Structure and Information

Stephen Binns

Department of Mathematics
King Fahd University of Petroleum and Minerals
binns@kfupm.edu.sa

Computability theory begins with the question:

What is a function?

- Functions are usually introduced to undergraduates as "black boxes" - things which take inputs and produce outputs.
- This is in keeping with the accepted set-theoretical definition of a function as a (single-valued) set of ordered pairs.
- There is no *process* that creates the output from the input - just an unexamined assignment of output to input.

Computability Theory.

Computability theory begins with the question:

What is a function?

- Functions are usually introduced to undergraduates as "black boxes" - things which take inputs and produce outputs.
- This is in keeping with the accepted set-theoretical definition of a function as a (single-valued) set of ordered pairs.
- There is no *process* that creates the output from the input - just an unexamined assignment of output to input.

Computability theory begins with the question:

What is a function?

- Functions are usually introduced to undergraduates as "black boxes" - things which take inputs and produce outputs.
- This is in keeping with the accepted set-theoretical definition of a function as a (single-valued) set of ordered pairs.
- There is no *process* that creates the output from the input - just an unexamined assignment of output to input.

In computability theory we are concerned with those functions (from $\mathbb{N}$ to $\mathbb{N}$) that can be evaluated by some kind of algorithmic or mechanistic process.

Definition (informal)

A *computable function* is a function whose black box is a *machine*.

- There is no *a priori* reason to believe that this vague notion of *machine* is formally definable - or even coherent.
- But it turns out that it can be captured by the mathematically definable concept of a *Turing Machine*.

In computability theory we are concerned with those functions (from $\mathbb{N}$ to $\mathbb{N}$) that can be evaluated by some kind of algorithmic or mechanistic process.

Definition (informal)

A *computable function* is a function whose black box is a *machine*.

- There is no *a priori* reason to believe that this vague notion of *machine* is formally definable - or even coherent.
- But it turns out that it can be captured by the mathematically definable concept of a *Turing Machine*.

In computability theory we are concerned with those functions (from $\mathbb{N}$ to $\mathbb{N}$) that can be evaluated by some kind of algorithmic or mechanistic process.

Definition (informal)

A *computable function* is a function whose black box is a *machine*.

- There is no *a priori* reason to believe that this vague notion of *machine* is formally definable - or even coherent.
- But it turns out that it can be captured by the mathematically definable concept of a *Turing Machine*.

Alan Turing's Machines

- A Turing machine (TM) consists of an infinite (in one direction) tape divided into cells. Each cell has a 0 or a 1 written in it.
- The machine can read the contents of a cell and write over it if necessary. The reading head can move to the left or right as required. At each stage of operation, the machine is in a given *state* - indexed by a natural number.
- Inputs and outputs are given by (finite) initial sequences of 1s.

Alan Turing's Machines

- A Turing machine (TM) consists of an infinite (in one direction) tape divided into cells. Each cell has a 0 or a 1 written in it.
- The machine can read the contents of a cell and write over it if necessary. The reading head can move to the left or right as required. At each stage of operation, the machine is in a given *state* - indexed by a natural number.
- Inputs and outputs are given by (finite) initial sequences of 1s.

- A Turing machine (TM) consists of an infinite (in one direction) tape divided into cells. Each cell has a 0 or a 1 written in it.
- The machine can read the contents of a cell and write over it if necessary. The reading head can move to the left or right as required. At each stage of operation, the machine is in a given *state* - indexed by a natural number.
- Inputs and outputs are given by (finite) initial sequences of 1s.

The TM is programmed with lists of instructions of the form:

If the machine is in state n reading i $\in \{0, 1\}$,
write j $\in \{0, 1\}$, *move to the left (or right)*
and go into state m.

Definition (formal)

A *computable function* is a function that can be evaluated using a Turing machine.

There is nothing canonical about this definition - there are potentially thousands of other definitions of *machine* - and hence of computable function.

Essential Properties of TMs.

