Computational complexity with experiments as oracles

Edwin Beggs†, José Félix Costa‡, Bruno Loff‡ & John Tucker†

† School of Physical Sciences, Swansea University, Swansea, SA2 8PP, UK
‡ Instituto Superior Técnico, Technical University of Lisbon, Lisbon, Portugal
& ‡ Centro de Matemática e Aplicações Fundamentais, University of Lisbon, Lisbon, Portugal

March 20, 2008

Abstract

We discuss combining physical experiments with machine computations and introduce a form of analogue-digital Turing machine. We examine in detail a case study where an experimental procedure based on Newtonian kinematics is combined with a class of Turing machines. Three forms of analogue-digital machine are studied, in which physical parameters can be set exactly and approximately. Using non-uniform complexity theory, and some probability, we prove theorems that show that these machines can compute more than classical Turing machines.

1 Introduction

Consider a physical experiment in which some independent physical quantities $x_1, x_2, \ldots$ may vary and a series of corresponding behaviours $y_1, y_2, \ldots$ can be observed. The experiment defines a function $y_i = f(x_i)$ or, more generally, a relation $r(x_i, y_i)$ for $i = 1, 2, \ldots$.

1. *How can we use such an experiment to compute a function f, or decide a relation r, on a set of data? How do we measure its performance?*

To answer the question we must be precise about data representation by physical quantities, experimental equipment and experimental procedures, all of which will be based upon a thorough understanding of one (or more) physical theories. Let us call this type of calculation *experimental computation*; clearly, experimental computation includes calculation by analogue computers. For any type of experimental computation we can ask the question:

2. *Is experimental computation equivalent to algorithmic computation? If not, what is the difference?*

Perhaps, for a certain class of experiments, we can show that a function f, or relation r, is computable by an experiment belonging to the class if, and only if, it is computable by an algorithm? Perhaps, we can show there is a difference in performance: could an algorithmic problem of exponential time be computable by an experiment in polynomial time? The question is relevant for analysing the concept of a faithful computable simulation of a physical experiment.

There is a third question, connected to the last:

3. *If a physical experiment were to be coupled with algorithms, would new functions and relations become computable or, at least, computable more efficiently?*

In this case, can we explore if, and when, analogue-digital computation extends the purely digital. To pursue this question, we imagine using an experiment as an oracle to a Turing machine, which on being presented with, say, x_i as its i-th question, returns y_i to the Turing machine. In this paper we will consider this idea of coupling physical experiments and Turing machines in detail.

The first thing to note is that choosing a physical experiment to use as an oracle is a major undertaking. The experiment comes with plenty of theoretical baggage: concepts of equipment, experimental procedure, measurement, observable behaviour, etc. Theorising about the input-output behaviour of the experiment, and about the analogue-digital machine, involves us with all three questions above.

In earlier work [5], an experiment was devised to measure the position of the vertex of a wedge to arbitrary accuracy, by scattering particles that obey some laws of elementary Newtonian kinematics. Let *SME* denote this *scatter machine experiment.* The Newtonian theory was specified precisely and *SME* was put under a theoretical microscope: theorems were proved that showed that *the experiment was able to compute positions that were not computable by algorithms.* Indeed, the experiment *SME* could, in principle, measure *any* real number. Thus, [5] contains a careful attempt to answer question 2 above, in the negative; it does so using a methodology developed and applied in earlier studies [3, 4].

To address question 3, here we propose to use the *SME* as an oracle to a Turing machine and to classify the computational power of the new type of machine, which we call an *analogue-digital scatter machine.* Given the results in [5], the use of *SME* could enhance the computational power and efficiency of the Turing machine. To investigate this, we must establish some principles that do not depend upon the *SME.*

In a Turing machine the oracle is normally specified very abstractly by a set. Here we have to design a new machine where the oracle is replaced by a specification of some physical equipment and an experimental procedure for operating it. The design of the new machine depends heavily upon the interface and interaction between the experiment and the Turing machine.

Consider the Turing machine performing a digital computation in which it generates queries that activate an experiment. The Turing machine will generate a (dyadic) rational number x_i which can be used to set an experimental parameter p_i of *SME* in one of two ways: we call the machine *error-free* if $p_i = x_i$; and *error-prone* if we can ensure that $p_i \in [x_i - \varepsilon, x_i + \varepsilon]$ for an error margin $\varepsilon > 0$. In the error-prone case, we further differentiate between fixed accuracy, where $\varepsilon > 0$ is fixed for the particular *SME*, and arbitrary accuracy, where $\varepsilon > 0$ can be made arbitrarily small.

Secondly, the whole process of receiving data from, and passing digital data to, the *SME* is some form of analogue-digital (AD) protocol that consumes time (or other physical resources). By classifying AD-protocols, we introduce these different types of analogue-digital Turing computation: *polynomial time error-free*, *exponential time error-free* and *polynomial time error-prone* (with arbitrary and fixed precision).

Thirdly, the *SME* has within it a wedge whose vertex is the site of singular physical behaviour: a particle hitting the vertex of the wedge may scatter randomly. Since the experiment may be non-deterministic, the machines are divided further into deterministic and non-deterministic.

We analyse the complexity of computations by these new machines. Following inspiration found in the work of Siegelmann and Sontag [18], we use *non-uniform complexity classes* of the form $\mathcal{B}/\mathcal{F}$ where $\mathcal{B}$ is the class of computations and $\mathcal{F}$ is the advice class. Examples of interest for $\mathcal{B}$ are P and BPP; examples for $\mathcal{F}$ are *poly* and *log*. The power of the machines will correspond to different choices of $\mathcal{B}$ and $\mathcal{F}$.

We prove the following for the error-free machines:

Theorem 1.0.1 *The class of sets which are decidable in polynomial time by error-free deterministic AD scatter machines with a polynomial AD-protocol is exactly* P/*poly*.

Theorem 1.0.2 *The class of sets which are decidable in polynomial time by error-free deterministic AD scatter machines with strictly an exponential AD-protocol is exactly* P/*log*∗.

For the error-prone machines we prove:

Theorem 1.0.3 *An AD-scatter machine (either error free or error prone arbitrary precision) with a polynomial AD-protocol can decide* P/*poly in polynomial time.*

Theorem 1.0.4 *An error-prone analog-digital scatter machine with arbitrary precision and with an exponential AD-protocol can decide any set in* P/*log*∗ *in polynomial time.*

Theorem 1.0.5 *An error-prone analog-digital scatter machine with fixed precision, given an independent uniformly distributed cannon position in the error interval, can decide any set in* BPP//*log*∗ *in polynomial time (independently of the protocol).*

First, in Section 2, we recall briefly the principles used to investigate experimental computation and tackle the questions 1 and 2 above; we extend the methodology to tackle the new question 3. The method was introduced and applied in [3, 4] and its principles were used in the analysis of the scatter machine experiment in [5]. The reader is referred to these papers for further explanations. In Section 3, we recall the *SME* from [5] and in Section 4 we recall some theory of Turing machines with various oracles. In Section 5 we define the analogue-digital scatter machines. In the subsequent sections we undertake the classification of their complexity. Finally, in Section 9, we reflect on some possible interpretations and implications of these results. We assume the reader is familiar with our [5]. Some speculations on mechanics and complexity are in [23].

2 Investigating experimental computation using physical theories

We summarise a methodology for the theoretical investigation of Questions 1 and 2 about experimental computation and extend it to answer Question 3.

2.1 Experimental computation

To capture the diversity of examples, old and new, and to organise the subject of computation as physical process, we have introduced the idea of *experimental computation.* In this paper, we will need only some simple general ideas about experimental computation. For example, we suppose a partial function $y = f(x)$, or relation $r(x, y)$, can be computed by an experimental procedure in three stages:

(i) input data x are used to determine initial conditions of the physical system;

(ii) the system operates for a finite or infinite time; and

(iii) output data y are obtained by measuring the observable behaviour of a system at specified times.

The concept can be found in modelling natural systems (e.g., in classical wave mechanics [15, 16, 12, 22, 20, 19]) and technologies for designing machines (e.g., in the 19th century mechanical systems of analogue computers [21, 7, 17, 14, 13, 10]). Experimental computation depends upon a choosing physical system, which may be a part of nature or a machine, and is therefore dependent on specific physical theories that are needed to reason about what can be computed.

