IOP Concise Physics

Computation in Science

Konrad Hinsen

Chapter 1

What is computation?

This book is about computation in the context of scientific research. Scientists should always try to be precise about what they say, so I will start by explaining what this term actually means. It turns out that this is not as trivial as it may seem to be. Moreover, there are also pragmatic reasons for discussing the nature of computation, because a good grasp of what computation actually *is* makes it much easier to appreciate what it can and cannot be used for in scientific research.

Common dictionary definitions of computation are remarkably imprecise. Oxford proposes 'the action of mathematical calculation' or 'the use of computers, especially as a subject of research or study'. Merriam-Webster has 'the act or action of computing: calculation' or 'the use or operation of a computer' but also 'a system of reckoning'. Both dictionaries refer to 'calculation', but do not provide useful definitions for that word either. The vagueness of these dictionary definitions can be traced back to the long-lasting confusion about what computation is and how it relates to mathematics. It was only the development of formal logic and mathematical formalism in the early 20th century that led to a precise definition of computation, which helped to pave the way to the development of automatic computing machines. This definition is the topic of section 1.1.

Beyond the exploration of what defines computation, I will also look at what computation is for scientists, i.e. *why* computation is so important in scientific research. Most scientists probably think of computers and computation as tools that they use to process experimental data or mathematical equations. However, there are other roles that computation plays in science, and which I will briefly discuss in section 1.2.

1.1 Defining computation

1.1.1 Numerical computation

The most familiar computations are the numerical calculations that we all do in everyday life: adding up prices, multiplying the length and width of a room to

doi:10.1088/978-1-6817-4093-5ch1

compute its surface, dividing a quantity into equal parts, etc. Most people today have even more occasions for doing numerical calculations in their professional lives. Basic arithmetic on anything that can be quantified is so fundamental that it takes up a significant part of training in the first years of school. Mechanical aids for numerical operations, such as the abacus, have been used for at least 2000 years and perhaps even for much longer.

We do not think much about how we do simple arithmetic operations on small numbers, and in fact we often just recall the result that we have internalized due to frequent use. But as soon as we work on larger numbers, mental calculation becomes a mechanical activity based on rules we have learned. An example for such a rule is: to multiply a number by 9, multiply it by 10 and then subtract the original number.

When operations become too complex to be handled in the head, we turn to pen and paper for a more reliable record of the intermediate results in our calculations. A large number of calculation techniques with pen and paper has been developed in the course of the centuries, ranging from addition via long division to the calculation of cube roots.

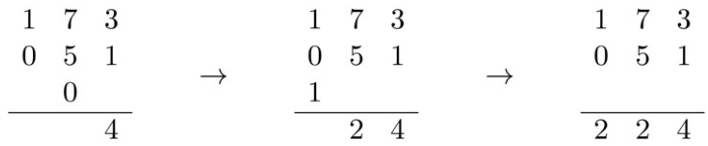
As a simple example, consider adding the numbers 173 and 51. One way to do it systematically starts by writing one below the other, adding zeros to the left of the smaller number to make the number of digits equal:

$$\begin{array}{ccc} 1 & 7 & 3 \\ 0 & 5 & 1 \end{array}$$

We then process the digits from right to left, starting by adding 3 and 1. We 'know' that the result is 4, because we have done this so often. But for the slightly more complex operation of multiplication, most readers probably remember how they memorized multiplication tables in school—tables that were actually printed in books. To be precise in our description of addition, we will use an equally explicit addition table for one-digit integers:

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

We actually need *two* such tables, the second one being for the 'carry' digit, which is 1 when the sum of the two digits is 10 or more, and which is 0 otherwise. We note the one-digit sum and the carry digit, and move on to the left to handle the next digit:



This method, and all the other arithmetic operations we use, rely on the positional notation for numbers that is used all around the world today. Any natural number can be written as a sequence of the digits 0 to 9. Another symbol, the minus sign, takes care of negative integers, and one further symbol, either the decimal point or the division slash, makes it possible to express fractions. The rules for arithmetic can then be formulated as rules for manipulating sequences of symbols, as shown above for addition, which can be applied mechanically.

