RANDOMNESS AND COMPLEXITY IN GEOMETRY

ROBERT YOUNG

1. Prologue

Probabilistic arguments are a powerful tool in many areas of math, but they often lead to a certain pattern: frequently, you can prove that a certain property is overwhelmingly common, but be unable to construct a single example with that property. (The classic example is Ramsey theory — the best bounds on large Ramsey numbers come from probabilistic arguments that imply, for instance, that 99.9% of 1,000,000-vertex graphs have no 34-clique or anticlique, but there are no examples of a graph known to have this property.)

In many situations (graphs, surfaces, manifolds), the objects that we can construct explicitly form a tiny minority of the objects that exist; the rest are “typical”, “generic”, or “random”. This course is about exploring the landscape of “random” objects and how randomness affects geometry.

2. Outline

• Preliminaries: computation, randomness, Kolmogorov complexity
• Random graphs
• Random surfaces
• Random manifolds
• The geometry of the moduli space of Riemannian metrics
• The geometry of embedded surfaces

Date: January 28, 2015.
3. What is random?

Suppose \( a = \{a_n\} \) is an infinite 0-1 sequence. Is \( a \) random?

Well, a random 0-1 sequence should pass some basic statistical tests. For instance, 0’s and 1’s should occur an equal number of times (equidistribution). Similarly, all the strings of length \( k \) should appear a roughly equal number of times as \( n \to \infty \) (normality). But that’s not enough: \( 0.01234567891011\ldots \) is a standard example of a number that’s normal but clearly not random.

Richard von Mises proposed an interesting definition: a sequence is a collective if there is no strategy to bet on that sequence with positive return. (Note that this definition was formulated before martingales were formulated, so “strategy” is a little different than the usual martingale version.) Let \( \{0,1\}^* \) be the set of finite 0-1 strings. Suppose that \( \phi : \{0,1\}^* \to \{0,1\} \). Let \( k_i \) be the sequence

\[
\{k_1, k_2, \ldots \} = \{k \mid \phi(a_1 \ldots a_{k-1}) = 1\}.
\]

(We call \( \phi \) a place-selection rule, and think of it as selecting the sequence of places \( k_1, k_2, \ldots \).) Then we say that \( a \) is a collective if

\[
\lim_{n \to \infty} \frac{\sum_{i=1}^{n} a_{k_i}}{n} = \frac{1}{2}
\]

for every \( \phi \) that produces an infinite sequence of \( k_i \)’s.

Examples:

- An infinite sequence in which \( 3/4 \) of the entries are 1 is non-random
- Repeating sequences are non-random
- Non-normal sequences are non-random

Unfortunately, this definition doesn’t quite work — given a sequence \( a \), we can define \( \phi \) so that \( \phi(a_1 a_2 \ldots a_{k-1}) = a_k \), so \( \phi \) predicts \( a \) exactly, so no sequence is a collective. On the other hand, a place-selection-rule “wins” against a measure-zero set of sequences, so, if \( S \) is a countable set of place-selection-rules, then almost every sequence is a collective with respect to \( S \).

What countable set? Church proposed a very natural collection of strategies — the computable strategies.

3.1. Computable functions. Recall the notion of a Turing machine: an infinitely long tape combined with a read/write head. The tape is divided into infinitely many cells and each cell contains one of a finite number of symbols (for example, 0, 1, and *). The head has a finite amount of memory — it is in one of finitely many states at any given time. A computation starts with an input written on the tape (delimited by *’s) and the head in a special state, Start. At each step, the head:

(1) reads the symbol under it
(2) depending on the symbol and the state, might write a new symbol
(3) depending on the symbol and the state, moves one cell left or right
(4) changes its state.

If the head is in the other special state, End, the computation terminates.

(Turing's motivation was to replicate pencil-and-paper computation: the
person doing the computation acts as the head and can remember a finite
amount of information and see a finite portion of the tape at any given time.
Exercise: If you were given two natural numbers with at least one million
digits on an infinitely long tape, how could you calculate their sum?)

A Turing machine is a concrete way to describe a computation; to each
Turing machine $T$, we can associate a partial function $T : \{0, 1\}^* \to \{0, 1\}^*$
so that $T(s)$ is the result of writing $s$ on the tape, setting the head to the
start state, and running the Turing machine until termination. (Note that
a Turing machine may or may not terminate on a given input.) Frequently,
we’ll associate binary strings with the corresponding natural numbers, so we
can view this as a function $T : \mathbb{N}$, etc.

