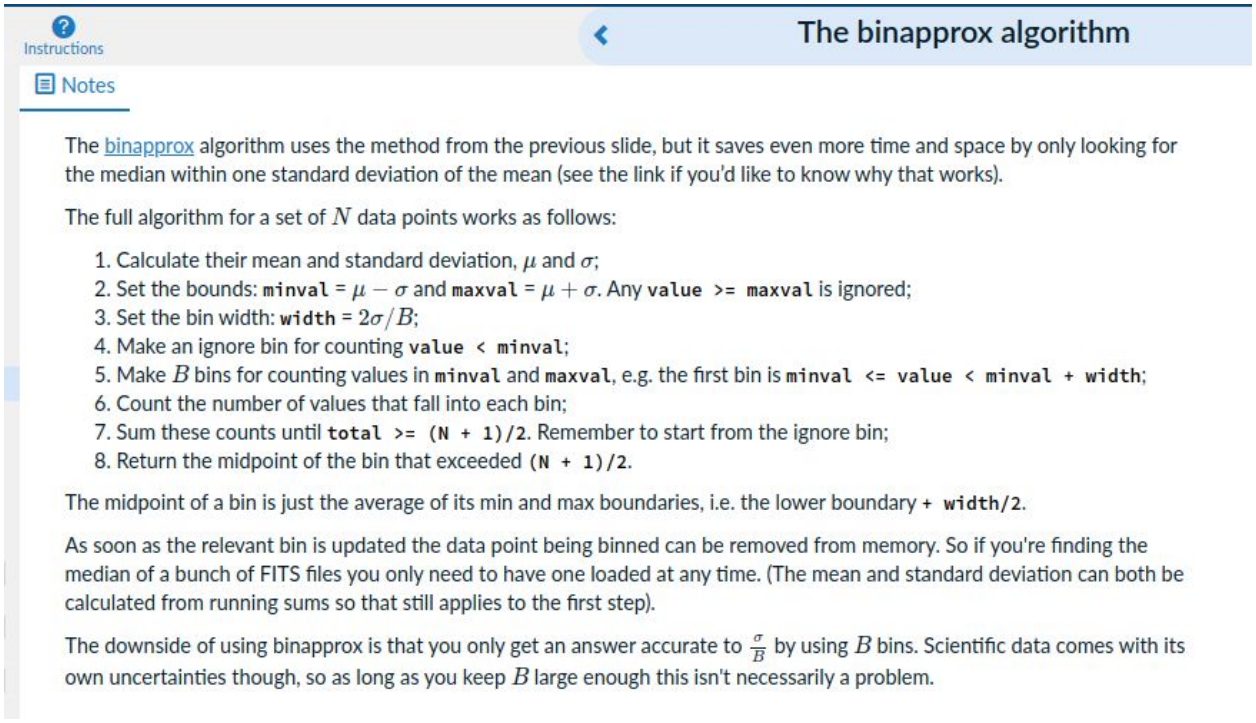
Introduction:

Link to binapprox research paper : here



Instructions

The binapprox algorithm

Notes

The binapprox algorithm uses the method from the previous slide, but it saves even more time and space by only looking for the median within one standard deviation of the mean (see the link if you'd like to know why that works).

The full algorithm for a set of N data points works as follows:

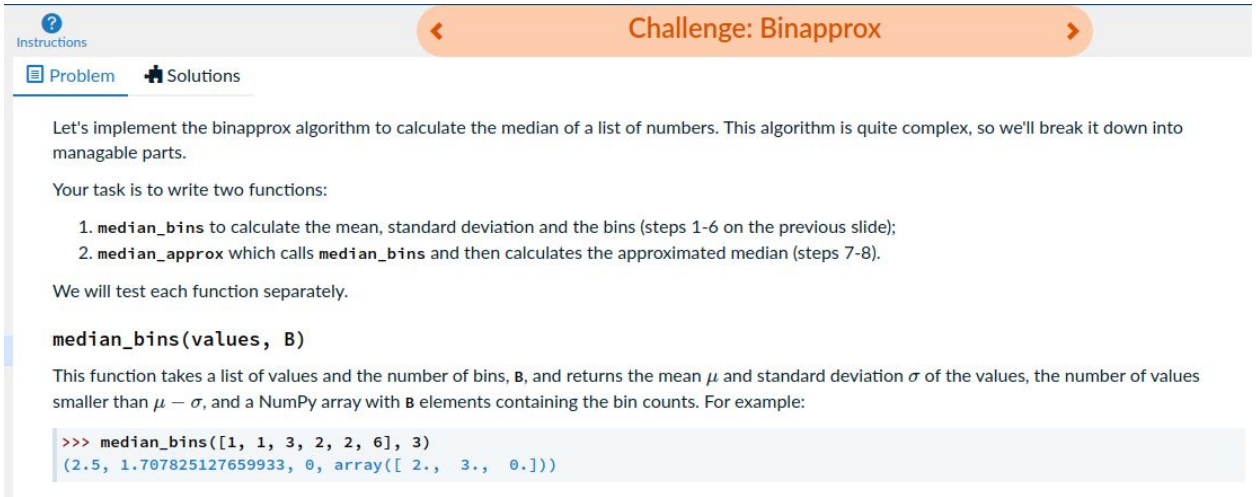
1. Calculate their mean and standard deviation, μ and σ;
2. Set the bounds: `minval` = $\mu - \sigma$ and `maxval` = $\mu + \sigma$. Any `value >= maxval` is ignored;
3. Set the bin width: `width` = $2\sigma/B$;
4. Make an ignore bin for counting `value < minval`;
5. Make B bins for counting values in `minval` and `maxval`, e.g. the first bin is `minval <= value < minval + width`;
6. Count the number of values that fall into each bin;
7. Sum these counts until `total >= (N + 1)/2`. Remember to start from the ignore bin;
8. Return the midpoint of the bin that exceeded `(N + 1)/2`.

The midpoint of a bin is just the average of its min and max boundaries, i.e. the lower boundary `+ width/2`.

As soon as the relevant bin is updated the data point being binned can be removed from memory. So if you're finding the median of a bunch of FITS files you only need to have one loaded at any time. (The mean and standard deviation can both be calculated from running sums so that still applies to the first step).

The downside of using binapprox is that you only get an answer accurate to $\frac{\sigma}{B}$ by using B bins. Scientific data comes with its own uncertainties though, so as long as you keep B large enough this isn't necessarily a problem.

Assignment



Instructions

Challenge: Binapprox

Problem Solutions

Let's implement the binapprox algorithm to calculate the median of a list of numbers. This algorithm is quite complex, so we'll break it down into managable parts.

Your task is to write two functions:

1. `median_bins` to calculate the mean, standard deviation and the bins (steps 1-6 on the previous slide);
2. `median_approx` which calls `median_bins` and then calculates the approximated median (steps 7-8).

We will test each function separately.

`median_bins(values, B)`

This function takes a list of values and the number of bins, `B`, and returns the mean μ and standard deviation σ of the values, the number of values smaller than $\mu - \sigma$, and a NumPy array with `B` elements containing the bin counts. For example:

```
>>> median_bins([1, 1, 3, 2, 2, 6], 3)
(2.5, 1.707825127659933, 0, array([ 2.,  3.,  0.]))
```

`median_approx(values, B)`

This function takes the same input as **`median_bins`**. It should return the approximate median using **`median_bins`** to calculate the bins. Using the same data as above, it should work like this:

```
>>> median_approx([1,1,3,2,2,6], 3)
2.5
```

To make sure your functions work for the general case, here's another example which uses four bins:

```
>>> median_bins([1, 5, 7, 7, 3, 6, 1, 1], 4)
(3.875, 2.521780125229002, 3, array([ 0.,  1.,  1.,  1.]))
>>> median_approx([1, 5, 7, 7, 3, 6, 1, 1], 4)
4.50544503130725
```

Here's an example in which binapprox is a bad approximation:

```
>>> median_bins([0, 1], 5)
(0.5, 0.5, 0, array([ 1.,  0.,  0.,  0., 0.]))
>>> median_approx([0, 1], 5)
0.90000000000000002
```