1.1.2 From numbers to symbols

It is indeed important to realize that the method outlined above does not work on *numbers*, but on a specific *representation* for numbers. Numbers are an abstract concept, which cannot be manipulated using mechanical rules. Different representations lead to different methods for doing arithmetic. Even though the decimal character string '42', the roman-numeral character string 'XLII', and the English-language character string 'forty-two' refer to the same number, they cannot be manipulated in the same way. In fact, our recipe for addition never refers to numbers. It takes two sequences of digits as input, and produces one sequence of digits as output. Applying the recipe does not require any knowledge of numbers, only the capacity to work with a finite set of symbols and apply rules to them.

A recipe for solving a specific problem by manipulating symbols is called an *algorithm*. The word is derived from the name of the Persian mathematician al-Khwārizmī, who lived in the 9th century. His book describing the 'Indian numbers', which today we call Arabic numerals, introduced our modern decimal notation and its rules for arithmetic into Europe [1]. The use of this system was called 'algorism' in the late middle ages, and later the spelling and meaning transformed into today's 'algorithm'. The positional notation for numbers transformed arithmetic from a difficult craft performed by trained specialists into a routine task that could be mastered by almost everyone.

Today the decimal representation of numbers seems so obvious to us that we often make no difference between a number and its decimal representation. This phenomenon is not limited to numbers. We rarely distinguish carefully between a word and its meaning either, and in quantum physics, to cite an example from science, the confusion between a quantum state and one of its many possible representations is very common. When thinking about computation, it is often important to recall that the Universe of symbols and the Universe of meanings are separate. In computer science, this is known as the distinction between *syntax* and *semantics*. Syntax defines which sequences of symbols a particular algorithm deals with, for example 'a sequence of any number of the digits 0 to 9'. Semantics defines how such sequences of symbols are interpreted, such as 'a natural number'.

No knowledge of semantics is needed to *apply* our algorithm for adding two natural numbers, but it is essential to understand what the algorithm does, and in particular which problems it can help to solve.

A symptom of the confusion between numbers and their representations is the popular saying that 'computers work only on numbers'. This is patently false: what today's digital computers work on is sequences of bits, a bit being a symbol from an alphabet containing two symbols. We often choose the digits 0 and 1 to represent this alphabet, suggesting an interpretation as binary numbers, i.e. numbers represented in a positional notation with base 2. The idea that computers work on numbers is mistaken because bits can equally well represent information other than numbers. It is also misleading in another way because it suggests that any number-related problem can be solved by a computer. However, most numbers cannot be represented by sequences of bits and therefore cannot enter in any computation.

It is easy to see that bits can represent any information that can be written as sequences of symbols at all. Suppose we have an alphabet with N symbols, for example the $N = 26$ letters of the English alphabet. We can then make up a translation table that assigns a unique set of values for five bits to each symbol in our alphabet. With five bits, we have $2^5 = 32$ distinct values, so six values will be left unused. Our translation table allows us to encode any English word in terms of bit sequences.

It is less obvious and perhaps even surprising to many readers that most numbers cannot be represented as bit sequences. For natural numbers, there is no problem: any sequence of bits can be interpreted as a natural number in base 2 notation. Inversely, every natural number can be written in base 2 notation, and therefore as a sequence of bits. It is thus possible to define a one-to-one correspondence between natural numbers and bit sequences. In mathematical terminology, the set of natural numbers is isomorphic to the set of bit sequences. Since we can perform computations on sequences of bits, we can perform computations on natural numbers. In fact, any set of values that we wish to do computations on must be isomorphic to the set of bit sequences, or equivalently to the set of natural numbers, or to a subset of such a set. Such sets are called *countable*. All finite sets are countable: just write down the elements in some order and then write a number below to each element, starting from 1. Infinite sets that are countable are called *countably infinite* sets.

