GriSPy: A Python package for Fixed-Radius Nearest Neighbors Search

```
>>> pip install grispy
```

Martin Chalela[a,b,c], Emanuel Sillero[a,b,c], Luis Pereyra[a,b,c], Mario Alejandro Garcia[d], Juan B. Cabral[e,a], Marcelo Lares[a], Manuel Merchán[a,b]

a Instituto de Astronomía Teórica y Experimental - Observatorio Astronómico de Córdoba (IATE, UNC–CONICET)
b Observatorio Astronómico de Córdoba, Universidad Nacional de Córdoba.
c Facultad de Matemática, Astronomía y Física, Universidad Nacional de Córdoba (FaMAF–UNC)
d Facultad Regional Córdoba, Universidad Tecnológica Nacional (FRC–UTN)
e Centro Internacional Franco Argentino de Ciencias de la Información y de Sistemas (CIFASIS, CONICET–UNR)

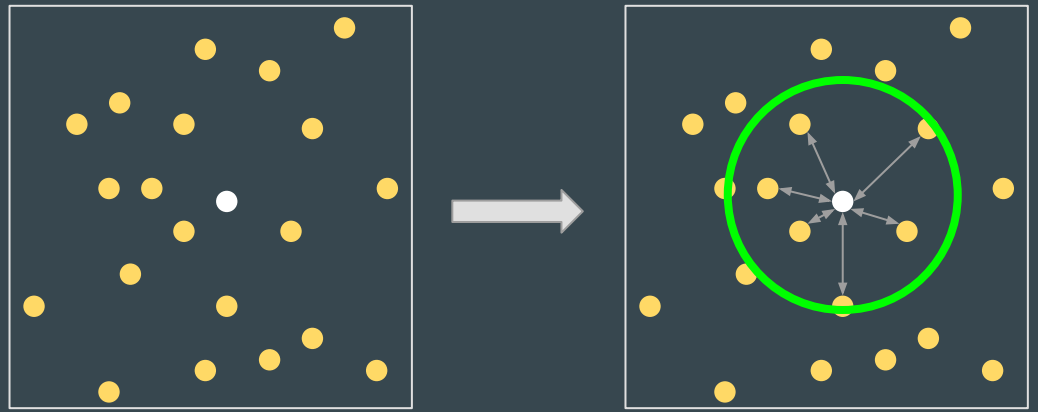
We present a new near neighbors searching algorithm at fixed radius, developed in Python. This module indexes a set of k-dimensional points on a regular grid, with optional periodic conditions, providing a quick approach to close neighbor queries. In this first version, we implement three types of queries: bubble, shell and nearest nth; as well as three different metrics of astronomic interest: the Euclidean and two distance functions in spherical coordinates of variable precision, Haversine and Vincenty; and also, the ability to provide a custom distance function. This package is particularly useful for large data sets where a brute force search becomes impractical.

Chalela, et al. 2021. arXiv 1912.09585

What is the Nearest Neighbors Problem?

The nearest neighbors search (NNS) problem can be defined as follows: given a set ***P*** of points defined in a multidimensional space, run an algorithm that, given a query point ***Q***, finds every point of ***P*** within distance *D* from ***Q***. This problem arises in a wide range of scientific fields, including machine learning, robotics , chemistry, astronomy and many other areas of application. In the particular field of astronomy, the everyday increasing amount of observational and simulated data requires algorithms that can handle the computational demands.

Several methods have been proposed for solving the NNS problem. The most popular method is to apply a partitioning-indexing scheme to track the approximate location of points in the multidimensional space. Among the algorithms that apply this concept are the binary-tree and cell techniques. GriSPy implements a cell technique method to efficiently solve the NNS problem.



How can we solve this problem with GriSPy?

Indexing

Given an initial set of *k*-dimensional points, a regular grid of N^k cells is built in the domain of the data. After the grid is defined, every point is associated with a cell.

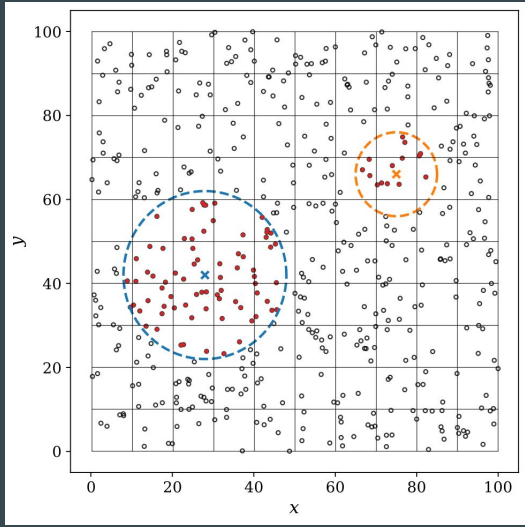


Figure 1. Example of a 2D uniform distribution of points (empty circles). We used GriSPy to index these points in a regular grid. The color crosses mark the centres of interest and their corresponding neighbors are shown as red circles.

Searching

Given a *centre point* and a search radius, we search for neighbors only within those cells touched by the radius.

The `GriSPy()` class provides the indexed set of points. Then, we can perform 3 different types of queries:

`.bubble_neighbors()`
Find neighbors within a given radius. A different radius for each centre can be provided.
Figure 1 is an example of this method.

`.shell_neighbors()`
Find neighbors within given lower and upper radii.

`.nearest_neighbors()`
Find the *n*-th nearest neighbors for each centre.

