Computational upgrades to the high energy physics analysis pipeline for future LHC/HL-LHC runs

Saransh Chopra (University College London)
Supervisor: Dr. Jim Pivarski (Princeton University)

Enabling auto-differentiation for the Scikit-HEP ecosystem

Saransh Chopra

Motivation

- Physicists require tuning of hyperparameters in their analysis pipeline, and using any arbitrary function as a loss function in the middle of the pipeline is sometimes required.

- Each part of the pipeline must be individually differentiable to allow picking this loss function in the middle of the pipeline efficiently.

- A really nice resource I found on this was Nathan Simpson's thesis with IRIS-HEP - Data Analysis in High-Energy Physics as a Differentiable Program.

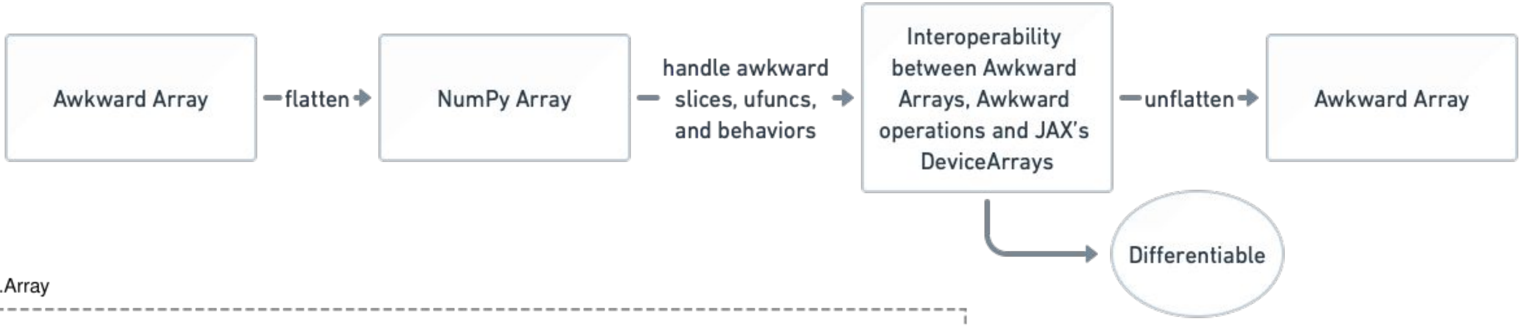
- This thesis resulted in making the statistical part of the pipeline differentiable, including a few common operations like cut - gradhep/neos, gradhep/relaxed.

Motivation

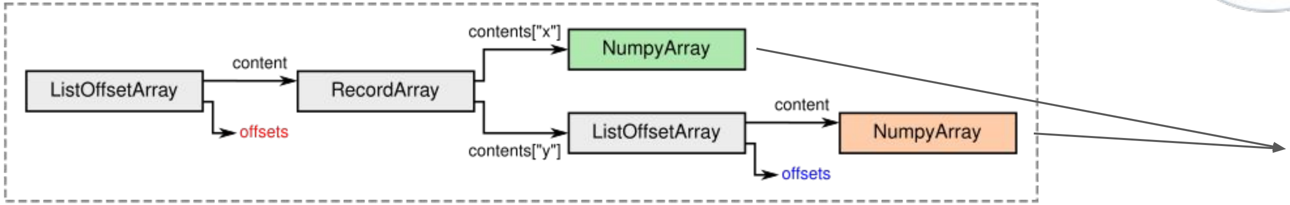
- Awkward's JAX backend existed, but it had some known bugs that was blocking the integration of autodiff in Analysis Grand Challenge.

- The development of the JAX backend had stalled recently to prioritise other work.

Working



ak.Array



represents the following (color-coded):

```
[
    [{"x": 1.1, "y": [1]}, {"x": 2.2, "y": [1, 2]}, {"x": 3.3, "y": [1, 2, 3]}],
    [],
    [{"x": 4.4, "y": [3, 2]}, {"x": 5.5, "y": [3]}]
]
```

Results

```python
import jax
import awkward as ak
import numba
import numpy as np

ak.jax.register_and_check()
```

```python
def f(x):
    return np.power(x[[2, 2, 0], ::-1], 3)
```

```python
primals = ak.Array([[1.0, 2, 3], [], [5, 6]], backend="jax")
tangents = ak.Array([[0.0, 1, 0], [], [0, 0]], backend="jax")
```

```python
val, grad = jax.jvp(f, (primals,), (tangents,))
```

```python
val, grad
```

```
(<Array [[216.0, 125.0], [...], [27.0, 8.0, 1.0]] type='3 * var * float32'>,
 <Array [[0.0, 0.0], [0.0, ...], [0.0, 12.0, 0.0]] type='3 * var * float32'>)
```

```python
print(jax.grad(np.sum)(primals))
```

```
[[1.0, 1.0, 1.0], [], [1.0, 1.0]]
```

More results

```python
import jax
import awkward as ak

ak.jax.register_and_check()
```

```python
a = ak.Array([[1.0, 2, 3], [5, 6]], backend="jax")

def f(x):
    return ak.mean(ak.sum(x) * x)

f(a), jax.grad(f)(a)
```

```
(Array(57.8, dtype=float32),
 <Array [[6.8, 6.8, 6.8], [6.8, 6.8]] type='2 * var * float32'>)
```

