SIAM Review
Vol. 65, No. 1, pp. 261–290

© 2023 Society for Industrial and Applied Mathematics

Downloaded 11/06/25 to 103.178.194.193 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

Chaos Game Representation*

Eunice Y. S. Chan†
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, and CGR was able to uncover patterns in DNA or proteins in them that were previously unknown. We also show how CGR might be introduced in the classroom, either in a modeling course or in a dynamical systems course. Some sequences that are tested are taken from the On-line Encyclopedia of Integer Sequences, and others are taken from sequences that arose mainly from a course in experimental mathematics.

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

MSC codes. 28A80, 65P20

DOI. 10.1137/20M1386438

Contents

*Received by the editors December 16, 2020; accepted for publication (in revised form) February 22, 2022; published electronically February 9, 2023.
https://doi.org/10.1137/20M1386438
Funding: This work was supported by The University of Western Ontario (a.k.a. Western University), NSERC (grant RGPIN-2020-06438), and Ontario Graduate Scholarship (OGS).

†School of Medicine, The Chinese University of Hong Kong, Shenzhen, Guangdong, P. R. China (eunicechan@cuhk.edu.cn).
‡Department of Computer Science, Western University, London, ON, Canada, and Cheriton School of Computer Science, University of Waterloo, Waterloo, ON, Canada (rcorless@uwo.ca).

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 11/06/25 to 103.178.194.193 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

1. Introduction.

> A heavy warning used to be given [by lecturers] that pictures are not rigorous; this has never had its bluff called and has permanently frightened its victims into playing for safety. Some pictures, of course, are not rigorous, but I should say most are (and I use them whenever possible myself).
>
> —J. E. Littlewood [24, p. 53]

Finding hidden patterns in long sequences can be both difficult and valuable. Representing these sequences in a visual way can often help. The human brain is very adept at detecting patterns visually. Indeed, it is sometimes *too* adept: when humans detect spurious patterns that *aren't* really there, it's called *pareidolia*. That human tendency is what originally provoked the "heavy warning" against pictures that Littlewood complained about above. Nonetheless, pictures can help a lot, to generate conjectures and sometimes even indicate a path toward a proof, which might "only" be an experimental proof using comparison with data.

The so-called chaos game representation (CGR) of a sequence of integers is a particularly useful technique for pattern detection that visualizes a one-dimensional sequence in a two-dimensional space. The CGR is presented as a scatter plot, most frequently on a square in which each corner represents an element that appears in the sequence. The results of these CGRs can look like fractals, but even so can be visually recognizable and distinguishable from each other. Interestingly, the distinguishing characteristics can sometimes be made *quantitative*, with a good notion of "distance between images." This is an indirect way of constructing an equivalent notion of "distance between sequences," but it turns out to be simpler to think about.

Many applications, such as the analysis of DNA sequences [13, 16, 17, 19], e.g., the DNA of human beta globin region on chromosome 11 (see Figure 1), and protein structure [5, 11], have shown the usefulness of CGR and the notion of "distance between images"; 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 indicate to us, or sometimes even tell us rigorously, 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, at first we use the word "random" very loosely, on purpose. Indeed, we will use several words loosely,

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

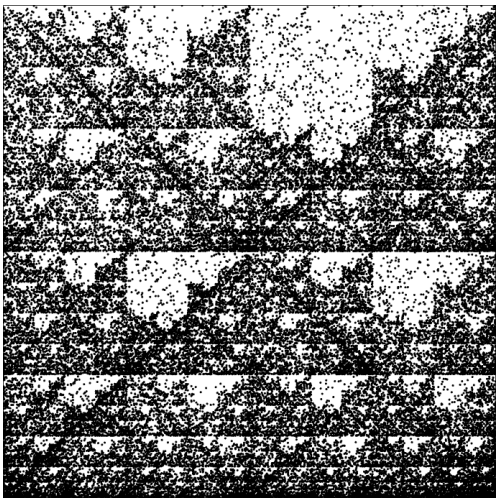


Fig. 1 *DNA of human beta globin region on chromosome* 11 *(HUMHBB)—*73308 *bp.*

such as "fractal" and "attractor" or "attracting set." We promise to provide references at the end that will provide more careful definitions and tighten up the discussion.

> 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 [14]

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

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

$$w(x_1, x_2) = (ax_1 + bx_2 + e, cx_1 + dx_2 + f) \ , \tag{2.1}$$

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

$$\begin{aligned} w(x) = w\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} &= \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} e \\ f \end{bmatrix} \\ &= Ax + t \, . \end{aligned} \tag{2.2}$$

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

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} = \begin{bmatrix} r_1 \cos\theta_1 & -r_2 \sin\theta_2 \\ r_1 \sin\theta_1 & r_2 \cos\theta_2 \end{bmatrix} \ , \tag{2.3}$$

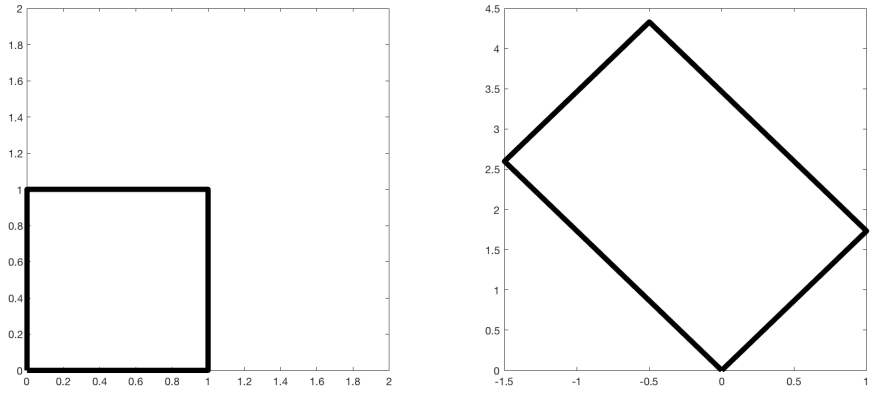
where (r_1, θ_1) are the polar coordinates of the point (a, c) and $(r_2, (\theta_2 + \pi/2))$ are the polar coordinates of the point (b, d). An example of a *linear* transformation

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \to A \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \tag{2.4}$$

Downloaded 11/06/25 to 103.178.194.193 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 11/06/25 to 103.178.194.193 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy



(a) Shape before transformation (b) Shape after transformation

Fig. 2 *Transformation of a unit square using the transformation matrix A (as shown in* (2.3)*), where* $r_1 = 2$*,* $r_2 = 3$*,* $\theta_1 = \pi/3$*, and* $\theta_2 = \pi/6$*.*

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)[1] is a finite set of contraction mappings on a metric space. These can be used to create pictures 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:

$$
\begin{aligned}
w_1(x,y) &= \begin{bmatrix} 1/2 & 0 \\ 0 & 1/2 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \\
w_2(x,y) &= \begin{bmatrix} 1/2 & 0 \\ 0 & 1/2 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} 0 \\ 1/2 \end{bmatrix}, \\
w_3(x,y) &= \begin{bmatrix} 1/2 & 0 \\ 0 & 1/2 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} 1/2 \\ 1/2 \end{bmatrix},
\end{aligned} \tag{2.5}
$$

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 (2.2) and p represents the probability factor. The corresponding table for (2.5) can be found in Table 1.

To create a fractal picture using an 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

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

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Table 1 *Table representing the IFS from* (2.5).

