Available online at www.sciencedirect.com

SCIENCE DIRECT®

Applied Mathematics and Computation 178 (2006) 93–102

APPLIED MATHEMATICS AND COMPUTATION

www.elsevier.com/locate/amc

The Church–Turing thesis: Still valid after all these years?

Antony Galton

School of Engineering, Computer Science and Mathematics, University of Exeter, Exeter EX4 4QF, United Kingdom

Abstract

This paper discusses whether recent proposals for so-called hypercomputation would, if realised in practice, invalidate the Church–Turing thesis.
© 2005 Elsevier Inc. All rights reserved.

Keywords: Turing machine; Church–Turing thesis; Hypercomputation

1. What is the Church–Turing thesis?

In 1936, the English mathematician Alan Turing published a ground-breaking paper entitled "On computable numbers, with an application to the Entscheidungsproblem" [1]. In this paper, Turing introduced the notion of an abstract model of computation as an idealisation of the practices and capabilities of a human computer, that is, a person who follows a precisely laid down and reliable procedure to derive numerical values (i.e., outputs) bearing a specified relation to the initial data (inputs). This abstract model is what has since come to be known as the Turing machine model of computation.

A Turing machine is a mechanism that performs calculations by reading and writing symbols on an unbounded linear tape divided into discrete squares. The machine has a 'head' which is able to scan one square of the tape at a time; it can read the symbol inscribed on the square and, if appropriate, replace it by another one (for this purpose, an empty square is treated as if it has a special 'blank' symbol inscribed in it). For a given machine, there is a fixed finite alphabet A of available symbols that may be inscribed on the tape. In addition the machine itself is capable of assuming any of some fixed finite repertoire Q of internal 'states', including both a 'start state' q_0 and a 'halt state' h. The action of the machine can be expressed by means of a listing of quintuples of the form (q, a, q', a', d), indicating that if the machine is in state $q \in Q$ scanning symbol $a \in A$, then it changes state to $q' \in Q$, inscribes symbol $a' \in A$ in the currently scanned square, and then undergoes a displacement $d \in \{-1, 0, 1\}$. A computation is initiated by inscribing some finite string of symbols on the tape as the input (the remaining squares being blank), and setting the machine to state q_0 scanning the first symbol of the input. The quintuple listing then takes care of the subsequent steps of the computation. If at any stage the machine moves into state h, then the process terminates, and the contents of the tape at that stage constitute the output.

E-mail address: a.p.galton@exeter.ac.uk

0096-3003/$ - see front matter © 2005 Elsevier Inc. All rights reserved.
doi:10.1016/j.amc.2005.09.086

Turing derived this conception from a careful analysis of the essential features of computations performed by human computers. The state set Q corresponds to the possible 'states of mind' of the human (insofar as these are relevant to the computation); the alphabet A is assumed to be finite on the grounds that a human cannot reliably discriminate infinitely many symbols. The linear tape is an idealisation of the working surface, typically sheets of paper, employed by the human, its unboundedness reflecting the fact that, in principle, if one runs out of paper, more can always be ordered. The displacement corresponds to the human's shift of attention from one part of the working surface to another. The restriction to scanning only one square of the tape at a time, and moving only between neighbouring squares, is more apparent than real: if the machine is allowed to scan more than one square at a time, or to move its head between squares which are not immediate neighbours, no extra computing power is thereby obtained.

Turing explored the scope and limitations of this model of computation in some depth. He showed how to construct a *universal Turing machine* which, if given as input the quintuple listing for another Turing machine **T**, together with a representation of a possible input i to the latter, would simulate the action of **T** on i. This provides a model for the general purpose digital computer which can be programmed to simulate arbitrary special-purpose machines—in short, what we now know as computers, in contradistinction to the human computers of Turing's day. Turing paid particular attention to the *printing problem*: Can one construct a Turing machine **P** which, when given as input the quintuple listing for an arbitrary Turing machine **T** and one of its tape symbols s, will determine whether or not **T** eventually prints s when run with a blank input tape? Turing showed that no such machine **P** can exist; thus there are well-defined problems which cannot be solved by means of Turing machines. He used this result to resolve—in the negative—an outstanding problem of mathematical logic, the decision problem for first-order predicate calculus (Hilbert's *Entscheidungsproblem*). By showing that this was equivalent to the printing problem for Turing machines, he was able to transfer the undecidability result for the latter directly to the former.