More results (with numba)

```python
behavior = {}

input_arr = ak.Array([1.0], backend="jax")

@numba.vectorize(
    [
        numba.float32(numba.float32, numba.float32),
        numba.float64(numba.float64, numba.float64),
    ]
)
def _some_kernel(x, y):
    return x * x + y * y
```

```python
@ak.mixin_class(behavior)
class SomeClass:
    @property
    def some_kernel(self):
        return _some_kernel(self.x, self.y)

ak.behavior.update(behavior)

arr = ak.zip({"x": input_arr, "y": input_arr}, with_name="SomeClass")

arr.some_kernel
```

```
[2.0]
-----------------
type: 1 * float32
```

More results (with coffea)

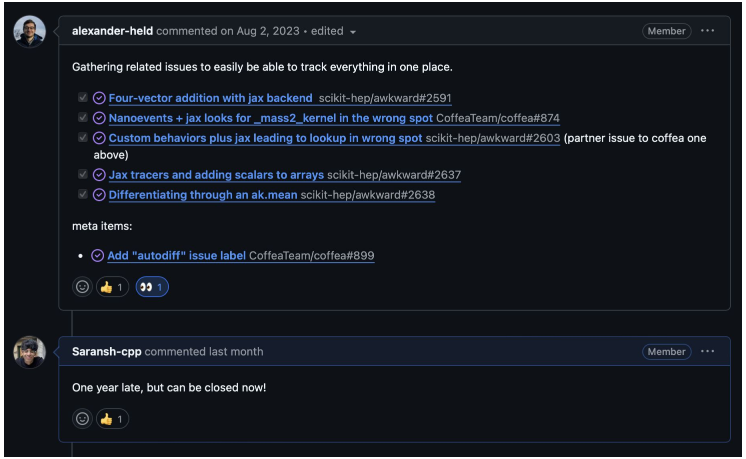
```
ak.behavior.update(candidate.behavior)

ttbar_file = "https://github.com/scikit-hep/scikit-hep-testdata/"\
    "raw/main/src/skhep_testdata/data/nanoAOD_2015_CMS_Open_Data_ttbar.root"

with uproot.open(ttbar_file) as f:
    arr = f["Events"].arrays(["Electron_pt", "Electron_eta", "Electron_phi",
                              "Electron_mass", "Electron_charge"])

px = arr.Electron_pt * np.cos(arr.Electron_phi)
py = arr.Electron_pt * np.sin(arr.Electron_phi)
pz = arr.Electron_pt * np.sinh(arr.Electron_eta)
E = np.sqrt(arr.Electron_mass**2 + px**2 + py**2 + pz**2)

evtfilter = ak.num(arr["Electron_pt"]) >= 2

els = ak.zip({"pt": arr.Electron_pt, "eta": arr.Electron_eta, "phi": arr.Electron_phi,
              "energy": E, "charge": arr.Electron_charge}, with_name="PtEtaPhiECandidate")[evtfilter]
els = ak.to_backend(els, "jax")

els[:, 0].mass
```

```
[0.03125,
 0.0,
 nan,
 0.0,
 0.03125]
-----------------
type: 5 * float32
```

Final results

- Awkward Array, Coffea, and Vector(!) are now differentiable without any *known* issues, paving the way for introducing automatic-differentiation in Analysis Grand Challenge.

- Had a nice long chat with Lino at PyHEP.dev about him possibly picking up autodiff in AGC.



Preparing vector for future LHC/HL-LHC runs

Motivation

- Several issues/design discussions popped up in vector following:
    - its adoption in Coffea
    - the new JAX backend of awkward
    - dask's adoption in the analysis pipeline
    - …
- All the work done changed vector to either solve bugs or adapt the library to physicists' requirements.

Results - v1.2

- fix:
  - **syncing backends to follow the same promotion/demotion scheme for geometric dimensions (demote to the lowest dimension)**
  - returning the correct awkward record when changing dimensions
  - **infix operations should not depend on the order of arguments**
  - **respect user defined awkward mixin subclasses**
- docs:
  - better API docs and tutorials
- chore:
  - migrate to ruff
  - migrate to pytest-doctestplus

Results - v1.3

- feat:
  - allow momentum coords in `to_Vector*D` methods
  - coordinate transformation functions with momentum names
  - `like` method for projecting vector to the coordinate space of a given vector to mandate strict dimensionality checks (`vector_3d + vector_4d` will now error out but `vector_3d + vector_4d.like(vector_3d)` will work)
- fix:
  - **error out on operations on vectors of different geometric dimensions**

Results - v1.3.1

- feat:
  - **make momentum-ness infectious**
  - support dask-awkward 2024.3.0
- fix:
  - **momentum coords should not be repeated with generic coords in subclasses**

Results - v1.4

- feat:
  - **a sympy backend (a whole another project)**
  - allow coord values in to_<coord_names> methods
- fix:
  - call the square implementation for power 2 on object vectors
  - use negfactor in negfactor scale test

Results - v1.4.1

- fix:
  - **sympy backend on numpy 2.0 (full numpy 2.0 support)**
  - add lower and upper bounds for deltaangle
  - maximum for SymPy backend is the identity function now
  - get coordinate classes to work for numpy
- docs:
  - add basic docs for sub-classing awkward mixins
- maintenance:
  - add missing compute function tests
  - add GitHub artifact attestations to package distribution

Final results

- I added support for 1 new backend (SymPy) to vector (and extended support for 2 more backends through Awkward - Dask and JAX).
- A lot of bug fixes pointed out by physicists using vector (physicists are using vector!)
- A few new features (more quality of life upgrades) requested by physicists.
- Vector crossed 1 million downloads!

downloads 1M

VECTOR

Migrating Coffea to Scikit-HEP/vector

Motivation

- Coffea’s vector module pre-dates Scikit-HEP/vector, but now that vector has achieved maturity, it made sense to migrate coffea’s internals to Scikit-HEP/vector.