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

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', []);
```

(x_0, y_0) to obtain our next point (x_1, y_1). We do this again many times (determined by the user) until we see a pattern. 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[2] the Sierpinski triangle [3, 10], whose three vertices are located at $(0, 0)$, $(0, 1)$, and $(1, 1)$.

IFSs can be quite versatile; a set of several attractors can be used to create different shapes and can even resemble real-life objects. An example of this is Barnsley's fern, which resembles the black spleenwort [3]. 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 4.

[2]It can be proved that in the limit of infinite computation for a random sequence of infinite length, the picture really is that of the Sierpinski triangle [4].

Downloaded 11/06/25 to 103.178.194.193 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

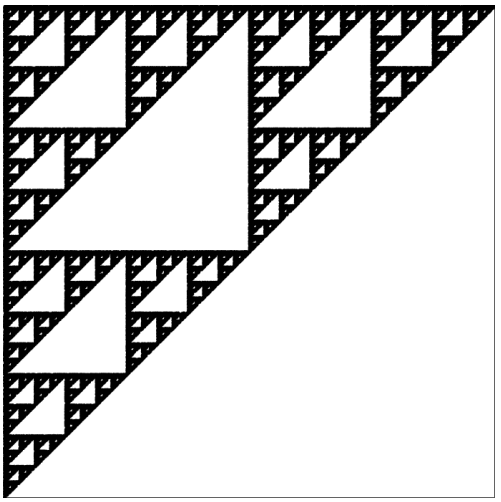


Fig. 3 *Results from Listing* 1. *We see visible regularity arising from a program that uses randomness.*

Fig. 4 *Barnsley's fern* ($n = 50000$).

3. Chaos Game. CGR is based on a technique from chaotic dynamics called the "chaos game,"[3] popularized by Barnsley [3] in 1988. The chaos game is an algorithm which allows one to produce pictures of fractal structures [16]. In its simplest form,

[3]The word "game" here does not refer to an adversarial game with choices and strategies as in *game theory*, but instead it treats sequences like the outcomes of a game of chance such as tossing coins. The use of "chaos" here happened because the biologist H. J. Jeffrey used the word, likely to indicate from which area of mathematics the ideas were taken from in general.

Downloaded 11/06/25 to 103.178.194.193 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

this can be demonstrated with a piece of paper and pencil using the following steps (adapted from [16]):

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. These will determine the "*gameboard*," where the picture will ultimately lie.
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 to the point with the corresponding number, and put a dot halfway 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;[4] 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 are labeled with the values 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 is 2, then we take the average between our initial point (0, 0) and the vertex that corresponds to 2, which in our case is $(1, 1)$. Our new point is $(0.5, 0.5)$. We then repeat this process until deciding to stop.

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

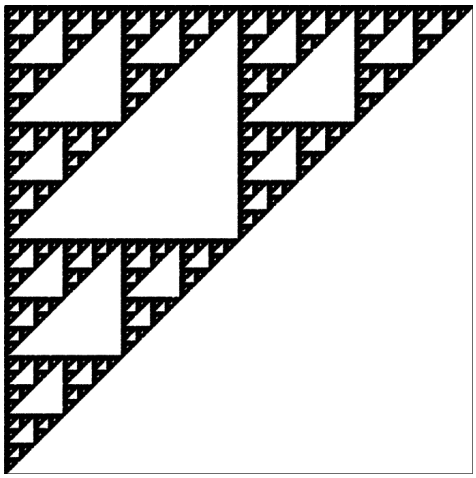
```
n = 50000;
pts = zeros(2, n);
rseq = randi([0, 2], 1, n-1);
for i = 2:n
  if rseq(i-1) == 0
    pts(:, i) = 0.5*(pts(:, i-1) + [0; 0]);
  elseif rseq(i-1) == 1
    pts(:, i) = 0.5*(pts(:, i-1) + [0; 1]);
  else
    pts(:, i) = 0.5*(pts(:, i-1) + [1; 1]);
  end
end
plot(pts(1, :), pts(2, :), 'k.',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

One might expect that the outcome of Listing 2 would show that the dots appear

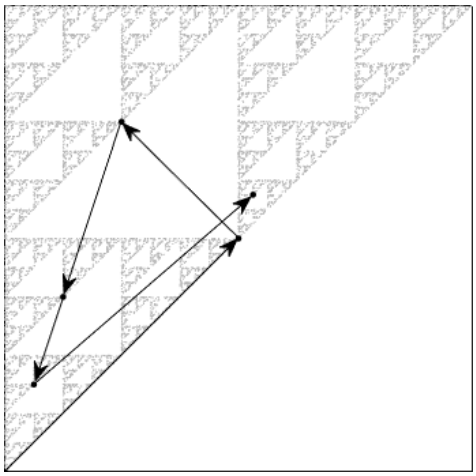
[4] We tried this, though not in class. We got up to 80 points on one figure, using the die roller on www.random.org. It's kind of fun, in a physical "drawing and measuring" way, but it's hard not to make mistakes. It could make a useful "active learning" exercise, and we might try it in class in the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

Downloaded 11/06/25 to 103.178.194.193 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.



(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.*

randomly across the gameboard; however, this is not the case: Figure 5(a) shows the result of the chaos game if it 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. However, what has happened here is that the *randomness* of the sequence has translated into a *uniform* distribution *on the fractal*. So, in essence, here the gameboard is the fractal! We will return to this point in the next section.

Figure 5(b) 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 into why this particular chaos game with three vertices gives 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 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. For another description of why this chaos game creates the Sierpinski triangle, see [12].

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

Downloaded 11/06/25 to 103.178.194.193 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 11/06/25 to 103.178.194.193 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

vertices (triangle) using a random number generator, but we are not limited to only this method. There are many variations of CGR, and some are more useful than others. In this section, we will look into what patterns arise when we perform CGR on different polygons, changing the placement of the ith point relative to the $(i-1)$th and its 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 CGR where the ith point is placed halfway between the $(i-1)$th point and the vertex corresponding to the ith base, for various polygons, using a sequence of 50000 uniformly generated (pseudo)random integers. As we can see from Figure 6(a), when the CGR is a square, there is no apparent pattern: the chaos game produced a square uniformly (to the eye) filled with points. This particularly conforms to our intuition: a random sequence should *look* random. The other figures from Figure 6 appear, in contrast, to exhibit patterns. These patterns are due to several parts of their attractors overlapping,[5] or failing to overlap at all, indicated by the uneven distribution of points (where points do appear); i.e., we can see some darker areas and some lighter areas and even gaps 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 one-dimensional sequences. We will see in section 3.1.3 that we can use other polygons (we will be referring to them as m-gons) for CGR to uncover patterns in sequences not with four elements by using the appropriate dividing rate r; that is, instead of placing a point halfway to the target vertex, we will place it at some other position along the line, with $0 < r < 1$. A proper choice of r will ensure that a random sequence will generate a uniform density of points, possibly on a fractal.

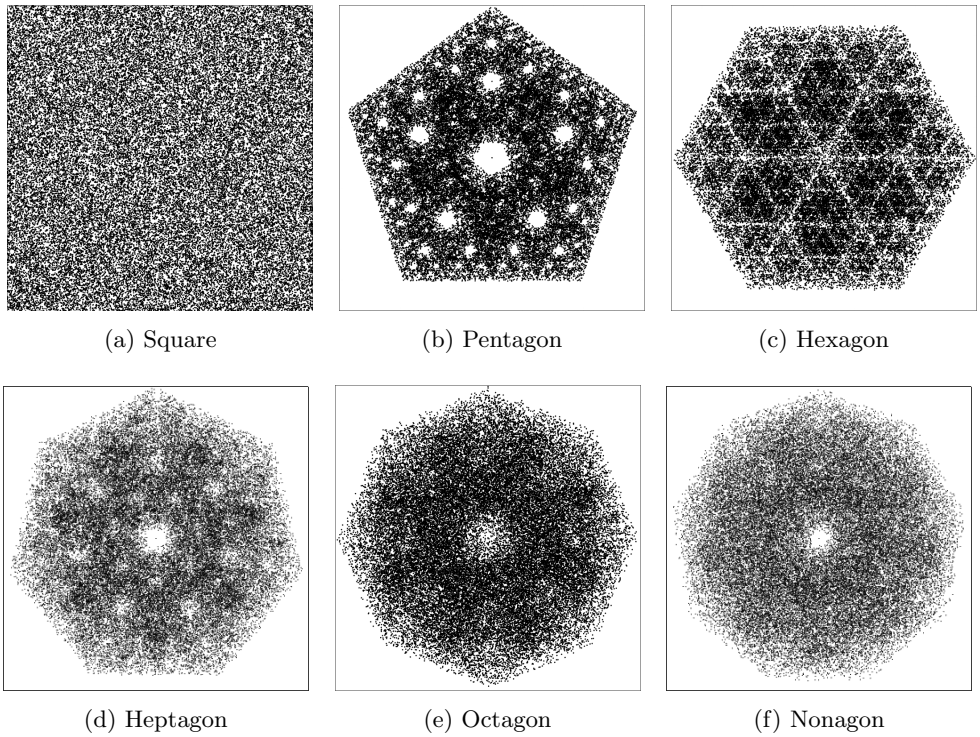
3.1.2. Dividing Rate. In the previous subsection, we saw different patterns appear for different shapes. We saw that shapes with more than four vertices seemed to have several parts of the attractor overlapping, producing visibly nonuniform density (some areas were darker than others). Here we will show that this attractor overlapping also happens when choosing a *small enough* dividing rate r.

To be explicit, 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 (or even outside, if you are willing to let the points land outside the convex hull of the vertices). 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 the distance between the new point and current point and the distance between the current point and the corresponding vertex. This is called the *dividing rate*, r. The new point p_{i+1} is thus $p_{i+1} = p_i + r(V - p_i)$. Therefore, when the new point is placed halfway between the current point and the vertex, the dividing rate is $r = 0.5$; 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 continue 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

[5]This is a loose way of thinking about it: we somehow imagine regions of the gameboard that attract elements of the sequence equally, and if these regions overlap, then we see spuriously dense regions on the gameboard. Alternatively, we could think (more rigorously) about the *probability measure* of the attractor (now really spread over the whole gameboard). One would want a "random" sequence to have a uniform measure across the gameboard.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.