Turing [1] claims that 'the [Turing machine] computable numbers include all numbers which could naturally be regarded as computable'. By the computable numbers, he means 'the real numbers whose expressions as a decimal are calculable by finite means'. An example is the irrational number $\pi = 3.1415926\ldots$, whose decimal expansion, although apparently patternless, indeed 'random', can be mechanically churned out as far as we please by means of any of a large number of known procedures. But Turing points out that this restriction to computable *numbers* is not a serious limitation, since 'it is almost equally easy to define and investigate computable functions of an integral variable or a real or computable variable, computable predicates, and so forth'. We can think of his claim as referring more generally to the computation of *functions*, i.e., well-defined and determinate relations between input and output, leading to the more general thesis that *a function is computable (or effectively calculable) if and only if there is a Turing machine which computes it*.

It is normal to restrict this to functions $f : \mathbb{N}^n \to \mathbb{N}$ over the natural numbers. This too is not as restrictive as it sounds: if our inputs and outputs are finite strings over some pre-specified finite alphabets A and B respectively, then such strings can always be interpreted as natural numbers expressed in bases m and n, where m, n are the cardinalities of A, B respectively; and hence the function mapping input to output can be regarded as a function of the required kind.

To assess the validity or otherwise of the thesis, we must first pin down what we want to mean by 'computable', and then determine whether or not the class of functions thereby identified coincides with the class of Turing-computable functions. What makes this problematic is that whereas the latter class is completely well-defined, the former is subject to many different possible interpretations, all of which seem to be in various ways *ill*-defined.

The primary evidence in support of the thesis is that a number of different independent attempts at characterising the class of computable functions have led to definitions which have subsequently been proved to be equivalent to Turing's formulation, the earliest example being the *recursive* and *λ-definable* functions of Church [2]. Church himself proved the equivalence of these two notions, and Turing subsequently (in the Appendix to [1]) proved them to be equivalent to the Turing-computable functions. Church put forward the class of recursive functions in an attempt to characterise the notion of effective calculability, and hence it is entirely fitting that his name should be linked with that of Turing in what we now call the Church–Turing thesis (CTT).

Elsewhere [3] I have considered in some detail the problematic nature of CTT and the question of what it would take to discredit it. Here I want to pursue some different, though related, lines of enquiry, focussing on the question of what precise meaning should be given to CTT, and whether or not various recently proposed notions of computability should be regarded as discrediting it.

2. Infinity and idealisation

There being no dispute about what is meant by a Turing-computable function, the uncertainty surrounding CTT naturally accrues to the other term of the claimed equivalence, computability or effective calculability. Copeland [4] has reminded us that in the 1930s the word ‘computer’ referred to a *person*, not to what we would now call a computer. It follows that in Turing’s formulation of the thesis, ‘computable’ should be interpreted to mean ‘such as can (in principle) be derived by a human computer following an effective procedure’. The ‘in principle’ here allows us to invoke an idealised version of the human computer and the conditions under which their computations are performed. The Turing machine is built in the image of such an idealised human computer. Under this interpretation, CTT may be stated as follows:

Human C–T thesis (HCTT): A function is computable by an idealised human computer if and only if there is a Turing machine which computes it.

A key word is ‘idealised’. The idealisation here amounts to the supposition that (1) the computer never makes a mistake, either in reading and writing symbols or in following the prescribed instructions, and (2) the computer has at her disposal unlimited time and space—but only ever uses finitely much of both. Thus HCTT is true if the functions computable by an idealised human computer are identical to those computable by Turing machines. Since the Turing machine is so closely modelled on the idealised human computer, it seems reasonable, indeed unremarkable, that HCTT should be true—a claim enshrined in Robin Gandy’s version of the thesis as

Turing’s theorem: Any function which is effectively calculable by an abstract human being following a fixed routine is effectively calculable by a Turing machine… and conversely [5].

—although to establish this with certainty, one should have to characterise in precise mathematical terms the nature of an ‘abstract human being following a fixed routine’, with the obvious danger that any attempt to do this is liable to result in a reinvention of the Turing machine and consequent collapse of the ‘theorem’ into vacuity. For this reason, there is an inherent vagueness about HCTT which must always cast a shadow of doubt over any claims that this version of the thesis is definitively either true or false.

In contrast to the above remarks, many commentators on CTT have interpreted the notion of computability much more broadly as having reference to the capabilities of *all possible physical computing devices*, and not just idealised human computers. This leads us to the following formulation:

Physical C–T thesis (PCTT): A function is computable by means of a physically possible computing device if and only if there is a Turing machine which computes it.