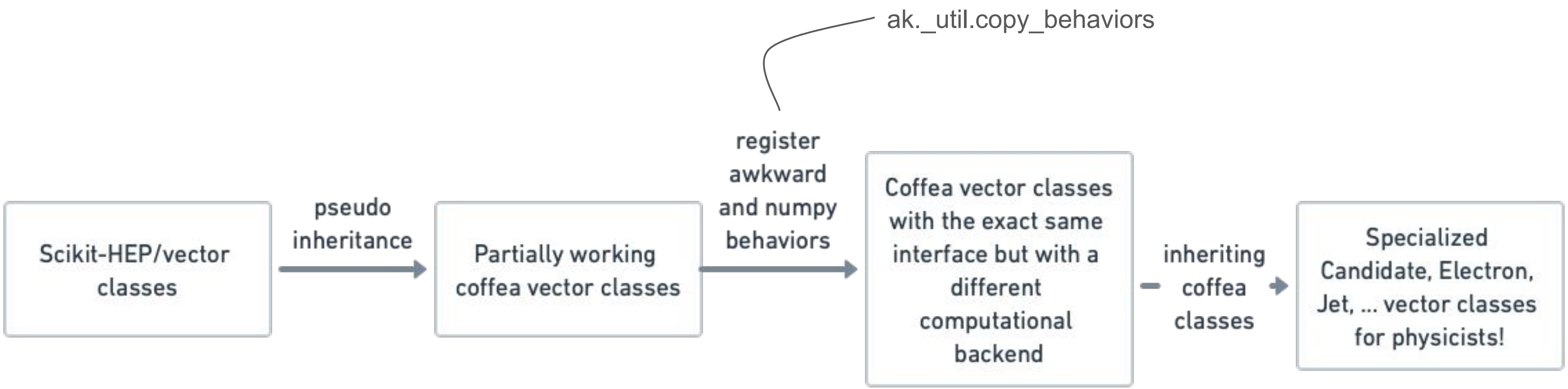
- Scikit-HEP/vector is now much more sophisticated and functional than coffea’s vector sub-package, including support for third party libraries, such as, JAX, Dask, and SymPy.

Coffea

+

VECTOR

Working


ak._util.copy_behaviors
Scikit-HEP/vector classes
pseudo inheritance
Partially working coffea vector classes
register awkward and numpy behaviors
Coffea vector classes with the exact same interface but with a different computational backend
inheriting coffea classes
Specialized Candidate, Electron, Jet, ... vector classes for physicists!

Results

```
filename = "https://raw.githubusercontent.com/CoffeaTeam/coffea/master/tests/samples/nano_dy.root"
events = NanoEventsFactory.from_root(
    {filename: "Events"},
    schemaclass=NanoAODSchema,
    metadata={"dataset": "DYJets"},
).events()
```

```
events.Jet.compute().__repr__()
```

```
"<JetArray [[Jet, ..., Jet], ...] type='40 * var * Jet[area: float32[paramet...'>"
```

Results

```
events.Jet.compute().__class__.__mro__
```

```
(coffea.nanoevents.methods.nanoaod.JetArray,
 coffea.nanoevents.methods.nanoaod.Jet,
 coffea.nanoevents.methods.candidate.PtEtaPhiMCandidate,
 coffea.nanoevents.methods.candidate.Candidate,
 coffea.nanoevents.methods.vector.PtEtaPhiMLorentzVector,
 coffea.nanoevents.methods.vector.LorentzVector,
 vector.backends.awkward.MomentumAwkward4D,
 vector._methods.LorentzMomentum,
 vector._methods.SpatialMomentum,
 vector._methods.PlanarMomentum,
 vector._methods.Momentum,
 vector._methods.MomentumProtocolLorentz,
 vector.backends.awkward.VectorAwkward4D,
 vector.backends.awkward.VectorAwkward,
 vector._methods.Lorentz,
 vector._methods.Spatial,
 vector._methods.Planar,
 vector._methods.Vector4D,
 vector._methods.Vector,
 vector._methods.VectorProtocolLorentz,
 vector._methods.MomentumProtocolSpatial,
 vector._methods.VectorProtocolSpatial,
 vector._methods.MomentumProtocolPlanar,
 vector._methods.VectorProtocolPlanar,
 vector._methods.VectorProtocol,
 coffea.nanoevents.methods.base.NanoCollection,
 coffea.nanoevents.methods.base.Systematic,
 awkward.highlevel.Array,
 awkward._operators.NDArrayOperatorsMixin,
 collections.abc.Iterable,
 collections.abc.Sized,
 object)
```

Results

```
muons = ak.zip(
    {
        "pt": events.Muon.pt,
        "eta": events.Muon.eta,
        "phi": events.Muon.phi,
        "mass": events.Muon.mass,
    },
    with_name="LorentzVector",  # change accordingly - Muon, Jet, ...
    behavior=vector.behavior,  # change accordingly - nanoaod.behavior, candidate.behavior, ...
)
```

```
muons.compute().__repr__()
```

```
"<LorentzVectorArray [[], [], [], [], ..., [], [], [], []] type='40 * var * ...'>"
```

```
import vector

muons = ak.zip(
    {
        "pt": events.Muon.pt,
        "eta": events.Muon.eta,
        "phi": events.Muon.phi,
        "mass": events.Muon.mass,
    },
    with_name="Momentum4D",
    behavior=vector.backends.awkward.behavior,  # ideally use vector.register_awkward()
)
```

```
muons.compute().__repr__()
```