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

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

Fig. 6 *Chaos game of polygons with different numbers of vertices.*

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 patterns appear, especially those with a dividing rate close to $r = 0.5$. More importantly, we see a nonuniformity of distribution in the square. In contrast, we can see that for $r = 0.6$ the points fill the corners of the plot area, but they are unevenly spaced (in comparison to what we have seen for $r = 0.5$). This suggests that when $r < 0.5$, the attractors are overlapping, indicated by the regions of the plot that are darker due to the points being more densely packed, and that when $r > 0.5$ the attractor lies on a fractal and is apparently uniformly distributed thereon. For r significantly less than 0.5, we can see that the CGR becomes more circular with the points spaced more closely together. This implies 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 rate $r < 0.5$). The patterns shown for the CGRs with a dividing ratio $r > 0.5$ resemble each other, as seen in Figure 7. Figure 7(f) looks like a smaller version of Figure 7(e), which, similarly, looks like a sparser version of Figure 7(d). In contrast to the CGR with the dividing rate $r = 0.5$, we can see that the points are spread apart. This could indicate that the attractors do not overlap and even do not touch each other. Although this can be used to distinguish any nonrandomness, it would be much more difficult to see with the human eye than when using the dividing rate $r = 0.5$.

Downloaded 11/06/25 to 103.178.194.193 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

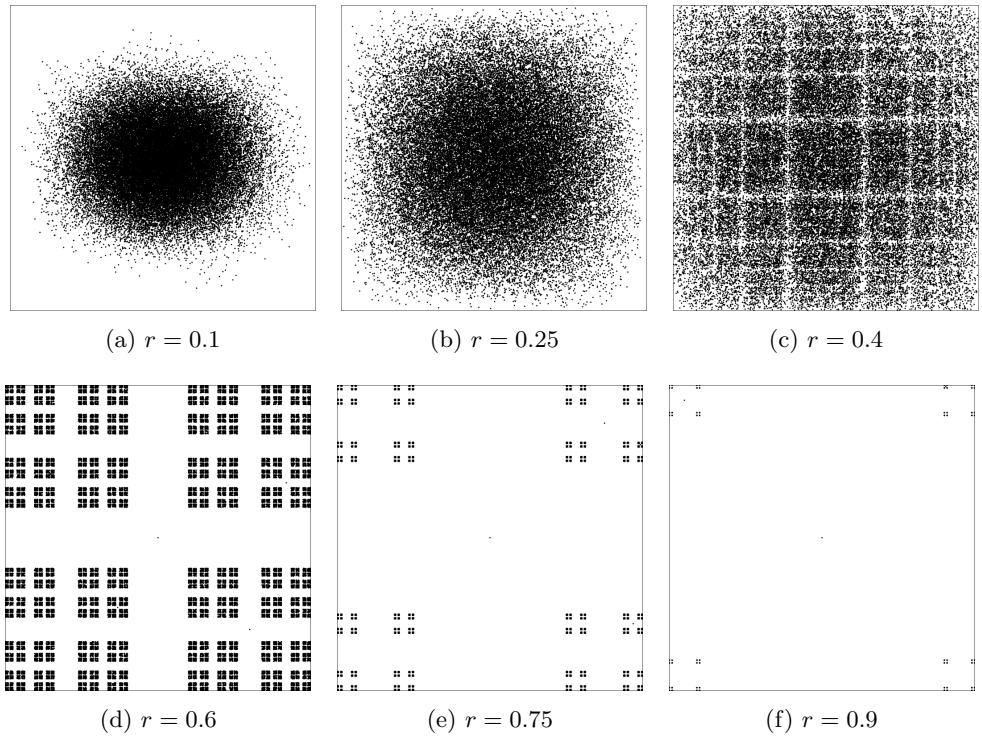


Fig. 7 *Illustration of how different values of r affect how a square CGR looks using* 50000 *points.*

For applications that need a number of vertices different from $m = 4$, it turns out to be optimal to find a dividing rate r for different polygons that can offer maximum packing of nested, nonoverlapping attractors, in order to help find a pattern in a given sequence. In the next subsection, we look at this optimal value of r as reported in the literature.

3.1.3. Generalization of CGR. In [11], Fiser, Tusnady, and Simon generalized CGR to be applicable for sequences of any number of elements $m > 2$. There are many ways of generalizing CGR, some of which are highlighted in [2], but the generalization the authors of [11] chose was to use an m-sided polygon (which they refer to as an m-gon), where m is the number of elements in a sequence that is represented [11]. As we saw in Figure 6, the attractors of the m-gons when $m > 4$ and $r = 0.5$ overlapped, as shown by the uneven distribution of the points. Because of this, the picture is ambiguous and is not as useful as a way to identify patterns in sequences. To combat this, Fiser, Tusnady, and Simon introduced a formula to calculate the dividing rate for m-gons for different values of m: by putting each attractor piece inside a nonoverlapping circle, they derived

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

Although the dividing rate calculated from this formula does prevent attractors from overlapping, Almeida and Vinga noticed in [2] that the resulting attractors are not optimally packed. For example, for $m = 4$, the solution gives a dividing rate $r =$

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 11/06/25 to 103.178.194.193 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

Downloaded 11/06/25 to 103.178.194.193 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

0.585786 instead of $r = 0.5$, which works perfectly well (and indeed is maybe better). In [2], the authors gave a more complicated formula for the dividing rate, which they say they derived by using basic trigonometry to pack polygons (instead of circles) into the outer m-gon. We have simplified their formula considerably as

$$r = \frac{\sin\left(\frac{(2k-1)\pi}{m}\right)}{\sin\left(\frac{(2k-1)\pi}{m}\right) + \sin\left(\frac{\pi}{m}\right)},$$

where

$$k = \text{round}\left(\frac{m+2}{4}\right).$$

If we use $k = (m+2)/4$ without rounding to an integer, this formula simplifies to the previous one. This fact was not noted in [2]. We note some special values: when $m = 3$ or $m = 4$, the formula gives $r = 0.5$. When $m = 5$, the formula gives $r = (\sqrt{5}-1)/2 = 0.618\ldots$. Various other m give algebraic values. When $(m+2)/4$ is halfway between integers, either choice of rounding for k gives the same r.

Figure 8 shows the generalized CGRs for various m-gons using the division formula from [11] (left) and [2] (right).

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

4. CGR of Biological Sequences. The CGR has been applied to biological areas. In this section, we will look at how the chaos game has been applied to DNA and protein (amino acids) sequencing and visualization. We also 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 a 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 that it might 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 CGR as a novel way of visualizing nucleotide sequences. This revealed previously unknown patterns in certain proteins [16, 17]. 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 for each iteration as we did in section 3, we follow the genetic sequence for which we want to create a CGR.

To demonstrate the chaos game for DNA sequences, let us walk through plotting the first five bases of the DNA sequence HUMHBB[6] (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 halfway between our initial point and the "G" corner. The next base in the sequence is "A," so we

[6]Full nucleotide sequence can be found here: https://www.ncbi.nlm.nih.gov/nuccore/U01317.1?report=fasta.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

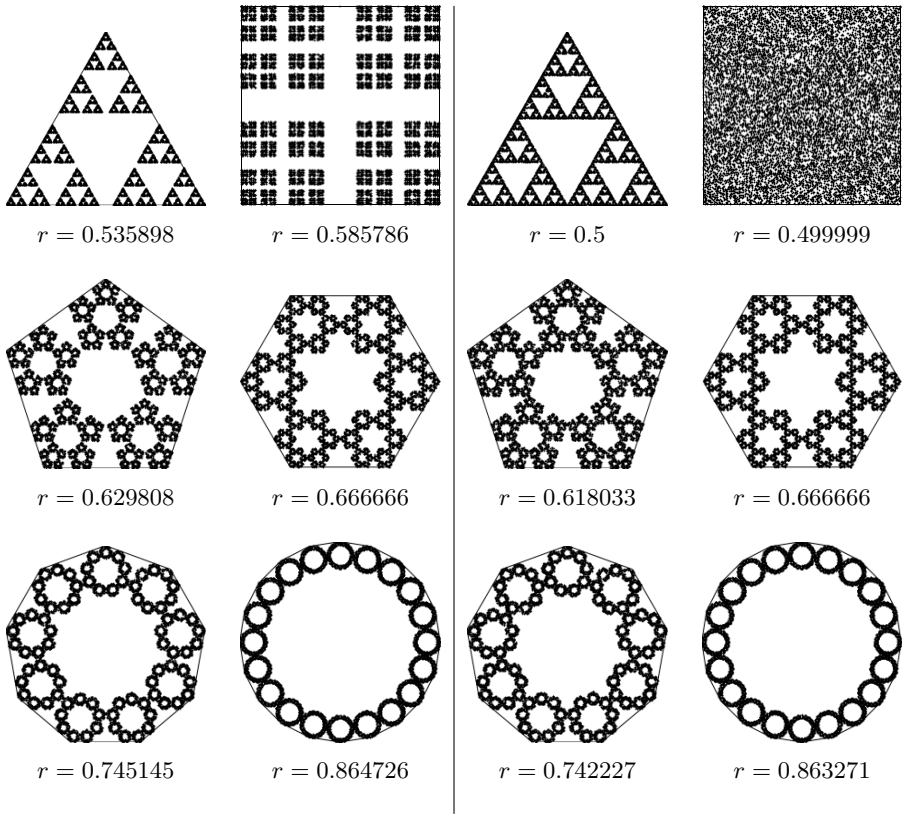


Fig. 8 *Graphical comparison of proposed CGR generalization using* 50000 *points. The left side uses the dividing rate formula from* [11]*, and the right side uses the dividing rate formula from* [2].

plot a point halfway between the point that we just plotted and the "A" corner. We leave the reader to continue the CGR for the rest of the HUMHBB DNA sequence.

