Irish Math. Soc. Bulletin
Number 95, Summer 2025, 23– 34
ISSN 0791-5578

Computing $\sqrt{2}$ with FRACTRAN

KHUSHI KAUSHIK, TOMMY MURPHY AND DAVID WEED

ABSTRACT. The FRACTRAN programs $\sqrt{2}$GAME and NR$\sqrt{2}$GAME are presented, both of which compute the decimal expansion of $\sqrt{2}$. Our $\sqrt{2}$GAME is analogous to Conway's PIGAME program. In fact, our proof carries over to PIGAME to produce a simpler proof of Conway's theorem as well as highlight how the efficiency of the program can be improved. NR$\sqrt{2}$GAME encodes the canonical example of the Newton–Raphson method in FRACTRAN.

1. INTRODUCTION

FRACTRAN is a Turing complete esoteric programming language with several notable features (c.f. [4], [5]). It is simple to understand how the language works. One can code any standard mathematical algorithm in FRACTRAN, and moreover the Gödel number of any program is straightforward to explicitly compute. Conway developed this language in [2], and used it to formalize examples proving that natural generalizations of the famed Collatz conjecture are undecidable [3]. He produced several explicit examples of algorithms in FRACTRAN in [2]. Two examples are PRIMEGAME, which computes, in order, every prime number, and PIGAME, which generates, in order, the digits of the decimal expansion of π. In fact PIGAME ties in with a classical and fundamental question at the intersection of real analysis and theoretical computer science; namely how to compute the decimal expansion of a computable irrational number. Turing defined the computable numbers as the real numbers whose decimal expansions can be computed algorithmically (i.e. with a Turing machine), and they play a central role in the work of both Turing and Gödel. For a fascinating and readable account of this theory, the interested reader is referred to the first two chapters of [6]. Although this book was written in 1989, Penrose expressed prescient thoughts on the role computers and A.I. will play in mathematical research which are extremely relevant today.

As Conway himself states, the proof that PIGAME actually works is nontrivial. It involves using some heavy machinery (e.g. Mähler's famed irrationality measure for π) together with Wallis' infinite product formula for $\frac{\pi}{2}$ to ensure that truncating this infinite product after a certain even number $E \geq 4\times 2^{10^n}$ terms is sufficiently accurate to compute the n-th digit of π. One initial motivation for this work was to actually explain what Conway does, as many details are omitted. The first main theorem of this paper ($\sqrt{2}$GAME) then computes, in order, the decimal expansion of $\sqrt{2}$ via Catalan's [1] infinite product expansion of $\sqrt{2}$. The mechanics of proof are largely analogous to Conway's, however we find a simpler proof that our truncated approximation is sufficiently accurate to compute. This simpler proof also carries over to PIGAME: one then sees *a posteriori* that a simpler and faster program could be written to compute the decimal expansion of π.

2010 *Mathematics Subject Classification.* 68Q04.
Key words and phrases. FRACTRAN, decimal expansions.
Received on 02-04-2025; revised 10-07-2025.
DOI: 10.33232/BIMS.0095.23.34.
K.K acknowledges support from the LSAMP grant at CSUF. K.K. and D.W were supported with a summer undergraduate research grant in 2023 by the Department of Mathematics at CSU Fullerton. K.K. also thanks the Undergraduate Research Opportunity Center, CSU Fullerton, for travel support. The authors thank Aaron Cottle for helpful comments. We also thank the referee for helping us to improve the paper with several insightful comments.

There are of course several extremely efficient ways to compute the decimal expansion of $\sqrt{2}$, or for that matter π. The most standard, familiar to generations of calculus students, is the Newton–Raphson method. Our second theorem presents NR$\sqrt{2}$GAME, which computes the n-th digit of $\sqrt{2}$ via this standard algorithm.

We wish to emphasize that all our programs are useless in practice: there are much quicker programs to compute the decimal expansion of a given computable number and this is a whole field of research. Even to compute the number E is extremely time consuming as it is double exponential as a function of n. Why then should you care about FRACTRAN? To our knowledge, it is the simplest way to convert an explicit algorithm (i.e. the rules of a Turing machine) into a genuine program in a language. Additionally the language makes the interplay between the state the machine is in (the register) and the commands of the program very clear and easy to follow. As such FRACTRAN is an excellent educational aid which highlights how simple mathematical computation (the code only involves multiplying rational numbers together) is powerful enough to describe every Turing machine. Another very good reason to study FRACTRAN is that Conway *explicitly* produces in [2] a universal Turing machine in this language. In other words, he produces a short list of fractions, and every single computer program can be described by choosing one natural number (the Gödel number of this program, or what Conway calls the catalog number) and running this fixed FRACTRAN program with that input. Finally, Penrose comments ([6], Chapter 2) that computing the decimal expansion of a computable irrational number is precisely the sort of model algorithm one should be able to generate to truly understand a programming language. He lists π and $\sqrt{2}$ as the two canonical examples of such numbers, which motivates generalizing PIGAME to $\sqrt{2}$GAME as a natural problem.

