CHAOS GAME REPRESENTATION*

EUNICE Y. S. CHAN† AND ROBERT M. CORLESS‡

Abstract. The chaos game representation (CGR) is an interesting method to visualize one-dimensional sequences. In this paper, we show how to construct a chaos game representation. The applications mentioned here are biological, in which CGR was able to uncover patterns in DNA or proteins that were previously unknown. We also show how CGR might be introduced in the classroom, either in a modelling course or in a dynamical systems course. Some sequences that are tested are taken from the Online Encyclopedia of Integer Sequences, and others are taken from sequences that arose mainly from a course in experimental mathematics.

Key words. Chaos game representation, iterated function systems, partial quotients of continued fractions, sequences

AMS subject classifications.

1. Introduction. Finding hidden patterns in long sequences can be both difficult and valuable. Representing these sequence in a visual way can often help. The so-called chaos game representation (CGR) of a sequence of integers is a particularly useful technique, that visualizes a one-dimensional sequence in a two-dimensional space. The CGR is presented as a scatter plot (most frequently square), in which each corner represents an element that appears in the sequence. The results of these CGRs can look very fractal-like, but even so can be visually recognizable and distinguishable. The distinguishing characteristics can be made quantitative, with a good notion of "distance between images."

Many applications, such as analysis of DNA sequences [12, 15, 16, 18] and protein structure [4, 10], have shown the usefulness of CGR; we will discuss these applications briefly in section 4. Before that, in section 2, we will look at a random iteration algorithm for creating fractals. In section 5, we will apply CGR to simple abstract mathematical sequences, including the digits of π and the partial quotients of continued fractions. Additionally, we will look at certain "distances between images" that are produced. Finally, we will explain how some of the patterns arise, and what these depictions can tell us about the sequences.

One important pedagogical purpose of this module is to provoke a discussion about randomness, or what it means to be random. In this paper, we at first use the word "random" very loosely, on purpose.

> "What is probability? I asked myself this question many years ago, and found that various authors gave different answers. I found that there were several main schools of thought with many variations."
>
> — Richard W. Hamming[1] [13]

2. Creating Fractals using Iterated Function Systems (IFS). The random iteration algorithm for creating pictures of fractals uses the fixed attracting set of an *iterated function system* (IFS). Before defining IFS, we will first define some necessary preliminaries.

*Submitted to the editors DATE.

†Department of Anesthesia and Perioperative Medicine, MEDICI Centre, Western University, London, Ontario, Canada (echan295@uwo.ca).

‡Ontario Research Center for Computer Algebra, School of Mathematical and Statistical Sciences, and Rotman Institute of Philosophy, Western University, London, Ontario, Canada (rcorless@uwo.ca).

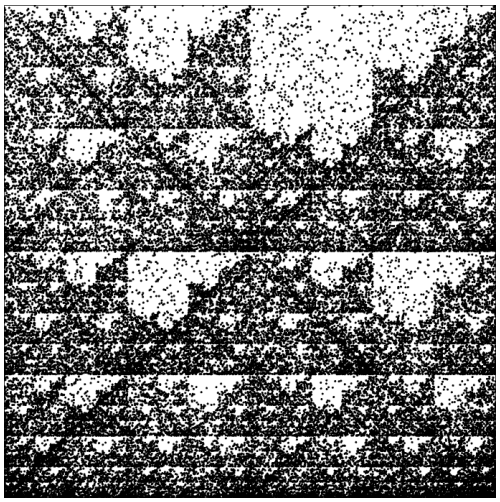


Fig. 1: DNA of human beta globin region on chromosome 11 (HUMHBB)—73308 bps

2.1. Affine Transformation in the Euclidean Plane. One can write a two-dimensional *affine* transformation in the Euclidean plane $w : \mathbb{R}^2 \to \mathbb{R}^2$ in the form

$$w(x_1, x_2) = (ax_1 + bx_2 + e, cx_1 + dx_2 + f) \tag{1}$$

where a, b, c, d, e, and f are real numbers [3]. We will use the following more compact notation:

$$\begin{aligned} w(x) = w\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} &= \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} e \\ f \end{bmatrix} \\ &= Ax + t\,. \end{aligned} \tag{2}$$

The matrix A is a 2×2 real matrix and can always be written in the form

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} = \begin{bmatrix} r_1 \cos\theta_1 & -r_2 \sin\theta_2 \\ r_1 \sin\theta_1 & r_2 \cos\theta_2 \end{bmatrix}\,, \tag{3}$$

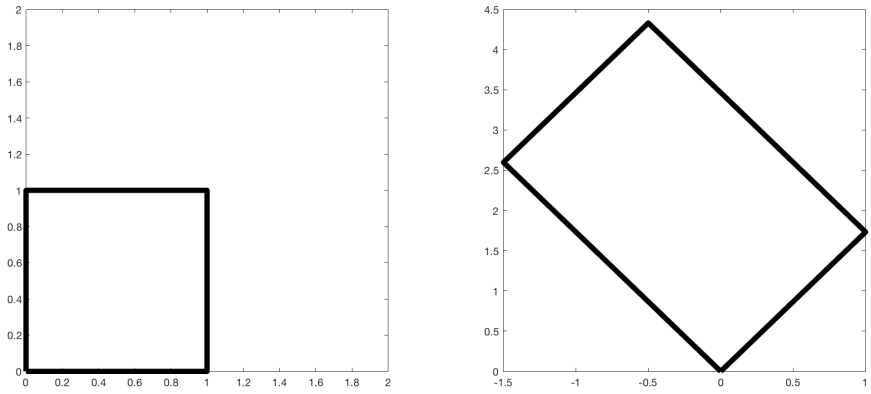
where (r_1, θ_1) are the polar coordinates of the point (a, c) and $(r_2, (\theta_2 + \pi/2))$ are the polar coordinates of the point (b, d). An example of a *linear* transformation

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \to A \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \tag{4}$$

in $\mathbb{R}^2$ is shown in figure 2. We can see from this figure that the original shape remains as a parallelogram, but now has changed in size and in rotation. This is what we mean by *linear* transformation.

2.2. Iterated Function System. An *iterated function system* (IFS)[2] is a finite set of contraction mappings on a metric space. These can be used to create pictures

[2]To be more specific, this is a *hyperbolic* iterated function system. However, in practice, the word "hyperbolic" is sometimes dropped in practice [3, p. 82].



(a) Shape before transformation

(b) Shape after transformation

Fig. 2: Transformation of a unit square using the transformation matrix A (as shown in equation (3)), where $r_1 = 2$, $r_2 = 3$, $\theta_1 = \pi/3$, and $\theta_2 = \pi/6$.

of fractals. These pictures essentially show what is called the attractor of the IFS. We will be defining what an attractor is in Section 3. In this paper, we will only be looking at the form where each contraction mapping is an affine transformation on the metric space $\mathbb{R}^2$.

We will illustrate the algorithms for an IFS as an example. Consider the following maps:

$$
\begin{aligned}
w_1(x,y) &= \begin{bmatrix} 1/2 & 0 \\ 0 & 1/2 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \\
w_2(x,y) &= \begin{bmatrix} 1/2 & 0 \\ 0 & 1/2 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} 0 \\ 1/2 \end{bmatrix}, \\
w_3(x,y) &= \begin{bmatrix} 1/2 & 0 \\ 0 & 1/2 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} 1/2 \\ 1/2 \end{bmatrix},
\end{aligned} \tag{5}
$$

each with the *probability factor* of $1/3$. These values can also be displayed as a table, where the order of the coefficients a through f corresponds to the order presented in equation (2.1) and p represents the probability factor. The corresponding table for equation (2.2) can be found in table 1.

Table 1: Table representing the IFS from equation (2.2)

w	a	b	c	d	e	f	p
1	$1/2$	0	0	$1/2$	0	0	$1/3$
2	$1/2$	0	0	$1/2$	0	$1/2$	$1/3$
3	$1/2$	0	0	$1/2$	$1/2$	$1/2$	$1/3$

To create a fractal picture using IFS, we first choose a starting point (x_0, y_0). We then *randomly* choose a map from our IFS and evaluate it at our starting point

(x_0, y_0) to get our next point (x_1, y_1). We do this again many times (determined by the user) until a pattern (usually fractal) appears. The MATLAB code shown in Listing 1 computes and plots 50000 points corresponding to the IFS from table 1. To choose which map to use for each iteration, we used a uniform random number generator `randi` (built-in function in MATLAB) to choose a number between (and including) 0 and 2. Each value corresponds to a map in our IFS. The picture produced from Listing 1 is shown in Figure 3. It is clear that this figure shows a fractal, which appears to be the Sierpinski triangle [3, 9], where its three vertices are located at $(0, 0)$, $(0, 1)$ and $(1, 1)$.

Listing 1 MATLAB Code for an IFS generating a Sierpinski triangle

```
n = 50000;
pts = zeros(2, n);
rseq = randi([0, 2], 1, n-1);
for i = 2:n
        if rseq(i-1) == 0
                pts(:, i) = [0.5, 0; 0, 0.5]*pts(:, i-1) + [0; 0];
        elseif rseq(i-1) == 1
                pts(:, i) = [0.5, 0; 0, 0.5]*pts(:, i-1) + [0; 0.5];
        else
                pts(:, i) = [0.5, 0; 0, 0.5]*pts(:, i-1) + [0.5; 0.5];
        end
end
plot(pts(1, :), pts(2, :), 'k.', 'MarkerSize', 1);
axis('square');
set(gca, 'xtick', []);
set(gca, 'ytick', []);
```

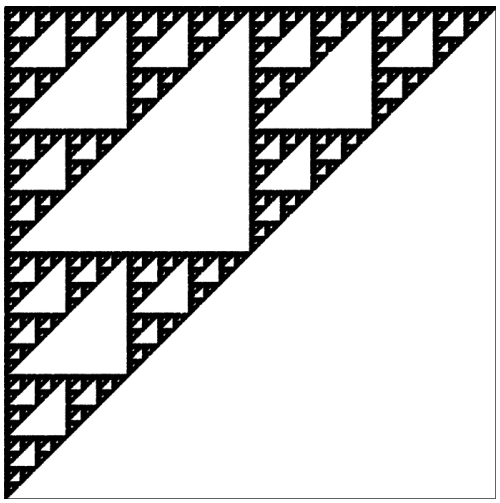


Fig. 3: Results from Listing 1. We see visible regularity arising from a program that uses randomness.