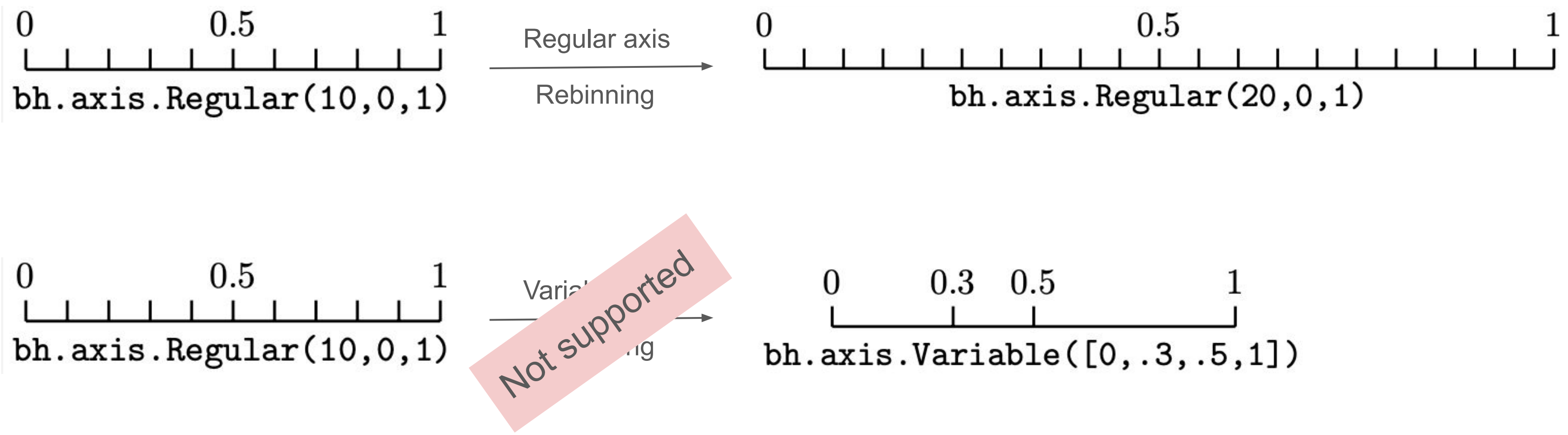
```
"<MomentumArray4D [[], [], [], [], ..., [], [], [], []] type='40 * var * Mom...'>"
```

Final results

- Migration changes were a part of Coffea v2024.8.0.
- Can expect Coffea to entirely scrape its vector module and ask users to depend on Scikit-HEP/vector by the end of the year.

Implementing non-uniform rebinning in boost-histogram

Motivation



0
0.5
1
bh.axis.Regular(10,0,1)
Regular axis
Rebinning
0
0.5
1
bh.axis.Regular(20,0,1)
0
0.5
1
bh.axis.Regular(10,0,1)
Not supported
0
0.3
0.5
1
bh.axis.Variable([0,.3,.5,1])

Motivation

Math

LM, JR, MD, JH, DS

Priority 1:

- PyROOT: better histos and graphs interoperability with numpy arrays, honour UHI protocol
- Histograms: advance current RHist implementation to one testable by experiments
- Improve interface to pass initial error values or covariance matrix to Minuit2
- Release a library for Lorentz vector computations on accelerators in SYCL (also using generic n-dim arrays as inputs)

Priority 2:

- Deliver plan and prototype of algorithmic improvements when dealing with parameter constraints in ROOT's minimisers
- PyROOT: Pythonise TF{1,2,3} and numerical algorithms interfaces (e.g. minimisers)
- Prototype SYCL kernels to be JITted (see Interpreters objectives)
- Histograms: Model and prototype of pipelining GPU histogram filling

ROOT 2024 Plan of Work - D. Piparo, CERN EP-SFT - 15-1-2024 30

#345 [FEATURE] Non-uniform rebinning

scikit-hep/hist | Nov 15th, 2021 | Added by GitHub

rebin.py

fabriceMUKARAGE/rebinning_histogram | Added by GitHub

1 1

+1 39 replies Last reply 2 months ago

Full UHI #208

Open henryiii opened this issue on Nov 12, 2019 · 1 comment

henryiii commented on Nov 12, 2019 Member

The full UHI includes arbitrary reducers and rebinnings. This will need to be implemented before 1.0, but depends on #194 and #189; those should go into 0.6.0 so let's nominally target 0.7.0 for a full UHI implementation.

