GNA:

New Framework for Statistical Data Analysis

A. Fatkina M. Gonchar D. Naumov K. Treskov

JINR DLNP

CHEP 2018

Outline

Introduction

GNA (Global Neutrino Analysis) — flexible, extensible framework for the statistical data analysis.

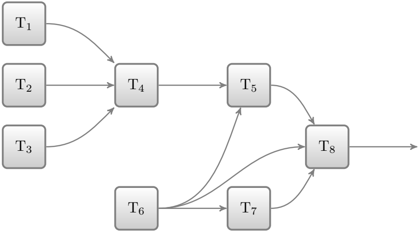
GNA goals

- Comprehensive models with a large number of parameters.
- Data analysis for JUNO and Daya Bay experiments.
- Global analysis of neutrino data (experiments: Daya Bay, JUNO, NOνA, T2K, etc).

Introduction

The idea of GNA

- Dataflow paradigm,
- Physical and programming issues are separated.
- Computations are represented by the graph,
- Nodes of the graph — transformations,
- Computations occur on demand in lazy manner.



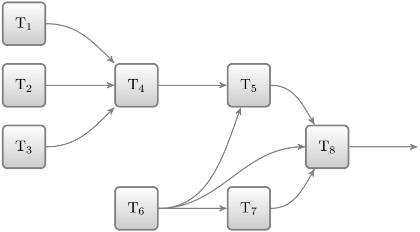
Introduction

GNA highlights

- Efficient complex models with a huge number of parameters,
- High performance fitting.

Expected execution time

- Seconds for a single model evaluation,
- Minutes or hours for multidimensional fit,
- Days or months for MC based methods.



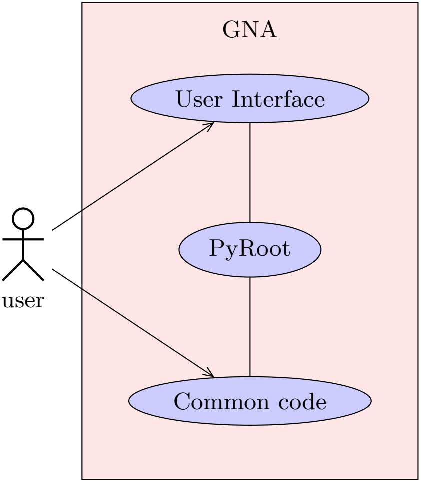
User categories

End-user

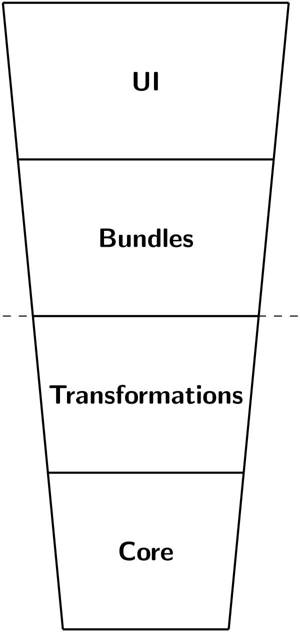
- Assembles computational chain by binding blocks (transformations) via Python UI.

Third-party developer

- Implements algorithms in C++.
- Implements interface in Python, integrates it into GNA environment.
- Other programming issues.



GNA Structure



- Comprehensive command line chain.
- Computational graphs.
- Statistical analysis.

- Read configuration.
- Variables.
- Small computational graph.

Python
(flexibility)

C++
(efficiency)

- Linear algebra
- Integration
- Statistics
- Physics

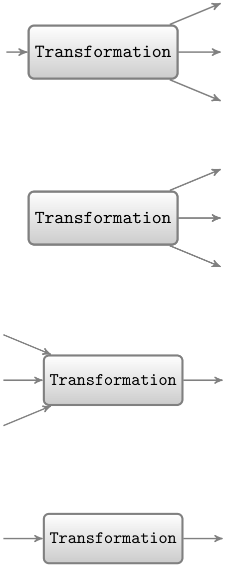
- Data
- Variable
- Transformation

Transformation

Transformation is an encapsulated function, basic component of computations.

Highlights

- May have zero or more inputs and has at least one output (arrays).
- May depend on parameters.
- Data container is associated with each transformation output.
- Transformation has taint flag. It is recomputed in case of it was tainted only (lazy evaluation).