- Church-Turing Thesis: Any algorithmic procedure can be carried out on a Turing machine.
- The domain and co-domain of a computable function can be any countable sets of finitely describable objects. For example finite binary strings, finite graphs, finite algebraic structures and so on.
- Every attempt to define the intuitive concept of a computing machine has turned out to be no stronger than the TM definition - that is every general computing machine computes only Turing-computable functions.
- There is a *universal* Turing machine - one that can take a program and a natural number n as input and implement that program on n.

Essential Properties of TMs.

- Church-Turing Thesis: Any algorithmic procedure can be carried out on a Turing machine.
- The domain and co-domain of a computable function can be any countable sets of finitely describable objects. For example finite binary strings, finite graphs, finite algebraic structures and so on.
- Every attempt to define the intuitive concept of a computing machine has turned out to be no stronger than the TM definition - that is every general computing machine computes only Turing-computable functions.
- There is a *universal* Turing machine - one that can take a program and a natural number n as input and implement that program on n.

Essential Properties of TMs.

- Church-Turing Thesis: Any algorithmic procedure can be carried out on a Turing machine.
- The domain and co-domain of a computable function can be any countable sets of finitely describable objects. For example finite binary strings, finite graphs, finite algebraic structures and so on.
- Every attempt to define the intuitive concept of a computing machine has turned out to be no stronger than the TM definition - that is every general computing machine computes only Turing-computable functions.
- There is a *universal* Turing machine - one that can take a program and a natural number n as input and implement that program on n.

Essential Properties of TMs.

- Church-Turing Thesis: Any algorithmic procedure can be carried out on a Turing machine.
- The domain and co-domain of a computable function can be any countable sets of finitely describable objects. For example finite binary strings, finite graphs, finite algebraic structures and so on.
- Every attempt to define the intuitive concept of a computing machine has turned out to be no stronger than the TM definition - that is every general computing machine computes only Turing-computable functions.
- There is a *universal* Turing machine - one that can take a program and a natural number n as input and implement that program on n.

Essential properties cont.

- No consideration of time or space usage is made.
- When studying computability theory we rarely have any particular model of computation in mind. Any programable generalised device is a good intuition.
- Countable objects may be *computable* or *non-computable* - that is there may be a Turing machine that can produce them, or there may not be.

  For example $\pi = 3.14159...$ is computable. But...

  Matiyasevich's solution to Hilbert's Tenth Problem: There is no computer program that given a Diophantine equation decides whether or not it has a solution, so the set of solvable Diophantine equations is non-computable.

Essential properties cont.

- No consideration of time or space usage is made.
- When studying computability theory we rarely have any particular model of computation in mind. Any programable generalised device is a good intuition.
- Countable objects may be *computable* or *non-computable* - that is there may be a Turing machine that can produce them, or there may not be.

  For example $\pi = 3.14159...$ is computable. But...

  Matiyasevich's solution to Hilbert's Tenth Problem: There is no computer program that given a Diophantine equation decides whether or not it has a solution, so the set of solvable Diophantine equations is non-computable.

Essential properties cont.

- No consideration of time or space usage is made.
- When studying computability theory we rarely have any particular model of computation in mind. Any programable generalised device is a good intuition.
- Countable objects may be *computable* or *non-computable* - that is there may be a Turing machine that can produce them, or there may not be.

For example $\pi = 3.14159...$ is computable. But...

Matiyasevich's solution to Hilbert's Tenth Problem: There is no computer program that given a Diophantine equation decides whether or not it has a solution, so the set of solvable Diophantine equations is non-computable.

Essential properties cont.

- No consideration of time or space usage is made.
- When studying computability theory we rarely have any particular model of computation in mind. Any programable generalised device is a good intuition.
- Countable objects may be *computable* or *non-computable* - that is there may be a Turing machine that can produce them, or there may not be.

For example $\pi = 3.14159...$ is computable. But...

Matiyasevich's solution to Hilbert's Tenth Problem: There is no computer program that given a Diophantine equation decides whether or not it has a solution, so the set of solvable Diophantine equations is non-computable.

Essential properties cont.