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 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 we are examining [13]. Figure 10(a) shows the relationship between the corresponding area of the CGR and the DNA sequence. In reference to this figure, we can see that any point that corresponds with base G 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 subquadrants (labeling them in the same order as the quadrants) and, depending on where the point is, we can determine the previous base of the sequence. We can repeat this step again and again to find the order in which the bases appear in the sequence. Figure 10(b) 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

Downloaded 11/06/25 to 103.178.194.193 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

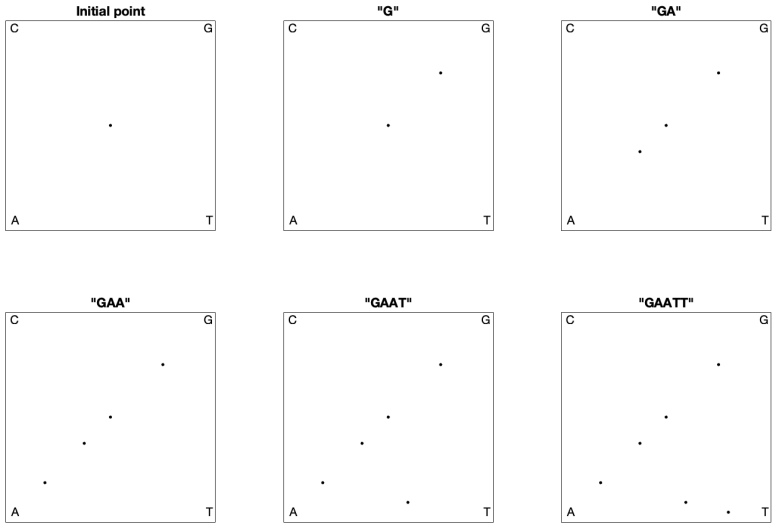
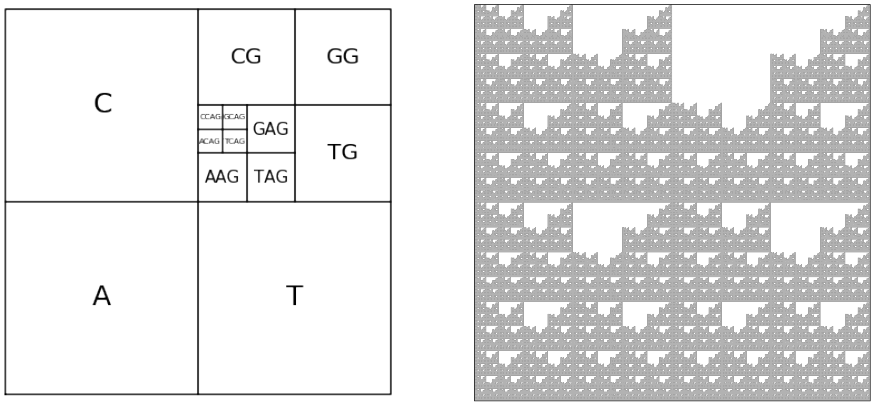


Fig. 9 *First five steps of producing CGR using HUMHBB DNA.*



(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

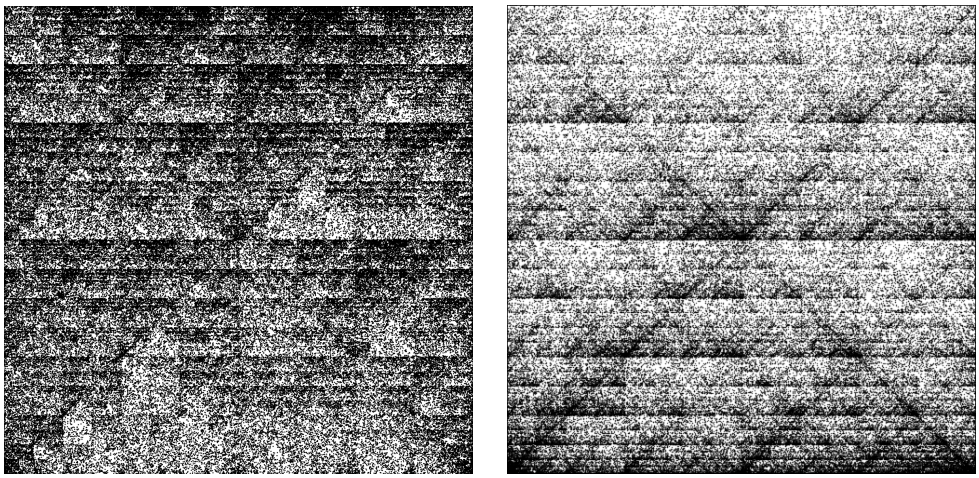
Fig. 10 *CGR quadrants to explain biological phenomena such as the double scoop pattern.*

sequence to produce it), we still see 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 further examples of CGRs for DNA sequences. We can see that Figure 11(a) (CGR of DNA sequence of human herpesvirus strain) also exhibits the double scoop pattern that we have seen for HUMHBB. This agrees with the earlier comment that it is common to not see C and G dinucleotides together within the

Downloaded 11/06/25 to 103.178.194.193 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.



(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.*

human genome. Figure 11(b), on the other hand, shows the CGR of the DNA sequence of the chloroplast of quinoa plant and does not exhibit a double scoop pattern. We can see that by 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 [20] 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 they used include CGR, the structural dissimilarity (DSSIM) index, and multidimensional scaling (MDS). In the paper [20], this method was applied to a variety of cases that have been historically controversial and it was demonstrated to have the potential for taxonomical clarification. In what follows, we will discuss the methods used from this paper, in particular the DSSIM index and MDS, since we have already looked into the CGR for nucleotides.

To understand the DSSIM index, we will first have to explain what the structural similarity (SSIM) index is. The 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 [34]. 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:

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

Downloaded 11/06/25 to 103.178.194.193 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 11/06/25 to 103.178.194.193 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

where

$$l(x,y) = \frac{2\mu_x\mu_y + C_1}{\mu_x^2 + \mu_y^2 + C_1}\,, \tag{4.1}$$

$$c(x,y) = \frac{2\sigma_x\sigma_y + C_2}{\sigma_x^2 + \sigma_y^2 + C_2}\,, \tag{4.2}$$

$$s(x,y) = \frac{\sigma_{xy} + C_3}{\sigma_x\sigma_y + C_3}\,, \tag{4.3}$$

and 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 $C_3 = {}^{C_2}/_2$, 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 a high value indicates high similarity. In [20], instead of calculating the overall SSIM index, the authors 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 indices 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 value.