Computational graph

Highlights

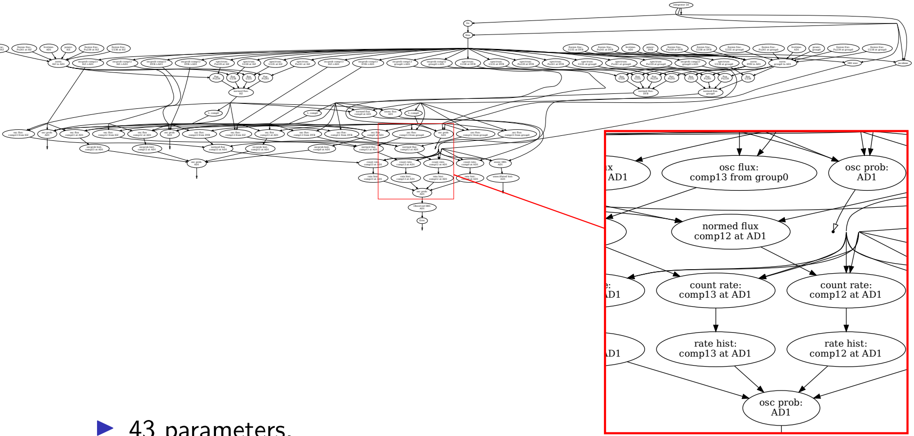
- GNA provides dataflow computations.
- Transformation results are cached.

Two stages of the computational graph usage:

1. Building the computational graph — occurs once and is being made by the framework:
   - Check inputs types.
   - Infer output types.
2. Evaluation on demand.

Computational graph example

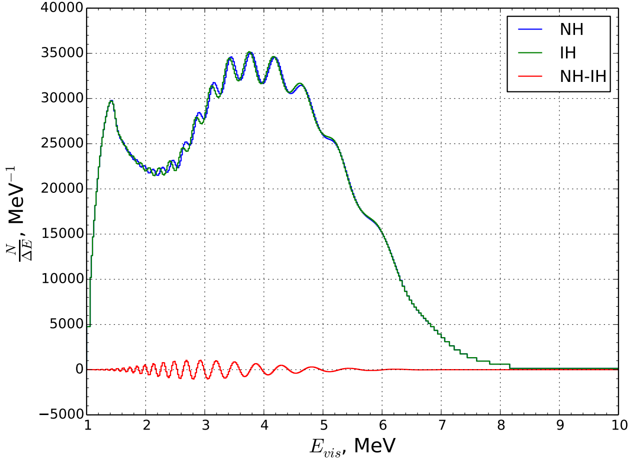
The whole JUNO graph



- 43 parameters.
- The JUNO graph contains 110 nodes and 174 edges.
- It produces a histogram of 280 bins.

Computational graph

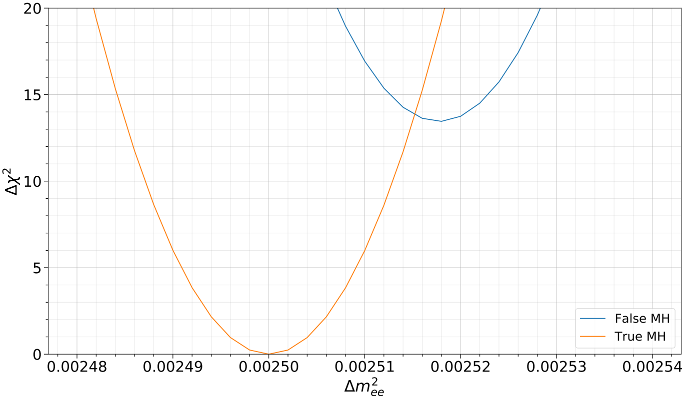
Example of JUNO graph output



Antineutrino spectra expected to be observed in JUNO experiment for different mass hierarchies.

Computational graph

JUNO sensitivity estimate



χ^2 profiles for normal and inverted hierarchies.

Features (Performance)

Lazy evaluation

Computations are performed when (and only in case) the value is used.

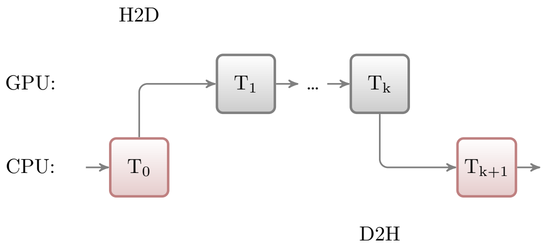
Caching

Transformation is computed once and the result may be reused.

GPU support

Separate library within the framework.

Achieved x20 acceleration for oscillation probability transformation (double precision values, computing-only time, GTX 970M vs. Core 7).



Prospects and summary

The framework is being actively developed now.

Current status:

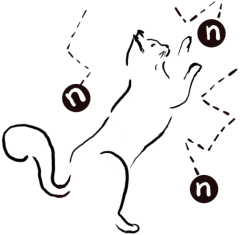
- Flexible framework for data analysis of neutrino experiments.
- May be extended by user-defined transformations.
- The framework is used for the JUNO sensitivity studies.

Our plans include:

- Implementation of the Daya Bay and NOvA oscillation analyses.
- Global analysis of neutrino data produced by several experiments.
- Multicore CPU + GPU systems support.
- Unit-tests, general documentation, release on github.

Release is expected by the end of 2018!

Thank you for your attention!



http://astronu.jinr.ru/wiki/index.php/GNA