CDM

Iteration

Klaus Sutner
Carnegie Mellon University
www.cs.cmu.edu/~sutner

Battleplan

- The Collatz Problem
- Ducci Sequences
- Iteration and Orbits
- Fixed Points
- Calculus and Fixed Points
- Finding Cycles
- Fast Iteration
- Decidability and Conway
- Theory of Fixed Points

The Collatz Problem

A Termination Question

Here is a seemingly innocent question: Does the following program halt for all $x \geq 1$?

```
while( x > 1 )   // x positive integer
{
        if( x even )
            x = x/2;
        else
            x = 3 * x + 1;
}
```

The body of the while-loop could be written as a LOOP program of nesting depth 1, so there is nothing complicated going on there.

The Collatz Loop Program

```
//  Collatz:  x --> z

        e = 1; o = 0;
        do x : t = e; e = o; o = t; od

        u = 0; v = 0;
        do x : t = u; u = v; v = t; u++; od

        w = 0;
        do x : w++; w++; w++; od
        w++;

        do e : z = u; od
        do o : z = w; od
```

Exercise 1. *Figure out exactly how this program works.*

Collatz $3x + 1$

Let's ignore complexity issues for the time being. The Collatz Problem revolves around the following function C on the positive integers. There are several variants of this in the literature, under different names.

$$C(x) = \begin{cases} 1 & \text{if } x = 1, \\ x/2 & \text{if } x \text{ even}, \\ (3x+1)/2 & \text{otherwise.} \end{cases}$$

Here are the first few values.

x	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	...
$C(x)$	1	1	5	2	8	3	11	4	14	5	17	6	20	7	23	8	...

Digression: The Name of the Game

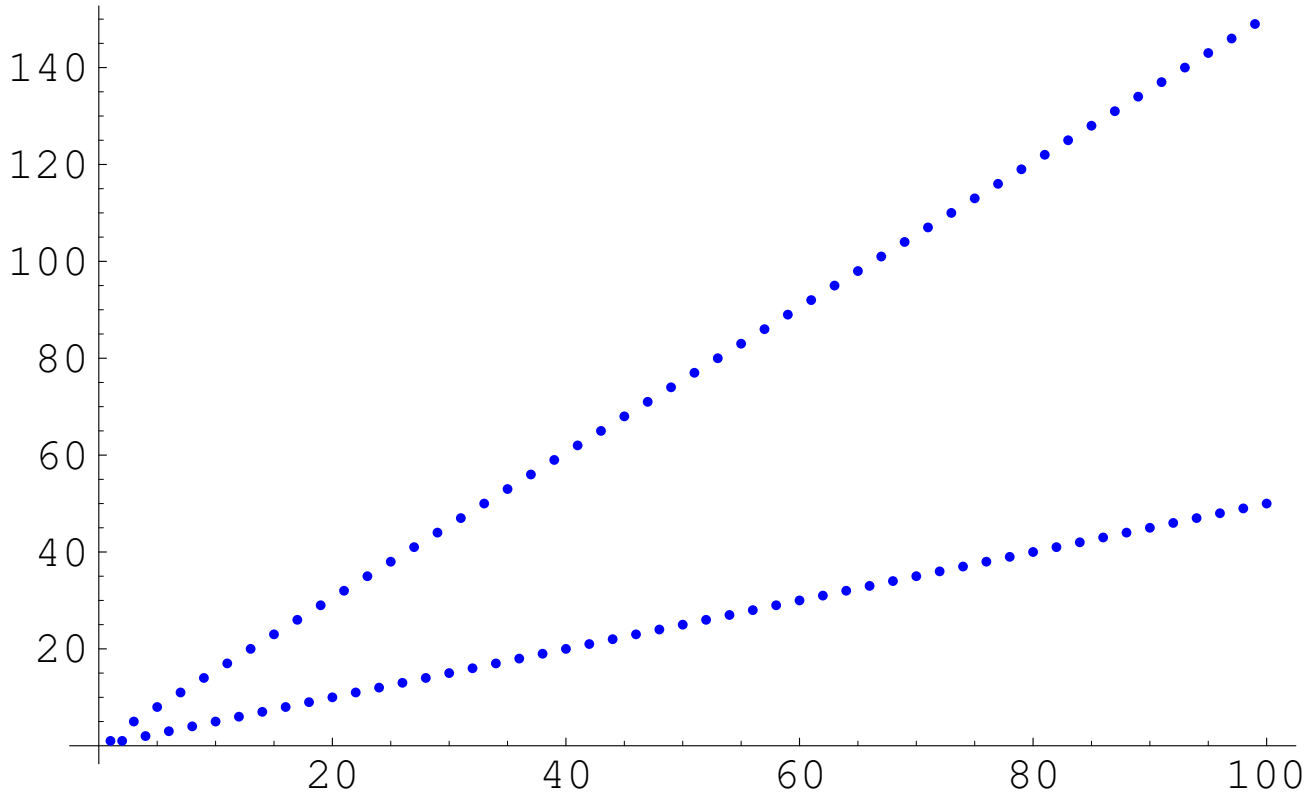
The Collatz problem was invented by Lothar Collatz around 1937, when he was about 20 years old.

Since then, it has assumed a number of aliases:

Ulam, Hasse, Syracuse, Hailstone, Kakutani, . . .

Amazingly, in 1985 Sir Bryan Thwaits wrote a paper titled "My Conjecture" claiming fathership. Talking about ethics . . .

Boring Plot



Repetition

Of course, we are interested not in single applications of C but in repeated application. In fact, the Collatz program keeps computing C until 1 is reached, if ever.

Starting at 18:

$$18, 9, 14, 7, 11, 17, 26, 13, 20, 10, 5, 8, 4, 2, 1, 1, 1, \ldots$$

Starting at 1000:

$$1000,500,250,125,188,94,47,71,107,161,242,121,182,91,137,$$

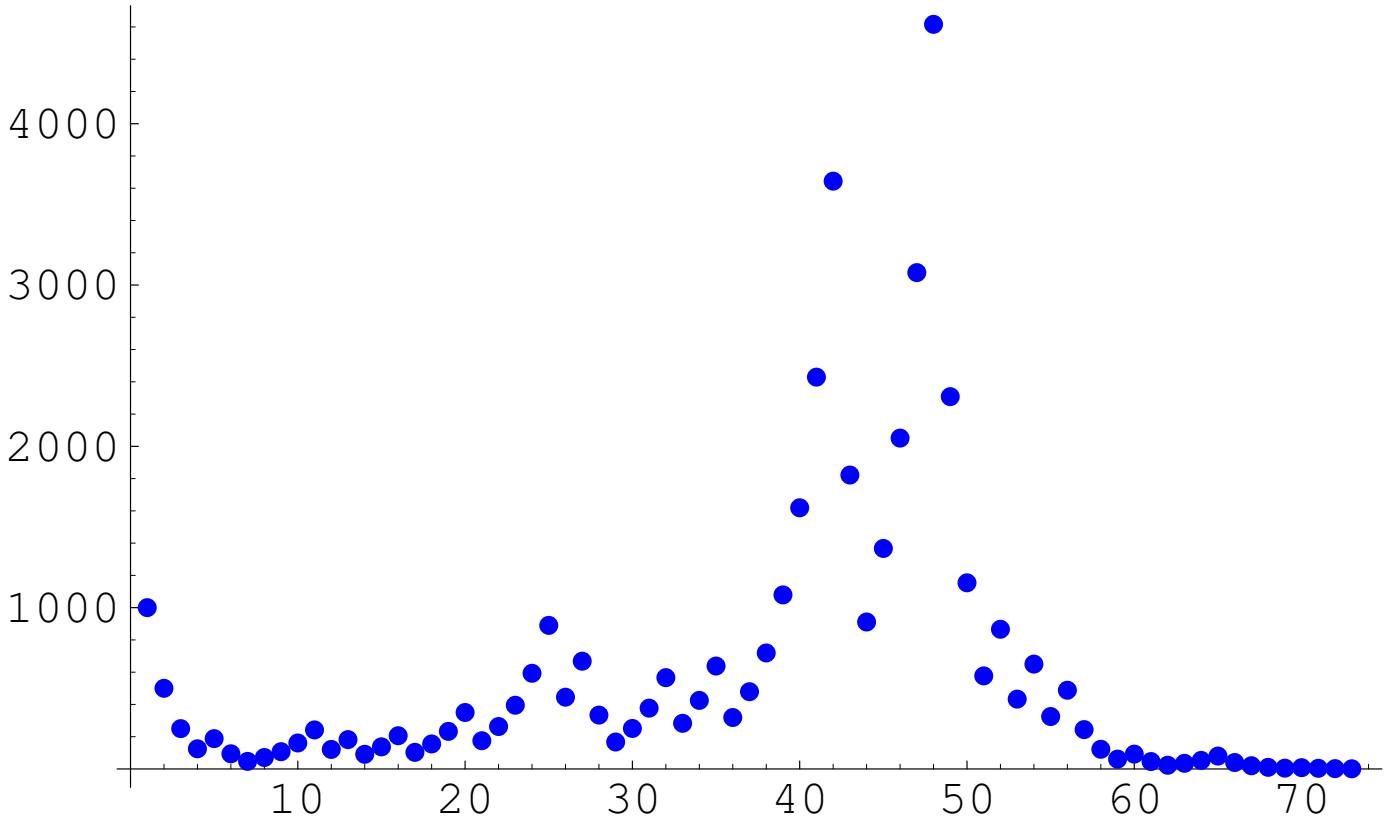
$$206,103,155,233,350,175,263,395,593,890,445,668,334,167,251,$$

$$377,566,283,425,638,319,479,719,1079,1619,2429,3644,1822,911,$$

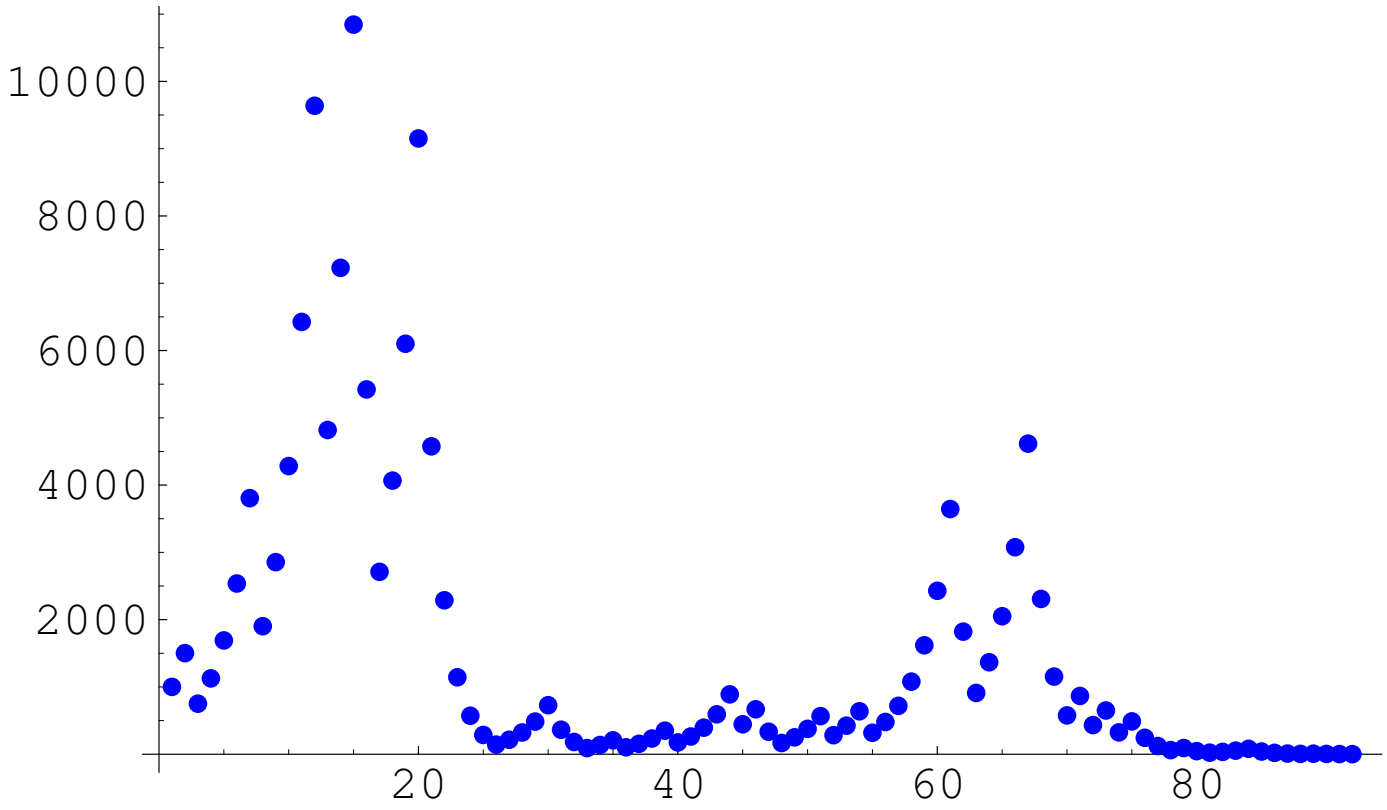
$$1367,2051,3077,4616,2308,1154,577,866,433,650,325,488,244,$$

$$122,61,92,46,23,35,53,80,40,20,10,5,8,4,2,1,1,1,1,\ldots$$

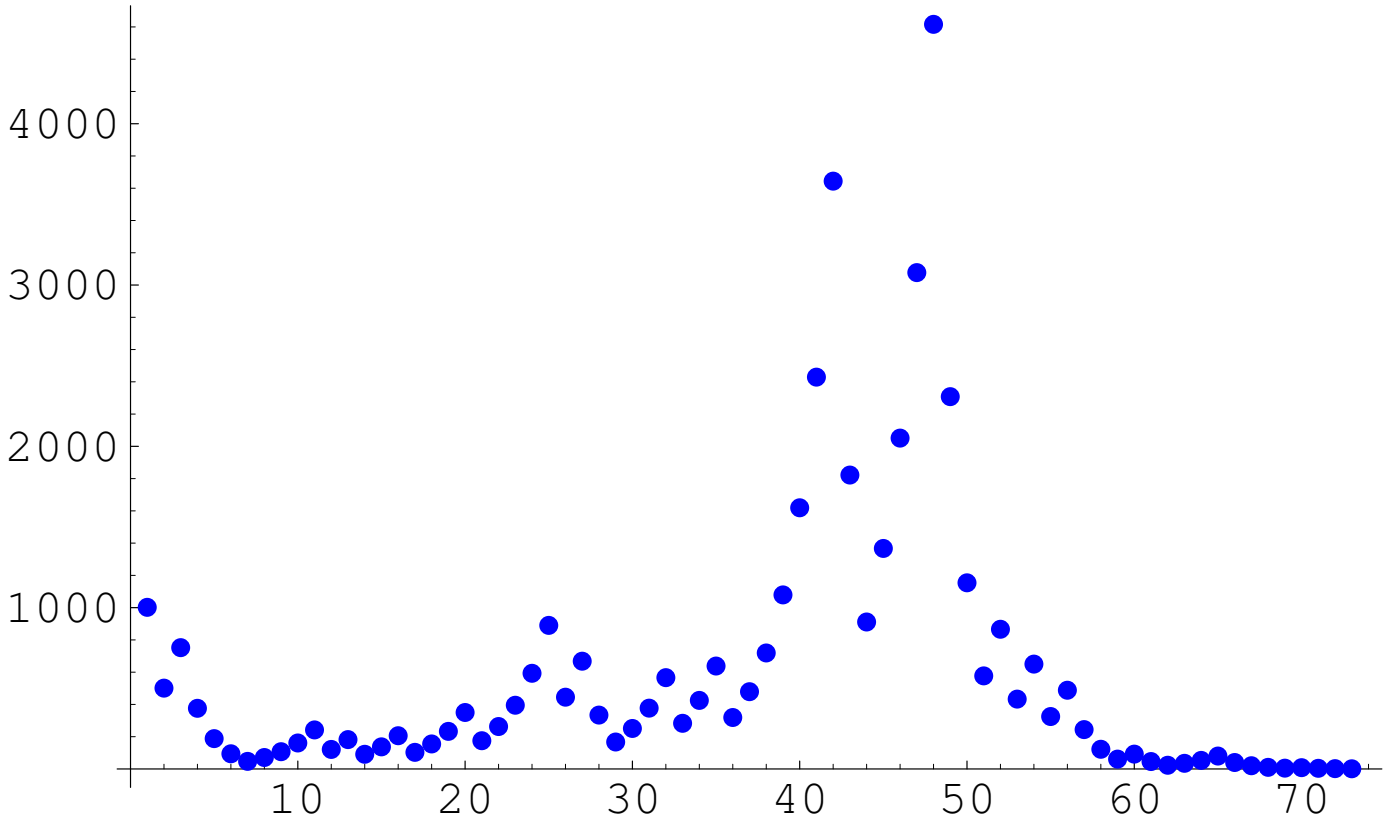
$$x = 1000$$



$$x = 1001$$

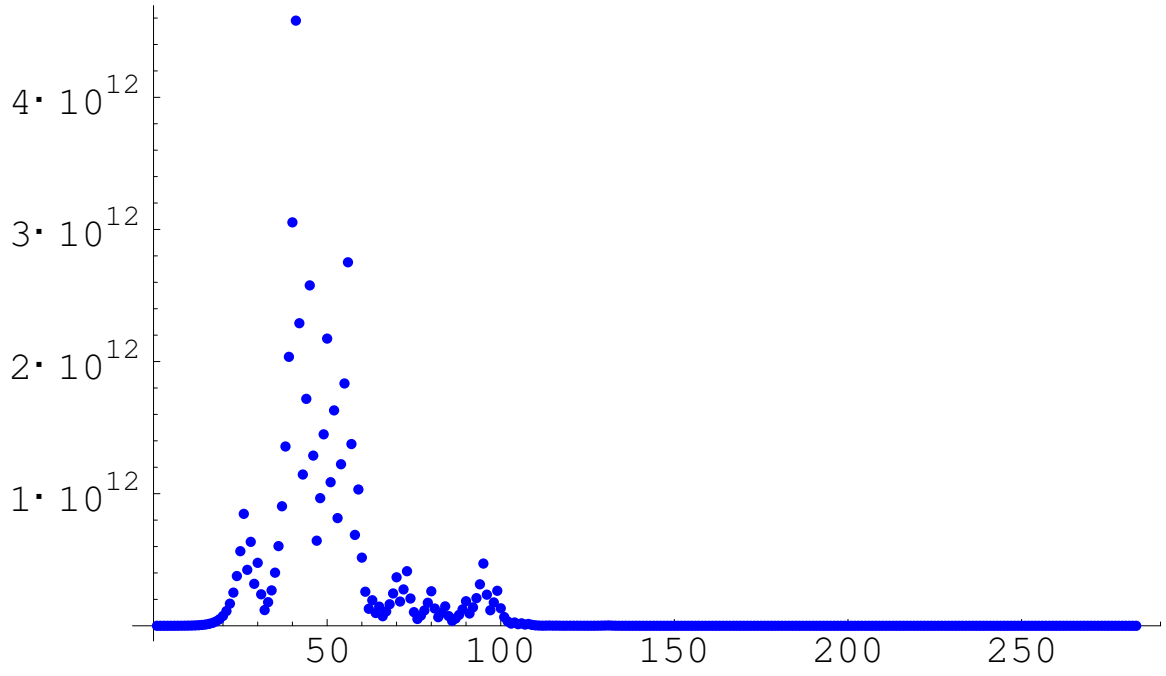


$$x = 1002$$



Near Powers of 2

Starting at $2^{25} - 1 \approx 3.35 \times 10^7$.