- No consideration of time or space usage is made.
- When studying computability theory we rarely have any particular model of computation in mind. Any programable generalised device is a good intuition.
- Countable objects may be *computable* or *non-computable* - that is there may be a Turing machine that can produce them, or there may not be.

  For example $\pi = 3.14159...$ is computable. But...

  Matiyasevich's solution to Hilbert's Tenth Problem: There is no computer program that given a Diophantine equation decides whether or not it has a solution, so the set of solvable Diophantine equations is non-computable.

Kolmogorov Complexity.

How complex is a finite binary string?

1000110101010101000101101001001000001010101
is quite complex, but

0000000000000000000011111111111111111111111
is not.

Definition

The *Kolmogorov complexity of* σ, denoted $C(\sigma)$, is the length of shortest program (written in binary) needed to output σ.

We think of the program as a *description* of the output string. Strings with short descriptions have low complexity.

Kolmogorov Complexity.

How complex is a finite binary string?

10001101010101010001011010010010000101010101
is quite complex, but

00000000000000000000011111111111111111111111
is not.

Definition

The *Kolmogorov complexity of* σ, denoted $C(\sigma)$, is the length of shortest program (written in binary) needed to output σ.

We think of the program as a *description* of the output string. Strings with short descriptions have low complexity.

Kolmogorov Complexity.

How complex is a finite binary string?

1000110101010101000101101001001000001010101
is quite complex, but

0000000000000000000001111111111111111111111
is not.

Definition

The *Kolmogorov complexity of* σ, denoted $C(\sigma)$, is the length of shortest program (written in binary) needed to output σ.

We think of the program as a *description* of the output string. Strings with short descriptions have low complexity.

$C(\sigma)$ depends on the type of computer and the language used, so we should define:

$C_M(\sigma)$ = length of shortest program needed to output σ on machine M.

Fact: If M and N are two different (universal) machines, then there is a constant $k = k(M, N)$ such that for all σ

$$C_M(\sigma) \leqslant C_N(\sigma) + k.$$

So we just fix a (universal) machine, U and let $C(\sigma) := C_U(\sigma)$. Then for any machine M,

$$C(\sigma) = C_M(\sigma) + k,$$

(with k depending only on M.)

$C(\sigma)$ depends on the type of computer and the language used, so we should define:

$C_M(\sigma)$ = length of shortest program needed to output σ on machine M.

Fact: If M and N are two different (universal) machines, then there is a constant $k = k(M, N)$ such that for all σ

$$C_M(\sigma) \leqslant C_N(\sigma) + k.$$

So we just fix a (universal) machine, U and let $C(\sigma) := C_U(\sigma)$. Then for any machine M,

$$C(\sigma) = C_M(\sigma) + k,$$

(with k depending only on M.)

$C(\sigma)$ depends on the type of computer and the language used, so we should define:

$C_M(\sigma)$ = length of shortest program needed to output σ on machine M.

Fact: If M and N are two different (universal) machines, then there is a constant $k = k(M, N)$ such that for all σ

$$C_M(\sigma) \leqslant C_N(\sigma) + k.$$

So we just fix a (universal) machine, U and let $C(\sigma) := C_U(\sigma)$. Then for any machine M,

$$C(\sigma) = C_M(\sigma) + k,$$

(with k depending only on M.)

$C(\sigma)$ depends on the type of computer and the language used, so we should define:

$C_M(\sigma)$ = length of shortest program needed to output σ on machine M.

Fact: If M and N are two different (universal) machines, then there is a constant $k = k(M, N)$ such that for all σ

$$C_M(\sigma) \leqslant C_N(\sigma) + k.$$

So we just fix a (universal) machine, U and let $C(\sigma) := C_U(\sigma)$. Then for any machine M,

$$C(\sigma) = C_M(\sigma) + k,$$

(with k depending only on M.)

