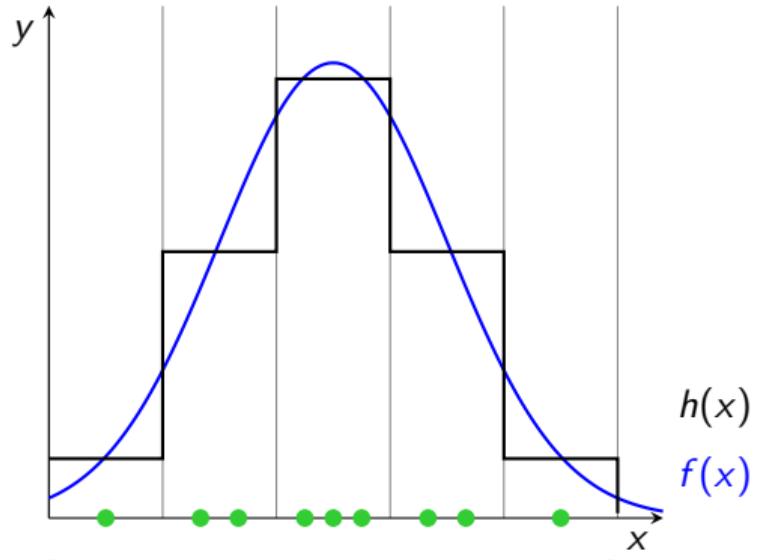
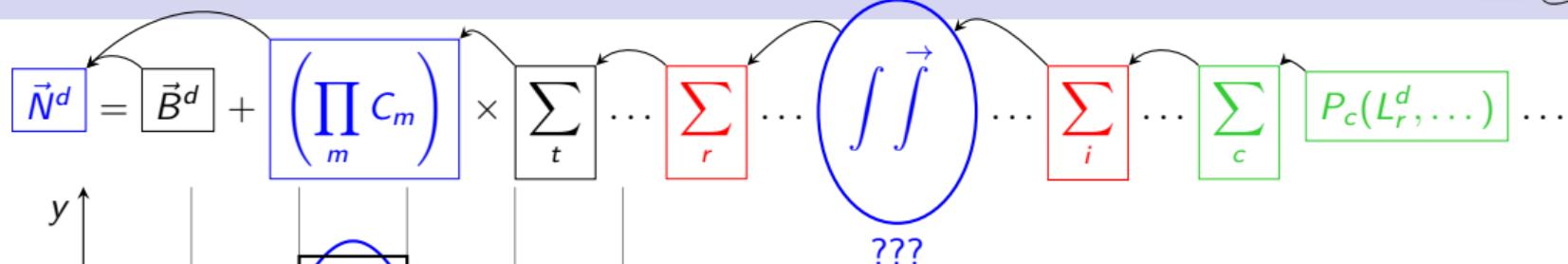


VECTORIZED INTEGRATION





VECTORIZED INTEGRATION

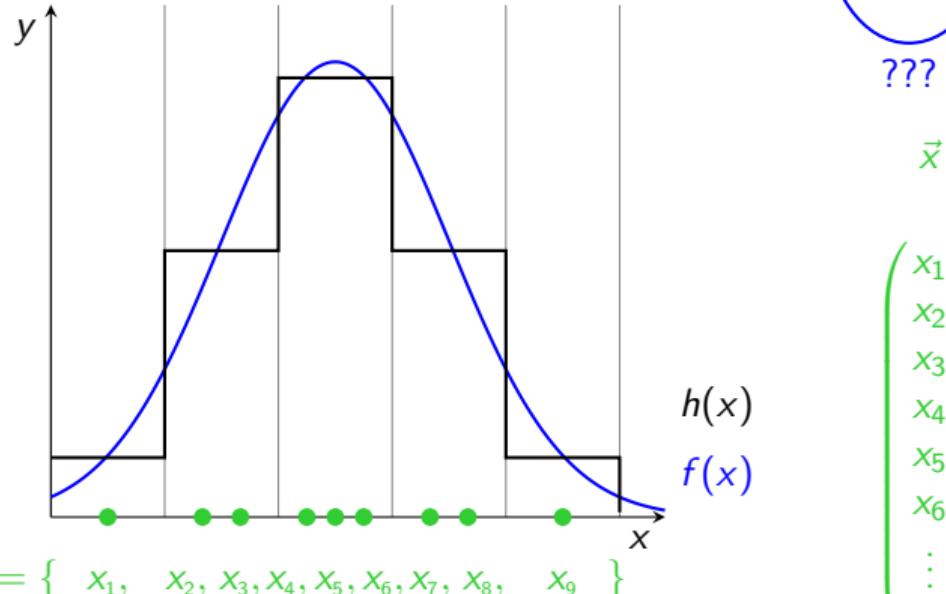


$$\vec{x} = \{ x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9 \}$$



VECTORIZED INTEGRATION

$$\vec{N}^d = \vec{B}^d + \left(\prod_m C_m \right) \times \sum_t \dots \sum_r \dots \int \int \dots \sum_i \dots \sum_c \dots P_c(L_r, \dots) \dots$$

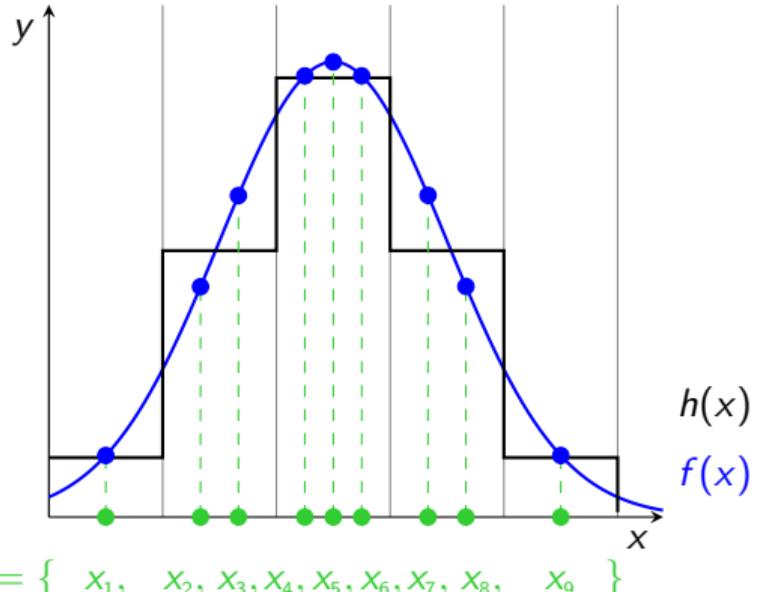


$$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ \vdots \\ x_N \end{pmatrix}$$



VECTORIZED INTEGRATION

$$\vec{N}^d = \vec{B}^d + \left(\prod_m C_m \right) \times \sum_t \dots \sum_r \dots \int \int \dots \sum_i \dots \sum_c \dots P_c(L_r, \dots) \dots$$



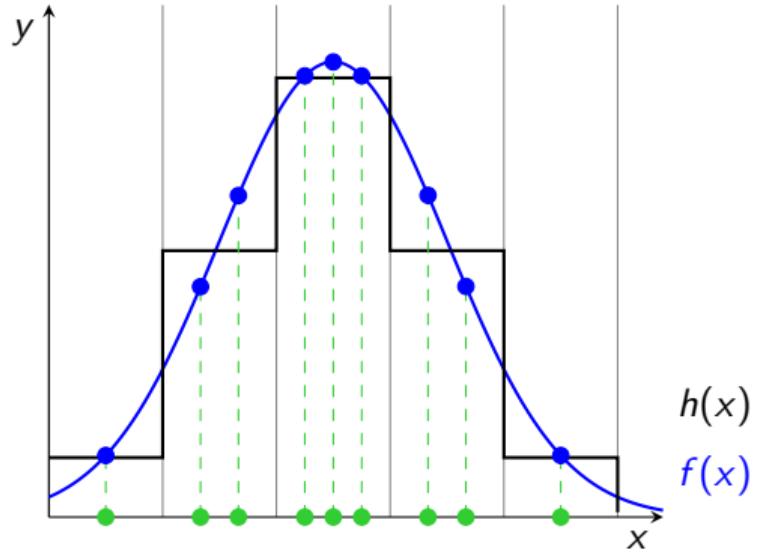
$$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ \vdots \\ x_N \end{pmatrix} \quad \begin{pmatrix} f(x_1) \\ f(x_2) \\ f(x_3) \\ f(x_4) \\ f(x_5) \\ f(x_6) \\ \vdots \\ f(x_N) \end{pmatrix}$$

$$\vec{x} \longrightarrow f(\vec{x})$$



VECTORIZED INTEGRATION

$$\vec{N}^d = \vec{B}^d + \left(\prod_m C_m \right) \times \sum_t \dots \sum_r \dots \int \int \dots \sum_i \dots \sum_c \dots P_c(L_r, \dots) \dots$$



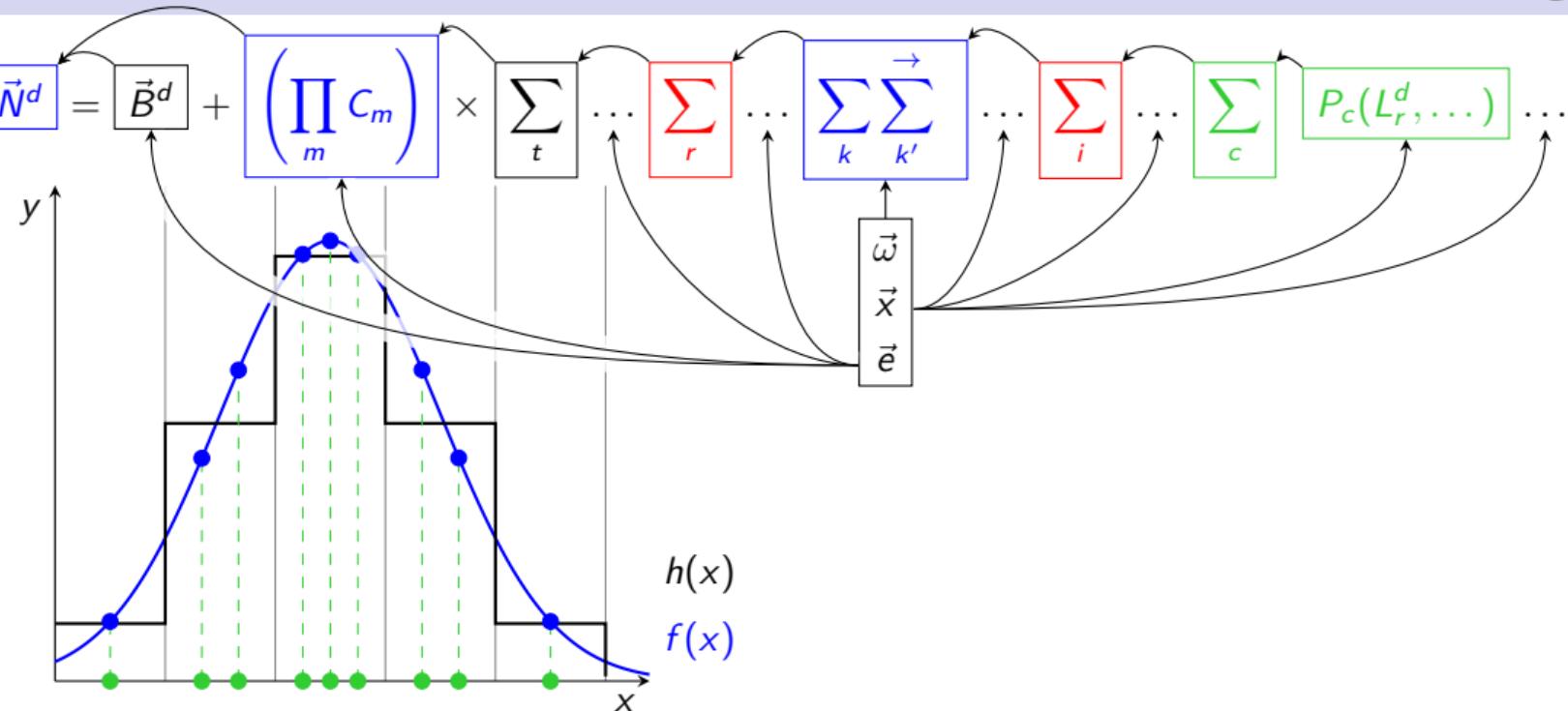
$$\vec{x} = \{ x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9 \}$$

$$\vec{\omega} = \{ \omega_1, \omega_2, \omega_3, \omega_4, \omega_5, \omega_6, \omega_7, \omega_8, \omega_9 \}$$

$$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ \vdots \\ x_N \end{pmatrix} \xrightarrow{f(x)} \begin{pmatrix} f(x_1) \\ f(x_2) \\ f(x_3) \\ f(x_4) \\ f(x_5) \\ f(x_6) \\ \vdots \\ f(x_N) \end{pmatrix} \xrightarrow{\vec{\omega}} \begin{pmatrix} h_1 \\ h_2 \\ h_3 \\ \vdots \\ h_M \end{pmatrix}$$



