Sequence Analysis
MANUSCRIPT

William F. Barnes
[1]

October 9, 2024

Contents

Chapter 1

Sequence Analysis

1 Sequences

In the broadest sense, a *sequence* is any particular ordering of things or symbols. A sequence could be digits of a phone number, a list of strategic moves, the order of cards in a deck, etc.

1.1 Members and Elements

The N total constituents of a sequence are called *members*. The unique (non-repeated) members in a sequence are called *elements*.

Introducing notation to represent an example sequence, consider the statement

$$S_{16}\left(\phi_3\right) = (1\,2\,1\,1\,3\,2\,1\,3\,3\,2\,3\,1\,2\,1\,3\,1) \ .$$

On the right, we have an ordered list of digits 1, 2, 1, 1, etc., and there are $N = 16$ members in total. To represent this symbolically, on the left we first write S_{16}, which means 'a sequence with sixteen members'. Following this we represent the *element vector* ϕ_3, which for the example sequence means

$$\phi_3 = \{1, 2, 3\} \ .$$

The members of the sequence are each drawn from one list of three elements.

Individual Items

The jth item in a sequence containing N members is written

$$S_N^j\left(\phi\right) \ ,$$

and similarly the jth item in an element vector of size M is written

$$\phi_M^j \ .$$

Either of these could be used to represent the members of a sequence.

1.2 Element Regrouping

Have another look at the example sequence above, namely

$$1\,2\,1\,1\,3\,2\,1\,3\,3\,2\,3\,1\,2\,1\,3\,1\,,$$

and delete every second space to write

$$S_8\left(\phi_{3,2}\right) = (12\,11\,32\,13\,32\,31\,21\,31)\ .$$

Representing the same information as $S_{16}\left(\phi_3\right)$, we have a new sequence with $16/2 = 8$ members, and the element vector is no longer ϕ_3. In particular, the element vector becomes

$$\phi_{3,2} = \{11, 12, 13, 21, 22, 23, 31, 32, 33\}\ ,$$

which lists every possible pairing of the digits 1, 2, 3.

Order-Two Regrouping

In more symbolic terms, the move we just made amounts to the *order-two regrouping*

$$S_N\left(\phi_M\right) = S_{N/2}\left(\phi_{M,2}\right) + R_2\left(\phi_M\right)\ ,$$

where the sequence $S_{N/2}\left(\phi_{M,2}\right)$ reads

$$S_{N/2}\left(\phi_{M,2}\right) = \left(S^1_{N/2}\left(\phi_{M,2}\right)\ S^2_{N/2}\left(\phi_{M,2}\right)\ \cdots\ S^{N/2}_{N/2}\left(\phi_{M,2}\right)\right)\ ,$$

and the remainder term $R_2\left(\phi_M\right)$ is included for when N is not an even number.

Generalized Regrouping

A regrouped sequence contains the same information as the original while making the trade of having fewer members with a larger basis vector.

Continuing the example on hand, one may imagine turning the ϕ_3 basis vector into one containing $3^3 = 27$ elements:

$$\phi_{3,3} = (111, 112, 113, \ldots, 331, 332, 333)$$

With this, a sequence $S_N\left(\phi_M\right)$ regroups as

$$S_N\left(\phi_M\right) = S_{N/3}\left(\phi_{M,3}\right) + R_3\left(\phi_M\right)\ ,$$

where the remainder term contains the tail end of the original sequence, one or two digits in this case.

In the general case, one may perform an order-g regrouping, so long as g never exceeds N. To express this we write

$$S_N\left(\phi_M\right) = S_{N/g}\left(\phi_{M,g}\right) + R_g\left(\phi_M\right)\ , \qquad (1.1)$$

and the number of members in $R_g\left(\phi_M\right)$ is N modulo g.

1.3 Worked Example

To demonstrate the utility of element regrouping, and to motivate further analysis, consider the binary sequence

$$S_{24}\left(\phi_2\right) = (010011001110010011001110)\ .$$

One may argue, at a glance, that the sequence given has no discernible order, or was perhaps generated by a random process. We shall use element regrouping to find out.

Proceed by conceiving the basis vector

$$\phi_{2,2} = \{00, 01, 10, 11\}\ ,$$

or equivalently, just changing symbols,

$$\phi_{2,2} = \{A, B, C, D\}\ .$$

The given sequence becomes

$$S_{12}\left(\phi_{2,2}\right) = (BADADCBADADC)\ .$$

No reason to stop there, of course. One can now imagine pairs of elements such that

$$\phi_{2,4} = \{AA, AB, AC, \ldots, DB, DC, DD\}\ ,$$

or equivalently, switching to lowercase letters,

$$\phi_{2,4} = \{a, b, c, \ldots, n, o, p\}\ .$$

In terms of the vector $\phi_{2,4}$, the given sequence boils down to six symbols:

$$S_{24}\left(\phi_2\right) = S_6\left(\phi_{2,4}\right) = (e\ m\ o\ e\ m\ o)$$

From the above we see the original sequence is anything but random, as the regrouped sequence is two instances of the same string *emo*.

Further Analysis

It's worth going one more step. Consider yet another element vector that is an order-three regrouping of the previous one, namely

$$\psi_{16,3} = (aaa, aab, aac, \ldots, ppn, ppo, ppp)\ ,$$

having $16^3 = 4096$ elements, of which *emo* is just one of them, where for some unique k, we have

$$\psi^k_{16,3} = emo\,.$$

The equivalence

$$S_{24}\left(\phi_2\right) = S_2\left(\psi_{16,3}\right) = \left(\psi^k_{16,3}\ \psi^k_{16,3}\right)$$

tells us that either sequence - the original or the regrouped - embeds a probability

$$P = \frac{1}{4096}$$

that the second half is equal to the first half. Finally, we can rewrite the original sequence in a way that betrays its (now obvious) repetition:

$$S_{24}(\phi_2) = S_2(\phi_{2,12}) = (010011001110\,010011001110)$$

Relevance to Birthday Problem

There is a famous question in probability courses that asks 'if there are N people in the room, what is the probability $P(N)$ that any two people share a birthday?'

The solution to this problem is a fun exercise, and the answer happens to be

$$P(N) = 1 - \frac{365!}{365^N\,(365-N)!}\ ,$$

which makes use of the factorial (!) operator.

It just so happens that we just puzzled out the $N = 2$ case of the above, where the term 4096 replaces the number 365 in the formula.

For fun, suppose the example binary sequence was instead $N = 12 \times 3 = 36$ digits in length, consisting of three copies of the same sub-sequence, i.e.

$$S_{36}(\phi_2) = S_{12}(\psi_{16,3}) = \left(\psi^k_{16,3}\,\psi^k_{16,3}\,\psi^k_{16,3}\right)\ .$$

Like the $N = 2$ case, we can wonder about the chances of such a sequence occurring, which amounts to calculating

$$P(3) = 1 - \frac{4096!}{4096^3\,(4096-3)!}\ ,$$

resulting in a number slightly larger than $1/4096$.

One might wonder why $P(3)$ is not significantly smaller than $1/4096$, because there's a feeling that 'twins are more common than triplets'. Keep in mind though that $P(N)$ calculates whether *any* of the N members are the same, not an original specific member.

1.4 Element Translation

In any basis ϕ, suppose we have a sample sequence

$$S_7(\phi) = (A\,B\,C\,D\,E\,F\,G)\ .$$

One useful manipulation involves demoting the first member of the sequence (A) to the last position (after G), thereby 'promoting' all other members. This is called *element translation*, and to denote this we attach a superscripted digit on S as follows:

$${}^1S_7(\phi) = (B\,C\,D\,E\,F\,G\,A)\ .$$

The same sequence can be translated more than once by adjusting the new superscript:

$${}^2S_7(\phi) = (C\,D\,E\,F\,G\,A\,B)$$
$${}^3S_7(\phi) = (D\,E\,F\,G\,A\,B\,C)$$

Moreover, translations work in reverse when using a negative subscript:

$${}^{-1}S_7(\phi) = (G\,A\,B\,C\,D\,E\,F)$$
$${}^{-2}S_7(\phi) = (F\,G\,A\,B\,C\,D\,E)$$

In the general case, for any sequence (in shorthand notation)

$$S_N(\phi) = \left(S^1_N\,S^2_N\,\cdots S^N_N\right)\ ,$$

a translation of t members is represented by:

$${}^tS_N(\phi) = \left(S^{1+t}_N\,S^{2+t}_N\,\cdots\,S^N_N\,\cdots\,S^1_N\,S^2_N\,\cdots\,S^t_N\right) \tag{1.2}$$

Regrouping after Translation

Translation of a sequence has significant consequence when regrouping its members. Using the same sequence from the worked example above, take the $t = 1$ translation to write

$${}^1S_{24}(\phi_2) = \tilde{S}_{24}(\phi_2) = (100110011100100110011100)$$

In terms of the basis vector $\phi_{2,2}$, the above reads

$$\tilde{S}_{12}(\phi_{2,2}) = (CBCBDACBCBDA)\ ,$$

and furthermore using $\phi_{2,4}$ we have

$$\tilde{S}_6(\phi_{2,4}) = (j\,j\,m\,j\,j\,m)\ .$$

Notice right away that jjm is different from *emo* that corresponds to the non-translated sequence. Unsurprisingly though, the translated sequence still resolves to two copies of *something*, namely

$$\psi^q_{16,3} = jjm$$

for some unique q.