Gödel's Incompleteness Theorem

- Kurt Gödel proved in 1931 that mathematics was *incomplete*. He produced a mathematical sentence (in fact a sentence in number theory) that could neither be proved nor disproved from the axioms of number theory (Peano axioms). Any sufficiently strong axiom system will have these sentences. If we take ZFC - the axioms of Set Theory - as the axiomatic foundation for mathematics, then there are mathematical statements that can neither be proved nor disproved using any mathematical technique (for example the Continuum Hypothesis).
- The sentence Gödel produced was extremely complicated and had no independent mathematical standing other than to prove his theorem.
- There are however simple statements in Komolgorov complexity theory that cannot be proved or disproved mathematically (from ZFC).

Gödel's Incompleteness Theorem

- Kurt Gödel proved in 1931 that mathematics was *incomplete*. He produced a mathematical sentence (in fact a sentence in number theory) that could neither be proved nor disproved from the axioms of number theory (Peano axioms). Any sufficiently strong axiom system will have these sentences. If we take ZFC - the axioms of Set Theory - as the axiomatic foundation for mathematics, then there are mathematical statements that can neither be proved nor disproved using any mathematical technique (for example the Continuum Hypothesis).
- The sentence Gödel produced was extremely complicated and had no independent mathematical standing other than to prove his theorem.
- There are however simple statements in Komolgorov complexity theory that cannot be proved or disproved mathematically (from ZFC).

Gödel's Incompleteness Theorem

- Kurt Gödel proved in 1931 that mathematics was *incomplete*. He produced a mathematical sentence (in fact a sentence in number theory) that could neither be proved nor disproved from the axioms of number theory (Peano axioms). Any sufficiently strong axiom system will have these sentences. If we take ZFC - the axioms of Set Theory - as the axiomatic foundation for mathematics, then there are mathematical statements that can neither be proved nor disproved using any mathematical technique (for example the Continuum Hypothesis).
- The sentence Gödel produced was extremely complicated and had no independent mathematical standing other than to prove his theorem.
- There are however simple statements in Komolgorov complexity theory that cannot be proved or disproved mathematically (from ZFC).

Gödel's Incompleteness Theorem

- Kurt Gödel proved in 1931 that mathematics was *incomplete*. He produced a mathematical sentence (in fact a sentence in number theory) that could neither be proved nor disproved from the axioms of number theory (Peano axioms). Any sufficiently strong axiom system will have these sentences. If we take ZFC - the axioms of Set Theory - as the axiomatic foundation for mathematics, then there are mathematical statements that can neither be proved nor disproved using any mathematical technique (for example the Continuum Hypothesis).
- The sentence Gödel produced was extremely complicated and had no independent mathematical standing other than to prove his theorem.
- There are however simple statements in Komolgorov complexity theory that cannot be proved or disproved mathematically (from ZFC).

Theorem

There is an $N \in \mathbb{N}$ such that, for any binary string σ, no statement of the form $C(\sigma) \geqslant N$ is provable in mathematics (from ZFC).

This is peculiar because it is easy to see that

$$\forall n \in \mathbb{N}\ \exists \sigma\ C(\sigma) > n,$$

because there are only a limited number of short descriptions, and so there must be strings with arbitrarily long descriptions. In other words it is impossible to prove that any given binary string has a complexity above a certain limit, however one can prove that such strings must exist.

Chaitin's Incompleteness Theorem

Theorem

There is an $N \in \mathbb{N}$ such that, for any binary string σ, no statement of the form $C(\sigma) \geqslant N$ is provable in mathematics (from ZFC).

This is peculiar because it is easy to see that

$$\forall n \in \mathbb{N}\ \exists \sigma\ C(\sigma) > n,$$

because there are only a limited number of short descriptions, and so there must be strings with arbitrarily long descriptions. In other words it is impossible to prove that any given binary string has a complexity above a certain limit, however one can prove that such strings must exist.

Proof.

Consider a machine M that works as follows: taking input n in binary form, it searches for a string σ and a proof from ZFC of the statement $C(\sigma) > n$. If our theorem is incorrect, then the computer will always eventually find such a string. The computer then outputs σ. Thus n serves as a description for σ via machine M. So

$$C_M(\sigma) \leqslant \log_2(n) + 1.$$

Also, $C(\sigma) \leqslant C_M(\sigma) + k$, where k is a constant depending only on M. Now choose N so that

$$N > \log_2(N) + 1 + k.$$

Proof of Chaitin's Incompleteness Theorem

Proof.