VECTORIZED INTEGRATION

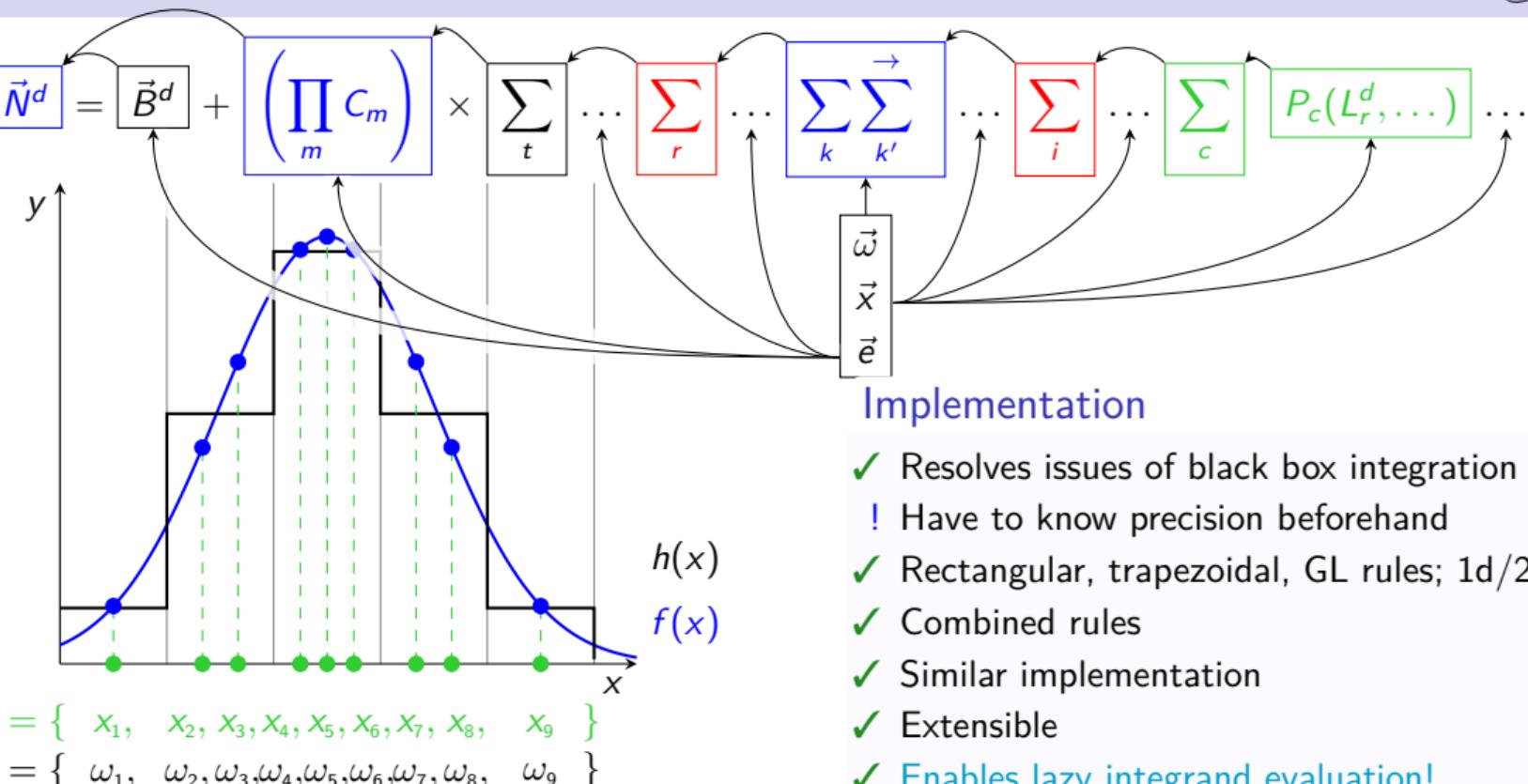


$$\vec{x} = \{x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9\}$$

$$\vec{\omega} = \{\omega_1, \omega_2, \omega_3, \omega_4, \omega_5, \omega_6, \omega_7, \omega_8, \omega_9\}$$



VECTORIZED INTEGRATION



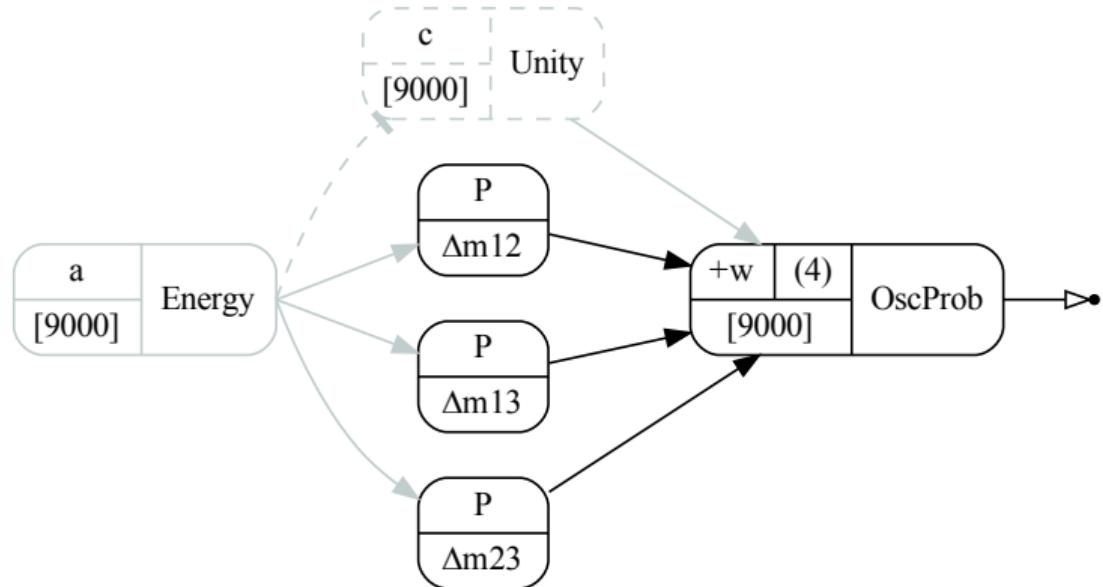


NEUTRINO OSCILLATION PROBABILITY

$$P_{\text{sur}} = P_0 + \sum_i \omega_i(\theta_{12}, \theta_{13}, \theta_{23}) \cos \left(C \frac{L \Delta m_i^2}{E} \right)$$



NEUTRINO OSCILLATION PROBABILITY



$$P_{\text{sur}} = P_0 + \sum_i \omega_i(\theta_{12}, \theta_{13}, \theta_{23}) \cos \left(C \frac{L \Delta m_i^2}{E} \right)$$

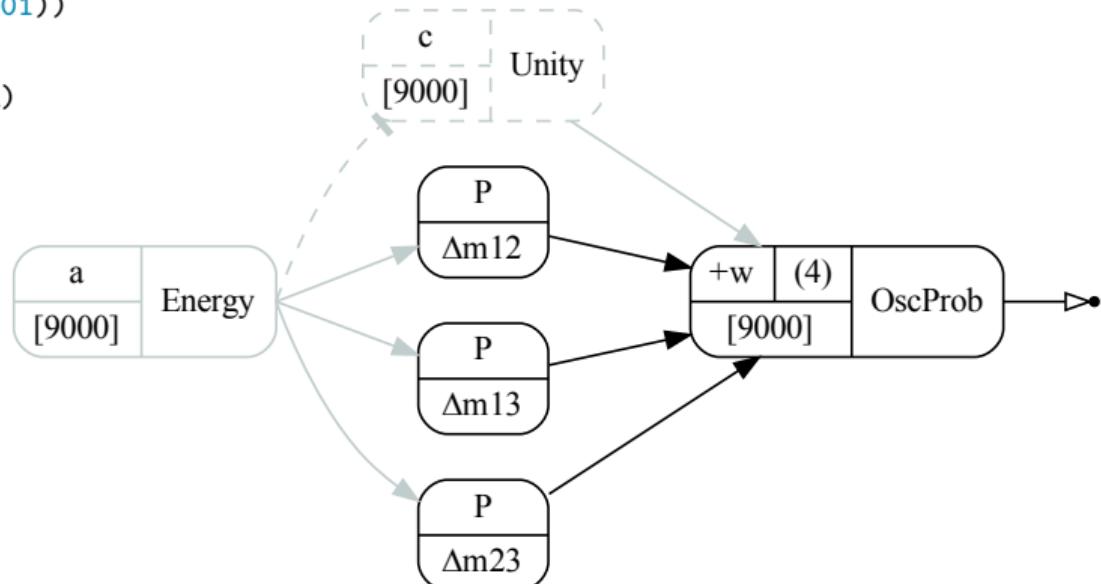
- Data flow: arrays.
- Nodes evaluated lazily.



NEUTRINO OSCILLATION PROBABILITY

```
E = C.Points(np.arange(1.0, 10.0, 0.001))
with ns:
    oscprob = C.OscProb3(from_nu, to_nu)
    unity = C.FillLike(1)
    ws = C.WeightedSum(weights, labels)

E >> unity.fill
E >> oscprob.comp12
E >> oscprob.comp13
E >> oscprob.comp23
unity      >> ws.sum.comp0
oscprob.comp12 >> ws.sum.item12
oscprob.comp13 >> ws.sum.item13
oscprob.comp23 >> ws.sum.item23
```



$$P_{\text{sur}} = P_0 + \sum_i \omega_i(\theta_{12}, \theta_{13}, \theta_{23}) \cos \left(C \frac{L \Delta m_i^2}{E} \right)$$

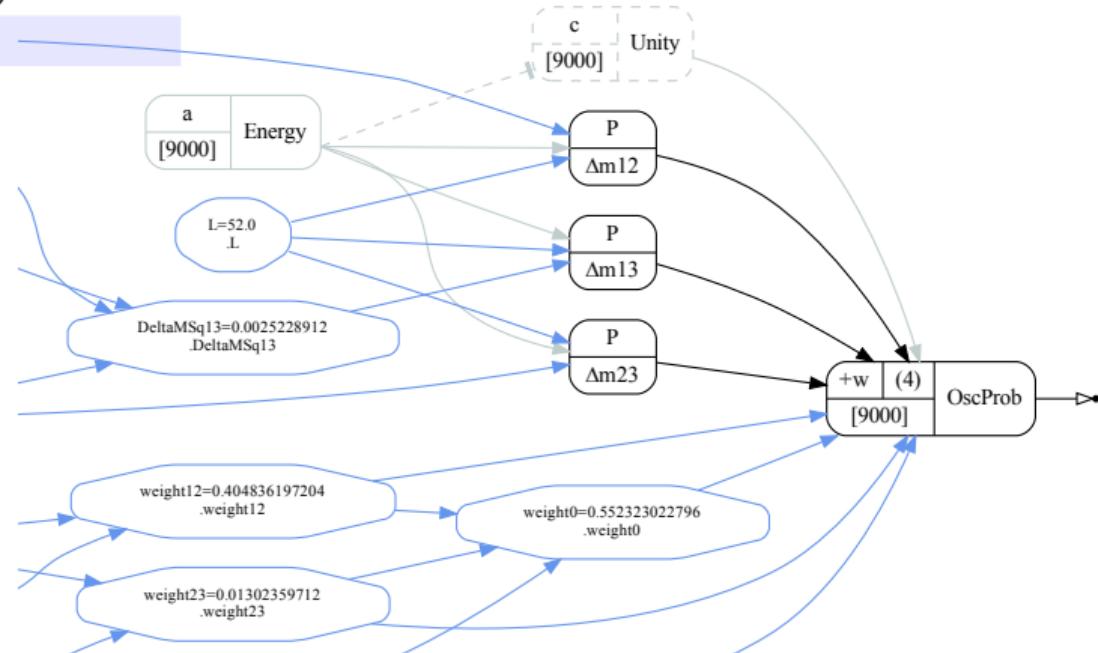
- Data flow: arrays.
- Nodes evaluated lazily.



NEUTRINO OSCILLATION PROBABILITY

```
E = C.Points(np.arange(1.0, 10.0, 0.001))
with ns:
    oscprob = C.OscProb3(from_nu, to_nu)
    unity = C.FillLike(1)
    ws = C.WeightedSum(weights, labels)

E >> unity.fill
E >> oscprob.comp12
E >> oscprob.comp13
E >> oscprob.comp23
unity      >> ws.sum.comp0
oscprob.comp12 >> ws.sum.item12
oscprob.comp13 >> ws.sum.item13
oscprob.comp23 >> ws.sum.item23
```



- Variables are maintained in recursive namespace.
- Nodes do not own variables they depend upon.



EXPRESSIONS AND BRANCHING

Example

$$g(f[j](x[i]()))$$

Goal

- Math-like expression → graph, **branching**.
- Python for evaluation.
- Only graph: no information on constituents.

 x_1 x_2

Example

- $i = 1, 2$
- $j = a, b$



EXPRESSIONS AND BRANCHING

Example

$$g | f[j] | x[i]()$$

Goal

- Math-like expression → graph, **branching**.
- Python for evaluation.
- Only graph: no information on constituents.

 x_1 x_2

Example

- $i = 1, 2$
- $j = a, b$



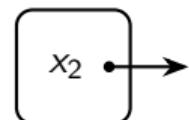
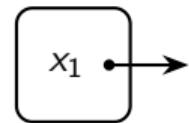
EXPRESSIONS AND BRANCHING

Example

$g | f[j] | x[i]()$

Goal

- Math-like expression → graph, **branching**.
- Python for evaluation.
- Only graph: no information on constituents.



Example

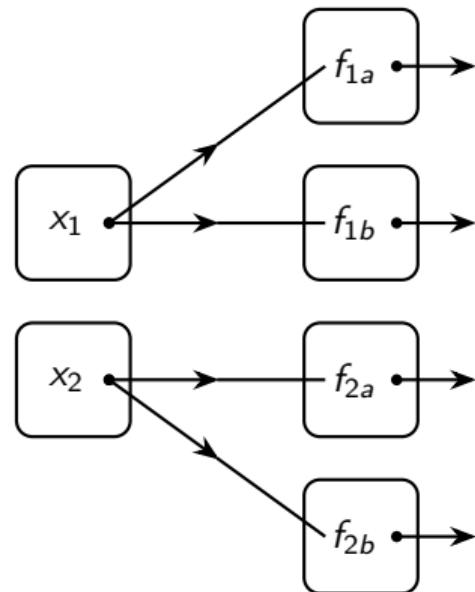
- $i = 1, 2$
- $j = a, b$



EXPRESSIONS AND BRANCHING

Example

$g | f[j] | x[i]()$



Goal

- Math-like expression → graph, **branching**.
- Python for evaluation.
- Only graph: no information on constituents.

Example

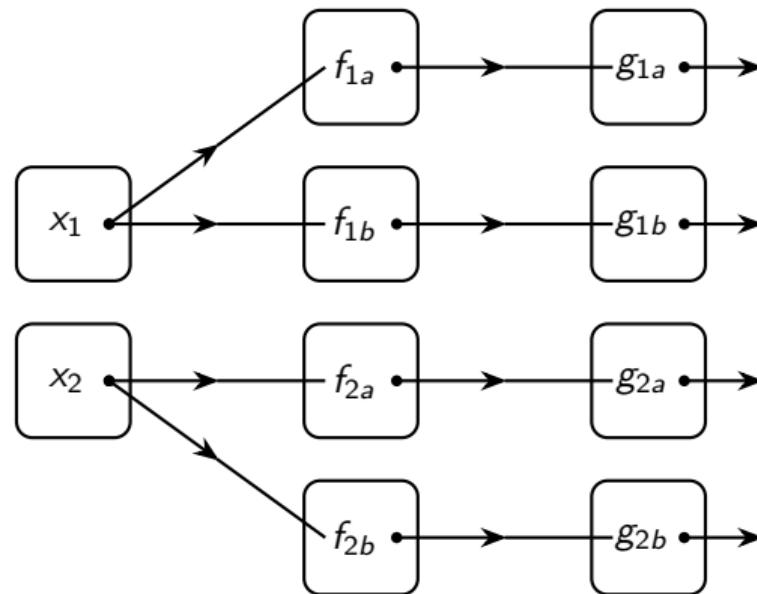
- $i = 1, 2$
- $j = a, b$



EXPRESSIONS AND BRANCHING

Example

$g | f[j] | x[i]()$



Goal

- Math-like expression → graph, **branching**.
- Python for evaluation.
- Only graph: no information on constituents.