[FEATURE] Non-uniform rebinning #345

Open **yimuchen** opened this issue on Nov 15, 2021 · 10 comments

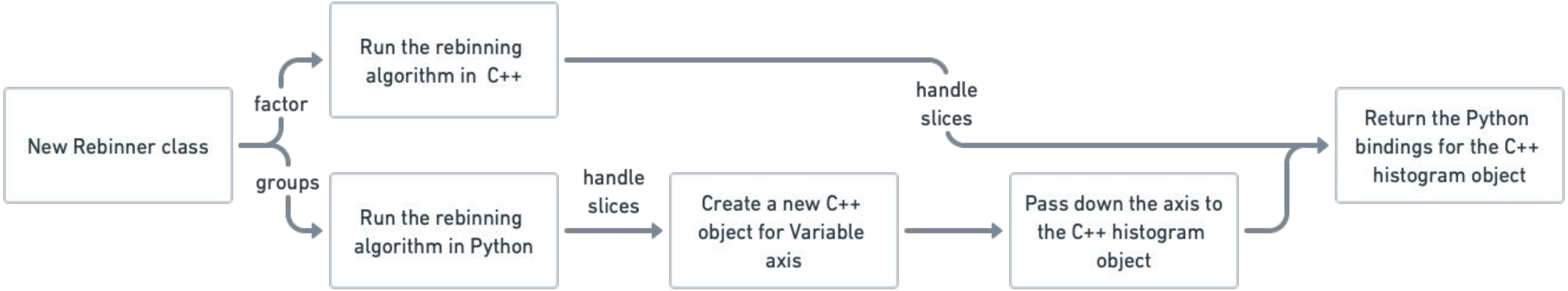
yimuchen commented on Nov 15, 2021 · edited by henryiii

Right now, histograms can only be rebinned by some integer amount via the `hist.rebin` indicator. I would be nice if there was some way to rebin a certain axis arbitrary bin edges, as we might want to rebin a regular axis to be irregular just for low statistic region, without requiring the exact binning scheme to be known during histogram construction.

I'm not sure what the most optimal method should be, maybe something like extending the existing `hist.rebin` class to include something like `hist.rebin( <new_axis>/<new bin edges> )` to specify the new binning scheme of interest?

7

Working



New Rebinner class
factor
groups
Run the rebinning algorithm in C++
Run the rebinning algorithm in Python
handle slices
Create a new C++ object for Variable axis
Pass down the axis to the C++ histogram object
handle slices
Return the Python bindings for the C++ histogram object

Results

```
h = bh.Histogram(bh.axis.Regular(10, 0, 1))
h.fill(np.random.normal(size=1_000_000))
```

```
Histogram(Regular(10, 0, 1), storage=Double()) # Sum: 341530.0 (1000000.0 with flow)
```

```
rebin = bh.rebin(factor=2)
```

```
h[::rebin]
```

```
Histogram(Regular(5, 0, 1), storage=Double()) # Sum: 341530.0 (1000000.0 with flow)
```

```
rebin = bh.rebin(groups=[1, 2, 3, 4])
```

```
h[::rebin]
```

```
Histogram(Variable([0, 0.1, 0.3, 0.6, 1], metadata=...), storage=Double()) # Sum: 341530.0
```

Results

```python
s = bh.tag.Slicer()
```

```python
h = bh.Histogram(
    bh.axis.Regular(20, 1, 3),
    bh.axis.Regular(30, 1, 3),
    bh.axis.Regular(40, 1, 3)
)
```

```python
h[{0: s[:: bh.rebin(groups=[1, 2, 3, 4, 10])]}].axes.size
```

```
(5, 30, 40)
```

```python
h[
    {
        0: s[:: bh.rebin(groups=[1, 2, 3, 4, 10])],
        2: s[:: bh.rebin(groups=[1, 2 ,3, 4, 10, 20])]
    }
].axes[2].edges
```

```
array([1.  , 1.05, 1.15, 1.3 , 1.5 , 2.  , 3.  ])
```

