CHAPTER 1

Introduction

Why are computational methods in mathematics important? What can we do with these methods? What is the difference between computation by hand and by computer? What do I need to know to perform computations on computers?

These are natural questions for a student to ask before starting a course on computational methods. And therefore it is also appropriate to try and provide some short answers already in this introduction. By the time you reach the end of the notes you will hopefully have more substantial answers to these as well as many other questions.

1.1 A bit of history

A major impetus for the development of mathematics has been the need for solving everyday computational problems. Originally, the problems to be solved were quite simple, like adding and multiplying numbers. These became routine tasks with the introduction of the decimal numeral system. Another ancient problem is how to determine the area of a field. This was typically done by dividing the field into small squares, rectangles or triangles with known areas and then adding up. Although the method was time-consuming and could only provide an approximate answer, it was the best that could be done. Then in the 18th century Newton and Leibniz developed the differential calculus. This made it possible to compute areas and similar quantities via quite simple symbolic computations, namely integration and differentiation.

In the absence of good computational devices, this has proved to be a powerful way to approach many computational problems: Look for deeper structures in the problem and exploit these to develop alternative, often non-numerical, ways to compute the desired quantities. At the beginning of the 21st century

this has developed mathematics into an extensive collection of theories, many of them highly advanced and requiring extensive training to master. And mathematics has become much more than a tool to solve practical computational problems. It has long ago developed into a subject of its own, that can be valued for its beautiful structures and theories. At the same time mathematics is the language in which the laws of nature are formulated and that engineers use to build and analyse a vast diversity of man-made objects, that range from aircrafts and economic models to digital music players and special effects in movies.

An outsider might think that the intricate mathematical theories that have been developed have quenched the need for old fashioned computations. Nothing could be further from the truth. In fact, a large number of the developments in science and engineering over the past fifty years would have been impossible without huge calculations on a scale that would have been impossible a hundred years ago. The new device that has made such calculations possible is of course the digital computer.

The birth of the digital computer is usually dated to the early 1940s. From the beginning it was primarily used for mathematical computations, and today it is an indispensable tool in almost all scientific research. But as we all know, the usefulness of computers goes far beyond the scientific laboratories. Computers are now essential tools in virtually all offices and homes in our society, and small computers control a wide range of machines.

The one feature of modern computers that has made such an impact on science and society is undoubtedly the speed with which a computer operates. We mentioned above that the area of a field can be computed by dividing the field into smaller parts like triangles and rectangles whose areas are very simple to compute. This has been known for hundreds of years, but the method was only of limited interest as complicated shapes had to be split into a large number of smaller parts to get sufficient accuracy. The symbolic computation methods that were developed worked well, but only for certain special problems. Symbolic integration, for example, is only possible for a small class of integrals; the vast majority of integrals cannot be computed by symbolic methods. The development of fast computers means that the old methods for computing areas, based on dividing the shape into simple parts, are highly relevant as we can very quickly sum up the areas of a large number of triangles or rectangles.

With all the spectacular accomplishments of computers one may think that formal methods and proofs are not of interest any more. This is certainly not the case. A truth in mathematics is only accepted when it can be proved through strict logical reasoning, and once it has been proved to be true, it will always be true. A mathematical theory may lose popularity because new and better theories are developed, but the old theory remains true as long as those who

discovered it did not make any mistakes. Later, new discoveries are made which may bring the old, and often forgotten, theory back into fashion again. The simple computational methods is one example of this. In the 20th century mathematics went through a general process of more abstraction and many of the old computational techniques were ignored or even forgotten. When the computer became available, there was an obvious need for computing methods and the old techniques were rediscovered and applied in new contexts. All properties of the old methods that had been established with proper mathematical proofs were of course still valid and could be utilised straightaway, even if the methods were several hundred years old and had been discovered at a time when digital computers were not even dreamt of.