Example

- $i = 1, 2$
- $j = a, b$



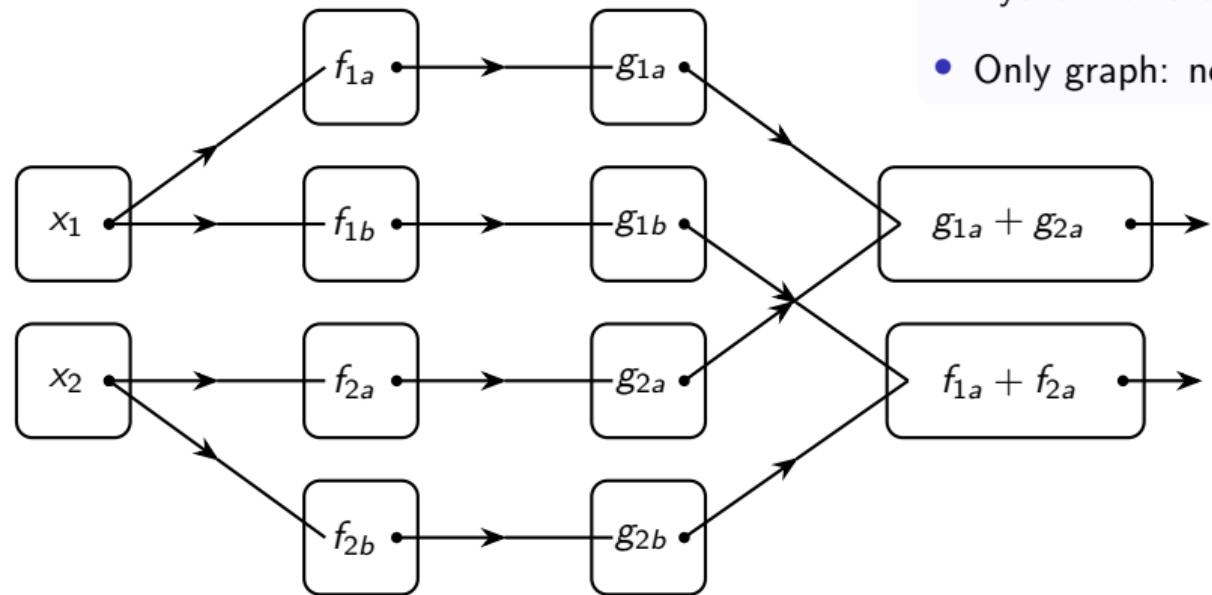
EXPRESSIONS AND BRANCHING

Example

sum[i] | g| f[j]| x[i]()

Goal

- Math-like expression → graph, **branching**.
- Python for evaluation.
- Only graph: no information on constituents.





BUNDLES

Bundles

- Build small computational graphs based on simple configuration.
- Define how the graphs are scaled.



BUNDLES

Bundles

- Build small computational graphs based on simple configuration.
- Define how the graphs are scaled.

Configuration

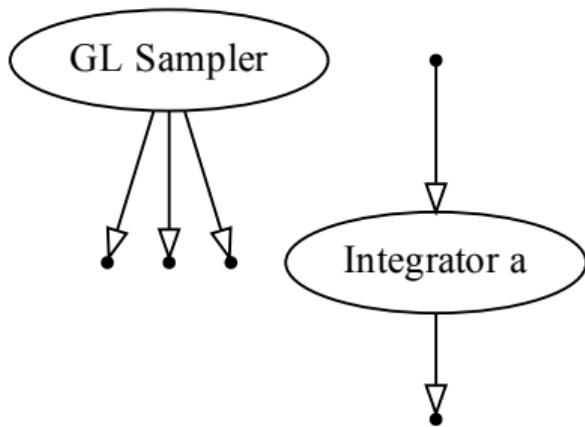
```
cfg=dict(  
    bundle = dict(name='integral_1d', version='v02'),  
    variable = 'evis',  
    edges = N.linspace(0.0, 12.0, 241, dtype='d'),  
    orders = 3,  
    labels = dict(  
        sampler = 'GL Sampler',  
        integrator = 'Integrator {autoindex}'  
    )  
)
```



BUNDLES

Bundles

- Build small computational graphs based on simple configuration.
- Define how the graphs are scaled.



Configuration

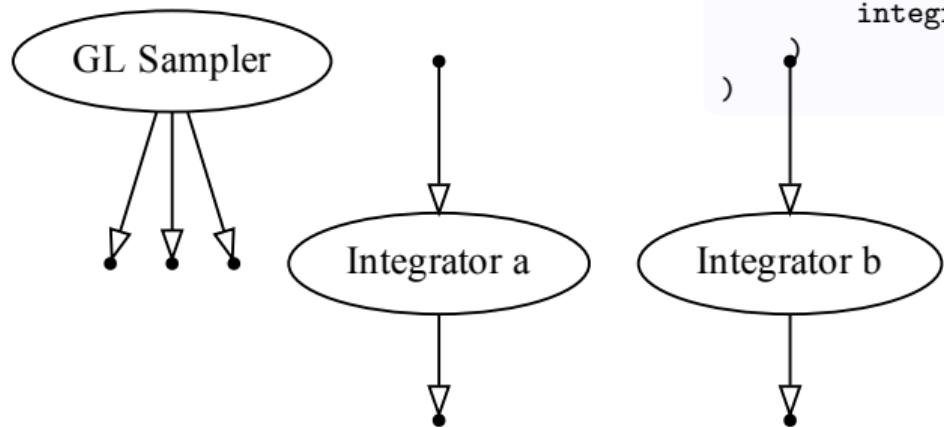
```
cfg=dict(  
    bundle = dict(name='integral_1d', version='v02'),  
    variable = 'evis',  
    edges = N.linspace(0.0, 12.0, 241, dtype='d'),  
    orders = 3,  
    labels = dict(  
        sampler = 'GL Sampler',  
        integrator = 'Integrator {autoindex}'  
    )  
)
```



BUNDLES

Bundles

- Build small computational graphs based on simple configuration.
- Define how the graphs are scaled.



Configuration

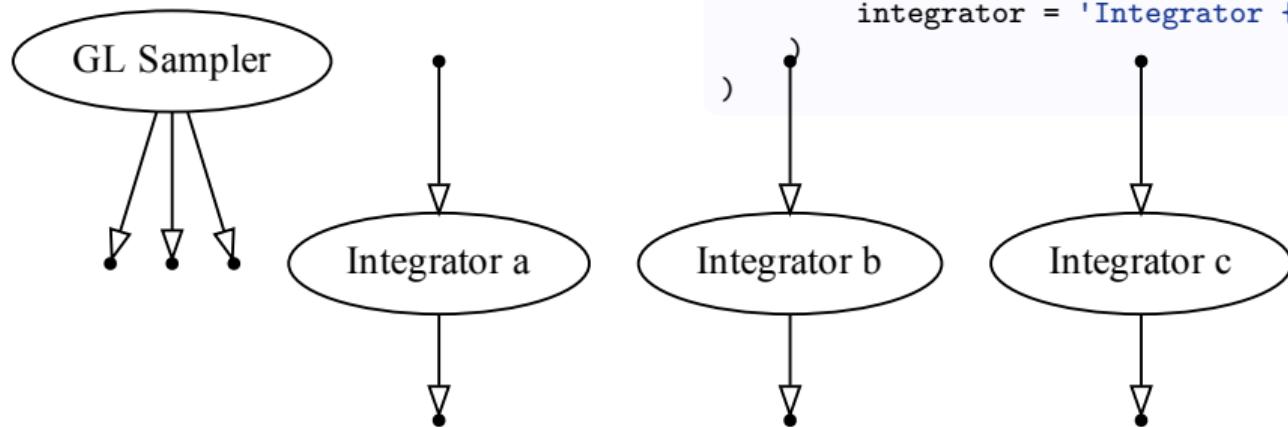
```
cfg=dict(  
    bundle = dict(name='integral_1d', version='v02'),  
    variable = 'evis',  
    edges = N.linspace(0.0, 12.0, 241, dtype='d'),  
    orders = 3,  
    labels = dict(  
        sampler = 'GL Sampler',  
        integrator = 'Integrator {autoindex}'
```



BUNDLES

Bundles

- Build small computational graphs based on simple configuration.
- Define how the graphs are scaled.



Configuration

```
cfg=dict(  
    bundle = dict(name='integral_1d', version='v02'),  
    variable = 'evis',  
    edges = N.linspace(0.0, 12.0, 241, dtype='d'),  
    orders = 3,  
    labels = dict(  
        sampler = 'GL Sampler',  
        integrator = 'Integrator {autoindex}'  
)
```



EXPERIMENT WITH REACTOR ANTINEUTRINOS: FORMULA

Partial expression

```

eres[d] |
lsnl[d] |
iav[d] |
integral2d|
sum[r] |
baselineweight[r,d]*  

ibd_xsec(enu(), ctheta())*  

jacobian(enu(), ee(), ctheta())
sum[i] (
power_livetime_factor[d,r,i])*  

anuspec[i](enu() )*
sum[c] |
pmns[c]*oscprob[c,d,r](enu())

```

Partial equation

$$\vec{N}_d = C_{eres} \times C_{lsnl} \times C_{IAV} \times \int d \cos \theta \int dE_{vis} \sum_r \frac{1}{(4\pi L_{dr}^2)} \frac{d\sigma(E_\nu, \cos \theta)}{d \cos \theta} \frac{dE_\nu}{dE_{vis}} \sum_i (P_{dri} S_i(E_\nu)) \sum_c \omega_c P_c(E_\nu, L_{dr})$$

- Each element may be configured independently of others.



EXPERIMENT WITH REACTOR ANTINEUTRINOS: FORMULA

Partial expression

```

eres[d] |
lsnl[d] |
iav[d] |
integral2d|
sum[r]|
baselineweight[r,d]*
ibd_xsec(enu(), ctheta())*
jacobian(enu(), ee(), ctheta())*
sum[i](
power_livetime_factor[d,r,i])*  

anuspec[i](enu())*
sum[c]|
pmns[c]*oscprob[c,d,r](enu())

```

← Energy resolution
 ← Energy scale bias
 ← IAV effect
 ←
 ←
 ← IBD cross section
 ← Jacobian
 ← $\bar{\nu}_e$ spectra
 ← Survival probability

Partial equation

$$\vec{N}_d = C_{eres} \times C_{lsnl} \times C_{IAV} \times \int d\cos\theta \int dE_{vis} \sum_r \frac{1}{(4\pi L_{dr}^2)} \frac{d\sigma(E_\nu, \cos\theta)}{d\cos\theta} \frac{dE_\nu}{dE_{vis}} \sum_i (P_{dri} S_i(E_\nu)) \sum_c \omega_c P_c(E_\nu, L_{dr})$$

- Each element may be configured independently of others.



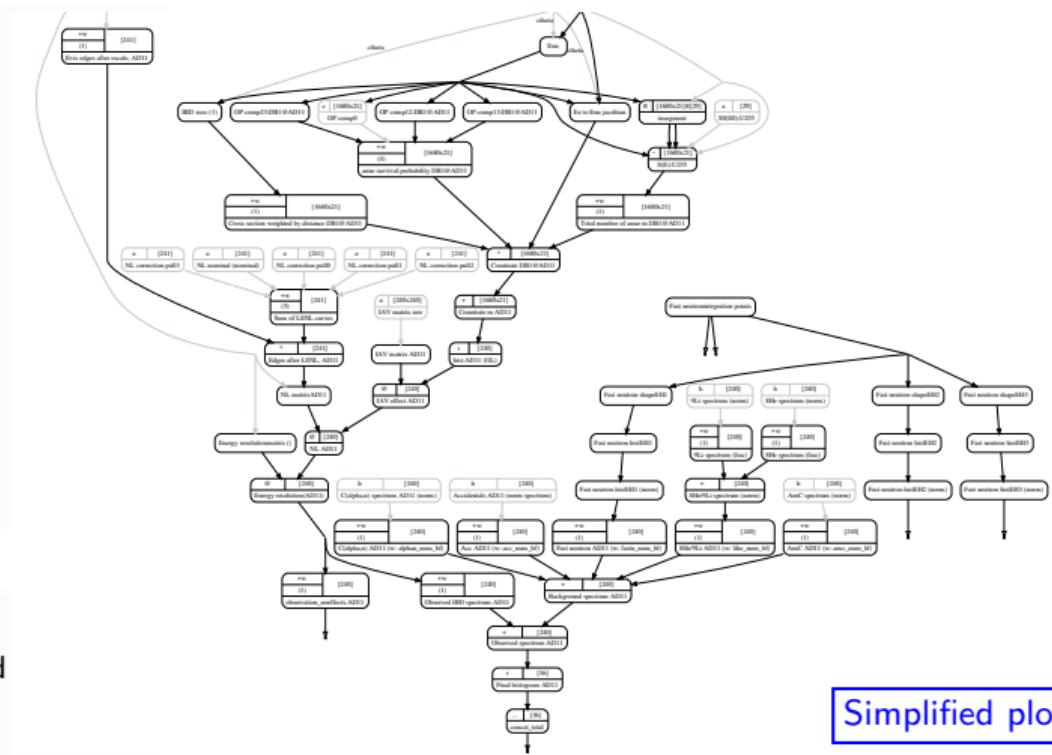
DYB OSCAR EQUATION

Expression (partial)

```

eres[d] |
lsnl[d] |
iav[d] |
integral2d|
sum[r] |
baselineweight[r,d]*
ibd_xsec(enu(), ctheta())*
jacobian(enu(), ee(), ctheta())
sum[i](
power_livetime_factor[d,r,i])* 
anuspec[i](enu() )*
sum[c] |
pmns[c]*
oscprob[c,d,r](enu())

```



Summary

- 732 nodes/1624 edges
- 498 pars: 97 variables/114 fixed/287 evaluated
- Full computation time 10 ms
- Allocates 13.6 Mb in 736 outputs

Simplified plot



GNA EXPRESSIONS STATUS

Current implementation

- Uses `python eval`
- Implements OO structure with overridden
+/-/... operators
- Modified global environment



GNA EXPRESSIONS STATUS

Current implementation

- Uses `python eval`
- Implements OO structure with overridden $+/-\dots$ operators
- Modified global environment

Future version

- Properly implemented grammar in `Lark` parser
- https://git.jinr.ru/gna/gna_parser
- To be introduced soon instead of pythonic one
- To be the default workflow
- Regardless of the core used



CONCLUSIONS

Present

- A workflow to define large scale models.
- The GNA framework implementing the workflow.
- A set of the reactor neutrino related analyses (Daya Bay, JUNO/TAO).



CONCLUSIONS

Present

- A workflow to define large scale models.
- The GNA framework implementing the workflow.
- A set of the reactor neutrino related analyses (Daya Bay, JUNO/TAO).

Future

- Discontinue development of GNA core.
- Where to go next?



CONCLUSIONS

Present

- A workflow to define large scale models.
- The GNA framework implementing the workflow.
- A set of the reactor neutrino related analyses (Daya Bay, JUNO/TAO).