Iterated function systems can be quite versatile; a set of several attractors can be

use to create different shapes and even can resemble real-life objects. An example of this is Barnsley's fern, which resembles the Black Spleenwort [3] (see Figure 4b). The map is displayed in the following table:

w	a	b	c	d	e	f	p
1	0.00	0.00	0.00	0.16	0.00	0.00	0.01
2	0.85	0.04	−0.04	0.85	0.00	1.60	0.85
3	0.20	−0.26	0.23	0.22	0.00	1.60	0.07
4	−0.15	0.28	0.26	0.24	0.00	0.44	0.07

The plot produced by this IFS is shown in figure 4a.

(a) Barnsley's Fern (n = 50000)

(b) Black spleenwort[a]

[a] http://www.aphotoflora.com/images/pteridophyta_aspleniaceae_ferns/asplenium_adiantum-nigrum_leaf_27-09-08.jpg

Fig. 4: Comparison of Barnsley's Fern to the black spleenwort, which is what the IFS was based on.

3. Chaos game. CGR is based on a technique from chaotic dynamics called the "chaos game," popularized by Barnsley [3] in 1988. The chaos game is an algorithm which allows one to produce pictures of fractal structures [15]. In its simplest form, this can be demonstrated with a piece of paper and pencil using the following steps (adapted from [15]):

1. On a piece of paper, draw three points spaced out across the page; they do not necessarily need to be spaced out evenly to form an equilateral triangle though they might be.
2. Label one of the points with "1, 2," the other point with "3, 4," and the last point with "5, 6."
3. Draw a point anywhere on the page: this will be the starting point.
4. Roll a six-sided die. Draw an imaginary line from the current dot towards to the point with the corresponding number, and put a dot half way on the line. This becomes the new "current dot."
5. Repeat step 4 (until you get bored).

Obviously, one would not want to plot a large number of points by hand[3]; instead, this can be done by computer; the code is shown in Listing 2. For this algorithm, we assigned $(0, 0)$, $(0, 1)$ and $(1, 1)$ as the vertices of the triangle; they each are labelled with the value 0, 1, and 2, respectively. Instead of using a die to decide which vertex to choose, similarly to Listing 1 we use MATLAB's built-in function `randi`, which picks from the set $\{0, 1, 2\}$.

To calculate the new point (shown in Listing 2), we take the average between the previous point and the corresponding vertex. For example, if the first random number generated was 2, then we would take the average between our initial point (0, 0) and the vertex that corresponds to 2, which is in our case is $(1, 1)$. Our new point would be $(0.5, 0.5)$. We then would repeat this process until one decides to stop.

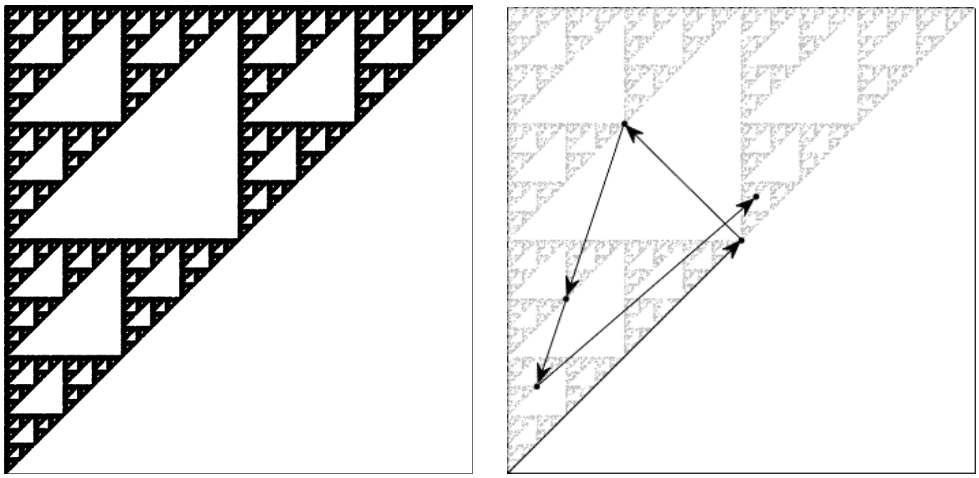
Listing 2 MATLAB Code for CGR with three vertices using a random number generator

```
n = 50000;
pts = zeros(2, n);
rseq = randi([0, 2], 1, n-1);
for i = 2:n
        if rseq(i-1) == 0
                pts(:, i) = 0.5*(pts(:, i-1) + [0; 0]);
        elseif rseq(i-1) == 1
                pts(:, i) = 0.5*(pts(:, i-1) + [0; 1]);
        else
                pts(:, i) = 0.5*(pts(:, i-1) + [1; 1]);
        end
end
plot(pts(1, :), pts(2, :), 'k.', 'MarkerSize', 1);
axis('square');
set(gca, 'xtick', []);
set(gca, 'ytick', []);
```

One would expect that the outcome of Listing 2 would show that the dots would appear randomly; however, this is not the case: Figure 5a shows the result of the chaos game if the game was played 50000 times. We can see that this figure shows a fractal-like pattern (such as one of a Sierpinski triangle), and looks almost (if not) identical to the figure which resulted from the IFS.

It is quite interesting that a sequence of random numbers could produce such a distinct pattern. Indeed, some bad *pseudorandom number generators* were identified as being bad precisely because they failed a similar-in-spirit visualization test [22]. We will discuss this issue further in section 6. Figure 5b shows an example of the first five steps of the chaos game (overlaid on top of a CGR with 10000 points), which corresponds to the following values that were randomly generated: 2, 1, 0, 0, 2. Displaying the CGR in this manner can give us some insight as to why this particular chaos game with three vertices gives us such a striking fractal: we can see from the figure that each of the points from the first five steps is located at a vertex (in this case the right-angled one) of one of the many triangles within the fractal. A

[3]We tried this, though not in class. We got up to 80 points on one figure, using the die roller on random.org. It's kind of fun, in a physical "drawing and measuring" way, but it's hard not to make mistakes. It could make a useful "active learning" exercise and we might try it in class in the future.



(a) Sierpinski triangle ($n = 50000$)

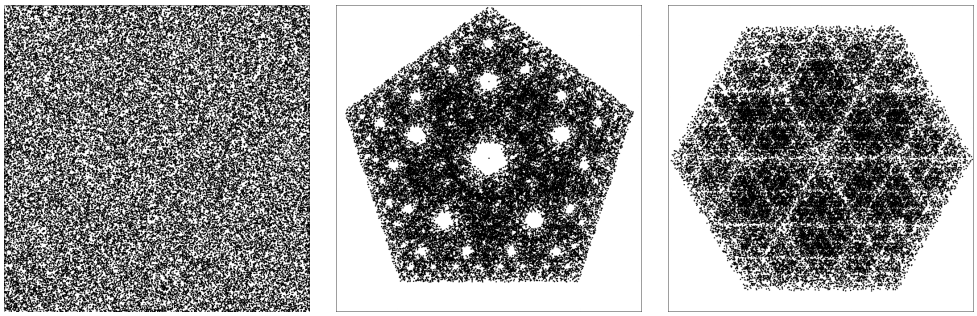
(b) First 5 steps of chaos game with three vertices. In this example, the first 5 random integers generated are 2, 1, 0, 0, 2.

Fig. 5: CGR with three vertices of randomly generated integers.

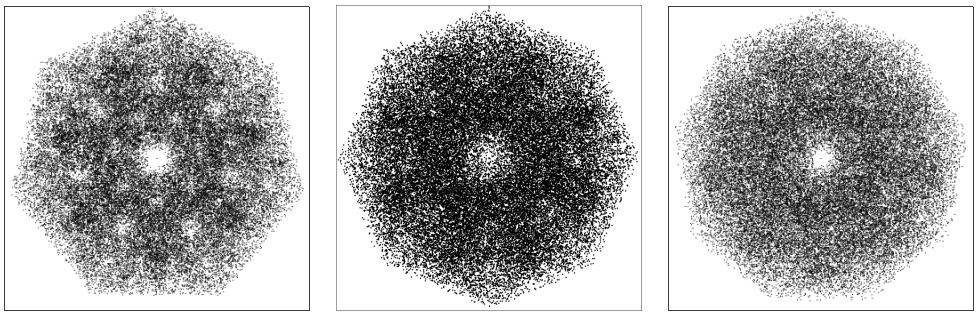
careful observer would notice that the triangle at which the point is located becomes smaller each iteration. This implies that within the next few iterations, the triangle associated with the computed point would be too small to be seen in this illustration. [Note: if we zoom into that microscopic level, the dots would look random. Indeed, the random dots are ultimately dense on the Sierpinski triangle.] This sequence of points generated by the chaos game is called the *orbit* of the seed and it is attracted to the Sierpinski triangle, sometimes called a *strange attractor*. For another description of why this chaos game creates the Sierpinski triangle, see [11].

3.1. Generalization of CGR. We have already seen that a fractal pattern can appear when we follow the chaos game described above for a polygon with three vertices (triangle) using a random number generator, but we are not limited to only this. There are many variations for CGR, though some are more useful than others. In this section, we will look into what patterns arises when we perform CGR on different polygons, changing the placement of the m-th point relative to the $(m-1)$-th and corresponding vertex, and explore which of these could potentially be useful for some applications presented in Section 4.