Now that we know how to compute the SSIM index, we are able to compute the DSSIM index,

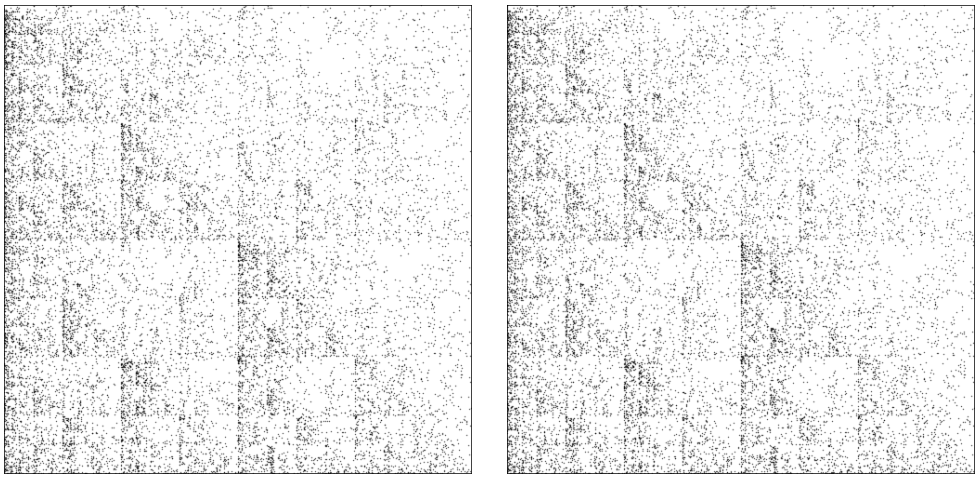
$$\mathrm{DSSIM}(x,y) = 1 - \mathrm{SSIM}(x,y)\,,$$

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 when one is completely white, while the other is completely black. Typically, the range that DSSIM falls into is $[0,1]$—the authors from [20] noted that almost all (over 5 million) distances 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, 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 (Figure 12(a)) and the Neanderthal (Figure 12(b)). We can see that the two CGRs are quite similar (which agrees with the fact that the two species are from the same genus) and, therefore, we expect the DSSIM index to be small. Using MATLAB's `ssim` function, the SSIM index between the two images is 0.8777, so the DSSIM 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 13(a)) and the pearlfish (Figure 13(b)). 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 indices DSSIM, we can produce what is called a distance matrix, which is a square, real, symmetric matrix. Each element of the distance matrix is the DSSIM index of

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.



(a) CGR of human (*Homo sapiens sapiens*) mtDNA—16569 bp

(b) CGR of Neanderthal (*Homo sapiens neanderthalensis*)—16565 bp

Fig. 12 *CGR of human and Neanderthal mitochondrial DNA.*

the two species corresponding with their respective row and column. The distance matrix for the four species that we have already produced the CGR for is

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

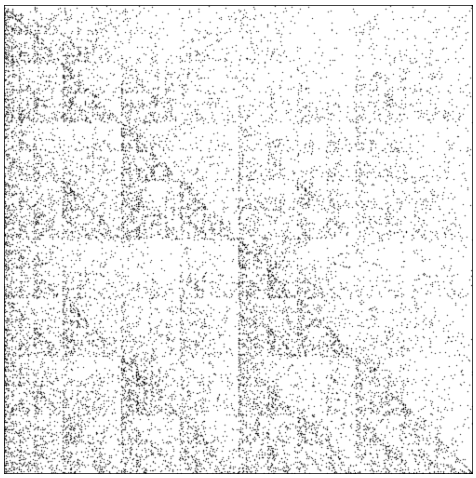
where the first row/column represents humans, the second row/column represents the Neanderthals, the third row/column represents the great spotted kiwi, and the fourth row/column represents the pearlfish. Obviously, the diagonal of the matrix is 0, since the DSSIM of two identical images produces a result of 0. As we have seen previously, $\mathrm{DSSIM}(\text{human}, \text{Neanderthal}) = 0.1223$, and vice versa. From this matrix, we can see that the CGR of the mtDNA of pearlfish is most different from the rest—and this agrees with what we have observed.

Multidimensional scaling (MDS) is a means of visualizing the level of similarity or dissimilarity of individual cases of a dataset. It has been used in various fields such as cognitive science, information science, pychometrics, marketing, ecology, social science, and other areas of study [6]. 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 [35]. In our case, MDS takes in the distance matrix and outputs a two-dimensional map, where each item is represented by a point. In [20], the authors used a classical MDS, which assumes that all the distances (from the distance matrix) are Euclidean. For the algorithm for classical MDS, we refer the reader to [35, p. 10]. MATLAB has its own built-in function that does MDS, called `cmdscale`.

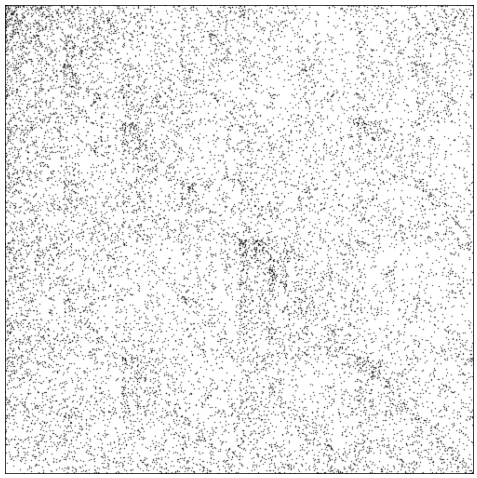
Figure 14 shows an example of a molecular distance map of 4844 animals from various classes of the chordate phylum (meaning that all these animals are vertebrate), colored according to their class. The sequences were taken from the NCBI Reference Sequence Database (RefSeq). We can see from the figure that this method (Map of

Downloaded 11/06/25 to 103.178.194.193 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.



(a) Great spotted kiwi (*Apteryx haastii*) mtDNA—16980 bp



(b) Pearlfish (*Carapus bermudensis*)—16613 bp

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

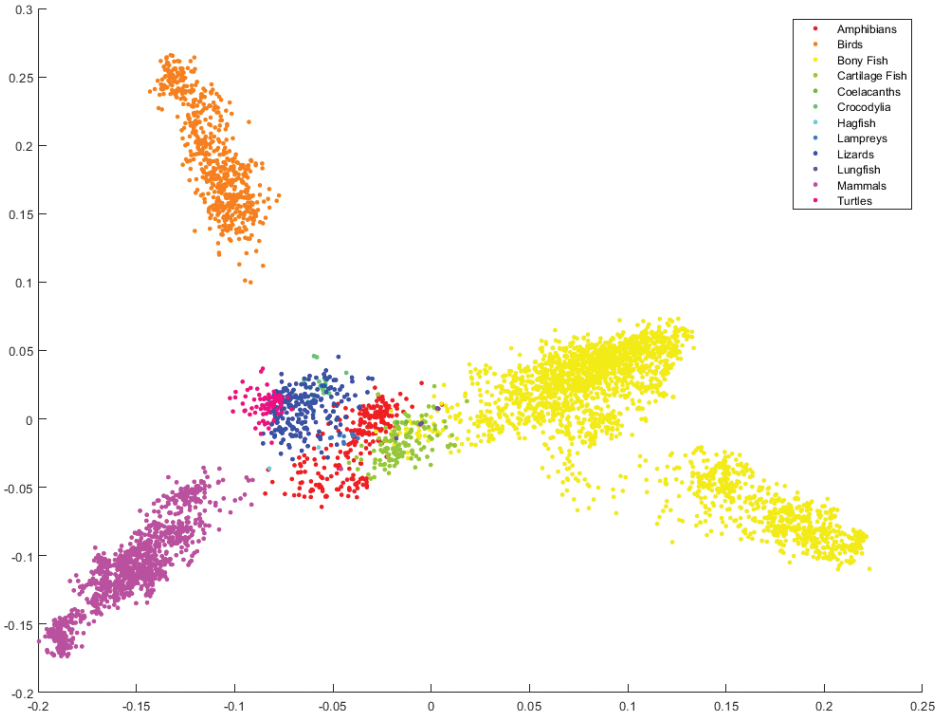


Fig. 14 *Molecular distance map of* 4844 *animals from various classes of the chordate phylum, colored according to their class.*

Downloaded 11/06/25 to 103.178.194.193 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

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

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 the location of the points representing the lungfish—they are in the area where the bony fish, cartilage fish, and amphibians touch. This is truly remarkable as the lung fish has qualities of all these classes.