Future

- Discontinue development of GNA core.
- Where to go next? Lots of possibilities to study: Python: TensorFlow and zfit, JAX, PyTorch; Julia.



CONCLUSIONS

Present

- A workflow to define large scale models.
- The GNA framework implementing the workflow.
- A set of the reactor neutrino related analyses (Daya Bay, JUNO/TAO).

Future

- Discontinue development of GNA core.
- Where to go next? Lots of possibilities to study: Python: TensorFlow and zfit, JAX, PyTorch; Julia.
- Need GPU and automatic differentiation.



CONCLUSIONS

Present

- A workflow to define large scale models.
- The GNA framework implementing the workflow.
- A set of the reactor neutrino related analyses (Daya Bay, JUNO/TAO).

Future

- Discontinue development of GNA core.
- Where to go next? Lots of possibilities to study: Python: TensorFlow and zfit, JAX, PyTorch; Julia.
- Need GPU and automatic differentiation.

Data flow

- Overall impression: life changing.
- Requires a paradigm shift, more thinking and planning.

Thank you for your attention!

Spare slides:

5 GNA STRUCTURE

6 OPTIMIZATION

7 LIKELIHOOD

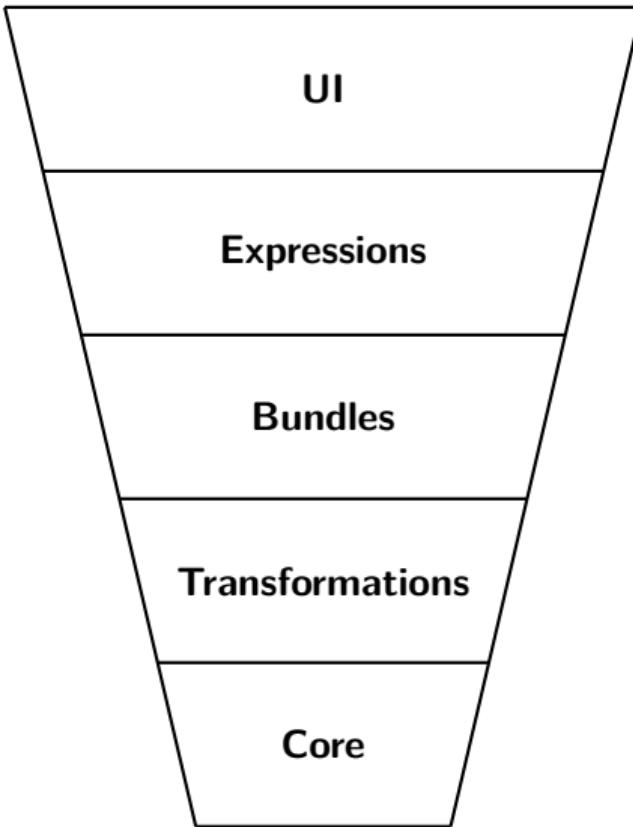
- χ^2 with pull terms
- χ^2 with pull terms

8 PRACTICAL EXAMPLE: ENERGY SCALE DISTORTION

- Matrix method
- Implementation



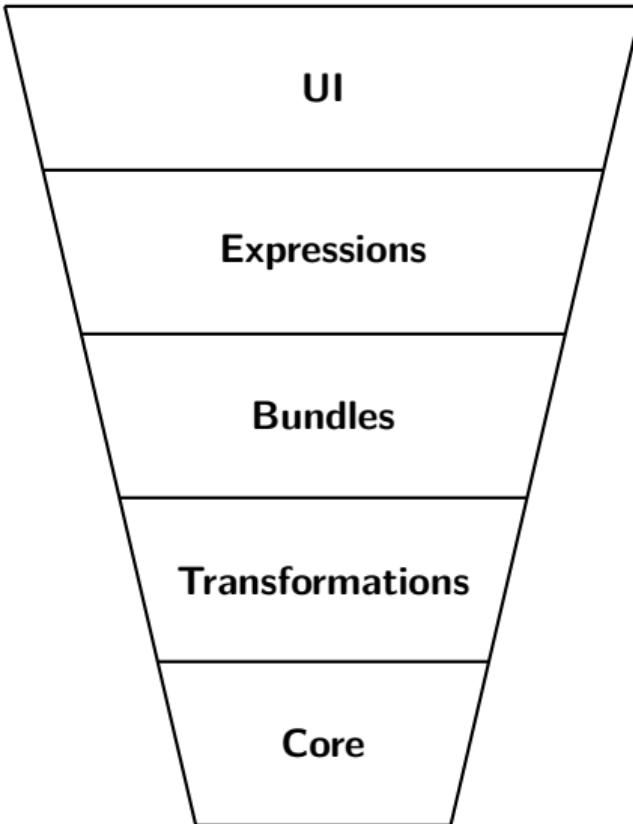
GNA APPROACH OVERVIEW



- Variables and data
- Transformations
- Scheduling



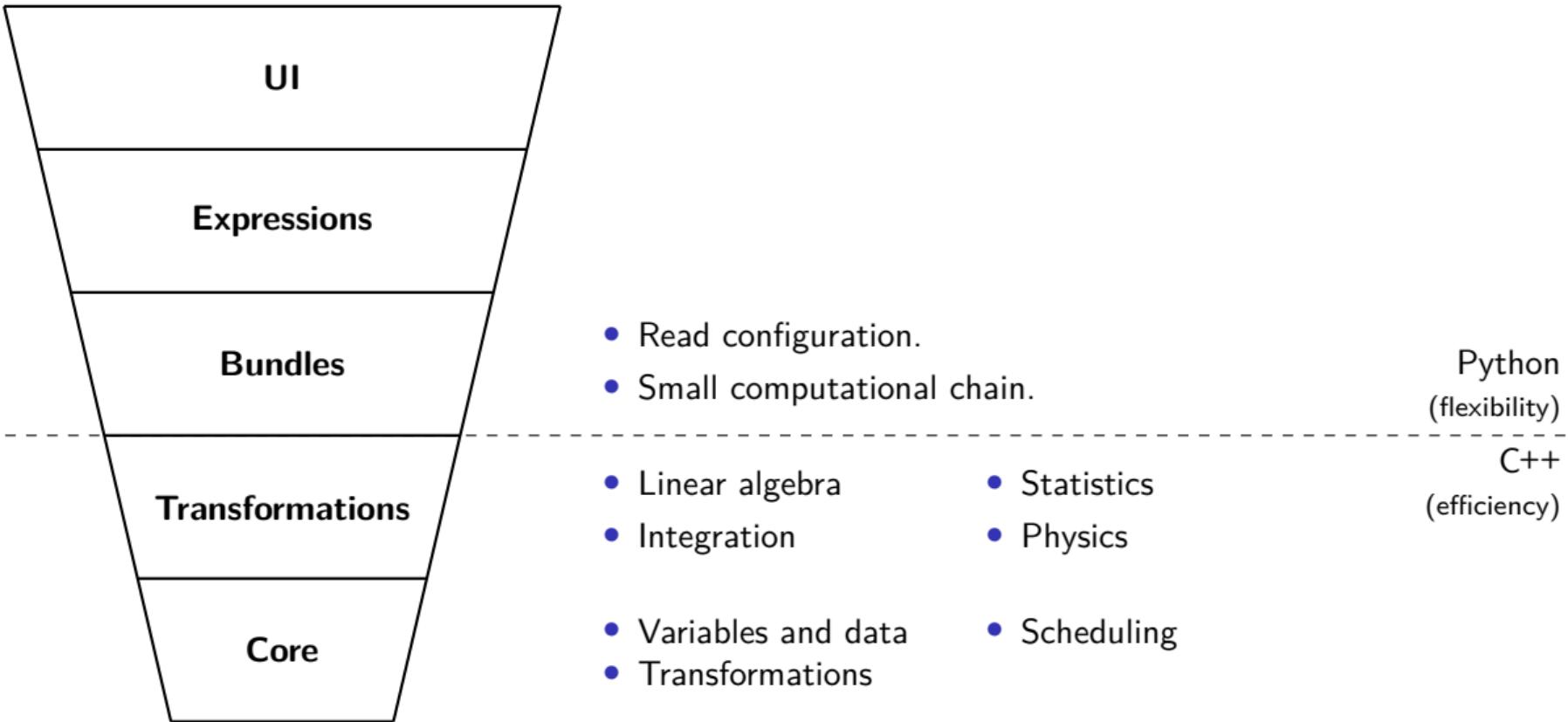
GNA APPROACH OVERVIEW



- Linear algebra
- Integration
- Statistics
- Physics
- Variables and data
- Transformations
- Scheduling