2.2 Methodological principles

The theory of algorithms is based on the idea that all data and computations on data can be analysed independently of any form of physical implementation. Specification, computation, reasoning and performance are matters for algebra and logic rather than physical theory. Upon this abstraction, exemplified in the Turing machine, rests our deep understanding of digital computation and its spectacular practical applications. The idea of experimental computation is to attempt to analyse physical models of computation *independently of the theory of algorithms*. Physical theories play a fundamental role in understanding experimental computation, which we have discussed at length elsewhere [3, 4]. To seek conceptual clarity, and mathematical precision and detail, we have proposed, in [3, 4], the following four principles and stages for an investigation of any class of experimental computations:

Principle 1. Defining a physical subtheory: *Define precisely a subtheory T of a physical theory and examine experimental computation by the systems that are valid models of the subtheory T.*

Principle 2. Classifying computers in a physical theory: *Find systems that are models of T that can through experimental computation implement specific algorithms, calculators, computers, universal computers and hyper-computers.*

Principle 3. Mapping the border between computer and hyper-computer in physical theory: *Analyse what properties of the subtheory T are the source of computable and non-computable behaviour and seek necessary and sufficient conditions for the systems to implement precisely the algorithmically computable functions.*

Principle 4. Reviewing and refining the physical theory: *Determine the physical relevance of the systems of interest by reviewing the truth or valid scope of the subtheory. Criticism of the system might require strengthening the subtheory T in different ways, leading to a portfolio of theories and examples.*

To study experimental computation and seek answers to Questions 1 and 2, the key idea is to lay bare all the concepts and technicalities of examples by putting them under a mathematical microscope using the theory T and, furthermore, to look at the computational behaviour of classes of systems that obey the laws of T. Our methodology requires a careful formulation of a physical theory T, which can best be done by axiomatisations, ultimately formalised in a logical language, i.e., a formal specification of a fragment of the physical theory.

2.3 Combining experiments and algorithms

The methodology above is designed to explore computation by physical systems in ways that are independent of the theory of algorithms. However, it is also interesting to consider the interaction between experiments and algorithms and to extend our methodology to answer Question 3 of the Introduction. Two types of interaction can arise naturally: (i) an algorithm may be needed to perform an experiment; (ii) an experiment may be used as a component to boost the performance of an algorithm.

First, it is easy to find examples of experiments for which some algorithmic computation is needed, e.g., experiments in which measurements are substituted into an algebraic formula. In such cases, we must also define precisely what algorithm will be used and reflect on how these calculations could be accomplished physically. Typically, in mechanics, we find algebraic formulae based on the operations of $x + y, -x, x \cdot y, x^{-1}$, extended by $\sqrt{x}, sin(x), cos(x), \ldots$. Suppose that a calculator is to be used, which computes using rational numbers. We may be tempted to add this calculator to the specification of the theory T. However, with Principle 2 in mind, we may ask: Can this calculator be implemented within the physics of the theory T, or some modest extension of T? Equivalently: Is the theory T strong enough to implement the calculator? The same holds for any and all algorithms.

Secondly, and more dramatically, we can consider using a physical system as a component in an algorithm or class of algorithms. In this case, computations involve analogue and digital procedures and some form of *protocol* for exchanging data between them. Such hybrid computing systems have long been used in control theory.

A simple general way to do this is to choose an algorithmic model and incorporate the experiment as an oracle. There are many algorthmic models: Turing machines, register machines, programming languages, equational calculi, ... The advantage of choosing Turing machines is their rich theory of computational complexity.

Suppose we wish to study the complexity of computations by *Turing machines with experimental or analogue oracles.* Given an input string w over the alphabet of the Turing machine, in the course of a finite computation, the machine will generate a finite sequence of oracle queries and receive a finite sequence of oracle answers. Specifically, as the i-th question to the oracle, the machine generates a string that is converted into a rational number x_i and used to set an input parameter p_i to the equipment. The experiment is performed and, after some delay, returns as output a rational number measurement, or observation, y_i, which is converted into a string or state for the Turing machine to process. The Turing machine may pause while interacting with the oracle. As an oracle, the experiment defines a function $y_i = f(x_i)$ or, more generally, a relation $r(x_i, y_i)$ for $i = 1, 2, \ldots$. But, clearly, there are digital-analogue and analogue-digital conversions involved in the protocol for data exchange. In summary:

Principle 5. Combining experiments and algorithms: *Use a physical system as an oracle in a model of algorithmic computation, such as Turing machines. Determine whether the subtheory T, the experimental computation, and the protocol extends the power and efficiency of the algorithmic model.*

3 The scatter machine experiment

Experimenting with scatter machines is exactly as described in [5]; for convenience, we review the SMEs. First, following the methodology, we recall the underlying physical theory.

3.1 Theory specification

The *scatter machine experiment (SME)* is defined using a subtheory T of Newtonian mechanics, comprising of the following laws and assumptions:

1. Point particles obey Newton's laws of motion in the two dimensional plane.
2. Straight line barriers have perfectly elastic reflection of particles, i.e., kinetic energy is conserved exactly in collisions.
3. Barriers are completely rigid and do not deform on impact.
4. Cannons, which can be moved in position, can project a particle with a given velocity in a given direction.
5. Particle detectors are capable of telling if a particle has crossed a given region of the plane.
6. A clock measures time.

In performing the experiment, we will *not* need: (i) absolute precision in measurements, either in space or time, or (ii) calculations with algebraic formulae. The theory T is independent of any notion of computation.

3.2 Specification of scattering machine

The machine consists of a cannon for projecting a point particle, a reflecting barrier in the shape of a wedge and two collecting boxes, as in Figure 1.

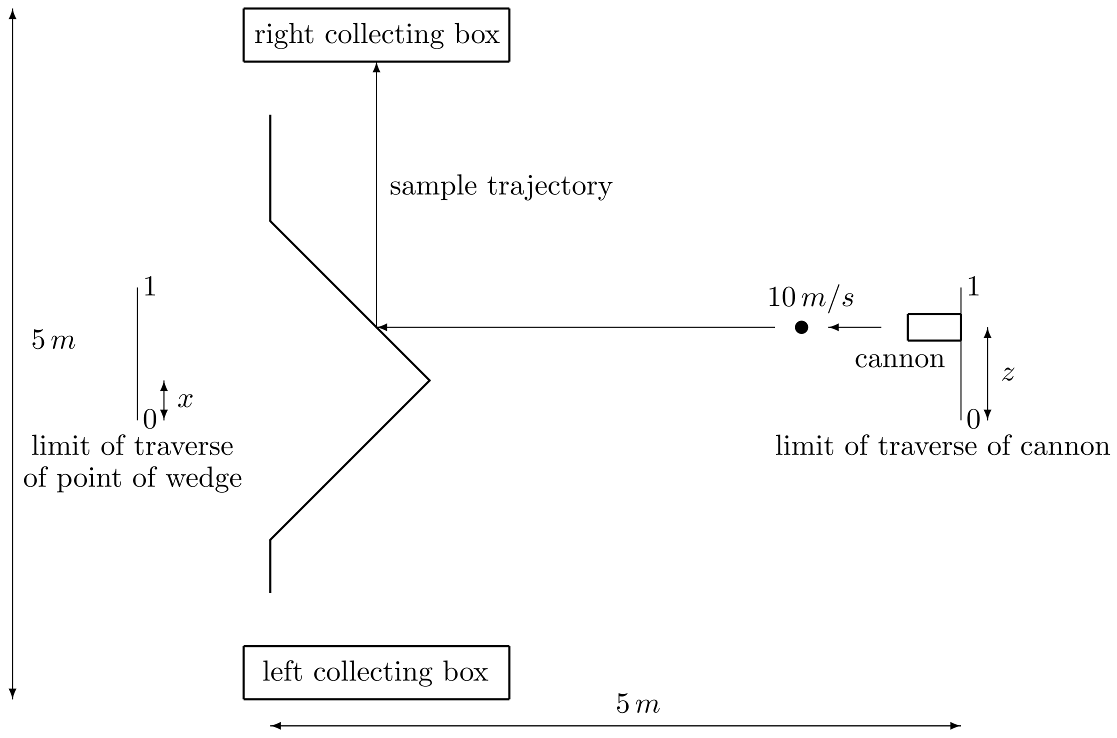

Figure 1: A schematic drawing of the scatter machine

The wedge can be at *any* position. But we will assume it is fixed for the duration of all the experimental work. Under the control of the Turing machine, the cannon will be moved and fired repeatedly to find information about the position of the wedge. Specifically, the way the *SME* is used as an oracle in Turing machine computations, is this: a Turing machine will set a position for the canon as a query and will receive an observation about the result of firing the cannon as a response. For each input to the Turing machine, there will be finitely many runs of the experiment.

The sides of the wedge are at 45° to the line of the cannon, and we take the collision to be perfectly elastic, so the particle is deflected at 90° to the line of the cannon, and hits either the left or right collecting box, depending on whether the cannon is to the left or right of the point of the wedge. Since the initial velocity is 10m/s, the particle will enter one of the two boxes within 1 second of being fired. Any initial velocity $v > 0$ will work with a corresponding waiting time. The wedge is sufficiently wide so that the particle can only hit the 45° sloping sides, given the limit of traverse of the cannon. The wedge is sufficiently rigid so that the particle cannot move the wedge from its position.