3.1.1. Different Polygons. Figure 6 shows the chaos game representation, where the m-th point is placed halfway between the $(m-1)$-th point and the vertex corresponding to the m-th base, for various polygons, using 50,000 points. As we can see from Figure 6a, there is no apparent pattern: the chaos game produced a square uniformly and randomly filled with points. The other figures from Figure 6 appear to exhibit some patterns. These patterns are due to several parts of their attractors overlapping, indicated by the the uneven distribution of points (where points do appear); i.e. we can see some darker areas and some lighter areas in the figures. Therefore, for this particular chaos game (in which we say that the dividing rate r is 0.5—we will talk about this in section 3.1.2), the square is the best choice to visualize



(a) Square (b) Pentagon (c) Hexagon



(d) Heptagon (e) Octagon (f) Nonagon

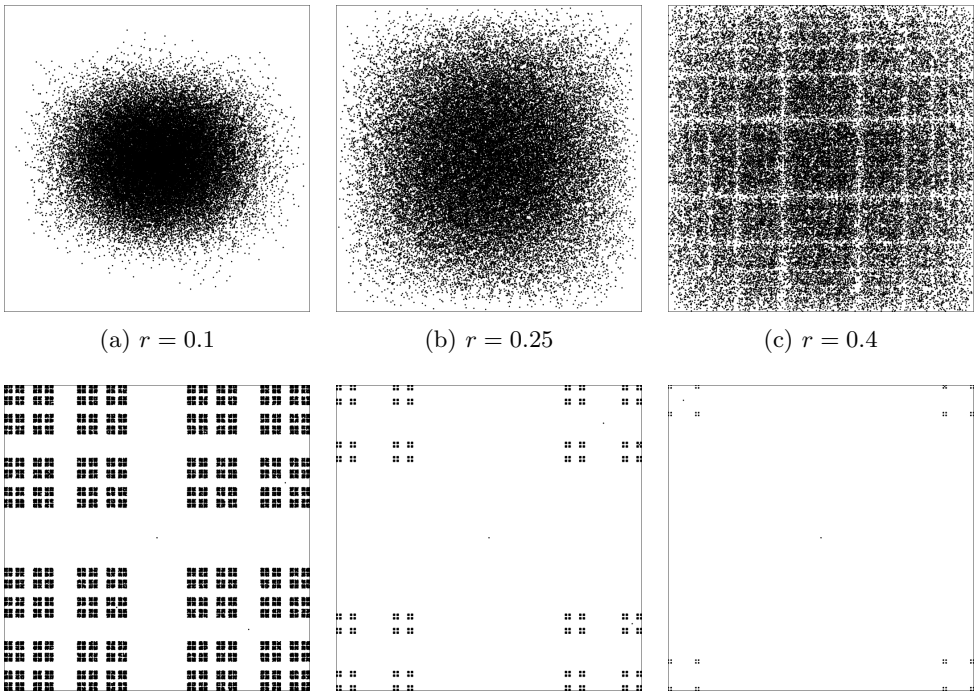
Fig. 6: Chaos game of polygon with different number of vertices

one-dimensional sequences. We will see in section 3.1.3 that we can use other polygons (we will be referring to them as n-gons) for CGR to uncover patterns in sequences not with 4 elements by using the appropriate dividing rate.

3.1.2. Dividing rate. In the previous subsection, we saw the different patterns appear for different shapes. We saw that for shapes with more than 4 vertices had several parts of the attractor overlapping. Here we will show that attractor overlapping also happens when choosing a small dividing rate.

The chaos game is not restricted to the "rule" in which the new point has to be placed halfway between the current point and the corresponding vertex as depicted in the previous examples. The new point can be placed anywhere within the line segment created by the two points of reference. The placement of the new point affects how the attractor of the CGR looks. We will see this in the following example. To quantify the placement of the point, we take the proportion of which the distance between the new point and current point is from the distance between the current point and the corresponding vertex. This is called the *dividing rate*, r. Therefore, when the new point is placed halfway between the current point and the vertex, the dividing rate is $r = 0.5$, and if the new point is placed closer to the location of the current point, then the dividing rate $r < 0.5$, whereas if the point is closer to the vertex, then the dividing rate $r > 0.5$.

To begin this discussion, let us vary r for the square-shaped CGR of a sequence of random integers. As we have seen in the previous subsection, the square-shaped



(a) $r = 0.1$ (b) $r = 0.25$ (c) $r = 0.4$

(d) $r = 0.6$ (e) $r = 0.75$ (f) $r = 0.9$

Fig. 7: Illustration of different r affecting how a square CGR looks using 50000 points.

CGR did not exhibit any fractal patterns when $r = 0.5$, so we wondered if we varied r, would this change? Figure 7 shows the square CGRs for $r = 0.1,\ 0.25,\ 0.4,\ 0.6,\ 0.75,$ and 0.9. From these figures, we see how r affects the visualization.

For $r < 0.5$, we can see that some pattern appear, especially those with a dividing rate close to $r = 0.5$. We can see for $r = 0.6$, the points cover the entire plot area, but they are unevenly-space (in comparison to what we have seen for $r = 0.5$). This suggests that for this dividing rate, the attractors are overlapping, indicated by the darker regions of the plot due to the points being more densely packed. As r decreases, we can see that the CGR becomes more circular with the points spaced more closely together. From this, it infers that as r decreases, more of the attractors overlap.

For $r > 0.5$, we can see that a different pattern appears (compared to those with a dividing ratio $r < 0.5$). The pattern shown for the CGRs with a dividing ratio $r > 0.5$ is similar to one another, as seen in Figure 7. Figure 7f looks like a smaller version of Figure 7e, which similarly, looks like a smaller version of Figure 7d. In contrast to the CGR with the dividing rate $r > 0.5$, we can see that the points are spread apart from each other. This can possibly indicate that the attractors do not overlap, as well as do not touch each other. Although this can be used to distinguish any nonrandomness, it would be much more difficult and using the dividing rate $r = 0.5$.

In applications, it is optimal to find a dividing rate for different polygons that can offer maximum packing of nested, non-overlapping attractors in order to find a pattern in a given sequence. In the next subsection, we look at this generalization of

CGR.

3.1.3. Generalization of CGR. In [10], Fiser et al. generalized CGR to be applicable for sequences of any number of elements. There are many ways of generalizing CGR, which are highlighted in [2], but the generalization the authors had chosen was to use an n-sided polygon (which they refer to as an n-gon), where n was the number of elements in a sequence that should be represented [10]. As we saw in Figure 6, the attractors of the n-gons for when $n > 4$ overlapped shown by the uneven distribution of the points. Because of this, the attractor is ambiguous and is not useful to serve as a way to identity patterns in sequences. To combat this, Fiser et al. introduced a formula to calculate the dividing rate for n-gons for different values of n:

$$r = \left(1 + \sin\left(\frac{\pi}{n}\right)\right)^{-1} .$$

Although the dividing ratio calculated from this formula does prevent attractors from overlapping, Almeida and Vinga noticed that the attractors are not optimally packed in [2]. For example, for $n = 4$, the solution gives a dividing ratio $r = 0.585786$ instead of the *typical* $r = 0.5$. In [2], the authors improved upon this formula for the dividing ratio:

$$r = \frac{2\cos\left(\pi\left(\frac{1}{2} - \frac{k}{n}\right)\right) - 2\cos\left(\pi\left(\frac{1}{2} - \frac{1}{2n}\right)\right)\cos\left((2k-1)\frac{\pi}{2n}\right)\left(1 + \frac{\tan\left((2k-1)\frac{\pi}{2n}\right)}{\tan\left(\pi - (n+2k-2)\left(\frac{\pi}{2n}\right)\right)}\right)}{2\cos\left(\pi\left(\frac{1}{2} - \frac{k}{n}\right)\right)}$$

where

$$k = \text{round}\left(\frac{n+1}{4}\right) .$$

Figure 8 shows the generalized CGRs for various n-gons using the division formula from [10] (left) and [2] (right).

In the next section, we will look at applications of CGR, mainly biological and mathematical. For the following examples, unless otherwise specified, assume that $r = 0.5$.

4. CGR of biological sequences. The chaos game representation has been applied to biological areas. In this section, we will look at how the chaos game has been applied the DNA and protein (amino acids) sequencing and visualizations. We also will give a brief overview of the *Map of Life*, an extension of the visualization of DNA sequences. The Map of Life shows potential in allowing researchers to quantitatively classify species that was once unclear in its taxonomy.

Life certainly has random elements, but it has deterministic elements as well. The hope of CGR is to reveal some of the latter amidst the clutter of the former.

4.1. Nucleotide sequences. H. J. Jeffrey was the first to propose using the chaos game representation as a novel way of visualizing nucleotide sequences. This revealed previously unknown patterns in certain proteins [15, 16]. The genetic sequence is made up of four bases: adenine (A), guanine (G), cytosine (C), and either thymine (T) or uracil (U) for DNA or RNA, respectively. Using a square-shaped CGR, we label the four corners with the name of each base. In this paper, A is in the bottom-left corner, C in the top-left, G in the top-right, and U/T in the bottom right. Rather than use a random number generator to determine which map to use

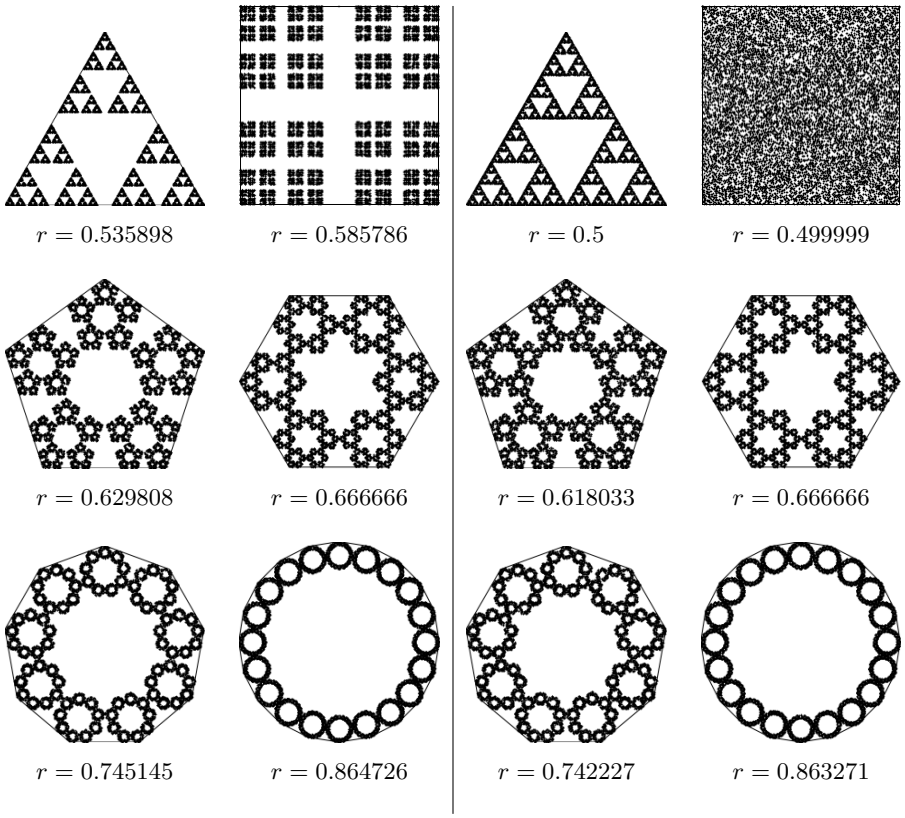


Fig. 8: Graphical comparison of proposed CGR generalization using 50000 points. The left uses the dividing ratio formula from [10], and the right uses the dividing ratio formula from [2].

for each iteration as we did in Section 3, we follow the genetic sequence that we want to create a CGR for.