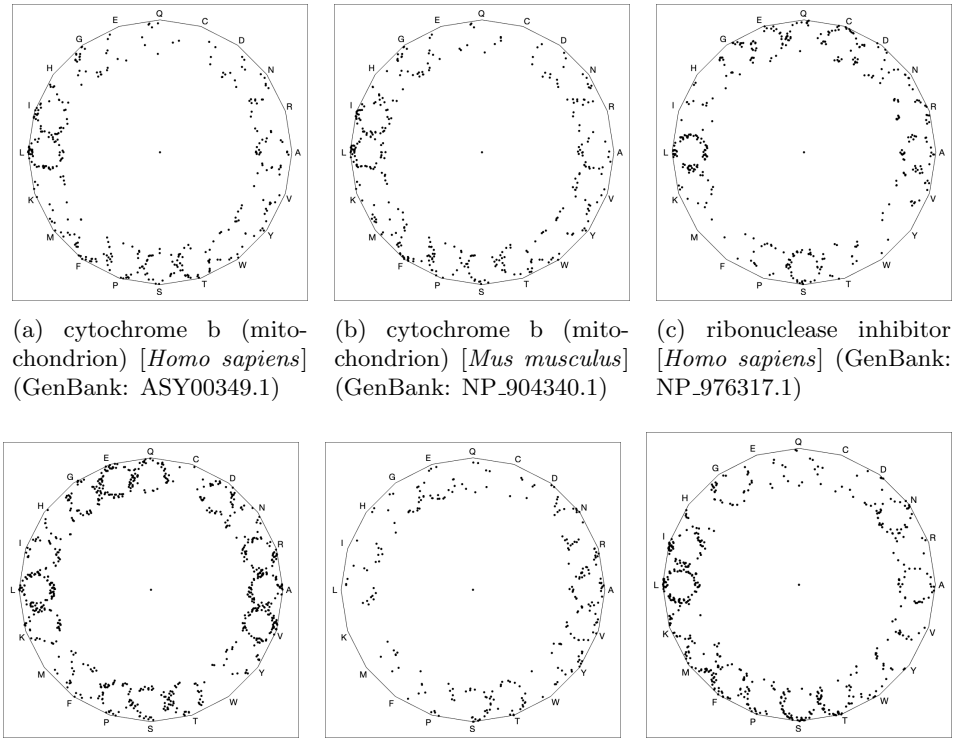
4.3. Protein Visualization. CGR has also been applied to visualizing and analyzing both the primary and secondary structures of proteins. The primary structure of a protein is simply an amino acid sequence. To analyze the primary structure of proteins, the authors of [11] used a 20-sided regular polygon, each side 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 prevent the attractor from overlapping itself, we will use the dividing rate for the 20-gon shown from [2] in Figure 8, $r = 0.863271$. Some CGRs for protein visualization can be found in Figure 15.

According to Basu et al. [5], there are two serious limitations in using a 20-vertex CGR to identify patterns. The first limitation in using the 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 [11]. The second limitation is that the amino acid residues in different positions are often replaced by similar amino acids [5]. 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 [5], Fiser, Tusnady, and Simon [11] introduced the use of CGR to study three-dimensional structures of proteins. A chain of amino acids is what is called a polypeptide. Due to each amino acid having a specific structure that contributes to its 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 [31]. This is known as the secondary structure. The two most common structures that appear in this stage are

Downloaded 11/06/25 to 103.178.194.193 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.



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

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

(c) ribonuclease inhibitor [*Homo sapiens*] (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 CGR of proteins (specifically using their amino acids).*

the α-helices and the β-sheets. Using the chaos game on the secondary structure can reveal any nonrandomness 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 [5, 11].

5. CGR of Mathematical Sequences. In the first-year course we gave in 2015 [7], we applied the CGR to mathematical sequences found in the On-line Encyclopedia of Integer Sequences (OEIS) [30]. In what follows, we show some CGR experiments from that first-year course. Some sequences used from the 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 class, but 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.).

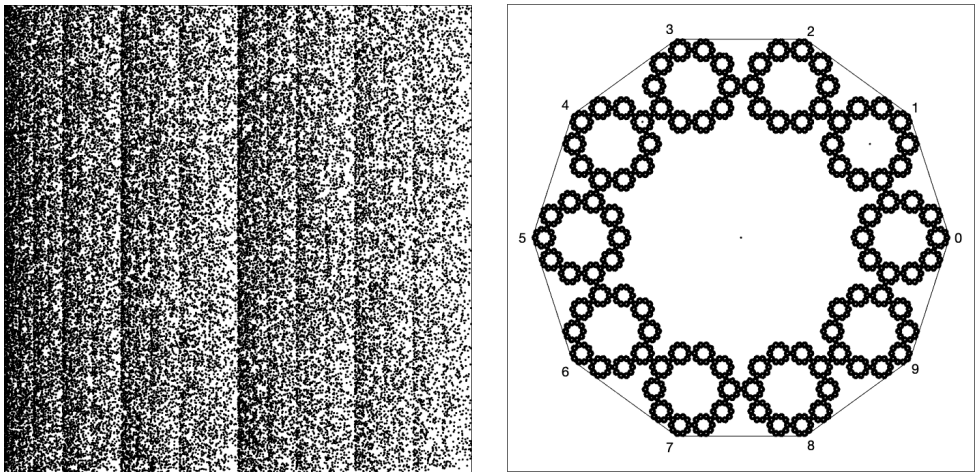
Downloaded 11/06/25 to 103.178.194.193 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 11/06/25 to 103.178.194.193 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

5.1. Digits of Pi. We first applied CGR to the digits of π (A000796 from the OEIS[7]), since we believe that the sequence of the digits is 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 16(a). 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 6(a)). However, we can see from Figure 16(a) 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 16(a) and 6(a) 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 this is 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. For this reason, Figure 16(a) is not a good visual representation of the digits of π.

Another CGR to visualize the digits of π is shown in Figure 16(b). 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.



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

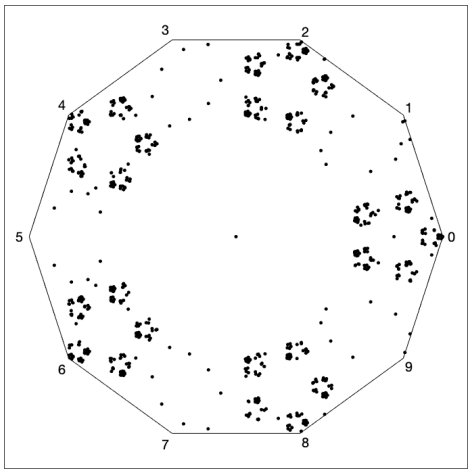
Fig. 16 *CGR of the digits of* π.

5.2. Fibonacci Sequence. We also created a visualization of the Fibonacci sequence (A000045 from the OEIS), one example being the visualization shown in Figure 17. This is a 10-vertex CGR of the Fibonacci sequence modulo 10; that is, a sequence of the final digits of F_n. It appears from the density of the even blocks in Figure 17(a) that most (by far and away the most) of the numbers of the Fibonacci sequence are even; this conclusion seems very strange, and deserves a closer look.

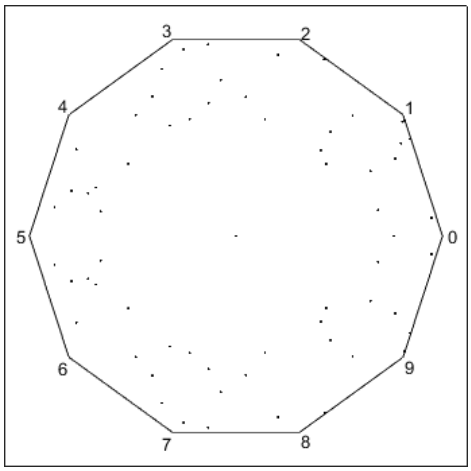
In fact, the sequence is ultimately periodic (as of course it must be, because the set of decimal digits is finite), and indeed it has period 60: The first 65 entries are $1, 1, 2, 3, 5, 8, 3, 1, \ldots, 9, 1, 0, 1, 1, 2, 3, 5$ and it repeats from there. Just by counting, we

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

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.



(a) Calculate the Fibonacci sequence in floating-point arithmetic before taking the remainder mod 10



(b) Take the remainder mod 10 before calculating the next term in the Fibonacci sequence

Fig. 17 10-*vertex CGR of the first* 3000 *Fibonacci sequence modulo* 10. *Here we see that the order in which the remainder is computed makes a difference, if one uses floating-point arithmetic. In Figure* 17(a)*, we first computed the first* 3000 *numbers in the Fibonacci sequence and then computed the remainders. On the other hand, in Figure* 17(b)*, we computed the remainders before calculating the next number in the Fibonacci sequence. Between the two figures, we can clearly see that there is a difference. This difference is because our* `fibonacci` *routine uses floating point, and after the Fibonacci numbers become large enough (starting from the* 79*th term, about* 1.4×10^{16}*), the unit's placement is not even represented and cannot be recovered. Indeed, in a base-*2 *floating-point system, all large enough numbers are even; but before that happens here the* round-to-even *rule is invoked at the* 79*th Fibonacci number.*

can see that 40 out of these 60 numbers are *odd* and only 20 are even, so the conclusion that "almost all Fibonacci numbers are even" that we drew from Figure 17(a) is completely wrong!

The correct behavior is shown in Figure 17(b), which reveals the periodicity by its sparseness. It is important to do the modular reduction as you go, if you are working in floating-point arithmetic;[8] once F_n becomes large enough it won't be possible to compute the trailing digit correctly and all you will see is rounding error, which in Figure 17(a) looks pretty uniform on each *even* block. In contrast, small enough Fibonacci numbers can be represented *exactly* as floating-point integers (also called "flints"), and arithmetic with flints is exact, so long as the result is small enough to also be represented as a flint. What we are seeing in the figure is an effect of rounding errors in *large* floating-point integers; further, a closer look at f_{79} shows the *round-to-even* rule for IEEE floating-point arithmetic.[9] Since f_{78} really is even, and f_{79} is rounded to an even number by the round-to-even rule, all computed floating-point

[8]MATLAB, before version 2021a, did not have the built-in `fibonacci` function, so we created our own implementation of `fibonacci` in MATLAB that uses floats.