The notion of ‘physical’ invoked here relates, of course, to the physics of the universe as it actually is, and not to our current state of knowledge about that physics. Since it is clear that our current physical theories are incomplete and very possibly incorrect in many respects, it follows that we are not, in fact, in a position to determine the truth or falsity of PCTT to any greater degree of certainty than our confidence in current physical theory warrants.

Although the word ‘idealised’ does not appear in the statement of PCTT, idealisation is still at work. ‘Computing a function’ here implies the possibility, in principle, of evaluating the function for *any* of its possible arguments. The words ‘in principle’ are essential. Not all the computations that can in principle be performed by a physically possible computer are actually possible. For example, a physical adding machine cannot in practice be used to add *all* pairs of numbers—e.g., if the adding machine has registers holding 10 digits each, then we cannot use the machine in the standard way to add the numbers 88,888,888,888 and 99,999,999,999. Why then do we say that this adding machine computes the function $\mathrm{add} : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ rather than some function on the set $\{0, 1, 2, \ldots, 10^{10} - 1\}$ of numbers representable in the machine’s registers? It is because its mechanism is such that in a suitably idealised version of the machine or the world in which it operates, it can handle any numbers as inputs: the physical adding machine embodies, as closely as possible given certain limitations of size, convenience, etc., an idealised abstract machine which does compute precisely the function *add*.

These are *reasonable* idealisations. We do not reject the idea of an unlimited Turing machine tape or unlimited time in which to complete a computation, since we know that any actual Turing machine computation only uses a finite amount of tape and time—and we are prepared to accept the fact that if that finite amount is too big the computation cannot be carried out in practice. On the other hand, we surely *would* reject as an unreasonable idealisation the notion of a computing device which required us to use the *whole* of an infinite tape: e.g., if the input was required to be the complete decimal expansion of π. The difference in this case is the requirement to admit an *actually infinite* object or process into the computation, whereas the standard Turing machine idealisation only makes reference to *potential* infinity. It is this which explains why the former idealisation is acceptable whereas the latter is not.

The distinction between potential and actual infinity may be explained as follows. Euclid showed that, given any prime number p, it is possible to find a prime q greater than p—all we have to do is to multiply together all the primes up to and including p, add 1, and then find the least prime factor of the result, all perfectly determinate operations which we know how to carry out. Since we can repeat this construction as often as we wish, there is no limit, in principle, to how many extra primes we can generate. This result is often expressed by saying that 'there are infinitely many primes', but a more careful way of expressing it would be to say that for any positive integer n, we can find more than n primes; more precisely

For any $n \in \mathbb{N}$, there is a set of primes P of cardinality greater than n,

which has the logical form $(\forall n \exists P)\phi$. Although the sets of primes under consideration here are all finite, there is no limit to their size; hence we say that the primes are *potentially infinite*. But this does not require us to acknowledge the existence of any actually infinite totality of primes. If we do make that leap, then we are saying that

There is a set of primes P such that, for any $n \in \mathbb{N}$, the cardinality of P is greater than n,

which has the logical form $(\exists P \forall n)\phi$. As is well known we cannot validly infer $(\exists P \forall n)\phi$ from $(\forall n \exists P)\phi$; this is an example of the 'quantifier shift fallacy' [6, Chapter 1].

I propose the following tentative rule: that when considering computation and computability, idealisations which refer to *potentially* infinite (i.e., unbounded) quantities are generally acceptable, whereas idealisations which refer to *actually* infinite quantities are at least *prima facie* unacceptable. On this basis the supposition that every Turing-computable function can be computed by an idealised human computer would appear to be acceptable; in which case, we should have no qualms about accepting HCTT as true—as has indeed been asserted by many commentators. But if the Human thesis is true, then for the Physical thesis to be true as well we require that *an idealised human computer can compute any function that is computable by any physical means*, which implies that for any possible computing device, the function(s) it computes can also be computed by an idealised human computer. The apparent implausibility of this has led a number of researchers to assert that there are physically possible computations that are not (even in principle, under any acceptable form of idealisation) humanly possible. Such computations have been discussed under the banner of *hypercomputation*, a term introduced by Copeland and Proudfoot in [7].