What happens if the particle hits the point of the wedge? This is the only exception to the description above. In this case, we shall not make any assumption about what happens, as any assumption would doubtless prove controversial physically. We shall suppose that

Non-determinism Assumption *After each projection, the particle could enter either box, or not enter any box. This means that the scatter machine is non-deterministic.*[1]

3.3 Operation of scattering machine

Suppose that x is the arbitrarily chosen, but fixed, position of the point of the wedge. For a given cannon position z, there are three outcomes of an experiment:

- **(r)** One second after firing, the particle is in the right box.
- **(l)** One second after firing, the particle is in the left box.
- **(o)** One second after firing, the particle is in neither box.

Figure 2 shows how the outcomes (l,r,o) are related to the position of the cannon relative to the wedge.

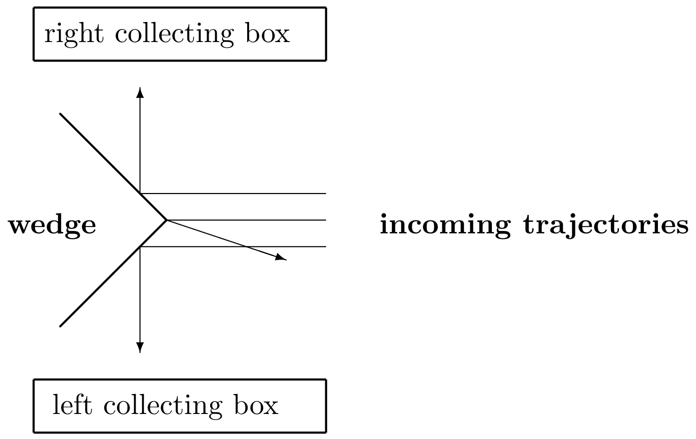

Figure 2: The behaviour of a particle in the scatter machine experiment.

From the mechanics of the system, these outcomes can be connected to the wedge position x and parameter z in the following manner:

One second after firing, the particle is in the right box. Conclusion: $z \geqslant x$.

One second after firing, the particle is in the left box. Conclusion: $z \leqslant x$.

One second after firing, the particle is in neither box. Conclusion: $z = x$.

[1]In a physical sense, not the technical definition of non-determinism in Turing machines.

The *SME* was designed to find x to arbitrary accuracy by altering z, so in our machine $0 \leqslant x \leqslant 1$ will be fixed, and we will perform observations at different values of $0 \leqslant z \leqslant 1$. What values of z are allowed? We take the special case of the *dyadic scatter machine*, where we can set z to be any fraction in the range $0 \leqslant z \leqslant 1$ with denominator a power of 2.

Consider the precision of the experiment. When measuring the output state the situation is simple: either the ball is in one tray, or in the other tray, or in neither. Errors in observation do not arise.

Now consider some of the non-trivial ways in which precision depends on the positions of the cannon. There are different postulates for the precision of the cannon, and we list some in order of decreasing strength:

Definition 3.3.1
The SME is error-free if the position of the cannon can be set exactly to any given dyadic rational number.
The SME is error-prone of arbitrary precision if the position of the cannon can be set to within any arbitrarily small, but non-zero, precision.
The SME is error-prone of fixed precision if the position of the cannon can be set only to within a fixed non-zero precision.

The following is a theorem in [5], which shows that the behaviour of the *SME* is not computable by algorithms:

Theorem 3.3.2 *The error-free dyadic scatter machine can compute the position of any point on a unit line segment to arbitrary accuracy: given any point $x \in [0,1]$ the machine can generate an infinite sequence $[p_n, q_n]$ of rational number interval approximations of x such that for each n, we have*

$$x \in [p_n, q_n] \text{ and } |p_n - q_n| < \tfrac{1}{2^{n-1}}.$$

The entire system is bounded in the following sense:
Space: The system can be contained within an arbitrary shallow box.
Mass: The total mass of the system is bounded.
Time: The calculation of the $n-th$ approximation takes place in linear time $O(n)$.
Energy: The calculation of the $n-th$ approximation uses linear $O(n)$ energy.

This is proved by using an experimental procedure based on bisection. We start with the interval $[0,1]$, and fire the cannon at the end points and the mid point of the interval. This will give which of the two half-intervals $[0, \frac{1}{2}]$ and $[\frac{1}{2}, 1]$ the vertex of the wedge is in. Then repeat using the appropriate half-interval, etc. See [5] for details.

With a little modification, the weaker assumption of an error prone dyadic scatter machine with arbitrary precision would have sufficed. (To do this we would have to use overlapping intervals and a "times 3/4 method", rather than a bisection method.)

4 Turing machines, oracles and non-uniform complexity theory

To prepare for combining experiments and algorithms, we note some points about oracle Turing machines and probabilistic Turing machines. Then we recall briefly some notions of non-uniform complexity, which we use to analyse our new model of computation.

4.1 The oracle Turing machine

We assume the reader is familiar with idea of the deterministic Turing machine and its many definitions, which may vary the finite alphabet Σ of symbols, the control unit with its finitely many states, and the memory with its finitely many tapes. The oracle Turing machine is a device to which is added an *oracle*, which is thought of as a black box that can answer questions. The Turing machine can compute a word in Σ^*, which is interpreted as a question, and to each question the oracle deterministically gives an answer, yes or no. We will assume that an oracle Turing machine has at least three tapes: one or more *work tapes*, where calculations can be carried out; one *query tape*, for the purpose of asking questions to the oracle; and one *input tape* for input.

The control unit governs the computation, by reading and writing symbols in the tapes and by consulting the oracle. Among the states of the finite control unit are six special states. Three of these states are used to begin and halt the computation: these are called the *initial state*, the *accepting state*, and the *rejecting state*. The remaining three states serve to interact with the oracle: these are called the *query state*, the *"yes" state*, and the *"no" state*. In order to ask the oracle a question, the control unit writes the question in the query tape and moves to the query state. The finite control will then be interrupted, and the oracle will answer the question by resuming the computation in either the "yes" state or the "no" state, with the obvious meaning. Then the query tape is erased.

A computation of the oracle Turing machine on a word w begins with w written in the input tape of the machine, with the input tape head placed on the left-most symbol of w, and the control unit in the initial state. The computation will proceed as long as the control unit does not enter either the accepting or rejecting states. The input w is said to be *accepted* by the machine if the computation halts in the accepting state, and *rejected* if it halts in the rejecting state.

Let $A \subseteq \Sigma^*$ be a set of words over Σ. We say that A is decided by an oracle Turing machine $\mathcal{M}$ if, for every input $w \in \Sigma^*$, w is accepted by $\mathcal{M}$ when $w \in A$ and rejected by $\mathcal{M}$ when $w \notin A$. We say that $\mathcal{M}$ decides A in polynomial time, if $\mathcal{M}$ decides A, and, for every $w \in \Sigma^*$, the number of steps of the computation is bounded by a polynomial in the size of w.

To each oracle corresponds a unique set O containing exactly the words to which the oracle answers *yes*. We may thus denote an oracle by its corresponding set O for which the Turing machine may calculate w and decide if $w \in O$. We write $\mathrm{P}(O)$ to stand for the class of sets decidable in polynomial time by machines with the oracle O, and abbreviate $\mathrm{P} = \mathrm{P}(\varnothing)$. An oracle is sparse if, for each size m, the number of words in it of size $\leqslant m$ is bounded by a polynomial.

4.2 The probabilistic Turing machine

A probabilistic Turing machine is similar to the oracle Turing machine, but instead of a black box that answers questions, the finite control uses a *coin*. The probability of the coin turning up *heads* is $p \in [0, 1]$, and the probability of it giving *tails* is $q = 1 - p$. It is common to assume that $p = \frac{1}{2}$, i.e., that the coin is fair. This assumption is vital for the usual understanding of probabilistic machines, since a probabilistic machine with a rational p can be simulated by a deterministic machine, but this is not necessarily true for an arbitrary real p.

The coin can be seen as a non-deterministic oracle which, when queried, will always answer *yes* with probability p, and *no* with probability q. There is no need for the query tape but otherwise the behaviour of the probabilistic machine is like that of the oracle Turing machine. We thus choose to rename the query, "yes" and "no" states to be the *toss state*, the *heads state* and the *tails state*, respectively.

The decision criterion, however, must be different, because unless the control unit makes no use of the coin, the machine will not deterministically accept or reject the input. We will here describe one of the most common probabilistic decision criterion.