[9]The round-to-even rule is a topic that many students stumble over. As is pointed out in [15, section 1.17], rounding errors are not random, but the round-to-even rule was invented to reduce the *bias* of rounding error in long sequences of computation. This sounds paradoxical, but although rounding errors are not random, or independent, there are situations where it is useful to assume so; see [15, section 2.8] for further discussion of this point.

Downloaded 11/06/25 to 103.178.194.193 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 11/06/25 to 103.178.194.193 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

Fibonacci numbers thereafter are even. Of course, once they become large enough that the unit's placement falls completely off the significand, they will all be reported as "even" anyway.

Other visualizations for the Fibonacci sequence can be made, such as taking the remainder using a different divisor and using a CGR with a different number of vertices. However, since the Fibonacci sequence grows very quickly, like ϕ^n where $\phi = (1+\sqrt{5})/2 \doteq 1.618$ is the golden ratio, students in the class also learned about another limitation of floating point since they were unable to take more than the first 3000 Fibonacci numbers before the computation overflowed, registering the number as infinity. One possible remedy for this is to use the arbitrary precision integer datatype of Python. This prevents overflow, but the computing cost is significantly larger and the program will run more slowly.

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[10]). We considered four different ways of creating the CGR for prime numbers.

We first visualized the prime numbers in a similar way as in Figures 16(b) and 17 by taking the sequence of prime numbers modulo 10. This is shown in Figure 18(a). We notice that almost all of the prime numbers end with a 1, 3, 7, or 9. Because of this observation, we decided to create a square CGR labeling the vertices with these values. The result is shown in Figure 18(b).

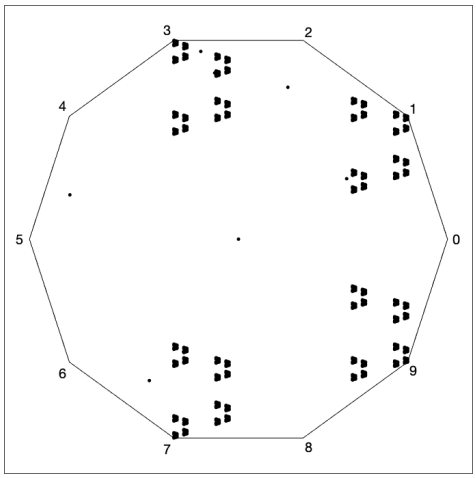
Figure 18(c) shows the third visualization of prime numbers: a CGR of the $103+k$th prime numbers $\bmod 4$, a diagonal line which spans from the "1" corner to the "3" corner. As we all know, prime numbers larger than 2 will never be even, so the corner points "0" and "2" which are representative "even"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

Last, we can visualize the prime numbers in a different way, as seen in Figure 18(d). Here, we took the sequence of prime numbers less than one million and took $\bmod 8$ of the numbers. We thought that this would be interesting because taking any prime number larger than 2 modulo 8 will only result in the following numbers: 1, 3, 5, and 7. Because of this, we relabeled 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 labeled "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 6(a), 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.

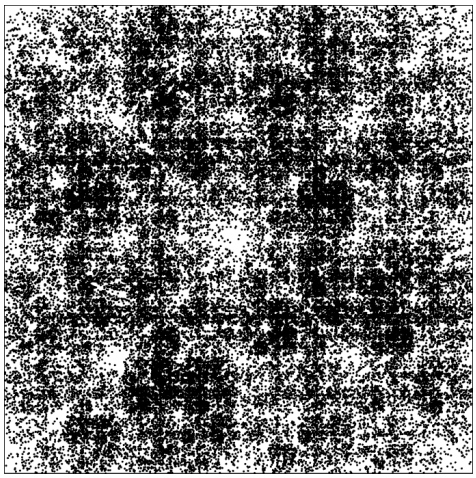
5.4. Partial Quotients of Continued Fractions. We then explored visualizing the sequences taken from the partial quotients of continued fractions, which were taught at the beginning of the course. Continued fractions are fractions written in

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

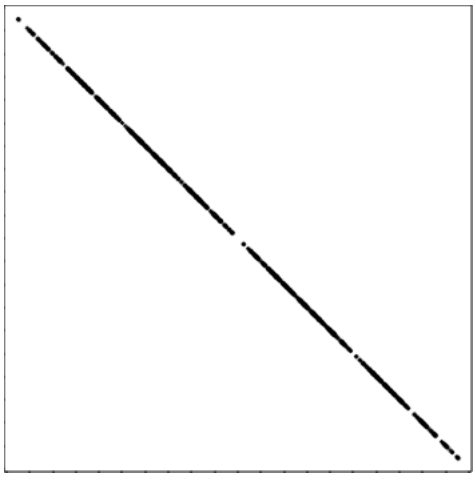
Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.



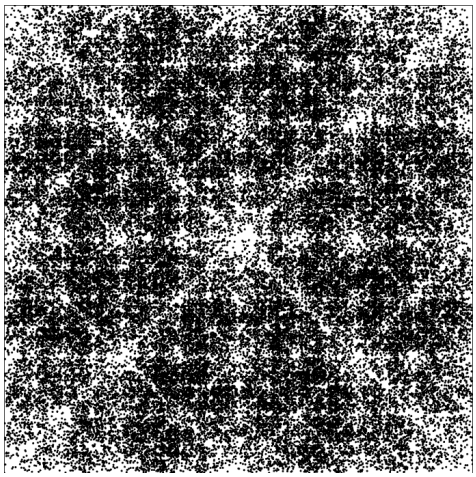
(a) All prime numbers less than 1000000 taken mod 10



(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.*

the following general form [28, 8]:

$$a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{a_3 + \cfrac{1}{a_4 + \cfrac{1}{\ddots}}}}}, \tag{5.1}$$

Downloaded 11/06/25 to 103.178.194.193 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 11/06/25 to 103.178.194.193 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

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

$$\frac{9}{7} = 1 + \frac{2}{7} = 1 + \frac{1}{7/2} = 1 + \frac{1}{3 + 1/2} = 1 + \cfrac{1}{3 + \cfrac{1}{1 + 1/1}} . \tag{5.2}$$

Here, the partial quotients of $9/7$ are the elements of $1 + [3, 1, 1]$. The students in the class 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 CGRs.

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 CGR of the quotients of the continued fraction of $\sqrt{2}$. As we had seen earlier in the course, the sequence goes like

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

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 19(a). The students also tried the partial quotients of e, which is equal to

$$2 + [1, 2, 1, 1, 4, 1, 1, 6, \dots] \tag{5.3}$$

[29], shown in Figure 19(b). What is seen here makes sense as the sequence mostly contains 1's and these alternate with even values—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, they are 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 19(c), 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.

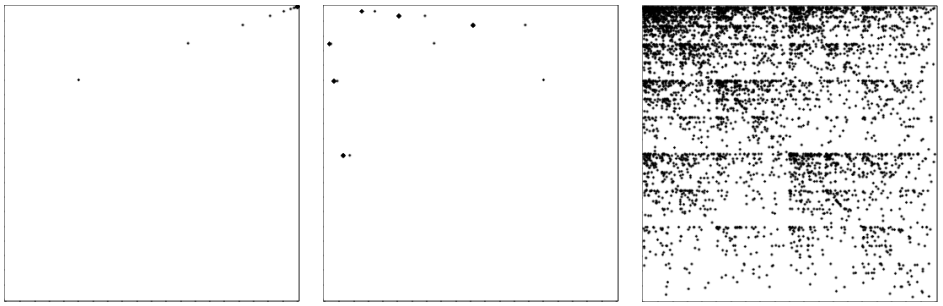
In fact, the distribution of partial quotients is very well known, owing to the results of Khinchin [21]. As a very startling aside, Bill Gosper recently remarked on an amazing identity

$$\prod k!^{\Delta_3 \ln k} = K^{\ln 2} , \tag{5.4}$$

where K is Khinchin's constant. We did not inform our students of Khinchin's remarkable results because the 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 exact trajectories, and, therefore, it may be more difficult to grasp. There are many different works on the fundamentals of probability which discuss this difficulty; see, for instance, [14], but also [18]. We believe that combining deterministic dynamics with randomness

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.



(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* (a) $\sqrt{2}$, (b) e, *and* (c) π.