To demonstrate the chaos game for DNA sequences, let us walk through plotting the first five bases of the DNA sequence HUMHBB (human beta globin region, chromosome 11), "GAATT," shown in Figure 9. We first mark the center as our initial point. The first base in the sequence is "G" so, we plot a point half way between our initial point and the "G" corner. The next base in the sequence is "A" so we plot a point half way between the point that we just plotted and the "A" corner. We leave the reader to check the placement of the remaining steps.

The finished CGR is shown in Figure 1, which certainly has an identifiable fractal pattern. The most prominent pattern is what is referred to as the "double scoop," which actually appears in almost all vertebrate DNA sequences. This pattern is due to the fact that there is a comparative sparseness of guanine following cytosine in the gene sequence since CG dinucleotides are prone to methylation and subsequently mutation.

To fully understand the double scoop pattern, we must understand the biological meaning of the CGR. Each point plotted in the CGR corresponds to a base, and depending on where it is placed, we can trace back and figure out parts of the sequence

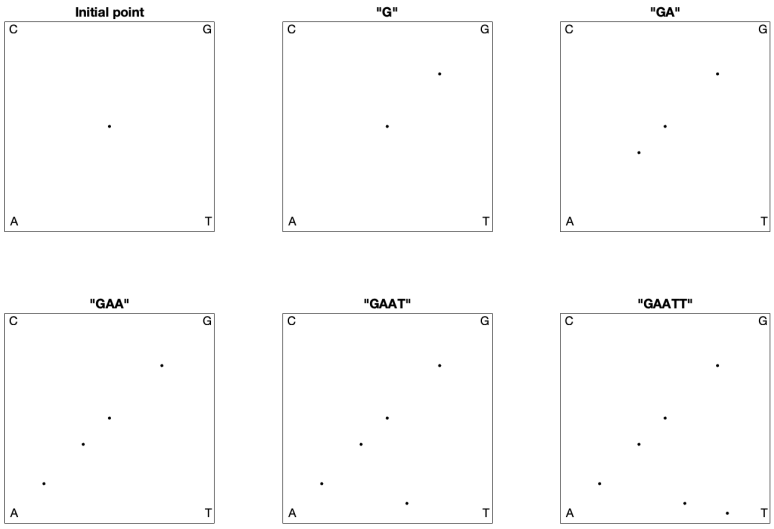
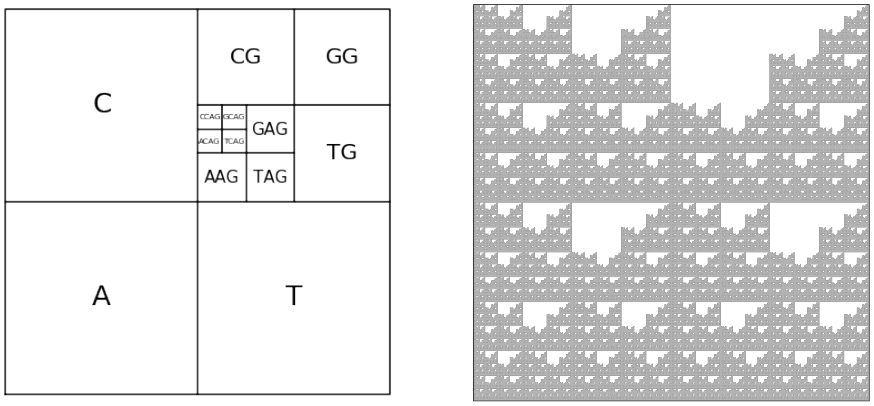


Fig. 9: First five steps of producing CGR using HUMHBB DNA

we are examining [12]. Figure 10a shows the relationship corresponding area of the CGR and the DNA sequence. In reference to this figure, we can see that for any point that corresponds with base G, it will be located in the upper right quadrant of the CGR plot. To see what the previous base is, we can divide the quadrant into sub-quadrants (labeling them in the same order as the quadrants), and depending on where the point is, we can determine what the previous base of the sequence is. We can repeat this step again and again to find the order in which the bases appear in the sequence. Figure 10b shows a CGR square where all the CG quadrants are unfilled. We can see from this figure that even though it is not a real CGR (since we did not use a sequence to produce it), we still get the same double scoop pattern found in Figure 1. By identifying regions of the CGR square in this way, it is possible to identify features of DNA sequences that correspond to patterns of the CGR.

Figure 11 shows other examples of CGRs for DNA sequences. We can see that Figure 11a (CGR of DNA sequence of human herpesvirus strain) also exhibit the double scoop pattern that we have seen for HUMHBB. This agrees with what was mentioned earlier in that it is common to see the lack of C and G dinucleotides together within the human genome. Figure 11b, on the other hand, shows the CGR of the DNA sequence of the chloroplast of quinoa plant, and does not show a double scoop pattern. We can see from using the CGR of DNA sequences, we are able to distinguish between different species very easily.

4.2. Map of Life. Taking DNA sequence visualization one step further, the authors of [19] proposed a novel combination of methods to create *Molecular Distance Maps*. Molecular distance maps visually illustrate the quantitative relationships and patterns of proximities among the given genetic sequences and among the species they represent. To compute and visually display relationships between DNA sequences, the three techniques that were used include chaos game representation, structural dissimilarity index (DSSIM), and multi-dimensional scaling. In the paper [19], this method was applied to a variety of cases that have been historically controversial and



(a) Correspondence between DNA sequences and areas of the CGR of DNA sequences

(b) Explanation of the double scoop pattern—plot of CGR square with all CG quadrants unfilled.

Fig. 10: CGR quadrants to explain biological phenomenon such as double scoop pattern

was there demonstrated to have the potential for taxonomical clarification. In the following, we will discuss the methods used from this paper, more particularly the structural dissimilarity index and the multi-dimensional scaling as we have already looked into the chaos game representation for nucleotides.

To understand the structural dissimilarity index, we will first have to explain what the structural similarity index is. The structural similarity (SSIM) index is a method for measuring the similarity between two images based on the computation of three terms, namely luminance distortion, contrast distortion, and linear correlation [31]. It was designed to perform similarly to the human visual system, which is highly adapted to extract structural information. The overall index is a multiplicative combination of the three terms:

$$\mathrm{SSIM}(x,y) = [l(x,y)]^{\alpha} \cdot [c(x,y)]^{\beta} \cdot [s(x,y)]^{\gamma} ,$$

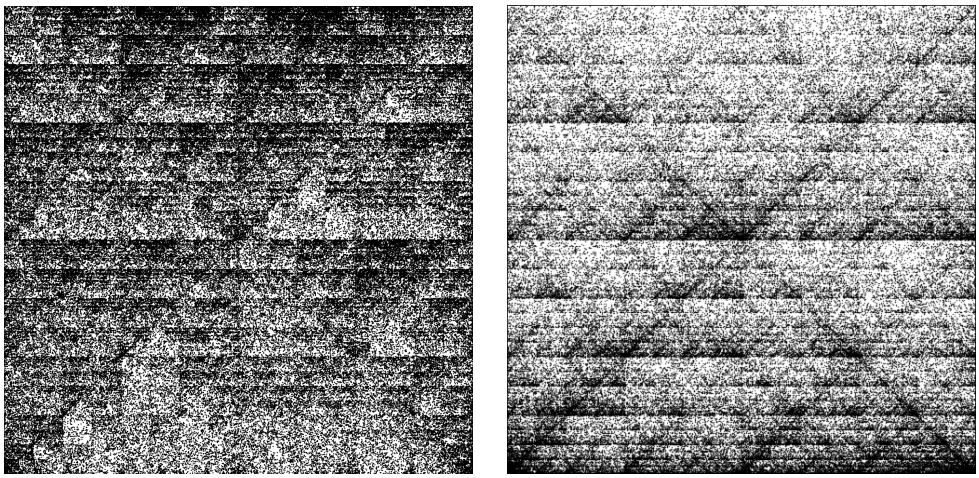
where

$$l(x,y) = \frac{2\mu_x\mu_y + C_1}{\mu_x^2 + \mu_y^2 + C_1} \tag{6}$$

$$c(x,y) = \frac{2\sigma_x\sigma_y + C_2}{\sigma_x^2 + \sigma_y^2 + C_2} \tag{7}$$

$$s(x,y) = \frac{\sigma_{xy} + C_3}{\sigma_x\sigma_y + C_3} , \tag{8}$$

where μ_x, μ_y, σ_x, σ_y and σ_{xy} are the local means, standard deviations, and cross-covariance for images x and y. C_1, C_2, and C_3 are the regularization constants for the luminance, contrast, and structural terms, respectively. If $\alpha = \beta = \gamma = 1$, and



(a) DNA of Human herpesvirus 5 strain TR, complete genome—235681 bp

(b) DNA of Chenopodium quinoa cultivar Real Blanca chloroplast, complete sequence, whole genome shotgun sequence–152282 bp

Fig. 11: Examples of CGR of DNA sequences. From the bottom left corner, going clockwise, the bases are A, C, G, T.

