Entropy calculation

Consider the following sequence $S = [1, 0, 0, 1, 0, 0, 1, 0, 0....]$ of length l, l times and putting it into a square matrix named M and E is a function that counts the number of 1s within a matrix or a binary string of desired length. Known as $np.sum()$ in python.

$$M = \begin{bmatrix} S_1 \\ S_2 \\ S_3 \\ ... \\ S_l \end{bmatrix}$$

Next, adopt an imperfect estimation using Shannon entropy. The Shannon entropy is defined as:

$$H = -\sum_{i=1}^{N}(P_i)log_2(P_i) \tag{1}$$

However, I prefer the alternative version of Shannon entropy explained by Josh Starmer which is:

$$H = \sum_{i=1}^{N}(P_i)log_2(\frac{1}{P_i}) \tag{2}$$

In plain English, Equation 2 says that for the associated probability, multiply it by its *surprise* and sum it up for all the probabilities.

To compute the entropy for each of the iterations of M_k. I would rearrange matrix M and flatten it to a 1-dimensional sequence. This sequence G can be written as:

$$G = [S_1, S_2...S_l]$$

The sequence G will almost look like a tape of 1s and 0s with length l^2. The next step is to partition the tape into 2-bit binary digits. This means that there will be $\frac{l^2}{2}$ partitions of 2-bit binary digits if l^2 were even. If l^2 is an odd number, the number of partitions of 2-bit binary digits is $\lfloor \frac{l^2}{2} \rfloor - 1$. Now, that the partitioning is done, we can then calculate the Shannon entropy for each of the 2-bit binary partitions.

While we can stop here, a specific situation could arise where if l^2 were an odd number and the digit at the last index of G is a 1, the algorithm I described would miss it entirely. Hence, modifications need to be made but anyways the computation of this simple algorithm is demonstrated in the table down below:

Parameter	1s	0s
Probability	$\frac{E}{2}$	$\frac{2-E}{2}$
Surprise	$\log_2 \frac{2}{E}$	$\log_2 \frac{2}{2-E}$

Noting that E is a function that just calculates how many 1s are within a defined binary string of arbitrary length or just $np.sum(S)$ in Python. The possible sequences of a 2-bit binary string are $(0, 0), (1, 0), (0, 1), (1, 1)$. For the situation of $(0, 0)$ the E within the above table is 0. This means that the Shannon entropy for the $(0, 0)$ string is:

$$H((0, 0)) = \frac{0}{2}\log_2 \frac{2}{0} + \frac{2}{2}\log_2 \frac{2}{2}$$

The first term is undefined while the 2nd term yields a 0. Getting an undefined result is OK within the context of Shannon entropy as the surprise of an event that never happens can be considered to be 0. Hence for the string $(0, 0)$ the Entropy associated with it is 0. This is also the case for the string $(1, 1)$ where the entropy also adds up to 0.

For the situation of the strings $(1, 0)$ and $(0, 1)$ the entropy calculation yields:

$$H((1,0)) = \frac{1}{2}\log_2\frac{2}{1} + \frac{1}{2}\log_2\frac{2}{1} \longrightarrow 1$$

The results for computing 4 different permutations of 2-bit binary strings are shown in the table below:

Bit	1	0
1	**False**	**True**
0	**True**	**False**

False (0) values are the result of having $(1, 1), (0, 0)$ pairs, and True (1) is returned when the pairs have different elements such as $(1, 0), (0, 1)$. This is the **XOR** gate.

Modification

I have described partitioning G to 2-bit digits. However, I decided to apply some parts of Crolls method [1] and not partition G into $\frac{l^2}{2}$ or $\lfloor\frac{l^2}{2}\rfloor - 1$. But instead apply a function towards G called XOR(G) which will result in a tape of length $l^2 - 1$.

For instance, the work by Grenville J. Croll [1] analyzed the entropy of a finite binary string. His algorithm is to compute the entropy of the original string, store that value and then take the XOR value of the original binary string of length l 2-bits at a time. The resulting 1st *derivative* of the string is a string of length $l - 1$. We can then compute the entropy of the 1st derivative and then store this value. Next, we apply the XOR function again to the first derivative resulting in the 2nd *derivative* and take its entropy. This algorithm stops until its last derivative is all 0s or 1s. Based on this information, we can then determine whether the original string contains periodic patterns. Finally, after we computed the entropy for each derivative we multiplied it with a weight w and sum them up to find the entropy of the original binary string.

While the above algorithm will give an accurate measurement of the entropy of a binary string of a flattened matrix M or G, it would be computationally demanding to implement it alongside the original simulation. This is especially true for large square matrices such as 200x200 ($l = 200$) with 40000 elements in matrix G. Hence for the simulation, I decided to opt for just applying the XOR function once which will result in a $l^2 - 1$ bit binary string.

Once the $l^2 - 1$ bit binary string is computed, the next step is to just sum up all the 1s of this string which reveals the number of unique occurrences of pairs of $(1, 0), (0, 1)$ within the original string. Depicting how this works is shown within Figure 1 down below:

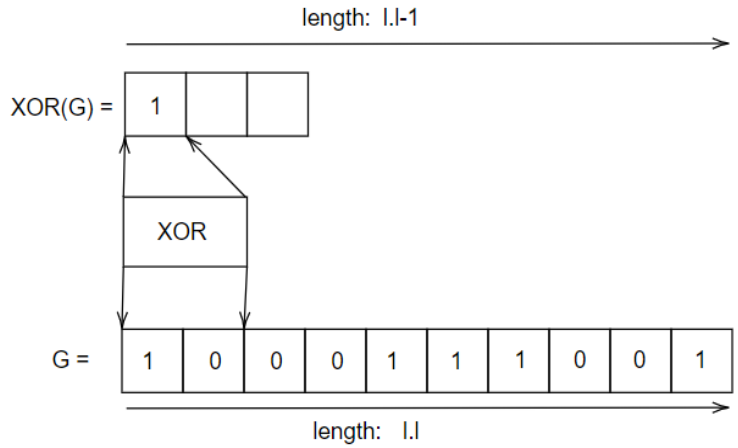

Figure 1: The XOR machine moves 1 square right per iteration.

Using the algorithm depicted above, it is actually possible to determine the upper bound sum of the number of 1s of XOR(G). Noting that I defined E as the number of 1s in matrix M and that E is a constant. The upper bound and lower bounds of this proxy entropy (H) are:

$$0 \leq H \leq 2E$$

The only way H can actually be equal to $2E$ is if only if the sequence G is in the form of $G = [0, 1, 0, 1, 0, 1....1, 0]$. A fully alternating binary string with the first and last index positions being a 0. In this scenario, the function XOR(G) will count unique occurrences of $(1, 0), (0, 1)$ twice and will double count the number of 1s in G.

The method that I described above can be criticized as being rather *caveman* like in its way of thinking and there are some major flaws that are:

- The algorithm fails when $E \geq 0.5l^2$.
- $G = [0, 1, 0, 1, 0, 1....1, 0]$ is not exactly disordered or entropic.

Noting that again E is just the number of 1s in the square matrix M, for a large square Matrix M where $l = 200$, I would want to limit E to be $E \leq 0.05l^2$. In this case, applying the game rules (encoded in the function F) to matrix M such that $M_{k+1} = F(M_k)$ the underlying grid will be more disordered if there are more unique occurrences of $(1, 0), (0, 1)$.

Bibliography

[1] G. J. Croll, "Bi-entropy: The approximate entropy of a finite binary string," *arXiv preprint arXiv:1305.0954*, 2013.