For a set $A \subseteq \Sigma^*$, a probabilistic Turing machine $\mathcal{M}$, and an input $w \in \Sigma^*$, the *error probability* of $\mathcal{M}$ for input w is the probability of $\mathcal{M}$ rejecting w if $w \in A$, or the probability of $\mathcal{M}$ accepting w if $w \notin A$. We say that $\mathcal{M}$ decides A with *bounded error probability* if there is a number $\gamma < \frac{1}{2}$, such that the error probability of $\mathcal{M}$ for any input w is smaller than γ.[2] A is decided in polynomial time with bounded error probability if, for every input w, the number of steps in the possible computations is always bounded by a polynomial in the length of w. We write BPP to stand for the class of sets decidable in polynomial time

[2]Under the assumption that $p = \frac{1}{2}$, one can represent every possible computation of a probabilistic Turing machine on an input w as a complete binary tree, where each node of the tree corresponds to the global state, or configuration, of the machine after each possible sequence of coin tosses. Then the probability of error is simply the fraction of leaves of this tree which incorrectly accept or reject w.

with bounded error probability using a balanced coin.

4.3 Non-uniform complexity classes

We will see that non-uniform complexity gives the most adequate characterisations of the computational power of the analog-digital scatter machine.[3] Non-uniform complexity classifies problems by studying families of finite machines (e.g., circuits) $\{\mathcal{C}_n\}_{n\in\mathbb{N}}$, where each $\mathcal{C}_n$ decides the restriction of some problem to inputs of size n. It is called *non-uniform*, because for every $n \neq m$ the finite machines C_n and C_m can be entirely unrelated, while in uniform complexity the algorithm is the same for inputs of every size. A way to connect the two approaches is by means of *advice classes*: one assumes that there is a unique algorithm for inputs of every size, which is aided by certain information, called *advice*, which may vary for inputs of different sizes. The input is given, for a word w, by the concatenation of w and the advice, $\langle w, f(|w|)\rangle$, by means of a pairing function $\langle\cdot,\cdot\rangle : \Sigma^* \times \Sigma^* \to \Sigma^*$ and its inverses $(\cdot)_1$ and $(\cdot)_2$, which are all computable in linear time.

Definition 4.3.1 *Let $\mathcal{B}$ be a class of sets and $\mathcal{F}$ a class of total functions. The non-uniform class $\mathcal{B}/\mathcal{F}$ is the class of sets A for which some $B \in \mathcal{B}$ and some $f \in \mathcal{F}$ are such that, for every w, $w \in A$ if and only if $\langle w, f(|w|)\rangle \in B$.*

$\mathcal{F}$ is called the *advice class* and f is called the *advice function.* Examples for $\mathcal{B}$ are P, or BPP. We will be considering two instances for the class $\mathcal{F}$: *poly* is the class of functions bounded by a polynomial, i.e., *poly* is the class of functions $f : \mathbb{N} \to \Sigma^*$ such that, for some polynomial p, $|f(n)| \in \mathrm{O}(p(n))$; *log* is the class of functions $g : \mathbb{N} \to \Sigma^*$ such that $|g(n)| \in \mathrm{O}(\log(n))$. We will also need to treat prefix non-uniform complexity classes. For these classes we may only use prefix functions, i.e., functions f such that $f(n)$ is always a prefix of $f(n+1)$. The idea behind prefix non-uniform complexity classes is that the advice given for inputs of size n must also be useful to decide smaller inputs.

Definition 4.3.2 *Let $\mathcal{B}$ be a class of sets and $\mathcal{F}$ a class of functions. The prefix non-uniform class $\mathcal{B}/\mathcal{F}*$ is the class of sets A for which some $B \in \mathcal{B}$ and some prefix function $f \in \mathcal{F}$ are such that, for every length n and input w with $|w| \leqslant n$, $w \in A$ if and only if $\langle w, f(n)\rangle \in B$.*

As examples we have P/*log*∗, or BPP/*log*∗. It is a matter of some controversy whether this is the appropriate definition of BPP/$\mathcal{F}$. Notice that by demanding that there is a set $B \in$ BPP, and a function $f \in \mathcal{F}$ (in this order) such that $w \in A$ if and only if $\langle w, f(|w|)\rangle \in B$, we are demanding a fixed $\varepsilon > 0$ (fixed by the Turing machine chosen to

[3]For an interesting definition of how we can compare computational power among different models of computability, see the work of Udi Boker and Nachum Dershowitz [6].

witness that $B \in$ BPP) *for any possible advice*, instead of the more intuitive idea that the error only has to be bounded after choosing the correct advice. This leads to the following definitions.

Definition 4.3.3 BPP//*poly is the class of sets A for which a probabilistic Turing machine $\mathcal{M}$, a function $f \in poly$, and a constant $\gamma < \frac{1}{2}$ exist such that $\mathcal{M}$ rejects $\langle w, f(|w|)\rangle$ with probability at most γ if $w \in A$ and accepts $\langle w, f(|w|)\rangle$ with probability at most γ if $w \notin A$.*

Definition 4.3.4 BPP//*log* is the class of sets A for which a probabilistic Turing machine $\mathcal{M}$, a prefix function $f \in log$, and a constant $\gamma < \frac{1}{2}$ exist such that, for every length n and input w with $|w| \leqslant n$, $\mathcal{M}$ rejects $\langle w, f(n)\rangle$ with probability at most γ if $w \in A$ and accepts $\langle w, f(n)\rangle$ with probability at most γ if $w \notin A$.*

It can be shown that BPP//*poly* = BPP/*poly*, but it is unknown whether BPP//*log*∗ $\subseteq$ BPP/*log*∗. After the work of [2], we can safely assume without loss of generality that, for P/*log*∗ and BPP/*log*∗, the length of any advice $f(n)$ is exactly $\lfloor a \log n + b \rfloor$, for some $a, b \in \mathbb{N}$ which depend on f. Their proof generalises for BPP//*log*∗.

It is important to note that the usual non-uniform complexity classes contain undecidable sets, e.g., P/*poly* contains the halting set. The non-recursive enumerability of the non-uniform complexity classes results exclusively from the non-computability of the advice functions. In fact, for any class $\mathcal{B}$ of decidable sets, and any class $\mathcal{F}$ of advice functions, $\mathcal{B}/(\mathcal{F} \cap \mathrm{PR}) \subseteq \mathrm{REC}$, where PR is the class of partial recursive functions and REC is the class of recursive sets. This means that computable advice results in computable behaviour, but possibly speeded up. [4]

There exists a characterisation of the class P/*poly* that codifies the relationships between obtaining external information from polynomial length advice and obtaining it from oracles. The proof can be found in [1], chapter 5.

Theorem 4.3.5 P/*poly* $= \cup_{O \,\mathrm{sparse}} P(O)$.

One possible oracle that we can associate to an SME is the set of finite prefixes of the vertex position of the wedge written in binary.

5 The analog-digital scatter machine

The Turing machine is connected to the SME in the same way as it would be connected to an oracle: we replace the query state with a *shooting state* (q_s), the "yes" state with a *left*

[4]The reader should note that exponential length advice suffices to decide all sets, because for length n there are at most 2^n words, which can be concatenated to give the advice.

state (q_l), and the "no" state with a *right state* (q_r). The resulting computational device is called the *analog-digital scatter machine*, and we refer to the *vertex position* of an analog-digital scatter machine when we mean to discuss the vertex position of the corresponding SME.

In order to invoke a scatter machine experiment, the analog-digital scatter machine will write a word z in the query tape and enter the shooting state. This word will either be "1", or a binary word beginning with 0. We will use z to denote both a word $z_1 \ldots z_n \in \{1\} \cup \{0s : s \in \{0,1\}^*\}$ and the corresponding dyadic rational

$$\sum_{i=1}^{n} 2^{-i+1} z_i \in [0,1].$$

In this case, we write $|z|$ to denote n, i.e., the size of $z_1 \ldots z_n$, and say that the analog-digital scatter machine is *aiming* at z.

After entering the shooting state, the computation will be interrupted, and the SME will attempt to set the cannon at z. The place where the cannon is actually set at depends on whether the SME is error-free or error-prone. If the SME is error-free, the cannon will be placed exactly at z.

If the SME is error-prone with arbitrary precision, the cannon will be placed somewhere in the interval $[z - \varepsilon, z + \varepsilon]$, where $\varepsilon > 0$ is arbitrarily small. Of course, the interface between the Turing machine and the experimental apparatus needs to be able to specify this accuracy, and the most convenient way is to interpret the length of the word specifying z as giving the accuracy, so that the the cannon will be placed at some point in the interval $[z - 2^{-|z|-1}, z + 2^{-|z|-1}]$. This means that for different words representing the same dyadic rational, the longest word will give the highest precision. To increase the accuracy, it is only necessary to add more zeros to the binary representation of z.

If the SME is error-prone with fixed precision ε then the cannon will be placed somewhere in the interval $[z - \varepsilon, z + \varepsilon]$. In this case we will have to specify information on the probability distribution on the interval $[z - \varepsilon, z + \varepsilon]$, and we will take this to be the uniform probability distribution.