It is straightforward to see that integers are still countable: use one bit for the sign, and then a natural-number bit sequence for the absolute value. It takes a bit more effort to show that the rational numbers, i.e. the set of all quotients of integers, are countable. By the definition of countability, this requires the assignment of a unique natural number to each rational number. The standard procedure is based on a two-dimensional matrix-like arrangement of the rational numbers:

$$\begin{array}{cccccc}
\frac{1}{1} & \frac{2}{1} & \frac{3}{1} & \frac{4}{1} & \cdots \\
\frac{1}{2} & \frac{2}{2} & \frac{3}{2} & \frac{4}{2} & \cdots \\
\frac{1}{3} & \frac{2}{3} & \frac{3}{3} & \frac{4}{3} & \cdots \\
\frac{1}{4} & \frac{2}{4} & \frac{3}{4} & \frac{4}{4} & \cdots \\
\vdots & \vdots & \vdots & \vdots & \ddots
\end{array}$$

The entries of this infinite matrix can now be enumerated along diagonals:

$$\begin{array}{cccccc} 1 & 3 & 6 & 10 & 15 & \dots \\ 2 & 5 & 9 & 14 & & \dots \\ 4 & 8 & 13 & & & \dots \\ 7 & 12 & & & & \dots \\ 11 & & & & & \dots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{array}$$

A more sophisticated enumeration scheme would skip over each number that is equal to one that already received an index earlier. For example, 2/2 would be skipped because it is equal to 1/1.

The proof that the set of real numbers is *not* countable is more involved, and I will not reproduce it here. Like many other proofs concerning infinite sets, it goes back to Georg Cantor, a German mathematician who laid the foundations of set theory in the late 19th century, and actually provided the first rigorous definition of the real numbers. The complex numbers, being a superset of the real numbers, are also uncountable. There are, however, countable number sets larger than the rational numbers. A well-known one in mathematics is the set of *algebraic numbers*, defined as the roots of polynomials with integer coefficients. In the context of computation, the largest useful countable subset of the real numbers is the set of *computable numbers*, which was introduced by Alan Turing in the same 1936 publication as the Turing machine. I will come back to this subject in chapters 2 and 3, because it is of central importance for the use of computation in science.

We can now write down a first definition of computation, which will be refined in chapter 3:

> **Computation is the transformation of sequences of symbols according to precise rules.**

What will need refinement is the 'precise rules', which must be expressed so precisely that a machine can apply them unambiguously.

1.1.3 Non-numerical computation

Once we get rid of the idea that computation is about numbers, we can easily identify other operations that qualify as computations. One example is solving equations by algebraic manipulations. The steps leading from

$$y + 2x = z$$

to

$$x = \frac{1}{2}(z - y)$$

are completely mechanical and can be formulated as an algorithm. The practical evidence is that computers can do the job. Software packages that implement such

operations are called *computer algebra* systems, emphasizing algebraic manipulations. However, computer algebra systems also perform other non-numerical algorithms, for example finding the derivative or integral of an elementary function. The algorithm for computing derivatives is simple and taught in every calculus course. In contrast, the algorithm for computing indefinite integrals is very complicated [2] and was implemented as a computer program only in 1987 [3].

A perhaps more surprising use of computation in mathematics is the validation of proofs. A proof is a sequence of deduction steps, each of which establishes the truth of a statement based on other statements already known to be true and a finite set of rules of logic deduction. In textbooks and mathematical publications, proofs are written concisely for human readers who have a prior knowledge of mathematics. But when all the details are spelled out, proofs consist of nothing but applications of a finite set of deduction rules. These rules are applied to statements that must respect the rules of a formal language. The use of computers to check such detailed proofs is becoming more and more common, both in mathematical research and in industrial applications. This involves writing the proof in some formal language, as a sequence of symbols. The 'proof checking' computation transforms this sequence of symbols into a 'true' or 'false' statement about the proof's validity.

Leaving the narrow scope of mathematics, we find a huge number of domains where computation is applied to textual data. Finding a word in a text is a computation: it transforms the input data (the text and the word to look for) into output data (the position in the text where the word occurs), both of which can be encoded as sequences of symbols. Likewise, aligning two DNA sequences, translating a sentence from English to French, or compiling a Fortran program, are computations in which the data being processed are text. In fact, numbers and text are the two kinds of elementary data from which almost everything else is made up by composition: images are represented as two-dimensional arrays of numbers, dictionaries as sets of word pairs, etc. However, this fundamental role of numbers and text is due to their importance to humans, not computers.