2. RULES OF THE GAME

2.1. **Initial Comments.** Turing Machines model the action of a computer. The main parts of a Turing machine are a way to store data, originally (abstractly) thought of as an infinitely long tape, and a set of rules that allow the conditional change of that data. The starting state of the tape is thought of as the input of the code, and the resulting state is the output. Different programs are then made by changing the rules. In FRACTRAN, the entirety of the data is stored in a finite set of fractions (the program) and a single integer (the register). Via the Fundamental Theorem of Arithmetic, each integer $n \in \mathbb{N}$ admits a unique prime decomposition $n = \prod_{i=1}^{k} p_i^{\alpha_i}$. Conway's simple idea is to encode a Turing machine starting with a number N so that each power in its prime decomposition gives the state the Turing machine is in. Each power of the prime appearing in N tells us the initial state of our system: it tells us what is in each register. Now multiply N by a fraction f_i so that $f_i N$ is also a whole number: if we take the prime factor decomposition of the numerator and denominator of f_i, we have that $f_i N \in \mathbb{N}$ if, and only if, the powers appearing in the prime decomposition of N have been redistributed. Included in this statement is the possibility that the primes with 0 in their register will initially become non-empty. This exactly models the mechanisms of a Turing machine and is the basic idea underlying FRACTRAN. More formally, we need an initial state (a stored number) $N \in \mathbb{N}$ which is in our *register* and a fixed list of fractions $\{f_1, f_2, \ldots, f_n\}$. Compute $f_i N$, with $i = 1, 2, \ldots, n$, until we reach the first instance where $f_j N \in \mathbb{N}$, $j \in \{1, 2, \ldots, n\}$. Then change the register to $f_j N$ and iterate. In practice, Conway thinks of FRACTRAN as a flow chart that proceeds from one node to another. To indicate where to go, the nodes are connected by arrows with a well-defined notational hierarchy as follows:

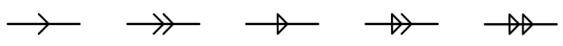

FIGURE 1. Hierarchy of arrows

These arrows are listed in hierarchical order of precedence, read from left to right. These arrows are then labeled with fractions, which tell us how to update the register. There is a well-understood algorithm to convert this flow chart into a list of fractions [2]. We will explain this in due course, and in the appendix a Python code for this algorithm is presented. We will also dwell a little on how to understand the hierarchy of arrows as we explain PIGAME. We present the simplest example to begin.

Example 2.1. Let us write a program to add two given whole numbers, say a and b. Store these numbers in our initial register as $N = 2^a 3^b$. Then build a single loop labeled with $2/3$.

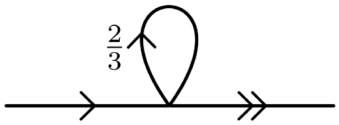

FIGURE 2. A flow chart for addition

Every time we go around the loop, we multiply N by $\frac{2}{3}$. It is easy to see this game ends with output 2^{a+b}. The FRACTRAN code is easy to derive in this example: $N = 2^a 3^b$ is our initial state, and $\{\frac{2}{3}\}$ is our list of fractions.

3. CONWAY'S PIGAME

Theorem 3.1. *(PIGAME* [2]*) When started at $2^n \cdot 89$, the FRACTRAN code*

$$\frac{365}{46}\ \frac{29}{161}\ \frac{79}{575}\ \frac{679}{451}\ \frac{3159}{413}\ \frac{83}{407}\ \frac{473}{371}\ \frac{638}{355}\ \frac{434}{335}\ \frac{89}{235}\ \frac{17}{209}\ \frac{79}{122}\ \frac{31}{183}\ \frac{41}{115}\ \frac{517}{89}\ \frac{111}{83}\ \frac{305}{79}\ \frac{23}{73}\ \frac{73}{71}$$

$$\frac{61}{67}\ \frac{37}{61}\ \frac{19}{59}\ \frac{89}{57}\ \frac{41}{53}\ \frac{833}{47}\ \frac{53}{43}\ \frac{86}{41}\ \frac{13}{38}\ \frac{23}{37}\ \frac{67}{31}\ \frac{71}{29}\ \frac{83}{19}\ \frac{475}{17}\ \frac{59}{13}\ \frac{41}{291}\ \frac{1}{7}\ \frac{1}{11}\ \frac{1}{1024}$$

will terminate at $2^{\pi(n)}$, where $\pi(n)$ is the n-th digit in the decimal expansion of π.

The first order of business is to show how this list of fractions is generated via Conway's algorithm from the following flow chart:

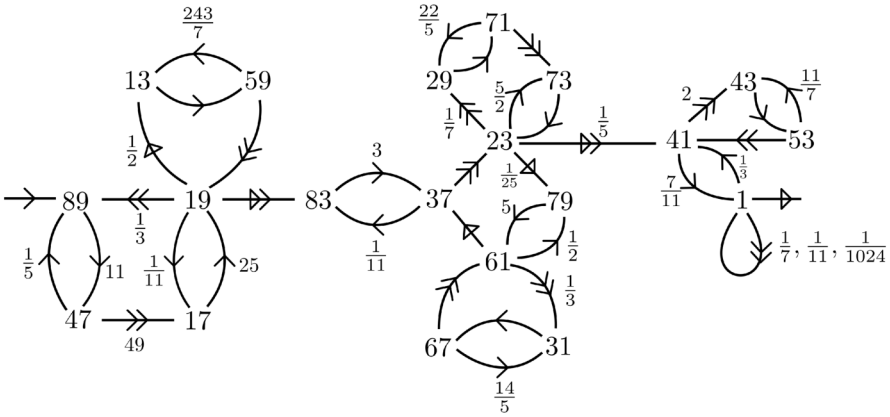

FIGURE 3. The flow chart for PIGAME

Notice that you can identify which part of the flow chart a given fraction corresponds to by looking at the prime decomposition of the numerator and denominator. For instance, the fraction $41/115$ corresponds to moving from node 23 to node 41. We start at node 23, meaning the register N has exactly one copy of 23 in its prime decomposition. For a given prime p, Conway defines r_p as the power of p in the prime decomposition of the register N, so here $r_{23} = 1$. Since $115 = 23 \times 5$, multiplying $(41N)/115$ clears 23 from the register (so $r_{23} = 0$) and puts 41 in its place, since after the multiplication N has updated to have $r_{41} = 1$. As we move from the 23 cell/node to the 41 one, the program tells us to adjust N by reducing r_5 by 1. In full generality, if one goes from a node labelled p to a node labelled q and the program requires us to update N by multiplying by the fraction M_1/M_2, the resulting fraction in our code is $\frac{M_1 q}{M_2 p}$. To make this all work smoothly, it is important to have different primes corresponding to the nodes and the actual program.