After setting the cannon, the SME will fire a particle, wait one second and then check if the particle is in either box. If the particle is in the right collecting box, then the Turing machine computation will be resumed in the state q_r. If the particle is in left box, *or if it is in neither box*, then the Turing machine computation will be resumed in the state q_l.

With this behaviour, we obtain three distinct analog-digital scatter machines.

Definition 5.0.6 *An* error-free analog-digital scatter machine *is a Turing machine connected to an error-free SME. We define an* error-prone analog-digital scatter machine with arbitrary precision*, and an* error-prone analog-digital scatter machine with fixed precision *similarly.*

Consider the total time to be the sum of the times taken to communicate, initialise and perform the experiment. In the case of the *SME*, conducting an atomic experiment (one firing of the cannon and the subsequent measurement) takes constant time.

Definition 5.0.7 *We say an AD-protocol is $f(n)$-bounded if the total time taken by SME to process a query string of size n and return an answer is bounded by $f(n)$. Polynomial and exponential bounds will be used. In the case of exponential bounds, we will say that a machine has a strictly exponential AD-protocol if there is an exponential upper and lower bound to the time taken.*

The error-free analog-digital scatter machine has a very simple behaviour. If such a machine, with vertex position $x \in [0, 1]$, aims at a dyadic rational $z \in [0, 1]$, we are certain that the computation will be resumed in the state q_l, if $z < x$, and in the state q_r, when $z > x$. If we make the further assumption that x is *not* a dyadic rational, and we will freely make this assumption throughout this text, then the error-free analog-digital scatter machine will behave deterministically. We can thus define the following decision criterion.

Definition 5.0.8 *Let $A \subseteq \Sigma^*$ be a set of words over Σ. We say that an error-free analog-digital scatter machine $\mathcal{M}$ **decides** A if, for every input $w \in \Sigma^*$, w is accepted if $w \in A$ and rejected if $w \notin A$. We say that $\mathcal{M}$ **decides** A **in polynomial time**, if $\mathcal{M}$ decides A, and there is a polynomial p such that, for every input $w \in \Sigma^*$, the number of steps of the computation of $\mathcal{M}$ on w is bounded by $p(|w|)$.* [5]

The error-prone analog-digital scatter machines, however, do not behave in a deterministic way. If such a machine aims the cannon close enough to the vertex position, and with a large enough error, there will be a positive probability for both the particle going left or right. If the error can be made arbitrarily small, then for certain vertex positions we can recover deterministic behaviour by ensuring that the error interval lies entirely on one side of the vertex position. However, in general, a deterministic decision criterion is not suitable for error prone machines. For a set $A \subseteq \Sigma^*$, an error-prone analog-digital scatter machine $\mathcal{M}$, and an input $w \in \Sigma^*$, let the *error probability* of $\mathcal{M}$ for input w be either the probability of $\mathcal{M}$ rejecting w, if $w \in A$, or the probability of $\mathcal{M}$ accepting w, if $w \notin A$.

Definition 5.0.9 *Let $A \subseteq \Sigma^*$ be a set of words over Σ. We say that an error-prone analog-digital scatter machine $\mathcal{M}$ **decides** A if there is a number $\gamma < \frac{1}{2}$, such that the error probability of $\mathcal{M}$ for any input w is smaller than γ. We call **correct** to those computations which correctly accept or reject the input. We say that $\mathcal{M}$ **decides** A **in polynomial time**, if $\mathcal{M}$ decides A, and there is a polynomial p such that, for every input $w \in \Sigma^*$, the number of steps in every correct computation of $\mathcal{M}$ on w is bounded by $p(|w|)$.*

[5]Note that as scatter machines can decide all sets, the interest here is the characterisation of the time constraints.

Standard proof techniques of structural complexity can be used to show that if there is an error-prone analog-digital scatter machine $\mathcal{M}$ which decides A in polynomial time with an error probability bounded by $\gamma < \frac{1}{2}$, then, for any polynomial q, there is another error-prone machine which decides A in polynomial time with an error probability, for inputs of size n, bounded by $\frac{1}{2^{q(n)}}$. We may also assume, without loss of generality, that if an error-prone analog-digital scatter machine $\mathcal{M}$ decides a set A in polynomial time, then all of its computations halt after the same number of steps [1].

6 AD scatter machines compute P/*poly* and P/*log*∗

In the following subsections we will study lower bounds for the computational power of the analog-digital scatter machine under different assumptions on its behaviour.

6.1 Coding the advice function as a wedge position

Given an advice function $f : \mathbb{N} \to \Sigma^*$, we place the wedge at the position given by the binary expansion

$$x(f) \;=\; 0 \cdot c(f(0))\,001\,c(f(1))\,001\,c(f(2))\,001\,c(f(3))\,001\;\ldots\;\in\;[0,1]\;.$$

To determine the coding $c(f(m))$ of $f(m)$ we first code the word $f(m)$ as a binary number, using a binary code for every letter in its alphabet Σ (e.g. Unicode or ASCII). For example we might suppose that the binary form of $f(1)$ was

$$00101\;.$$

To find $c(f(m))$, replace every 0 in its binary form by 100 and every 1 by 010. Thus our example becomes

$$100100010100010\;.$$

To find each $f(m)$ from $x(f)$, read the binary digits of $x(f)$ in triples. When the triple 001 is found, it means that we start reading the next $c(f(m))$, as the triple 001 never occurs as a triple in $c(f(m))$ (i.e. we use 001 as a separator). It is important to note that the length of $c(f(m))$ is a linear function of the length of $f(m)$.

Of course, not every element of $[0,1]$ occurs as an $x(f)$. The reader may note that no dyadic rational can occur, as they would have to end in $0000\ldots$ or $1111\ldots$, and the triples 000 and 111 do not occur in any $x(f)$. This is good news, in that we cannot have any ambiguity in the binary expansion. For example, $\frac{1}{2}$ has the binary expansions $0\cdot1000\ldots$ and $0\cdot0111\ldots$, but this sort of ambiguity is limited to dyadic rationals. Later in the paper we will need a rather stronger result about which numbers can't be of the form $x(f)$:

Lemma 6.1.1 *Given a dyadic rational $k/2^{3n} \in [0,1]$ for integer k, $|x(f) - k/2^{3n}| > 1/2^{3n+3}$ for all f.*

Proof: To prove this it is only necessary to note that whatever the first $3n$ binary digits of $x(f)$, the next triple is of the form 100 or 010 or 001. $\Box$

Corollary 6.1.2 *Given a dyadic rational $k/2^n \in [0,1]$ for integer k, $|x(f) - k/2^n| > 1/2^{n+5}$ for all f.*

Proof: Consider the cases $n = 3m$, $n = 3m + 1$ and $n = 3m + 2$ separately and use lemma 6.1.1. $\Box$

As we shall see, the purpose of these results is to ensure that various error intervals for the cannon lie entirely on one side of the wedge position.

6.2 AD-scatter machines with a polynomial AD-protocol can decide P/*poly* in polynomial time.

In this subsection we will show that both error free and error prone arbitrary precision AD-scatter machines with a polynomial AD-protocol can decide sets in P/*poly* in polynomial time. We shall leave the more complicated problem of determining an upper bound for the complexity of the sets that these machines can decide until later.

Let A be a set in P/*poly*, and, by definition, let $B \in \mathrm{P}$, $f \in \mathit{poly}$ be such that

$$w \in A \iff \langle w, f(|w|)\rangle \in B.$$

Now take an AD-scatter machine with the position of the wedge set to $x(f)$, as described in subsection 6.1. For a word w, if we can determine the advice $f(|w|)$ in polynomial time in $|w|$, then the Turing machine can determine $\langle w, f(|w|)\rangle$ in polynomial time, so the problem is solved in polynomial time. In turn, to determine $f(|w|)$ it suffices to show that we can read the first n binary places of the wedge position $x(f)$ in polynomial time in n.

Proposition 6.2.1 *An error free AD-scatter machine with a polynomial AD-protocol can determine the first n binary places of the wedge position $x(f)$ in polynomial time in n.*

Proof: Use the bisection procedure already outlined, which requires n steps, with step m requiring an amount of time polynomial in m. Multiplying this gives a total amount of time polynomial in n. The only way in which the bisection procedure could go wrong is if the wedge is at a dyadic rational, and this cannot be the case for $x(f)$, as noted in subsection 6.1. $\Box$

Now we come to the error prone arbitrary precision case, which is solved in almost exactly the same way. The work lies in choosing the errors so that the same process actually works, and for that we need corollary 6.1.2.

