11_quadrimpulso-classe

June 2, 2023

[ ]:

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

import numpy as np

# A class that define a four-vector and its Lorentz transformations.
# Each instance is initialized with the particle mass and calculates the
# energy and momentum in a frame with desired velocity.

class Quadrimpulso():

    def __init__(self, massa):
        # Of course instance variables need not be named as the arguments to
        # the __init__ function.
        self.m = massa
        # Constants can be used to initialize instance variables as well.
        self.c = 3.0e+8
        # Class methods can be called in the init function.
        # Notice that to call class methods have to be called against the
        # self variable.
        self.boost(0.0)

    def gamma(self, v):
        g = 1.0 / np.sqrt(1 - v*v / self.c**2) # Lorentz factor
        return g

    def boost(self, v):
        # Apply the Lorentz boost to the four-vector.
        g = self.gamma(v)
        e = g * self.m * self.c**2
        p = g * self.m * v
        # Notice that instance variables can be defined at any point and not
        # only in the __init__ function.  This provides a lot of flexibility,
        # potentially at the expence of consistency, because different
        # instance could have different variables defined.
        self.v = v
        self.e = e
        self.p = p

    def __add__(self, p1):
        # Operator overload.
        # The argument is another instance of the Quadrimpulso class.
        # By defining the __add__ function we enable the usage of the "+"
        # symbol with infix notation.
        # In this case, the result of the sum is a new four-vector
        # with the total mass equal to the sum of the masses, that is, we
        # consider the system of two particles at rest.
        # Notice that this is a different definition than simply adding the
        # energies and the momenta!
        ps = Quadrimpulso(self.m + p1.m)
        return ps


# Define two instances.  The argument is the particle mass.
pr1 = Quadrimpulso(0.001)
pr2 = Quadrimpulso(0.002)
# Thanks to operator overloading we have defined the sum of the two instances.
prs = pr1 + pr2
print(prs.m)

# Boost the two four-vectors and check that the boosting each particle
# separately and the summing the energies is the same as summing the masses
# in the rest frame and then boosting the result.
v = 2.0e+8
pr1.boost(v)
pr2.boost(v)
prs.boost(v)
print(  ( pr1.e + pr2.e - prs.e  ) / prs.e )
```

RIASSUNTO:

- le costanti possono essere usate per inizializzare delle variabili
- la funzione **add** implementa l'utilizzo del "+" come operatore "overload"