2 Random Number Analysis

Now we use sequence analysis to solve a 'real' problem. Namely, suppose you are given a sequence of 10^4 of integers in the range $[0, \ldots, 9]$ with the claim that the digits in the sequence are randomly distributed.

2.1 Histograms

One may visualize a given sequence using a *histogram*, which using our terms, is a 2D bar graph vertically representing the total occurrence count of each element in a given sequence.

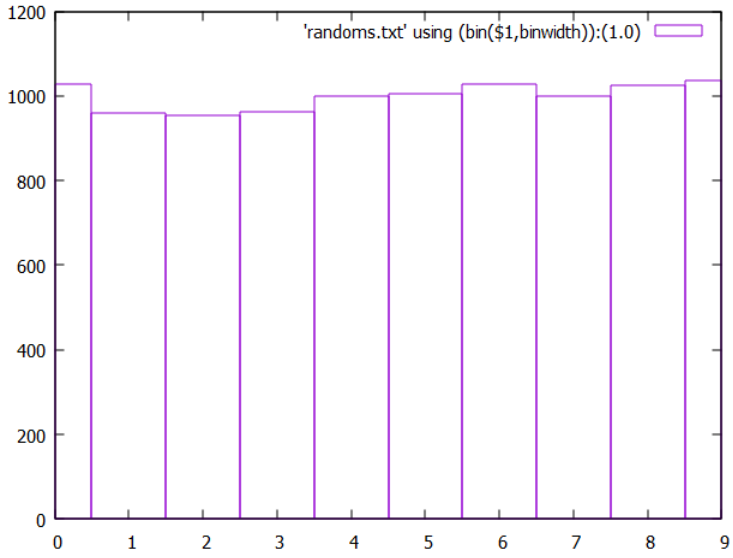

Figure 1.1: Histogram of $N = 10000$ random integers ranging 0 to 9.

If the incoming sequence is *truly* random, there should, at the very least, be no discernible structure visible in the corresponding histogram as shown in Figure 1.1. The horizontal axis lists the elements in the sequence, i.e. the integers 0 to 9. The vertical axis represents the number of occurrences of each integer, and we see an average of about 1000 per element. This makes sense, as the given sequence contains ten total elements and is ten thousand members large. The slight 'bumpiness' along the line through 1000 gives a sense of the standard deviation around this average.

Histograms can be blind to certain structures within the data they represent. For this case, we don't have a way of knowing, for instance, whether the pattern 123 occurs as often as 321. This, if it were the case, would alone would be enough to taint the randomness of the sample sequence while the histogram remains innocent. The sequence analysis methods we develop here are a remedy to this.

Since computer automation will be essential as we trudge forward, it's worth having a reliable system for manipulating large lists of data and generating histograms. To this end we employ the famous *gnuplot*, freely available for most workstations. The above histogram was generated using three `gnuplot` statements:

```
binwidth=1
bin(x,width)=width*floor(x/width)
plot 'randoms.txt' using (bin($1,binwidth)):(1.0) smooth freq with boxes
```

In `randoms.txt`, members of the sequence are separated by the 'newline' character.

2.2 Tainted Sample

Now, suppose we are handed a new sequence stored in `seq_10_0.txt` of comparable length to the above (at least 10^4 members). Running this sequence through the same `gnuplpt` program used above, we get a starkly different histogram as shown in Figure 1.2.

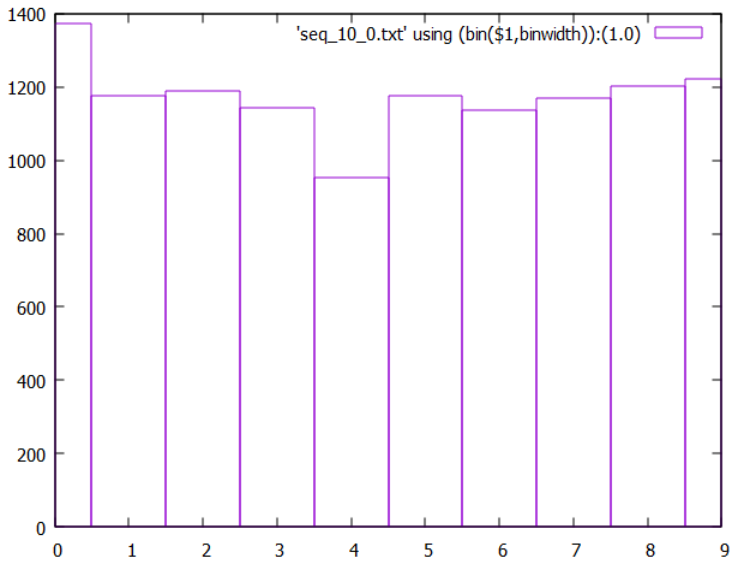

Figure 1.2: Histogram of $N \approx 11000$ non-random integers ranging 0 to 9.

Significant structure is visible in the histogram, thus the incoming sequence is tainted somehow. At minimum, we see a over-representation of the element 0 while 4 is strikingly under-represented. The remaining elements seem to share an average, at least qualitatively, but that's the best we can say so far.

2.3 Base-Ten Element Vectors

Now things get gritty. First, observe that the element vector behind any sequence containing the integers 0 to 9 is

$$\phi_{10} = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\} \ .$$

In the spirit of element regrouping, we can reimagine the incoming sequence as being half as long while consisting of members between the numbers 00 to 99, suggesting the element vector

$$\phi_{10,2} = \{00, 01, \ldots, 98, 99\} \ .$$

Of course, we can keep going with this to write

$$\phi_{10,3} = \{000, 001, \ldots, 998, 999\} \ ,$$

and so on for a general grouping number g as suggested in Equation (1.1).

Without knowing what causes the non-random quality of the given sequence, it is most prudent to include as many regrouping of the sequence as is reasonable. For the sake of demonstration, we choose a cutoff at $g = 8$ while including all values beneath this. Also, since the incoming sequence is sufficiently large, we ignore anything to do with remainder terms.

2.4 Translation and Regrouping

Starting with the incoming sequence $S^N(\phi_{10})$, suppose we produce the $t = 1$ translation ${}^1S^N(\phi_{10})$. This move has no effect whatsoever on the corresponding histogram, i.e. Figure 1.2 remains the same under this change.

This isn't always the case if we translate before regrouping. To illustrate, suppose the first six members of the incoming sequence are

$$S_N(\phi_{10}) = (0\,0\,6\,0\,4\,0\,\cdots) \ .$$

From this, the $t = 1$ translation reads

$${}^1S_N(\phi_{10}) = (0\,6\,0\,4\,0\,\cdots\,0) \ ,$$

and the $t = 2$ translation is

$${}^2S_N(\phi_{10}) = (6\,0\,4\,0\,\cdots\,0\,0) \ .$$

Looking closely, we see that $S^N(\phi_{10})$ and ${}^1S_N(\phi_{10})$ will yield different results under a $\phi_{10,2}$ regrouping. To convince yourself quickly, let $A = 00$, $B = 60$, $C = 40$ and see what happens to these after translation. By the same token though, ${}^2S^N(\phi_{10})$ does not yield new useful information, as it simply kicks 00 to the back of the line. The corresponding histogram would be blind to this order.

What we learn from this is, if we are concerned with the $g = 2$ regrouping of the incoming sequence, then only $S_N(\phi_{10})$ and ${}^1S_N(\phi_{10})$ need to be brought into consideration. This argument extends upward for increasing g.

That is, if we are concerned with the $g = 3$ regrouping of the incoming sequence, then $S_N(\phi_{10})$, ${}^1S_N(\phi_{10})$, ${}^2S_N(\phi_{10})$ are relevant, and there is no

need for ${}^3S_N(\phi_{10})$, for it just kicks the first member to the back.

An in-house script has been prepared to generate all of the translated and regrouped sequences required. Starting from the incoming sequence stored in `seq_10_0.txt`, the secondary data files required are suggested below.

- Order-two regrouping $\phi_{10,2}$ requires:
  - `seq_10_2_0.txt`
  - `seq_10_2_1.txt`
- Order-three regrouping $\phi_{10,3}$ requires:
  - `seq_10_3_0.txt`
  - `seq_10_3_1.txt`
  - `seq_10_3_2.txt`
- Order-g regrouping $\phi_{10,g}$ requires:
  - `seq_10_g_0.txt`
  - `seq_10_g_1.txt`
  - ...
  - `seq_10_g_g-1.txt`

As mentioned, and for no good reason, we cut the analysis after $g = 8$. This can be expanded as needed.

2.5 Stacked Histograms

Despite having different structure, regrouped translations of a sequence share an element vector, and thus have the same number of horizontal bins in a corresponding histogram. For this reason it makes sense to plot each regrouping on the same graph.