It takes 282 steps to get to 1.

Stopping Times

More computation shows that for all

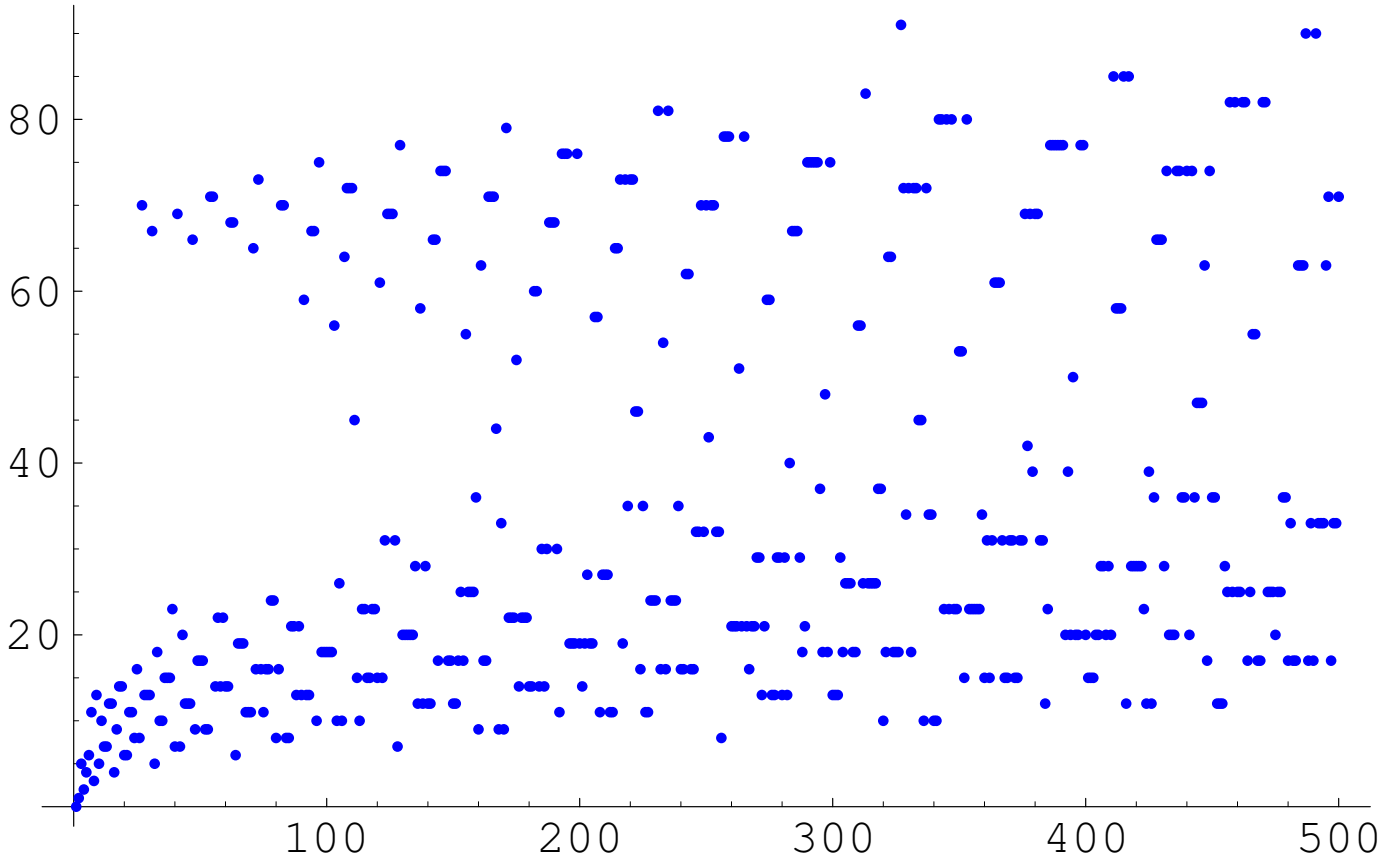
$$x \leq 3 \cdot 2^{53} \approx 2.7 \cdot 10^{16}$$

the program always halts: C reaches the fixed point 1. Many other values of x have also been tested.

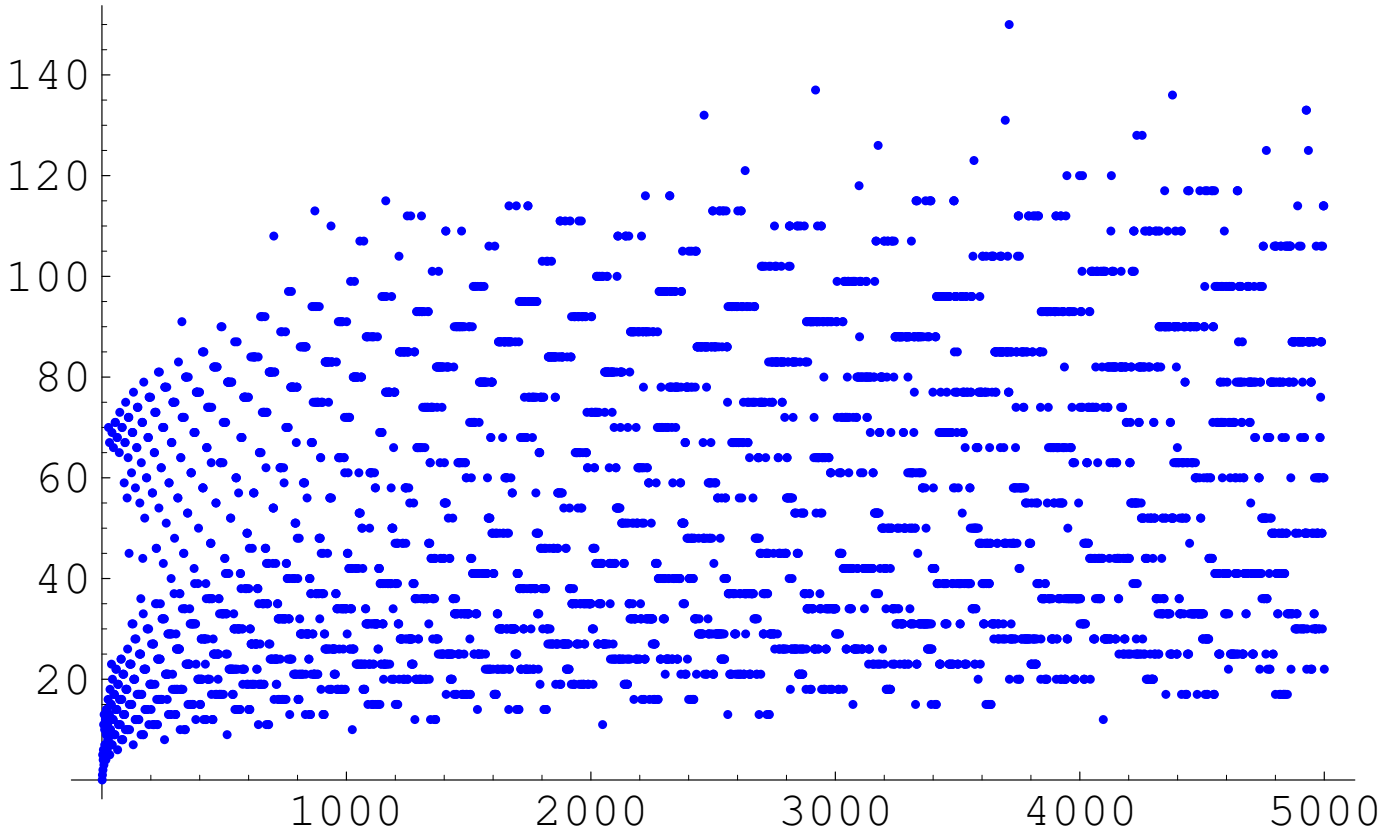
It is natural to try to investigate the *stopping time*: number of executions of the loop before 1 is reached.

The stopping time function seems slightly more regular than C itself, but it's still rather complicated.

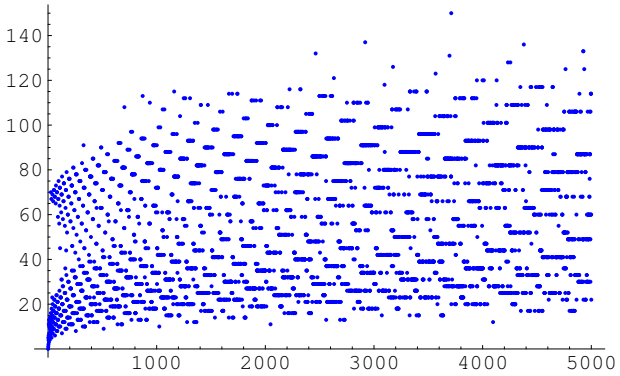
Stopping Times up to 500



Stopping Times up to 5000



Stopping Times up to 5000



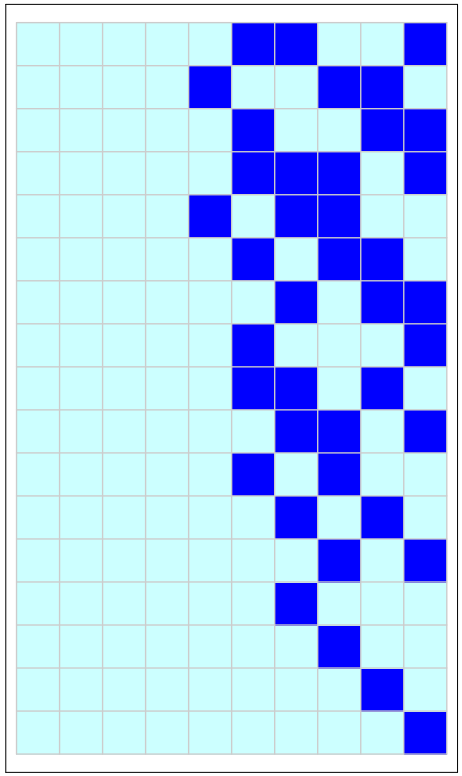
There clearly is some structure here (the dots are certainly not random), but what exactly is this mysterious structure?

E.g., we would like to predict the position of dot 5001, from knowledge of the first 5000 dots, without having to compute the orbit of $x = 5001$.

No one is currently able to do this (at least not for arbitrary values of 5001).

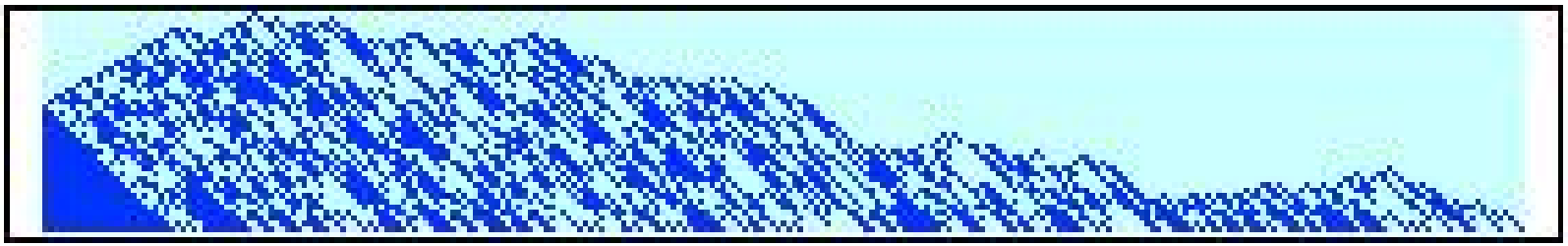
Orbit of 25 in Binary

Another potential way to gain insight in the behavior of the Collatz function is to plot the numbers in binary (an implementation of C for large arguments would use binary arrays anyways).

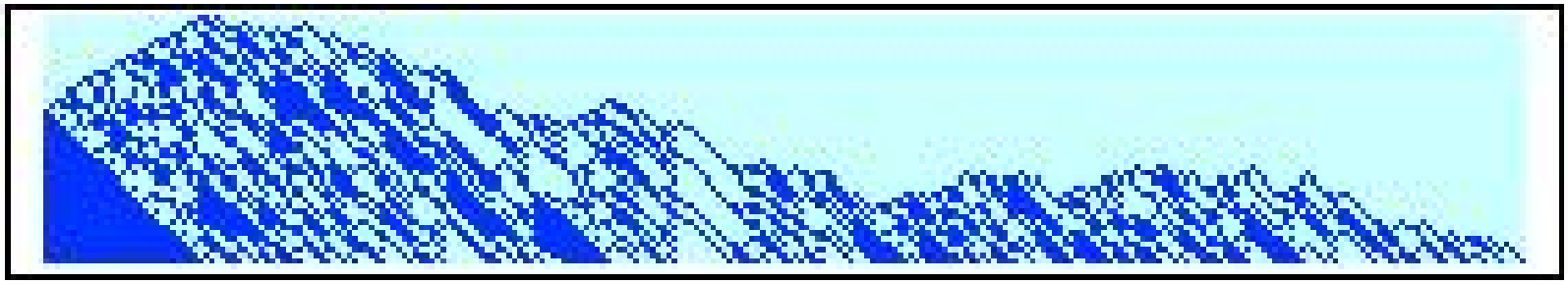


Collatz Mountains

$x = 2^{25} - 1 \approx 3.35 \times 10^7$



$x = 2^{27} - 1 \approx 1.34 \times 10^8$



Computing the Collatz Function

The question arises: what is the best way to compute the Collatz function? Let's say the input is given as an array of bits (of significant size, hundreds or thousands of bits).

Division by 2 is easy; for the odd case we have

$$C(x) = 2 \cdot x + x + 1$$

so we can scan over the input once and compute the output as we go along. In fact, we can write the output into the same array (more or less).

The whole computation is requires $cn + d$ steps where n is the number of bits and c and d are small constants.

Here is a more formal way of describing this method. Essentially, we can use a finite state machine to do the computation. The only twist is that we need machines with output rather than DFAs.

Digression: Transducers

It is easy to generalize DFAs to machines that generate output: associate each transition with an output symbol:

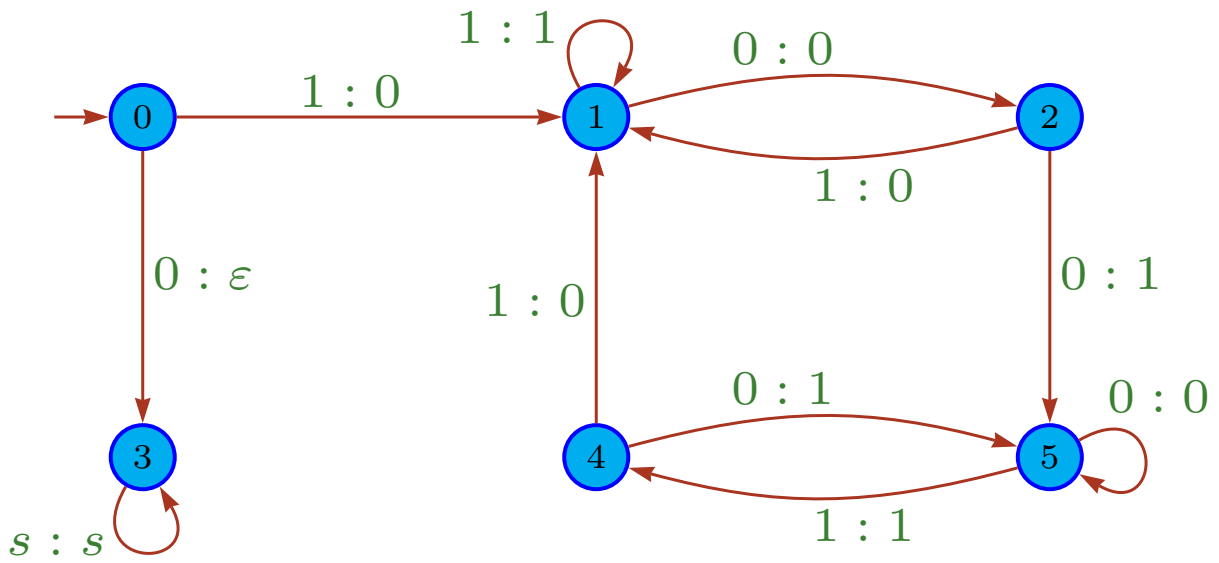
$$\delta(p, a) = (b, q)$$

meaning that the machine outputs b and performs a transition to state q, starting at state p and under input a.

So each input word $x \in \Sigma^\star$ produces an output word $y \in \Gamma^\star$ where Γ is the alphabet from which the b's are chosen.

Such a device is called a *transducer*: it translates strings from $\Sigma^\star$ to strings from $\Gamma^\star$.

A Collatz Transducer



A transducer for the Collatz function. One little glitch: the input must be coded as $x00$ where x is the reverse binary expansion (LSD first, pad right by two 0's).

Exercise 2. *Explain precisely how to compute the Collatz function efficiently.*

Ducci Sequences

Ducci Sequences

Here is another classical example, introduced by Enrico Ducci (1864-1940) in the 1930's.

Place 4 integers on a circle. Compute the absolute values of all differences of adjacent pairs of numbers. Write these values between the corresponding numbers and erase the numbers themselves.

Repeat.
What happens?

Ducci's problem was mostly forgotten till Honsberger's "Ingenuity in Mathematics" appeared in 1970.

Less Informally

We want to iterate the following function on $\mathbb{N}^4$:

$$D(x_1, x_2, x_3, x_4) = (|x_1 - x_2|, |x_2 - x_3|, |x_3 - x_4|, |x_4 - x_1|)$$

Is there anything one can say about the orbits?

Sample Ducci Sequences

Here are two samll examples, we iterate D on "random" initial conditions.

0	10	13	4	20
1	3	9	16	10
2	6	7	6	7
3	1	1	1	1
4	0	0	0	0

0	94	68	11	85
1	26	57	74	9
2	31	17	65	17
3	14	48	48	14
4	34	0	34	0
5	34	34	34	34
6	0	0	0	0

Somewhat surprisingly, after a few steps we reach the 0 vector which is, of course, a fixed point of the operation.

Exercise 3. *Produce initial values so that it takes many steps to reach* $\mathbf{0}$.

The Basics

The first observation is that $\mathbf{0}$ must always be reached after finitely many steps, regardless of the initial values.

Theorem. *Ducci*
After finitely many steps the fixed point $(0, 0, 0, 0)$ *is reached.*

It is rather surprisingly difficult to choose initial condition that lead to long runs before $\mathbf{0}$ is reached, but there is no bound on the number of steps.

Lemma. *Webb 1982*
There is no bound on the number of steps needed to reach $\mathbf{0}$.

To construct hard initial conditions, consider "tribonacci numbers"

$$t_n = t_{n-1} + t_{n-2} + t_{n-3} \qquad t_0 = 0 \qquad t_1 = t_2 = 1$$

and starting configurations $(t_{n-3}, t_{n-2}, t_{n-1}, t_n)$.

Digression: How About the Reals?

We might try to force divergence by using irrational or even transcendental numbers. At least the following valiant attempt fails.