The arrow which we are discussing in the flow chart is third in the hierarchy of Figure 1. The arrow going to node 73 ranks first, and the arrow going to node 79 is second. Hence, the corresponding fractions (namely $365/46$ and $79/575$) must come before $41/115$ in the code. Note that when the machine is at the 23 node, all other fractions in the code will not adjust the register as $f_i N \in \mathbb{N}$ if, and only if, 23 is in the denominator of the fraction f_i. So they do not play a role at all when we are at this stage. However, we have to list these three fractions corresponding to the 23 node in the order mandated by the arrows. This means we want to always go to 73 node first until that will violate the rules of FRACTRAN. In particular, every time we go to the 73 node, r_2 will decrease by 1 and r_5 will increase by 1. This stops when $r_2 = 0$, because if $r_2 = 0$, multiplying N by $5/2$ will not be a whole number, and analogously for all later arrows.

Remark 3.2. Conway has an unwritten convention of arranging the fractions in his code in order of decreasing denominators. There have to be some exceptions to this depending on the code. We have just discussed how the fraction $41/115$ corresponds to the arrow moving from node 23 to node 41; but all other arrows emanating from node 23 come before this arrow in the hierarchy and so their corresponding fractions must come before $41/115$ in the code. That is why, for instance, $365/46$ is located as the first fraction despite 46 being a small denominator in the list.

Remark 3.3. There is a small bug in Conway's code, known to experts, where he incorrectly states the code starts at 2^n. A corrected statement is presented here.

4. THE PROOF OF PIGAME: SETUP

Since the proof of Theorem A is based on PIGAME, and his proof that the algorithm actually works is short on details, it is natural first to discuss the proof and fill in some of the steps. For $n \in \mathbb{N}$, the claim is that running PIGAME will compute the n-th decimal digit of π. The flow chart breaks into three phases.

Phase 1 From node 89 to node 83, the program computes E, an even number $\geq 4 \times 2^{10^n}$.

Phase 2 From node 83 to node 41, the program computes

$$10^n N_E = 10^n 2E(E-2)^2 \dots 4^2 2^2, \text{ and}$$
$$D_E = (E-1)^2 (E-3)^2 \dots 3^2 1^2.$$

Phase 3 The program computes the integer part of $\frac{10^n N_E}{D_E}$ and reduces it modulo 10.

The mechanism of these phases are all fully explained in [2]. The number computed in Phase 3 is the n-th term in the decimal expansion of

$$\pi_E = \frac{N_E}{D_E} = \frac{2E(E-2)^2 \dots 4^2 2^2}{(E-1)^2(E-3)^2 \dots 3^2 1^2}.$$

Multiplying the numerator of π_E by 10^n shifts the decimal unit of π_E exactly n places to the right. Taking the floor function turns this into an integer, and reducing mod 10 allows us to find the n-th term in the decimal expansion of π_E. To complete the proof, one has to compute explicitly how close π_E is to π. Another issue to bear in mind comes from the well-known fact that two numbers can be very close together but have differing decimal expansions due to the identification $1 = 0.99\dot{9}$.

So, to show the program actually works, it remains to prove that the n-th decimal digit of π and π_E agree. To this end, Conway states without proof that

$$|\pi - \pi_E| < \frac{\pi}{E} \tag{1}$$

Then $|\pi - \pi_E| < \frac{\pi}{E} < 10^{-n}$, meaning π and π_E agree to n decimal places unless one of them has a decimal expansion containing only zeros from the n-th decimal place onwards (where we make the usual identification $1 = 0.99\dot{9}$). The proof thus reduces to two key steps; (i) establish Equation (1), and (ii) show that $10^n \pi_E$ cannot be an integer.

5. Establishing Equation (1)

The first step is to show that $\pi < \pi_E$ holds for all E even. By way of contradiction, if $\pi_{E_0} < \pi$ then $\pi_{E_0+2} < \pi_{E_0}$, since cancelling common terms we have

$$\pi_{E_0+2} < \pi_{E_0} \iff \frac{E_0(E_0+2)}{(E_0+1)^2} < 1$$

which is true for all E_0. Iterating this argument we obtain (with $E = 2j$ denoting the subsequence of even integers)

$$\pi = \lim_{E \to \infty} \pi_E < \pi_{E_0} < \pi$$

a contradiction. Now for E even, we define

$$\pi_{\tilde{E}} = \pi_E \left(\frac{E}{E+1} \right)$$

A directly analogous argument left to the reader shows that $\pi_{\tilde{E}} < \pi$. Putting these two facts together we obtain

$$\pi_{\tilde{E}} < \pi < \pi_E \tag{2}$$

Equation (2) implies the desired Equation (1). This is a simple computation:

$$|\pi - \pi_E| \stackrel{\Delta}{=} \pi_E - \pi < \frac{\pi}{E} \iff \pi_{\tilde{E}} < \pi.$$

Note both inequalities in Equation (2) are used. The fact $\pi < \pi_E$ is used for $\stackrel{\Delta}{=}$. Then

$$\pi_E - \pi < \frac{\pi}{E} \iff \pi_E < \pi \left(\frac{E+1}{E} \right)$$

which rearranges to the statement that $\pi_{\tilde{E}} < \pi$, i.e. the other inequality in Equation (2).

Now we know π and π_E are within 10^{-n} of each other, it remains to show their decimal expansion agrees in the n-th decimal place. To this end, Conway utilizes the following result.