3. Questioning the physical Church–Turing thesis: accelerating Turing machines and infinite computation

The standard definition of a Turing machine says nothing about how long each computation step takes, but in considering what such machines can compute there is a general presumption that in a finite stretch of time, only finitely many steps can be executed. By an *Accelerating Turing Machine* (ATM) is meant a machine which is exactly like a Turing Machine except that in any one computation, each computation step takes half the time of the preceding step; such a machine can execute infinitely many steps in a finite time.[1]

[1] See [8] for an account of the history of this idea.

There are persuasive physical reasons for rejecting ATMs as a model of computation. *Almost all* the computations in the accelerating infinite sequence must take place over less than 10^{-43} s, the Planck time, which at least on one interpretation is regarded as the smallest interval of time over which physical change can occur.[2] Before rejecting ATMs on this ground, however, note that there are recursive functions on the natural numbers for which the smallest *ordinary* Turing machine has, say, 10^{80} states, more than the number of protons in the observable universe and hence unrealisable in practice—yet we do not use this to reject the standard (non-accelerating) Turing machine as a model of computation.[3] It is true of both TMs and ATMs that some of them are too big to be actually constructed; it is also true that for any machine of either type, some of its computations cannot be executed because they require prohibitively large amounts of tape. However, with ATMs there is an additional reason why some of the computations cannot be executed, and that is that the computation steps are too short. And the crucial point is that it is precisely *these* computations in virtue of which the computational power of ATMs as a class exceeds that of ordinary Turing machines. But let us set aside any worries we may have about the acceptability of ATMs and see what we could do with them if we had them.

A computation performed by an ATM would be an example of a *supertask*, a notion first introduced by Thomson [9] in connection with Zeno's paradoxes. A supertask is a task comprising an infinite sequence of (ordinary) tasks, each of which must be completed for the supertask to be completed (see [10] for a general discussion). To be definite, suppose that we want a machine to decide for us the truth or otherwise of some formula $\exists x P(x)$, where $P(x)$ is a Turing-computable property of natural numbers. Let **P** be a Turing machine which can decide for an arbitrary input $x \in \mathbb{N}$ whether or not $P(x)$ holds. Then we can construct a Turing machine $\mathbf{P}^*$ which behaves as follows:

> Starting with a blank tape, $\mathbf{P}^*$ initially writes '*F*' on square 0. Then, beginning with 0, it considers each natural number n in turn and simulates the action of **P** on input n. If this simulation returns the answer 'false', then $\mathbf{P}^*$ moves on to $n + 1$, but if it returns 'true' then it replaces the symbol on square 0 by a '*T*' and halts.

This is an ordinary Turing machine, and it halts if and only if $\exists x P(x)$. If this formula *is* true, then $\mathbf{P}^*$ could, in principle, be used to discover this (though the smallest n for which $P(n)$ holds may be too large for us to discover it in practice). But if the formula is false, we cannot discover this using $\mathbf{P}^*$ in the ordinary way, since however long we have been running the machine without its halting we can never be sure that it will not halt if we continue the computation just a little bit longer. We say that the formula $\exists x P(x)$ is 'semi-decidable'.

If instead of running $\mathbf{P}^*$ 'in the ordinary way' (i.e., with all steps executing in constant time), we run it as an ATM, then we could use it to determine the truth value of $\exists x P(x)$. If the nth computation step takes 2^{-n} s, then all the steps will be completed after 1 s. At that point, if the symbol in tape square 0 is '*F*', then the formula is false, since no satisfying instance has been found and every possibility has been tried; but if it is '*T*', then the formula is true.

Amongst the problems that can be solved in this way are some which are known to be unsolvable by an ordinary (non-accelerating) Turing machine—for example, the printing problem. The printing problem for machine M with symbol s may be expressed as $\exists x P(M, s, x)$, where $P(M, s, x) = 1$ if machine M prints s at the xth step, and 0 otherwise—which for any given M and s is a Turing-computable function of x. Thus the ATM is a theoretical construct which, if regarded as an acceptable model of computation, might lead us to cast doubt on the validity of CTT. It would not, presumably, affect HCTT, since to imagine a human capable of the required accelerating performance is surely to stretch the limits of idealisation beyond the bounds of the acceptable. But the broader PCTT, referring to any possible computing device, becomes vulnerable to any serious indication that ATMs might be an acceptable idealisation of some form of physically realisable computing device.

The key feature enabling an ATM to compute functions beyond those accessible to ordinary Turing machines is the performance of infinitely many computation steps in a finite amount of time. So long as this is understood in such a way that some of the computation steps are of impossibly short duration, then if our

[2] However, an alternative interpretation is that it is the smallest interval of time over which change can be *measured*. This interpretation might not be so inimical to the existence of ATMs.