Proposition 6.2.2 *An error prone arbitrary precision AD-scatter machine with a polynomial AD-protocol can determine the first n binary places of the wedge position $x(f)$ in polynomial time in n.*

Proof: Use the bisection process again. At the mth stage in the bisection process (involving dyadic rationals with denominators 2^m), we set the error in the desired cannon position $k/2^m$ to be $1/2^{m+5}$. By corollary 6.1.2 the wedge position cannot be in the error interval about the given dyadic rational, and thus the result of the experiment is the same as though the cannon was at the given dyadic rational $k/2^m$. The cost in doing this is polynomial in m, and if we sum to $m = n$ we get the total time polynomial in n. $\square$

Theorem 6.2.3 *An AD-scatter machine (either error free or error prone arbitrary precision) with a polynomial AD-protocol can decide* P*/poly in polynomial time.*

Proof: From the discussion earlier in this subsection, and propositions 6.2.1 and 6.2.2. $\square$

6.3 AD-scatter machines with an exponential AD-protocol can decide P/*log∗* in polynomial time.

In this subsection we will show that both error free and error prone arbitrary precision AD-scatter machines with an exponential AD-protocol can decide sets in P/*log∗* in polynomial time.

First we must decide on a coding for log ∗ advice, where the reader is reminded that the advice $f(n)$ can be used to decide all words w with $|w| \leqslant n$. We can recursively define $y(f)$ as the limit sequence of $y(f)(n)$, using the coding c given in subsection 6.1 by starting with

$$y(f)(0) \;=\; 0 \cdot c(f(0))$$

and using the following cases, where $f(n+1) = f(n)s$:

$$y(f)(n+1) \;=\; \begin{cases} y(f)(n)\, c(s) & \text{if } n+1 \text{ not a power of } 2 \\ y(f)(n)\, c(s)\, 001 & \text{if } n+1 \text{ a power of } 2 \end{cases}$$

The net effect is to add a 001 at the end of the codes for $n = 1, 2, 4, 8 \ldots$.

Now take w with $2^{m-1} < |w| \leqslant 2^m$. We read the binary expansion until we have counted a total of $m+1$ triples 001. Now the extra 001s are deleted, and we can reconstruct $f(2^m)$, which can be used as advice for w. But how many binary digits from $y(f)$ must we read in order to reconstruct $f(2^m)$? We start with $|c(f(n))| \leqslant L\ \log_2(n) + K$ (for some constants K and L) from the definition of log length. This means that we must read at most $L\,m + K + 3\,m$ digits when we add in the separators, so the number of digits is logarithmic in $|w|$. Using the bisection method we need to make $O(\log(|w|))$ experiments, and there is a maximum length to the binary form of the dyadic rationals for these experiments which is also $O(\log(|w|))$. Then the total experimental time is $O(\log(|w|))$ times an exponential of $O(\log(|w|))$, which is polynomial.

Proposition 6.3.1 *An AD-scatter machine (either error free or error prone arbitrary precision) with an exponential AD-protocol can decide* P$/$*log*$*$ *in polynomial time.*

Proof: The wedge position is $y(f)$ (as just described) for the advice function f. We only have to revise the statement of the propositions 6.2.1 and 6.2.2 to say that they can read $\log(m)$ digits in time polynomial in m. $\Box$

7 An upper bound for the computational power of error-free machines

7.1 What do deterministic AD error free scatter machines compute?

We have seen that AD error free scatter machines can solve any problem in P$/$*poly* (for polynomial protocols) or in P$/$*log*$*$ (for exponential protocols) in polynomial time. Now we may ask the converse, if a set is decidable by an AD error free scatter machine in polynomial time, what complexity class does it belong to? One difficulty here is what happens if, during the calculation, the projectile hits the edge of the wedge? In section 6 the codings were constructed to produce wedge positions that were not dyadic rationals, so this problem did not arise.

Let us begin by considering a deterministic machine. This means that the same result is always given if the projectile hits the edge of the wedge, there is no probability necessary. If at most n binary places can have been written in the query tape describing the cannon position, then the cannon position must be set to a position $m/2^n$ for some integer $0 \leqslant m \leqslant 2^n$. In this case it might be thought that the first n binary places of the wedge position x would suffice to simulate the scatter machine, but this is not necessarily true. (We always use the terminating form of the binary expansion, i.e. $0{\cdot}1$ not $0{\cdot}0111\ldots$.) For example, suppose that $n = 3$ and we have $x = 0{\cdot}010$ to 3 places (no rounding!). For any position $m/2^3 \neq 0{\cdot}010$ we have determined the scattering, but what about $m/2^3 = 0{\cdot}010$? This depends on how the binary expansion of x continues. If $x > 0{\cdot}010$ (for example $x = 0{\cdot}010000100\ldots$) then the scattering is determined, but if $x = 0{\cdot}010$ then the particle has hit the vertex of the wedge, and something different may occur. The wedge position x could be coded $010{=}$ (meaning $x = 0{\cdot}010$) or $010{>}$ (meaning $x > 0{\cdot}010$).

Now we can code up all the binary expansion of x into an infinite word $q(x)$. After each power of two binary places we append either an $=$ or a $>$, depending on whether x is equal to its expansion truncated to that number of places or not. For example we could have the following:

$$0\cdot0{>}0{>}00{>}0010{>}00001100\ldots$$
$$0\cdot0{>}0{>}00{>}0010{=}00000000\ldots$$

but not these,

$$0\cdot0{>}0{>}01{=}0000{>}00000000\ldots$$
$$0\cdot0{>}0{>}00{>}0010{=}00010000\ldots$$

as once we have equality we must have all zeros and all equalities. To get the value of x, just strip out all the $=$'s and $>$'s. Thus the allowed codes above give the numbers $0{\cdot}0000001000001100\ldots$ (with more digits to follow) and $0{\cdot}0000001$ exactly.

Let O be the (sparse) set of finite prefixes of $q(x)$.

Proposition 7.1.1 *Suppose that $\mathcal{M}$ is an error free AD scatter machine, with a polynomial AD-protocol and deterministic scattering at the wedge position. Suppose that $\mathcal{M}$ decides a subset $A \subseteq \Sigma^*$ in polynomial time. Then A is in* P/*poly.*

Proof: By definition 5.0.8, there is a polynomial p such that, for every $w \in \Sigma^*$, the number of steps of the computation is bounded by $p(|w|)$. Then at most $p(|w|)$ symbols will have been written on the query tape. If we take the least n so that $2^n \geqslant p(|w|)$, then the prefix of maximum length we need during the computation has length 2^n+n+1, which is polynomial in $|w|$. The scatter machine $\mathcal{M}$ deciding A can be modified by substituting all calls for the SME by calls to the oracle O. Applying theorem 4.3.5 we get A in P/*poly*. $\square$

Theorem 7.1.2 *The subsets of Σ^* which are computable in polynomial time by deterministic error free AD scatter machines with a polynomial AD-protocol is exactly* P/*poly.*

Proof: Immediate from propositions 7.1.1 and 6.2.3. $\square$

Proposition 7.1.3 *Suppose that $\mathcal{M}$ is an error free AD scatter machine, with a strictly exponential AD-protocol and deterministic scattering at the wedge position. Suppose that $\mathcal{M}$ decides a subset $A \subseteq \Sigma^*$ in polynomial time. Then A is in* P/*log∗.*

Proof: By definition 5.0.8, there is a polynomial p such that, for every $w \in \Sigma^*$, the number of steps of the computation is bounded by $p(|w|)$. Then the maximum number of symbols which have been written on the query tape and used to set the scatter machine is logarithmic in $|w|$, say $\leqslant A\ \log_2(|w|) + B$ (this is where we use *strictly* exponential). If we take the least n so that $2^n \geqslant A\ \log_2(|w|) + B$, then the prefix of maximum length we need during the computation has length $2^n + n + 1$ (which is logarithmic in $|w|$). The proof follows the proof of 7.1.1, but paying attention to the fact that we only need to consult words of logarithmic size. $\square$

Theorem 7.1.4 *The subsets of Σ^* which are computable in polynomial time by deterministic error free AD scatter machines with a strictly exponential AD-protocol is exactly* $\mathrm{P}/log*$.

Proof: Immediate from propositions 7.1.3 and 6.3.1. $\square$

7.2 What do non-deterministic AD error free scatter machines compute?

Now we come to a nondeterministic scatter machine. Here we must make some assumptions – there must be some description printed on the box of the AD scatter machine – in order to make sensible proofs. The simplest is that the wedge position is not a dyadic rational:

Theorem 7.2.1 *The subsets of Σ^* which are computable in polynomial time by non-deterministic error free AD scatter machines which are known to have non-dyadic rational wedge position with a polynomial AD-protocol is exactly* $\mathrm{P}/poly$.