$C_3 = {}^{C_2}/_2$, the index simplifies to:

$$\mathrm{SSIM}(x,y) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)} \,.$$

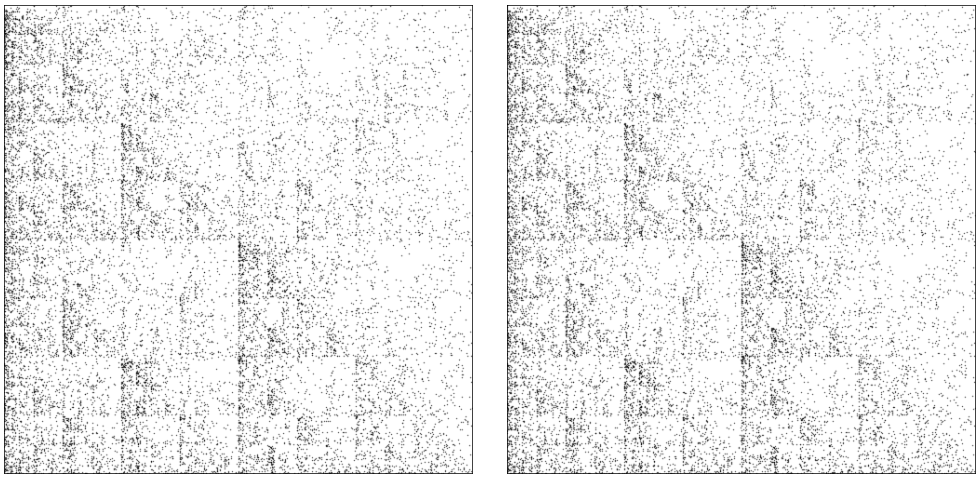
The theoretical range of $\mathrm{SSIM}(x,y) \in [-1,1]$, where the high value indicates high similarity. In [19], instead of calculating the overall SSIM, they computed the local SSIM value for each pixel in the image x. They refer to this as a distance matrix. Both global and local SSIM index can be computed in MATLAB by using the function `ssim`. For our own experiments presented in this paper, we use the global SSIM value rather than the local.

Now that we know how to compute the SSIM index, we are able to compute the structural dissimilarity (DSSIM) index:

$$\mathrm{DSSIM}(x,y) = 1 - \mathrm{SSIM}(x,y) \,,$$

whose theoretical range is $[0,2]$ with the distance being 0 between two identical images, and 2 if the two images are negatively correlated. An example of two images that are negatively correlated would be if one is completely white, while the other is completely black. Typically, the range that DSSIM falls with is $[0,1]$—the authors from [20] noted that almost all (over 5 million) distances that found were between 0 and 1, with only half a dozen exceptions of distances between 1 and 1.0033.

To demonstrate this, let us compare the "anatomically modern" human (Homo sapiens sapiens) mitochondrial DNA and the neanderthal (Homo sapiens neanderthalensis) mitochondrial DNA (mtDNA). As seen from their scientific classification, we can see that these two species are from the same genus, which suggests that their DNA should be similar. Figure 12 shows the CGRs of the "anatomically modern" human



(a) CGR of human (Homo sapiens sapiens) mitochondrial DNA — 16,569 bp

(b) CGR of neanderthal (Homo sapiens neanderthalensis) — 16,565 bp

Fig. 12: CGR of human and neanderthal mitochondrial DNA.

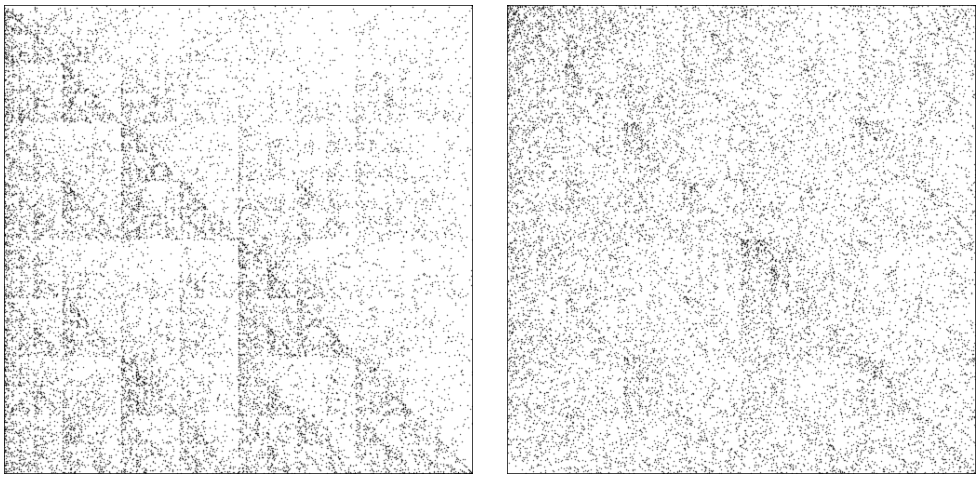
(12a) and of the neanderthal (12b). We can see that the two CGRs are quite similar (which is agrees with the fact that the two species are from the same genus) and therefore, we expect the the DSSIM to be small. Using MATLAB's `ssim` function, the structural similarity index between the two images is 0.8777, so the structural dissimilarity index is $1 - 0.8777 = 0.1223$, which agrees with our expectations.

We can look at another example that compares the mtDNA of humans to other species, such as the great spotted kiwi (Figure 13a) and pearlfish (Figure 13b). Just visually, the CGR of the mtDNA of the great spotted kiwi looks more similar to the CGRs of the human and neanderthal than the CGR of the pearlfish. Using all of the DSSIM, we can produce what is called a distance matrix, which is a square, real, symmetric matrix where each element is the structural dissimilarity index of two species with corresponds to the row and column. The distance matrix for the four species that we have already produced the CGR for is

$$\text{DSSIM}(x, y) = \begin{bmatrix} 0 & 0.1223 & 0.7823 & 0.8541 \\ 0.1223 & 0 & 0.7821 & 0.8533 \\ 0.7823 & 0.7821 & 0 & 0.8519 \\ 0.8541 & 0.8533 & 0.8519 & 0 \end{bmatrix},$$

where the first row/column represents humans, the second row/column represents the neanderthals, the third row/column represents the great spotted kiwi, and lastly, the fourth row/column represents the pearlfish. Obviously, the diagonal of the matrix would be 0, since the DSSIM of two of the same images would produce a result of 0. As we have seen previous, the DSSIM(human, neanderthal) = 0.1223, and vice versa. From this matrix, we can see that the CGR of the mtDNA of pearlfish is most different than the rest—this agrees with what we have observed.

Multi-dimensional scaling (MDS) is a means of visualizing the level of similarity or dissimilarity of individual case of a dataset. It has been used in various fields such



(a) CGR of the great spotted kiwi (Apteryx haastii) mitochondrial DNA — 16,980 bp

(b) CGR of the pearlfish (Carapus bermudensis) — 16,613 bp

Fig. 13: CGR of other species' mitochondrial DNA.

as cognitive science, information science, pychometrics, marketing, ecology, social science and other areas of study [5]. The goal of MDS is to find a spatial configuration of objects when all that is known is some measure of their general similarity or dissimilarity [32]. In our case, MDS takes in the distance matrix and outputs a two-dimensional map, where each item is represented by a point. In [19], the authors used a classical MDS, which assumes that all the distances (from the distance matrix) are Euclidean. For the algorithm for classical MDS, refer to [32, p. 10]. MATLAB has its own built-in function that does MDS, called `cmdscale`.

Figure 14 shows an example of a molecular distance map of 4844 animals from various classes from the chordate phylum (meaning that all these animals have a vertebrate), coloured according to their class. The sequences were taken from NCBI Reference Sequence Database (RefSeq). We can see from this figure that this method (Map of Life) does quite a good job in sorting species into different categories. One particular aspect that we want to point out in this figure is where the points representing the lungfish are—they are in the area of where the bony fish, cartilage fish, and amphibians touch. This is truly remarkable as lung fish has qualities of all these classes.

4.3. Protein visualization. CGR has also been applied to visualizing and analyzing both the primary and secondary structures of proteins. The primary structure of a protein is simply an amino acid sequence. To analyze the primary structure of proteins, the authors from [10] using an 20-sided regular polygon, each representing an amino acid. Table 2 shows all 20 amino acids along with their 3-letter code, as well as their 1-letter code. To avoid the attractor from overlapping itself, we will use the dividing rate for a 20-gon shown from [2] in Figure 8, $r = 0.863271$. Some CGRs for protein visualizations can be found in Figure 15.

According to Basu et al. [4], there are two serious limitations for using a 20-vertex CGR to identify patterns. The first limitation the authors mentioned about using the

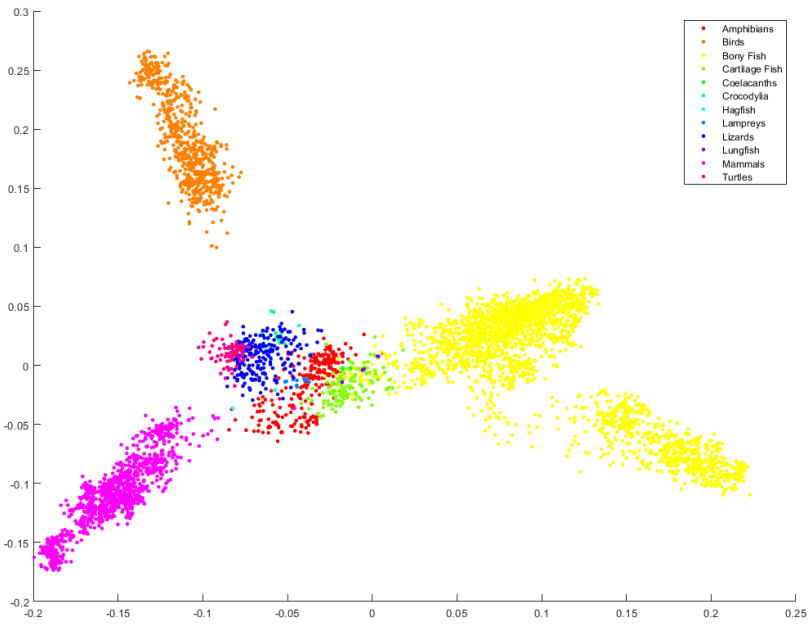


Fig. 14: Molecular distance map of 4844 animals from various classes from the chordate phylum, coloured according to their class.

20-vertex is that it is hard to visualize different protein families since one would have to plot the sequences in different polygons rather than plotting a random sampling of proteins of different functions and origins in a single CGR [10]. The second limitation is that there the amino acid residues in different positions are often replaced by similar amino acids [4]. This means that the 20-vertex CGR cannot be used to differentiate between similar and dissimilar residues and the visualization could be different for proteins within the same family.