Lemma 5.1. *(Mähler's irrationality measure) If p/q is any rational number with $gcd(p, q) = 1$,*

$$\left| \pi - \frac{p}{q} \right| > \frac{1}{q^{42}}.$$

Write $\pi_E = p/q$, with $\gcd(p,q) = 1$. Applying Mähler's Lemma

$$\frac{1}{q^{42}} < \left|\pi - \frac{p}{q}\right| < \frac{\pi}{E} < \frac{1}{10^{42n}},$$

whence (since $x \to x^{42}$ is an increasing function) $q > 10^n$. Assume that

$$10^n \pi_E = \frac{10^n N_E}{D_E} = \frac{10^n p}{q}$$

is an integer. Since $q > 10^n$, there is a prime number r whose multiplicity in the prime decomposition of q is greater than the multiplicity of r in the prime decomposition of 10^n (the power of r could be zero in the prime decomposition of 10^n). Hence r divides p, which is a contradiction as p and q are coprime. This proves fully that Conway's algorithm works.

There is actually an elementary proof that $\frac{10^n p}{q}$ is not an integer. This will used in the proof of our first main theorem since there is no irrationality measure for $\sqrt{2}$ (it is algebraic). Supposing $\frac{10^n p}{q}$ is an integer means that q divides 10^n. Since q is odd, that implies $q = 5^j$ where $j \leq n$. However, when we cancel all the common factors in N_E and D_E to get p and q, we cannot cancel the largest prime in D_E. This is a consequence of Dirichlet's theorem, which states there must be at least one (odd) prime between E and $E/2$: this number is greater than 5, and no number in N_E divides into it. This is the desired contradiction

Remark 5.2. It is apparent from our discussion that PIGAME can be simplified: there is no need to generate such a large E. The size of E is exploited when using Mähler's irrationality measure, but we have seen this is not needed.

6. $\sqrt{2}$GAME

Our first main result is as follows.

Theorem A. *When started at $2^n \cdot 173$, the Fractran code*

$$\frac{424375}{173}, \frac{101}{1067}, \frac{89}{291}, \frac{13}{194}, \frac{4897}{97}, \frac{2425}{101}, \frac{1243}{89}, \frac{89}{565}, \frac{4949}{113}, \frac{109}{13}, \frac{3159}{763}, \frac{97}{109}, \frac{321}{83}, \frac{83}{1177},$$

$$\frac{103}{107}, \frac{365}{206}, \frac{29}{721}, \frac{79}{64375}, \frac{41}{2575}, \frac{103}{73}, \frac{71}{29}, \frac{638}{355}, \frac{4393}{4189}, \frac{73}{71}, \frac{71}{191}, \frac{1525}{79}, \frac{79}{122}, \frac{12931}{183}, \frac{107}{61},$$

$$\frac{2669}{12931}, \frac{149}{1273}, \frac{18745}{521461}, \frac{31}{67}, \frac{329322079}{18055}, \frac{67}{157}, \frac{157}{131}, \frac{385447}{2533}, \frac{151}{149}, \frac{149}{139}, \frac{2329}{7097}, \frac{67}{151},$$

$$\frac{151}{137}, \frac{67}{163}, \frac{938}{1333}, \frac{61}{31}, \frac{7}{943}, \frac{254}{41}, \frac{41}{11}, \frac{1}{3}, \frac{1}{7}, \frac{1}{13}, \frac{1}{17}, \frac{1}{19}, \frac{1}{23}, \frac{1}{47}, \frac{1}{1024}, \frac{53}{127}, \frac{2921}{371}, \frac{41}{53},$$

will terminate at $2^{\sqrt{2}(n)}$, where $\sqrt{2}(n)$ is the n-th digit in the decimal expansion of $\sqrt{2}$.

This list of fractions is generated from the flow chart in Figure 4, where we label each node with a distinct prime number and break all loops up as per Conway's algorithm. For economy of space, the term $1/C$ in the figure refers to the list of fractions

$$\frac{1}{3}, \frac{1}{7}, \frac{1}{13}, \frac{1}{17}, \frac{1}{19}, \frac{1}{23}, \frac{1}{47}, \frac{1}{1024}.$$

It is obvious from the flow chart that our proof is based on PIGAME. Theorem B, presented in the next section, will show a more standard algorithm for computing $\sqrt{2}$. Nevertheless, $\sqrt{2}$GAME has the merit of fitting into the same framework as PIGAME.

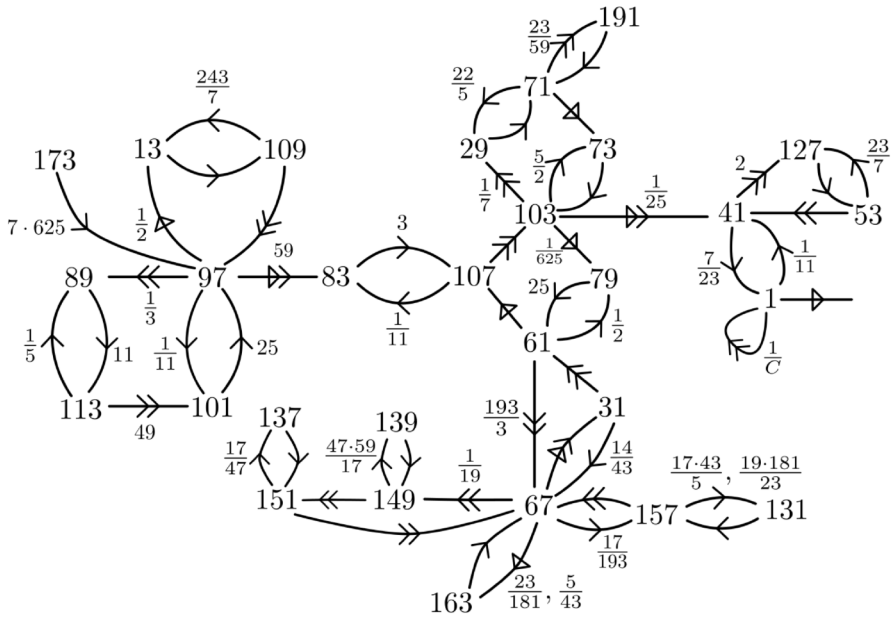

FIGURE 4. The flow chart for $\sqrt{2}$GAME

Proof. The starting point is the infinite product formula for $\sqrt{2}$ due to Catalan in 1874 [1] and which is very similar to Wallis' formula: viz.