Consider a machine M that works as follows: taking input n in binary form, it searches for a string σ and a proof from ZFC of the statement $C(\sigma) > n$. If our theorem is incorrect, then the computer will always eventually find such a string. The computer then outputs σ. Thus n serves as a description for σ via machine M. So

$$C_M(\sigma) \leqslant \log_2(n) + 1.$$

Also, $C(\sigma) \leqslant C_M(\sigma) + k$, where k is a constant depending only on M. Now choose N so that

$$N > \log_2(N) + 1 + k.$$

Proof of Chaitin's Incompleteness Theorem

Proof.

Consider a machine M that works as follows: taking input n in binary form, it searches for a string σ and a proof from ZFC of the statement $C(\sigma) > n$. If our theorem is incorrect, then the computer will always eventually find such a string. The computer then outputs σ. Thus n serves as a description for σ via machine M. So

$$C_M(\sigma) \leqslant \log_2(n) + 1.$$

Also, $C(\sigma) \leqslant C_M(\sigma) + k$, where k is a constant depending only on M. Now choose N so that

$$N > \log_2(N) + 1 + k.$$

Proof.

Consider a machine M that works as follows: taking input n in binary form, it searches for a string σ and a proof from ZFC of the statement $C(\sigma) > n$. If our theorem is incorrect, then the computer will always eventually find such a string. The computer then outputs σ. Thus n serves as a description for σ via machine M. So

$$C_M(\sigma) \leqslant \log_2(n) + 1.$$

Also, $C(\sigma) \leqslant C_M(\sigma) + k$, where k is a constant depending only on M. Now choose N so that

$$N > \log_2(N) + 1 + k.$$

□

Proof cont.

Then run machine M on input N. If σ is the output, then we have:

$$C(\sigma) \leqslant C_M(\sigma) + k \leqslant \log_2(N) + 1 + k < N.$$

Thus $C(\sigma) < N$ and yet ZFC proves that $C(\sigma) > N$. So if ZFC is consistent, we get a contradiction. $\square$

Proof cont.

Then run machine M on input N. If σ is the output, then we have:

$$C(\sigma) \leqslant C_M(\sigma) + k \leqslant \log_2(N) + 1 + k < N.$$

Thus $C(\sigma) < N$ and yet ZFC proves that $C(\sigma) > N$. So if ZFC is consistent, we get a contradiction. □

Proof cont.

Then run machine M on input N. If σ is the output, then we have:

$$C(\sigma) \leqslant C_M(\sigma) + k \leqslant \log_2(N) + 1 + k < N.$$

Thus $C(\sigma) < N$ and yet ZFC proves that $C(\sigma) > N$. So if ZFC is consistent, we get a contradiction. $\square$

Proof cont.

Then run machine M on input N. If σ is the output, then we have:

$$C(\sigma) \leqslant C_M(\sigma) + k \leqslant \log_2(N) + 1 + k < N.$$

Thus $C(\sigma) < N$ and yet ZFC proves that $C(\sigma) > N$. So if ZFC is consistent, we get a contradiction. $\square$

Hausdorff Dimension.

For $A \subseteq \mathbb{R}^n$ we define, for each $s \in \mathbb{Q}$:

Definition

$$\mathcal{H}^s_\delta(A) = \inf\Big\{k_n \sum_{B_i \in \mathcal{C}} \operatorname{diam}(B_i)^s\Big\}$$

where the infimum is taken over all open covers $\mathcal{C}$ of A consisting of n-dimensional balls B_i of diameter less than δ.

Definition

$$\mathcal{H}^s(A) = \lim_{\delta \to 0} \mathcal{H}^s_\delta(A).$$

Definition

The *Hausdorff dimension* of $A \subseteq \mathbb{R}^n$ is

$$\dim_{\mathcal{H}}(A) = \inf\{s : \mathcal{H}^s(A) = 0\} = \sup\{s : \mathcal{H}^s(A) = \infty\}.$$

Hausdorff Dimension.