When should I use GriSPy?

We compared the time performance of GriSPy against similar packages, cKDTree (scipy) and BallTree (scikit-learn), considering the case of a gravitational N-body simulation. In Figure 2 we show the time spent in each step of the process, Build Time (indexing of points) and Query Time (searching for neighbors). The principal gain that GriSPy offers is a fast building time, contrary to a fast querying time provided by the other two methods. **When dealing with large sets of ~10^7 data points, GriSPy outperforms the rest.**

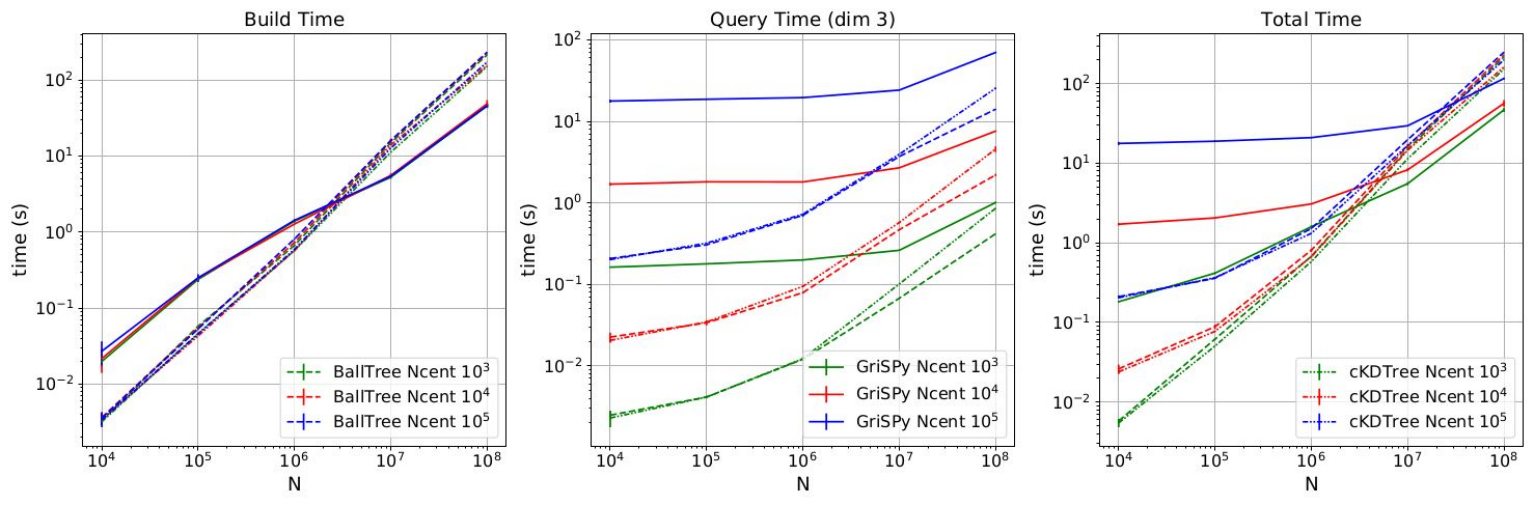
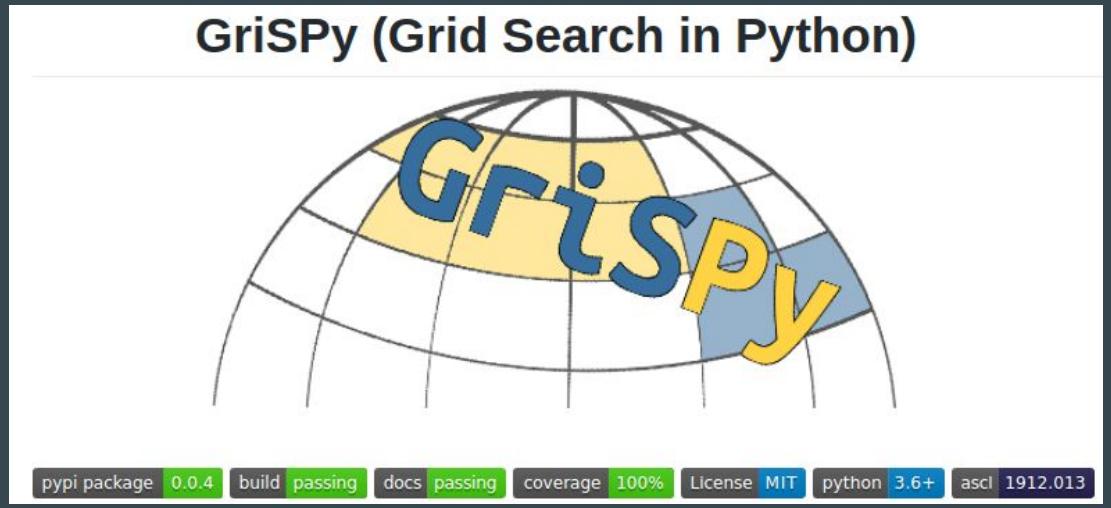


Figure 2. Time comparison of GriSPy, cKDTree and BallTree in a gravitational N-body simulation. Each column represents the time spent in each step, from left to right: Build, Query and Total time. For clarity the legend specifying the package and number of centres is splitted across the three subplots.

Repository, installation and documentation

Github: https://github.com/mchalela/GriSPy



Documentation: https://grispy.readthedocs.io/en/latest/

https://github.com/mchalela/GriSPy

Thanks!