Python for Physics & Data Handling

From Interpreter Limitations to High-Performance Vectorized Architectures in HEP

Prof. Dr. Edmar Egidio Purcino de Souza

15th International Doctorate Network in Particle Physics, Astrophysics, and Cosmology - IDPASC - Natal/2026

Federal University of Bahia
edmar.egidio@ufba.br / edmar.egidio@cern.ch

The HEP Computational Challenge

Data Volume vs. Latency

- **The LHC Scale:** Billions of collisions per second produce multi-terabyte data streams.
- **The Dilemma:** C++ offers bare-metal execution speed but possesses a slow prototyping workflow.
- **The Strategy:** Python acts as an interactive steering layer triggering compiled C/C++ backends.

The Modern HEP Paradigm
Maximize user productivity via high-level Python application layers without abandoning the raw execution performance of hardware-optimized infrastructures.

Why Not Pure Python Loops?

The Interpreter Bottleneck

- Python is a **dynamically typed, interpreted language**.
- Every variable evaluation requires checking its underlying C-structure metadata at runtime.
- Loops like `for x in data:` infer type matching on every single iteration.

The Cost of Iteration

Evaluating $100,000$ particle collisions via explicit loops generates massive CPU instruction overhead, resulting in severe processing delays.

The Global Interpreter Lock (GIL)

Concurrency Limitations

- The GIL prevents multiple native CPU threads from executing Python bytecodes simultaneously.
- Pure Python loops cannot exploit multi-core nodes.
- **The Solution:** Push intensive arrays outside the GIL footprint using contiguous binary chunks.

```
# The GIL Trap: Parallel loops
# slow down due to lock overhead.
for thread in threads:
    thread.compute_hep_loop()
```

Understanding Memory Layouts

Standard Python List (Array of Pointers)
Elements are scattered across arbitrary memory addresses. Destroys L1/L2 CPU cache locality because pointer chasing forces continuous RAM lookups.

Contiguous Array Layout (NumPy Style)
Data values sit sequentially in contiguous blocks of memory. Enables advanced SIMD (Single Instruction, Multiple Data) CPU pipelines to fetch data ahead of execution.

The Principle of Vectorization

Delegating to C-Extensions

- Operations apply across whole data blocks simultaneously instead of scalar lines.
- Loops are pushed down to compiled C/Fortran routines internal to NumPy.
- Reduces interpreter overhead from $O(N)$ type checks down to a single $O(1)$ function binding configuration.

$$\vec{C} = \vec{A} + \vec{B}$$

NumPy: The Scientific Substrate

The ndarray object
A homogeneous collection of fixed-size items reflecting a true memory pointer with a fixed strided index map.

Fixed Data Types
Forcing continuous `float64` or `int32` elements discards Python boxing, aligning data directly with hardware architectures.

Element-wise Ops
Standard mathematical symbols (+, −, ∗) are automatically mapped to low-level compiled array iterators.

Data Handling Scale with Pandas

The DataFrame Abstraction

- Built on top of NumPy arrays to bring relational metadata (column identifiers and indices) to scientific code.
- Optimized for complex structured I/O (handling CSV, HDF5, and Parquet formats).
- Enables rapid relational operations like filtering and slicing events.

```
# Instantaneous, zero-loop column
# creation using mathematical
    series
df['pT'] = np.sqrt(df['Px']**2 + df
    ['Py']**2)
```

Fundamentals of Kinematic Reconstruction

Invariant Mass ($M_{\mu\mu}$) Theory Relativistic particle kinematics relies on the conservation of the 4-momentum vector. For a two-body decay:

$$M^2 = (E_1 + E_2)^2 - \|\vec{p}_1 + \vec{p}_2\|^2$$

LINK TO LAB 1A: Kinematic Arrays

This fundamental conservation law is the backbone of **Exercise 1A**. Students leverage this exact linear algebra representation to compute arrays of invariant masses over millions of rows simultaneously without loops.