$$\sqrt{2} = \left(\frac{2.2}{1.3}\right)\left(\frac{6.6}{5.7}\right)\left(\frac{10.10}{9.11}\right)\cdots$$

This program will truncate Catalan's product formula in an analogous manner to PIGAME. Defining

$$\sqrt{2}_E := \frac{2^2}{1.3}\frac{6^2}{5.7}\cdots\frac{(E-4)^2}{(E-5)(E-3)}\frac{E}{E-1} = \frac{N_E}{D_E},$$

we claim that the approximation $\sqrt{2}_E$ is sufficiently close to $\sqrt{2}$ and show that $10^n\sqrt{2}_E$ is never an integer to prove that the program does indeed compute the n-th digit in the decimal expansion of $\sqrt{2}$.

The first phase, from node 89 to node 83, is almost identical to PIGAME. The only difference is the requirement to kill the extra 2 Conway has in his formula for N_E. This comes from the fact that Wallis' infinite product formula is for $\frac{\pi}{2}$, so he needs to double his numerator to approximate π. In contrast, we have an infinite product formula for $\sqrt{2}$, so a slight modification of Conway's argument is needed. We skip his first pass around the square region, instead first going into the upper triangular region of Phase 1 with initial values $r_5 = 4$ and $r_7 = 49$. From here, the first phase proceeds in the same fashion as Conway, and it is easy to check we reach node 83 with

$$r_2 = 0,\ r_3 = 1,\ r_5 = E,\ r_7 = 10^n,\ r_{11} = 0,\ r_{59} = 1$$

where E is a very large even number. For Conway, it suffices to know that each pass around the square region "at least doubles" r_5 (which is initially set to 4) and keeps it even. This is because there is a well-defined truncation π_E for π for any even number E. However, for Catalan's formula for $\sqrt{2}$ to be appropriately truncated, one has to check that $E \equiv 2 \bmod 4$. Entering the square region for the first time, we have $r_5 = 4$, and we exit with $r_4 = 10$. In fact if we enter the square region with $r_5 = 4j + 2$, we exit with $r_5 = 4(2j + 1) + 2$, as the reader can easily check.

In the second phase, from node 83 to node 41, the essential point is that we copy Conway but modify how much we subtract from r_5 during each loop to generate N_E. However, we have to

also generate D_E, which has a different form than the denominator of π_E. We break each pass around the region into part (i), where we travel up from node 107, and part (ii), going down from node 61. Just as in PIGAME, part (i) sets $r_7 = 0$ and multiples r_5 by r_7, storing this number in r_{11}. However we also transfer r_{59} to r_{23} and reset $r_{59} = 0$.

Moving onto part (ii) of our loop, the hierarchy of arrows (corresponding to the order we carry out the operations) becomes more delicate. As in PIGAME, we transfer r_{11} to r_3, while preserving r_{11}. Note r_5 decreases by 2 (as opposed to Conway, who decreases r_5 by 1). With this modified value of r_5, we multiply r_3 by r_5, storing this in r_7. Then we add 1 to r_5 (storing it in r_{17}), and multiply this new number by r_{23}, storing it in r_{59}. The program continues in this phase until r_5 reaches a value of 2. The program then starts phase (i) of the final loop, but cannot go to phase (ii) of the loop and exits to start phase (iii) at node 41. At the end of the second phase $r_{11} = 10^n N_E$ and $r_{23} = D_E$. In the following chart, we summarize how each register updates during the second phase, breaking each loop into (i) and (ii) schematically.

up (i)	down (ii)
$r_{11} = r_5.r_7$	$r_3 = r_{11}$
$r_5 = r_5$	$r_5 = r_5 - 2$
$r_7 = 0$	$r_7 = r_3 \cdot r_5$
$r_{23} = r_{59}$	$r_{17} = r_5 + 1$
$r_{59} = 0$	$r_{59} = r_{23} \cdot r_{17}$

To clarify the proof, let us explicitly perform four loops of the second phase (numbered I–IV) in the following table. Each loop is broken into parts (i) and (ii).

I(i)	I(ii)	II(i)	II(ii)
$r_{11} = 10^n E$	$r_5 = E - 2$	$r_{11} = (E-2)^2$	$r_5 = E - 4$
$r_5 = E$	$r_7 = E - 2$	$r_5 = E - 2$	$r_7 = 10^n(E)(E-4)$
$r_{23} = 1$	$r_{17} = E - 1$	$r_7 = 0$	$r_{17} = E - 3$
$r_7 = r_{59} = 0$	$r_3 = 10^n E$	$r_{23} = E - 1$	$r_3 = (E-2)^2$
	$r_{59} = E - 1$		$r_{59} = (E-1)(E-3)$

III(i)	III(ii)
$r_{11} = 10^n(E)(E-4)^2$	$r_5 = E - 6$
$r_5 = E - 4$	$r_7 = (E-2)^2(E-6)$
$r_7 = 0$	$r_{17} = E - 5$
$r_{23} = (E-1)(E-3)$	$r_3 = 10^n(E)(E-4)^2$
	$r_{59} = (E-1)(E-3)(E-5)$

IV(i)	IV(ii)
$r_{11} = (E-6)^2(E-2)^2$	$r_5 = (E-8)$
$r_5 = (E-6)$	$r_7 = 10^n(E)(E-4)^2(E-8)$
$r_7 = 0$	$r_{17} = (E-7)$
$r_{23} = (E-1)(E-3)(E-5)$	$r_{59} = (E-1)(E-3)(E-5)(E-7)$
$r_{59} = 0$	$r_3 = (E-6)^2(E-2)^2$

Continuing on one more loop and recording the key register of interest, note that at the end of loop V(i) we have $r_{11} = 10^n E(E-4)^2(E-8)^2$. Since $r_5 \equiv 2 \bmod 4$, the program will complete an even number of full loops until $r_5 = 2$. It will then go into phase (i) of an odd numbered loop, but it cannot go into phase (ii) and so the program passes to the third phase with $r_{11} = 10^n N_E$ as claimed.

