E8 spec sheet:
empiricalSampleBounds

Mission command preamble: As in general, we won't tell you *how* to do something. That is up to you and your creative problem solving skills. However, we will tell you what we would like this function to do. So here are the specs of the function we would like you to write.

Purpose: After resampling (drawing with replacement) from a sample, we often end up with a distribution. This is useful because it is quite common to then compare a null (or noise) distribution with a distribution that contains the signal, for instance in Signal Detection Theory, in order to construct an ROC curve. In order to do that, it is often necessary to determine the bounds of that empirical distribution – to assess what proportion of the sample is in the tails of the distribution (commonly used bounds are 50, 95 and 99).

Specific things we would like this function to do:

a) Take in 2 input arguments, in this order: 1) A variable that represents the dataset/distribution/sample.

b) The function should compute the upper bound (where the right tail starts) and the lower bound (where the left tail starts)

c) The function should return the values computed in b), either as two variables, or one variable with two elements.

d) Assumptions: The first input argument can be anything – a list, dataframe or numpy array containing real numbered values, but you can assume it to be a 1D numpy array with arbitrary length, by default. The second input argument should be a real number from 0.01 to 99.99 that determines the location of the bounds (where the tails start).

e) Make sure the function has a clear header as to what inputs the function assumes, what outputs it produces and when it was written.

Hints

(you can do whatever you want, and there is an infinite number of ways to do this that are all correct and valid, but some people find some suggestions helpful, so here they are. But this is just one possible approach):

1) Sort the input array/sample (the first input argument) in ascending order.
2) Determine the length of the array/sample (the first input argument).
3) Using this length (from 2), determine how many samples in the array correspond to one percentile.
4) Determine the probability mass contained in the tails. For instance, if the 2nd argument is 95, the total probability mass in the tails is 100-95 = 5. To determine the probability mass of each tail, further divide by 2, to yield 2.5. In this particular example.
5) Determine the percentile of the upper tail. In this case – if the 2nd argument is 95, this would be 100-2.5 = 97.5 Do the same for the lower tail, in this case as 0+2.5 = 2.5.
6) Use the numbers determined in 5), together with how many samples are in a percentile, to determine what index in the sorted sample corresponds to the 97.5th and 2.5th (in our example, but make sure it works for any 2nd input argument from 0.01 to 99.99) percentile. Make sure this is an integer (as the sample number is an integer), so you can round. But make sure to avoid off-by-one errors (as Python indexes from 0).
7) Use the indices you calculated in 6) to find the sample values that correspond to the upper bound and lower bound, cutting off the probability mass requested in the 2nd input argument.

Input / output examples:

Input: empiricalSampleBounds(sampleInput1.csv,95)

Output: lowerBound = -1.992, upperBound = 1.944

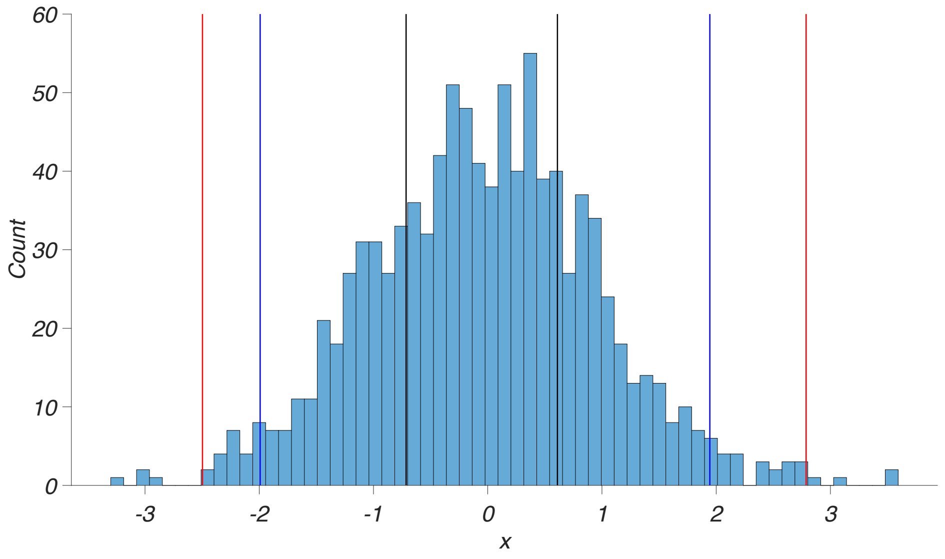
Input: empiricalSampleBounds(sampleInput1.csv,99)

Output: lowerBound = -2.497, upperBound = 2.787

Input: empiricalSampleBounds(sampleInput1.csv,50)

Output: lowerBound = -0.715, upperBound = 0.610

Below is a visualization of the situation (a histogram of the input sample, with the bounds drawn in as lines - black = 50, blue = 95, red = 99, so you can see what is going on. Your function doesn't have to do this).



Input: empiricalSampleBounds(sampleInput2.csv,95)

Output: lowerBound = 0.029, upperBound = 0.976

Input: empiricalSampleBounds(sampleInput2.csv,99)

Output: lowerBound = 0.0034, upperBound = 0.9952

Input: empiricalSampleBounds(sampleInput2.csv,50)

Output: lowerBound = 0.267, upperBound = 0.757