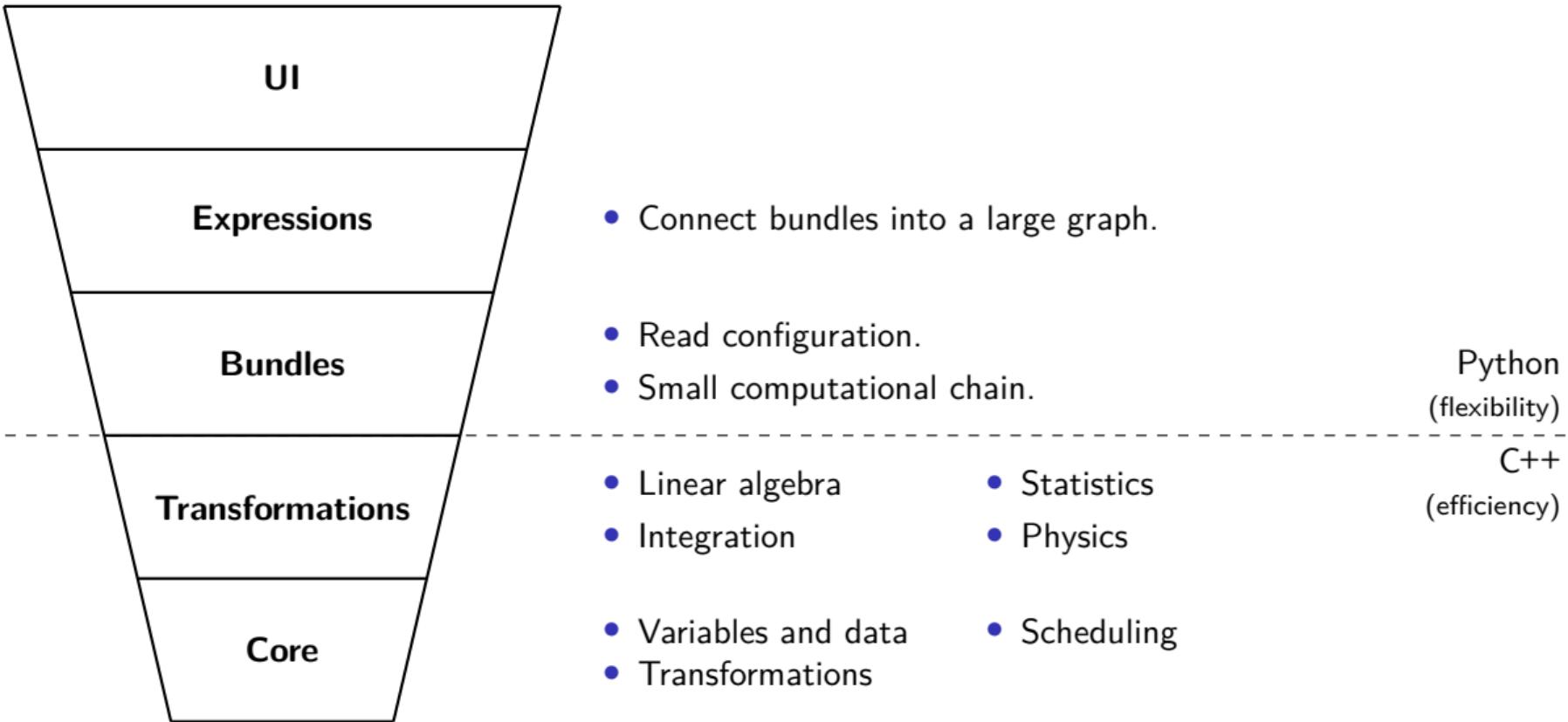


GNA APPROACH OVERVIEW



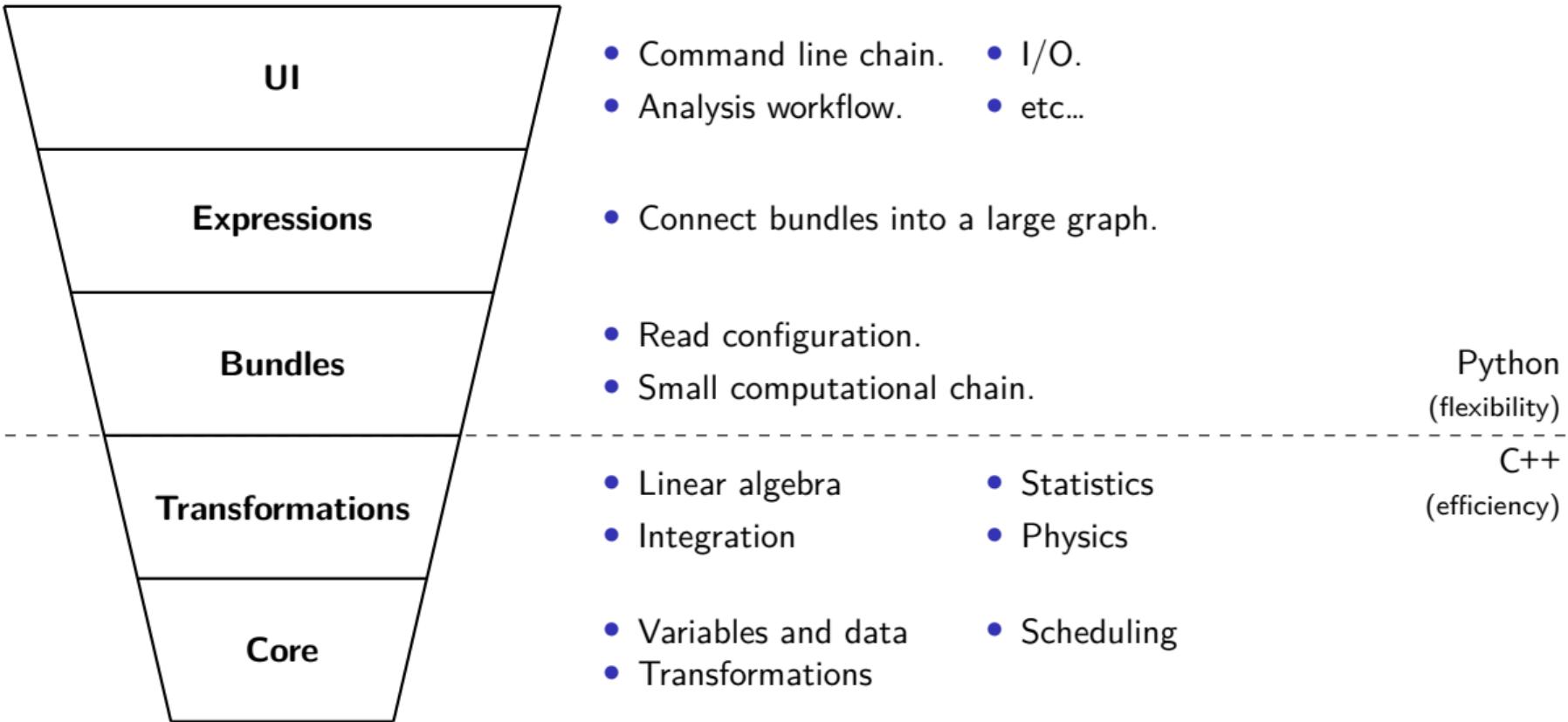


GNA APPROACH OVERVIEW



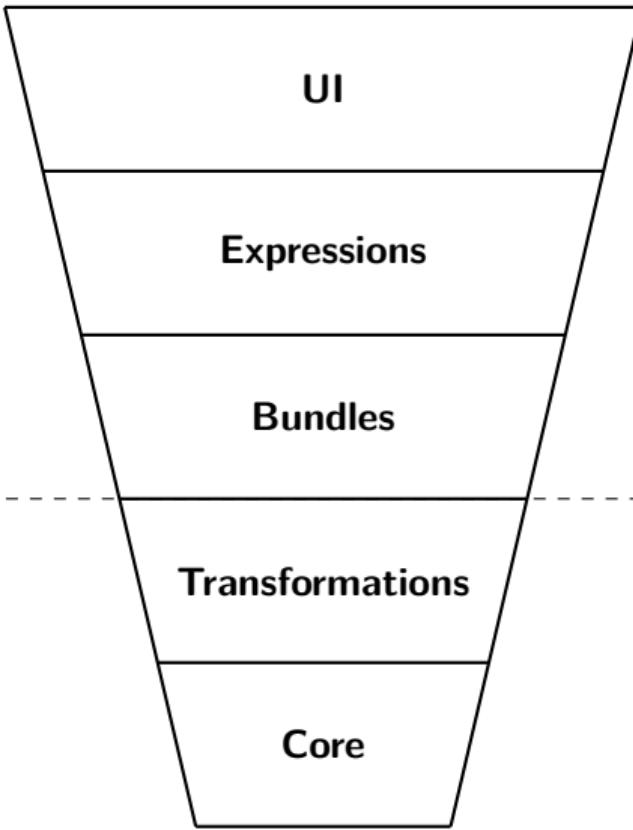


GNA APPROACH OVERVIEW





GNA APPROACH OVERVIEW



at least 3 layers of abstraction

Python
(flexibility)

C++
(efficiency)



OPTIMIZATION

Gradient descent

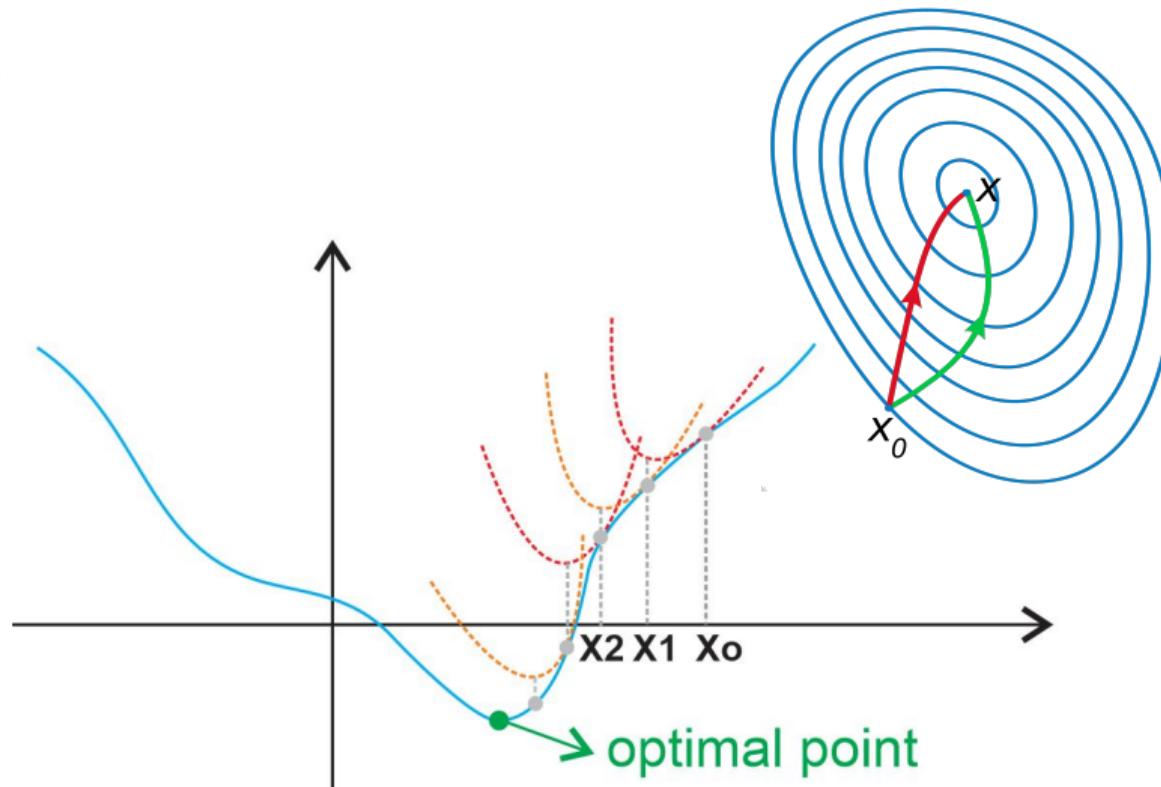
- 1st derivative Jacobian
- ✓ Simple
- ✗ Derivative \leftrightarrow step size
is arbitrary: learning rate

Quasi-Newton's method

- 2nd derivative Hessian
approximated from gradients
- ✓ Confined
- BFGS (scipy), DFG (minuit), ...

Differentiation

- Finite diff.: 2 or 4 points
- Automatic differentiation





χ^2 WITH PULL TERMS

- ✓ Estimates maximum of Gaussian likelihood

$$\chi_{\text{pull}}^2 = \sum_i \frac{(x_i - \mu_i(\theta, \eta))^2}{\sigma_i^2} + \sum_j \frac{(\eta_j - \eta_j^0)^2}{(\sigma_\eta^2)_j}$$



χ^2 WITH PULL TERMS

- ✓ Estimates maximum of Gaussian likelihood

- The model

$$\chi_{\text{pull}}^2 = \sum_i \frac{(x_i - \hat{\mu}_i(\theta, \eta))^2}{\sigma_i^2} + \sum_j \frac{(\eta_j - \eta_j^0)^2}{(\sigma_\eta^2)_j}$$



χ^2 WITH PULL TERMS

- ✓ Estimates maximum of Gaussian likelihood

- Free parameters
- The model

$$\chi_{\text{pull}}^2 = \sum_i \frac{(x_i - \hat{\mu}_i(\theta, \eta))^2}{\sigma_i^2} + \sum_j \frac{(\eta_j - \eta_j^0)^2}{(\sigma_\eta^2)_j}$$



χ^2 WITH PULL TERMS

✓ Estimates maximum of Gaussian likelihood

- Constrained parameters
- Free parameters
- The model

$$\chi_{\text{pull}}^2 = \sum_i \frac{(x_i - \hat{\mu}_i(\theta, \eta))^2}{\sigma_i^2} + \sum_j \frac{(\eta_j - \eta_j^0)^2}{(\sigma_\eta^2)_j}$$



χ^2 WITH PULL TERMS

✓ Estimates maximum of Gaussian likelihood

- Constrained parameters
- Free parameters
- The model
- Data

$$\chi_{\text{pull}}^2 = \sum_i \frac{(\hat{x}_i - \hat{\mu}_i(\theta, \eta))^2}{\sigma_i^2} + \sum_j \frac{(\eta_j - \eta_j^0)^2}{(\sigma_\eta^2)_j}$$



χ^2 WITH PULL TERMS

✓ Estimates maximum of Gaussian likelihood

- Constrained parameters
- Free parameters
- The model
- Data

$$\chi_{\text{pull}}^2 = \sum_i \frac{(\hat{x}_i - \hat{\mu}_i(\theta, \eta))^2}{\sigma_i^2} + \sum_j \frac{(\eta_j - \eta_j^0)^2}{(\sigma_\eta^2)_j}$$

- Stat errors



χ^2 WITH PULL TERMS

- ✓ Estimates maximum of Gaussian likelihood

- Constrained parameters
- Free parameters
- The model
- Data

$$\chi_{\text{pull}}^2 = \sum_i \frac{(\hat{x}_i - \hat{\mu}_i(\theta, \eta))^2}{\sigma_i^2} + \sum_j \frac{(\eta_j - \eta_j^0)^2}{(\sigma_\eta^2)_j}$$

- Stat errors
- Pull terms or punishment



χ^2 WITH PULL TERMS

✓ Estimates maximum of Gaussian likelihood

- Constrained parameters
- Free parameters
- The model
- Data

$$\chi_{\text{pull}}^2 = \sum_i \frac{(\hat{x}_i - \hat{\mu}_i(\theta, \eta))^2}{\sigma_i^2} + \sum_j \frac{(\eta_j - \eta_j^0)^2}{(\sigma_\eta^2)_j}$$

- Stat errors
- Pull terms or punishment
- Systematic uncertainties



χ^2 WITH PULL TERMS

- ✓ Estimates maximum of Gaussian likelihood

- Constrained parameters
- Free parameters
- The model
- Data

$$\chi^2 = (x - \mu(\theta, \eta))^T V_{\text{stat}}^{-1} (x - \mu(\theta, \eta)) +$$

$$+ (\eta - \eta^0)^T V_{\eta}^{-1} (\eta - \eta^0)$$

- Stat errors
- Pull terms or punishment
- Systematic uncertainties

math symbols: matrices and columns



χ^2 WITH PULL TERMS

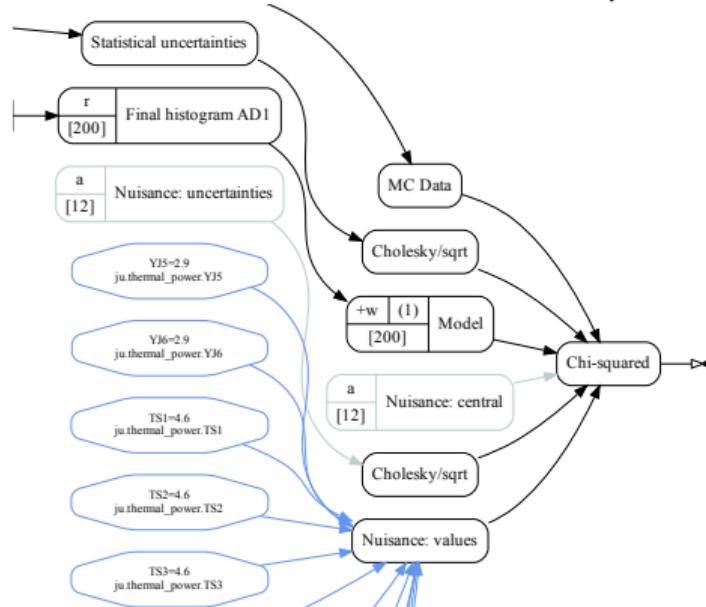
✓ Estimates maximum of Gaussian likelihood

- Constrained parameters
- Free parameters
- The model
- Data

$$\chi^2 = (x - \mu(\theta, \eta))^T V_{\text{stat}}^{-1} (x - \mu(\theta, \eta)) + \\ + (\eta - \eta^0)^T V_{\eta}^{-1} (\eta - \eta^0)$$

- Stat errors
- Pull terms or punishment
- Systematic uncertainties

Statistical methods follow data flow procedure:



math symbols: matrices and columns



χ^2 WITH COVARIANCE MATRIX

- ✓ Given: linear expansion is good: $\mu(\theta, \eta) \approx \mu(\theta, \eta_0) + D(\eta - \eta_0)$
- ✓ Using:
 - ▶ Frequentist: profile χ^2_{pull} over η
 - ▶ Bayesian: marginalize normal likelihood over η

math symbols: matrices and columns



χ^2 WITH COVARIANCE MATRIX

✓ Given: linear expansion is good: $\mu(\theta, \eta) \approx \mu(\theta, \eta_0) + D(\eta - \eta_0)$

✓ Using:

- ▶ Frequentist: profile χ^2_{full} over η
- ▶ Bayesian: marginalize normal likelihood over η

- Obtain: χ^2 with covariance matrix

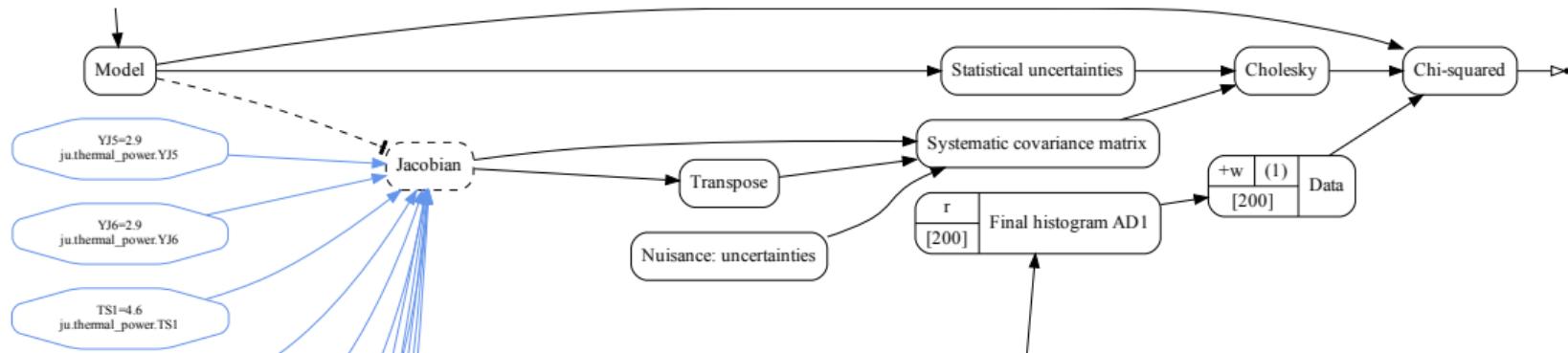
$$\chi^2 = (x - \mu(\theta, \eta_0))^T V_{\text{full}}^{-1} (x - \mu(\theta, \eta_0)),$$

