IRIS-HEP Fellowship 2022
Vector: Constructors, documentation, and benchmarks

Saransh Chopra (University of Delhi)
Mentors: Henry Schreiner (Princeton University), Jim Pivarski (Princeton University)
Vector

- Vector allows working with 2D, 3D, and Lorentz vectors, to solve common physics problems in a NumPy-like way.

- Uses names and conventions set by ROOT’s TLorentzVector and Math::LorentzVector, henryiii/hepvector, and coffea.nanoevents.methods.vector

- Users can create vectors in a variety of coordinate systems, including Cartesian, cylindrical, spherical, and any combination of these with time or proper time for Lorentz vectors.

- Seamlessly integrates with the existing Scikit-HEP libraries, especially with Awkward.

- Comes loaded with 3 + 2 backends; a pure Python object backend, a NumPy backend, an Awkward Array backend, an Object-Numba, and an Awkward-Numba backend to leverage JIT (Just In Time) compiled calculations on vectors.
Basic usage example

```python
In [1]: import vector
In [2]: v1, v2 = vector.obj(x=1, y=2), vector.obj(x=2, y=3)
In [3]: v1, v2
Out[3]: (VectorObject2D(x=1, y=2), VectorObject2D(x=2, y=3))
In [4]: v1 + v2
Out[4]: VectorObject2D(x=3, y=5)
In [5]: v1.to_Vector3D(), v1.to_rhophietat()
Out[5]:
(VectorObject3D(x=1, y=2, z=0),
 VectorObject4D(zho=-2.23606797749979, phi=1.1071487177940904, eta=0, t=0))
In [6]: v1 & v2
Out[6]: 8
In [7]: v1.azimuthal
Out[7]: AzimuthalObjectXY(x=1, y=2)
In [8]: v1.rotateZ(0.5), v1.scale(2), v1.is_perpendicular(v2)
Out[8]:
(VectorObject2D(x=-0.08126851531801325, y=2.234590662384985),
 VectorObject2D(x=2, y=4), False)
```
Awkward Arrays of vector

```
In [1]: import vector, skhep_testdata, uproot
In [2]: data = uproot.open(skhep_testdata.data_path("uproot-HZZ.root"))
In [3]: tree = data['events']; branches = tree.arrays()
In [4]: px, py, pz, E = branchesMuon_Px, branchesMuon_Py, branchesMuon_Pz, branchesMuon_E
In [5]: muons = vector.zip({"px": px, "py": py, "pz": pz, "E": E})
In [6]: muons
Out[6]: <MomentumArray4D [[[x: -52.9, y: -11.7, ... t: 69.6]]] type='2421 * var * Moment...'>
In [7]: muons.scale(2), muons.azimuthal, muons.px
Out[7]:
(<MomentumArray4D [[[x: -106, y: -23.3, ... t: 139]]] type='2421 * var * Momentum...'>,
AzimuthalAwkwardXY(<Array [[-52.9, 37.7], ... 1.14], [23.9]] type='2421 * var * float32'>,
<Array [[-11.7, 0.693], ... 63.6], [-35.7] type='2421 * var * float32'>),
<Array [[-52.9, 37.7], ... 1.14], [23.9]] type='2421 * var * float32'>)
```
Detailed usage examples

- Vector's README - [https://github.com/scikit-hep/vector#readme](https://github.com/scikit-hep/vector#readme)
Goals

1. Constructors
   Expose the inner constructors and classes to the user API.

2. Documentation
   Improve the existing API documentation, add tutorials, and add documentation for new features.

3. Benchmarks
   Develop a benchmark suite for Vector which will then be propagated to other Scikit-HEP packages.

Summarising: Preparing Vector for its first major release!
Motivation - constructors

- The user-facing API of Vector did not expose its true API, which were the Python classes. The API provided wrapper functions (named as `obj`, `arr`, and `awk`) to users.

- These functions were also displayed in the `__repr__` method of the classes making the vector constructors even more ambiguous.
Object type constructors - before

```python
In [1]: import vector
In [2]: vector.obj(x=1, y=2)
Out[2]: vector.obj(x=1, y=2)
In [3]: vector.obj(px=1, y=2)
Out[3]: vector.obj(px=1, y=2)
In [4]: vector.obj(px=1, y=2, eta=3, m=4)
Out[4]: vector.obj(px=1, py=2, eta=3, mass=4)
In [5]: vector.VectorObject2D.from_xy(1, 2)
Out[5]: vector.obj(x=1, y=2)
In [6]: vector.VectorObject2D(1, 2)
---------------------------------------------------------------------------
TypeError                       Traceback (most recent call last)
Cell In [6], line 1
    ----> 1 vector.VectorObject2D(1, 2)
TypeError: __init__() takes 2 positional arguments but 3 were given
In [7]:
```
Object type constructors - now

