Introduction to Random Strings

Introduction to the problem, and background information

DNA is the backbone of all organisms, permitting them to produce proteins for growth, repair, and survival. Many different organisms share DNA sequences, or at the very least have similar sequences for particular gene products. This is an homage to the conservation of genetic code among organisms, and these similar motifs can be used to locate sequences of interest; sequences which do not occur by chance, but by having a true physiological function. An example of such a conserved protein would be insulin, the gene product of the beta-cells within the islets of Langerhans of the pancreas. Insulin is required for glucose homeostasis across all animals, and the DNA sequence encoding insulin is conserved among humans, pigs, cows, etc. Although there might be small regional differences between the different organisms, pig insulin is so closely related to that of human insulin that it was used for the treatment of diabetes in the early 20th Century.

How do these DNA motifs relate to the task at hand? The purpose of this task is to find the minimum length of DNA where on average you can say with a fair degree of certainty that the sequence is not formed by random chance (a random string), but it is formed instead by having a physiological function. In other words, we need to calculate the probability that a given motif is constructed by chance.

The shorter a DNA sequence is, the more likely it is that the sequence is randomly formed within the DNA. Whereas the longer the motif, the less likely it is formed by pure chance.

The task set out is given a DNA sequence, *s*, and GC-content probabilities (array A) of other strings, return the probability that a sequence constructed with the GC-content of array A will match the DNA sequence '*s'* in the form of a second array, array B.

Implementation

To implement, the user is asked to include the DNA sequence they are analysing, and an array of GC-contents.

Once the inputs have been made, the application will first count the number of adenine and thymine nucleotides, and group the integer value to the variable 'AT'. This process is repeated for guanine, and cytosine. An area of improvement here is to subtract the number of AT nucleotides from the total length of the sequence, which will return the value of GC without having to individually process the sequence again; although it will have a negligible effect on processing time, it is a nice way of getting the same result.

The equation which calculates the probability for array B is as follows:

*P = ( x / 2 )GC . ( (1-x) / 2 )AT*

Where *x* is the value of the GC-content, and *GC/ AT* is the number of corresponding nucleotides

This equation is run for each element within array A, and is implemented using a simple *for* loop which takes an element from the list, stores it as *x*, and parses it into the variables within the loop before moving along onto the next element.

Before the probability can be passed into array B, it must be the log value. This was the cause of a great debate for me – either I use the *math* library to be able to use the log command (Python does not come with a native way of calculating base-10 logs), or I could program my own one. I decided to use the math library due to it being computationally fast, and accurate. However, this gave me a challenge to program my own logarithm algorithm, to which there are many solutions available after a brief period of research.

Limitations, and Improvements

This program could do with an 'improvement' in the form of programming my own logarithm algorithm. It would provide more transparency in the code, and be a fascinating challenge.