0	0	$\sqrt{2}$	$\sqrt{7}$	π
1	$\sqrt{2}$	$-\sqrt{2}+\sqrt{7}$	$-\sqrt{7}+\pi$	π
2	$2\sqrt{2}-\sqrt{7}$	$-\sqrt{2}+2\sqrt{7}-\pi$	$\sqrt{7}$	$-\sqrt{2}+\pi$
3	$-3\sqrt{2}+3\sqrt{7}-\pi$	$\sqrt{2}-\sqrt{7}+\pi$	$\sqrt{2}+\sqrt{7}-\pi$	$-3\sqrt{2}+\sqrt{7}+\pi$
4	$4\sqrt{2}-4\sqrt{7}+2\pi$	$-2\sqrt{7}+2\pi$	$-4\sqrt{2}+2\pi$	$-2\sqrt{7}+2\pi$
5	$4\sqrt{2}-2\sqrt{7}$	$4\sqrt{2}-2\sqrt{7}$	$4\sqrt{2}-2\sqrt{7}$	$4\sqrt{2}-2\sqrt{7}$
6	0	0	0	0

Exercise 4. *Explain why this and similar attempts will fail to produce divergence.*

An Amazing Exception

One might conclude that convergence to $\mathbf{0}$ always occurs, but that is false. Let q be the real root of $x^3 - x^2 - x - 1 = 0$, so

$$q = \frac{1}{3}\left(1 + \sqrt[3]{19 - 3\sqrt{33}} + \sqrt[3]{19 + 3\sqrt{33}}\right) \approx 1.83929$$

Then $(1, q, q^2, q^3)$ does not converge. E.g. after 5 steps we get a term

$$\frac{1}{9}\left(24 - 4\sqrt[3]{19 + 3\sqrt{33}} - 6\left(19 + 3\sqrt{33}\right)^{2/3} + \left(19 - 3\sqrt{33}\right)^{2/3}\right.$$
$$\left.\left(-6 + 4\sqrt[3]{19 + 3\sqrt{33}}\right) + 4\sqrt[3]{19 - 3\sqrt{33}}\left(-1 + \left(19 + 3\sqrt{33}\right)^{2/3}\right)\right)$$

This was shown by M. Lotan in 1949.

How About $n \neq 4$?

Over $\mathbb{N}^n$ evolution does no longer lead to a fixed point in general, but does lead to a limit cycle.

Theorem. *Ciamberlini, Marengoni 1937*
Every Ducci sequence of width n ends in fixed point $\mathbf{0}$ if, and only if, n is a power of 2.

When n is not a power of 2 the sequence evolves to a limit cycle.

Exercise 5. *Why must any Ducci sequence over $\mathbb{N}^n$ are ultimately periodic (i.e., they lead to a limit cycle)?*

Example: $n = 5$

4	2	2	3	1	3	1	2	1	2	1	1
6	4	5	2	4	2	3	1	3	1	0	1
10	9	7	6	6	5	4	4	2	1	1	1
1	2	1	0	1	1	0	2	1	0	0	2
3	1	1	1	2	1	2	1	1	0	2	1

0	0	1	0	0	0	1	1	1	1	0	1	0	1	1	0
0	1	1	0	0	1	0	0	0	1	1	1	1	0	1	0
1	0	1	0	1	1	0	0	1	0	0	0	1	1	1	1
1	1	1	1	0	1	0	1	1	0	0	1	0	0	0	1
0	0	0	1	1	1	1	0	1	0	1	1	0	0	1	0

It is a bit surprising that after 12 steps the system is in a binary state. From then on, it repeats every 15 steps.

The Limit Cycles

Needless to say, the binary behavior in the example is no coincidence.

Theorem. *Burmester, Forcade, Jacobs 1978*

All Ducci sequences over the integers are ultimately periodic. The vectors on the limit cycle are binary in the sense that $x_i \in \{0, c\}$ *for some* c*.*

Once the vector is binary, the Ducci operator degenerates into exlusive or:

$$D(\boldsymbol{x}) = \boldsymbol{x} \text{ xor } L(\boldsymbol{x})$$

where L denotes the cyclic left-shift operation. We'll come back to this later in our discussion of cellular automata.

Iteration and Orbits

Repeated Function Application

Time for some formal definitions.

Definition 1. *Let $f : A \to A$ be a function (an endofunction). The kth power of f (or kth iterate of f) is defined by induction as follows:*

$$f^0 = I_A$$
$$f^k = f \circ f^{k-1}$$

Here I_A denotes the identity function on A and $f \circ g$ denotes composition of functions.

Informally, this just means: compose f $k-1$ times with itself.

$$f^k = \underbrace{f \circ f \circ f \circ \ldots \circ f}_{k \text{ terms}}$$

General Laws

Without any further knowledge about f there is not much one can say about the about the iterates f^k. But the following always holds.

Lemma 1. *Laws of Iteration*

1. $f^n \circ f = f^{n+1}$
2. $f^n \circ f^m = f^{n+m}$
3. $(f^n)^m = f^{n \cdot m}$

Proof by straightforward induction using associativity of composition.

So iteration behaves very much like ordinary exponentiation x^n of, say, real numbers.

Orbits

Definition 2. *The orbit of $a \in A$ under f is the infinite sequence*

$$\mathrm{orb}_f(a) = a, f(a), f^2(a), \ldots, f^n(a), \ldots$$

Note that an orbit is by definition a sequence, not the set $\{ f^i(a) \mid i \geq 0 \}$ of points that lie on the orbit. As a set, an orbit may well be finite.

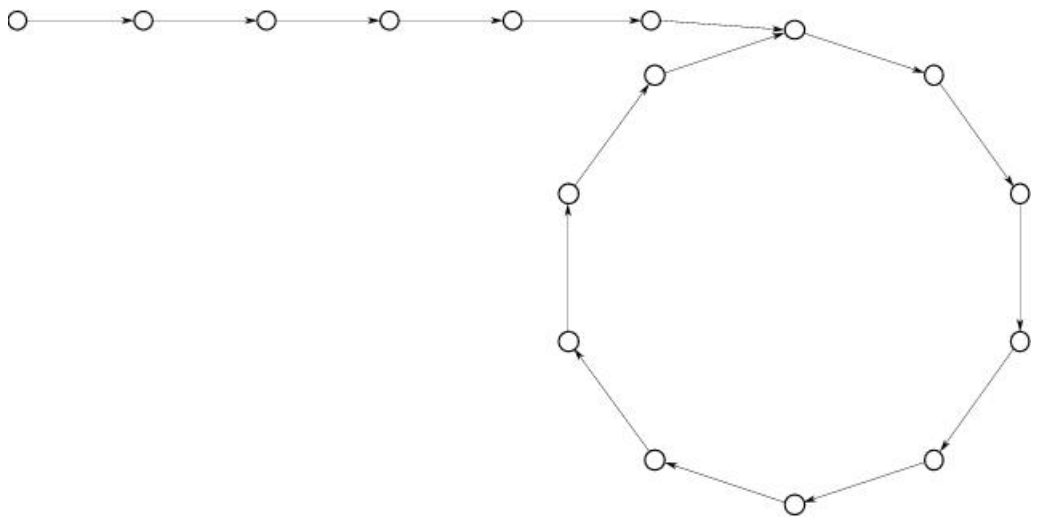
The set of all infinite sequences with elements from A is often written A^ω. Hence the orbit is an operation of type

$$\mathrm{orb} : (A \to A) \times A \to A^\omega,$$

it associates a function on A and element in A with an infinite sequence over A.

The Lasso

If the carrier set is finite, all orbits must ultimately wrap around:



What changes is only the length of the transient part and the length of the cycle.

Cycles and Periods

Definition 3. *Let $f : A \to A$ be a function.*

$a \in A$ is a fixed point of f if $f(a) = a$.

A sequence $a_0, \ldots, a_{n-1}$ in A is a cycle of f if $f(a_i) = a_{i+1 \bmod n}$. A cycle of length n is also called an n-cycle.

The orbit of a under f is periodic if it consists just of a cycle: $\exists\, p > 0\ f^p(a) = a$.

It is ultimately periodic if it ultimately winds up in a cycle: $\exists\, t, p > 0\ f^{t+p}(a) = f^t(a)$.

Cycles and fixed points are closely related:

$a_0, \ldots, a_{n-1}$ is an n-cycle of f iff a_0 is a fixed point of f^n.

Transient and Period

If A is finite, then any orbit of $f : A \to A$ must be ultimately periodic:

$$f^t(x) = f^{t+p}(x)$$

for some $t \geq 0$, $p > 0$, which values depend on x.

Definition 4. *The least t and p such that $f^t(x) = f^{t+p}(x)$ is the* transient length *and the* period length *of the orbit of x (wrt. f).*

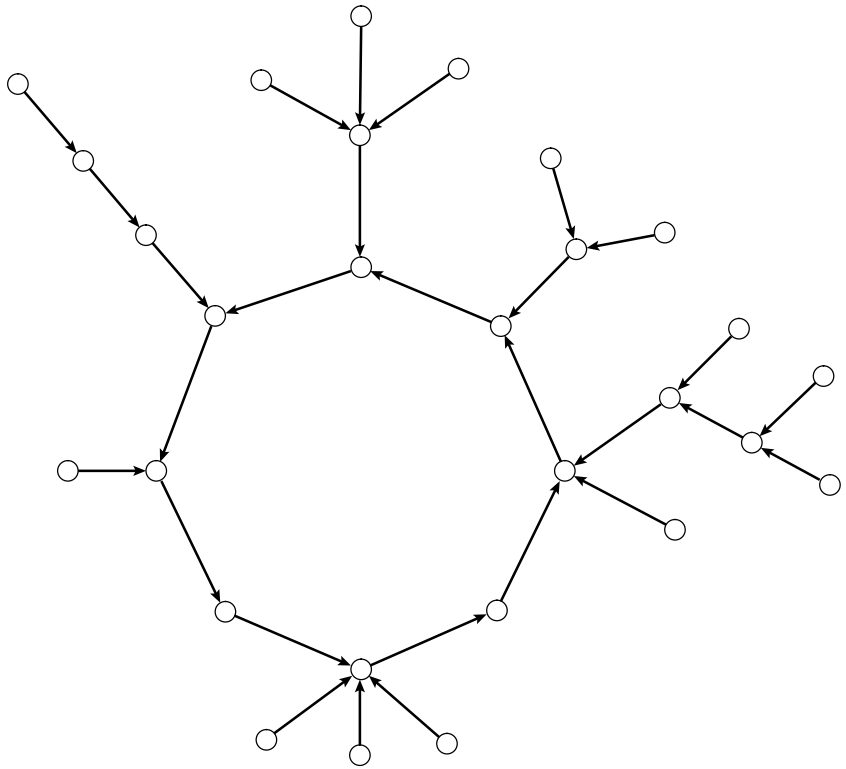
Thus, an orbit is periodic iff the transient is 0.

Also, a function on a finite set has only transients of length 0 iff the function is injective iff it is a permutation.

Limit Cycles

The lasso shows the general shape of any single orbit, but in general orbits overlap.

All orbits with the same limit cycle are sometimes called a *basin of attraction*.



The Functional Digraph

As the last picture shows, it is natural to think of f as a directed graph on the carrier set where the edges indicate the action of f

Definition 5. *The functional digraph (or diagram) of* $f : A \longrightarrow A$ *is defined as* $G_f = \langle A, E \rangle$ *where* $E = \{ (x, f(x)) \mid x \in A \}$.

Note that every vertex in G_f has outdegree 1, but indegrees may vary.

The non-trivial strongly connected components of the digraph are the limit cycles of the function.

The weakly connected components are the basins of attraction.

Analyzing the Diagramm

There are several natural parameters associated with the digraph that provide useful information about the function in question.

- Indegree. If all nodes have the same indegree k the function is k-to-1. Otherwise, determine the maximum/minimum indegree, the distribution of values.
- Periods. Count the number of limit cycles, and their length.
- Transients. Determine the length of the transients leading to limit cycles.

At least when the carrier set is finite we would like to be able to determine these parameters easily. Alas, even for relatively simple maps this turns out to be rather difficult.

Reachability

The geometric perspective afforded by the diagram als suggests to study path-existence problems.

Definition 6. *Let f be a function on A and $a, b \in A$ two points in A. Then point y is* reachable *from x if for some $i \geq 0$:*

$$f^i(x) = y.$$

In other words, point y belongs to the orbit of x.

Proposition. *Reachability is reflexive and transitive but in general not symmetric.*

Reachability is symmetric when A is finite and f injective (and therefore a permutation): each orbit then is a cycle and forms an equivalence class.

Confluence (aka Basins of Attraction)

Definition 7. *Let f be a function on A and $a, b \in A$ two points in A. Points a and b are* confluent *if for some $i, j \geq 0$:*

$$f^i(a) = f^j(b).$$

In other words, the orbits of a and b merge, they share the same limit cycle.

Proposition. *Confluence is an equivalence relation.*

Reflexivity and symmetry are easy to see, but transitivity requires a little argument. Let $f^i(x) = f^j(y)$ and $f^k(y) = f^l(z)$, assume $j \leq k$. Then with $d = k - j \geq 0$ we have

$$f^{i+d}(x) = f^{j+d}(y) = f^k(y) = f^l(z).$$

Each equivalence class contains exactly one cycle of f, and all the points whose orbits lead to this cycle – just as in the last picture.

Small Example

Here is a somewhat frivolous operation on binary lists: given L, replace the first element of L by 0, and then rotate to the left by 2 places.

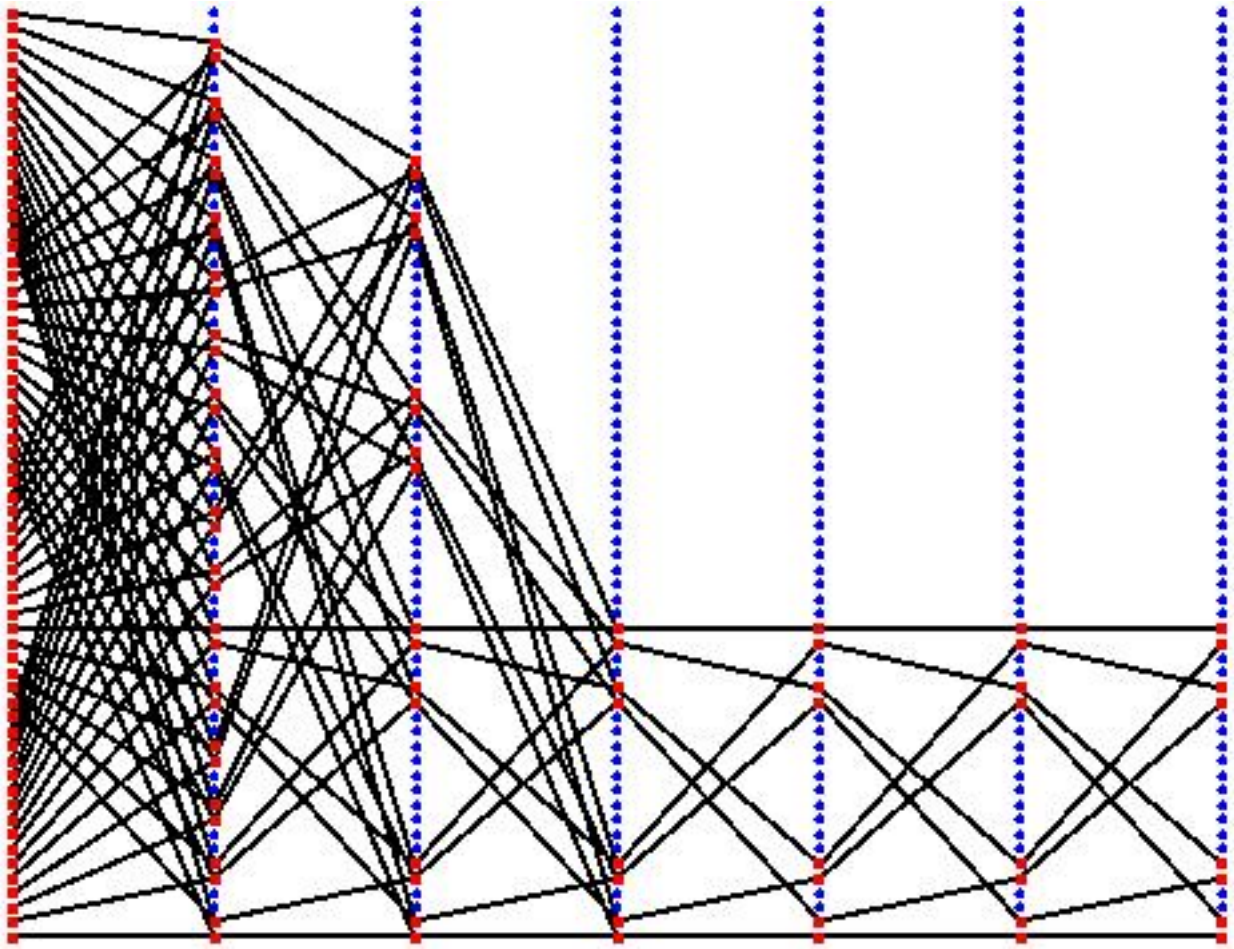
Here is the orbit of a generic list (with symbolic entries) of length 6 under this operation:

$$
\begin{array}{lcccccc}
0: & x_1 & x_2 & x_3 & x_4 & x_5 & x_6 \\
1: & x_3 & x_4 & x_5 & x_6 & 0 & x_2 \\
2: & x_5 & x_6 & 0 & x_2 & 0 & x_4 \\
3: & 0 & x_2 & 0 & x_4 & 0 & x_6 \\
4: & 0 & x_4 & 0 & x_6 & 0 & x_2 \\
5: & 0 & x_6 & 0 & x_2 & 0 & x_4 \\
6: & 0 & x_2 & 0 & x_4 & 0 & x_6
\end{array}
$$

So both transient and period are 3 in the generic case. But note the for special values of x_2, x_4 and x_6 the period may be shorter (ditto for the transient and x_1, x_3 and x_5).

