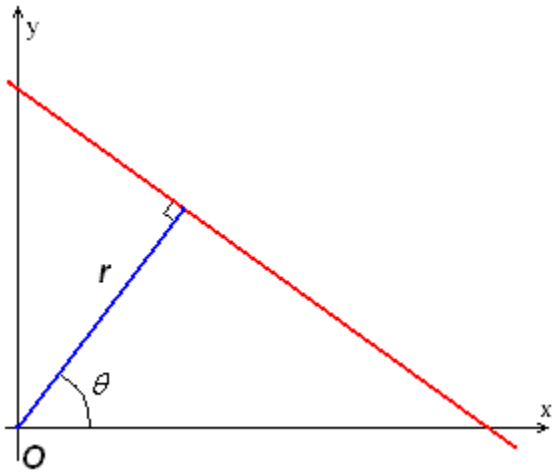
QUANTUM MOTION

Python Test

Introduction

You have been provided with a **.npy** file that contains an array of shape [779, 3]. Each of the rows of this array describes a line described in polar coordinates (shown in the figure below). Lines in polar coordinates are described by the length of their normal vector (**r**) that passes through the origin and the angle that vector makes with the x-axis (**θ/theta**). The first column in the array contains the r value whilst the second contains the theta value. The third column contains a score value attributed to each line (this has no bearing on the equation of the line).



The challenge is to take the lines in this array and output a dataframe to .**csv** whose rows describe pairs of these lines. Each pair should contain one line of positive gradient and one line of negative gradient. The dataframe should be sorted by how close the x-intercepts of the lines in the pair are, containing only the **100** closest pairs. For example, a pair whose lines cross the x-axis at 0.4 and 0.5 should rank higher than a pair whose lines cross at 1.0 and 2.0. The data frame should contain columns that describe each of the line's gradients and y-intercepts, their x-intercept difference, and the sum of their score value from the third column of the input array.

Please use **python**, along with any libraries that you find useful (e.g. numpy, pandas) to complete this task. Whilst you can take any approach you see fit, a simple version of this might work like this:

- Load the array
- Convert each line described by the array into cartesian coordinates (y = mx + c)
- Split the array into two new arrays, one with negative gradient lines and one with positive gradient lines
- Find the **x-intercept** of each line
- Create a new array that contains the x-intercept difference for each line of positive gradient to each line of negative gradient

- Use this array to construct a dataframe that holds the 100 pairs that have the closest x-intercepts
- Save this dataframe to a file

Some considerations:

- There is no target solution that you will be marked against so use whichever design pattern you feel is best for the problem and maintains readability and clarity.
- This will be used by a python literate individual, so there is no need to create a graphical user interface. A typical use case would be to import the solution into a jupyter notebook and run it on an existing numpy array or to point it to a .npy file saved on disk.
- Whilst efficiency is important, this program will run in sequence with an experimental program that takes at least 1s to run, so there is no need to spend time producing a heavily optimised solution. There will be diminishing returns below ~10 ms run time.
- Small scale demonstrations of how code can be made robust will be helpful, but please don’t spend hours catching every possible edge case.