Memory Strain and Data Chunking

RAM Saturation Mitigation

- Loading massive HEP datasets directly into memory can crash development environments.
- Pandas and NumPy support **lazy evaluation chunking** and stream generator pipelines.
- Managing chunk size maximizes CPU cache line performance.

```
# Processing in manageable data
    blocks
for chunk in pd.read_csv(file,
    chunksize=50000):
    process_hep_vectors(chunk)
```

Masking: Conditional Vector Logic

Boolean Indexing Arrays

- Standard Python `if` statements cannot be used inside vectorized execution blocks.
- Instead, we evaluate conditional logic across an entire array to generate a **Boolean Mask**.
- This mask extracts only the matching memory slots at native speed.

```
# Creates an array of True/False
    entries
mask = df['Massa_Invariante'] >
    90.0

# Extracting the matching subset
z_peak_events = df[mask]
```

Bitwise Operators in Vector Masks

The Logical Trap
Standard Python `and` / `or` evaluate truth values for an entire object as a single scalar entity. This throws errors when applied to arrays.

Bitwise Pipeline Solution (Lab 1B Prep)
Use bitwise operators (`&`, `|`) to evaluate logical expressions element-by-element across independent memory grids.

```
# Vectorized multi-conditional cut
cut = (df['pt1'] > 20) & (df['pt2'] > 20)
```

NumPy Where Architecture

Branchless Selection Execution

- `np.where(condition, x, y)` acts like a vectorized ternary operator.
- It avoids standard CPU branch mispredictions by evaluating both possibilities or masking memory writes directly.
- Crucial for handling mathematical boundaries like avoiding division by zero.

```
# Safeguarding against negative
    mass squares
clean_m2 = np.where(m2 > 0, m2,
    0.0)
```

Radial Detector Geometry Concepts

The HEP Coordinate Workspace

- **Pseudorapidity (η):** Spatial angle relative to the primary beam axis line.
- **Azimuth (ϕ):** Rotation angle around the beam cylinder profile.
- Used because particle production is roughly invariant to Lorentz boosts along the longitudinal beam axis.

LINK TO LAB 1B: Spatial Isolation Cuts
This coordinate transformation framework forms the geometric engine of **Exercise 1B**, where we filter data based on spatial topologies.

The Radial Distance Invariance Effect

Understanding ΔR Separation Quantifying angular distances between particles uses a flat non-Euclidean representation:

$$\Delta R = \sqrt{(\eta_1 - \eta_2)^2 + (\phi_1 - \phi_2)^2}$$

HANDS-ON IMPLEMENTATION: Exercise 1B

In **Lab 1B**, students translate this non-Euclidean equation into a vectorized NumPy chain to compute the distance vector ΔR for thousands of lepton pairs simultaneously, applying a clean isolation cut $\Delta R > 0.4$.

Periodic Boundary Corrections

The Azimuthal Wrap Trap

- The azimuth variable ϕ wraps around at the boundary $[-\pi, +\pi]$.
- Simple subtraction across this boundary yields false large values.
- **Solution:** Vectorized correction adjustments mapped across angular transformations.

```
# Shifting deltas back into true
    circle limits
d_phi = np.where(d_phi > np.pi,
    d_phi - 2*np.pi, d_phi)
```

Element-Wise Bounds Processing

The Global Reduction Error
`np.max()` compresses an entire array into a single maximum value across the whole data table.

LINK TO LAB 1C: Sorting Particle Momentum
In **Exercise 1C**, students deploy `np.maximum()` and `np.minimum()` to split un-sorted tracking series instantly into structural *Leading* and *Sub-leading* arrays.

```
# Extracting the leading particle per event
df['pt_leading'] = np.maximum(df['pt1'], df['pt2'])
```

Physics of Asymmetry Metrics

Quantifying System Balance

- Asymmetry indices scale variations into normalized ranges between $[0, 1]$.
- Helps distinguish different production and decay topologies.
- Enables rapid identification of highly boosted parent states vs rest-frame decays.

$$A_{p_T} = \frac{p_{T,\text{lead}} - p_{T,\text{sub}}}{p_{T,\text{lead}} + p_{T,\text{sub}}}$$

Particle Asymmetry Profiles: Z vs. J/ψ

Heavy Rest Systems (Z Boson)
Massive states produced nearly at rest decay back-to-back, evenly splitting energy between final state particles. This drives the asymmetry distribution close to 0.

APPLIED PHYSICS: Lab 1C Verification
In **Exercise 1C**, students plot and compare these two exact behavioral profiles, analyzing how parent masses shift the system's structural balance in the laboratory frame.