- Where: $V_{\text{full}}(\theta) = V_{\text{stat}} + D_\eta V_\eta D_\eta^T.$

math symbols: matrices and columns



χ^2 WITH COVARIANCE MATRIX



✓ Given: linear expansion is good: $\mu(\theta, \eta) \approx \mu(\theta, \eta_0) + D(\eta - \eta_0)$

✓ Using:

- ▶ Frequentist: profile χ^2_{pull} over η
- ▶ Bayesian: marginalize normal likelihood over η

• Obtain: χ^2 with covariance matrix

$$\chi^2 = (x - \mu(\theta, \eta_0))^T V_{\text{full}}^{-1} (x - \mu(\theta, \eta_0)),$$

• Where: $V_{\text{full}}(\theta) = V_{\text{stat}} + D_\eta V_\eta D_\eta^T.$

math symbols: matrices and columns

PRACTICAL EXAMPLE:
ENERGY SCALE DISTORTION



NONLINEARITY APPLICATION: INTEGRATION

$$N_i^a = \int d\cos\theta \int_{E_i}^{E_{i+1}} \sigma(E^\nu, \cos\theta) S(E^\nu) P(E^\nu) \frac{dE^\nu}{dE^{\text{vis}}} dE^{\text{vis}},$$

$$E^\nu = E^\nu(E^{\text{vis}}, \cos\theta).$$

- No energy nonlinearity: N^a



NONLINEARITY APPLICATION: INTEGRATION

$$N_i^a = \int_{E_i}^{E_{i+1}} \int X(E^\nu) dE^{\text{vis}},$$

$$E^\nu = E^\nu(E^{\text{vis}}, \cos \theta).$$

- No energy nonlinearity: N^a



NONLINEARITY APPLICATION: INTEGRATION

$$N_i^a = \int_{E_i}^{E_{i+1}} \int X(E^\nu) dE^{\text{vis}}, \quad N_i^b = \int_{E^q(E_i)}^{E^q(E_{i+1})} \int X(E^\nu) \frac{dE^{\text{vis}}}{dE^q} dE^q,$$

$$\begin{aligned} E^\nu &= E^\nu(E^{\text{vis}}, \cos \theta). & E^\nu &= E^\nu(E^{\text{vis}}, \cos \theta), \\ E^{\text{vis}} &= E^{\text{vis}}(E^q) \text{ (inverse),} \\ E^q &= E^q(E^{\text{vis}}). \end{aligned}$$

- No energy nonlinearity: N^a
- With energy nonlinearity: $N^b \hookrightarrow N^a \equiv N^b$



NONLINEARITY APPLICATION: INTEGRATION

$$N_i^a = \int_{E_i}^{E_{i+1}} \int X(E^\nu) dE^{\text{vis}}, \quad N_i^b = \int_{E^q(E_i)}^{E^q(E_{i+1})} X(E^\nu) \frac{dE^{\text{vis}}}{dE^q} dE^q, \quad N_i^c = \int_{E_i}^{E_{i+1}} \int X(E^\nu) \frac{dE^{\text{vis}}}{dE^q} dE^q.$$

$$E^\nu = E^\nu(E^{\text{vis}}, \cos \theta), \quad E^\nu = E^\nu(E^{\text{vis}}, \cos \theta), \\ E^{\text{vis}} = E^{\text{vis}}(E^q) \text{ (inverse)}, \\ E^q = E^q(E^{\text{vis}}).$$

- No energy nonlinearity: N^a
- With energy nonlinearity: $N^b \hookrightarrow N^a \equiv N^b$
- With energy nonlinearity: N^b , projected on the same binning as N^a .



NONLINEARITY APPLICATION: INTEGRATION

$$N_i^a = \int_{E_i}^{E_{i+1}} \int X(E^\nu) dE^{\text{vis}}, \quad N_i^b = \int_{E^{\text{q}}(E_i)}^{E^{\text{q}}(E_{i+1})} \int X(E^\nu) \frac{dE^{\text{vis}}}{dE^{\text{q}}} dE^{\text{q}}, \quad N_i^c = \int_{E_i}^{E_{i+1}} \int X(E^\nu) \frac{1}{\frac{dE^{\text{q}}}{dE^{\text{vis}}}(E^{\text{vis}}(E^{\text{q}}))} dE^{\text{q}}.$$

$$E^\nu = E^\nu(E^{\text{vis}}, \cos \theta).$$

$$E^\nu = E^\nu(E^{\text{vis}}, \cos \theta),$$

- Assuming $E^{\text{vis}}(E^{\text{q}})$ is a sum.

$$E^{\text{vis}} = E^{\text{vis}}(E^{\text{q}}) \text{ (inverse),}$$

$$E^{\text{q}} = E^{\text{q}}(E^{\text{vis}}).$$

- No energy nonlinearity: N^a
- With energy nonlinearity: $N^b \hookrightarrow N^a \equiv N^b$
- With energy nonlinearity: N^b , projected on the same binning as N^a .



NONLINEARITY APPLICATION: INTEGRATION

$$N_i^a = \int_{E_i}^{E_{i+1}} \int X(E^\nu) dE^{\text{vis}}, \quad N_i^b = \int_{E^{\text{q}}(E_i)}^{E^{\text{q}}(E_{i+1})} \int X(E^\nu) \frac{dE^{\text{vis}}}{dE^{\text{q}}} dE^{\text{q}}, \quad N_i^c = \int_{E_i}^{E_{i+1}} \int X(E^\nu) \frac{1}{\frac{dE^{\text{q}}}{dE^{\text{vis}}}(E^{\text{vis}}(E^{\text{q}}))} dE^{\text{q}}.$$

$$E^\nu = E^\nu(E^{\text{vis}}, \cos \theta).$$

$$E^\nu = E^\nu(E^{\text{vis}}, \cos \theta),$$

- Assuming $E^{\text{vis}}(E^{\text{q}})$ is a sum.

$$E^{\text{vis}} = E^{\text{vis}}(E^{\text{q}}) \text{ (inverse),}$$

$$E^{\text{q}} = E^{\text{q}}(E^{\text{vis}}).$$

- No energy nonlinearity: N^a
- With energy nonlinearity: $N^b \hookrightarrow N^a \equiv N^b$
- With energy nonlinearity: N^b , projected on the same binning as N^a .
- Requirements:
 - Direct conversion.
 - Inverse conversion.
 - Gradient.



NONLINEARITY APPLICATION: INTEGRATION

$$N_i^a = \int_{E_i}^{E_{i+1}} \int X(E^\nu) dE^{\text{vis}}, \quad N_i^b = \int_{E^{\text{q}}(E_i)}^{E^{\text{q}}(E_{i+1})} \int X(E^\nu) \frac{dE^{\text{vis}}}{dE^{\text{q}}} dE^{\text{q}}, \quad N_i^c = \int_{E_i}^{E_{i+1}} \int X(E^\nu) \frac{1}{\frac{dE^{\text{q}}}{dE^{\text{vis}}}(E^{\text{vis}}(E^{\text{q}}))} dE^{\text{q}}.$$

$$E^\nu = E^\nu(E^{\text{vis}}, \cos \theta).$$

$$E^\nu = E^\nu(E^{\text{vis}}, \cos \theta),$$

- Assuming $E^{\text{vis}}(E^{\text{q}})$ is a sum.

$$E^{\text{vis}} = E^{\text{vis}}(E^{\text{q}}) \text{ (inverse),}$$

$$E^{\text{q}} = E^{\text{q}}(E^{\text{vis}}).$$

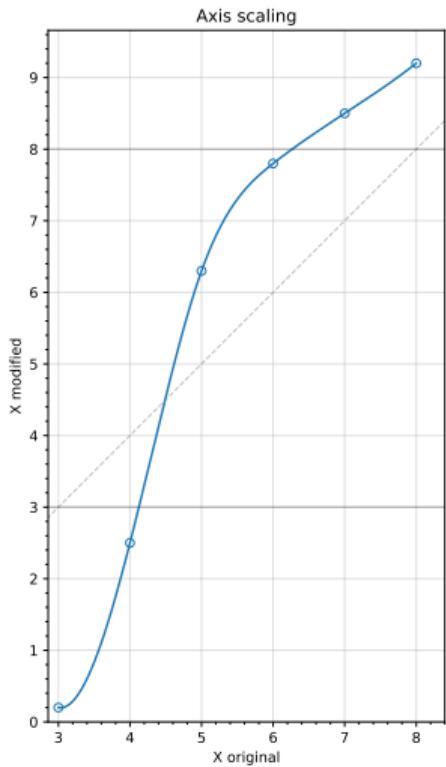
- No energy nonlinearity: N^a
- With energy nonlinearity: $N^b \hookrightarrow N^a \equiv N^b$
- With energy nonlinearity: N^b , projected on the same binning as N^a .
- Requirements:
 - Direct conversion.
 - Inverse conversion.
 - Gradient.
- Is it possible to introduce matrix C :

$$N^c = CN^a?$$



NONLINEARITY APPLICATION: MATRIX

- Is it possible to introduce matrix C : $N^c = CN^a$?

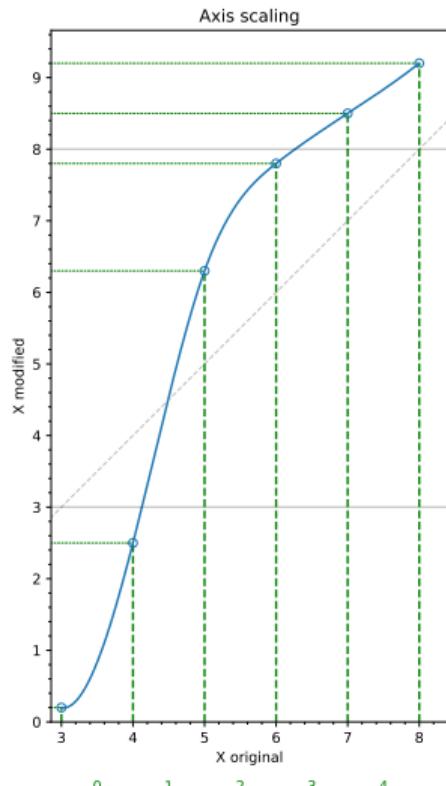




NONLINEARITY APPLICATION: MATRIX

- Is it possible to introduce matrix C : $N^c = CN^a$?

Dumb method



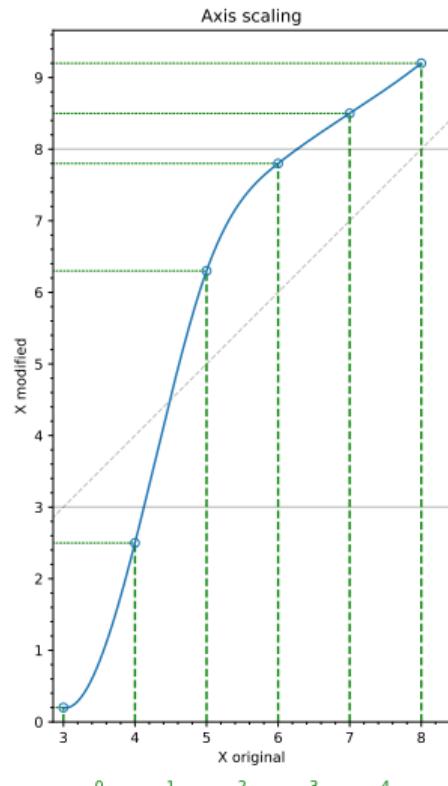
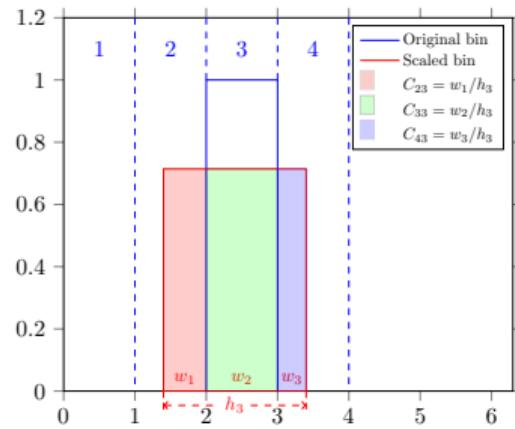


NONLINEARITY APPLICATION: MATRIX

- Is it possible to introduce matrix C : $N^c = CN^a$?

Dumb method

- See how the scaled bin edges overlap with original binning.



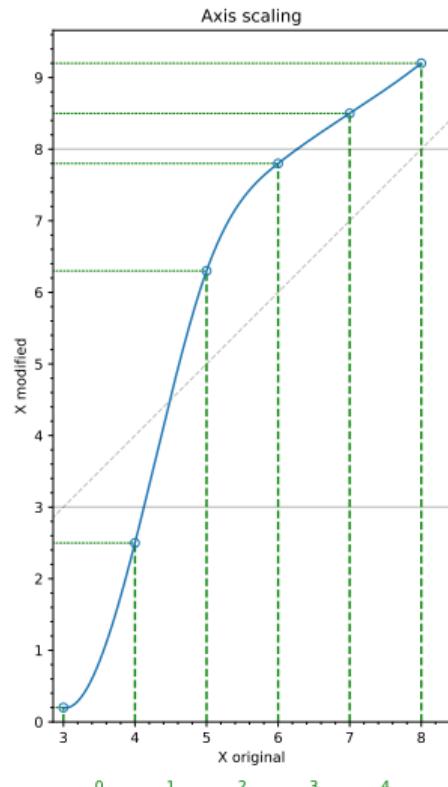
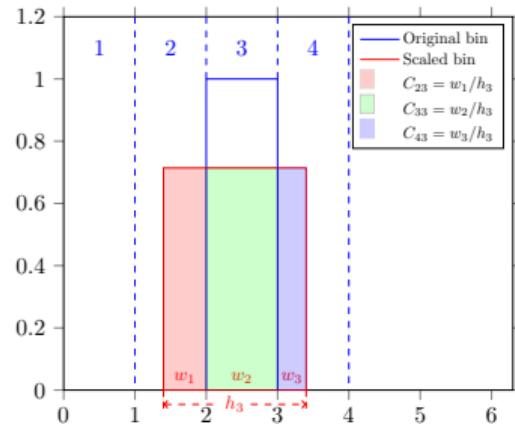


NONLINEARITY APPLICATION: MATRIX

- Is it possible to introduce matrix C : $N^c = CN^a$?

Dumb method

- See how the scaled bin edges overlap with original binning.
- Requirements:
 - Only direct energy scale conversion.