Double-Stacked Histogram

For the $g = 2$ case, we use the following `gnuplot` program:

```
set key off
binwidth=1
bin(x,width)=width*floor(x/width)
g=2
plot for[i = 0:g-1] "seq_10_".g."_".i.".txt"_
    using (bin($1,binwidth)):(1.0) smooth freq with boxes
```

The program uses a loop to stack histograms set by variable g that we'll tweak for other regroupings. For the $g = 2$ case, we get the resulting histogram shown in Figure 1.3.

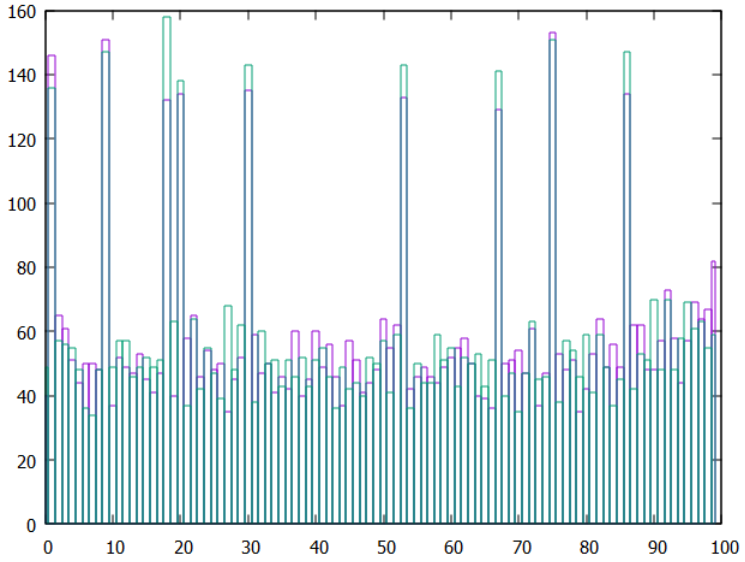

Figure 1.3: Stacked histogram containing the $g = 2$ regrouping.

The stacked histogram showing the regrouped sequences shows highly discernible structure. From lowest to greatest, there are peaks at

$$\omega_2 = (01, 09, 18, 20, 30, 53, 67, 75, 86) \ .$$

Interestingly, as hinted in the original histogram in Figure 1.2, the digit 4 is not involved in the above list, while the number 0 occurs more than once.

Triple-Stacked Histogram

The above analysis is now repeated with everything left in place except we now set $g = 3$, and the result is a triple-stacked histogram shown in Figure 1.4.

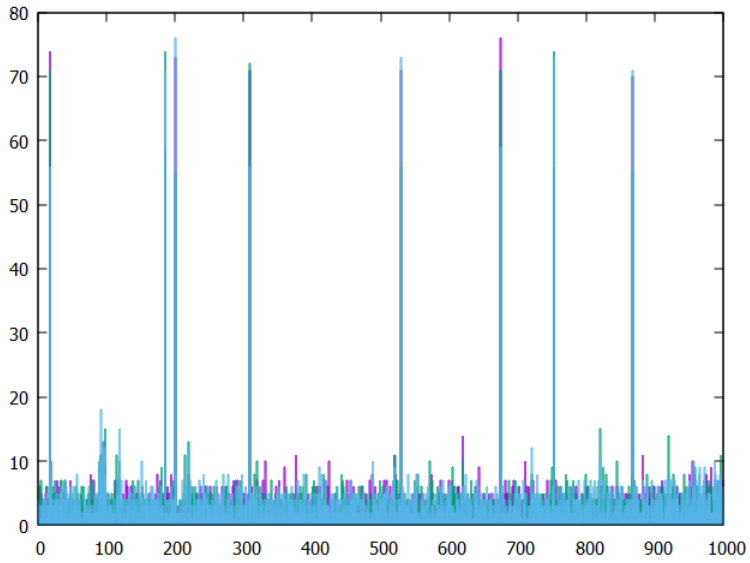

Figure 1.4: Stacked histogram containing the $g = 3$ regrouping.

With the triple-stacked histogram, the peaks are much higher and sharper than the background noise. Looking very closely (preferably in software at this point), we identify each peak:

$$\omega_3 = (018, 186, 201, 309, 530, 675, 753, 867) \ .$$

Quad-Stacked Histogram

Before the stacking game becomes too repetitive, let us hit the $g = 4$ case and count the number of peaks, which seems to decrease by one with every increase in g. Doing the now-familiar exercise leads to Figure 1.5.

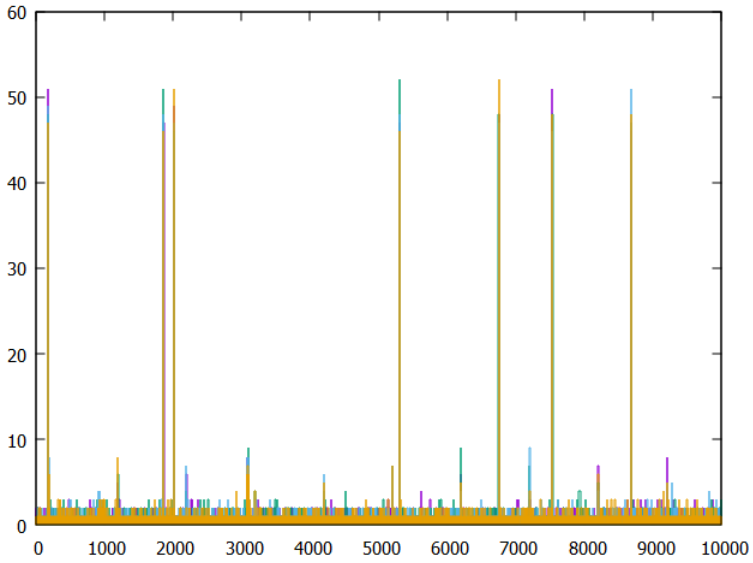

Figure 1.5: Stacked histogram containing the $g = 4$ regrouping.

Listing all peaks carefully, we find

$$\omega_4 = (0186, 1867, 2018, 5309, 6753, 7530, 8675) \ ,$$

totaling to 7. This is one fewer peaks than listed in ω_3, continuing the pattern of decreasing by one for every increase in g. Running this forward in imagination, one can estimate that somewhere around ω_{10} will there be just one peak in the histogram. Surely by then the culprit is found.

2.6 Higher Regroupings

With increasing g, the histogram becomes harder to discern at scale and it's better to use software to dig out the corresponding peaks. Showing only the results, one should find

$$\omega_5 = (01867, 18675, 20186, 67530, 75309, 86753)$$
$$\omega_6 = (018675, 186753, 201867, 675309, 867530)$$
$$\omega_7 = (0186753, 1867530, 2018675, 8675309)$$
$$\omega_8 = (01867530, 18675309, 20186753)$$

2.7 Results

In case it was not obvious yet, it seems that the (formerly famous) telephone number

$$2018675309$$

is embedded a handful of times in the otherwise random incoming sequence.

3 Alphanumeric Analysis

Here we tackle another example that is similar in nature but greater in scale than the previous one.

3.1 ASCII Basis

Consider a sequence consisting of capital letters, lowercase letters, numbers, punctuation marks, and other symbols visible on a standard U.S. keyboard. In particular, we take the ASCII characters within from 32 (a blank space) to 126 (the 'tilde' $\sim$ symbol). As an element vector, this amounts to

$$\phi_{95} = \{a, b, A, B, 1, 2, !, @, \dots\} \ ,$$

having 95 total elements.

3.2 Incoming Sequence

Now, suppose we are handed a file `seq_abc_0.txt` containing an ASCII sequence of $N \approx 10^5$ members claimed to be randomly chosen from ϕ_{95}. The incoming sequence could look like the following 100 characters indicative of the rest:

```
FCl9T8YSOW~q{rSFO'u%}+<t,B&a{,%HB39lO;5c
GWTeJ4J_ii2#tk\P-9eQ2.{'Pnz:x^op)JtY|{v|
CEn&Gw)6%/<OW'U(+P(y
```

The task is to determine if the sequence is truly random versus having embedded some kind of order, and if so, discovering its cause.

3.3 Data Preparation

Preparation for analysis begins by admitting that even modest regroupings $\phi_{95,2}$ with $95^2 = 9025$ elements, let alone $\phi_{95,3}$ with $95^3 = 857375$ elements, are unsuitable for viewing on a histogram. Foregoing visualization of any kind, instead we employ the classic Unix toolkit, namely those available in a *Bash shell*.

For the sake of efficiency, let us proceed using only the regrouping values $g = 12$, $g = 16$, $g = 20$, $g = 24$. If the incoming sequence contains N members, this means we need, in addition to ${}^{N}S\left(\phi_{95}\right)$:

$$
\begin{aligned}
&{}^{N/12}S\left(\phi_{95,12}\right)\\
&{}^{N/16}S\left(\phi_{95,16}\right)\\
&{}^{N/20}S\left(\phi_{95,20}\right)\\
&{}^{N/24}S\left(\phi_{95,24}\right)
\end{aligned}
$$

As with the previous case, an in-house script has been prepared to generate all of the translated and regrouped sequences required. For each g ivolved, secondary data files occur as follows:

- Order-g regrouping $\phi_{95,g}$ requires:
  - `seq_10_g_0.txt`
  - `seq_10_g_1.txt`
  - ...
  - `seq_10_g_g-1.txt`

3.4 Processing

To review, an incoming sequence is given in the form `seq_abc_0.txt` containing $\approx 10^5$ members. Translated regroupings of this sequence are generated and stored in separate files `seq_10_12_0.txt`, `seq_10_12_1.txt`, and so on.