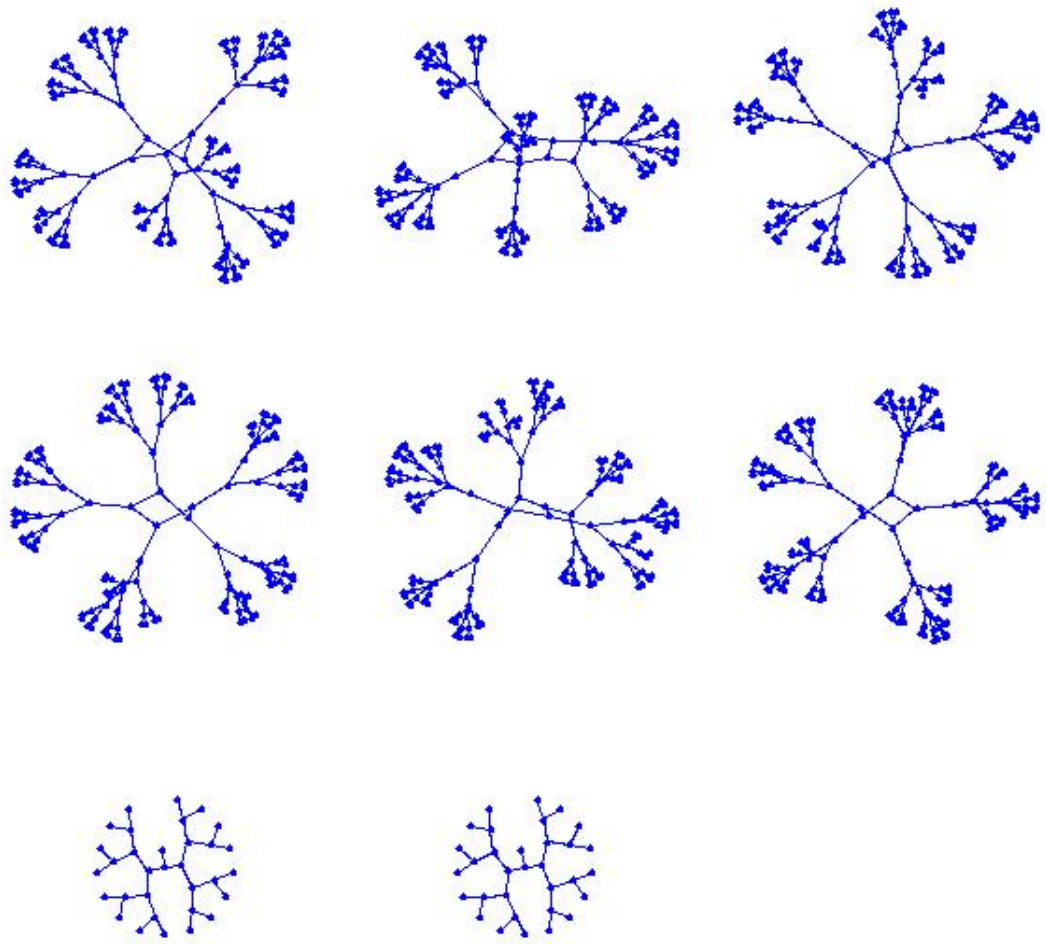
The following picture shows the behavior of all binary lists of length 6 under this operation.

All Orbits



There are two fixed points, and two 3-cycles.

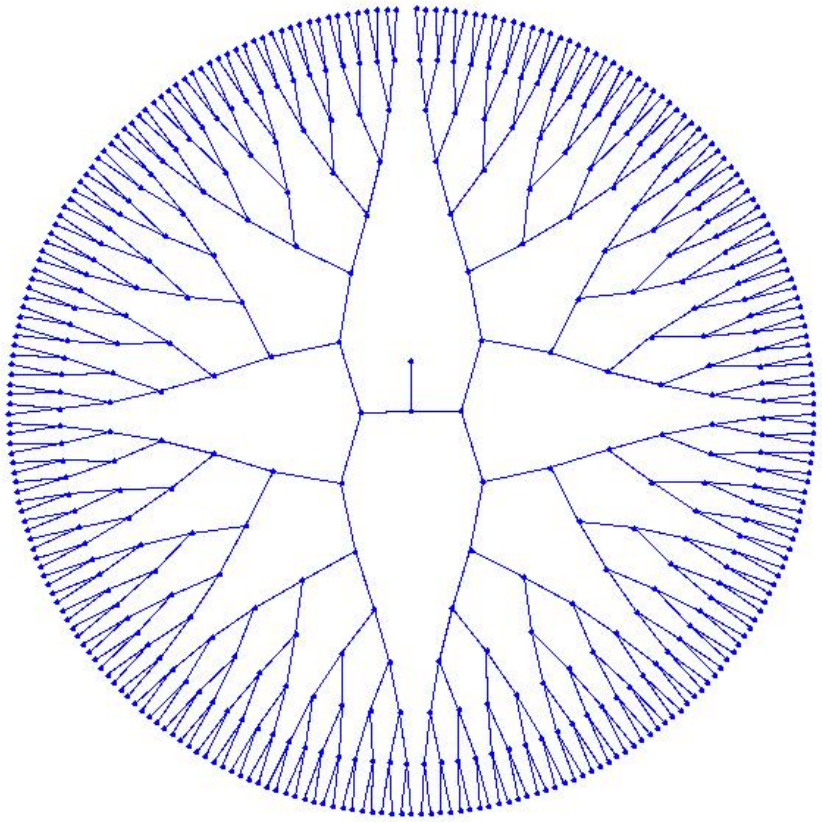
General Case



The whole diagram for $n = 10$, as rendered by an automatic graph layout program.

By Contrast . . .

The diagram for $n = 9$.



Exercise 6. *Explain the structure of the diagram of this map for lists of arbitray length.*

Example: Exclusive Or

Here is a slightly more ambitious example, though the analysis turns out still to be fairly easy in this case. Consider the map

$$f : \mathbf{2}^n \to \mathbf{2}^n$$

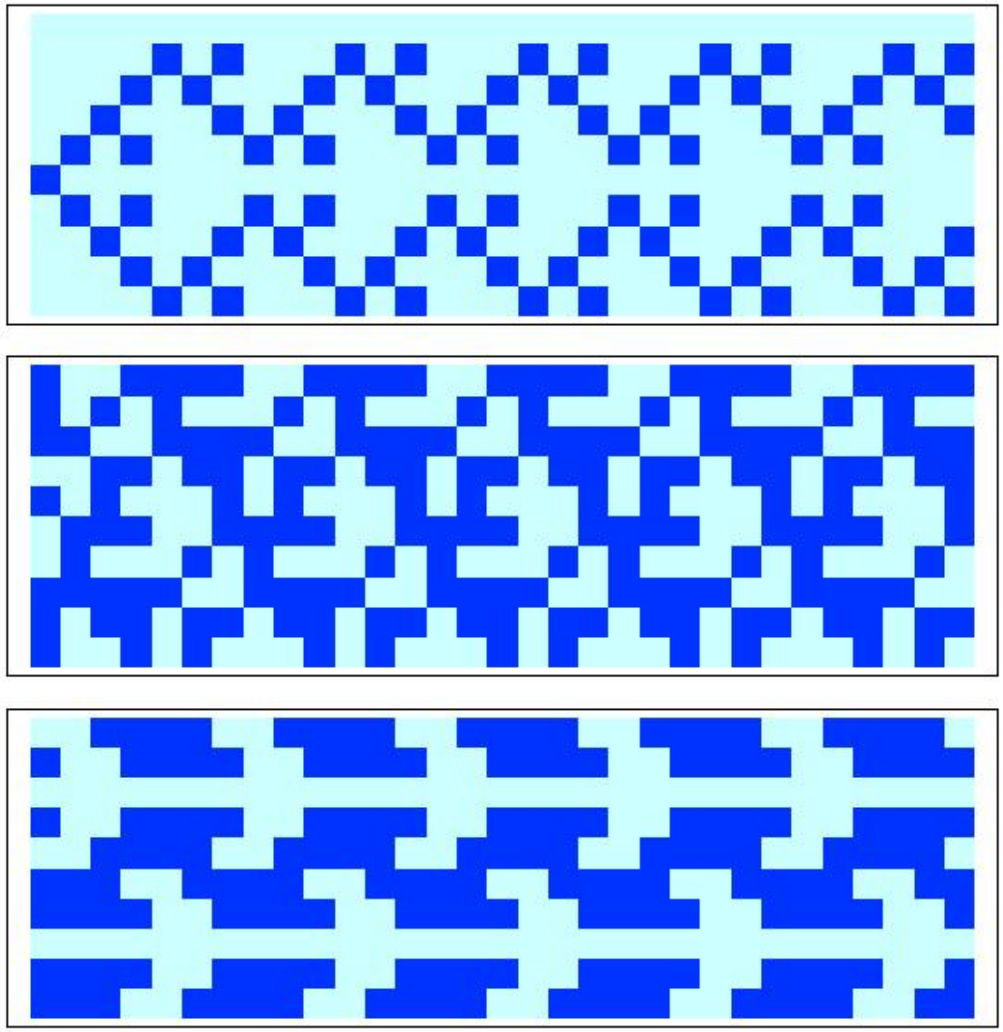
defined by

$$f(\boldsymbol{x}) = L(\boldsymbol{x}) \text{ xor } R(\boldsymbol{x})$$

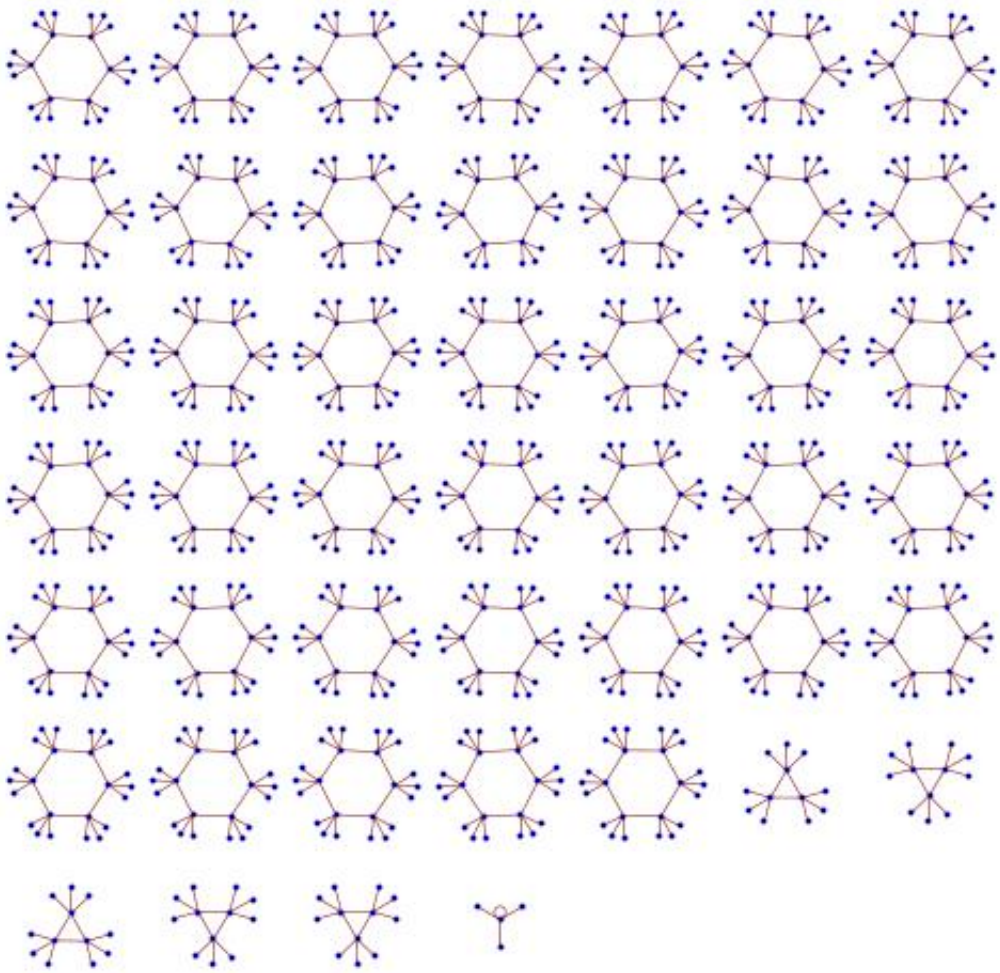
where L and R denote cyclic left- and right-shift, respectively, and xor is bit-wise exclusive or. This function is closely related to the Ducci operation on limit cycles.

E.g., for $n = 10$ we have $f(0, 0, 0, 1, 1, 1, 0, 0, 0, 0) = (0, 0, 1, 1, 0, 1, 1, 0, 0, 0)$

Some Orbits



The Whole Diagramm



The Parameters

The diagram is highly uniform in this case and can easily be described in terms of the general parameters.

- Every node has indegree 4.
- There are 40 limit cycles of length 6, 5 limit cycles of length 3 and one fixed point.
- The transient lengths for all points not on a limit cycle is 1.

Exercise 7. *Perform a similar analysis for other values of n. Observations?*

Fixed Points

Fixed Points

Fixed points are particularly interesting. Quite a few computational tasks can be rephrased as a fixed point computation. In the right environment this approach produces very elegant algorithms.

Definition 8. *Let $f : A \to A$ and $a \in A$. Write $\mathrm{FP}(f, a)$ for the fixed point on the orbit of a under f if it exists (undefined otherwise).*

So our claim is that there are lots of examples where an algorithm boils down to computing $\mathrm{FP}(f, a)$ for the right choice of f.

Note that most current programming language do not directly support the operation FP, though some like Mathematica do.

Binary Expansions

Here is a more utilitarian example. Define the following operation on numbers and binary lists.

$$f : \mathbb{N} \times \mathsf{List}(\mathbf{2}) \to \mathbb{N} \times \mathsf{List}(\mathbf{2})$$
$$f(0, L) = (0, L)$$
$$f(x, L) = (x \text{ div } 2, \mathsf{prep}(L, x \bmod 2))$$

Then $\mathrm{FP}(f, (x, \mathsf{nil})) = (0, L)$ where L is the binary expansion of x.

In the right environment, this provides a *one-liner* for conversion:

$$\mathsf{tobin}(x) = \mathsf{last}(\mathrm{FP}(f, (x, \mathsf{nil})))$$

Experimental, Interactive Computing

The one-liner may not seem particularly impressive.

In fact, when constructing large programs it may be preferable to use lots of simple operations, rather than somewhat cryptic and complicated "primitives" such as FP. Kenneth Iverson's APL from the 1960's is a perfect example of a language that tends to produce "write-only" code.

But for quick-and-dirty one-shot computations this is the way to go.

In experimental computing it is important to get results quickly so one try out various possibilities – there is no time to write, debug and compile a complicated program.

One has to rely on an expressive language (including for example list manipulation primitives) together with a large base of algorithms (such as integration, factorization, . . .).

Transitive Closure

Convergence problems disappear when we deal with discrete sets and functions.

Suppose $\mathrm{squ}(\rho) = \rho^2 = \rho \bullet \rho$ returns the composition of a given relation ρ on a finite carrier set.

If ρ is a reflexive relation, then $\mathrm{FP}(\mathrm{squ}, \rho)$ is the transitive reflexive closure of ρ.

Suppose $I \subseteq \rho \subseteq [n] \times [n]$.

We know that $\rho^* = \rho^t$ where $t \leq n - 1$.

Hence we have to iterate squ at most $\log_2 n$ times.

On can implement this idea using boolean matrices.

Application: Equivalence Relations

We have seen that an equivalence relation E on A can be represented by a selector function $f : A \longrightarrow A$ such that $E = K_f$, i.e.

$$x \; E \; y \iff f(x) = f(y)$$

Computationally this is advantageous since we can use a simple array to represent f:

This means we can test $x \; E \; y$ in $O(1)$ steps, and the data structure has size $O(n)$.

Example 1. *This representation is used in the classical forward algorithm minimization of finite state machines.*

Fixed Points and Union/Find

We can think of the Union/Find algorithm as a fixed point construction.

We use a function $f : A \longrightarrow A$ such that

$$x \; E \; y \iff \mathrm{FP}(f, x) = \mathrm{FP}(f, y).$$

Thus, an equivalence class consists of one basin of attraction of f.

A priori equivalence testing is just $O(n)$: the transients might be very long.

The Algorithm

Initially, $f(x) = x$ for all x.

Primitive version:

For every new pair (a, b):

- Compute $a_0 = \mathrm{FP}(f, a)$ and $b_0 = \mathrm{FP}(f, b)$.
- If $a_0 = b_0$, do nothing.
- Otherwise, set $f(b_0) = a_0$.

Clearly maintains the fixed point property.

By adding ranked union and path compression we can keep the transients so small that the whole algorithm is essentially linear in the number of operations.

Exercise 8. *Read up on the details of the Union/Find algorithm.*

Decimal Fractions

Computations that end when a fixed point has been reached are particularly elegant, but sometimes one has to to stop after going around the limit cycle for the first time. Note that this makes it quite a bit more difficult to decide when to stop.

A classical example: conversion of a rational number $0 < x < 1$ into a decimal fraction.

$$x = \sum_{i \geq 1} d_i \cdot 10^{-i} \text{ where } 0 \leq d_i \leq 9$$

Conversion algorithm:

$$f : \mathsf{List}(\mathbb{N}) \times \mathbb{Q} \to \mathsf{List}(\mathbb{N}) \times \mathbb{Q}$$
$$f(L, x) = (\mathsf{app}(L, \mathsf{floor}(10x)), \mathsf{fract}(10x))$$

Stop when $\mathsf{last}(f^i(\mathsf{nil}, x))$ assumes the same value twice.

Example

For $x = \frac{1}{123}$ we get

$$\frac{1}{123}, \frac{10}{123}, \frac{100}{123}, \frac{16}{123}, \frac{37}{123}, \frac{1}{123}, \ldots$$

leading to digits

$$0.008130081300813\ldots$$

Check: $\frac{813}{99999} = \frac{1}{123}$.

A Long Cycle

But for $x = \frac{1}{1234}$ there is a transient of 1, and a period of 88.

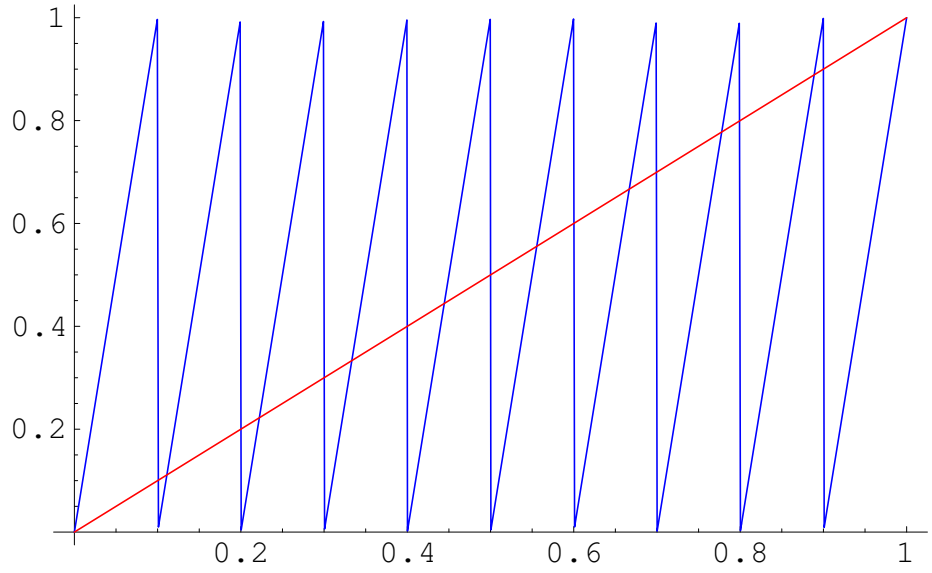
$$\frac{1}{1234}, \frac{5}{617}, \frac{50}{617}, \frac{500}{617}, \frac{64}{617}, \frac{23}{617}, \frac{230}{617}, \frac{449}{617}, \frac{171}{617}, \frac{476}{617},$$
$$\frac{441}{617}, \frac{91}{617}, \frac{293}{617}, \frac{462}{617}, \frac{301}{617}, \frac{542}{617}, \frac{484}{617}, \frac{521}{617}, \frac{274}{617}, \frac{272}{617},$$
$$\frac{252}{617}, \frac{52}{617}, \frac{520}{617}, \frac{264}{617}, \frac{172}{617}, \frac{486}{617}, \frac{541}{617}, \frac{474}{617}, \frac{421}{617}, \frac{508}{617},$$
$$\ldots$$
$$\frac{76}{617}, \frac{143}{617}, \frac{196}{617}, \frac{109}{617}, \frac{473}{617}, \frac{411}{617}, \frac{408}{617}, \frac{378}{617}, \frac{78}{617}, \frac{163}{617},$$
$$\frac{396}{617}, \frac{258}{617}, \frac{112}{617}, \frac{503}{617}, \frac{94}{617}, \frac{323}{617}, \frac{145}{617}, \frac{216}{617}, \frac{309}{617}, \frac{5}{617}$$

A Little Question

What, if any, are the fixed points of the function $f(x) = \text{fract}(10x)$?