In [1]: import vector

In [2]: vector.obj(x=1, y=2)
Out[2]: VectorObject2D(x=1, y=2)

In [3]: vector.obj(px=1, y=2)
Out[3]: MomentumObject2D(px=1, py=2)

In [4]: vector.obj(px=1, y=2, eta=3, m=4)
Out[4]: MomentumObject4D(px=1, py=2, eta=3, mass=4)

In [5]: vector.VectorObject2D.from_xy(1, 2)
Out[5]: VectorObject2D(x=1, y=2)

In [6]: vector.VectorObject2D(x=1, y=2)
Out[6]: VectorObject2D(x=1, y=2)
NumPy type constructors - before

```python
In [1]: import vector

In [2]: vector.array([[1.1, 2.1], (1.2, 2.2)], dtype=[("x", float), ("y", float)])
Out[2]: VectorNumpy2D([[1.1, 2.1], (1.2, 2.2)], dtye=[('x', '<f8'), ('y', '<f8')])

In [3]: vector.array({"x": [1, 2], "y": [3, 4]})
Out[3]: <VectorArray2D [{x: 1, y: 3}, {x: 2, y: 4}] type='2 * Vector2D["x": int64, "y": ...']

In [4]: vector.array([[1.1, 2.1], (1.2, False)], dtype=[("x", float), ("y", float)])
Out[4]: VectorNumpy2D([[1.1, 2.1], (1.2, 0.0)], dtye=[('x', '<f8'), (y', '<f8')])

In [5]: vector.array([[1.1, 2.1], (1.2, False), ("x", float), ("y", complex)]
Out[5]: VectorNumpy2D([[1.1, 2.1+0.0j], (1.2, 0.0+0.0j)],
                      dtye=[('x', '<f8'), (y', '<c16')])
```
NumPy type constructors - now

In [1]: import vector

In [2]: vector.array([[1.1, 2.1], [1.2, 2.2]], dtype=[("x", float), ("y", float)])
Out[2]: VectorNumpy2D([[1.1, 2.1], [1.2, 2.2]], dtype=[('x', '<f8'), ('y', '<f8')])

In [3]: vector.arr({"x": [1, 2], "y": [3, 4]})
Out[3]: VectorNumpy2D([[1., 3.], [2., 4.]], dtype=[('x', '<f8'), ('y', '<f8')])

In [4]: vector.array([[1.1, 2.1], [1.2, False]], dtype=[("x", float), ("y", float)])
Out[4]: VectorNumpy2D([[1.1, 2.1], [1.2, 0.]], dtype=[('x', '<f8'), ('y', '<f8')])

In [5]: vector.array([[1.1, 2.1], [1.2, False]], dtype=[("x", float), ("y", complex)])
---------------------------------------------------------------------------
TypeError          Traceback (most recent call last)
...               TypeWarning: a coordinate must be of the type numpy.integer or numpy.float

In [6]: vector.VectorNumpy2D([[1.1, 2.1], [1.2, 2.2]], dtype=[("x", float), ("y", float)])
Out[6]: VectorNumpy2D([[1.1, 2.1], [1.2, 2.2]], dtype=[('x', '<f8'), ('y', '<f8')])
Awkward type constructors - before

```python
In [1]: import vector, skhep_testdata, uproot

In [2]: data = uproot.open(skhep_testdata.data_path("uproot-HZZ.root"))

In [3]: tree = data['events']; branches = tree.arrays()


In [5]: muons = vector.awk({"px": px, "py": py, "pz": pz, "E": E})

In [6]: muons # required -> <MomentumArray4D [[[x: -52.9, y: -11.7, ... t: 69.6]]] type='2421 * var * Moment...'
Out[6]: <MomentumArray4D [[[x: [-52.9, 37.7], ... t: [69.6]]] type='2421 * Momentum4D"x"...'>

In [7]: vector.awk({"x": [complex(1, 2), False], "y": [1, 2]})
Out[7]: <VectorArray2D [[[x: (1+2j), y: 1], ... x: False, y: 2]] type='2 * Vector2D"x": ...'>

In [8]: vector._backends.awkward_.AzimuthalAwkwardRhoPhi(1, 2)
Out[8]: <vector._backends.awkward_.AzimuthalAwkwardRhoPhi at 0x138fed8c0>
```
Awkward type constructors - now

```python
In [1]: import vector, skhep_testdata, uproot
In [2]: data = uproot.open(skhep_testdata.data_path("uproot-HZZ.root"))
In [3]: tree = data['events']; branches = tree.arrays()
In [5]: muons = vector.awk({"px": px, "py": py, "pz": pz, "E": E})
---------------------------------------------------------------------------
TypeError                      Traceback (most recent call last)
...                           ...
TypeError: a coordinate must be of the type int or float
In [6]: vector.zip({"px": px, "py": py, "pz": pz, "E": E})
Out[6]: <MomentumArray4D [[(x: -52.9, y: -11.7, ... t: 69.6)]] type='2421 * var * Moment...'>
In [7]: vector.awk({"x": [complex(1, 2), False], "y": [1, 2]})
---------------------------------------------------------------------------
TypeError                      Traceback (most recent call last)
...                           ...
TypeError: a coordinate must be of the type int or float
In [8]: vector.backends.awkward.AzimuthalAwkwardRhoPhi(1, 2)
Out[8]: AzimuthalAwkwardRhoPhi(1, 2)
```
Motivation - documentation