What remains is to churn an answer from the incoming sequence and secondary data by preparing a script as follows:

main.sh

```
#!/bin/bash

# Combine like-width datasets.
for g in 12 16 20 24; do
    ./combine.sh $g
done

# Core analysis.
for filename in data/*.dat; do
    echo "working..."
    ./core.sh $filename
done

# Produce the result
./analyze.sh
```

In `main.sh`, the first salient line of code erects a 'for' loop that churns the values 12, 16, 20, 24. One at a time, these are sent as arguments to another script file `combine.sh`, detailed as follows:

combine.sh

```
#!/bin/bash

touch data/data.tmp

j=$1
for filename in data/seq_abc_"$j"_*.txt; do
    cat data/data.tmp $filename > data/tmpf && cat data/tmpf > data/data.tmp
done

rm data/tmpf

newfile=data/seq_abc_"$j"_all.dat
mv data/data.tmp $newfile
```

The script `combine.sh` requires all sequence files to occur in a directory called `data`. In analogy to building a stacked histogram, all files corresponding to the same element vector (same g), are joined and stored in a temporary file `data.tmp`. The temporary file is then named appropriately and stored with the `dat` extension. The above will lead to four `dat` files, as g takes on four values 12, 16, 20, 24.

Next, each `dat` file produced is sent to another script called `core.sh` as shown:

core.sh

```
#!/bin/bash

cp "$1" data_t.tmp

# Sort and count unique entries.
while read -r
do
    echo "$REPLY"
done < data_t.tmp > tmpf
sort tmpf | uniq -c | sort -nr > data_t.tmp

rm tmpf

mv data_t.tmp $(echo "$1" | cut -f 1 -d '.').res
```

The job of `core.sh`, somewhat analogous to analyzing a histogram, is to take an incoming dat file to tally and sort its members by occurrence count from top to bottom. The most-popular elements are reported along with their respective occurrence counts in a file with a similar name having extension `res`.

Finally, the script `analyze.sh` takes each res file and keeps only the top 20 records as detailed:

```
#!/bin/bash

for i in data/*.res; do
    awk 'NR>20{exit} {print $0}' $i
done > result.txt
```

The output of `analyze.sh` is stored in another file, `result.txt`.

3.5 Results

Executing `main.sh` delivers `results.txt`, which is the only output the user need interact with. However, it's enlightening to do the job manually using results written with the `res` extension. Sampling the top five records from each of these, we have:

seq_abc_12_all.res

```
13 ~pl@y m@ke$.
13 y m@ke$.j@ck
13 w0rk_&_n0~pl
13 rk_&_n0~pl@y
13 pl@y m@ke$.j
```

seq_abc_16_all.res

```
13 ~pl@y m@ke$.j@ck
13 y m@ke$.j@ck^@^D
13 w0rk_&_n0~pl@y m
13 rk_&_n0~pl@y m@k
13 pl@y m@ke$.j@ck^
```

seq_abc_20_all.res

```
13 ~pl@y m@ke$.j@ck^@^D
13 y m@ke$.j@ck^@^Dull
13 w0rk_&_n0~pl@y m@ke$
13 rk_&_n0~pl@y m@ke$.j
13 pl@y m@ke$.j@ck^@^Du
```

seq_abc_24_all.res

```
13 ~pl@y m@ke$.j@ck^@^Dull
13 w0rk_&_n0~pl@y m@ke$.j@c
13 rk_&_n0~pl@y m@ke$.j@ck^
13 pl@y m@ke$.j@ck^@^Dull b
13 n0~pl@y m@ke$.j@ck^@^Dul
```

Evidently, there is an ordered *something* that occurs 13 times in the original sequence. Since the list corresponding to $g = 24$ contains the same motifs as those with lower g-numbers, we proceed with the 24-case only.

Taking ten more records from the file `seq_abc_24_all.res` yields:

```
13 ll w0rk_&_n0~pl@y m@ke$.
13 l@y m@ke$.j@ck^@^Dull b0
13 l w0rk_&_n0~pl@y m@ke$.j
13 k_&_n0~pl@y m@ke$.j@ck^@
13 _n0~pl@y m@ke$.j@ck^@^Du
13 _&_n0~pl@y m@ke$.j@ck^@^
13 @y m@ke$.j@ck^@^Dull b0y
13 @ll w0rk_&_n0~pl@y m@ke$
13 0~pl@y m@ke$.j@ck^@^Dull
13 0rk_&_n0~pl@y m@ke$.j@ck
```

3.6 Final Analysis

While the final step could be automated, it's mostly harmless to work with the output of `seq_abc_24_all.res` directly to discern the hidden message in the original sequence. Playing a game akin to tree ring matching, we arrange the above information as follows:

```
              ~pl@y m@ke$.j@ck^@^Dull
     w0rk_&_n0~pl@y m@ke$.j@c
       rk_&_n0~pl@y m@ke$.j@ck^
              pl@y m@ke$.j@ck^@^Dull b
           n0~pl@y m@ke$.j@ck^@^Dul
  ll w0rk_&_n0~pl@y m@ke$.
               l@y m@ke$.j@ck^@^Dull b0
   l w0rk_&_n0~pl@y m@ke$.j
        k_&_n0~pl@y m@ke$.j@ck^@
          _n0~pl@y m@ke$.j@ck^@^Du
        _&_n0~pl@y m@ke$.j@ck^@^
               @y m@ke$.j@ck^@^Dull b0y
 @ll w0rk_&_n0~pl@y m@ke$
            0~pl@y m@ke$.j@ck^@^Dull
     0rk_&_n0~pl@y m@ke$.j@ck
----------------------------------------
@ll w0rk_&_n0~pl@y m@ke$.j@ck^@^Dull b0y
```

Finally, we unambiguously see the phrase 'All work and no play makes Jack a dull boy', heavily stylized, occurs in 13 locations throughout the incoming sequence.

4 Binary Sequence Prediction

Formally, a *binary* sequence is once that consists of just two symbols, whether they be 0/1, or Heads/Tails, etc. The world is full of binary sequences whether they occur naturally via *stochastic* processes, or artificially in telephone lines and computer circuits.

A stochastic process is usually physical in nature, and could be as simple as a game of coin flipping, or as disorderly as the drops falling from a rainstorm. While stochastic events may have deterministic causes, the mechanism responsible is never evident in the data collected from the system. In effect, then, sequences arising from stochastic processes are randomized.

Given the 'random' versus 'artificial' dichotomy in binary sequences, we shall pursue two goals using sequence analysis:

1. Develop a process for discerning whether a binary sequence is random versus artificial.
2. Develop a tool that predicts the next digit in a growing binary sequence given the previous digits.

Review

To bite into the issue, consider the example sequence

$$\begin{aligned} S_{48}\left(\phi_2\right) =& 101101001010101101100110 \\ & 101100010001110101011010 \, . \end{aligned}$$

As a binary sequence, we may invoke the the element vector

$$\begin{aligned} \phi_{2,2} &= \{00, 01, 10, 11\} \\ &= \{A, B, C, D\} \ , \end{aligned}$$

and the sequence is written

$$\begin{aligned} S_{24}\left(\phi_{2,2}\right) =& CDBACCCDBCBC \\ & CDABABDBBBCC \, . \end{aligned}$$

Going further with the basis vector

$$\begin{aligned} \phi_{2,4} &= \{0000, 0001, \ldots, 1110, 1111\} \\ &= \{AA, AB, \ldots, DC, DD\} \\ &= \{a, b, \ldots, o, p\} \ , \end{aligned}$$

the sequence is written

$$S_{12}\left(\phi_{2,4}\right) = (leklgglbbnfk) \ .$$

Hopefully the above rings familiar as review. One new case previously skipped is $\phi_{2,3}$, for which we have

$$\begin{aligned} \phi_{2,3} &= \{000, 001, \ldots, 110, 111\} \\ &= \{a, b, c, d, e, f, g, h\} \ , \end{aligned}$$

and the corresponding sequence is

$$S_{16}\left(\phi_{2,3}\right) = (ffbcffegfecbgfdc) \ .$$

4.1 Population Vector

Now we introduce the *population vector* ρ_M that, for a given sequence ${}^N S\left(\phi_M\right)$, tallies the total occurrences of each element in the sequence. The population vector is not a new concept - it's essentially a histogram converted to a line of data.