[3] This point was drawn to my attention by Yaacov Choueka.

earlier point about the Planck time is accepted, physical considerations must rule such machines out of court. However, all that is actually needed is not that infinitely many steps occupy a finite length of time, but that the results of infinitely many steps can become available a finite length of time after the computation is initiated. This requires that (1) an infinite sequence of computation steps is initiated at a certain time t, and (2) a result which may depend on *all* the steps in the sequence is made available at some later time $t + d$, where d is finite. Let us call a computation which satisfies these two conditions an *infinite computation*. Had it been physically possible, an accelerating Turing machine would have been one way of achieving infinite computation, and in Newtonian space–time, there does not appear to be any other possibility.

A number of authors have investigated the possibility of realising conditions (1) and (2) in some form of space–time sanctioned by general relativity [11–15]. A *Malament–Hogarth point* (M–H point) in a relativistic space–time is a point p whose past light-cone includes the entirety of some future-directed half-infinite timeline l. An example is the *anti-de Sitter space–time* portrayed schematically in Fig. 1. Here s represents the start point of the half timeline l, and c represents the boundary of the past light-cone of p. L is the half timeline beginning at s and passing through p.

The existence of M–H points is not incompatible with general relativity, but such points can only occur in certain special models of the theory. A spacetime which includes M–H points is called a M–H spacetime. Earman and Norton [13] discuss in some detail the characteristics of M–H spacetimes, and in particular consider whether our own universe may be of this kind, and if so whether the physical conditions required for us to be able to exploit them in the service of computation are realistic; they summarise their discussion in the words ‘it is not clear that any M–H spacetime qualifies as physically possible and physically realistic’.

Assuming that our universe does contain M–H points, then we can in principle exploit them to produce infinite—and therefore non-Turing-equivalent—computations. Instead of a single human computer, we need a two-person team consisting of, say, a master and a slave. The slave is in fact a human computer, who at s is despatched along l, and embarks on the infinite sequence of computations $P(1), P(2), P(3), \ldots$. Of course, at some stage after finitely many computations have been completed, the slave will die, and we have to suppose a sufficient community of slaves, of both sexes, to provide a never-ending supply of human computers to continue the computations into the infinite future. If at any stage a number n is found for which $P(n)$ evaluates to ‘true’, then the computations are stopped and a signal is sent to be received by the master at p—this must be possible since the whole of l is in the past of p. But so long as no such n is found, the computations continue. The master, having despatched his slave community, follows timeline L, and on arriving at p looks for a signal from the slaves. If there is a signal, then he knows that $\exists x P(x)$ is true; if not, then—in principle—he knows it is false. Of course this is a highly idealised situation; he cannot know that the community of slaves is not wiped out at a particular time along l. If this happens, then the computations of $P(n)$ for values of n greater than some n_0 are not performed, and the absence of a signal from l at p does not reliably testify to the non-existence

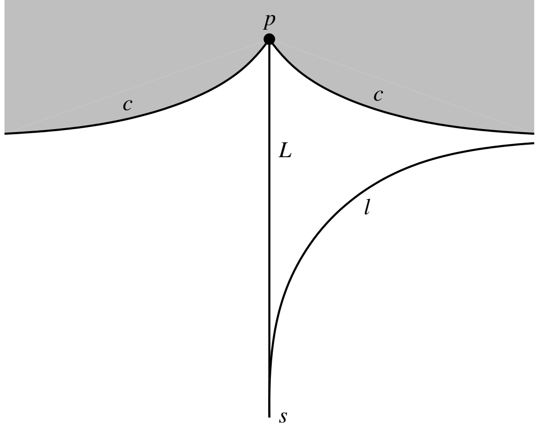

Fig. 1. Anti-de Sitter spacetime (adapted from [13]).

of an n for which $P(n)$ holds. But we are working with an idealisation which precludes catastrophes of this kind. Is this a 'reasonable' idealisation? Of course, we don't need to use a community of slaves—an indestructible Turing machine would do.

If we extend the interpretation of the phrase 'idealised human computer' to include the situation envisaged above, involving in fact an infinite supply of such computers, or alternatively two computers, one of whom lives forever, then perhaps we should admit that even HCTT is false. But we might decide that the notion of 'idealised human computer' cannot be extended in this way; that even if, at least under idealisation, the M–H scenario is regarded as physically possible, we do not regard this as exemplifying the notion of an idealised human computer. In that case we would have driven a clear wedge between the notions of 'what can be computed by an idealised human computer' and 'what can be computed by any physically possible means', the latter, but not the former, including the computation of arbitrary functions $\exists x P(x)$ where P is a Turing-computable predicate.