Final results

- Released as a feature in boost-histogram v1.5

Adding a sympy backend in vector

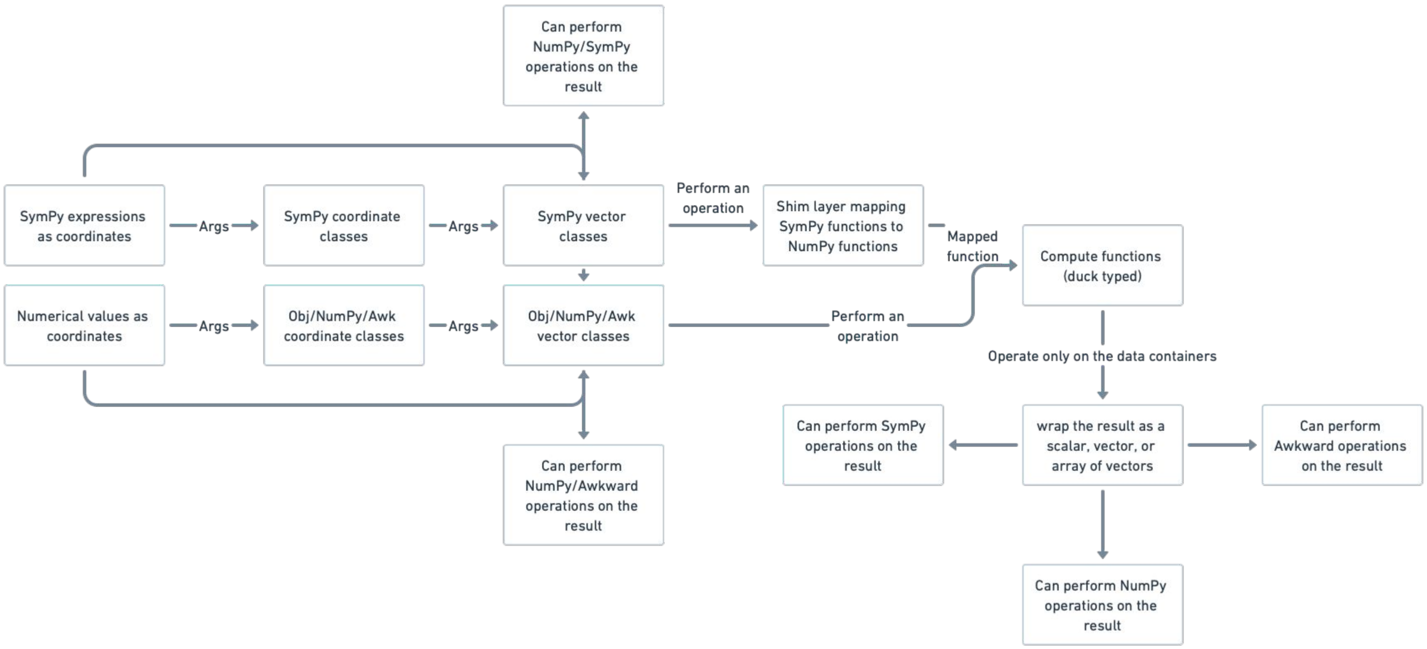
Motivation

- Along with experimental physicists using vector for numerical computations, the SymPy backend will enable theoretical physicists to utilize the library for symbolic computations.

- Since the SymPy vector classes and their momentum equivalents operate on SymPy expressions, all of the standard SymPy methods and functions work on the vectors, vector coordinates, and the results of operations carried out on vectors.

- Vector's *compute* functions operate on data containers, and this behavior is tested using uncompyle6 on python 3.8. Once Python 3.8 reaches EOL, the SymPy backend will allow testing of this behavior.

Working



Can perform NumPy/SymPy operations on the result
SymPy expressions as coordinates
Args
SymPy coordinate classes
Args
SymPy vector classes
Perform an operation
Shim layer mapping SymPy functions to NumPy functions
Mapped function
Compute functions (duck typed)
Numerical values as coordinates
Args
Obj/NumPy/Awk coordinate classes
Args
Obj/NumPy/Awk vector classes
Perform an operation
Operate only on the data containers
Can perform SymPy operations on the result
wrap the result as a scalar, vector, or array of vectors
Can perform Awkward operations on the result
Can perform NumPy/Awkward operations on the result
Can perform NumPy operations on the result

Results

```
import vector
```

```
v = vector.MomentumObject4D(pt=1, phi=2, eta=3, M=10)
v
```

```
MomentumObject4D(pt=1, phi=2, eta=3, mass=10)
```

```
v.to_beta3()
```

```
MomentumObject3D(pt=0.07047186284717237, phi=2, eta=3)
```

```
v.boost(v.to_beta3())
```

```
MomentumObject4D(px=-1.1810297606283302, py=2.580597106671111, pz=28.430850335643896, mass=10)
```

```
v.boost(v.to_beta3()).px
```

```
-1.1810297606283302
```

Results

```python
import vector; import sympy
```

```python
pt, phi, eta, M = sympy.symbols("pt phi eta M", real=True)
```

```python
v = vector.MomentumSympy4D(pt=pt, phi=phi, eta=eta, M=M)
v
```

```
MomentumSympy4D(pt=pt, phi=phi, eta=eta, mass=M)
```

```python
v.to_beta3()
```

```
MomentumSympy3D(pt=pt/sqrt(M**2 + 0.25*pt**2*(1 + exp(-2*eta))**2*exp(2*eta)), phi=phi, eta=eta)
```

```python
sympy.init_session()
```

```
IPython console for SymPy 1.12 (Python 3.11.5-64-bit) (ground types: python) •••
```

```python
v.boost(v.to_beta3()).px
```

$$\frac{pt^3 \sin^2(\phi)\cos(\phi)}{\left(1+\frac{1}{\sqrt{-\frac{pt^2\sin^2(\phi)}{M^2+0.25pt^2(1+e^{-2\eta})^2e^{2\eta}}-\frac{pt^2\cos^2(\phi)}{M^2+0.25pt^2(1+e^{-2\eta})^2e^{2\eta}}-\frac{pt^2\sinh^2(\eta)}{M^2+0.25pt^2(1+e^{-2\eta})^2e^{2\eta}}+1}}\right)\left(M^2+0.25pt^2(1+e^{-2\eta})^2e^{2\eta}\right)\left(-\frac{pt^2\sin^2(\phi)}{M^2+0.25pt^2(1+e^{-2\eta})^2e^{2\eta}}-\frac{pt^2\cos^2(\phi)}{M^2+0.25pt^2(1+e^{-2\eta})^2e^{2\eta}} - \cdots\right.}$$

Results

```
v.boost(v.to_beta3()).px
```

$$\frac{pt^3 \sin^2(\phi)\cos(\phi)}{\left(1 + \frac{1}{\sqrt{-\frac{pt^2\sin^2(\phi)}{M^2+0.25pt^2\left(1+e^{-2\eta}\right)^2 e^{2\eta}} - \frac{pt^2\cos^2(\phi)}{M^2+0.25pt^2\left(1+e^{-2\eta}\right)^2 e^{2\eta}} - \frac{pt^2\sinh^2(\eta)}{M^2+0.25pt^2\left(1+e^{-2\eta}\right)^2 e^{2\eta}} + 1}}\right)\left(M^2 + 0.25pt^2(1+e^{-2\eta})^2 e^{2\eta}\right)\left(-\frac{pt^2\sin^2(\phi)}{M^2+0.25pt^2(1+e^{-2\eta})^2 e^{2\eta}} - \frac{pt^2\cos^2(\phi)}{M^2+0.25pt^2(1+e^{-2\eta})^2 e^{2\eta}} - \cdots\right.}$$

```
v.boost(v.to_beta3()).px.simplify()
```

$$pt\left(\sqrt{M^2 + pt^2\cosh^2(\eta)}\left(\sqrt{\frac{1.0M^2e^{2\eta}+0.25pt^2e^{4\eta}-1.0pt^2e^{2\eta}\sinh^2(\eta)-0.5pt^2e^{2\eta}+0.25pt^2}{M^2e^{2\eta}+0.25pt^2(e^{2\eta}+1)^2}} + 1\right)\left(1.0M^2e^{2\eta} + 0.25pt^2e^{4\eta} - 1.0pt^2e^{2\eta}\sinh^2(\eta) - 0.5pt^2e^{2\eta} + \cdots\right.\right.$$