Proof: Almost immediate from propositions 7.1.1 and 6.2.3. For 6.2.3, we note that the required wedge position is non-dyadic. For 7.1.1, if we know that the wedge position is non-dyadic, then the behaviour is exactly that of a corresponding deterministic machine, as the cannon can never be set to the wedge position. $\square$

If we are told nothing about the wedge position for the machine, there is a procedure which will determine some information: Take your favourite computable function g from $\mathbb{N}$ to the dyadic rationals in $[0,1]$ which takes every value infinitely many times. The cannon is fired successively at the positions $g(n)$. At the first time that the result is different for $n \neq m$ where $g(n) = g(m)$, report 'the machine is non-deterministic with wedge position $g(n)$'. If the machine is non-deterministic with a dyadic rational wedge position, then the program will terminate with probability 1. Otherwise it will run indefinitely. There is no bound on the run time for the terminating cases.

Once a dyadic rational wedge position is known, we can perform sequential cannon firings at that point to try to find information about the probability of scattering right and left. For simplicity (and because it is not at all obvious what else to do) we assume that the results of sequential firings of the cannon at that position produce *independent* results. That is, the probability of going left or right at the nth firing is completely independent of the results of all the previous firings. Then we have an independent random generator, with probability q of giving result r and $1-q$ of giving result l.

Proposition 7.2.2 *The computational power of an error free AD scatter machine with known dyadic wedge position and independent scattering right from the edge of the wedge with probability q is the same as that of a Turing machine with an independent random number oracle with probability q. [Note that we use a probabilistic decision criterion here.]*

Proof: The position itself can be described by a finite string, and therefore could be hard-wired into a modified Turing machine. The only thing we are left with is the independent random generator. $\square$

The computational power of such an oracle depends on q. For $0 < q < 1$ and given $\gamma \in [0,1)$, it is possible to find a sequence converging with probability $\geqslant \gamma$ to q. This happens even if the real number q is not computable. If $q = 1/2$, then in polynomial time the AD scatter machine can compute exactly BPP (assuming an appropriate decision criterion). In fact, for any $q \in (0,1)$ we can compute BPP, by simulating a fair (probability 1/2) coin toss by using the following lemma:

Lemma 7.2.3 *Take a biased coin (probability of heads $q \in (\delta, 1-\delta)$ for some $\frac{1}{2} > \delta > 0$), and $\gamma \in (0,1)$. Then, up to probability $\geqslant \gamma$, there is a sequence of independent coin tosses of length*

$$\frac{n}{\delta(1-\delta)}\Big(1+\frac{1}{\sqrt{1-\gamma}}\Big)$$

which gives a sequence of independent fair coin tosses of length n.

Proof: The method to simulate one fair toss is to toss the biased coin twice, and:
1) if the result is HT, then output H as a fair toss.
2) if the result is TH, then output T as a fair toss.
3) if the result is HH or TT, then toss the coin twice again and continue.
The probability of this process halting in one step is $r = 2\,q\,(1-q)$, and the probability of having to toss again is $s = 1-r$. This process is repeated until cases (1) or (2) occur, and then the whole method is repeated n times. The total number of tosses T_n for this to happen is given by the negative binomial distribution, with mean and variance

$$\mu = \frac{2\,n\,s}{r} + 2\,n = \frac{2\,n}{r}\ , \quad \nu = \frac{4\,n\,s}{r^2}\ .$$

[Note: if you look in a text book for the mean, you will not find the $+2\,n$ part, this is included as we need to include the time for the successful tosses, not just the standard waiting time until success.] Now we use Chebyshev's inequality in the form

$$P[|T_n - \mu| \geqslant t] \ \leqslant \ \nu.t^{-2} \ .$$

If we put $t = \eta\, n$ and substitute the value of the variance, then

$$P[|T_n - \mu| \geqslant \eta\, n] \ \leqslant \ \frac{4\, s}{r^2\, n\, \eta^2} \ .$$

The worst case for the probability of failure is then the $n = 1$ case, and we get

$$P[T_n \geqslant \mu + \eta\, n] \ \leqslant \ \frac{4\, s}{r^2\, \eta^2} \ \leqslant \ \frac{4}{r^2\, \eta^2} \ .$$

To get the probability of 'failure', i.e. $T_n \geqslant \mu + \eta\, n$ less than $1 - \gamma$, we need

$$\eta \ \geqslant \ \frac{2}{r\sqrt{1-\gamma}} \ ,$$

and then using $r \geqslant 2\, \delta\, (1 - \delta)$ gives the worst case, and the answer. $\square$

8 Reducing the precision of the cannon

We will now study the class of sets decidable in polynomial time by an error-prone analog-digital scatter machine with fixed precision. We will show that such machines may, in polynomial time, make probabilistic guesses of up to a logarithmic number of digits of the position of the vertex. We will then conclude that these machines can decide BPP$//log*$. The reader is reminded that we assume that there is a fixed error $\varepsilon > 0$, and that if the cannon is instructed to fire at position x, then it actually fires in the interval $[x - \varepsilon, x + \varepsilon]$ with a uniform probability distribution. Further we assume that each time the cannon is set to a position, the corresponding random cannon position is independent of previous positions. First we consider some more probability:

We employ the coding of 6.3 for $\log *$ advice, and call the resulting number $r \in (0, 1)$. To determine the first k places of r, we need to determine the value of r to within 2^{-k-5}. This is because the distance between r and any dyadic rational with denominator 2^k is at least 2^{-k-5}. Now suppose that ε is the error when positioning the cannon. We then set the vertex of our analog-digital scatter machine at the position $x = \frac{1}{2} - \varepsilon + 2r\varepsilon$. Our method for guessing digits of r begins by commanding the cannon to shoot from the point $\frac{1}{2}$ a number z of times. If the scatter machine experiment is carried out by entering the shooting state after having written the word 01 in the query tape, the cannon will be placed at some point in the interval $[\frac{1}{2} - \varepsilon, \frac{1}{2} + \varepsilon]$, with a uniform distribution. Then we conclude that the particle will go left with a probability of r and go right with a probability of $1 - r$.

After shooting the cannon z times in this way, we count the number of times that the particle went left, which we denote by L. The value $\tilde{r} = \frac{L}{z}$ will be our estimation of r. In order for our estimation to be correct in the sufficient number of digits, it is required that $|r - \tilde{r}| \leqslant 2^{-k-5}$. By shooting the cannon z times, we have made z Bernoulli trials. Thus L is a random variable with expected value $\mu = zr$ and variance $\nu = zr(1-r)$. By Chebyshev's inequality, we conclude that, for every Δ,

$$\mathbb{P}(|L-\mu| > \Delta) = \mathbb{P}(|z\tilde{r} - zr| > \Delta) = \mathbb{P}\left(|\tilde{r} - r| > \frac{\Delta}{z}\right) \leqslant \frac{\nu}{\Delta^2}.$$

Choosing $\Delta = z\ 2^{-k-5}$, we get

$$\mathbb{P}(|\tilde{r} - r| > 2^{-k-5}) \leqslant \frac{r(1-r)2^{2k+10}}{z} \leqslant \frac{2^{2k+10}}{z}\ .$$

Thus if we allow a probability $\leqslant \gamma_k$ of making an error in determining the first k digits of r, we need a number z_k of cannon firings given by (up to rounding..)

$$z_k \ = \ \frac{2^{2k+10}}{\gamma_k}\ .$$

Our problem is now simple – how big does k have to be for a word of size n? We know that we require $a \log_2 n + b$ digits, but we do not *a priori* know a and b. We need to keep reading the digits of r until we have encountered the appropriate number of 001 markers to see that we have all of $f(n)$. So we fire the cannon z_k times, ask if we have enough 001 markers in the accurate k digits, and if not we fire the cannon z_{k+1} times, and so on. The problem is that if we make a mistake at any stage, we can prematurely terminate the process with the wrong answer. If the probability of making a mistake is the same for every k, it is very likely that we will make a mistake before we terminate for the correct reason. For this reason we set $\gamma_k = \gamma/2^{k+2}$, giving

$$z_k \ = \ \frac{2^{3k+12}}{\gamma}\ .$$

Now we implement the algorithm:

1) Set $k = 1$.
2) Fire the cannon z_k times.
3) Check to see if we have found all of $f(n)$ in the first k digits.
4) If not, increment k and go to (2).

The probability of making a mistake (i.e. incorrectly stopping at (3)) is now less than $\gamma/2$. Given that we dont make a mistake, the first time that k is large enough for enough 001s to be contained we will identify this with probability $\gamma/2^{k+2}$, so overall we have a probability of not correctly identifying $f(n)$ at the first possible stage of less than $1-\gamma$.

But how much time does this take? The total number of firings is

$$\frac{2^{12}}{\gamma} \sum_{k=1}^{a\log_2 n+b} 2^{3k} \leqslant \frac{2^{15+3a\log_2 n+3b}}{\gamma} = \frac{2^{15+3b}\, n^{3a}}{\gamma} ,$$

which is polynomial in n.