To fix the second limitation mentioned in [4], Fiser et al. [10] introduced using CGR to study 3D structures of proteins. A chain of amino acid is what is called a polypeptide. Due to each amino acid having a specific structure, which contributes to the amino acid's properties, such as hydrophobicity or hydrogen-bonding, depending on the sequence of the amino acid chain, the polypeptide chain will fold into its lowest energy configuration [28]. This is known as the secondary structure. The two most common structures that appear in this stage are the α-helices and the β-sheets. Using the chaos game on the secondary structure can indicate any non-randomness of the structural elements in proteins. One way to achieve this is to divide the 20 amino acids into 4 groups based on different properties and assign each group to a corner. Some properties that have been considered for CGR include hydrophobicity, molecular weight, isoelectric point (pI), α propensity, and β propensity. For details, see [4, 10].

5. CGR of mathematical sequences. In the first-year course we gave in 2015 [6], we applied the chaos game representation to mathematical sequences found

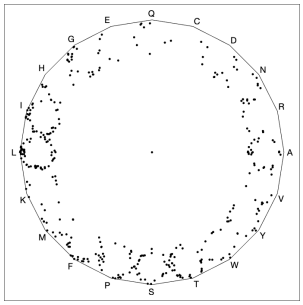
Table 2: Table of all 20 amino acids

Name	3 letter code	1 letter code
alanine	ala	A
arginine	arg	R
asparagine	asn	N
aspartic acid	asp	D
cysteine	cys	C
glutamine	gln	Q
glutamic acid	glu	E
glycine	gly	G
histidine	his	H
isoleucine	ile	I
leucine	leu	L
lysine	lys	K
methionine	met	M
phenylalanine	phe	F
proline	pro	P
serine	ser	S
threonine	thr	T
tryptophan	trp	W
tyrosine	tyr	Y
valine	val	V

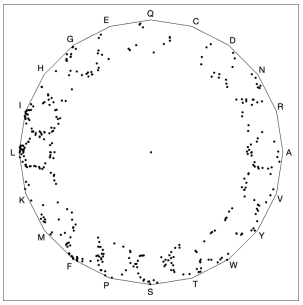
on the Online Encyclopedia of Integer Sequences (OEIS) [14]. In the following, we show some CGR experiments that came from the first-year course. Some sequences that were used from the Online Encyclopedia of Integer Sequences (OEIS) included digits of π (A000796), Fibonacci Numbers (A000045), prime numbers (A000040), and some from the continued fractions section (which were introduced at the beginning of the course). We will not only look at the visualizations created in the first-year class, but we will also determine whether these experiments are considered to be good visual representations. Note that for the following 4-vertex examples, the vertices are labeled from "0" to "3" starting from the bottom left corner of the plot going clockwise (i.e. "0" corresponds to the coordinate point $(-1,-1)$, "1" corresponds to the point $(-1,1)$, etc.).

5.1. Digits of pi. We first applied CGR to the digits of π (A000796 from the OEIS[4]), since we believe that the sequence of the digits are random. Since we focused on the 4-vertex CGR in the course, we naïvely took the digits of π modulo 4 and applied it to the 4-vertex CGR. The result of this is shown in Figure 16a. Since we assumed that the digits of π are random, this means that the visualization should be similar to the square-shaped CGR of random integer sequences (Figure 6a). However, we can see from Figure 16a that this is not the case; instead of a uniform covering of the space, we can see a pattern of vertical lines occurring. Quantitatively, the DSSIM index between Figures 16a and 6a is 0.9851, which indicates that they are not similar. The reason for this is because when we take 10 mod 4, the values are distributed to the corners unevenly: our 0 and 1 vertices would have one extra value each, and thus

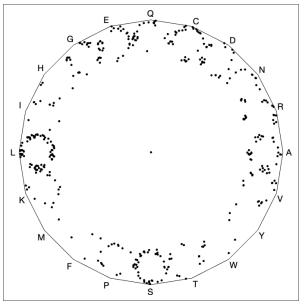
[4] https://oeis.org/A000796



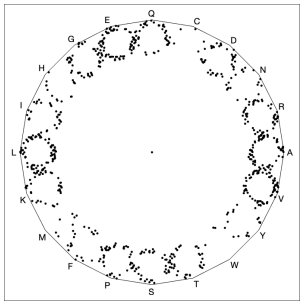
(a) cytochrome b (mitochondrion) [Homo sapiens] (GenBank: ASY00349.1)



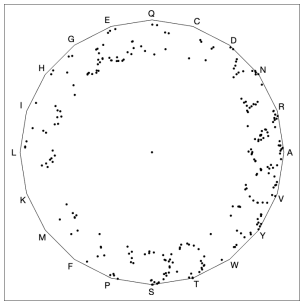
(b) cytochrome b (mitochondrion) [Mus musculus] (GenBank: NP_904340.1)



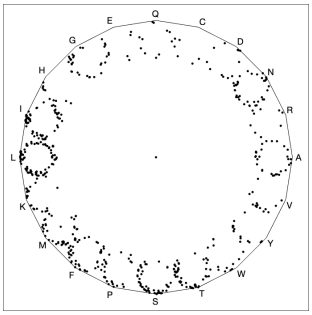
(c) ribonuclease inhibitor [Homo sapien] (GenBank: NP_976317.1)



(d) major vault protein [Rattus norvegicus] (GenBank: NP_073206.2)



(e) pertussis toxin subunit 1 [Bordetella pertussis Tohama I] (GenBank: NP_882282.1)



(f) NADH dehydrogenase subunit 5 (mitochondrion [Mus musculus] (GenBank: NP_904338.1)

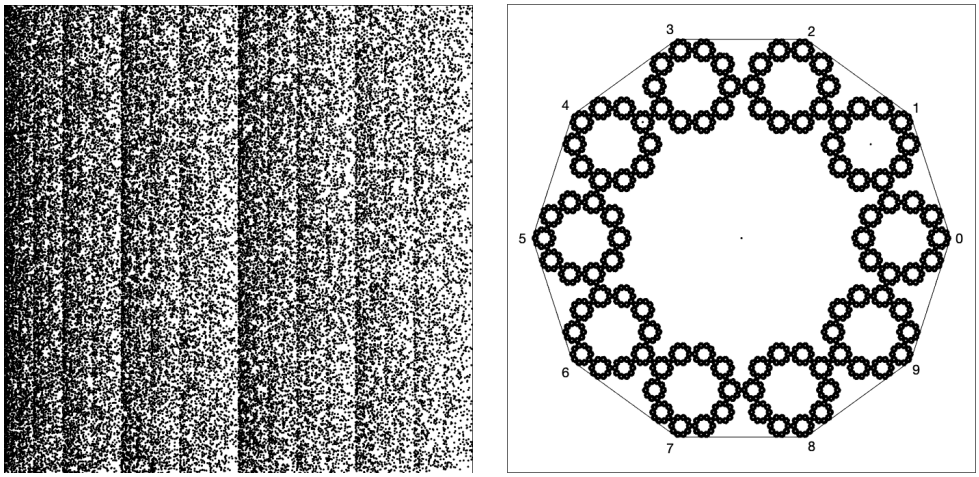
Fig. 15: Illustration of the chaos game representation of proteins (specifically using their amino acids).

why our CGR of the digits of π does not match the CGR of random integers even though we know for a fact that the digits of π are indeed random. Due to this reason, Figure 16a is not a good visual representation of the digits of π.

Another CGR to visualize the digits of π is shown in Figure 16b. We use a 10-vertex CGR ($r = 0.763932$) where each vertex represents an integer from 0 to 9. From this figure, we can see that the points produce a fractal pattern of the decagon, which suggests that the sequence is random.

5.2. Fibonacci Sequence. We also created a visualization of the Fibonacci Sequence (A000045 from the OEIS). One example of a visualization of this sequence is shown in Figure 17. It is a 10-vertex CGR of the Fibonacci sequence modulo 10. We can see from the figure that most of the numbers of the Fibonacci sequence are even. Other visualizations for the Fibonacci sequence can be done, such as taking the modulo of the numbers using a different divisor and using a CGR with different number of vertices.

However, since the Fibonacci sequence grows very quickly, the students learned about the limitations of floating point since they were unable to take more than the



(a) 4 vertex-CGR of 50000 digits of π (b) 10 vertex-CGR of 100000 digits of π

Fig. 16: CGR of the digits of π

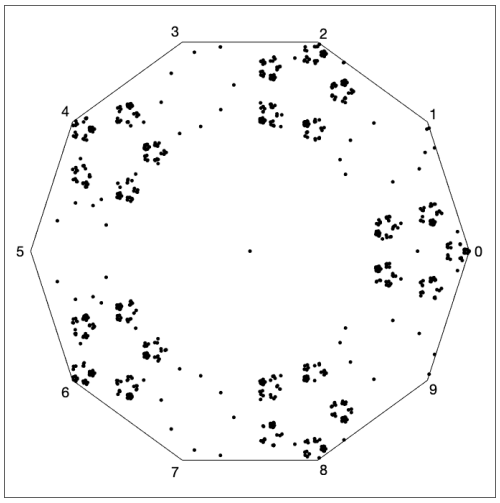


Fig. 17: 10 vertex-CGR of the first 3000 Fibonacci sequence modulus 10

first 3000 Fibonacci numbers before the computer registering the number as infinity.

5.3. Prime numbers. Following the visualization of the digits of π and the Fibonacci sequence using the chaos game, we looked at another popular sequence: the sequence of prime numbers (A000040 from the OEIS[5]). We looked at four different ways of creating the CGR for prime numbers.

We first visualized the prime numbers in a similar way as Figures 16b and 17 by taking the sequence of prime numbers modulo 10. This is shown in Figure 18a. We

[5] https://oeis.org/A000040

notice that almost all of the prime numbers ends with a 1, 3, 7, or 9. Because of this observation, we decided to create a square CGR labelling the vertices with these values. The result is shown in Figure 18b.