There are four big facts about Turing machines, which can be found in
any basic text on computability:

(1) There are only countably many Turing machines, and each one can
be described with a finite amount of data

(2) They don’t necessarily terminate. Determining whether a given Tur-
ing machine with a given input terminates is called the Halting Prob-
lem, and there is no algorithm to solve it in general.

(3) There is a Universal Turing Machine. To describe this, we’ll need
a way to talk about Turing machines with multiple inputs. If $x \in
\{0, 1\}^*$, let $\ell(x)$ be its length and let

$$\bar{x} = 1^{\ell(x)}0x.$$

This is a self-delimiting form for $x$ – if you concatenate $\bar{x}\bar{y}$, you can
recover $x$ and $y$ unambiguously.

Then there is a Universal Turing Machine $U$ such that $U(\bar{x}\bar{y})$ is
the output of Turing machine $x$ on the input $y$.

(4) The Church-Turing Thesis: Any partial function that can be com-
puted effectively (for instance, anything that can be computed in
the Lambda Calculus, by a program in a programming language, by
hand, etc) can be computed by a Turing machine.

In particular, by 1 and 4, there is a well-defined set of computable func-
tions (really, partial functions), namely the functions that can be computed
by a Turing machine, that we can use to define von Mises-Wald-Church
collectives.

3.2. Other versions of randomness. Unfortunately, this version of ran-
domness isn’t ideal. For one thing, it behaves poorly on finite sequences; if
$a$ is a infinite von Mises-Wald-Church random sequence, then so is $1^{10000}a$
(i.e., $a$ prefixed with 10000 1’s). For another, it doesn’t quite capture all
the statistical properties of a random sequence; there are examples of MWC
random sequences that don’t satisfy, for instance, the law of the iterated logarithm.

Another notion of randomness that’s better-adapted to finite strings is Martin-Löf randomness. The original definition of Martin-Löf randomness was based on effective statistical tests. Very roughly speaking, if a string passes every statistical test, then it’s Martin-Löf random. In this class, I’m going to use a definition based on compressibility which is a little simpler to state.

The idea of compressibility is that there are strings that are easy to describe and strings that are hard to describe; we can describe

\[ 10^{10} \ldots 10^{10}, \]

which is 2494 symbols long, as “‘10’ repeated 1247 times”. This is a dangerous line of reasoning, though, because it leads to descriptions like “the smallest number that cannot be defined in fewer than twelve words”, so we should formalize this.

If \( f : \{0, 1\}^* \to \{0, 1\}^* \) is a partial function and \( w \in \{0, 1\}^* \), we define the complexity of \( w \) with respect to \( f \) as

\[ C_f(w) = \min \{ \ell(p) \mid f(p) = x \}. \]

Then, if \( f \) is the function \( f(n) = (10)^n \), then

\[ C_f(10^{10} \ldots 10^{10}, \text{1247 times}) = \lceil \log_2 1247 \rceil \]

Clearly, this depends on the function you choose, but you have a lot of flexibility if you’re willing to lose a constant factor. Suppose, for instance that \( f_1, \ldots, f_r : \{0, 1\}^* \to \{0, 1\}^* \). Then we can combine the complexity functions to get a new complexity function; if \( F(ip) = f_i(p) \), then

\[ C_F(w) \leq 2\log(r) + 1 + \min \{C_{f_i}(w)\}. \]

Or better, we can use a universal Turing machine to get a “universal” complexity function. Let \( U \) be a Turing machine so that \( U(pd) \) is the output of the Turing machine \( p \) on the input \( d \) and let \( C = C_U \). This function measures the minimum total size of the Turing machine and the data that’s necessary to specify a string.

In particular, for the example \( w = 10^{10} \ldots 10^{10}, \text{1247 times} \), there is some Turing machine \( F \) such that \( F(n) = (10)^n \), so

\[ C((10)^n) \leq \ell(\tilde{F}) + \lceil \log_2 n \rceil. \]

For small values of \( n \), the overhead of specifying the Turing machine might be large, but that overhead is independent of \( n \), so when \( n \) is large, then \( \ell(\tilde{F}) + \lceil \log_2 n \rceil \ll 2n \).