Assume that $0 \leq x < 1$ is a rational. It is easy to see that $x = 0$ is a fixed point. Are there any others?

Try a picture first.



Looks like there are 9 fixed points. But where?

Calculus and Fixed Points

Calculus and Fixed Points

Perhaps the most interesting application of fixed points lies in the continuous domain, though. Finding a fixed point is an old method in calculus to compute certain numbers.

Suppose we need to calculate $\sqrt{2}$. Consider

$$f : \mathbb{R}^+ \to \mathbb{R}^+$$
$$f(x) = x/2 + 1/x$$

Of course, there is a problem: it is simply false to claim that

$$\mathrm{FP}(f, 1) = \sqrt{2}$$

All the numbers in the orbit are rational, but our goal is an irrational number, which therefor cannot be a fixed point. We should be dealing with limits and error bounds $|f(x) - x| < \varepsilon$. Just remember your calculus courses.

Limits

Still, we have

$$\lim_{n\to\infty} f^n(1) = \sqrt{2}.$$

and $\sqrt{2}$ is indeed a fixed point of f, the orbit just never reaches this particular point.

This is no problem in real life: we can stop when $|f(x) - x|$ is sufficiently small.

As a matter of fact, convergence is quite rapid:

0	1.0000000000000000
1	1.5000000000000000
2	1.4166666666666665186
3	1.4142156862745096646
4	1.4142135623746898698
5	1.4142135623730949234
6	1.4142135623730949234

Newton's Method

This is *Newton's Method*: to find a root of $f(x) = 0$, iterate

$$g(x) = x - \frac{f(x)}{f'(x)},$$

and pray that everything works.

Obviously f needs to be differentiable here, and we would like $f'(x)$ to be sufficiently far away from 0 so that the second term does not become unreasonably large.

Application: Inverse

A typical application of Newton's Method is to determine $1/a$ in high precision computations.

Here

$$f(x) = 1/x - a$$
$$g(x) = 2x - ax^2$$

The point here is that we can express division in terms of multiplication and subtraction (simpler operations in a sense).

Example

Numerical values for $a = 1.4142135623730950488 \approx \sqrt{2}$.

0	1.0000000000000000000
1	0.58578643762690495119
2	0.68629150101523960950
3	0.70649403654862594510
4	0.70710625021159255130
5	0.70710678118614880890
6	0.70710678118654752440
7	0.70710678118654752440

Verify the result:

$$0.70710678118654752440 \times 1.4142135623730950488 = 1.0000000000000000000$$

Brute Force

As we have seen, with luck, a fixed point for a continuous function $f : \mathbb{R} \to \mathbb{R}$ can be found by plain iteration, at least in the sense that we can produce a numerical approximation (perhaps even rapidly).

We compute $a_n = f^n(a)$ and exploit the fact that if there is a limit $b = \lim a_n$ then b is a fixed point of f.

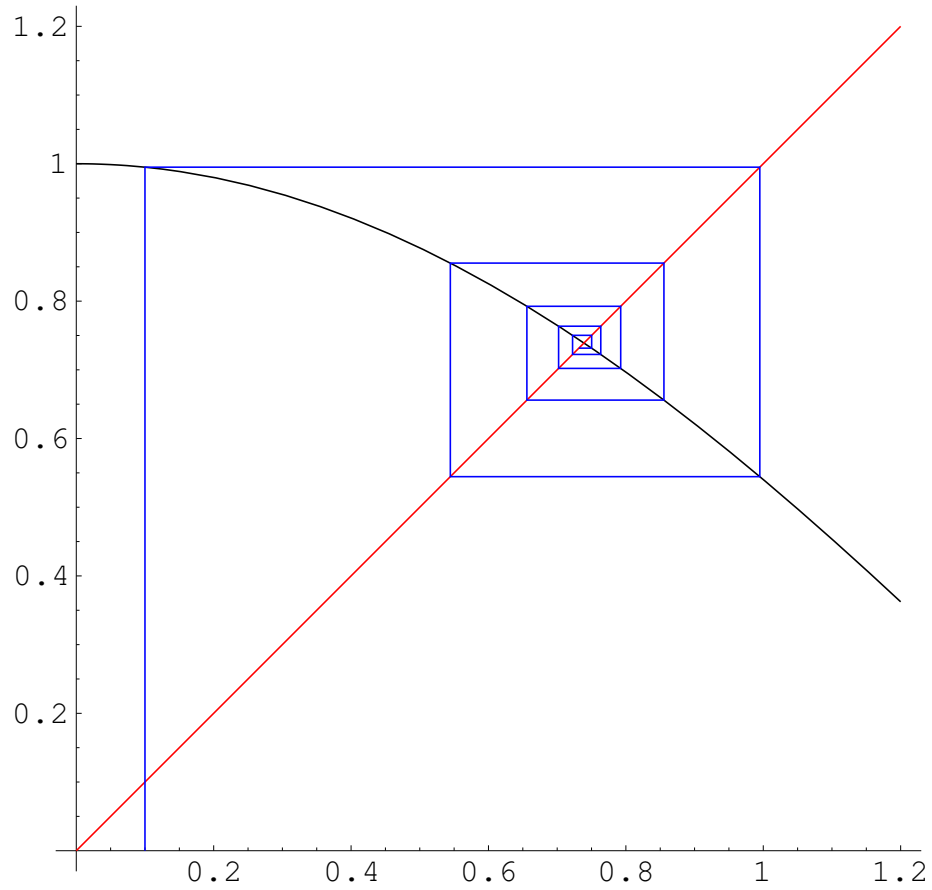
Moreover, a_n may be very close to b for reasonably small values of n so we can actually get our computational hands on a good approximation.

But the opposite direction is much more problematic.

Here are some examples.

A Nice Fixed Point

$\cos x = x$



The Logistic Map

Cosine is a transcendental function, but even with second order polynomials strange things happen under iteration.

$$f_p(x) = p \cdot x \cdot (1 - x)$$

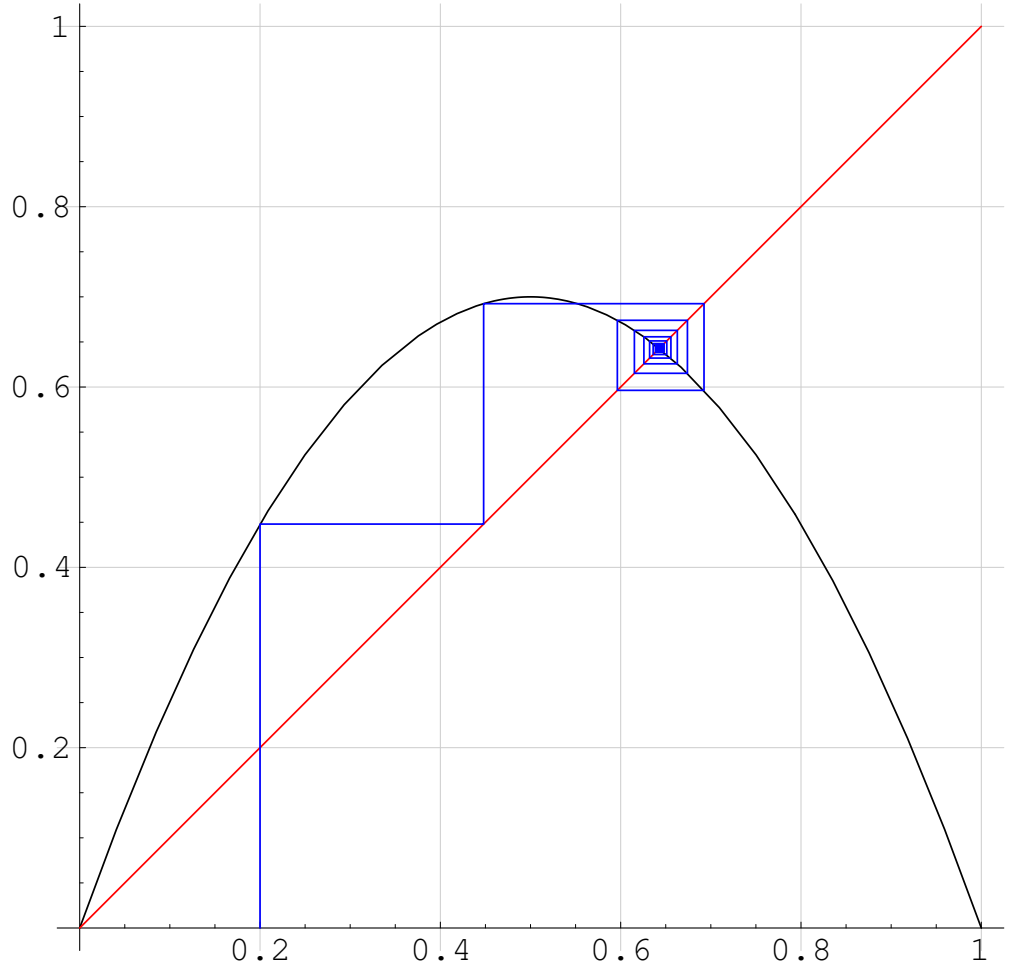
where $0 \leq p \leq 4$. Note that for these parameter values we have

$$f_p : [0, 1] \to [0, 1]$$

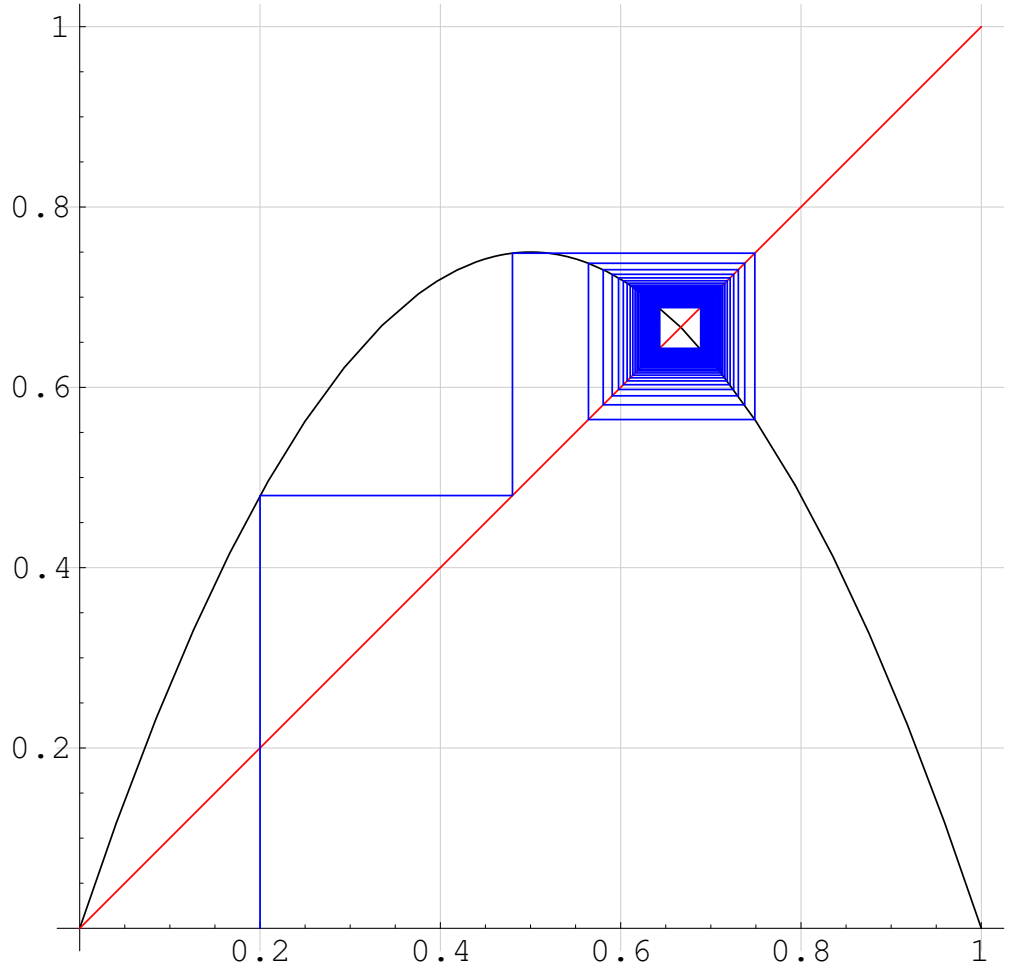
This is the so-called *logistic map*, a famous example in dynamics.

Its behavior depends drastically and very unexpectedly on the parameter p.