For $A \subseteq \mathbb{R}^n$ we define, for each $s \in \mathbb{Q}$:

Definition

$$\mathcal{H}^s_\delta(A) = \inf\Big\{k_n \sum_{B_i \in \mathcal{C}} \operatorname{diam}(B_i)^s\Big\}$$

where the infimum is taken over all open covers $\mathcal{C}$ of A consisting of n-dimensional balls B_i of diameter less than δ.

Definition

$$\mathcal{H}^s(A) = \lim_{\delta \to 0} \mathcal{H}^s_\delta(A).$$

Definition

The *Hausdorff dimension* of $A \subseteq \mathbb{R}^n$ is

$$\dim_{\mathcal{H}}(A) = \inf\{s : \mathcal{H}^s(A) = 0\} = \sup\{s : \mathcal{H}^s(A) = \infty\}.$$

Hausdorff Dimension.

For $A \subseteq \mathbb{R}^n$ we define, for each $s \in \mathbb{Q}$:

Definition

$$\mathcal{H}^s_\delta(A) = \inf\left\{k_n \sum_{B_i \in \mathcal{C}} \operatorname{diam}(B_i)^s\right\}$$

where the infimum is taken over all open covers $\mathcal{C}$ of A consisting of n-dimensional balls B_i of diameter less than δ.

Definition

$$\mathcal{H}^s(A) = \lim_{\delta \to 0} \mathcal{H}^s_\delta(A).$$

Definition

The *Hausdorff dimension* of $A \subseteq \mathbb{R}^n$ is

$$\dim_{\mathcal{H}}(A) = \inf\{s : \mathcal{H}^s(A) = 0\} = \sup\{s : \mathcal{H}^s(A) = \infty\}.$$

Computable Hausdorff Dimension.

Definition

We say $\mathcal{H}^{1,s}(A) = 0$ if there is a computable sequence of open covers $\mathcal{C}_n$ for A, each of which is a computable sequence of open balls $\langle B_i \rangle$ and such that for each n

$$\sum_{B_i \in \mathcal{C}_n} \mathrm{diam}(B_i)^s \leqslant 2^{-n}.$$

Definition

The *computable Hausdorff dimension* of $A \subseteq \mathbb{R}^n$ is

$$\dim^1_{\mathcal{H}}(A) = \inf\{s : \mathcal{H}^{1,s}(A) = 0\}.$$

Computable Hausdorff dimension - Results.

- For any $A \subseteq \mathbb{R}$, $\dim^1_{\mathcal{H}}(A) \geqslant \dim_{\mathcal{H}}(A)$.
- For $X \in \mathbb{R}$ it is possible (in fact usual) that

$$\dim^1_{\mathcal{H}}(X) := \dim^1_{\mathcal{H}}(\{X\}) \neq 0.$$

- $\dim^1_{\mathcal{H}}(A) = \sup\{\dim^1_{\mathcal{H}}(X) : X \in A\}$.
- For *computably closed classes* (that is, classes whose complements are computable sequences of open balls),

$$\dim^1_{\mathcal{H}}(P) = \dim_{\mathcal{H}}(P).$$

Computable Hausdorff dimension - Results.

- For any $A \subseteq \mathbb{R}$, $\dim^1_{\mathcal{H}}(A) \geqslant \dim_{\mathcal{H}}(A)$.
- For $X \in \mathbb{R}$ it is possible (in fact usual) that

$$\dim^1_{\mathcal{H}}(X) := \dim^1_{\mathcal{H}}(\{X\}) \neq 0.$$

- $\dim^1_{\mathcal{H}}(A) = \sup\{\dim^1_{\mathcal{H}}(X) : X \in A\}$.
- For *computably closed classes* (that is, classes whose complements are computable sequences of open balls),

$$\dim^1_{\mathcal{H}}(P) = \dim_{\mathcal{H}}(P).$$

Computable Hausdorff dimension - Results.