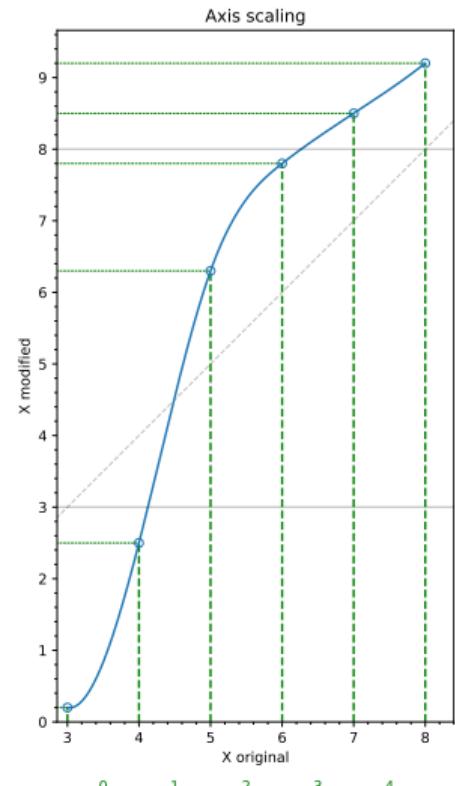
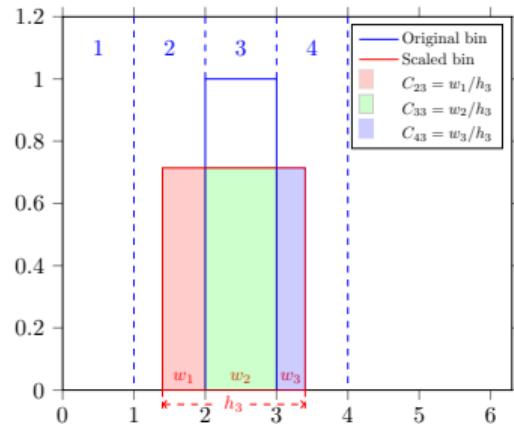


NONLINEARITY APPLICATION: MATRIX

- Is it possible to introduce matrix C : $N^c = CN^a$?

Dumb method

- See how the scaled bin edges overlap with original binning.
- Requirements:
 - Only direct energy scale conversion.



Assumptions

- Integrand within bin: flat
- Energy scale within bin: linear

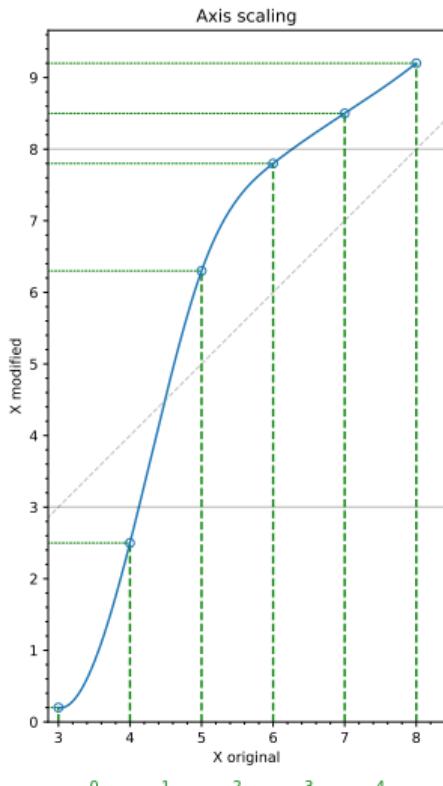


NONLINEARITY APPLICATION: MATRIX

- Is it possible to introduce matrix C : $N^c = CN^a$?

Proper method

- See how the scaled bin edges overlap with original binning.
- Take into account inverse projections: integrals over small segments equal.



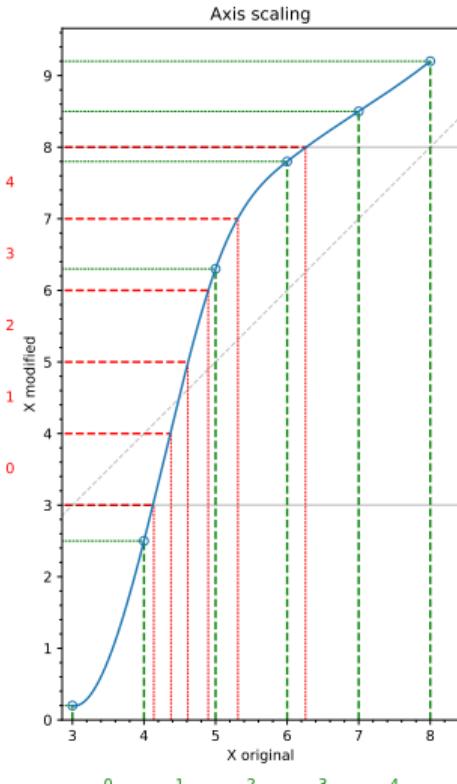


NONLINEARITY APPLICATION: MATRIX

- Is it possible to introduce matrix C : $N^c = CN^a$?

Proper method

- See how the scaled bin edges overlap with original binning.
- Take into account inverse projections: integrals over small segments equal.



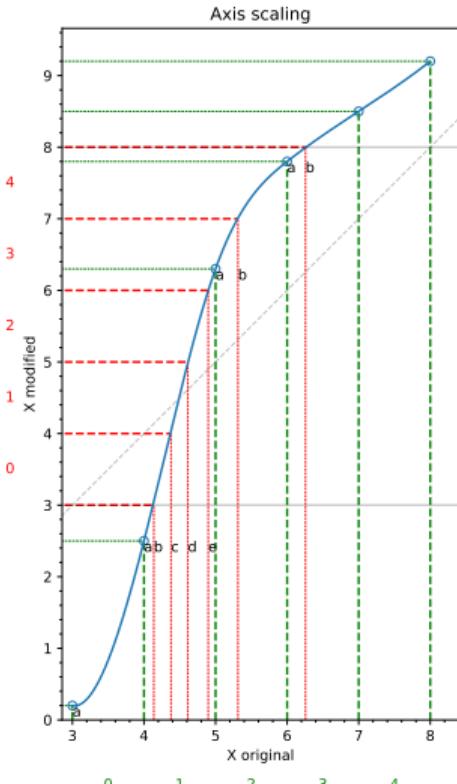


NONLINEARITY APPLICATION: MATRIX

- Is it possible to introduce matrix C : $N^c = CN^a$?

Proper method

- See how the scaled bin edges overlap with original binning.
- Take into account inverse projections: integrals over small segments equal.





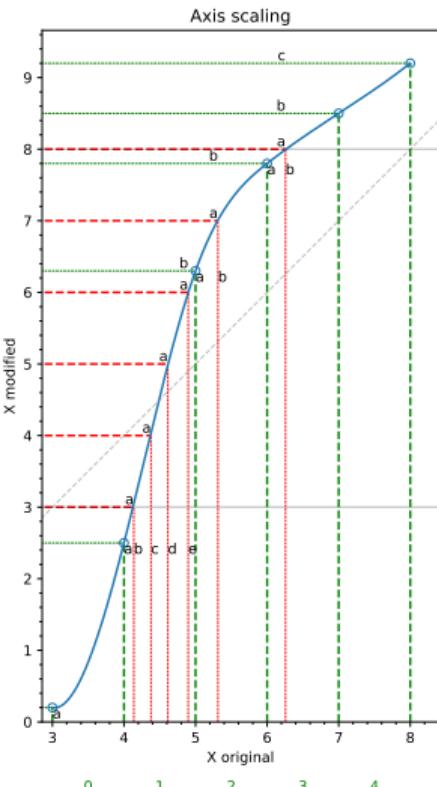
NONLINEARITY APPLICATION: MATRIX

- Is it possible to introduce matrix C : $N^c = CN^a$?

Proper method

- See how the scaled bin edges overlap with original binning.
- Take into account inverse projections: integrals over small segments equal.

$$C_{11} = \frac{1c}{1a+1b+1c+1d+1e}$$





NONLINEARITY APPLICATION: MATRIX

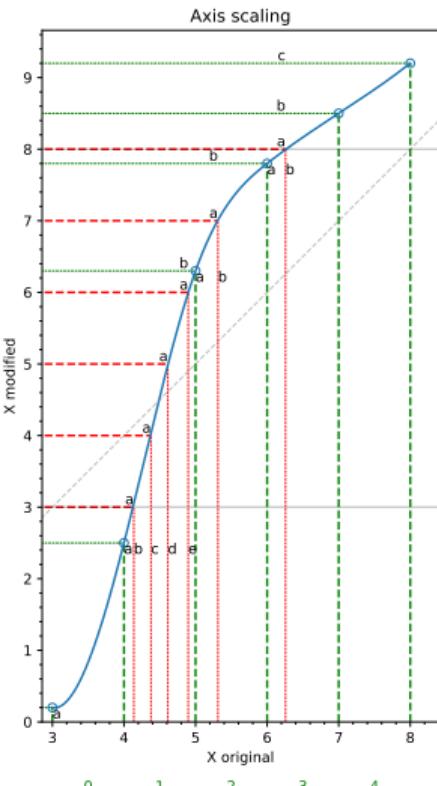
- Is it possible to introduce matrix C : $N^c = CN^a$?

Proper method

- See how the scaled bin edges overlap with original binning.
- Take into account inverse projections: integrals over small segments equal.

$$C_{11} = \frac{1c}{1a+1b+1c+1d+1e}$$

- Requirements:
 - Direct energy scale conversion.
 - Inverse energy scale conversion.





NONLINEARITY APPLICATION: MATRIX

- Is it possible to introduce matrix C : $N^c = CN^a$?

Proper method

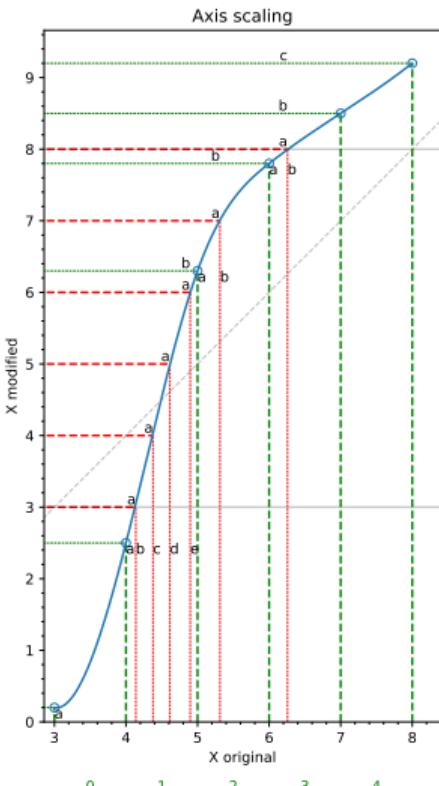
- See how the scaled bin edges overlap with original binning.
- Take into account inverse projections: integrals over small segments equal.

$$C_{11} = \frac{1c}{1a+1b+1c+1d+1e}$$

- Requirements:
 - Direct energy scale conversion.
 - Inverse energy scale conversion.

Assumptions

- Integrand within bin: flat





NONLINEARITY IMPLEMENTATION: EXPRESSION

Dumb method

- Normal and modified energy scale → matrix calculation

```
lsnl_edges| evis_hist, evis_edges()*sum[1] | lsnl_weight[1] * lsnl_component[1]()
enu| ee(evis()), ctheta()
```



NONLINEARITY IMPLEMENTATION: EXPRESSION

Proper method

- Normal and modified energy scale → direct/inverse interpolator, matrix calculation

```
lsnl_coarse = sum[l] | lsnl_weight[l] * lsnl_component_y[l]()
lsnl_interpolator| lsnl_x(), lsnl_coarse, evis_edges()
lsnl_edges| evis_hist
enu| ee(evis()), ctheta()
```



NONLINEARITY IMPLEMENTATION: EXPRESSION

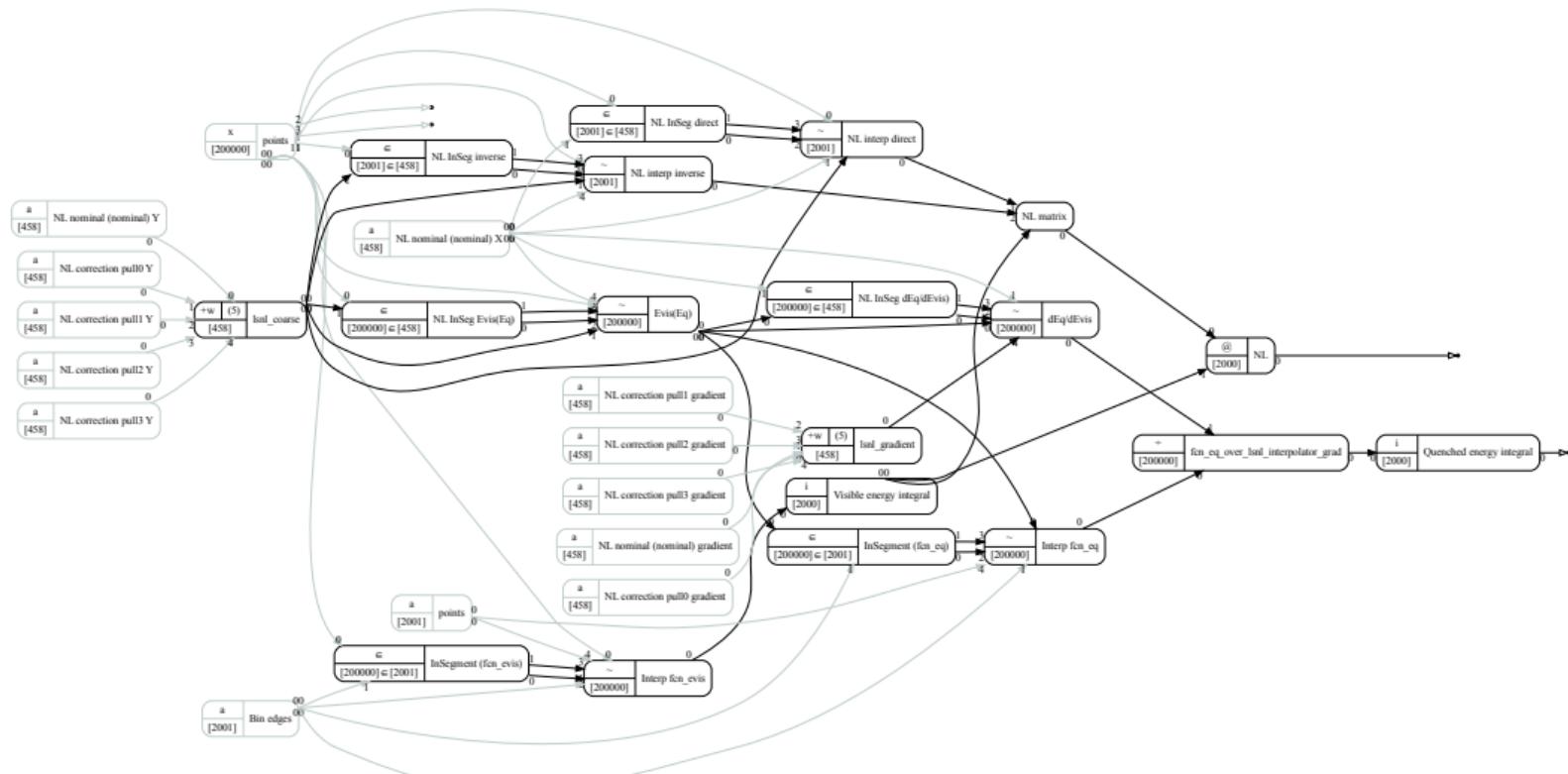
Substitution method

- Normal and modified energy scale → direct/inverse/gradient interpolators

```
lsnl_coarse = sum[1] | lsnl_weight[1] * lsnl_component_y[1]()
lsnl_gradient = sum[1] | lsnl_weight[1] * lsnl_component_grad[1]()
lsnl_interpolator| lsnl_x(), lsnl_coarse, evis_edges(), evis()
lsnl_interpolator_grad| lsnl_gradient
lsnl_edges| evis_hist
enu| ee(lsnl_evis()), ctheta()' or 'enu| ee(evis()), ctheta()'
```



NONLINEARITY IMPLEMENTATION: GRAPH



- The graph contains implementation of 2 methods, used for the cross check



NONLINEARITY APPLICATION IN GNA: TEST MATRIX

Setup

- Check implementation:
 - ▶ Dumb matrix method.
 - ▶ Proper matrix method^{new!}.
 - ▶ Variable substitution^{new!}.