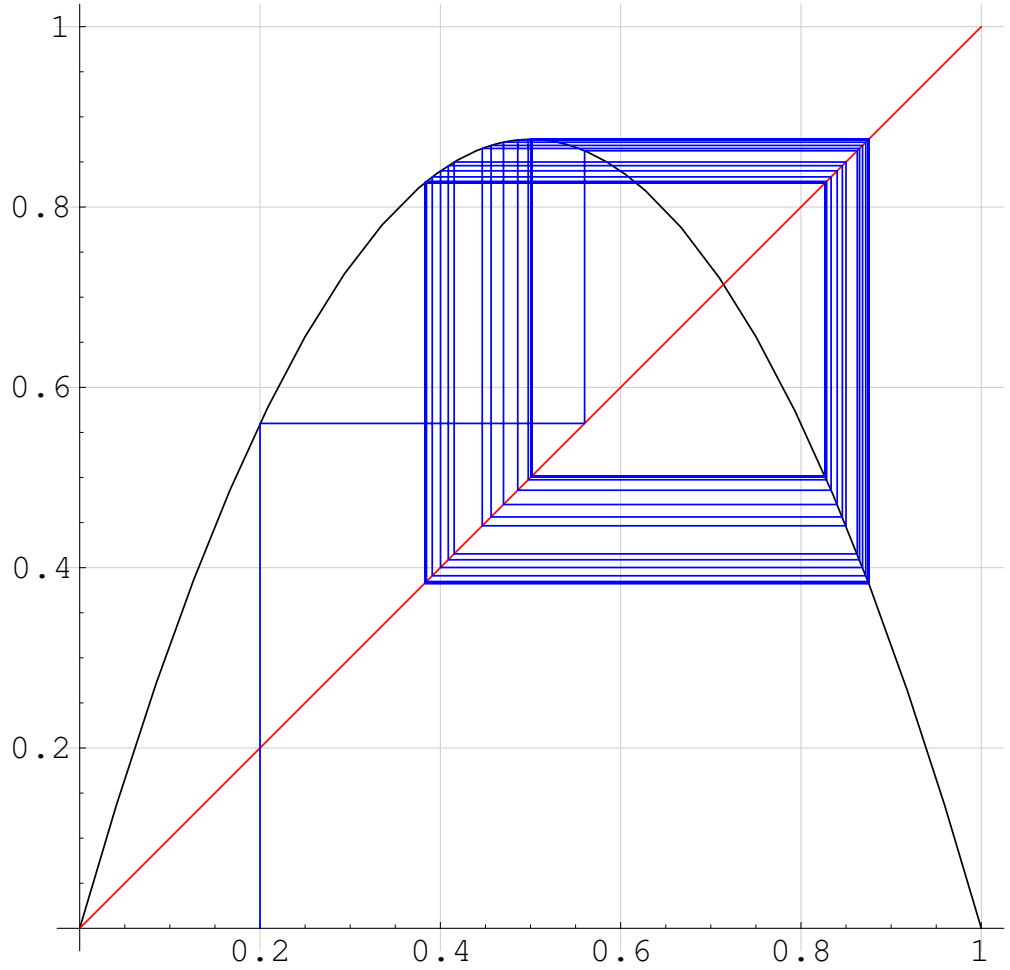
$$p = 2.8$$



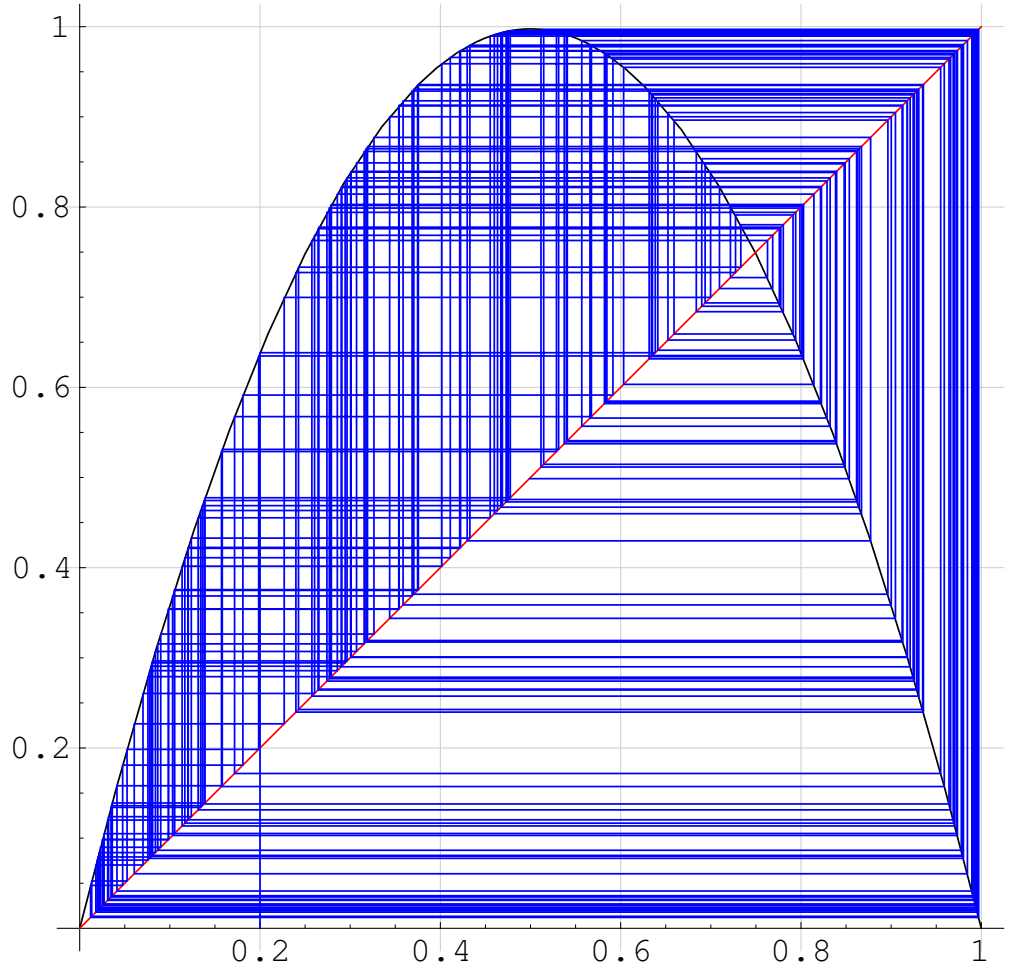
$$p = 3$$



$$p = 3.5$$

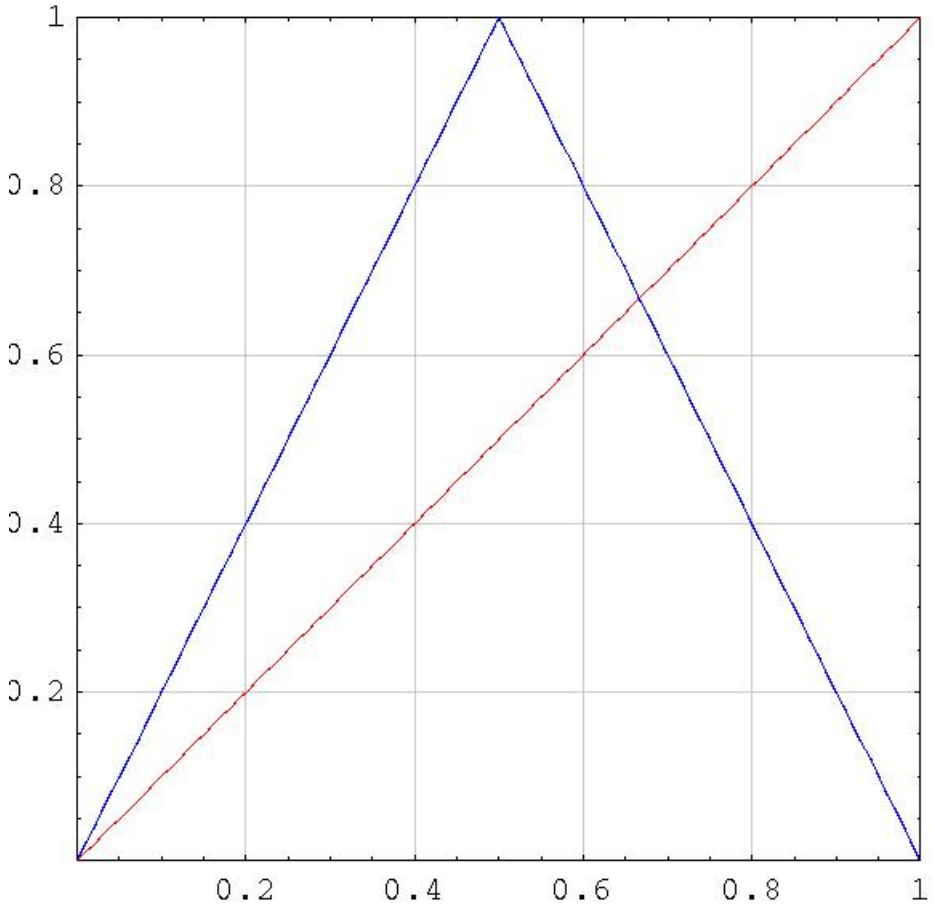


$$p = 3.99$$

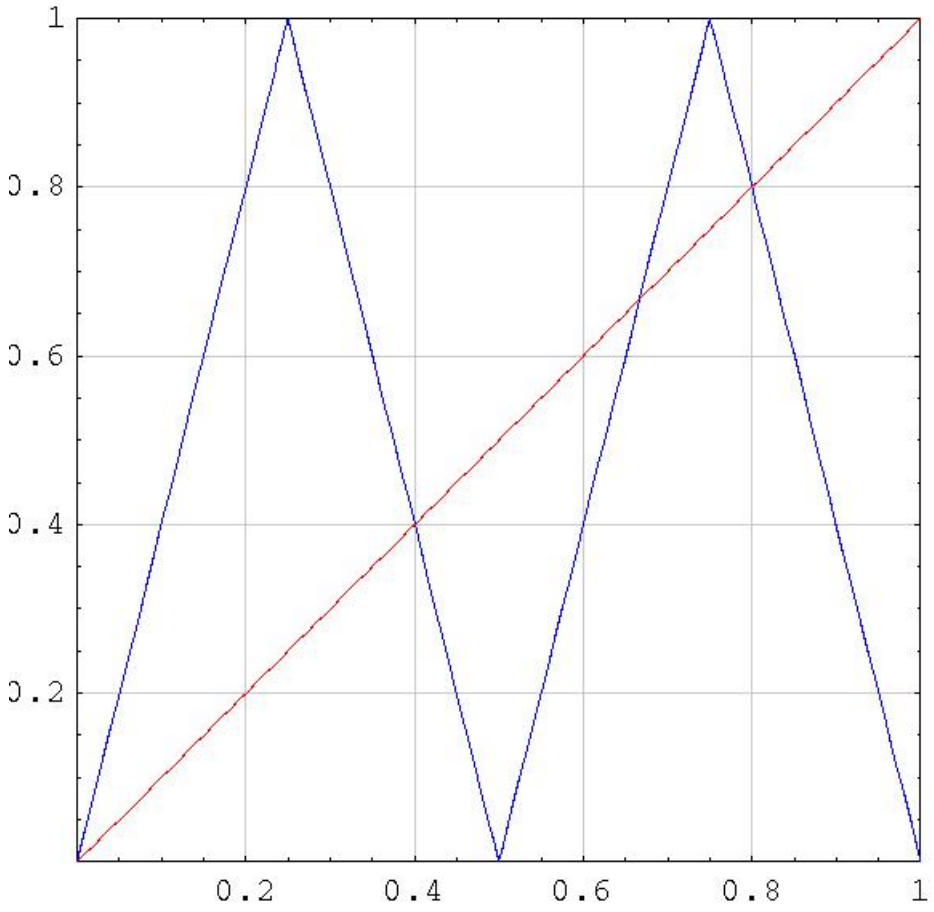


Tent Maps

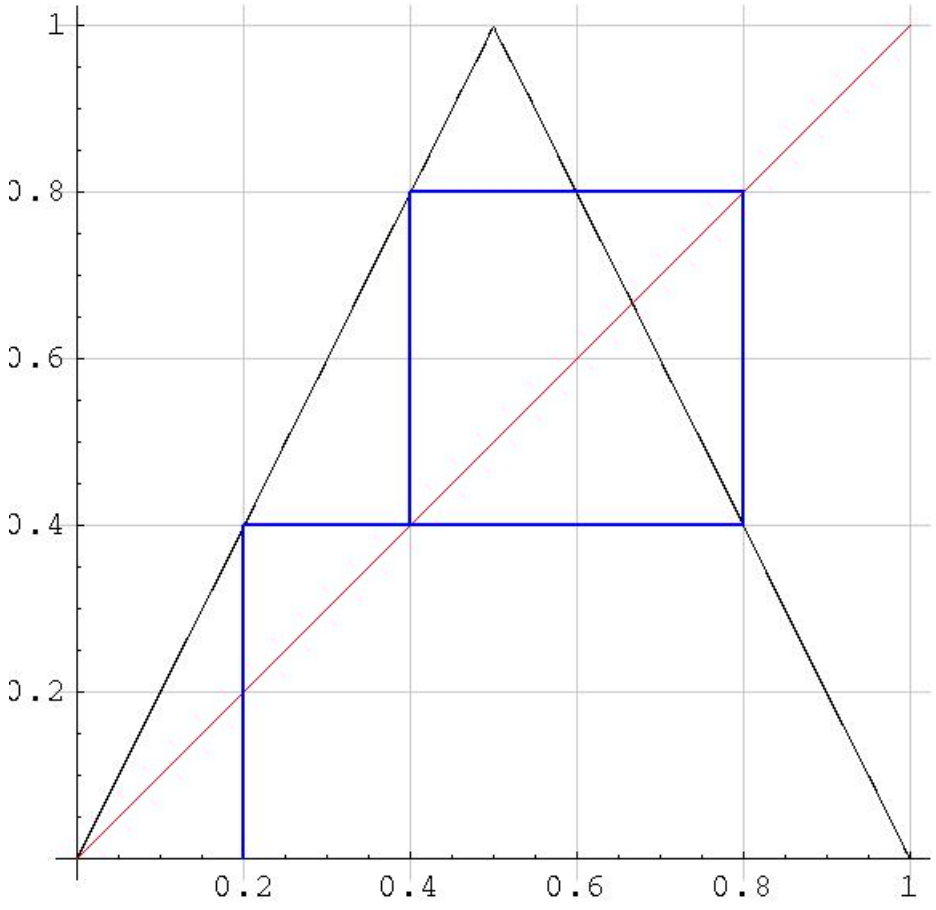
Differentiability is not necessary at all.



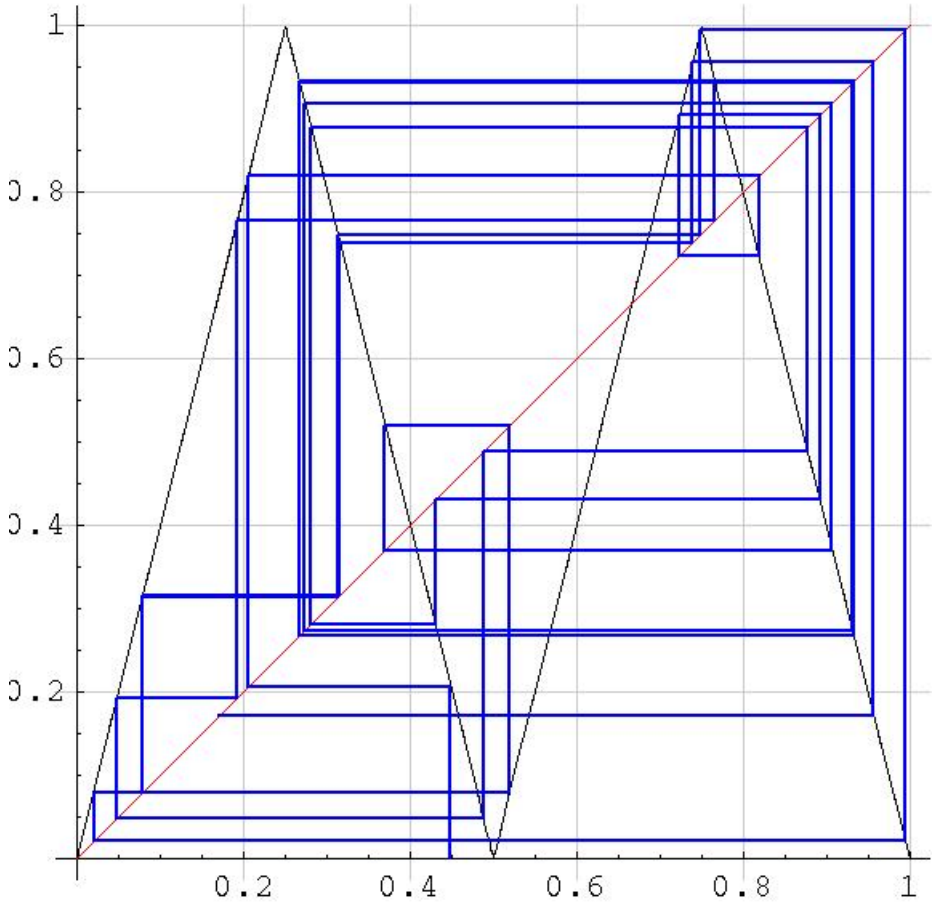
Tent Map Squared



A 2-Cycle



A Mess



Sarkovskii's Theorem

Here is a theorem describing chaos in real valued functions.

Consider the following weird ordering of the natural numbers, the so-called Sarkovskii ordering:

$$3, 5, 7, 9, \ldots, 2 \cdot 3, 2 \cdot 5, \ldots, 4 \cdot 3, 4 \cdot 5, \ldots, 2^3, 2^2, 2^1, 2^0.$$

Theorem 1. *For any continuous function $f : \mathbb{R} \to \mathbb{R}$: if f has a cycle of length α then f has a cycle of length β for all $\alpha < \beta$ in the Sarkovskii ordering.*

Hence, if there is a 3-cycle, then there are cycles of any length.

Finding Cycles

Calculating Transients and Periods

So how do we compute the transient t and period p of the orbit of $a \in A$ under $f : A \longrightarrow A$ for finite carrier sets A?

If f is a permutation all orbits are cycles and we can just walk around until you return to the starting point.

In general, we have to work harder. The brute force approach is to keep track of everything we have already seen:

$$x, f(x), f^2(x), \ldots, f^i(x)$$

and then to compare $f^{i+1}(x)$ to all these previous values.

In most cases, the data structure of choice is a hash table or tree: we can check whether $f^{i+1}(x)$ is already present in expected constant time or logarithmic time, respectively. Memory requirement is linear in the size of the orbit (we may have to pay for some extra pointers, though).

Floyd's Trick

Suppose you have a pointer-based structure in memory and you want to check if there are any cycles in the structure.

We can think of this as an orbit problem:

- A is the set of all nodes of the structure,
- $f(x) = y$ means there is a pointer from x to y.

We would like all orbits to end in at `nil`. So far, everything is straightforward.

The Problem:

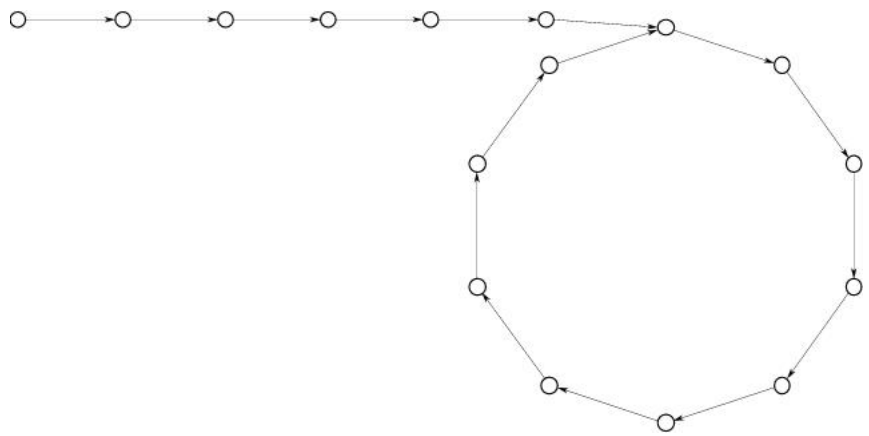
Suppose further the structure consumes 90% of memory, so we cannot afford to build a large hash table or tree.

So in general, can we compute transients and periods on $O(1)$ space?

A Memoryless Approach

At first glance, this may seem quite impossible: if we forget the elements we have already seen we cannot detect cycles. Nonetheless, the following code finds a point on the cycle in the orbit of a.

```
u = f(a);
v = f(u);
while( u != v )
{
  u = f(u);
  v = f(f(v));
}
```



Upon termination, $u = v$ is a position on the cycle.

Moving Pebbles

Think of two moving pebbles u (at speed 1) and v (at speed 2).

u enters the limit cycle at time t, the transient, when v is already there. From now on, v gains one place on u at each step. So pebble v must catch up at time s where $s \leq t + p$, where p is the period.

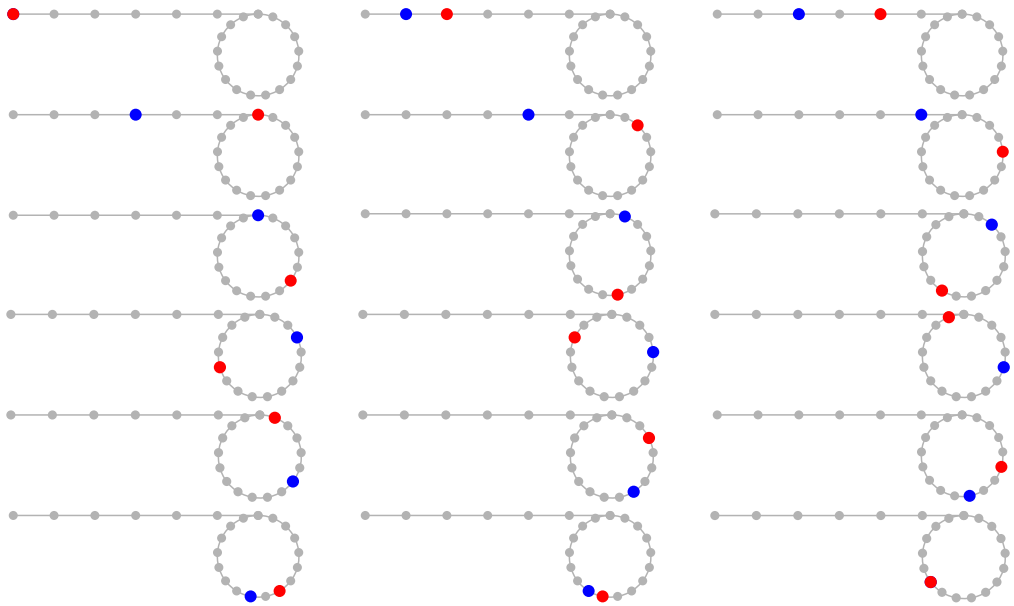
Also note that once we have our foothold on the cycle, we can compute the period: run around the cycle one more time, counting.

One can make a nice movie out of this.

OK, it is pretty boring after all, but what do you expect.

Example

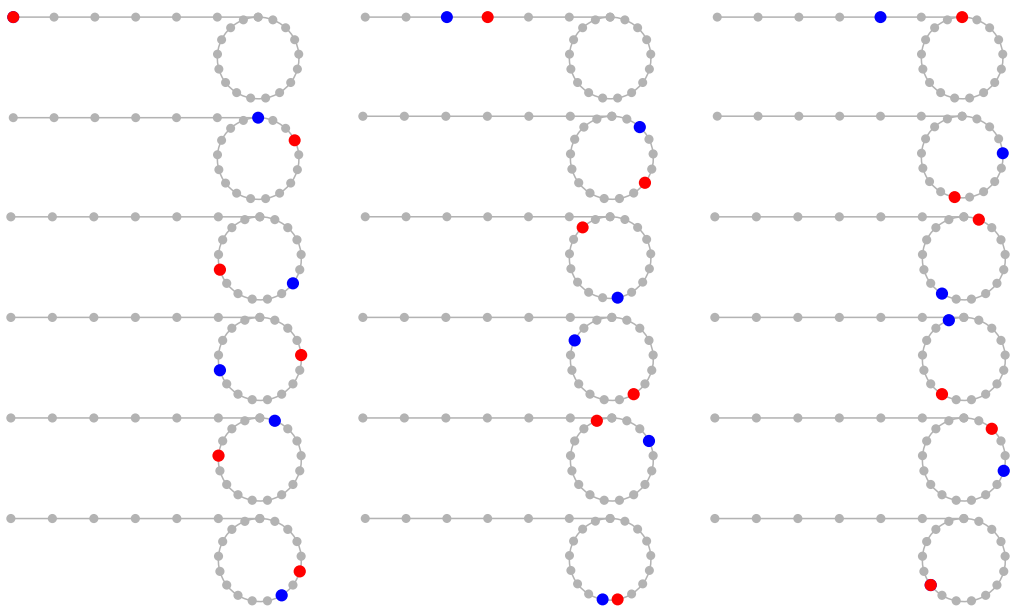
Here the transient is 6, and the period 17.



The pebbles meet at time 17.

Example

Same transient and period, but this time the pebble speeds are 2 and 3, respectively.



Again, the pebbles meet at time 17.