- For any $A \subseteq \mathbb{R}$, $\dim^1_{\mathcal{H}}(A) \geqslant \dim_{\mathcal{H}}(A)$.
- For $X \in \mathbb{R}$ it is possible (in fact usual) that

$$\dim^1_{\mathcal{H}}(X) := \dim^1_{\mathcal{H}}(\{X\}) \neq 0.$$

- $\dim^1_{\mathcal{H}}(A) = \sup\{\dim^1_{\mathcal{H}}(X) : X \in A\}$.
- For *computably closed classes* (that is, classes whose complements are computable sequences of open balls),

$$\dim^1_{\mathcal{H}}(P) = \dim_{\mathcal{H}}(P).$$

Computable Hausdorff dimension - Results.

- For any $A \subseteq \mathbb{R}$, $\dim^1_{\mathcal{H}}(A) \geqslant \dim_{\mathcal{H}}(A)$.
- For $X \in \mathbb{R}$ it is possible (in fact usual) that

$$\dim^1_{\mathcal{H}}(X) := \dim^1_{\mathcal{H}}(\{X\}) \neq 0.$$

- $\dim^1_{\mathcal{H}}(A) = \sup\{\dim^1_{\mathcal{H}}(X) : X \in A\}$.
- For *computably closed classes* (that is, classes whose complements are computable sequences of open balls),

$$\dim^1_{\mathcal{H}}(P) = \dim_{\mathcal{H}}(P).$$

- If $X \in \mathbb{R}$, then

$$\dim^1_{\mathcal{H}}(X) = \liminf_n \frac{C(X \upharpoonright n)}{n}.$$

That is, the computable Hausdorff dimension of an element of $\mathbb{R}$ is the limit infimum of the *information density* of its initial segments.

- If P is a computably closed class, then

$$\dim_{\mathcal{H}}(P) = \sup_{X \in P}\{\liminf_n \frac{C(X \upharpoonright n)}{n}\}.$$

Computable Hausdorff dimension and Complexity.

- If $X \in \mathbb{R}$, then

$$\dim^1_{\mathcal{H}}(X) = \liminf_n \frac{C(X \upharpoonright n)}{n}.$$

  That is, the computable Hausdorff dimension of an element of $\mathbb{R}$ is the limit infimum of the *information density* of its initial segments.

- If P is a computably closed class, then

$$\dim_{\mathcal{H}}(P) = \sup_{X \in P}\{\liminf_n \frac{C(X \upharpoonright n)}{n}\}.$$

The previous theorem again...

If P is a computably closed class, then

$$\dim_{\mathcal{H}}(P) = \sup_{X \in P}\{\liminf_{n} \frac{C(X \upharpoonright n)}{n}\}.$$

Notice:

- This is a “global vs local" type equation.
- It is also a “classical vs computable" type equation.
- Is it possible to compute classical dimensions of sets using the concept of complexity?

The previous theorem again...

If P is a computably closed class, then

$$\dim_{\mathcal{H}}(P) = \sup_{X \in P}\{\liminf_{n} \frac{C(X \upharpoonright n)}{n}\}.$$

Notice:

- This is a “global vs local" type equation.
- It is also a “classical vs computable" type equation.
- Is it possible to compute classical dimensions of sets using the concept of complexity?

The previous theorem again...

If P is a computably closed class, then

$$\dim_{\mathcal{H}}(P) = \sup_{X \in P}\{\liminf_{n} \frac{C(X \upharpoonright n)}{n}\}.$$

Notice:

- This is a “global vs local" type equation.
- It is also a “classical vs computable" type equation.
- Is it possible to compute classical dimensions of sets using the concept of complexity?

The previous theorem again...

If P is a computably closed class, then

$$\dim_{\mathcal{H}}(P) = \sup_{X \in P}\{\liminf_{n} \frac{C(X \upharpoonright n)}{n}\}.$$

Notice:

- This is a “global vs local" type equation.
- It is also a “classical vs computable" type equation.
- Is it possible to compute classical dimensions of sets using the concept of complexity?