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, and whether entries in a sequence are "really" independent. 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 a computer? (The answer is yes.) And what is the difference, anyway? There are knots here that the students (and professors) can tie themselves into: 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 by very nonrandom events (selection). We have used some "clearly" nonrandom (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 include the *multiple-recursive method* and families formed by *shift-register methods* [9]. The multiple-recursive method is an extension of the linear congruential method that replaces the first-order linear recursion by one of

Downloaded 11/06/25 to 103.178.194.193 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 11/06/25 to 103.178.194.193 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

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 [26], which is the default for MATLAB's pseudorandom number generating functions, which include `rand` (uniformly distributed random number), `randn` (normally distributed random numbers), and `randi` (uniformly distributed pseudorandom integers) [27]. 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 bit accuracy [9]. 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 the pseudorandom number algorithms typically used in computer programs. The randomness in their program comes from atmospheric noise.

Using a pseudorandom number generator differs from using dice or the atmospheric-sourced generators at random.org. One obtains 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 [32] 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 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 instructor could emphasize 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. MATLAB code to reproduce the figures in this paper can be found in our GitHub repository: github.com/echan295/Chaos-Game-Representation.

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 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. Modeling viral evolution using randomness [33], 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

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

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]

Indeed, much the same comments can be made about our use of the words "fractal" and "attractor" here. All of our computations are finite, so *nothing* that we see is actually a "true fractal," and similarly our computations show only parts of the transients, not the attracting set (much less the attractor). This doesn't matter; again, what matters is that fractals and attractors, in the infinite limit, provide good models for the pictures that our finite computations produce. For more details about both the rigor and the utility of fractals, consult [25]. As a final remark, what convinced us of the value of CGR for DNA sequences was nothing mathematical, but rather its wonderfully apropos placement of curious creatures on the "Map of Life": for instance, picking out an organism mapped to a point near the border between the group of "fish" and the group of "amphibians," we find an air-breathing fish!

This raises a final point. This technique really works for DNA sequences, and it can be genuinely informative for other kinds of sequences, as we have seen. For instance, we were able to detect the influence of rounding errors (for the Fibonacci sequence computation). One also needs an explanation of *why* this works; this can lead to good discussion points with students. Consider the impact of short stretches of so-called "junk" or noncoding DNA, distributed throughout a long coding sequence. Will the "junk" make a difference to the CGR representation? In some cases not, but if there is enough junk, then maybe it will. But in many cases we could claim that one reason this works is that CGR really gives a "whole picture" of the sequence and is relatively robust under errors such as shifts and interruptions with short insertions. After discussions one should investigate such claims, to see if they are true.

Acknowledgments. We would like to thank Lila Kari for introducing us to the chaos game representation. We would 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 (a.k.a. Western University), NSERC, and Ontario Graduate Scholarship (OGS).

REFERENCES

[1] A. D. Aleksandrov, A. N. Kolmogorov, and M. A. Lavrent'ev, *Mathematics: Its Content, Methods and Meaning*, Courier Corporation, 1999. (Cited on p. 288)

[2] J. S. Almeida and S. Vinga, *Biological sequences as pictures—a generic two dimensional solution for iterated maps*, BMC Bioinform., 10 (2009), art. 100. (Cited on pp. 271, 272, 273, 279)

Downloaded 11/06/25 to 103.178.194.193 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 11/06/25 to 103.178.194.193 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

[3] M. F. Barnsley, *Fractals Everywhere*, Academic Press, 2014. (Cited on pp. 263, 264, 265, 266)
[4] M. F. Barnsley and A. Vince, *The chaos game on a general iterated function system*, Ergodic Theory Dynam. Systems, 31 (2011), pp. 1073–1079. (Cited on p. 265)
[5] S. Basu, A. Pan, C. Dutta, and J. Das, *Chaos game representation of proteins*, J. Molecular Graph. Model., 15 (1997), pp. 279–289. (Cited on pp. 262, 279, 280)
[6] I. Borg and P. J. F. Groenen, *Modern Multidimensional Scaling: Theory and Applications*, Springer Science & Business Media, 2005. (Cited on p. 277)
[7] E. Y. S. Chan and R. M. Corless, *A random walk through experimental mathematics*, in Springer Proceedings in Mathematics & Statistics, Springer, 2020, pp. 203–226, https://doi.org/10.1007/978-3-030-36568-4_14. (Cited on p. 280)
[8] R. M. Corless, *Continued fractions and chaos*, Amer. Math. Monthly, 99 (1992), pp. 203–215. (Cited on p. 284)
[9] M. R. Dennis, P. Glendinning, P. A. Martin, F. Santosa, and J. Tanner, *The Princeton Companion to Applied Mathematics*, Princeton University Press, 2015. (Cited on pp. 286, 287)
[10] D. P. Feldman, *Chaos and Fractals: An Elementary Introduction*, Oxford University Press, 2012. (Cited on p. 265)
[11] A. Fiser, G. E. Tusnady, and I. Simon, *Chaos game representation of protein structures*, J. Molecular Graph., 12 (1994), pp. 302–304. (Cited on pp. 262, 271, 272, 273, 279, 280)
[12] 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. (Cited on p. 268)
[13] N. Goldman, *Nucleotide, dinucleotide and trinucleotide frequencies explain patterns observed in chaos game representations of DNA sequences*, Nucleic Acids Res., 21 (1993), pp. 2487–2491. (Cited on pp. 262, 273)
[14] R. W. Hamming, *The Art of Probability for Scientists and Engineers*, Addison Wesley Publishing Company, 1991. (Cited on pp. 263, 285)
[15] N. J. Higham, *Accuracy and Stability of Numerical Algorithms*, SIAM, 2002, https://doi.org/10.1137/1.9780898718027. (Cited on p. 282)
[16] H. J. Jeffrey, *Chaos game representation of gene structure*, Nucleic Acids Res., 18 (1990), pp. 2163–2170. (Cited on pp. 262, 266, 267, 272)
[17] H. J. Jeffrey, *Chaos game visualization of sequences*, Comput. Graph., 16 (1992), pp. 25–33. (Cited on pp. 262, 272)
[18] M. Kac, *Probability and Related Topics in Physical Sciences*, Vol. 1, AMS, 1959. (Cited on p. 285)
[19] R. Karamichalis, L. Kari, S. Konstantinidis, and S. Kopecki, *An investigation into inter- and intragenomic variations of graphic genomic signatures*, BMC Bioinform., 16 (2015), p. 246. (Cited on p. 262)
[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), art. e0119815. (Cited on pp. 275, 276, 277)
[21] A. Ya. Khinchin, *Continued Fractions*, University of Chicago Press, Chicago, 1964. (Cited on p. 285)
[22] D. E. Knuth, *The Art of Computer Programming, Vol.* 2: *Seminumerical Algorithms*, 3rd ed., Addison Wesley, Reading, MA, 1997. (Cited on p. 268)
[23] A. N. Kolmogorov, *On logical foundations of probability theory*, in Probability Theory and Mathematical Statistics, Springer, 1983, pp. 1–5. (Cited on p. 286)
[24] J. E. Littlewood, *Littlewood's Miscellany*, Cambridge University Press, 1986. (Cited on p. 262)
[25] B. B. Mandelbrot, *Fractals and Chaos*, Springer, New York, 2004, https://doi.org/10.1007/978-1-4757-4017-2. (Cited on p. 288)
[26] M. Matsumoto and T. Nishimura, *Mersenne twister: A* 623*-dimensionally equidistributed uniform pseudo-random number generator*, ACM Trans. Model. Comput. Simul., 8 (1998), pp. 3–30. (Cited on p. 287)
[27] C. B. Moler, *Random number generators, Mersenne Twister*, in Cleve's Corner: Cleve Moler on Mathematics and Computing, 2015; available at https://blogs.mathworks.com/cleve/2015/04/17/random-number-generator-mersenne-twister/. (Cited on p. 287)
[28] C. D. Olds, *Continued Fractions*, Random House, New York, 1963. (Cited on p. 284)
[29] C. D. Olds, *The simple continued fraction expansion of e*, Amer. Math. Monthly, 77 (1970), pp. 968–974. (Cited on p. 285)
[30] *The On-line Encyclopedia of Integer Sequences*, 2019, https://oeis.org/. (Cited on p. 280)

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

[31] 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*, Proc. Natl. Acad. Sci. USA, 88 (1991), pp. 3661–3665. (Cited on p. 279)

[32] 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). (Cited on p. 287)

[33] L. M. Wahl and T. Pattenden, *Prophage provide a safe haven for adaptive exploration in temperate viruses*, Genetics, 206 (2017), pp. 407–416. (Cited on p. 287)

[34] 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. (Cited on p. 275)

[35] F. Wickelmaier, *An Introduction to MDS*, Sound Quality Research Unit, Aalborg University, Denmark, 2003. (Cited on p. 277)

Downloaded 11/06/25 to 103.178.194.193 . Redistribution subject to SIAM license or copyright; see https://epubs.siam.org/terms-privacy

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.