NONLINEARITY APPLICATION IN GNA: TEST MATRIX

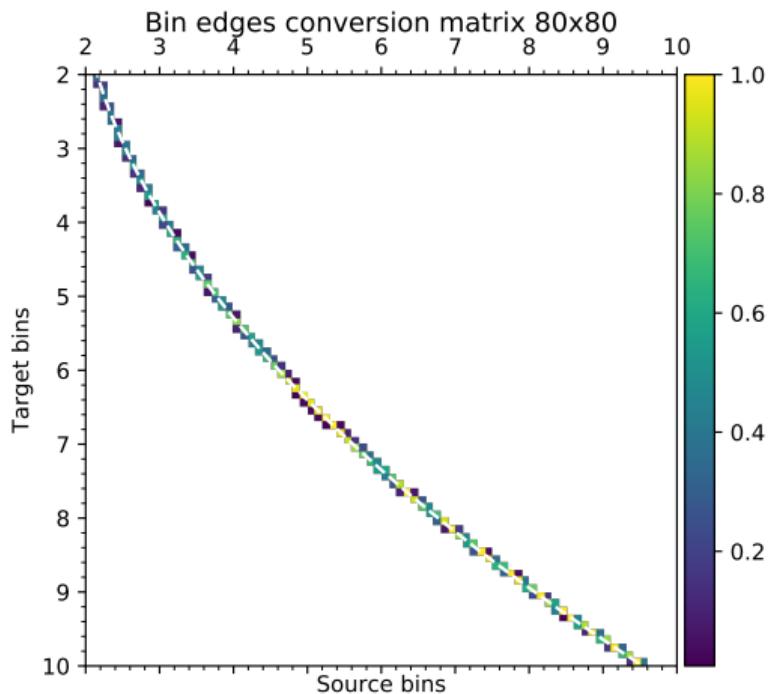
Setup

- Check implementation:
 - ▶ Dumb matrix method.
 - ▶ Proper matrix method **new!**.
 - ▶ Variable substitution **new!**.

Proper matrix

- Example nonlinearity matrix for 80 bins

Energy scale distortion





NONLINEARITY APPLICATION IN GNA: TEST MATRIX

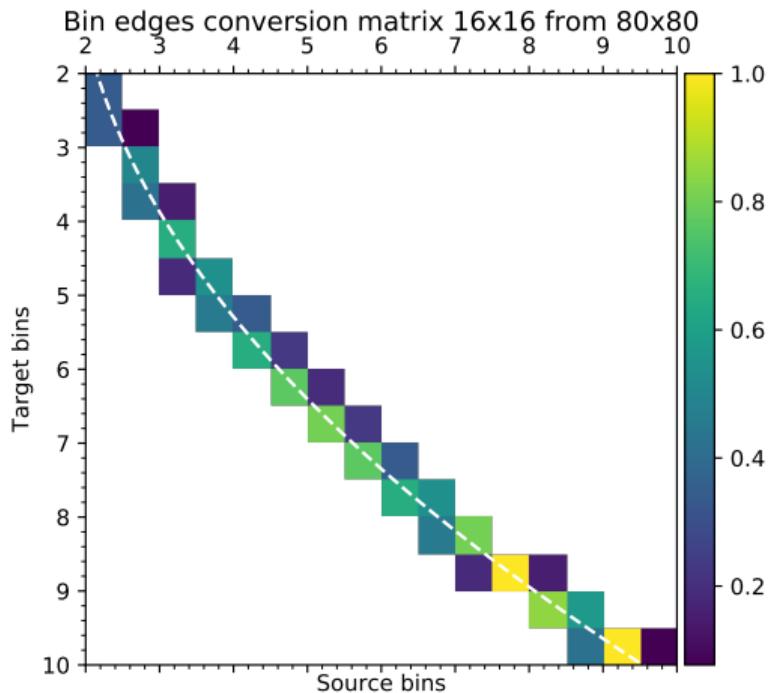
Setup

- Check implementation:
 - ▶ Dumb matrix method.
 - ▶ Proper matrix method **new!**.
 - ▶ Variable substitution **new!**.

Proper matrix

- Example nonlinearity matrix for 80 bins
- May be recalculated into reduced binning

Energy scale distortion





NONLINEARITY APPLICATION IN GNA: TEST MATRIX

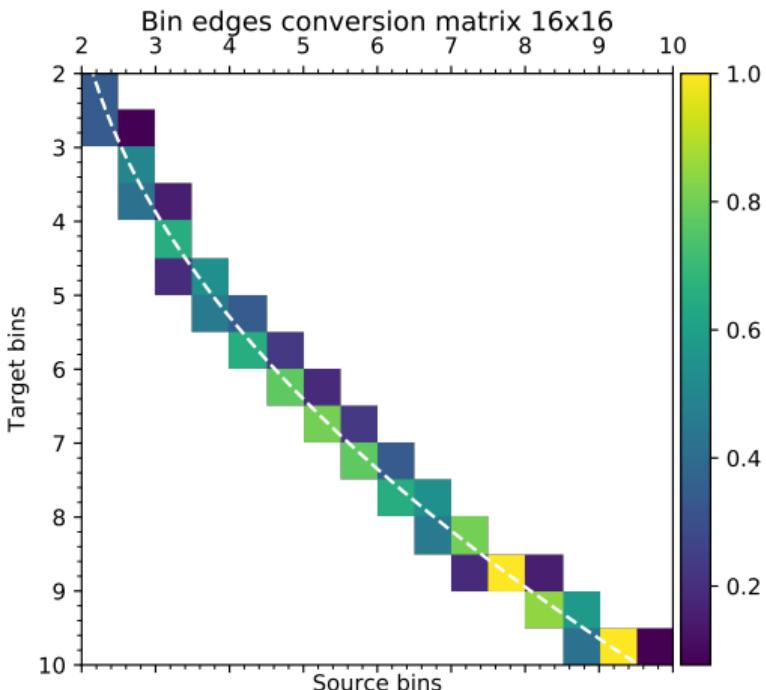
Setup

- Check implementation:
 - ▶ Dumb matrix method.
 - ▶ Proper matrix method **new!**.
 - ▶ Variable substitution **new!**.

Proper matrix

- Example nonlinearity matrix for 80 bins
- May be recalculated into reduced binning
- Identical to the matrix for 16 bins

Energy scale distortion





NONLINEARITY APPLICATION IN GNA: TEST MATRIX

Setup

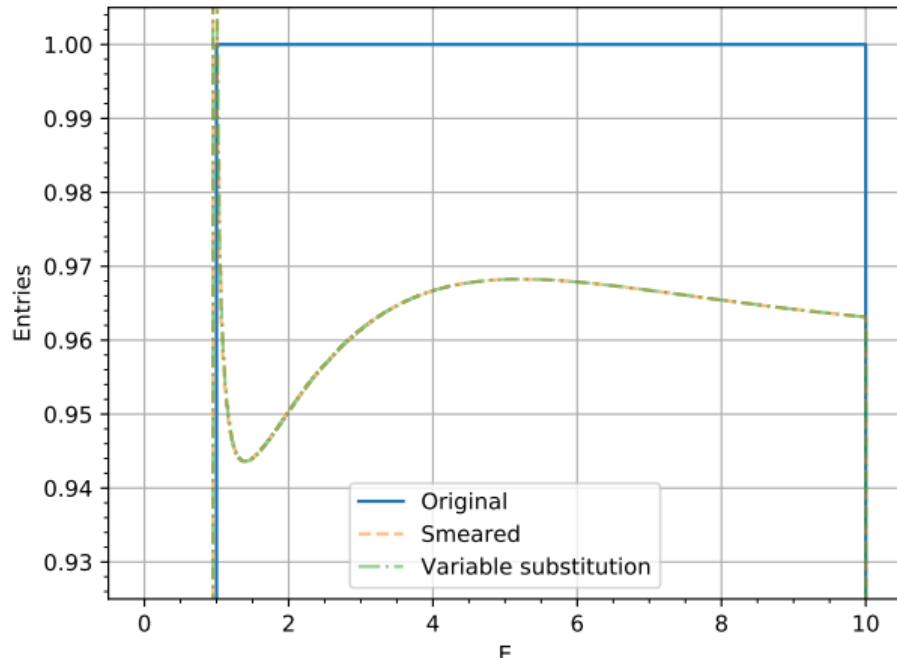
- Check implementation:
 - ▶ Dumb matrix method.
 - ▶ Proper matrix method **new!**.
 - ▶ Variable substitution **new!**.

Proper matrix

- Example nonlinearity matrix for 80 bins
- May be recalculated into reduced binning
- Identical to the matrix for 16 bins

Energy scale distortion

- Consistent for both methods
assuming flat integrand.





NONLINEARITY APPLICATION IN GNA: TEST MATRIX

Setup

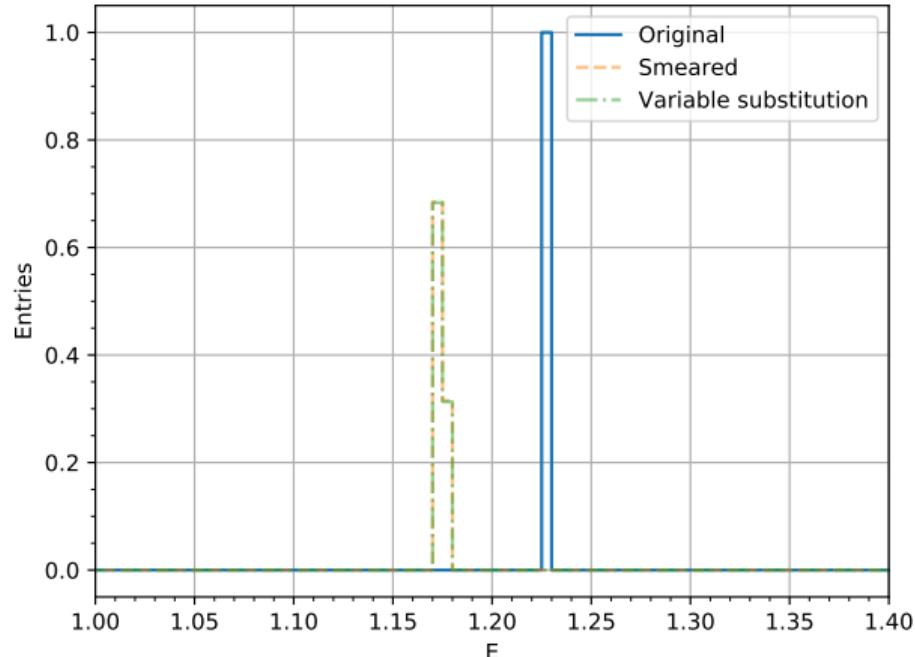
- Check implementation:
 - ▶ Dumb matrix method.
 - ▶ Proper matrix method **new!**.
 - ▶ Variable substitution **new!**.

Proper matrix

- Example nonlinearity matrix for 80 bins
- May be recalculated into reduced binning
- Identical to the matrix for 16 bins

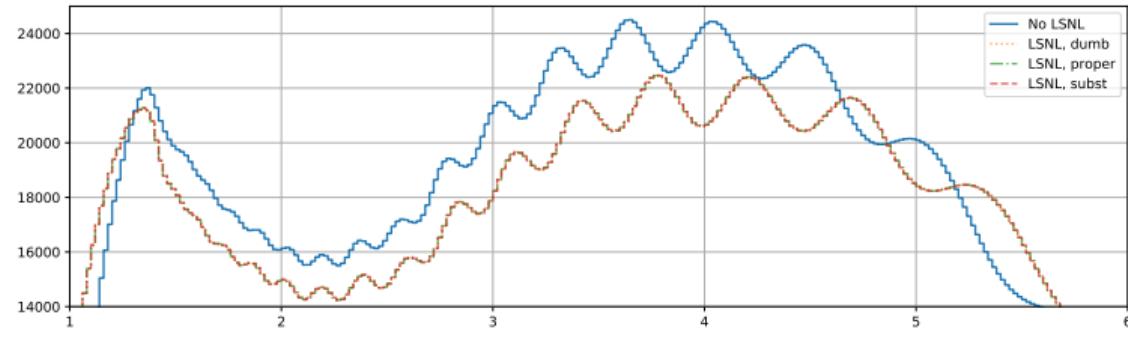
Energy scale distortion

- Consistent for both methods
assuming flat integrand.

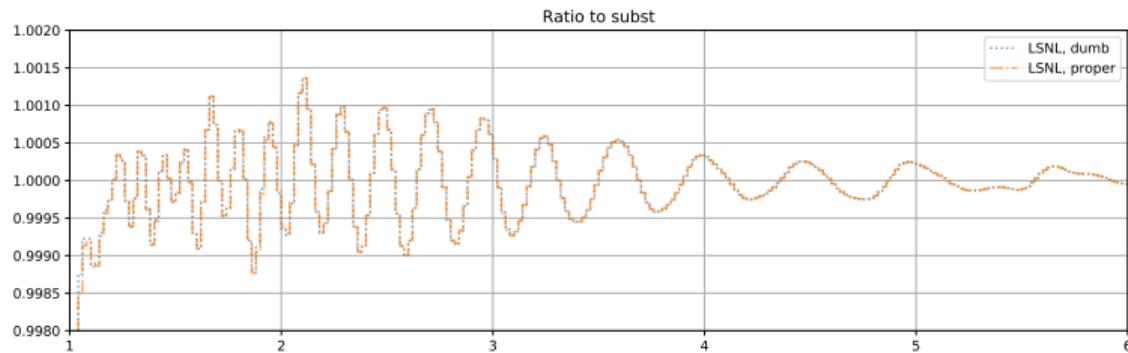




NONLINEARITY APPLICATION IN GNA



✓ Tiny difference



✗ Oscillatory structure