Some properties of \( C \):
(1) $C_U(w) \leq \ell(w) + O(1)$

Proof: Let $I$ be the Turing machine representing the identity function. Then $U(Iw) = w$ for all $w$, so

$$C_U(w) \leq \ell(I) + \ell(w)$$

(2) If $f$ is any computable function, then $C_U(w) \leq C_f(w) + O(1)$

Proof: Let $F$ be the Turing machine representing $f$. Then $U(Fw) = w$ for all $w$, so

$$C_U(w) \leq \ell(F) + \ell(w)$$

(3) $C_U$ is independent of the choice of the model of computation. There are a lot of ways to describe computable functions; we’re using Turing machines, but you can also describe computable functions using the lambda calculus, using a programming language, etc. Suppose that $f(\tilde{p}d)$ is the result of the Python program $p$, evaluated on the input $d$. Then this is a computable function, so there’s a Turing machine $F$ that calculates $F$, and

$$C_U(w) \leq C_f(w) + O(1)$$

Conversely, there’s a Python program that calculates $U$, so

$$C_f(w) \leq C_U(w) + O(1)$$

So, the complexity of $w$ in any programming language is the same up to a constant!

If $c > 0$, we say that $x$ is $c$-compressible if $C(x) < \ell(x) - c$ and $c$-incompressible if $C(x) \geq \ell(x) - c$.

**Proposition 3.1.** For all $c > 0$ and $L > 0$, at least a $(1 - 2^{-c})$-fraction of the strings of length $L$ are $c$-incompressible.

**Proof.** The proof is a counting argument: there are $\sum_{i=0}^{n-1} 2^i = 2^n - 1$ strings of length less than $n$. Therefore,

$$\#\{w \mid C(w) < n\} \leq 2^n - 1,$$

and there are at most $2^{L-c} - 1$ strings of length $L$ that are $c$-compressible. \(\square\)

So most strings are incompressible; a random string is $c$-incompressible with probability at least $(1 - 2^{-c})$. Can we go the other way? Is an incompressible string statistically random? For instance, can we prove:

**Proposition 3.2.** Let $c > 0$. Then for all $n$, if $w \in \{0,1\}^n$ is $c$-incompressible and $k = \sum w_i$, then $k - n/2 = O(n^{1/2})$ (with constant depending on $c$).

**Proof.** We claim that there is some integer $d$ such that if $n$ is large and $k > n/2 + d\sqrt{n}$, then $w$ is $c$-compressible.

Let $k_d = n/2 + d\sqrt{n}$. There are roughly $2^n e^{-2d^2}$ strings of length $n$ with at least $n/2 + d\sqrt{n}$ 1’s; if we list all of these in lexicographic order, then one
of them, say the \( i \)th is \( w \). So we can specify \( w \) exactly by specifying \( n \), \( d \), and \( i \).

Let \( f(\bar{n}d) \) be the \( i \)th string of length \( n \) with at least \( k_d \) 1's. Then

\[
C_U(w) \leq C_f(w) + O(1) \leq 2\log n + 2\log d + n - 2d^2 \log e + O(1),
\]

and if \( d \gg \sqrt{\log n} \), then this is smaller than \( n - c \).

This is nearly what we need; there's a small additional trick to get rid of the \( 2\log n \) in the equation above. Namely, define \( g(n - \ell(i)d) \); generally, \( n - \ell(i) \) is much smaller than \( n \), so if we repeat the analysis, we find that in fact, if \( w \) is \( c \)-compressible, then \( d = O(1) \). \( \square \)

Essentially the same argument works for any effective statistical test – if there is an algorithm to list all of the strings that violate a statistical test for randomness, then you can use the same argument to show that an incompressible string has to satisfy the test!

In fact, Martin-Löf's original definition of randomness was based on statistical tests rather than compressibility. He defined a notion of an effective statistical test and used Turing machines to define a universal effective statistical test; he showed that a string that passes this universal test is in fact incompressible.

The main problem with this definition is that it runs into the paradox we saw at the beginning of this class: the overwhelming majority of strings are incompressible, but it's nearly impossible to show that a given sequence of numbers is in fact incompressible, and any string that we can define concisely is, almost by definition, compressible. On the other hand, we can use exactly this property to prove statistical properties of incompressible or random strings; high complexity leads to interesting statistical properties.

What I want to do this semester is to try to extend this to graphs, surfaces, manifolds, etc — to try to understand how high complexity leads to geometric or topological properties.