Moving into the third phase, we copy with obvious modifications the third phase of PIGAME to compute the n-th decimal of $\sqrt{2}_E$. The balance of the proof will involve two steps.

Step 1 Establish the inequality

$$|\sqrt{2} - \sqrt{2}_E| < \frac{2\sqrt{2}}{E}. \tag{3}$$

Step 2 Show that $10^n\sqrt{2}_E$ is not an integer.

Since $\frac{2\sqrt{2}}{E} < 10^{-n}$, Steps 1 and 2 together prove that the program computes the n-th term in the decimal expansion of $\sqrt{2}$, just as in PIGAME. To prove Equation (3), firstly we show $\sqrt{2}_E > \sqrt{2}$. If to the contrary $\sqrt{2}_{E_0} < \sqrt{2}$ for some $E_0 = 4j + 2$, where $j \in \mathbb{N}$, then

$$\sqrt{2}_{E_0+4} < \sqrt{2}_{E_0} \iff E_0^2 + 4E_0 < E_0^2 + 4E_0 + 3$$

which is obviously true. Since the sub-sequence $\sqrt{2}_{4j+2} \to \sqrt{2}$, we again easily derive a contradiction in the exact same manner as in the proof of PIGAME. Setting

$$\sqrt{2}_{\tilde{E}} = \left(\frac{E}{E+2}\right)\sqrt{2}_E,$$

an analogous proof shows that $\sqrt{2}_{\tilde{E}} < \sqrt{2}$. In summary for all $E = 4j + 2$ we have

$$\sqrt{2}_{\tilde{E}} < \sqrt{2} < \sqrt{2}_E. \tag{4}$$

However, Equation (4) is equivalent to Equation (3), as

$$|\sqrt{2} - \sqrt{2}_E| \stackrel{\Delta}{=} \sqrt{2}_E - \sqrt{2} < \frac{\sqrt{2}}{x} \iff \sqrt{2}_E < \sqrt{2}\left(\frac{x+1}{x}\right)$$

where $\stackrel{\Delta}{=}$ uses the second inequality of Equation (4). However Equation (4) establishes $\sqrt{2}_{\tilde{E}} < \sqrt{2}$. Choose now $x = E/2$ to obtain

$$\sqrt{2}_{\tilde{E}} = \left(\frac{x}{x+1}\right)\sqrt{2}_E$$

and we see that Equation (3) immediately follows from the first inequality in Equation (4).

The final step is to prove $\frac{10^n p}{q}$ is never an integer, where $\sqrt{2}_E = \frac{p}{q}$ with p and q coprime. This is directly analogous to our explanation for PIGAME, and the proof of Theorem A is now complete. □

7. NR$\sqrt{2}$GAME

7.1. **Description.** The standard way to approximate $\sqrt{2}$ is to use Heron's algorithm, or equivalently the Newton–Raphson method applied to the function $f(x) = x^2 - 2$ with initial guess $x_1 = p_1/q_1 = 1/1$. This updates via

$$x_{k+1} = \frac{p_{k+1}}{q_{k+1}} = \frac{p_k^2 + 2q_k^2}{2p_k q_k}.$$

We claim that computing x_{2n} will generate a rational number sufficiently close to $\sqrt{2}$ to agree to n decimal places. Encoding this as a FRACTRAN program is our second main result.

Theorem B. *Starting at $2^n \cdot 89$, the following FRACTRAN code terminates at $2^{\sqrt{2}(n)}$:*

$$\frac{4979909}{89}, \frac{227,123,851}{466}, \frac{233}{239}, \frac{11809}{23533}, \frac{241}{251}, \frac{60,993}{1687}, \frac{267}{723}, \frac{267}{257}, \frac{17355}{2827}, \frac{277}{267}, \frac{271}{277}, \frac{3047}{1355},$$

$$\frac{241}{277}, \frac{2959}{1205}, \frac{233}{241}, \frac{283}{233}, \frac{859579}{8207}, \frac{283}{281}, \frac{24278273}{18961}, \frac{307}{283}, \frac{313}{5833}, \frac{3170}{2191}, \frac{313}{317}, \frac{331}{313}, \frac{2359}{1655},$$

$$\frac{307}{331}, \frac{311}{307}, \frac{8903}{622}, \frac{347}{307}, \frac{359}{14227}, \frac{3350}{15437}, \frac{359}{353}, \frac{367}{359}, \frac{16039}{16515}, \frac{367}{347}, \frac{17101}{694}, \frac{379}{347}, \frac{397}{9475}, \frac{389}{397},$$

$$\frac{3970}{18,283}, \frac{409}{397}, \frac{401}{409}, \frac{19223}{2005}, \frac{379}{409}, \frac{421}{379}, \frac{12151}{2947}, \frac{421}{419}, \frac{283}{40837}, \frac{467}{421}, \frac{433}{13543}, \frac{6465}{4763}, \frac{433}{431},$$
$$\frac{443}{433}, \frac{5423}{2215}, \frac{443}{439}, \frac{467}{443}, \frac{457}{467}, \frac{7}{30619}, \frac{457}{3}, \frac{922}{457}, \frac{5093}{3227}, \frac{461}{463}, \frac{457}{463}, \frac{449}{1024}, \frac{449}{3}, \frac{449}{67}, \frac{1}{449}.$$