Hogarth [14] is surely right when he says that 'the physically possible computing limit… is firmly tied to some contingent and as yet unknown facts about the world'—namely, facts about what is physically possible (e.g., whether there exist M–H points, and whether they can be exploited for computation without violating other physical constraints). But even if the realm of physical possibility were exhaustively known, this still would not enable us to give a final assessment of HCTT, for in order to do that we should have to determine what counts as an appropriate idealisation of the actual physical capabilities of humans.

Even if our universe contains M–H points, which we are in principle able to exploit for the purposes of infinite computation, the degree of idealisation involved here might seem too extreme for either version of CTT. For we must assume that the computer (human or otherwise) that is dispatched along timeline l can survive for an infinite time—in other words, for ever—without malfunctioning. This is quite different from the sort of idealisation required for the claim that an ordinary Turing machine, or a human computer, can compute a function such as addition of natural numbers. Here we need only idealise to the extent of saying that our computer can, in principle, perform any one of an infinite series of individually finite computations—but we do not have to suppose that it can perform the infinite task of carrying out *all* of those computations.[4]

These qualms might be overturned if, say, M–H based infinite computation were shown to work in practice. Suppose we find examples of M–H points (it is suggested that these may exist in the vicinity of slowly rotating massive black holes), and use them to successfully evaluate $\exists n P(n)$ for a large number of cases (involving a range of different Ps) in which the answer is already known. Would this not afford *prima facie* evidence that the actual infinities posited by the M–H process really exist, and provide adequate justification (albeit not conclusive) for using the same set-up to provide answers to problems for which the solution is *not* already known? Possibly: but it would remain true that the existence of actual infinities would be only an inference from the observed phenomena, they themselves not being in any sense directly observed. And that leaves it open that there may exist alternative equally valid interpretations of the phenomena which do not invoke any actual infinities.

4. Other routes to hypercomputation

If we are to open the discussion up to include arbitrary physical means of computation, then we must surely include computational paradigms such as analogue computers, neural networks, molecular computers, and quantum computers.

4.1. Neural networks

A neural network consists of a set of simple processing units ('nodes') connected by means of directed links to which are attached numerical weights. The nodes are of three types: input nodes, hidden nodes, and output

[4] Once again, note the quantifier shift—between 'for each n, there is possible situation in which the computer performs the computation for n' and 'there is a possible situation in which, for each n, the computer performs the computation for n'.

nodes. Activation at the input nodes is propagated through the network to produce activation at the output nodes. An outward signal is propagated from a node just so long as the sum of the input signals, weighted by the connection weights, exceeds some pre-determined threshold. Neural networks with integer or rational connection weights can only compute Turing-computable functions—as is obvious from the fact that the whole network can in that case be exactly simulated by means of a Turing machine.

However, Siegelmann [16] has shown that if the connection weights are allowed to be infinite-precision real numbers, then a neural network can compute a non-Turing-computable function. Idealisation here raises its head again if, as seems to me very plausible, the distinction between rational and irrational numbers only exists in the abstract idealised world of mathematical thought, and not in the physical world. Physically, this would imply that there is no way of setting a connection weight to be exactly π, as opposed to any of its infinitely many arbitrary close rational approximations. The level of idealisation required here would again be unacceptable since to have the full decimal expansion of π (or some equivalent representation) is to have an actual, and not merely potential infinity.

To have π to *potentially* infinite precision is easy: all we need is an algorithm for computing the successive decimals in the expansion as far as we want. Thus so long as the connection weights on our neural network are all Turing-computable real numbers, then the network as a whole can be simulated by means of a Turing machine, and can therefore only compute a Turing-computable function. Thus in order to compute a non-Turing-computable function, at least one of the weights in one of Siegelmann's networks must be a non-Turing-computable real number. Thus as Davis [17] says, 'the non-computability that Siegelmann gets from her neural nets is nothing more than the non-computability she has built into them'.