Illustrating with the example on hand, the sequence $S_{24}\left(\phi_2\right)$ listed above has

$$\rho_2 = \{23_0, 25_1\} \ ,$$

which means there are 23 zeros and 25 ones in the sequence. For the next regrouping, $S_{24}\left(\phi_{2,2}\right)$ is characterized by

$$\rho_{2,2} = \{3_A, 8_B, 9_C, 4_D\} \ .$$

Going further, the sequence $S_8\left(\phi_{2,3}\right)$ is characterized by

$$\rho_{2,3} = \{0_a, 2_b, 3_c, 1_d, 2_e, 6_f, 2_g, 0_h\} \ .$$

Often, it's more useful to organize ρ with the most popular entries first, which for the previous example means

$$\rho_{2,3} = \{6_f, 3_c, 2_e, 2_g, 1_b, 1_d, 0_a, 0_h\} \ .$$

Sorting by occurrence count, we can express more lengthy population vectors in abbreviated form, for instance

$$\rho_{2,4} = \{3_l, 2_b, 2_g, 2_k, 1_e, 1_f, 1_n, 0_a, 0_g, \dots\} \ .$$

Population after Translation

In light of element translation, it would make sense that a population vector $\rho_{x,y}$ will be different depending on the translation applied. To track this in the spirit of Equation (1.2), we include a superscript before the ρ symbol i.e. ${}^t\rho_{x,y}$. Illustrating using the running example, we have, for the $t = 1$ translations of the sequence:

$$\begin{aligned} {}^1S_{48}\left(\phi_2\right) =& 011010010101011011001101 \\ & 011000100011101010110101 \end{aligned}$$

$$\begin{aligned} {}^1S_{24}\left(\phi_{2,2}\right) =& BCCBBBBCDADB \\ & BCACADCCCDBB \end{aligned}$$

$${}^1S_{16}\left(\phi_{2,3}\right) = (dccfddbfdaedfcgf)$$

$${}^1S_{12}\left(\phi_{2,4}\right) = (gjfgmngcdklf)$$

From these, we write the translated population vectors (the ρ_2 case remains the same). While sharing some resemblance with the non-translated population vectors, the fine details are varied from the originals:

$$\begin{aligned} {}^1\rho_2 &= \{23_0, 25_1\} \\ {}^1\rho_{2,2} &= \{3_A, 9_B, 8_C, 4_D\} \\ {}^1\rho_{2,3} &= \{5_d, 4_f, 3_c, 1_a, 1_b, 1_e, 1_g, 0_h\} \\ {}^1\rho_{2,4} &= \{3_g, 2_f, 1_c, 1_d, 1_i, 1_k, 1_l, 1_m, 1_n, 0_a, \dots\} \ . \end{aligned}$$

Grand Population Vector

The population vector allows us to symbolize a process used twice previously, particularly in (i) histogram stacking, and (ii) creation of a `res` file.

For a given regrouping number g, we reasoned that the translation number t, starting from $t = 0$, should never exceed $t = g - 1$. In other words, a sequence $S_N\left(\phi_M\right)$ regrouped as $S_{N/g}\left(\phi_{M,G}\right)$ means we must also bring into consideration:

$$\begin{gathered} {}^1S_N\left(\phi_{M,g}\right) \\ {}^2S_N\left(\phi_{M,g}\right) \\ \dots \\ {}^{g-1}S_N\left(\phi_{M,g}\right) \end{gathered}$$

Meanwhile, each of the above implies a unique population vector ${}^t\rho_{M,g}$, namely:

$$\begin{gathered} {}^1\rho_{M,g} \\ {}^2\rho_{M,g} \\ \dots \\ {}^{g-1}\rho_{M,g} \end{gathered}$$

From a given $\phi_{M,g}$, we define the *grand population vector* to be

$$\sigma_{M,g} = \sum_{t=0}^{g-1} \left({}^t\rho_{M,g}\right) \ .$$

If the vectors ${}^t\rho_{M,g}$ are *not* sorted from greatest to least occurrence, then $\sigma_{M,g}$ corresponds to a g-stacked histogram. If, on the other hand the vectors ${}^t\rho_{M,g}$ are arranged from greatest occurrence to the least, then $\sigma_{M,g}$ corresponds to what we saw in any `res` file in previous work.

To demonstrate, consider the population vectors $\rho_{2,2}$, ${}^1\rho_{2,2}$ written above, namely

$$\begin{aligned} \rho_{2,2} &= \{3_A, 8_B, 9_C, 4_D\} \\ {}^1\rho_{2,2} &= \{3_A, 9_B, 8_C, 4_D\} \ , \end{aligned}$$

that correspond to $S_{24}\left(\phi_{2,2}\right)$, ${}^1S_{24}\left(\phi_{2,2}\right)$ respectively. Summing across like elements, we easily write the grand population vector for this case:

$$\sigma_{2,2} = \{6_A, 17_B, 17_C, 8_D\}$$

A more illustrative case explores the vectors $\rho_{2,3}$, ${}^1\rho_{2,2}$, ${}^2\rho_{2,3}$, which requires knowing $S_{16}\left(\phi_{2,3}\right)$, ${}^1S_{16}\left(\phi_{2,3}\right)$, ${}^2S_{16}\left(\phi_{2,3}\right)$. Two of these we have written already, and the third is

$${}^2S_{16}\left(\phi_{2,3}\right) = (gefcggdcgbahcffc) \ .$$

From these, we derive three population vectors (two are review):

$$\rho_{2,3} = \{0_a, 2_b, 3_c, 1_d, 2_e, 6_f, 2_g, 0_h\}$$
$${}^1\rho_{2,3} = \{1_a, 1_b, 3_c, 5_d, 1_e, 4_f, 1_g, 0_h\}$$
$${}^2\rho_{2,3} = \{1_a, 1_b, 4_c, 1_d, 1_e, 3_f, 4_g, 1_h\}$$

Summing by column, the grand population vector is

$$\sigma_{2,3} = \{2_a, 4_b, 10_c, 7_d, 4_e, 13_f, 8_g, 1_h\} \ ,$$

or equivalently,

$$\sigma_{2,3} = \{13_f, 10_c, 8_g, 7_d, 4_b, 4_e, 2_a, 1_h\} \ .$$

We conclude that motifs like $f = 101$ and $c = 010$ are much more common in the original sequence than things like $a = 000$ and $h = 111$.

4.2 Partial Guess

Now we're at the heart of the problem. Keeping the same working example, consider again the sequence

$$S_{48}(\phi_2) = 101101001010101101100110$$
$$10110001000111010101 1010 \, .$$

If such a sequence were to grow by adding one digit on the right, thereby becoming, $S_{49}(\phi_2)$, can we predict what the digit will be?

The answer is reliably 'no' if the sequence is truly random. However, if there are prevailing patterns in the sequence, it should be possible to detect them, and then to look at the rightmost digits and guess intelligently what digit would continue the sequence.

Partial Guess

The 'final guess' X of the new digit is calculated using the original sequence and its various regroupings. For each regrouping $S_{N/g}(\phi_{M,g})$ of the sequence, there is a notion of *partial guess*, x_g. The ensemble of partial guesses constitutes the final guess. Symbolically, this means

$$S_{N+1}(\phi) = (S_N(\phi) \ X) \ , \tag{1.3}$$

and X depends on each x_g in a way we'll cover soon:

$$X = f(x_1, x_2, x_3, \ldots, x_g)$$

Worked Example

Digging into the sequence on hand, and starting with the trivial regrouping $g = 1$ (no regrouping at all), we found

$$\sigma_2 = \rho_2 = \{23_0, 25_1\} \ ,$$

in which there is no sense of pattern beyond there being slightly more ones in the sequence than there are zeros. With this, we write

$$x_1 = 1 \, .$$

For $g = 2$, look at the end of the original, non-translated sequence, $S_{48}(\phi_2)$. We reason the last digit in the sequence, namely 0, could be the first digit of some element in $\phi_{2,2}$ whose second digit is x_2. Given the choices 00, 01, 10, 11, we see any element starting with 1 is ruled out, and we must decide between 00 and 01. The judge of the matter shall be the grand population vector $\sigma_{2,2}$, namely

$$\sigma_{2,2} = \{6_A, 17_B, 17_8, 4_D\} = \{6_{00}, 17_{01}, 17_{10}, 8_{11}\} \ .$$

The ratio 6:17 says the pattern 00 is less likely to occur than 01, thus the sequence ought to end with the pattern 01, and we have the partial guess

$$x_2 = 1 \, .$$

For $g = 3$, consider the last two digits in the original sequence, namely 10. We reason that these could be the first two digits in some element from $\phi_{2,3}$ whose third digit is x_3. There are eight elements for $g = 3$, but the items we need begin with 10, and these are 100, 101. The population grand vector $\sigma_{2,3}$ for this situation was found to be:

$$\sigma_{2,3} = \{13_f, 10_c, 8_g, \ldots\} = \{13_{101}, 10_{010}, 6_{110}, \ldots\}$$

The pattern 101 is more common in the sequence than is 100, so we have

$$x_3 = 1 \, .$$

For $g = 4$, we take the last three digits in the original sequence, namely 010. We reason that these could be the first three digits in some element from $\phi_{2,4}$ whose fourth digit is x_4. There are sixteen elements for $g = 4$, but only $e = 0100$, $f = 0101$ start with 010. The partial guess x_4 is determined by whichever of e or f is most common in the sequence, i.e. which has a higher value in $\sigma_{2,4}$.

For brevity we'll stop the manual analysis here. In practice, it's useful to extend beyond $g = 4$, best left to something automated up to some cutoff $g_{\max}$.