Results

```
values = {pt: 1, phi: 1, eta: 1, M: 1}
v.boost(v.to_beta3()).px.subs(values)
```

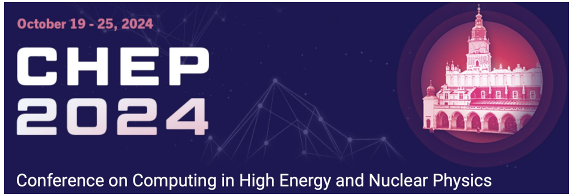
$$\frac{\sin^2(1)\cos(1)}{\left(1+0.25(e^{-2}+1)^2e^2\right)\left(1+\frac{1}{\sqrt{-\frac{\sinh^2(1)}{1+0.25\left(e^{-2}+1\right)^2e^2}-\frac{\sin^2(1)}{1+0.25\left(e^{-2}+1\right)^2e^2}-\frac{\cos^2(1)}{1+0.25\left(e^{-2}+1\right)^2e^2}+1}}\right)\left(-\frac{\sinh^2(1)}{1+0.25(e^{-2}+1)^2e^2}-\frac{\sin^2(1)}{1+0.25(e^{-2}+1)^2e^2}-\frac{\cos^2(1)}{1+0.25(e^{-2}+1)^2e^2}+1\right)} + \frac{}{\left(1 -\right.}$$

```
v.boost(v.to_beta3()).px.subs(values).evalf()
```

1.98699002164743

Final results

- The SymPy backend was released as a part of vector v1.4.
- Caveat: Operations on SymPy vectors are only 100% compatible with numeric vectors (Python, NumPy, and Awkward backends) if the vectors are positive time-like. The space-like and negative time-like cases have different sign conventions.
- Abstract accepted at CHEP 2024 as a poster (October).
- Abstract accepted at PyHEP (presented).



Histogramming on GPUs - cuda-histogram

Motivation

- Manasvi's work on Awkward Arrays on GPUs garnered a lot of interest, especially from Coffea developers.

- Though the work on Awkward Arrays is being carried out in full throttle, more pieces are required to perform a complete analysis of high energy physics data on GPUs.

- One of the major missing pieces is the ability to generate and manipulate histograms as objects on CUDA.

Working

- Implements a subset of the features of boost-histogram using CuPy (see API documentation for a complete list), completely independent from boost-histogram:
  - Axes
    - Regular and Variable axes
    - edges()
    - centers()
    - index(...)
    - ...
  - Histogram
- fill(..., weight=...) (including Nan flow)
  - simple indexing with slicing
  - values(flow=...)
  - variance(flow=...)
- Allows users to detach the generated GPU histogram to CPU -
  - to_boost() - converts to boost-histogram.Histogram
  - to_hist() - converts to hist.Hist

Working

- Differences from boost-histogram/hist API:
  - Has an additional NaN flow
  - Accepts only CuPy arrays
  - underflow is indexed as 0 and not -1
  - ax[...] will return a cuda_histogram.Interval object
  - No interpolation is performed
  - Hist indices should be in the range of bin edges, instead of integers

- Near future goals for the package -
  - Implement support for Categorical axes (exists internally but need refactoring to match boost-histogram's API)
  - Improve indexing (__getitem__) to exactly match boost-histogram's API

Results

```
import cuda_histogram; import cupy as cp

ax1 = cuda_histogram.axis.Regular(10, 0, 1)
ax2 = cuda_histogram.axis.Variable([0, 2, 3, 6])

h = cuda_histogram.Hist(ax1, ax2)

>>> ax1, ax2, h
(Regular(10, 0, 1), Variable([0. 2. 3. 6.]), Hist(Regular(10, 0, 1), Variable([0. 2. 3. 6.])
```

Results

```
h.fill(cp.random.normal(size=1_000_000), cp.random.normal(size=1_000_000))  # set weight=...

>>> h.values(), type(h.values())  # set flow=True for flow bins (underflow, overflow, nanflc
(array([[28532.,  1238.,    64.],
       [29603.,  1399.,    61.],
       [30543.,  1341.,    78.],
       [31478.,  1420.,    98.],
       [32692.,  1477.,    92.],
       [32874.,  1441.,    96.],
       [33584.,  1515.,    88.],
       [34304.,  1490.,   114.],
       [34887.,  1598.,   116.],
       [35341.,  1472.,   103.]]), <class 'cupy.ndarray'>)
```