Theorem 8.0.4 *An error-prone analog-digital scatter machine with fixed precision can decide any set in* BPP//*log*∗ *in polynomial time.*

9 Examining the results

We have introduced a class of machines in which physical and symbolic processes are combined: our analogue-digital Turing machine combines experimental and algorithmic procedures in a simple way, by thinking of the physical system as some sort of oracle to the Turing machine. We have considered the question: *How do these analogue-digital Turing machines compare with the Turing machines we know?.*

To begin to answer this question we chose a particular experiment, the scatter machine experiment *SME*, which had earlier been studied from the point of view of computation [5]. Here we extended the original SME to allow for both variable and fixed errors in its operating procedures. We analysed a portfolio of analogue-digital Turing machine based on these SME experiments, using non-uniform complexity theory, which is suited to studying computations with oracles. Let us reflect on the results.

The original SMEs are what we call the error-free machines. Here we find that for the deterministic case, error-free analog-digital scatter machines with the vertex at x compute the same as Turing machines with the set $[0, x] \cap D$ as an oracle (here D is the dyadic rationals). For the non-deterministic case, where the wedge produces random scattering, we found similar results for any non-dyadic vertex position $x \in [0, 1] - D$.

However, in the analysis of the new error-prone SMEs, there are two cases: (i) the SME can be set to an arbitrary error presented by the Turing machine, and (ii) the SME can be set to a fixed error ε. Here the connection with familiar oracle Turing machines breaks down. In case (i) we note that no physical assumptions about the SME are required other that the arbitrary accuracy assumption. In case (ii), however, we assumed a uniform probability distribution, and the independence of trials, for the positioning of the cannon within the fixed error margin
$[d - \varepsilon, d + \varepsilon]$.

The theorems show that each variant of the analogue-digital scatter machine has some hypercomputational power. For instance, if K is a set of codes for the halting problem then $\{0^n : n \in K\}$ can be decided by an error-free analog-digital scatter machine, or by an error-prone analog-digital scatter machine with arbitrary precision. Of course, the hypercomputational power of the analog-digital scatter machine arises from the nature of the vertex position x which is a matter for subtle physical analysis of the kind found in [5]. But whenever the vertex position is a computable real number, then the analog-digital scatter machine can compute no more than the Turing machine.

We assumed that we have a sharp wedge with respect to point particles, whose vertex position is measured by a precise point x. The existence of an arbitrarily sharp wedge contradicts atomic theory, and so it is easy to find reasons to object to the scatter machine as a counter-example to the commonly formulated physical Church-Turing theses. The relevance of the analog-digital scatter machine as a model of computation is that it allows one to think with *great mathematical precision and detail* about how computations can be made by physical systems which is one of the central motivations of our methodology and programme.

The work here addesses directly some aspects of the role of oracles in dealing with hypercomputation. Oracles are so versatile it would be surprising if the computability and complexity theories of machines with oracles could not be adapted to handle whatever computational behaviours we find in physical theories. Seen from established work with oracle Turing machines, one can interpret this work as developing carefully controlled *Gedankenexperimente* to implement certain oracles that arise from thinking about the physical world. From this point of view the technical question is: what can we compute if we have *the possibility of measuring an answer to the predicate $y \leqslant x$, for an* ARBITARY, FIXED *real value x and a* GIVEN *dyadic rational y.* We could have replaced the barriers, particles, cannons and particle detectors with any other physical system with this behaviour. So for the Turing machines the scatter machine becomes a tool to answer the more general question about *measurement*:

If we have a physical system to measure an answer to a predicate (such as $y \leqslant x$), to what extent can we use this system in feasible computations?

The error or difficulty that such a measurement entails is represented in the scatter machine experiment by making assumptions on the placement of the cannon. We have given an answer to this question for the case when there is no error and for two simplistic models of error, and studied the case when more precise measurements require a high amount of some resource. In each case, a non-computable x will result in hypercomputational properties.

Using the scatter machine in this metaphorical sense, it is very interesting to notice that a system to measure the answer to the predicate $y \leqslant x$ is very limited in its computational power, unless the precision of measurement can be made arbitrarily high — such as in Section 6. Without this proviso, the access to the information of x becomes difficult, or extremely slow. Even with arbitrarily small error, it does not appear that such a system can be feasibly used, e.g., to answer the halting problem or to solve NP-complete problems. In the first case, it would take an unfeasible number of $O(2^n)$ experiments to obtain an answer for the halting problem on inputs of size n, and for the second case, it is unlikely that NP is contained in P/*poly*, because this would imply the collapse of the polynomial hierarchy [11]. Thus, if we accept that "measuring a physical constant x" means "answering the predicate $y \leqslant x$" as above, our results can be seen as a critical analysis of the claim that hypercomputation could be achieved by measuring a presumably uncomputable physical constant (as suggested in [8], and severely criticised in [9]).

This is the sort of insight which we hope to achieve with our methodology for studying experimental computation.

Acknowledgements: The first and last named authors would like to thank the EPSRC for their support via grant EP/C525361/1.

References

[1] José Luis Balcázar, Josep Días, and Joaquim Gabarró. *Structural Complexity I.* Springer-Verlag, 1988.

[2] José Luis Balcázar and Montserrat Hermo. The structure of logarithmic advice complexity classes. *Theoretical Computer Science*, 207(1):217–244, 1998.

[3] Edwin Beggs and John Tucker. Embedding infinitely parallel computation in newtonian kinematics. *Applied Mathematics and Computation*, 178(1):25–43, 2006.

[4] Edwin Beggs and John Tucker. Can Newtonian systems, bounded in space, time, mass and energy compute all functions? *Theoretical Computer Science*, 371(1):4–19, 2007.

[5] Edwin Beggs and John Tucker. Experimental computation of real numbers by Newtonian machines. *Proceedings of the Royal Society*, 463(2082):1541–1561, 2007.

[6] Udi Boker and Nachum Dershowitz. How to compare the power of computational models. In Barry Cooper, Benedikt Löwe, and Leen Torenvliet, editors, *Computability in Europe 2005: New computational paradigms*, volume 3526 of *Lecture Notes in Computer Science*, pages 54–64. Springer–Verlag, June 2005.

[7] Vannevar Bush and Harold Hazen. The differential analyser. *Journal of the Franklin Institute*, 212(4):447–488, October 1931.

[8] Jack Copeland and Diane Proudfoot. Alan Turing's forgotten ideas in computer science. *Scientific American*, 280:99–103, 1999.

[9] Martin Davis. The myth of hypercomputation. In Christof Teuscher, editor, *Alan Turing: the life and legacy of a great thinker*, pages 195–212. Springer, 2006.

[10] Daniel Graça and José Félix Costa. Analog computers and recursive functions over the reals. *Journal of Complexity*, 19(5):644–664, October 2003.

[11] Richard Karp and Richard Lipton. Some connections between nonuniform and uniform complexity classes. In *STOC '80: Proceedings of the twelfth annual ACM symposium on Theory of computing*, pages 302–309. ACM Press, 1980.

[12] Georg Kreisel. Review of Pour-El and Richards. *Journal of Symbolic Logic*, 47(4):900–902, 1974.

[13] Leonard Lipshitz and Lee Rubel. A differentially algebraic replacement theorem. *Proceedings of the American Mathematical Society*, 99(2):367–372, 1987.

[14] Marian Pour-El. Abstract computability and its relations to the general purpose analog computer. *Transactions of the American Mathematical Society*, 199:1–28, 1974.

[15] Marian Pour-El and Ian Richards. The wave equation with computable initial data such that its unique solution is not computable. *Advances in Mathematics*, 39(4):215–239, 1981.

[16] Marian Pour-El and Ian Richards. *Computability in Analysis and Physics.* Springer-Verlag, 1989.

[17] Claude Shannon. Mathematical theory of the differential analyser. *Journal Mathematical Physics*, 20:337–354, 1941.

[18] Hava Siegelmann. *Neural Networks and Analog Computation: Beyond the Turing Limit.* Birkhäuser, Cambridge, MA, USA, 1999.

[19] Warren Smith. On the uncomputability of hydrodynamics. Published online at http://math.temple.edu/wds/homepage/works.html, 2001.

[20] Viggo Stoltenberg-Hansen and John Tucker. Computable and continuous partial homomorphisms on metric partial algebras. *Bulletin for Symbolic Logic*, 9(3):299–334, 2003.

[21] William Thomson and Peter Tate. *Treatise on Natural Philosophy*, pages 479–508. Cambridge University Press, second edition edition, 1880.

[22] Klaus Weihrauch and Ning Zhong. Is wave propagation computable or can wave computers beat the Turing machine? *Proceedings of the London Mathematical Society*, 85(2):312–332, 2002.

[23] Andrew Chi-Chih Yao. Classical physics and the Church–Turing thesis. *Journal of the ACM*, 50(1):100–105, 2003.