Undecided Cases

In the event that the contenders for a partial guess x_g have the same population number, then assigning $x_g = 0$ or $x_g = 1$ doesn't quite do justice. In such a case, we instead let

$$x_g = \frac{1}{2} \, ,$$

the average of the two.

Randomness

If it turns out that most (or all) of the partial guess values are undecided, it could be that (i) an insufficient range of g-values are being tested, or (ii) the sequence is 'random' as far as we can tell.

4.3 Final Guess

With all partial guess values $\{x_g\}$ in hand, the job is to assemble the final guess first symbolized in Equation (1.3). For this, we choose a weighted average scheme to combine partial guesses, namely

$$Y = \frac{1}{T}\sum_{g=1}^{g_{\max}} w_g x_g \,. \tag{1.4}$$

The unknown terms w_g are called *weights*, obeying normalization condition

$$T = \sum_{g=1}^{g_{\max}} w_g \,.$$

A given weight w_g tells much how much the corresponding partial guess x_g contributes to the final guess X.

The average Y is most generally a decimal value somewhere between 0 and 1. To finish the job, we must round Y to either 0 or 1, giving the final guess:

$$X = \text{Round}\,(Y) = \begin{cases} 0 & \text{if } Y < .5 \\ 1 & \text{if } Y \geq .5 \end{cases} \tag{1.5}$$

In this sense we've transformed the question of how to predict the next digit in a growing sequence into q question of choosing a proper right hand side for Y.

Weighting Models

A specific choice of coefficients $\{w_g\}$ is a *weighting model*, easily represented as a vector

$$W_q = (w_1, w_2, \ldots, w_{g_{\max}}) \,.$$

The subscript q is unique to the specific choices w_1/T, w_2/T, etc. Only one weighting model is required to calculate the final guess $X\,(Y)$, but choosing the right model is an art unto its own.

4.4 Guess-Reveal Algorithm

Reviewing binary sequence prediction in one sentence, we take an incoming binary sequence $S_N\,(\phi_2)$ and, based on any discernible order within the sequence, calculate the form of $S_{N+1}\,(\phi_2)$ given a weight model W_q.

Given an incoming sequence, consider now a truncated version

$$S_{n<N}\,(\phi_2) = \left(S^1\ S^2\ \ldots\ S^n\right)$$

whose members match the first n members of the original sequence. Now devise a 'game' where, given the truncated sequence, predict the next digit using a fixed model W_q. With the guess on hand, reveal the next digit in the actual sequence and compare the guess to reality. From this we keep score in the form of win/loss ratio, winning streak, etc. The longer the incoming sequence, the more data we generate.

When the sequence is finished, i.e. when the guessing game has been played on each truncation of the incoming sequence, the model's performance is summarized.

4.5 Model Space

While there are technically an infinite number of weighting models one could sift through, we tame the problem by having each w_g be limited to 0 or 1, i.e. a 'binary' weighting model. With this, every W_q-vector takes the form of a binary sequence, and for convenience we convert the sequence to base ten and assign that number to q:

$$\begin{aligned} W_0 &= (0\,0\,0\,0\,\cdots) \\ W_1 &= (1\,0\,0\,0\,\cdots) \\ W_2 &= (0\,1\,0\,0\,\cdots) \\ W_3 &= (1\,1\,0\,0\,\cdots) \\ W_4 &= (0\,0\,1\,0\,\cdots) \\ &\cdots \end{aligned}$$

The total number of weighting models is determined by $g_{\max}$, and for the sake of demonstration we choose $g_{\max} = 10$. This invokes a total of $2^{10} = 1024$ models in total, with the last being

$$W_{1023} = (1\,1\,1\,1\,1\,1\,1\,1\,1\,1) \,.$$

The term *model space* refers to the set of all models $\{W_q\}$ that could be used to calculate $X\,(Y)$, where for completeness, Y is given by Equation (1.4), and X is given by Equation (1.3).

By choosing a binary model system, the infinitude of available models reduces to $2^{g_{\max}}$ discrete points with the argument that enough of model space is covered. Moreover, a systematic stepping through model space is as easy as moving among numbers from 0 to $g_{\max}$ in binary.

4.6 Automation

All of the above-described processes can be abstracted and packaged into a suitable program that operates on any incoming binary sequence. For this study, a *QB64* source file called `Binary-Analyzer.BAS`, which compiles to `Binary-Analyzer.exe`, has been prepared to do so. All exact results will come from this one program.

A precise implementation of the so-called 'automated guess-reveal algorithm' is attached in the aforementioned `BAS` file.

Model Hunt Mode

For a given sequence $S_N(\phi_2)$, the program must (i) open an 'outer loop' for stepping through model space (in total or in part), and also (ii) open an 'inner loop' that steps through each truncation of the given sequence.

Once we're two loops deep, the guess-reveal algorithm gets to play. That is, a 'final' guess is calculated for whichever truncation we're on, i.e.

$$S_{n+1}(\phi) = (S_n(\phi)\ X)\ ,$$

and then X is compared with the *actual* $S_{n+1}(\phi)$ as it's given. From here we gather whether the model was right or wrong, along with other statistics, and the final output should be a single model number q that best matches the sequence.

A key output from so-called 'model hunt mode' is the winning model's *correctness rating*, denoted C_R, which a simple ratio of the number of times the guess-reveal algorithm yields correct results throughout the total number of attempts:

$$C_q = \frac{\text{number correct}}{\text{total attempts}} \tag{1.6}$$

Oracle Mode

A second running mode can also be prepared for interaction with a human (or similar stochastic system) in real time. In particular, the user is asked to manually input the ones and zeros of a handmade binary sequence, and the guess-reveal algorithm is applied and reported with each keystroke.

Like an oracle, the computer should be able to attain a higher correctness rating in real time than would be achieved by randomly guessing.

5 Binary Case Studies

5.1 Random Binary Sequences

Suppose now that we take a binary sequence known to be random. Like a fair coin-tossing game, we know that there should, on average, be an equal number of of heads and tales as the game goes on. For N total trials of such a system, we can write the *standard deviation*

$$\sigma_N = \frac{1}{2}\sqrt{N}\ ,$$

giving a sense of what's reasonable for off-average trends.

For example, in a case of $N = 100$ trials, i.e. a binary sequence of length 100, we write $\sigma_{100} = \sqrt{100}/2 = 5$. That is, after a game of 100 tosses, we can often expect off-average outcomes to be biased only by about 5 units. Or, taking $N = 1000$, the standard deviation comes out to approximately 16.

Introducing another definition, the so-called *coefficient of variation*, denoted C_V, divides σ_N by the sample size:

$$C_V = \frac{\sigma_N}{N} = \frac{1}{2\sqrt{N}}$$

That is, for $N = 100$ we have $C_V = 5\%$, and for $n = 1000$ we have $C_V \approx 2\%$.

If a sequence is to be classified as 'random', then the correctness rating C_q and C_V relate by

$$C_q \approx 50\% + C_V\ .$$

Just to be sure, let's use $C_q > 55\%$ as a clear indicator of order in a sequence that is at or more than 100 members in length.

QB64 RND Function

A binary sequence of $N = 2000$ digits has been prepared using the `RND` function built into QB64. Running this sequence through model hunt mode with q running from 0 to 1023 yields the following information:

- The best model is $W_{66} = (100010\ldots)$, rated 52%, with a longest streak of 13.
- Comparable to best-performing are W_{71}, W_{83}, W_{85}, W_{130}.
- The worst model is $W_1 = (100000\ldots)$, rated 48%.

With the best and worst correctness ratings being 52% and 48% respectively, it's reasonable to conclude

that sequence analysis finds no discernible order in the 2000-digit binary sequence generated by QB64. (A relief for both parties.)

Wolfram Rule 30

In the 1980s, researcher Stephen Wolfram was studying cellular automation using sets of rules encoded in binary, and was surprised by the result of so-called Rule 30, depicted in Figure 1.6.

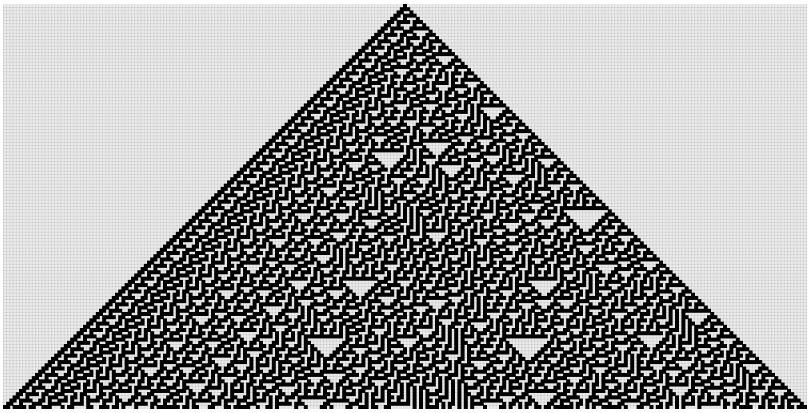

Figure 1.6: Wolfram's Rule 30.