Another kind of data that are becoming increasingly important for scientific computation are graphs, in particular when used to describe networks. Traffic flow, protein interactions in a cell, and molecular structures are all examples of what can be described by graphs. An example of a computation on graphs is a check for cycles. This transforms the input data (a graph) into output data (a list of all cycles in the graph). Encoding graphs, cycles, and lists as sequences of symbols is not as obvious as encoding numbers and text. Humans already represented numbers and text by sequences of symbols long before thinking about computers and computation. The human representation of graphs, on the other hand, takes the form of two-dimensional drawings. The precise techniques for representing graphs as sequences of symbols are beyond the scope of this book, but I will come back to the general question of information representation in computers in section 5.3.2.

1.2 The roles of computation in scientific research

Computation as a tool

The most visible role of computation in scientific research is its use as a tool. Experimentalists process their raw data, for example to correct for artifacts of their equipment. Then they fit a theoretical model to their data by adjusting parameters. Theoreticians compute numerical predictions from a model, in order to compare them to experimental results. Both experimentalists and theoreticians make heavy use of computation for understanding their data, in particular using visualization techniques.

It is worth making a distinction between *computation* as a tool and *computers* as a tool. Computers perform computations, but they also deal with other tasks. Scientists use computers for communicating with their peers, looking up information in Wikipedia, and for controlling lab equipment. None of these tasks involve much visible computation, though there is computation going on under the hood. Today's computers are as much communication devices as computation devices.

Computation for understanding

Richard Feynman had written on his blackboard: 'What I cannot create, I do not understand.' We cannot understand a theory, a model, or an approximation, unless we have done something with it. One way to gain experience with mathematical models is to apply them to concrete situations. Another one, even more powerful, is to implement them as computer programs. Donald Knuth has expressed this very succinctly [4]:

> It has often been said that a person does not really understand something until he teaches it to someone else. Actually a person does not really understand something until he can teach it to a computer, i.e. express it as an algorithm. The attempt to formalize things as algorithms leads to a much deeper understanding than if we simply try to comprehend things in the traditional way.

The utility of writing programs for understanding scientific concepts and mathematical models comes from the extreme rigor and precision required in programming. Communication between humans relies on shared knowledge, starting with the definition of the words of everyday language. A typical scientific article assumes the reader to have significant specialist knowledge in science and mathematics. Even a mathematical proof, probably the most precise kind of statement in the scientific literature, assumes many definitions and theorems to be known by the reader, without even providing a list of them. A computer has no such prior knowledge. We must communicate with a computer in a formal language which is precisely defined. Every aspect of our science that somehow impacts a computed result must be expressed in this formal language in order to obtain a working computer program.

Another reason why writing a program is often useful for understanding a mathematical model is that an algorithm is necessarily constructive. In the physical sciences, most theories take the form of differential equations. These equations fully

define their solutions, and are also useful for reasoning about their general properties, but provide no obvious way of *finding* one. Writing a computer program requires, first of all, to think about what this program *does*, and then about *how* it should go about this.

Implementing a scientific model as a computer program also has the advantage that, as a bonus, you get a tool for exploring the consequences of the model for simple applications. Computer-aided exploration is another good way to gain a better understanding of a scientific model (see [5, 6] for some outstanding examples). In the study of complex systems, with models that are directly formulated as algorithms, computational exploration is often the only approach to gaining scientific insight [7].

Computation as a form of scientific knowledge

A computer program that implements a theoretical model, for example a program written with the goal of understanding this model, is a peculiar written representation of this model. It is therefore an expression of scientific knowledge, much like a textbook or a journal article. We will see in the following chapters that much scientific knowledge can be expressed in the form of computer programs, and that much of today's scientific knowledge exists in fact *only* in the form of computer programs, because the traditional scientific knowledge representations cannot handle complex structured information. This raises important questions for the future of computational science, which I will return to in chapter 6.