This list of fractions was generated from the flow chart in Figure 5 following Conway's algorithm. In the flow chart $1/C$ denotes the following list of fractions;

$$\frac{1}{3}, \frac{1}{67}, \frac{1}{1024}.$$

The program initializes with $r_2 = n$. For the reader's ease, we will break the flow chart into phases (i), (ii), etc., in descending order. Phase (i) terminates at the 283 node. Initially the algorithm sets $r_{11} = r_{29} = r_{67} = 1$. Our initial guess for $\sqrt{2}$ is $x_1 = r_{29}/r_{67} = 1/1$. In general, we set $x_j = p_j/q_j$, where $p_j = r_{29}$ and $q_j = r_{67}$. As we move along phase 1, note we end with $r_{97} = 2n$. This is the number of iterations of the NR algorithm we must perform. Phase (i) also sets $r_{11} = 10^n$ as the reader can check.

In Phase (ii), we transfer both p_j and q_j, where $1 \leq j \leq 2n$ is the current stage of the NR algorithm, to three new registers to facilitate computation later. Phase (ii) ends when we reach the 347 node. Here $r_{29} = p_j^2$. Similarly phase (iii) ends when we reach the 379 node with $r_7 = 2q^2$, and phase (iv) ends at the 379 node with $r_{67} = 2p_jq_j$. We then travel back up to phase (ii) with

$$r_{29} = p_j^2 + 2q_j^2, \quad r_{67} = 2p_jq_j, \quad \text{and} \quad r_5 = 10^n.$$

This key step encodes the iterative loop. Here r_{13} decreases by 1, and we updated from x_k to x_{k+1} as we travel back through the flow chart to phase (ii).

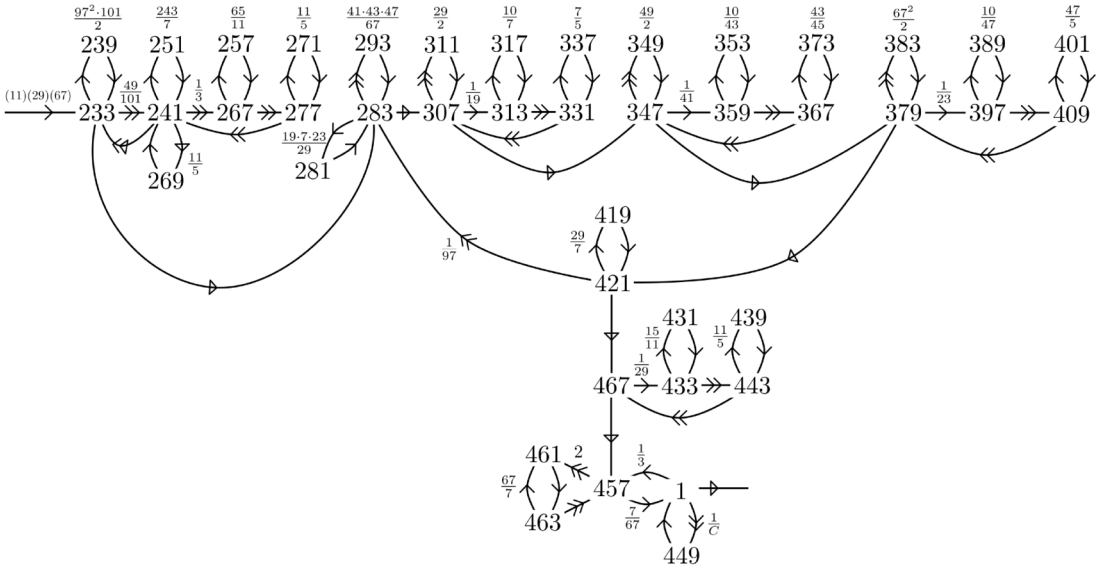

FIGURE 5. The flow chart for NR$\sqrt{2}$GAME

After $2n$ iterations, we move to phase (v) at node 467. Here we firstly multiply $r_{29} = p_{2n}$ by $r_{11} = 10^n$ (storing it in r_3) before, à la PRIMEGAME, computing the integer part of $\frac{10^n p_{2n}}{q_{2n}}$ modulo 10. This is the n-th term in the decimal expansion of x_{2n}. Now the mechanisms of the algorithm have been explained, we have to prove that x_{2n} is sufficiently close to $\sqrt{2}$ that both numbers agree in the n-th decimal place. To this end, we need the following lemma.

Lemma 7.1. *Suppose that f is a smooth function on $[1,2]$ with $|f'| \geq L$ and $|f''| < M$ for some L, $M > 0$. If $f(r) = 0$, then the error which arises from applying the Newton-Raphson algorithm to f, starting at x_1, N times is given by*

$$|x_{N+1} - r| < \frac{M}{2L}|x_N - r|^2.$$

Proof of Theorem B. To compute the n-th decimal digit of $\sqrt{2}$, we need to estimate $\epsilon_N := |x_N - \sqrt{2}|$. For later use $\epsilon_1 = \frac{1}{2}$. By induction, it is clear that $x_k \in [1,2]$ for all $k \in \mathbb{N}$. Applying the standard error estimates for Newton's method with $f(x) = x^2 - 2$, we have $|f''| = 2$ and $2 \leq |f'(x)| \leq 4$ on $[1,2]$. We iterate the error bound from Lemma 7.1 to obtain

$$\epsilon_N < \left(\frac{1}{2}\right)\epsilon_{N-1}^2 < \left(\frac{1}{4}\right)\epsilon_{N-2}^4 \cdots < \frac{\epsilon_1^{2N}}{2^N} = \frac{1}{2^{3N}}.$$

This shows that

$$\epsilon_N < \frac{1}{10^{n+1}} \iff \frac{1}{2^{3N}} < \frac{1}{10^{n+1}}$$

which clearly holds if $N = 2n$. With this error bound x_N and $\sqrt{2}$ must agree up to the n-th decimal place, once we know that $\frac{10^n p_{2n}}{q_{2n}}$ is not an integer.