Wolfram contends that the central column of alternating black/white squares constitutes a random binary sequence. Using an in-house utility to generate and extract $N = 2000$ digits down the middle column, we apply sequence analysis to find:

- The best model is $W_2 = (010000\ldots)$, rated 51%, with a longest streak of 11.
- Comparable to best-performing are W_2, W_4, W_5, W_7, W_{14}, W_{22}, W_{66}, W_{68}, W_{70}, W_{195}, W_{262}, W_{323}, W_{387}.
- The worst model is W_{416}, rated 46%.

There may or may not be a prize (still) available for anyone who can prove that the central column in the Rule 30 image is not a random sequence. One way or the other, we find no order by sequence analysis.

Stock Markets

It is widely claimed that the world stock market is generally unpredictable, so this ought to be evident by sequence analysis (or get rich if not). For a source of data, it turns out that *Yahoo Finance* freely avails historical information for participating entities. A query was made for `DIS`, short for 'Disney', which returned a table of opening and closing prices for business dates spanning 08/18/2022 to 08/18/2023.

Converting opening and closing prices to binary data goes as follows: if today's price is higher than yesterday's price, write 1. Otherwise, write 0. Doing precisely this for the `DIS` data produces a sequence with 252 members. For completeness, this sequence reads:

$$\begin{aligned} S_{252}\left(\phi_2\right) =& 000011010010011110100100 \\ & 000010011100001101001101 \\ & 101100000001100000001100 \\ & 111000111000001011001101 \\ & 111111100111111101110010 \\ & 000110101110001100100101 \\ & 101101010101001110010110 \\ & 110001100001100001011010 \\ & 000011101010100100011110 \\ & 111011011000110100101100 \\ & 011101000001 \end{aligned}$$

Running the above through the sequence analysis meat grinder leads to the following summary:

- The best model is W_{546}, rated 55%, with a longest streak of 7.
- Comparable to best-performing are W_{562}, W_{564}, W_{630}.
- The worst model is W_8, rated 47%.

In other words, a stock's rising and falling is as random as it gets.

5.2 Percussive Sequences

A *percussive* binary sequence is one that has obvious regularity, as if generated with intent by a drummer. In light of sequence regrouping, percussive sequences are easily spotted as one or a few repeated elements from some element vector $\phi_{2,g}$.

A percussive sequence may embed several aspects of repetition, and the sequence as a whole eventually repeats itself. The period of repetition, i.e. the number of digits that pass by before the sequence repeats, is called the *period*. To denote a percussive binary sequence we'll use the nomenclature $P_{T,N}$ with P replacing the letter S, and the subscript also carries the period number T.

Uniqueness

Only two period-1 sequences are possible in binary, and these are:

$$P_{1,N}(\phi_2) = 000000000000000000000000000000\ldots$$
$$P_{1,N}(\phi_2) = 111111111111111111111111111111\ldots$$

As for $P_{2,N}$, there are four total percussive sequences, but two of them are redundant to the above and are excluded:

$$P_{2,N}(\phi_2) = 010101010101010101010101010101\ldots$$
$$P_{2,N}(\phi_2) = 101010101010101010101010101010\ldots$$

However, note the pair of sequences $P_{2,N}(\phi_2)$ are identical up to a a $t = 1$ translation. A proper sequence analysis would treat each version roughly the same.

By similar reasoning, there two unique percussive sequences explicit to period three:

$$P_{3,N}(\phi_2) = 011011011011011011011011011011\ldots$$
$$P_{3,N}(\phi_2) = 001001001001001001001001001001\ldots$$

Seed

Percussive sequences can be expressed more concisely by only stating the minimum number of digits needed to reproduce the sequence. The span of digits is called the *seed*, denoted D_T. The length of the seed is exactly one period T, and the seed can be sampled from anywhere in the sequence. For this reason, translations of a seed are still considered the same seed.

Listing the seed for the above cases plus one more, we have

$$D_1 = \{0, 1\}$$
$$D_2 = \{01\}$$
$$D_3 = \{001, 011\}$$
$$D_4 = \{0001, 0011, 0111\}\ .$$

For the D_5 seeds, we not only have

$$D_5 = \{00001, 00011, 00111, 01111, \ldots\}\ ,$$

but also the concatenations of D_2 with D_3, namely

$$00101, 01011\ .$$

Convergence Time

Percussive sequences with a period $T \leq g_{\max}$ are picked up swiftly by sequence analysis, and all nontrivial models in $\{W_q\}$ (not all zeros or not all ones) will lead the guess-reveal algorithm toward a perfect correctness rating.

Given a percussive sequence $S_N(\phi_2)$ with any model W_q, define the *convergence time* τ_q as a count of how many iterations, i.e. how many guess-and-reveal rounds it takes for the sequence analyzer to completely stop making mistakes. This is a rather high bar for any prediction tool, but for low-period percussive sequences we can almost always shoot for 100%.

For long-period cases, i.e. $T \geq g_{\max}$, the sequence still exhibits plenty of order, yielding a high correctness rating, however convergence time may not be evident.

Measurement: D_5

Consider a percussive binary sequence $S_{240}(\phi_2)$ that is based on a D_5 seed, particularly

$$S_{240}(\phi_2) = (01011\cdots)$$

with 240 total members. Running the sequence through so-called model hunt mode in the sequence analyzer gives information on the performance on each model W_q tested, where q runs from 0 to 1023.

Reporting results in a way analogous to the random sequence study above, we have:

- The best model is W_{16}, rated 97%, with a longest streak of 226.
- Comparable to best-performing are W_{17}, W_{19}, W_{21}, W_{49}, W_{50}, W_{51}, W_{53}, W_{56}, W_{57}, W_{59}, W_{61}, W_{65}, W_{67}, W_{80} to W_{85}, W_{87}, W_{88}, W_{89}, W_{91}, W_{93}, W_{96}, W_{98} and plenty more.
- The worst model is W_1, rated 58%.

Evidently, it's hard to *not* find the clear order in the given sequence, as even the worst model performs well above 50%. The best model W_{16} is able to correctly predict the next digit via the guess-reveal algorithm in 97% of cases, with a longest continuous 'win' streak of 226.

Subtracting the 'longest streak' number from the total sequence length indicates the convergence time of the best model:

$$\tau_{16} \approx 240 - 226 = 14$$

Note that we define the 'best' model by that which has the highest correctness rating C_q. In certain cases, it's possible for a model without the highest C_q have the longest correctness streak.

Measurement: D_{10}

Upping the ante slightly, consider another percussive binary sequence $S_{240}(\phi_2)$ that is based on a D_{10} seed, particularly

$$S_{240}(\phi_2) = (0001110011\cdots)\ .$$

Running the sequence through model hunt mode gives information on the performance on each model W_q tested, where q runs from 0 to 1023.

Reporting results in a way analogous to the random sequence study above, we have:

- The best model is W_{162}, rated 97%, with a longest streak of 221.
- A second-best model is W_{38}, rated 96%, with a longest streak of 206. Comparable to this are W_{98}, W_{102}.
- The worst model is W_3, rated 30%.

Such a sequence clearly ordered as evident by the performance of the best model W_{162} along with its close contenders W_{38}, W_{98}, etc. From these we see the convergence time to be roughly

$$\tau_{162} \approx 240 - 221 = 19\ .$$

Interestingly, the worst-performing model W_3 gives a correctness rating of 30%, which, despite being an inverted answer, still tells us the sequence is ordered.

Measurement: D_{30}

A convenient pattern that seems to recur throughout the percussive example above is that, in both cases, the best model W_q is found long before the full range of q is traversed. From this we glean that low-order analysis with $g_{\max}$ set modestly to 10 can perform enough to achieve finite convergence time. Going for an extreme example, consider another percussive binary sequence $S_{240}(\phi_2)$ that is based on a D_{30} seed, particularly

$$S_{240}(\phi_2) = 010011000111000011110000011111\cdots\ .$$

Despite having a seed length of $T = 30$, which is three times $g_{\max}$, the analysis yields:

- The best model is W_{770}, rated 89%, with a longest streak of 175.
- Comparable to best-performing is W_{772}.
- The worst model is W_3, rated 34%.

Almost astonishingly, model W_{770} manages a correctness ratio of 89%, and achieves a convergence time after approximately

$$\tau_{770} \approx 240 - 175 = 65$$

digits.

5.3 Human Samples

Now we're ready for a fun and gratifying use of sequence analysis. A handful of individuals have volunteered hand-made binary sequences by pressing 0 and 1 on a keyboard 'as randomly as possible'. Samples submitted vary in length.

Data

A total of ten samples were collected from several individuals. Labeling the samples alphabetically, we have:

$$\begin{aligned}
A &= S_{1023} = 10101010101010101010101\ldots\\
B &= S_{511} = 01011101011000110101001 0\ldots\\
C &= S_{439} = 10110010100101001001001 1\ldots\\
D &= S_{311} = 01100101001010000110000 11\ldots\\
E &= S_{299} = 10101010101011101000011 1\ldots\\
F &= S_{299} = 10111010101010101010101 0\ldots\\
G &= S_{196} = 11111011110100101011111 1\ldots\\
H &= S_{314} = 01010100011001010010101 0\ldots\\
I &= S_{299} = 11011011011101011010001 0\ldots\\
J &= S_{343} = 10101000101010101100110 0\ldots
\end{aligned}$$

Testing

Each of the human-provided samples A to J is exposed to the full sequence analysis apparatus: translation, regrouping, grand population vector - the whole bit. For each sample, all we're shooting for is to find a model W_q that delivers a correctness rating of more than 55%, and then we'll know for sure that the sequence isn't inherently random.