- Vector lacked proper user as well as API documentation and tutorials.
- There were no detailed explanations in the existing tutorials, and most of the functions and classes did not contain docstrings.
Documentation - before

Vector: Constructors, documentation, and benchmarks

Saransh Chopra
Vector: Constructors, documentation, and benchmarks

Overview

Vector is a Python 3.7+ library for 3D, 30D, and Lorentz vectors, especially arrays of vectors, to solve common physics problems in a NumPy-like way.

Main features of Vector:

- Pure Python with NumPy as its only dependency. This makes it easier to install.
- Vectors may be represented in a variety of coordinate systems: Cartesian, cylindrical, pseudospherical, and any combination of these with time or proper time for Lorentz vectors. In all, there are 12 coordinate systems: \((x, y, \theta, \phi)\) in the azimuthal plane \((x, y, \theta, \phi)\) longitudinally \((x, y, t, \phi)\) temporally.
- Uses names and conventions set by ROOT's TLorentzVector and Math/TLorentzVector, as well as scikit-hep/math, uproot-methods.TLorentzVector, hepvector, and coffea.nanoevents.methods.vector.
- Implemented on a variety of backends:
  - pure Python objects
  - NumPy array of vectors (as a structured array subclass)
  - Awkward Arrays of vectors
  - potential for more: CuPy, TensorFlow, Torch, JAX...
- NumPy/Awkward backends also implemented in Numba for JIT-compiled calculations on vectors.
Motivation - benchmarks

- Scikit-HEP packages, including Vector, lack benchmarks.
- Benchmarking would be a valuable addition to the Vector project, which could then be propagated to all Scikit-HEP packages in the future.
A minimal benchmarking suite
A minimal benchmarking suite - zooming in
A minimal benchmarking suite - a regression?
Work done outside of the goals but on vector

- A new build backend - hatch
- Type safety - all constructors are now type safe
- CI developments - adding python 3.10 and 3.11-dev support, removing python 3.6 support, setting up codecov, doctest, pre-commits, …
- Awkward._v2 support - vector is now tested on awkward v1 and awkward._v2
- Miscellaneous fixes and additions - mypy fixes, constructor bugs, passing repo review, …
Work done outside of vector

scikit-hep/awkward
- A minor bug fix - Awkward._v2 now passes a copy of RecordArray’s internal fields in HL API

scikit-hep/cookie
- Added support for coverage (pytest-cov, Codecov, and GitHub Actions)

scikit-hep/scikit-hep.github.io
- Added a new page on coverage (covering tools like Codecov, pytest-cov and GitHub Actions)

scikit-hep/hist
- Restructured documentation
- Removed redundant metadata

Other miscellaneous work
- Migrated HEP tutorials (analysis-grand-challenge, atlas-outreach-data-tools, etc.) from plain Python to vector!
Final outcomes with regards to vector

Vector v0.9.0 is out!

- Features: improved reprs, deltaRapidityPhi, backends are public now, …
- Documentation: major documentation overhaul, doctests, CITATION.cff, …
- Bug fixes: type checks, fix nan_to_num, …
- Maintenance: hatchling backend, support awkward._v2 in tests, build and test on Python 3.10 and 3.11-dev, …

More about v0.9.0 here - https://vector.readthedocs.io/en/latest/changelog.html#version-0-9

Vector v0.10.0 is out too!

- Removed support for Python 3.6

More about v0.10.0 here - https://vector.readthedocs.io/en/latest/changelog.html#version-0-10
What’s next?

- Developing awkward constructors
- A solid support for awkward v2
- A major release (v1.0.0)
- Setting up an infrastructure and website for benchmarks
- Probably propagating the benchmarking suite to other Scikit-HEP packages
- Maintaining vector - keeping it up to date!
Talks and community

- Participated as vector's maintainer in PyHEP hackashop which unfortunately had no turnover (at least for vector).
- Engaged in a lot of discussions with Scikit-HEP developers.
Thank you!

Had a wonderful time contributing to the Scikit-HEP ecosystem!
A special thanks to my mentors Henry Schreiner and Jim Pivarski, and to the members of IRIS-HEP, Princeton Computing, and Scikit-HEP!