Results

```
>>> ax1.index(0.5)
array([6])

>>> ax1.index(-1)
array([0])

>>> ax1[0]
<Interval ((-inf, 0.0)) instance at 0x1c905208790>

>>> h[0, 0], type(h[0, 0])
(Hist(Regular(1, 0.0, 0.1), Variable([0. 2.])), <class 'cuda_histogram.hist.Hist'>)

>>> h[0, 0].values(), type(h[0, 0].values())
(array([[28532.]]), <class 'cupy.ndarray'>)

>>> h[0, :].values(), type(h[0, 0].values())
(array([[28532.,  1238.,    64.]]), <class 'cupy.ndarray'>)

>>> h[0.2, :].values(), type(h[0, 0].values()) # indices in range of bin edges
(array([[30543.,  1341.,    78.]]), <class 'cupy.ndarray'>)

>>> h[:, 1:2].values(), type(h[0, 0].values()) # no interpolation
C:\Users\Saransh\Saransh_softwares\OpenSource\Python\cuda-histogram\src\cuda_histogram\axis\
  warnings.warn(
(array([[28532.],
       [29603.],
       [30543.],
       [31478.],
       [32692.],
       [32874.],
       [33584.],
       [34304.],
       [34887.],
       [35341.]]), <class 'cupy.ndarray'>)
```

Results

```
h.to_boost()

>>> h.to_boost().values(), type(h.to_boost().values())
(array([[28532.,  1238.,    64.],
       [29603.,  1399.,    61.],
       [30543.,  1341.,    78.],
       [31478.,  1420.,    98.],
       [32692.,  1477.,    92.],
       [32874.,  1441.,    96.],
       [33584.,  1515.,    88.],
       [34304.,  1490.,   114.],
       [34887.,  1598.,   116.],
       [35341.,  1472.,   103.]]), <class 'numpy.ndarray'>)

h.to_hist()

>>> h.to_hist().values(), type(h.to_hist().values())
(array([[28532.,  1238.,    64.],
       [29603.,  1399.,    61.],
       [30543.,  1341.,    78.],
       [31478.,  1420.,    98.],
       [32692.,  1477.,    92.],
       [32874.,  1441.,    96.],
       [33584.,  1515.,    88.],
       [34304.,  1490.,   114.],
       [34887.,  1598.,   116.],
       [35341.,  1472.,   103.]]), <class 'numpy.ndarray'>)
```

Final results

- Moved to Scikit-HEP at PyHEP.dev! v0.1.0 available on PyPI!

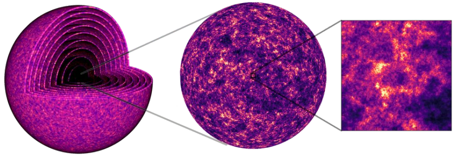
What am I doing now?

What am I doing now?

- Writing a JOSS paper for vector, preparing for CHEP, and grad school applications soon.

- Moved to London and joined UCL's Advanced Research Computing Centre as a full-time Assistant Research Software Engineer.
    - Developing contents/infrastructure and TAing for "Research Software Engineering with Python."
    - Adding GPU (CuPy) and autodiff (JAX) support to full-universe simulations for cosmology (GLASS - Generator for Large Scale Structure).

- Still maintaining vector+cuda-histogram and answering issues/discussions related to my work.
    - I get 5% FTE to spend on research software outside of my official work.



Acknowledgements

- Thank you IRIS-HEP for funding this work!

- I can definitely not end this without thanking Jim for being an amazing supervisor. Absorbing and working with such vast knowledge in such a short period was possible only because of his constant support!

- Given that my work encompassed multiple pieces or libraries of the data analysis pipeline, I was fortunate enough to be guided by several other incredible people - Henry Schreiner, Lindsey Gray, Nicholas Smith, Alexander Held, Matthew Feickert, … - and I am thankful to each one of them.

Thank you!

Backup

Introduction

- High energy physics data is not regular/uniform. A particular stream of collision events can produce different number of particles.

- Awkward Array is designed to make working with ragged arrays as trivial as manipulating regular (non-ragged) N-dimensional arrays in NumPy.

- JAX is Autograd and XLA, brought together for high-performance numerical computing. The high level API (`jax.numpy`) is basically JIT-compileable and differentiable numpy

- One can make custom data containers compatible with JAX API by registering a way to flatten and unflatten them.

Introduction

- Vector is a Python library for 2D, 3D, and Lorentz vectors, especially arrays of vectors, to solve common physics problems in a NumPy-like way.

- Vector has (had) 5 backends - pure Python Objects, NumPy arrays for vectors, and Awkward arrays of vectors + Numba support for Object type and Awkward type vectors for JIT compilation

- I worked on vector two years back!
  > Schreiner, H., Pivarski, J., & Chopra, S. vector [Computer software]. https://doi.org/10.5281/zenodo.5942082

Introduction

- Coffea provides basic tools and wrappers for enabling not-too-alien syntax when running columnar Collider HEP analysis.

- It makes use of Scikit-HEP libraries like uproot and awkward-array but also implements histogramming, plotting, and vector functionalities on its own + it is possible with coffea to scale a HEP analysis from a testing on a laptop to: a large multi-core server, computing clusters, and super-computers.

Introduction

- Boost-histogram provides the python bindings for Boost::Histogram, a C++14 library. This is one of the fastest libraries for histogramming, while still providing the power of a full histogram object.

- Universal Histogram Interface (UHI) is a standard for histogramming formalised by IRIS-HEP, but it has still not been adopted entirely by boost-histogram, ROOT, Hist, ….

- Every library is pushing to adopt the UHI standard.

- I worked on implementing the rebinning piece of UHI.

Introduction

- SymPy is a Python library for symbolic mathematics, a full-featured computer algebra system (CAS) written entirely in Python.

- Vector can perform numerical computation on high energy physics using pure Python, NumPy, and Awkward Arrays; hence, it is used by experimental physicists in their analysis pipelines.

- Vector's SymPy backend will create a stronger connection between software used by experimentalists and software used by theorists.