4.2. Analogue computation

Whereas digital computation is performed using physical systems capable of assuming a clearly identifiable set of discrete states, analogue computation can range more widely, and in particular exploit the apparent continuity which we find in many physical processes. This apparent physical continuity finds expression in the mathematical conception of continuous functions on the real numbers. Analogue computation exploits the fact that continuous physical processes can be described in mathematical terms as if they were mathematically continuous, with real numbers representing the values of continuously variable physical quantities; and hence such physical processes can be used to perform computations on the real numbers. However, it would be a mistake to imagine that the physical quantities we find in nature provide an accurate model of the real numbers. There is no such thing as infinite-precision measurement—for example, we know that at the quantum level the physical world simply does not behave in the smooth mathematically continuous way presupposed by classical physics. This accords with the view that the distinction between rational and irrational numbers (on which the mathematical conception of the real numbers depends) has no *physical* meaning.

Moore [18] has developed a recursion theory on the real numbers to provide a theoretical underpinning for continuous-time analogue computation. He uses *integration* as the closest analogue to recursion in the realm of real computation. Thus where standard recursion theory defines the operation of primitive recursion by means of the equations

$$\begin{aligned} &h(\mathbf{x},0) = f(\mathbf{x}),\\ &h(\mathbf{x},y+1) = g(\mathbf{x},y,h(\mathbf{x},y)), \end{aligned}$$

so in Moore's system we have *differential recursion* defined by

$$\begin{aligned} &h(\mathbf{x},0) = f(\mathbf{x}),\\ &\frac{\partial}{\partial y}h(\mathbf{x},y) = g(\mathbf{x},y,h(\mathbf{x},y)), \end{aligned}$$

leading to

$$h(\mathbf{x},y) = f(\mathbf{x}) + \int_0^y g(\mathbf{x},y',h(\mathbf{x},y'))\,\mathrm{d}y'.$$

It is a beautiful and ingenious theory, but as Moore himself admits, it is ‘almost certainly unphysical’. In some of his examples, ‘the variables and their derivatives go to infinity during the course of a computation. Any physical realization of such a computer would presumably run out of resources or explode’.

Thus once again we find a *mathematically* consistent theory of computation which could support hypercomputation, but because it requires actual infinities, rather than merely potential ones, it is unrealisable in practice.

4.3. Quantum computation

‘Standard’ quantum computation (SQC), as developed by Deutsch and others, is based on the use of *qubits*, which are analogous to the bits of ordinary computation. A qubit is a quantum system capable of assuming two states, which may exist in quantum superposition. A register of n qubits can hold 2^n different superposed values simultaneously, and a computation which uses such registers can, in principle, simultaneously compute output values corresponding to all the possible inputs. In addition, the phenomenon of quantum entanglement, by which the state of one qubit may be strongly constrained by the states of certain other qubits, can be exploited to control the massive parallelism obtained by superposition. The technicalities are complex, and in reality the benefits of quantum computation can only be realised in certain very special cases, with the result that as yet not many significant quantum algorithms have been developed.

The benefit of SQC is in the realm of efficiency rather than functionality, as already noted by Deutsch [19]. That is, SQC does not enable one to compute any non-Turing-computable functions, although certain functions may be computed very much more efficiently than is possible by means of any classical computer—a result which is already enough to make SQC an attractive proposition. However, an alternative model for Quantum Computation has been proposed by Kieu: Quantum Adiabatic Computation [20]. In Kieu’s words,

> The idea is to encode the solution of some problem to be solved in to the ground state, $|g\rangle$, of some suitable Hamiltonian, H_P. But as it is easier to implement the Hamiltonian than to obtain the ground state, we should start the computation in a different and readily obtainable initial ground state, $|g_I\rangle$, of some initial Hamiltonian, H_I, then deform this Hamiltonian in a time T into the Hamiltonian whose ground state is the desired one [20, p. 551].

Based on this idea, Kieu proposes a detailed quantum mechanical algorithm for solving Hilbert’s Tenth Problem, the problem of determining whether or not there exist integer solutions to an arbitrary Diophantine equation. This problem is known to be equivalent to the printing problem for Turing Machines, and therefore cannot be solved by any Turing-equivalent means.

Kieu points out two ‘key ingredients’ needed for the quantum algorithm to work:

- ‘The ability of the theory of Quantum Mechanics to describe and predict physical processes to the level of exactness required’.
- ‘Our ability to physically implement certain Hamiltonians having infinite numbers of energy levels’.

The first of these does not appear to conflict with our rule of thumb that potential, but not actual, infinities constitute acceptable idealisations for the purposes of defining computability: Kieu does not require that physical processes are predictable with infinite exactness, only that the predictions be *exact enough*. To be sure, this might in practice put an impossible strain on our physical capabilities, and if so, this kind of computation might be ruled out of court for practical reasons.