How about the Transient?

```
s = 0;
u = a;
v = iterate( f, a, p );                    // v = f^p(a)
while( u != v ) {
        u = f(u);
        v = f(v);
        s++;
}
```

Claim. *Upon completion,* $s = t$.

Proof.

v is exactly p places ahead of u. So, when u first enters the cycle, v has just gone around once, and they meet. □

Floyd's Trick

Let us assume f to be computable in time $O(1)$ and elements of the carrier set A to take space $O(1)$.

Theorem 2. *One can determine the transient t and period p of a point in A under f in time $O(t+p)$, and space $O(1)$.*

Linear time cannot be avoided in general (why?), so this is optimal.

Exercise 9. *The choice of speeds 1 and 2 for the pebbles in Floyd's algorithm is natural, but there are other possibilities. Discuss other choices.*

Beware of Permutations

Floyd's cycle finding algorithm is an excellent general purpose tool in particular when the evalution of the function in question is cheap.

But note that in the special case where the function is known to be a permutation on a finite domain there is, of course, no need to use Floyd's or similar cycle finding algorithms: since the components of the diagram are all cycles we can simply trace a cycle once to determine its length.

The natural method to compute cycle length is automatically memoryless (if we assume the objects in question can be stored in constant space).

Note that it may still be of interest to determine cycle lengths.

Example: Riffle Shuffle

Start with an even number of cards, split the deck in half, and then interleave the two halfs (perfect shuffle, alternate one card from each half-deck).

E.g., starting with the deck $[20]$ one obtains

$$11, 1, 12, 2, 13, 3, 14, 4, 15, 5, 16, 6, 17, 7, 18, 8, 19, 9, 20, 10$$

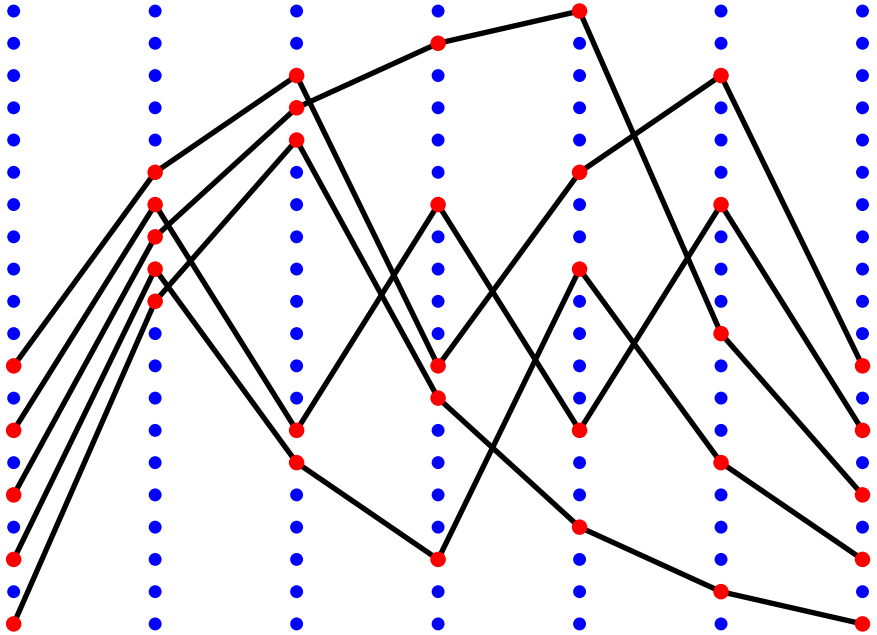
This clearly is a permutation (no cards disappear or are added). Hence, all transients are 0, and we get the cycle decomposition

$$(1, 11, 16, 8, 4, 2), (3, 12, 6), (5, 13, 17, 19, 20, 10), (7, 14), (9, 15, 18)$$

After six riffle shuffles we are back to the original deck of cards: the least common multiple of 6, 3, 6, 2, and 3 is 6.

The Orbits

This is also clear to see when we trace the orbit of the first point in each cycle.

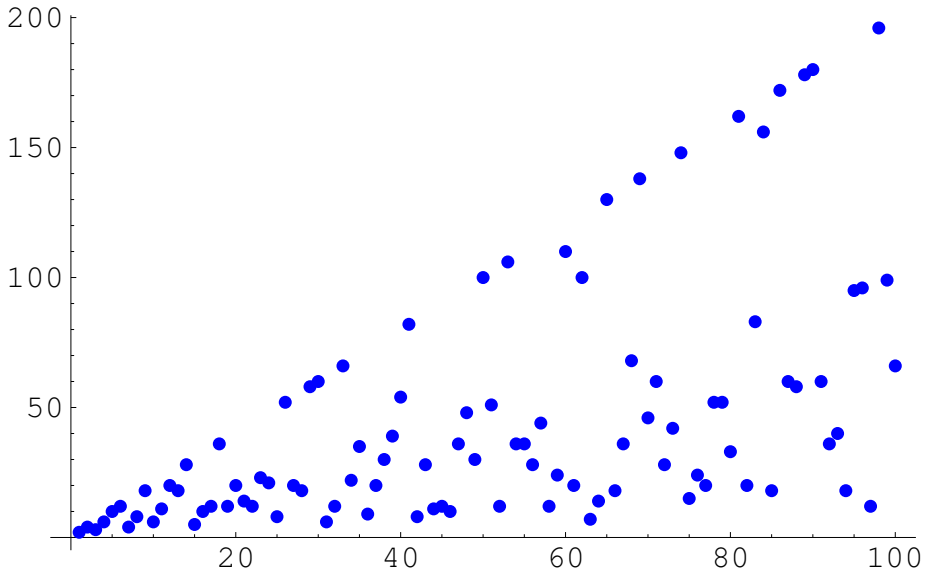


Note that the inverse map is a bit easier to understand: it boils down to multiplication by 2, modulo 21.

Various Deck Sizes

For any n, there must be some number k such that k repetitions of riffle shuffle on $2n$ cards return the deck to its original state.

How does k depend on n?



The relationship is a bit complicated, and we will not pursue the issue here.

A Simple Case

Consider the function

$$f_k : \mathbb{Z}_n \to \mathbb{Z}_n$$
$$z \mapsto z + k \bmod n$$

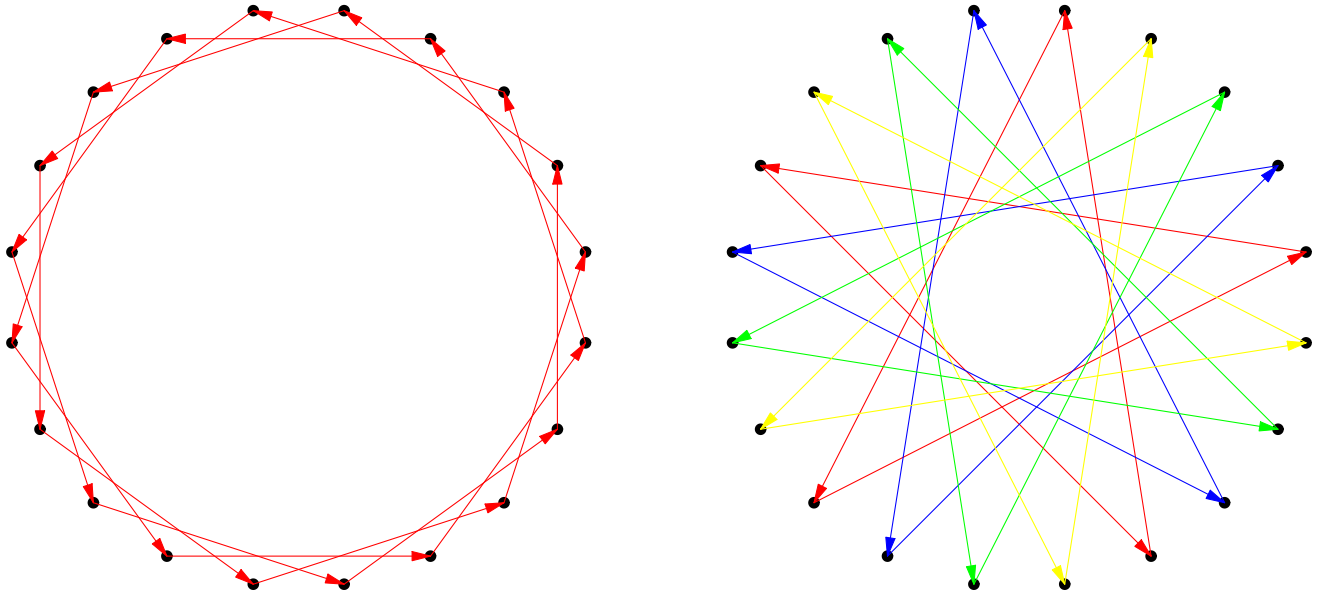
f_k is clearly injective, so the orbits are all cycles.

Moreover, since $f_k(x + d) = f_k(x) + d \pmod{n}$ all the cycles must have the same length.

So how many orbits are there?

Proposition 1. *f_k has $\gcd(n, k)$ distinct orbits, each of length $n/\gcd(n, k)$.*

$$n = 20$$



The stride is 3 on the left, 8 on the right.

The functional digraph consists of $\gcd(n, k)$ many disjoint cycles of length $n/\gcd(n, k)$ each.

How About Multiplication?

Can we come up with a similar analysis for

$$g_k(x) = x \cdot k \bmod n.$$

One should expect greater difficulties here:

g_k is not injective in general, so the orbits will have transients.

Moreover, the orbits cannot simply be translated into each other, not even the periodic parts.

A complete description of the digraph of g_k will be much more complicated than in the additive case.

Déjà Vu All Over Again

Note that we have already encountered this type of problem.

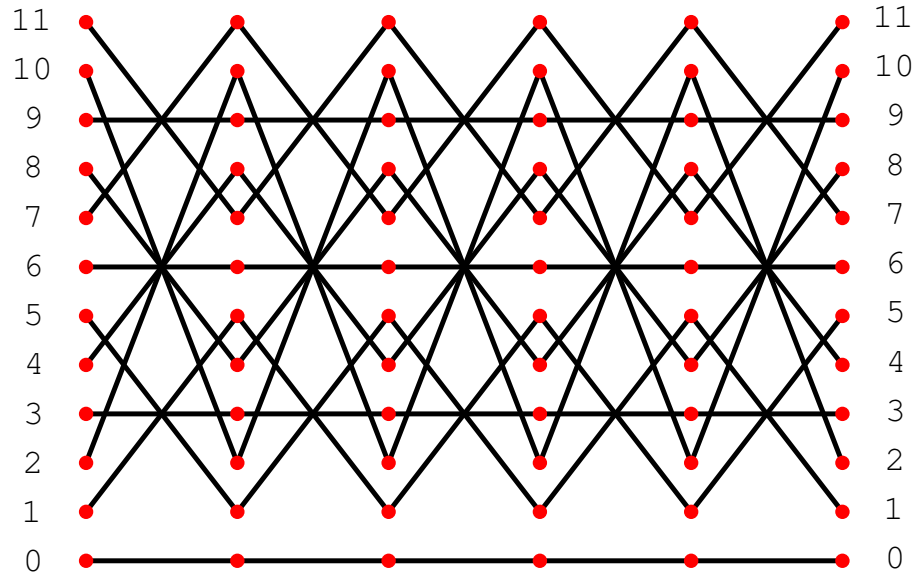
The brute-force construction of a DFA that recognizes strings in reverse binary notation denoting numbers that are multiples of 5 uses state set

$$\{0, 1, 2, 3, 4\} \times \{1, 2, 3, 4\}$$

where the second part is the orbit (rather: the underlying set) of 1 under g_2 with modulus $n = 5$.

$$1, 2, 4, 3, 1, 2, 4, 3, 1, 2, 4, 3, \ldots$$

$$n = 12,\ k = 5$$

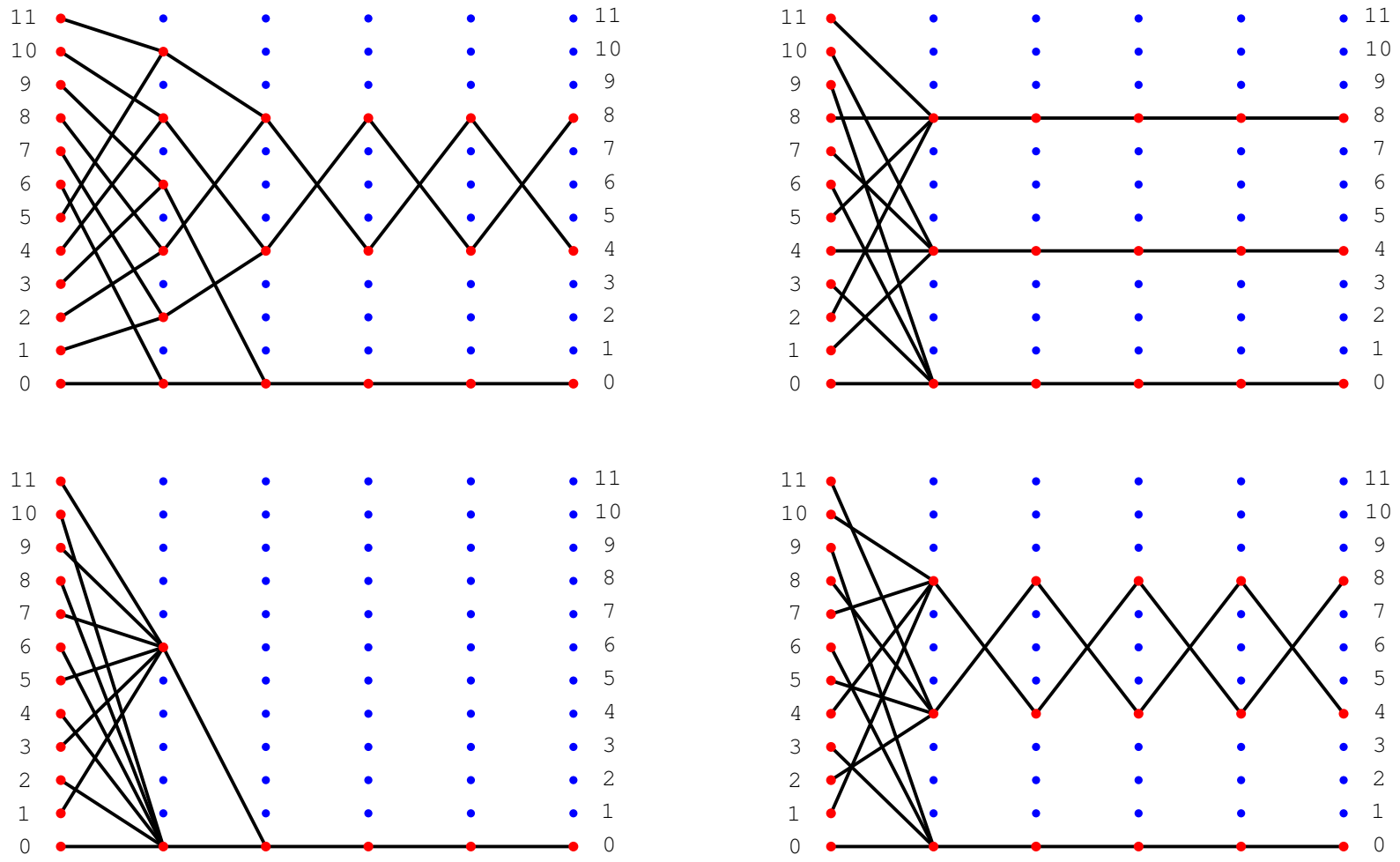


The picture shows all orbits simultaneously.

For example, the orbit of 6 is simply $6, 6, 6, \ldots$ (6 is a fixed point, as are 0, 3, 9)

The orbit of 1 is $1, 5, 1, 5, 1, \ldots$.

$$n = 12,\ k = 2, 4, 6, 8$$


11 10 9 8 7 6 5 4 3 2 1 0
11 10 9 8 7 6 5 4 3 2 1 0
11 10 9 8 7 6 5 4 3 2 1 0
11 10 9 8 7 6 5 4 3 2 1 0
11 10 9 8 7 6 5 4 3 2 1 0
11 10 9 8 7 6 5 4 3 2 1 0
11 10 9 8 7 6 5 4 3 2 1 0
11 10 9 8 7 6 5 4 3 2 1 0

The Easy Case

Suppose n and k are coprime. In this case g_k is injective.

So all orbits are periodic, there are no transients.

In other words, for some $p > 0$:

$$x = k^p \cdot x \pmod{n}$$

Note that $k^p = 1 \pmod{n}$ suffices, so the multiplicative order of k in $Z_n^\star$ is an upper bound for the period p.

But this bound is not tight, the choice of x also plays a role.

Fast Iteration

Fast Iteration

Coming back to the observation that iteration behaves just like ordinary exponentiation: since exponentiation can be computed quickly by repeated squaring, it is tempting to try the same for iteration.

Suppose we want to compute f^{1000}. The obvious way requires 999 compositions of f with itself.

But remember fast exponentiation, the standard trick for fast exponentiation when the exponents are large. Using this divide-and-conquer approach we have

$$f^{2n} = (f^n)^2$$
$$f^{2n+1} = f \circ (f^n)^2$$

This seems to suggest that we can compute $f^{1000}(x)$ in $O(\log n)$ applications of f.

After all, it's just like exponentiation, right?

Linear Maps

If the function f in question is linear it can be written as

$$f(x) = A \cdot x$$

where A is a square matrix over some suitable algebraic structure. Then

$$f^t(x) = A^t \cdot x$$

and A^t can be computed in $O(\log t)$ matrix multiplications.

So this is an exponential speed-up over the standard method.

Polynomial Maps

Another important case is when f is a polynomial map

$$f(x) = \sum a_i x^i$$

given by a coefficient vector $\boldsymbol{a} = (a_d, \ldots, a_1, a_0)$.

In this case the coefficient vector for $f \circ f$ can be computed explicitly by substitution. This is useful in particular when computation takes place in a quotient ring such as $R[x]/(x^n - 1)$ so that the expressions cannot blow up.

We will encounter this again later when discussing cellular automata.

But Beware of Hasty Conclusions

But we cannot conclude that $f^t(x)$ can always be computed in $O(\log t)$ operations.

The reason fast exponentiation and the examples above work is that we can explicitly compute the powers of the number and the powers of the function we are dealing with.

But in general, there are no shortcuts to to evaluating $f \circ f$: we have to evaluate f twice.

Just think of f as being given by a piece of C code. We can produce another piece of C code that computes f^2, and more generally for f^t, but the code just evaluates f t-times, in the obvious brute-force way.

This is very different from performing a squaring operation on, say, an integer: we obtain an integer, given an integer as input.