It turns out our task is relatively easy. Alas, instead of running through all weight numbers $q = 0$ to $q = 1023$, it suffices to test a range much narrower, and for this we test models from $q = 15$ to $q = 64$.

Results

Letting a program do the hard work, the following results emerge:

Sample	Length	Best q	Correct	Streak
A	1023	18	86%	24
B	511	27	65%	11
C	439	18	78%	17
D	311	26	60%	12
E	299	24	69%	16
F	299	15	90%	56
G	196	19	70%	12
H	314	18	70%	14
I	299	20	61%	9
J	343	18	87%	38

From the table, we observe that, by far, no sample provided is 'random enough' to fool the machine, despite having sampled a small subset of model space. Keep in mind the number to beat is 55% for a sample length of 100, so the bar is already high.

The most challenging cases are arguably D and I, and perhaps there are models beyond $q = 64$ that better fit these. Whoever typed out sequence F probably wasn't trying.

5.4 Pathological Sequences

Let us play with a strange idea. Take any weighting model W_q, and consider a sequence of length one, containing just a single zero:

$$S_1(\phi_2) = (\,0\,)$$

Now, apply half of the guess-reveal algorithm, just the 'guess' part, to this trivial sequence. Based on the existing zero in the sequence, the prediction will be a second zero, i.e.

$$S_2(\phi_2) = \left(\,0\,X^2\,\right)\ ,$$

with $X^2 = 0$.

However, let's do the opposite of what the algorithm predicts, and assign $X^2 = 1$ instead. Then, starting with the sequence

$$S_2(\phi_2) = (\,0\,1\,)\ ,$$

let the algorithm predict the third digit, and then disobey the prediction. Whatever the prediction X^3 is, append the sequence with its opposite Z^3:

$$S_3(\phi_2) = \left(\,0\,1\,Z^3\right)$$

Carrying this process on, we end up with a sequence that is tuned to perfectly outsmart the model W_q. Such a sequence is *pathological* to the model, and when tested against W_q, yields a correctness rate of 0%. Once a sequence is known to be pathological, we change nomenclature to Z, replacing S:

$$S \to Z$$

Seed and Evolution

The first members of a pathological sequence determine the evolution of the rest. For this reason the first members are called the *seed*. For simplicity we choose a one-member seed, namely 0. Importantly, choosing a seed of 1 instead, let alone a more complicated one, alters all results that follow.

Also fundamental to a pathological sequence is the range of regroupings used to analyze the seed characterized by $g_{\max}$.

Examples: Low-q

Pathological sequences can take surprising form, even for low-q weighting models. Going through the first few nontrivial cases, we find the following:

$$Z_1 = 0110101010101010101010101010101010\ldots$$
$$Z_2 = 0110111000101110001011100010111000010\ldots$$
$$Z_3 = 0110100110011001100110011001100110011001\ldots$$

$$Z_4 = 0111101101001111000101101000101111000\ldots$$
$$Z_5 = 0111001011101000101100101111000011110\ldots$$
$$Z_6 = 0111001011101000111000101111010001110\ldots$$

$$Z_7 = 0110100111100010101110001100101110000\ldots$$
$$Z_8 = 0111110111101100111010101100111111010\ldots$$
$$Z_9 = 0110111001101010001011111000000101110\ldots$$

Examples: Larger q

Higher q-numbers lead to equally mysterious pathological sequences:

$$Z_{254} = 01111011111101011011001110001110110 1\ldots$$
$$Z_{255} = 01101111101011100111110001101001010 0\ldots$$
$$Z_{256} = 01111111111011111111101110111011111\ldots$$

Ring and Resonance

Peeling apart a given pathological sequence Z_q, there are evidently two kinds of behavior, one we'll call *ring*, and the other, *resonance*. The term 'ring' refers to qualitatively unordered behavior, exhibiting randomness or something without order. On the other hand, 'resonance' means the sequence takes on a percussive nature.

For instance, look again at Z_2, and write the sequence as

$$Z_2 = (011) + (01110001) + (01110001) + \cdots .$$

In the new terminology, the first three members of Z_2 grouped on their own as 'ring' terms, and all other members occurring as resonating chunks of period $T = 8$.

The transition from ring to resonance, and perhaps back to ring, and perhaps a blend of the two, is not obvious in the general case.

```
11110011010110101110110101010110110010 11
11101111101101101001101101110001101110 10
01001111001001111100001111101010111001 01
00111011010111110010111111100011111100 11
01111111111011111101101111101110111101 11
01110110001111010001111110010011110010 10
10011100110101100110110100110111100111 01
11111101010101101010111101011011101011 10
01011001010011001011010001011011010010 01
00110001110010001110100001110110011010 10
00000111100100111101000111101101010100 10
10010111011011000011011100011011001001 01
10111011110111010110110100100110100110 11
00101111101001111101111110001111110011 11
10100011010101001010101011110101111000 00
11011110100001110100011100100011110101 00
01111100011001100011011100011101100011 11
10011010110111010111011010111110010111 11
```

Figure 1.7: Text editor view of Z_{256} (partial sequence).

To illustrate, the sequence Z_{256} started above was generated to about 3000 members and inspected with a text editor availing a spectacle as seen in Figure 1.7. A glance at the arrangement of digits shows *some* kind of pattern. It just happens that model W_1, along with many others, can manage a correctness rating of 57% for the Z_{256} sequence.

5.5 Taxonomy of Correctness

Having tested a whole zoo of binary sequences by now, we've seen, for each case, the correctness rating for any given model falling somewhere between 0% and 100%. Here we list five distinct 'correctness zones' based on Equation (1.6):

- $C_q = 100\% \rightarrow$ **Percussive** zone.
- $55\% < C_q \leq 99\% \rightarrow$ **Bias** zone.
- $45\% \leq C_q \leq 55\% \rightarrow$ **Random** zone.
- $1\% \leq C_q < 45\% \rightarrow$ **Transcendental** zone.
- $C_q = 0\% \rightarrow$ **Pathological** zone.

Any binary sequence incoming from the wild is, depending on its discernible order, classified as either percussive, biased, or random.

Transcendental Zone

It's mighty peculiar that we are able to speak of correctness ratings significantly lower than 50%, like a broken clock being correct more than twice a day. There are (at least) two ways of getting into the so-called *transcendental* zone.

- Send a pathological sequence Z_q through guess-reveal based on a different associated model $W_{r\neq q}$.
- Manually edit the first (few) digit(s) of a pathological sequence Z_q and re-evaluate using W_q.

That is, only sequences that come *from* the guess-reveal algorithm are able to enter the transcendental zone. Funny enough, any correctness rating significantly away from 50%, above or below, is enough to know the sequence is ordered.

6 Creating a PRNG

We learned from pathological sequences that the guess-reveal algorithm can be used in reverse to generate sequences instead of analyze them.

Here we engage in a new question: Can the same toolkit be used to create a *pseudo random number generator* (in binary, of course)? This is abbreviated PRNG, and the prefix 'pseudo' means, for our purposes, that the output is completely deterministic, i.e. running the same program twice gives the same results twice.

Multiplicity of Output

The output of the PRNG is determined by its seed value. For our purposes, a binary seed value D_{10} of 10 digits, relating to $g_{\max} = 10$, shall kick-start the PRNG. The seed is treated as an internal variable, which is to say that the seed does not automatically appear in the resulting sequence. There are are 2^{10} total seeds. Right away, we see that there are 2^{10} possible ways to begin a pseudo-random sequence.

6.1 Initializing the Sequence

To prepare a pseudo-random binary sequence, first imagine two new variables:

$$K = \text{temporary working sequence}$$
$$R = \text{return value}$$

The variable K is initialized to store the incoming seed, and the variable R is initialized as empty:

$$K = D_{10}$$
$$R = (\text{empty})$$

6.2 Main Loop

Now the main loop is ready to go:

1. Isolate the last (rightmost) ten members in K. Convert this base-two number to base-ten, storing in q.
2. Initialize the weight model W_q.
3. Run the guess-reveal algorithm on K, storing the result in X.
4. Invert X (0 becomes 1 or vice versa) and store in Z.
5. Append Z to the right side of K and R.
6. Right-truncate K to main fixed length.

On a technical note, it helps to keep K limited to a few dozen characters, keeping perhaps the latest 40. This keeps the guess-reveal algorithm on its heels by denying the full sequence history from influencing later evolution.

Repeating the above steps N times places N members into the return sequence R_N.

6.3 Testing

To check the proposed PRNG's quality, we use the same apparatus used for testing the 'natural' cases, i.e. sequences known already to be orderless. Checking various issues of R_N using different seeds and different sequence lengths tends to yield results characterized by

$$C_q \approx 48\% \pm 5\% \,,$$

hovering nicely near the so-called 'random' zone.

Now, due to a possible a rubber-ruler argument, we won't read *too* much into whether the proposed PRNG produces quality output, particularly because we're testing with the same tool that forms the PRNG itself. Nonetheless it seems to work.