But the second key ingredient does seem to involve us in the murky world of actual infinities—indeed, Kieu speaks of ‘regaining the lost computability through the physical world, *presumed infinite*’ (my italics). Kieu says that there are no known *physical* principles outlawing his key assumptions; but might there not be *logical* principles outlawing them? At any rate, there is no logical warrant for assuming the availability of infinitely many states, even if systems with arbitrarily large finite numbers of states can in principle be constructed.

4.4. Hypercomputation and the Church–Turing thesis

If hypercomputation is possible, is CTT thereby invalidated? This depends partly on which version of the thesis we are considering. So long as hypercomputation cannot be realised as an effective procedure carried out by humans, HCTT would be untouched by it. But we have to attend closely to the rules of the game here: exactly what counts as an effective humanly-executable procedure? Does the rigmarole with masters and slaves in the M–H scenario count? If it does, then even HCTT would become vulnerable to at least certain forms of hypercomputation.

On the other hand, PCTT would be invalidated by any hypercomputational scheme which is physically possible, modulo acceptable idealisations. But of course, at the present time there exists no evidence whatever that any form of hypercomputation is feasible in practice. Thus our conservative conclusion must be that in our present state of knowledge CTT, in *both* versions, remains intact, if not indubitable.

References

[1] A.M. Turing, On computable numbers, with an application to the Entscheidungsproblem, Proceedings of the London Mathematical Society 42 (1936) 230–265.
[2] A. Church, An unsolvable problem of elementary number theory, The American Journal of Mathematics 58 (1936) 345–363.
[3] A. Galton, The Church–Turing thesis: its nature and status, in: P. Millican, A. Clark (Eds.), The Legacy of Alan Turing, Machines and Thought, vol. I, Clarendon Press, Oxford, 1996, pp. 137–164.
[4] B. Jack Copeland, The Church–Turing thesis, in: E.N. Zalta (Ed.), Stanford Encyclopedia of Philosophy. August 2002. Available from: <http://plato.stanford.edu/entries/church-turing/>.
[5] R. Gandy, The confluence of ideas in 1936, in: Rolf Herken (Ed.), The Universal Turing Machine: A Half-Century Survey, Oxford University Press, Oxford, 1988, pp. 55–111.
[6] P.T. Geach, Logic Matters, Basil Blackwell, Oxford, 1972.
[7] B.J. Copeland, D. Proudfoot, Alan Turing's forgotten ideas in computer science, Scientific American 280 (1999) 76–81.
[8] B.J. Copeland, Accelerating Turing machines, Minds and Machines 12 (2002) 281–301.
[9] J.F. Thomson, Tasks and super-tasks, Analysis 15 (1) (1954) 1–13.
[10] J.P. Laraudogoitia, Supertasks, in: E.N. Zalta, (Ed.), Stanford Encyclopedia of Philosophy, November 2001. Available from: <http://plato.stanford.edu/entries/spacetime-supertasks/>.
[11] I. Pitowsky, The physical Church thesis and physical computational complexity, Iyyun 39 (1990) 81–99.
[12] M. Hogarth, Does general relativity allow an observer to view an eternity in a finite time? Foundations of Physics Letters 5 (1992) 173–181.
[13] J. Earman, J.D. Norton, Forever is a day: supertasks in Pitowsky and Malament–Hogarth spacetimes, Philosophy of Science 60 (1993) 22–42.
[14] M. Hogarth, Non-Turing computers and non-Turing computability, Proceedings of the Philosophy of Science Association (PSA) 1 (1994) 126–138.
[15] O. Shagrir, I. Pitowsky, Physical hypercomputation and the Church–Turing thesis, Minds and Machines 13 (2003) 87–101.
[16] H.T. Siegelmann, Computation beyond the Turing limit, Science 268 (1995) 545–548.
[17] M. Davis, The myth of hypercomputation, in: C. Teuscher (Ed.), Alan Turing: Life and Legacy of a Great Thinker, Springer, 2004, pp. 195–211.
[18] C. Moore, Recursion theory on the reals and continuous-time computation, Theoretical Computer Science 162 (1996) 23–44.
[19] D. Deutsch, Quantum theory, the Church–Turing principle, and the universal quantum computer, Proceedings of the Royal Society, London, A 400 (1985) 97–117.
[20] T.D. Kieu, Quantum hypercomputation, Minds and Machines 12 (2002) 541–561.