Weird Mod Sequence

Speaking about hasty conclusions, here is a simple inductively defined sequence of integers.

$$a_1 = 1$$
$$a_n = a_{n-1} + (a_{n-1} \bmod 2n)$$

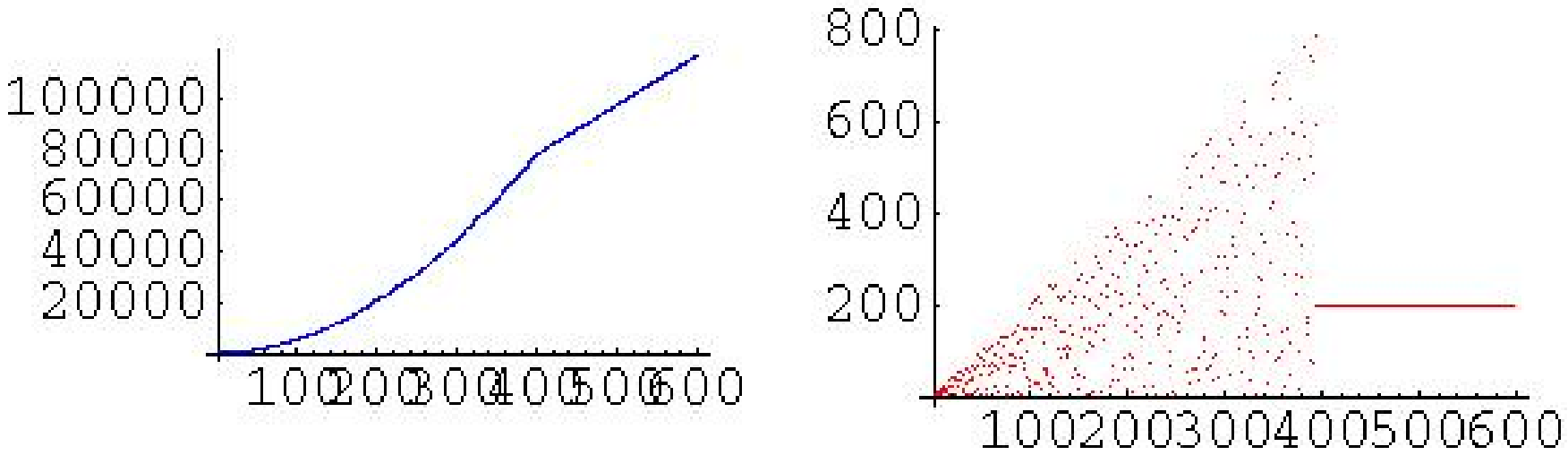
Thus, the sequence starts like so:

$$1, 2, 4, 8, 16, 20, 26, 36, 36, 52, 60, 72, 92, 100, 110, 124, 146, 148, 182, 204$$

This seems rather complicated, but a plot of the values up to 600 reveals a suprising fact.

Ultimately Constant

The sequence is ultimately linear: $a_{396+k} = a_{396} + k \cdot 194$ for $k \geq 0$.



The plot on the left is the sequence, on the right (in red) are the forward differences.

Exercise 10. *Figure out why the sequence is ultimately linear.*

Decidability and Collatz

Back To Collatz

The Collatz Conjecture simply says that all orbits of the Collatz function C end in the fixed point 1.

Collatz Conjecture: All orbits of C end in 1.

Unfortunately, though this problem is not included in the Clay challenge, some believe it to be enormously difficult.

> Mathematics is not ready for this kind of problem.
>
> Paul Erdös

This is not an instance of sour grapes, Erdös was one of the shining lights of 20th century discrete mathematics (Erdös number).

Can We Use Computability?

Here is a desperate idea: Perhaps we could use our knowledge of computation to shed some light on this problem? After all, the Collatz function is easily computable, even primitive recursive.

Indeed, all we need is a single while-loop wrapped around a LOOP program of depth 1 to get the orbits.

We already have two natural decision problems: Reachability and Confluence.

Unfortunately, iteration has a tendency to turn even very simple functions into something very complicated.

Collatz Decision Problem

So how do we turn Collatz into a decision problem?

Here is one fairly natural approach, really a version of Reachability.

Problem: **Collatz I**
Instance: A natural number $x \geq 1$.
Question: Does the orbit of x end in 1?

Observations:

- This problem is clearly semi-decidable.
- If the Collatz Conjecture is true, this problem is trivially decidable.
- Unfortunately, if this problem is decidable, the Collatz Conjecture may still be wrong (though the counterexamples are not terribly complicated: the set of all counterexamples is decidable).

Not too promising.

Another Version

Problem: **Collatz II**
Instance: A positive natural number x.
Solution: The stopping time of x if it exists, ∞ otherwise.

The stopping time is trivially computable, for the same reason that Collatz I is semi-decidable: we can enumerate the orbits.

But it might be a partial function: for some inputs x the computation (of the orbit of x under C) may not terminate.

Adding the condition that the algorithm must output ∞ when the orbit does not end in 1 may make the last problem unsolvable: there may be no recursive function that produces the appropriate outputs.

Fred Hacker's Brilliant Idea

One might think that the right question to ask is this:

Problem: **Fred Hacker's Collatz Problem**
Instance: The Collatz function (or, if you prefer: a banana).
Question: Is the Collatz Conjecture true?

Fred's Collatz problem is trivially decidable, albeit for entirely the wrong reasons.

The issue here is that there is only one instance.

Say What?

For decidability, all we need is an algorithm that solves Fred's Collatz Problem.

No problem, here are two candidates:

- Algorithm 1: Ignore the input and output Yes.
- Algorithm 2: Ignore the input and output No.

One of those two algorithms solves this decision problem, we just don't know which.

Note that no one said that we need to be able to write down the algorithm explicitly, we just have to make sure an algorithm exists. But one of the two candidates is guaranteed to work.

Theology

This type of argument caused a huge uproar in the mathematics community when first used by D. Hilbert in 1890 (finite basis theorem).

> "This is not mathematics, this is theology."
>
> Paul Gordon

Gordon was upset since he had found a constructive, computational proof for special case $n = 2$ in 1886, but had failed in all attempts to generalize the argument to arbitrary n.

Hilbert handled the general case not by explicitly computing a solution, but by showing that the assumption that there is no solution leads to a contradiction.

Finite Problems

The same trick works for any decision problem with only finitely many instances $x_1, x_2, \ldots, x_n$:

This time there are 2^n algorithms that are potential solutions:

$$A_0, A_1, \ldots, A_{2^n-2}, A_{2^n-1}$$

We just may not know which of these algorithms is the right one.

But: The algorithm does exist, so the problem is decidable.

In other words, classical computability theory is ill-equipped to deal with finite decision problems: the definition just does not bite.

So all this Computability Stuff is Useless?

In a sense, yes.

Problems with a single instance like the Collatz Conjecture, Riemann Hypothesis, Poincaré Conjecture, and so, don't naturally fit into this framework.

Also, in practical computations one only deals with instances of limited size and there are only finitely many of those. For bounded size input an algorithm always exists.

But, we have no clue which one it is. Moreover, undecidability and unsolvability cast a noticeable shadow in the realm of "small instances". For example, Diophantine equations are difficult even when restricted to apparently simple special cases. Testing a C program for termination is hard work.

And Collatz?

For the Collatz problem John Horton Conway, of Game-of-Life fame, found a beautiful way to show how undecidability lurks nearby.

Conway's idea: How about constructing infinitely many Collatz Conjectures?

More precisely, come up with a family of functions that generalize the Collatz function slightly.

Then ask if for one of these functions all orbits contain 1.

Conway's Theorem

Define a family of Collatz-like functions

$$\mathrm{Cn}(\boldsymbol{a}, \boldsymbol{b})(n) = a_i \cdot m + b_i$$

where $k = |\boldsymbol{a}| = |\boldsymbol{b}|$, $i = n \bmod k$, $m = n \operatorname{div} k$.

Classical Collatz is essentially the special case

$k = 2$, $a_0 = 1$, $a_1 = 6$, $b_0 = 0$, $b_1 = 4$,

so nothing is lost by considering Conway's functions.

But now we have infinitely many functions to deal with (though one of them is perhaps more interesting than all the others).

Conway-Collatz Problem

Problem: **Conway-Collatz Problem**
Instance: The parameters $\boldsymbol{a}$, $\boldsymbol{b}$.
Question: Does every orbit of $\mathrm{Cn}(\boldsymbol{a}, \boldsymbol{b})$ contain 1?

Theorem 3. *The Conway-Collatz Problem is undecidable.*

It remains undecidable even if all the b_i's are 0.

The theorem indicates that there is no good general way to answer questions about Collatz-like functions. So it is not surprising that the classical Collatz function is also very difficult to analyze.

Conway's T Function

A famous example of a Conway function other than the classical C is the following:

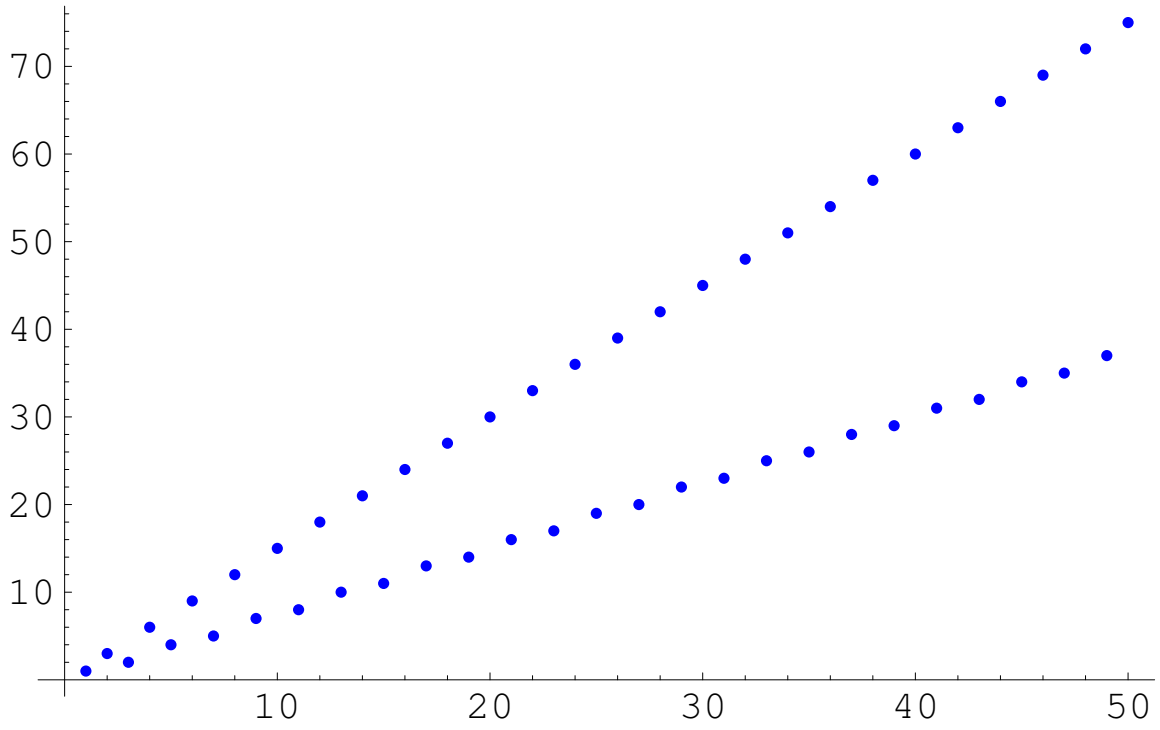
$$T(2n) = 3n$$
$$T(4n+1) = 3n+1$$
$$T(4n-1) = 3n-1$$

This is just Cn$(6, 3, 6, 3; 0, 3, 1, 2)$.

Proposition 2. *$T : \mathbb{N} \to \mathbb{N}$ is a bijection.*

Exercise 11. *Prove that the T function is a bijection. Then look for cycles under T.*

Plot



Looks very similar to the Collatz function.

But note that the lower line wobbles; there are really 3 linear functions here.

Conway's T Function

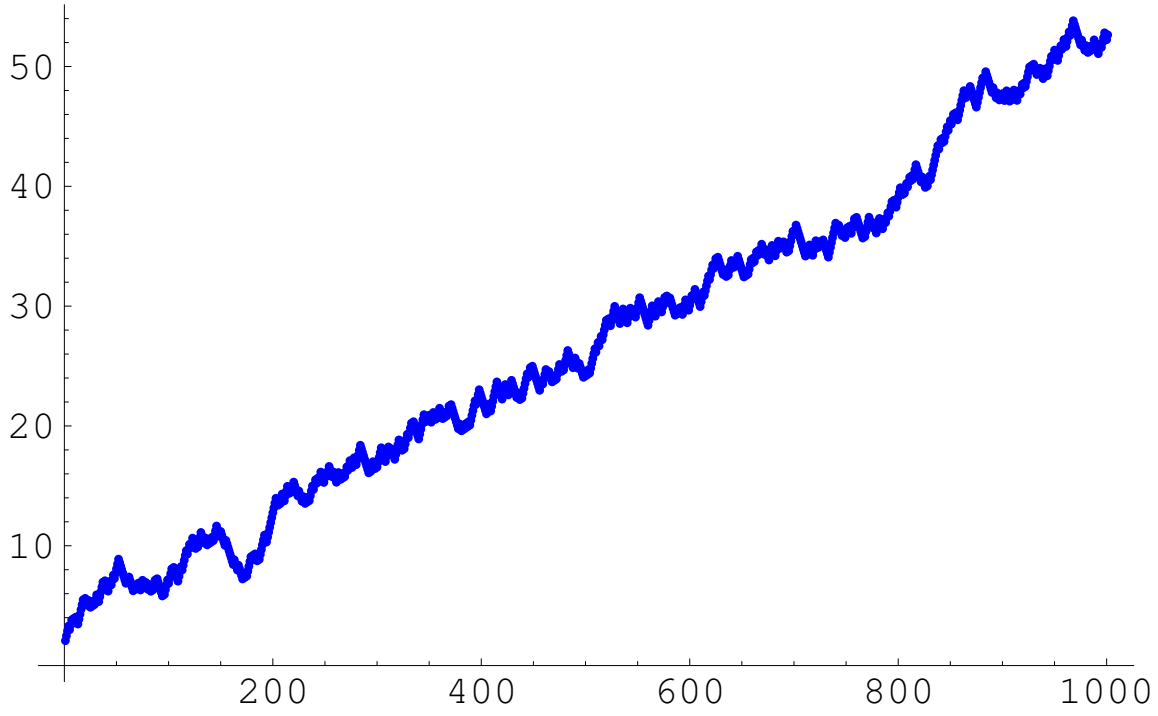
Known finite cycles are:

$$(1),$$
$$(2, 3),$$
$$(4, 6, 9, 7, 5),$$
$$(44, 66, 99, 74, 111, 83, 62, 93, 70, 105, 79, 59).$$

Open Problems:

Are there any other finite orbits?
In particular, is the orbit of 8 finite?

Orbit of 8



This is a log-plot. It seems to suggest the orbit of 8 grows without bound, but of course this is neither here nor there: the first 1000 values are really meaningless.

Recall Matiyasevic.

Reachability is Undecidable

If we iterate a function on the integers, even it is is simple, say, p.r., Reachability will be undecidable in general.

Theorem 4. *For a p.r. function $f : \mathbb{N} \longrightarrow \mathbb{N}$ reachability is undecidable in general. Likewise, confluence is undecidable in general.*

Similarly one cannot decide whether an orbit ends in a fixed point, in a cycle of length p and so on.

For finite carrier sets all these problems are trivially decidable, but they are typically hard from a computiational complexity point of view.

Theory of Fixed Points

Recall: Lattices

Definition 9. *A lattice is an algebraic structure $\mathcal{A} = \langle A, \sqcap, \sqcup \rangle$ whith two binary operations, referred to as meet and join that are both associative, commutative and idempotent. Moreover, the absorption law holds:*

$$x \sqcap (x \sqcup y) = x \sqcup (x \sqcap y) = x$$

Example 2. *Boolean values true and false with logical connectives "and" and "or" form a lattice.*

The powerset of any set with operations intersection and union form a lattice.

Binary relations on a set form a lattice whith intersection and union. Likewise, equivalence relations form a lattice, albeit with a different meet operation.

The positive integers with $\gcd$ *and* lcm *form a lattice.*

The Poset Interpretation

Another way to look at lattices is to consider a poset $\langle A, \leq \rangle$.

If the poset is properly behaved, we can define binary operations

$$\inf(x, y) = \max\big(z \in A \mid z \leq x, y \big)$$
$$\sup(x, y) = \min\big(z \in A \mid x, y \leq z \big)$$

The $\langle A, \inf, \sup \rangle$ is a lattice.

One the other hand, given a lattice we can define a partial order by

$$x \leq y \iff x \sqcap y = x.$$

This partial order has sups and infs which turn out to be exactly the join and meet operations of the lattice.

Complete Lattices

In a lattice, the join and meet operations are binary by definition. Of course, they can be generalized to a finite number of arguments, but there is requirement that sups and infs should exist for arbitrary subsets of A.

Definition 10. *A lattice is* complete *if every subset of the carrier set has a join and a meet.*

Note that every complete lattice must have a least and a largets element.

Example 3. *Every finite lattice is complete.*

The infinite examples from above are all complete.

The reals with standard order form an incomplete lattice.

The natural numbers with divisibility form a complete lattice (this is tricky).

Lattices and Fixed Points

In analysis, the existence of fixed points is a difficult topic. In our setting, there is a very powerful theorem that often can be used to demonstrate the existence of fixed points.

Theorem 5. *Knaster-Tarski*
Let L be a complete lattice and $f : L \longrightarrow L$ a monotonic map on L. Then the set of fixed points of f is a complete sublattice of L.

It follows from the theorem that fixed points exist; there may even be many of them.

Moreover, there is a least fixed point μf as well as a largest fixed point νf.

We will skip over the proof and quickly look at some applications.

Equivalential Closure

Suppose we have a binary relation ρ on A.

Define for any binary relation X on A:

$$f(X) = I_A \cup \rho \cup (X \circ X) \cup X^{-1}$$

The lattice of binary relations on A is clearly complete, and f is monotonic.

Then $\mu f = \bigcup_{n<\omega} f^n(\emptyset)$ is the equivalential closure of ρ. The closure ordinal is ω since the chains required by transitivity are all of finite length.

Banach's Lemma

Lemma 2. *Let $f : A \to B$ and $g : B \to A$ be two arbitrary functions. Then there are partitions $A = A_1 \cup A_2$ and $B = B_1 \cup B_2$ such that $f(A_1) = B_1$ and $g(B_2) = A_2$.*

Proof. We exploit the Knaster-Tarski theorem. Consider the powerset of A, clearly a complete lattice.

For any $X \subseteq A$ define

$$F(X) = A - g(B - f(X))$$

A moment's thought reveals that F is indeed monotonic and thus has a least fixed point.

Then $A_1 = \mu F$ works as required. □

Cantor-Bernstein

Note that the Cantor-Bernstein theorem is a simple corollary to Banach's lemma: when f and g are both injective we have $|A_1| = |B_1|$ and $|A_2| = |B_2|$.

This argument is considerably less complicted than the standard proof of Cantor-Bernstein.

Exercise 12. *Try to prove Banach's lemma from scratch, without reference to the Knaster-Tarski theorem. Try some special cases first, say, f maps to one point, is surjective, and so on.*

Exercise 13. *Look up some standard proofs for Cantor-Bernstein. How do they compare to the proof above.*

Summary

- Iteration produces complicated behavior even in simple functions.
- It matters little whether the domain is discrete or continuous.
- Many algorithms can be construed as fixed point constructions.
- Some computational environments such as Mathematica offer a fixed point operation as a primitive.
- There is a memoryless linear time algorithm to compute transient and period of a map on a finite carrier set.
- Questions about the orbits of functions may easily be undecidable, even if the functions in question are very simple.
- Undecidability does not directly apply to problems with finitely many instances.
- However, suitable modifications may yield undecidable problems, indicating that the original problem is difficult to deal with.