To establish this last claim, assume to the contrary that

$$\frac{10^n(p_{2n-1}^2 + 2q_{2n-1}^2)}{2p_{2n-1}q_{2n-1}} \tag{5}$$

is an integer. With $p_1 = q_1 = 1$, it follows p_n is odd and q_n is even for all $n > 1$. Let λ be a prime divisor of q_{2n-1}. Then $\lambda = 5$ or $\lambda | p_{2n-1}^2 + 2q_{2n-1}^2$, in which case $\lambda | p_{2n-1}$. So every prime divisor of q_{2n-1} aside from 5 divides into p_{2n-1}, meaning we can write

$$\frac{p_{2n-1}}{q_{2n-1}} = \frac{\mu}{5^j}$$

where μ is an even integer with $\gcd(5, \mu) = 1$. Feeding this into Equation (5) yields

$$\frac{10^n(p_{2n-1}^2 + 2q_{2n-1}^2)}{2p_{2n-1}q_{2n-1}} = \frac{10^n(\mu^2 + 2(5^{2j}))}{\frac{2\mu}{5^j}} \in \mathbb{N}$$

which implies that

$$\frac{5^{3j+n}2^n}{\mu} \in \mathbb{N},$$

whence $\mu = 2^r$ for some $r \in \mathbb{N}$. Putting this all together yields

$$\frac{p_{2n-1}}{q_{2n-1}} = \frac{2^r}{5^j}.$$

Cross-multipying yields a contradiction as it implies an even number is equal to an odd one. □

REFERENCES

[1] Catalan, E. *Sur la constante d'Euler et la fonction de Binet*, C.R. Acad. Sci. Paris Sér. I Math. 77 (1873), 198-201.

[2] Conway, J.H. *FRACTRAN: a simple universal programming language for arithmetic*, Open Problems in Communication and Computation, Springer-Verlag New York Inc. (1987), 4–26.

[3] Conway, J.H. *Unpredictable Iterations*, Proc 1972 Number Tehory Conf., Univ. Colorado, Boulder, pp. 49–52.

[4] Guy, R.K. *Conway's prime-producing machine*, Math. Mag. 56 (1983), no. 1, 26–33. 33.

[5] Lagarias, J.C. *Conway's Work on Iteration: In memory of John Horton Conway (1937–2020)* The Mathematical Intelligencer, (06), 2021, Vol.43 (2), p.3–9.

[6] Penrose, R. *The Emporer's New Mind*, Oxford University Press, 1989.

[7] Sondow, J. and Hi, H. *New Wallis- and Catalan-type infinite products for π, e and $\sqrt{2+\sqrt{2}}$, Amer. Math. Monthly, 117 (2010), no. 10, 912-917.*

APPENDIX: CONVERTING A FLOW CHART INTO A FRACTRAN CODE

The following code converts our flow charts into their corresponding list of FRACTRAN fractions. For a single node in the flow chart, we write a line describing to what node it is connected and through what fraction. Line 17 then shows how you convert that line into a series of fractions. This is described more in Section 8 of [2]. In a single line the order in which the connections are listed handles the hierarchy of the arrows.

```
#Each line should be  P, a/b->Q, c/d->R, ...

with open('fracn.txt') as file:
    fractionList = list()
    fractionFactored = list()
    for line in file:
        entries = line.split(', ')
        P = entries[0]
        for entry in entries[1:]:
            print(entry)
            a,temp = entry.split('/')
            b,Q = temp.split('->')
            Q = Q.strip()
            fractionFactored.append(a+'*'+Q+'/'+b+'*'+P)
            aQ = str(int(a)*int(Q))
            bP = str(int(b)*int(P))
            fractionList.append(aQ+'/'+bP)
    print(fractionFactored)
    print(fractionList)

with open('fractions.txt', 'w') as file:
    file.write(', '.join(fractionList))

with open('fractionsFactored.txt', 'w') as file:
    file.write(', '.join(fractionFactored))
```

Khushi Kaushik graduated in 2025 from CSU Fullerton with a degree in Computer Science. Broadly interested in machine learning and data science, she is starting her M.Sc. in Comp. Sci. at UC San Diego.
Tommy Murphy is Professor at Cal State Fullerton, having completed his Ph.D. under Jürgen Berndt at UCC and postdoctoral fellowships at the Université Libre de Bruxelles (Belgium) and McMaster University (Canada). His research interests span Riemannian and Kähler geometry, focused specifically on Einstein manifolds and symmetric spaces. He also maintains an active interest in the history of mathematics and collaborating with undergraduate students.
David Weed graduated in 2024 from CSU Fullerton with a degree in mathematics, and is currently undertaking his Ph.D. in mathematics at UC Riverside. He is exploring a range of topics in pure mathematics particularly around the representation and character varieties of surfaces and 3-manifolds.

(Kaushik) DEPARTMENT OF COMPUTER SCIENCE, CALIFORNIA STATE UNIVERSITY FULLERTON.
E-mail address: kkaushik@ucsd.edu

(Murphy) DEPARTMENT OF MATHEMATICS, CALIFORNIA STATE UNIVERSITY FULLERTON.
E-mail address: tmurphy@fullerton.edu
URL: http://www.fullerton.edu/math/faculty/tmurphy/

(Weed) DEPARTMENT OF MATHEMATICS, UC RIVERSIDE.
E-mail address: david@davidweed.net