Computation as a model for information processing in nature

Computers are physical devices that are designed by engineers to perform computation. Many other engineered devices perform computation as well, though usually with much more limited capacity. The classic example from computer science textbooks is a vending machine, which translates operator input (pushing buttons, inserting coins) into actions (deliver goods), a task that requires computation. Of course a vending machine does more than compute, and as users we are most interested in that additional behavior. Nevertheless, information processing, and thus computation, is an important aspect of the machine's operation.

The same is true of many systems that occur in nature. A well-known example is the process of cell division, common to all biological organisms, which involves copying and processing information stored in the form of DNA [8]. Another example of a biological process that relies on information processing is plant growth [9]. Most animals have a nervous system, a part of the body that is almost entirely dedicated to information processing. Neuroscience, which studies the nervous system, has close ties to both biology and computer science. This is also true of cognitive science, which deals with processes of the human mind that are increasingly modeled using computation.

Of course, living organisms are not *just* computers. Information processing in organisms is inextricably combined with other processes. In fact, the identification of computation as an isolated phenomenon, and its realization by engineered devices that perform a precise computation any number of times, with as little dependence

on their environment as is technically possible, is a hallmark of human engineering that has no counterpart in nature. Nevertheless, focusing on the computational aspects of life, and writing computer programs to simulate information processing in living organisms, has significantly contributed to a better understanding of their function.

On a much grander scale, one can consider all physical laws as rules for information processing, and conclude that the whole Universe is a giant computer. This idea was first proposed in 1967 by German computer pioneer Konrad Zuse [10] and has given rise to a field of research called *digital physics*, situated at the intersection of physics, philosophy, and computer science [11].

1.3 Further reading

Computation has its roots in numbers and arithmetic, a story that is told by Georges Ifrah in *The Universal History of Numbers* [12].

The use of computation for understanding has been emphasized in the context of physics by Sussman and Wisdom [13]. They developed an original approach to teaching classical mechanics by means of computation [14]. Computer programming is also starting to be integrated into science curricula because of its utility for understanding. See the textbook by Langtangen [15] for an example.

Bibliography

[1] Crossley J N and Henry A S 1990 Thus spake al-Khwārizmī: a translation of the text of Cambridge University Library Ms. Ii.vi.5 *Hist. Math.* **17**(2) 103–31

[2] Risch R H 1969 The problem of integration in finite terms *Trans. Amer. Math. Soc.* **139** 167–89

[3] Bronstein M 1990 Integration of elementary functions *J. Symb. Comput.* **9**(2) 117–73

[4] Knuth D E 1974 Computer science and its relation to mathematics *Am. Math. Mon.* **81** 323–43

[5] Victor B 2011 Kill math http://worrydream.com/KillMath/

[6] Victor B 2013 Media for thinking the unthinkable http://worrydream.com/#!/MediaForThinkingTheUnthinkable

[7] Downey A B 2012 *Think Complexity: Complexity Science and Computational Modeling* 1st edn (Sebastopol, CA: O'Reilly Media)

[8] Xing J, Mather W and Hong C 2014 Computational cell biology: past present and future *Interface Focus* **4**(3) 20140027

[9] Prusinkiewicz P, Hanan J S, Fracchia F D, Lindenmayer A, Fowler D R, de Boer M J M and Mercer L 1996 *The Algorithmic Beauty of Plants* (*The Virtual Laboratory*) (New York: Springer)

[10] Zuse K 1970 *Calculating Space* (Boston, MA: MIT Technical Translation) AZT-70-164-GEMIT

[11] Zenil H (ed) 2013 *A Computable Universe: Understanding and Exploring Nature as Computation* (Singapore: World Scientific)

[12] Ifrah G 2000 *The Universal History of Numbers: From Prehistory to the Invention of the Computer* (*The Universal History of Numbers: From Prehistory to the Invention of the Computer* vol 1) (New York: Wiley)

[13] Sussman G J and Wisdom J 2002 The role of programming in the formulation of ideas *MIT Artificial Intelligence Laboratory Technical Report* AIM-2002-018
[14] Sussman G and Wisdom J 2001 *Structure and Interpretation of Classical Mechanics* (Cambridge, MA: MIT Press)
[15] Langtangen H P 2014 *A Primer on Scientific Programming with Python (Texts in Computational Science and Engineering)* 4th edn (Berlin: Springer)