Figure 18c shows the third visualization of prime numbers: a CGR of the $103+k$th prime numbers mod 4, a diagonal line which spans from the "1" corner to the "3" corner. As we all know, prime numbers larger than 2 will never be even, so the corner points "0" and "2" which are representative "even" values would never occur, thus creating a straight diagonal line. If we had reassigned the values the coordinate points represent, the plot would have turned out differently; instead of a diagonal line, it could possibly be a horizontal (spanning from coordinate points $(-1,-1)$ to $(1,-1)$ or from $(-1,1)$ to $(1,1)$) or vertical line (spanning from $(-1,1)$ to $(-1,-1)$ or $(1,1)$ to $(1,-1)$), or a diagonal line the spanning the other two vertices. One needs to be mindful of how the coordinate points are assigned.

Lastly, we can visualize the prime numbers in a different way, as seen in Figure 18d. Here, we took the sequence of prime numbers less than one million and took mod 8 of the numbers. We thought that this would be interesting because taking any prime number larger than 2 mod 8 will only result in the following numbers: 1, 3, 5, and 7. Because of this, we relabelled the vertices to "1", "3", "5", and "7" starting from the bottom left corner, going clockwise (i.e. the coordinate point $(-1,-1)$ is labelled "1", $(-1,1)$ is labelled "3", etc.), and plotted the points according to the chaos game ($r = 0.5$). Surprisingly, this gives us a pattern, apparently meaning that there is a pattern in the sequence of prime numbers mod 8. However, compare this to Figure 6a, which, though random, showed patterns. This "evidence of pattern" is not conclusive! This is possibly simply a Procrustean effect, and we'd have to know more about the distribution of primes to say more.

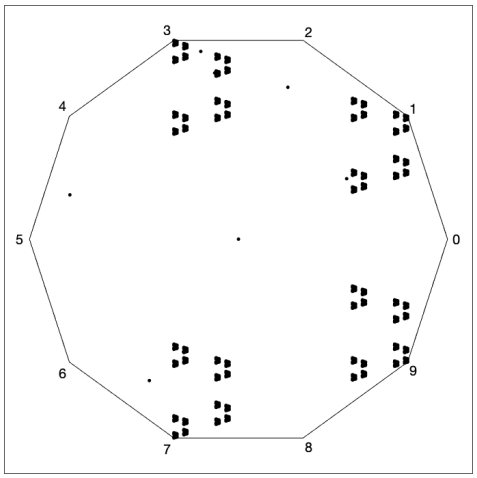
5.4. Partial quotients of continued fractions. We then explored visualizing the sequences taken from the partial quotients of continued fractions, which were taught at the beginning of the course. Continued fractions are fractions written in the following general form [26, 7]:

$$a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{a_3 + \cfrac{1}{a_4 + \cfrac{1}{\ddots}}}}}, \tag{9}$$

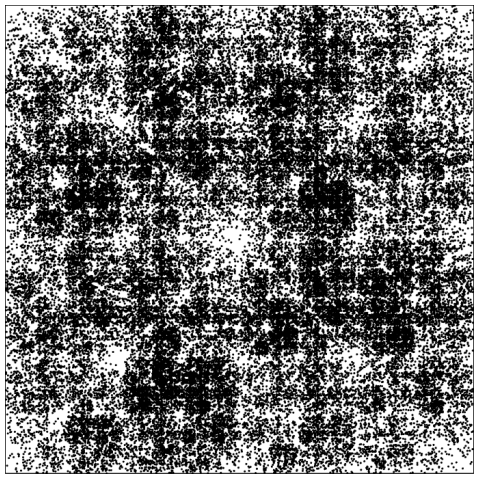
which produces a sequence of numbers, $a_0 + [a_1, a_2, a_3, a_4, \cdots]$, called the partial quotients of the continued fraction. This can be demonstrated with an example: we can rewrite $9/7$ in the form

$$\frac{9}{7} = 1 + \frac{2}{7} = 1 + \frac{1}{7/2} = 1 + \frac{1}{3 + 1/2} = 1 + \cfrac{1}{3 + \cfrac{1}{1 + 1/1}}. \tag{10}$$

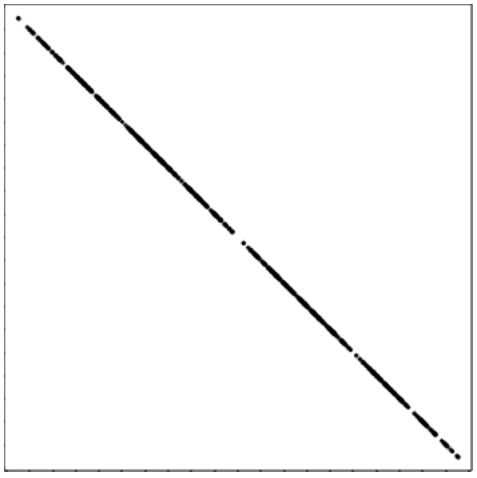
Here, the partial quotients of $9/7$ are the elements of $1 + [3, 1, 1]$. The students then looked at the continued fractions of other numbers, such as $\sqrt{2}$, e and π, where the sequences of their partial quotients were used in chaos game representations.



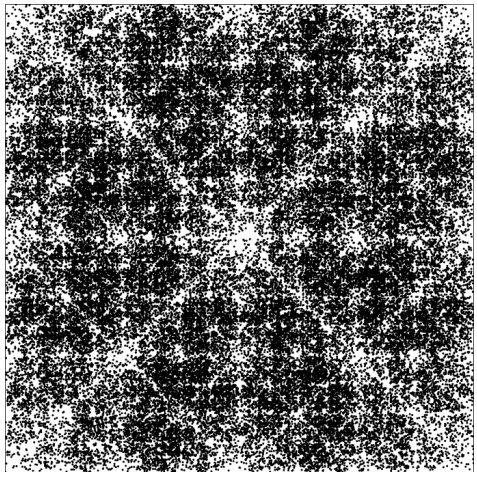
(a) All prime numbers less than 1000000 taken mod 10



(b) All prime numbers between 7 and 1000000 taken mod 10 where the vertices are 1, 3, 7, 9 clockwise from the bottom left



(c) $103 + k$th prime mod 4



(d) All prime numbers between 7 and 1000000 taken mod 8 where the vertices are 1, 3, 5, 7 clockwise from the bottom left

Fig. 18: Examples of CGRs of prime numbers

One recurring theme of the experimental mathematics course was $\sqrt{2}$, so with this in mind, some students jumped on the opportunity to plot the chaos game representation of the quotients of the continued fraction of $\sqrt{2}$. As we had seen earlier in the course, the sequence goes like

$$1 + [2, 2, 2, 2, \ldots, 2] .$$

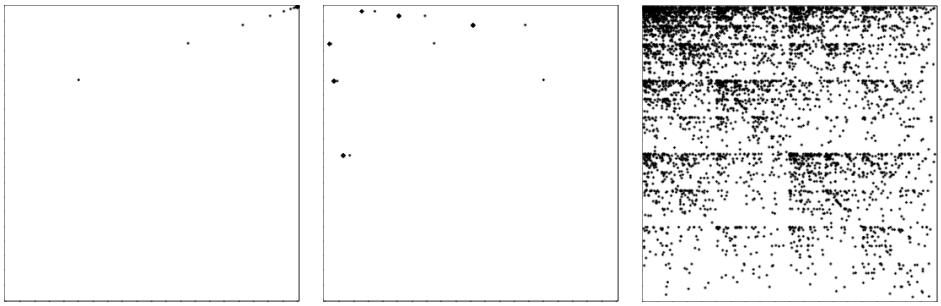
Because all elements of the sequence (apart from the first one) are 2's, it is not

surprising to see a (faint) diagonal line with most of the points in the upper right corner, shown in figure 19a. The students also tried the partial quotients of e, which is equal to

$$2 + [1, 2, 1, 1, 4, 1, 1, 6, \cdots] \tag{11}$$

[27], shown in figure 19b. What is seen here makes sense as the sequence mostly contains 1's and these alternates with even values, so in this case, either 0 or 2. Therefore, it is clear that there are no points in the lower right corner since that represents the value 3. Unfortunately, these two plots do not look all that impressive; in fact, it is pretty underwhelming. Disappointed with this result, the students decided to experiment with other sequences in which all four coordinates occur.

The students then thought, "Why not take the partial quotients of the continued fraction of π? The results *must* be random." As shown in figure 19c, unexpectedly, there is indeed a pattern, which shows that the sequence of partial quotients of the continued fraction of π is not as random as we thought at the beginning of the course. This showed the students that the distribution is not uniform.



(a) Partial quotients of continued fraction of $\sqrt{2}$ mod 4

(b) Partial quotients of continued fraction of e mod 4

(c) Partial quotients of continued fraction of π mod 4

Fig. 19: Examples of CGR for partial quotients of continued fractions of $\sqrt{2}$ (19a), e (19b), and π (19c).

In fact, the distribution of partial quotients is very well known, owing to the results of Khinchin [21]. As a very startling aside, recently, Bill Gosper has remarked on an amazing identity

$$\prod k!^{\Delta_3 \ln k} = K^{\ln 2}\,., \tag{12}$$

where K is Khinchin's constant. We did not inform our students of Khinchin's remarkable results because CGR was only done at the end of the course. Next time, perhaps!

6. Random vs Pseudorandom. The theory of probability arose much later in the development of science than the theory of dynamical systems and of exact trajectories, and therefore it may be supposed to be more difficult to grasp. There are many different works on the fundamentals of probability which discuss this difficulty; see for instance [13], but also [17]. We believe that combining deterministic dynamics with randomness is even more difficult; and that this, in fact, is the main business

nowadays of the applied mathematician. There have been, of course, millions of words written on the topic. We believe that it is crucial that entering students see some of the discussions of the fundamentals; they need to have a chance to grasp the deeper, most practical aspects of the theory. We believe that this module offers the instructor a chance to begin those discussions.

One important early part of the discussion is whether or not what the computer produces is "really" random. We alluded earlier to the fact that the poor quality of some early bad random number generators was detected by patterns arising in two-dimensional pictures. How are the patterns arising in the pictures in this paper different? Would they arise if "really" random numbers were used instead of the pseudorandom numbers generated by computer? [The answer is yes.] And what is the difference, anyway? There are knots here that the students (and professors) can tie themselves in: once a sequence of numbers is written down, however randomly it was generated, how can it be random, any more? It's now perfectly predictable! The most comforting words that we know about this come from Kolmogorov himself (quoted in full at the end of this paper) which we paraphrase as "it's only a model." Indeed see [23] for a brief discussion of a *mathematical* foundation for probability. This applies whether your sequence is "really" random, or only "pseudo" random.