Electroweak Dynamics & Charge Mapping

Chiral Symmetry Breaking Analysis

- Sorting tracks dynamically by particle charge allows us to probe parity-violating physics.
- We isolate the negatively charged lepton (μ^-) to measure its preferred scattering directions.
- This asymmetry reveals insights into the underlying vector and axial-vector couplings of weak force mediators.

```
# Extracting properties based on
    charge state
eta_minus = np.where(df['Q1'] ==
    -1,
                         df['eta1'],
                         df['eta2'])
```

The Forward-Backward Asymmetry (A_{FB})

Quantifying Parity Violation Measuring the angular distribution of decay products tells us about the structure of the underlying interaction:

$$A_{FB} = \frac{N_F - N_B}{N_F + N_B}$$

THE CAPSTONE CHALLENGE

This parity-violating measurement is the focus of the **Module 1 Capstone Challenge**. Students map out A_{FB}'s behavior as a function of the data's scale.

Data Binning Strategies in HEP

Resolving Structural Variation

- Grouping continuous mass spectra into physical bins helps isolate different physics regimes.
- Proper binning avoids smoothing over narrow resonances, while ensuring wide bins have enough statistics in low-count regions.
- Enables tracking how physical observables evolve across different energy scales.

```
# Grouping continuous data into
    bins
bin_mask = (df['Mass'] >= 70) & (df
    ['Mass'] < 110)
```

Statistical Error Propagation in Asymmetries

Ph.D. Precision Level Every experimental HEP measurement must include an evaluation of its statistical uncertainty. For a counting asymmetry:

$$\sigma_{A_{FB}} = \sqrt{\frac{4\,N_F\,N_B}{(N_F + N_B)^3}}$$

COMPUTATIONAL CHECK: Capstone Validation

In the **Capstone Challenge**, students go beyond simple averages to implement this standard statistical uncertainty propagation via vectorized equations across all 5 mass bins.

Matplotlib Internals for HEP

- **Logarithmic Scales:** Essential for exploring cross-sections that span several orders of magnitude, such as comparing QCD backgrounds to rare signal processes.
- **Step Plots:** Accurately represents data bins without implying unmeasured continuity between independent data points.
- **Errorbar Arrays:** Direct link between physical measurements and statistical uncertainties on the plot.

Raw Strings and LaTeX Rendering in Python

Avoiding Escape Sequences

- Python treats `\U` as a Unicode sequence, which breaks when writing LaTeX strings like `\Upsilon`.
- Prepend an `r` to define a **Raw String**, treating backslashes as literal characters.
- Enclose math in `$` to trigger Matplotlib's typesetting engine.

```
# The correct way to write LaTeX
    labels
plt.xlabel(r'$\mu\mu$ Mass [GeV/c$
    ^2$]')
```

Vector Performance Benchmarking

Quantifying Optimization

- We can use Python's `time.time()` module to benchmark code efficiency.
- Moving operations down to the C-layer via vectorization typically speeds up calculations by factors of **100x to 1000x**.

```
start = time.time()
# Run vectorized operations here...
duration = time.time() - start
```

Signal vs. Background Architectures

Standard Backgrounds (QCD)
High-rate, soft interactions that dominate the data stream. These events are typically characterized by lower momentum distributions.

Resonant Signals
Rare, high-mass states that stand out as narrow peaks against the continuum background once proper kinematic cuts are applied.

High-Level Trigger Selection Simulation

Emulating Online Data Filtering

- Saves storage bandwidth by weeding out soft background events early in the data pipeline.
- Isolates rare electroweak processes while keeping high-luminosity data streams manageable.
- Demonstrates how offline data handling techniques mirror the logic used in real-time online trigger systems.

Code Refactoring Checklist for Ph.D. Students

Best Practices for HEP Code

- Replace explicit loops with vectorized alternatives whenever possible.
- Pre-allocate memory using fixed NumPy data types (`np.float64`).
- Handle conditional selections using array masks instead of `if` blocks.
- Always include statistical error tracking for physical observables.

Next Step

Transition to Module 2

The ROOT Ecosystem & High-Performance I/O

Now that we have covered vectorized data handling, we are ready to explore the standard data formats of High-Energy Physics: ROOT files, TTrees, and the Uproot framework.