An example.

Theorem

If $\mathcal{C}$ = the Cantor middle third set, then $\dim_{\mathcal{H}}(\mathcal{C}) = \ln 2/\ln 3$.

Sketch of Proof.

Consider the elements of unit interval to be identified with their ternary expansions. $\mathcal{C}$ consists of all elements with no 1s in them. Given any n there are 2^n ternary strings that have no 1s in them. So it requires about $\log_3(2^n) = n\ln(2)/\ln(3)$ bits to describe one in ternary. Therefore, for any $X \in \mathcal{C}$, $C(X \upharpoonright n)$ is (in general) about $n\ln(2)/\ln(3)$. As $\mathcal{C}$ is a computably closed class,

$$\dim_{\mathcal{H}}(\mathcal{C}) = \liminf_n \frac{n\ln 2/\ln 3}{n} = \ln 2/\ln 3.$$

□

An example.

Theorem

If $\mathcal{C}$ =the Cantor middle third set, then $\dim_{\mathcal{H}}(\mathcal{C}) = \ln 2/\ln 3$.

Sketch of Proof.

Consider the elements of unit interval to be identified with their ternary expansions. $\mathcal{C}$ consists of all elements with no 1s in them. Given any n there are 2^n ternary strings that have no 1s in them. So it requires about $\log_3(2^n) = n\ln(2)/\ln(3)$ bits to describe one in ternary. Therefore, for any $X \in \mathcal{C}$, $C(X \upharpoonright n)$ is (in general) about $n\ln(2)/\ln(3)$. As $\mathcal{C}$ is a computably closed class,

$$\dim_{\mathcal{H}}(\mathcal{C}) = \liminf_n \frac{n\ln 2/\ln 3}{n} = \ln 2/\ln 3.$$

□

An example.

Theorem

If $\mathcal{C}$ = the Cantor middle third set, then $\dim_{\mathcal{H}}(\mathcal{C}) = \ln 2/\ln 3$.

Sketch of Proof.

Consider the elements of unit interval to be identified with their ternary expansions. $\mathcal{C}$ consists of all elements with no 1s in them. Given any n there are 2^n ternary strings that have no 1s in them. So it requires about $\log_3(2^n) = n\ln(2)/\ln(3)$ bits to describe one in ternary. Therefore, for any $X \in \mathcal{C}$, $C(X \upharpoonright n)$ is (in general) about $n\ln(2)/\ln(3)$. As $\mathcal{C}$ is a computably closed class,

$$\dim_{\mathcal{H}}(\mathcal{C}) = \liminf_n \frac{n\ln 2/\ln 3}{n} = \ln 2/\ln 3.$$

$\square$

Notes on the proof.

- It is essential here is that $\mathcal{C}$ contains a random sequence of 0s and 2's - that is one of maximum complexity - so it achieves this dimension. It is also essential that $\mathcal{C}$ is a computably closed class.
- The dimension was calculated with reference to only one of its elements. Any subset of $\mathcal{C}$ that contains a random sequence of 0s and 2 will have the same Hausdorff dimension.
- No use of self-similarity is made.

Notes on the proof.

- It is essential here is that $\mathcal{C}$ contains a random sequence of 0s and 2's - that is one of maximum complexity - so it achieves this dimension. It is also essential that $\mathcal{C}$ is a computably closed class.
- The dimension was calculated with reference to only one of its elements. Any subset of $\mathcal{C}$ that contains a random sequence of 0s and 2 will have the same Hausdorff dimension.
- No use of self-similarity is made.

Notes on the proof.

- It is essential here is that $\mathcal{C}$ contains a random sequence of 0s and 2's - that is one of maximum complexity - so it achieves this dimension. It is also essential that $\mathcal{C}$ is a computably closed class.
- The dimension was calculated with reference to only one of its elements. Any subset of $\mathcal{C}$ that contains a random sequence of 0s and 2 will have the same Hausdorff dimension.
- No use of self-similarity is made.

The End.

THANK YOU!