These discussions are important, in a situation such as that of this paper, where we are trying to tease out deterministic aspects of *apparently* random sequences (or of sequences such as that of DNA which is surely influenced both by random events (mutation, horizontal gene transfer) and very non-random events (selection). We have used some "clearly" non-random (in some sense) sequences such as the digits of π to show that we can detect a signature of randomness there; we have used the same techniques to detect a signature of regularity. To be convinced, the students must be allowed to discuss these issues at some length. In particular, this may be the first time the students have encountered pseudorandom numbers.

There are several high-quality methods for uniform pseudorandom number generation: the most popular includes the *multiple-recursive method* and families formed by *shift-register methods* [8]. The multiple-recursive method is an extension of the linear congruential method by replacing the first-order linear recursion by one of higher order. One family in the shift-register method generates uniform pseudorandom numbers by means of linear recurring sequences modulo 2. Another uses vector recursions modulo 2 of higher order; this family includes the very popular Mersenne twister MT19937 [24], which is the default for MATLAB's pseudorandom number generating functions, which includes `rand` (uniformly distributed random number), `randn` (normally distributed random numbers), and `randi` (uniformly distributed pseudorandom integers) [25]. Other pseudorandom number generators in MATLAB include SIMD-oriented Fast Mersenne Twister, Combined Multiple Recursive, Multiplicative Lagged Fibonacci and Legacy MATLAB generators. Users can use MATLAB's `rng` function to control which generator to use, along with the seed for the pseudorandom number generator to produce a predictable sequence of numbers.

As mentioned above, one of the pseudorandom number generators MATLAB's `rand` function uses is the Mersenne twister MT19937 to generate uniform pseudorandom numbers. The MT19937 produces sequences of uniform pseudorandom numbers with period length $2^{19937} - 1$ that possess 623-dimensional equidistribution up to 32 bits accuracy [8]. This would seem to create a better "random" outcome in comparison to an actual sequence of die rolls according to some measures. However, random.org claims that their "dice roller" is, for many purposes, better than pseudorandom number algorithms typically used in computer programs. The randomness in

their program comes from atmospheric noise.

Using a pseudorandom number generator differs to using dice or the atmospheric-sourced generators at random.org. One gets different results due to the different methods that are used. Now, physically-based systems that we believe are random are sometimes more problematic than we think. For instance, rolling a die is not quite as random as one would expect. Stein's article [29] in Inside Science states that dice rolls are not completely random: the initial position of the (fair) die affects the outcome of the die roll. And is anything in the atmosphere truly random? Or is this just a statement of our ignorance? Kolmogorov's words—it's only a model—are comforting here. But they help with the use of pseudorandom numbers, as well.

In practice, pseudorandom number generators are indispensable for working with probability, and an enormous amount of work has gone into making them of very high-quality indeed. We know of no way to distinguish—in practice—results from these good generators from results given by physically-based generators that nearly everyone believes are "really" random.

7. Concluding Remarks. This module can be used to teach students about a useful visualization of sequences, in which case the emphasis from the instructor could be on interpreting the results, perhaps by using structural similarity or distance maps. The module can also be used to teach elementary programming techniques, in which case the instructor can emphasize programming tools, correctness, efficiency, style, or analysis.

Using elementary sequences of integers makes the module accessible even to first-year students, avoiding difficult biological details: but even so, students are exposed to the deep concepts of pattern and randomness more or less straight away. We have only begun a discussion of what it means to be random, and what it means to have a pattern. Randomness in a model can be a way to hide our ignorance, or it can be a profound statement of our understanding. Modelling viral evolution using randomness [30], where mutations occur in large populations, is clearly warranted; similarly for the vast lengths of DNA sequences. For prime numbers, in one sense clearly not: primes cannot be random, even though they behave in some ways as if they are.

> "Such considerations may be repeated as often as we like, but it is clear that this procedure will never allow us to be free of the necessity, at the last stage, of referring to probabilities in the primitive imprecise sense of this term.
>
> It would be quite wrong to think that difficulties of this kind are peculiar in some way to the theory of probability. In the mathematical investigation of actual events, we always make a model of them. The discrepancies between the actual course of events and the theoretical model can, in its turn, be made the subject of mathematical investigation. But for these discrepancies we must construct a model that we will use without formal mathematical analysis of the discrepancies which again would arise in it in actual experiment."
>
> — Andrey Kolmogorov [1]

Acknowledgments. We would like to thank Lila Kari for introducing us to the chaos game representation. We also like to thank the students of the second run of the senior experimental mathematics course for their input and ideas on this topic. We would like to acknowledge financial support from The University of Western Ontario (aka Western University), NSERC, and OGS.

REFERENCES

[1] A. D. Aleksandrov, M. A. Lavrent'ev, et al., *Mathematics: Its content, methods and meaning*, Courier Corporation, 1999.

[2] J. S. Almeida and S. Vinga, *Biological sequences as pictures–a generic two dimensional solution for iterated maps*, BMC bioinformatics, 10 (2009), p. 100.

[3] M. F. Barnsley, *Fractals everywhere*, Academic press, 2014.

[4] S. Basu, A. Pan, C. Dutta, and J. Das, *Chaos game representation of proteins*, Journal of Molecular Graphics and Modelling, 15 (1997), pp. 279–289.

[5] I. Borg and P. J. F. Groenen, *Modern multidimensional scaling: Theory and applications*, Springer Science & Business Media, 2005.

[6] E. Y. S. Chan and R. M. Corless, *A random walk through experimental mathematics*, in Springer Proceedings in Mathematics & Statistics, Springer International Publishing, 2020, pp. 203–226, https://doi.org/10.1007/978-3-030-36568-4_14, https://doi.org/10.1007/978-3-030-36568-4_14.

[7] R. M. Corless, *Continued fractions and chaos*, The American mathematical monthly, 99 (1992), pp. 203–215.

[8] M. R. Dennis, P. Glendinning, P. A. Martin, F. Santosa, and J. Tanner, *The Princeton companion to applied mathematics*, Princeton University Press, 2015.

[9] D. P. Feldman, *Chaos and fractals: an elementary introduction*, Oxford University Press, 2012.

[10] A. Fiser, G. E. Tusnady, and I. Simon, *Chaos game representation of protein structures*, Journal of molecular graphics, 12 (1994), pp. 302–304.

[11] Y. Fisher, M. McGuire, R. F. Voss, M. F. Barnsley, R. L. Devaney, and B. B. Mandelbrot, *The science of fractal images*, Springer Science & Business Media, 2012.

[12] N. Goldman, *Nucleotide, dinucleotide and trinucleotide frequencies explain patterns observed in chaos game representations of DNA sequences*, Nucleic Acids Research, 21 (1993), pp. 2487–2491.

[13] R. W. Hamming, *Art of Probability*, Addison Wesley Publishing Company, 1991.

[14] O. F. Inc., *The on-line encyclopedia of integer sequences*. https://oeis.org/, 2019.

[15] H. J. Jeffrey, *Chaos game representation of gene structure*, Nucleic Acids Research, 18 (1990), pp. 2163–2170.

[16] H. J. Jeffrey, *Chaos game visualization of sequences*, Computers & Graphics, 16 (1992), pp. 25–33.

[17] M. Kac, *Probability and related topics in physical sciences*, vol. 1, AMS, 1959.

[18] R. Karamichalis, L. Kari, S. Konstantinidis, and S. Kopecki, *An investigation into inter-and intragenomic variations of graphic genomic signatures*, BMC bioinformatics, 16 (2015), p. 246.

[19] L. Kari, K. A. Hill, A. S. Sayem, N. Bryans, K. Davis, and N. S. Dattani, *Map of life: Measuring and visualizing species' relatedness with" molecular distance maps"*, arXiv preprint arXiv:1307.3755, (2013).

[20] L. Kari, K. A. Hill, A. S. Sayem, R. Karamichalis, N. Bryans, K. Davis, and N. S. Dattani, *Mapping the space of genomic signatures*, PloS one, 10 (2015), p. e0119815.

[21] A. Khinchin and T. Teichmann, *Continued fractions*, Physics Today, 17 (1964), p. 70.

[22] D. E. Knuth, *Seminumerical Algorithms*, vol. 2 of The Art of Computer Programming, Addison Wesley, Reading, MA, 3 ed., 1997.

[23] A. N. Kolmogorov, *On logical foundations of probability theory*, in Probability theory and mathematical statistics, Springer, 1983, pp. 1–5.

[24] M. Matsumoto and T. Nishimura, *Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator*, ACM Transactions on Modeling and Computer Simulation (TOMACS), 8 (1998), pp. 3–30.

[25] C. B. Moler, *Random number generators, Mersenne Twiser*. In *Cleve's Corner: Cleve Moler on Mathematics and Computing*. Available at https://blogs.mathworks.com/cleve/2015/04/17/random-number-generator-mersenne-twister/, 2015.

[26] C. D. Olds, *Continued fractions*, vol. 18, Random House New York, 1963.

[27] C. D. Olds, *The simple continued fraction expansion of e*, American Mathematical Monthly, (1970), pp. 968–974.

[28] I. Simon, L. Glasser, and H. A. Scheraga, *Calculation of protein conformation as an assembly of stable overlapping segments: application to bovine pancreatic trypsin inhibitor.*, Proceedings of the National Academy of Sciences, 88 (1991), pp. 3661–3665.

[29] B. P. Stein, *Dice rolls are not completely random*. Available at https://www.insidescience.org/news/dice-rolls-are-not-completely-random (September 22, 2012).

[30] L. M. Wahl and T. Pattenden, *Prophage provide a safe haven for adaptive exploration in temperate viruses*, Genetics, 206 (2017), pp. 407–416.
[31] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, *Image quality assessment: from error visibility to structural similarity*, IEEE trans. image process., 13 (2004), pp. 600–612.
[32] F. Wickelmaier, *An introduction to MDS*, Sound Quality Research Unit, Aalborg University, Denmark, 46 (2003).