This kind of renaissance of old computational methods happens in most areas when computers are introduced as a tool. However, there is usually a continuation of the story that is worth mentioning. After some years, when the classical computational methods have been adapted to work well on modern computers, completely new methods often appear. The classical methods were usually intended to be performed by hand, using pencil and paper. Three characteristics of this computing environment (human with pencil and paper) is that it is quite slow, is error prone, and has a preference for computations with simple numbers. On the other hand, an electronic computer is fast (billions of operations pr. second), is virtually free of errors and has no preference for particular numbers. A computer can of course execute the classical methods designed for humans very well. However, it seems reasonable to expect that even better methods should be obtainable if one starts from scratch and develops new methods that exploit the characteristics of the electronic computer. This has indeed proved to be the case in many fields where the classical methods have been succeeded by better methods that are completely unsuitable for human operation.

1.2 Computers and different types of information

The computer has become a universal tool that is used for all kinds of different purposes and tasks. To understand how this has become possible we must know a little bit about how a computer operates. A computer can really only work with numbers, and in fact, the numbers even have to be expressed in terms of 0s and 1s. It turns out that any number can be represented in terms of 0s and 1s so that is no restriction. But how can computers work with text, sound, images and many other forms of information when it can really only handle numbers?

1.2.1 Text

Let us first consider text. In the English alphabet there are 26 letters. If we include upper case and lower case letters plus comma, question mark, space, and other common characters we end up with a total of about 100 different characters. How can a computer handle these characters when it only knows about numbers? The solution is simple; we just assign a numerical code to each character. Suppose we use two decimal digits for the code and that 'a' has the code 01, 'b' the code 02 and so on. Then we can refer to the different letters via these codes, and words can be referred to by sequences of codes. The word 'and' for example, can be referred to by the sequence 011404 (remember that each code consists of two digits). Multi-word texts can be handled in the same way as long as we have codes for all the characters. For this to work, the computer must always know how to interpret numbers presented to it, either as numbers or characters or something else.

1.2.2 Sound

Computers work with numbers, so a sound must be converted to numbers before it can be handled by a computer. What we perceive as sound corresponds to small variations in air pressure. Sound is therefore converted to numbers by measuring the air pressure at regular intervals and storing the measurements as numbers. On a CD for example, measurements are taken 44 100 times a second, so three minutes of sound becomes 7 938 000 measurements of air pressure. Sound on a computer is therefore just a long sequence of numbers. The process of converting a given sound to regular numerical measurements of the air pressure is referred to as digitising the sound, and the result is referred to as digital sound.

1.2.3 Images

Images are handled by computers in much the same way as sound. Digital cameras have an image sensor that records the amount of light hitting its rectangular array of points called pixels. The amount of light at a given pixel corresponds to a number, and the complete image can therefore be stored by storing all the pixel values. In this way an image is reduced to a large collection of numbers, a digital image, which is perfect for processing by a computer.

1.2.4 Film

A film is just a sequence of images shown in quick succession (25-30 images pr. second), and if each image is represented digitally, we have a film represented

by a large number of numerical values, a digital film. A digital film can be manipulated by altering the pixel values in the individual images.

1.2.5 Geometric form

Geometric shapes surround us everywhere in the form of natural objects like rocks, flowers and animals as well as man-made objects like buildings, aircrafts and other machines. A specific shape can be converted to numbers by splitting it into small pieces that each can be represented as a simple mathematical function like for instance a cubic polynomial. A cubic polynomial is represented in terms of its coefficients, which are numbers, and the complete shape can be represented by a collection of cubic pieces, joined smoothly together, i.e., by a set of numbers. In this way a mathematical model of a shape can be built inside a computer.

Graphical images of characters, or fonts, is one particular type of geometric form that can be represented in this way. Therefore, when you read the letters on this page, whether on paper or a computer screen, the computer figured out exactly how to draw each character by computing its shape from a collection of mathematical formulas.

1.2.6 Laws of nature

The laws of nature, especially the laws of physics, can often be expressed in terms of mathematical equations. These equations can be represented in terms of their coefficients and solved by performing computations based on these coefficients. In this way we can simulate physical phenomena by solving the equations that govern the phenomena.

1.2.7 Virtual worlds

We have seen how we can represent and manipulate sound, film, geometry and physical laws by computers. By combining objects of this kind, we can create artificial or virtual worlds inside a computer, built completely from numbers. This is exactly what is done in computer games. A complete world is built with geometric shapes, creatures that can move (with movements governed by mathematical equations), and physical laws, also represented by mathematical equations. An important part of creating such virtual worlds is to deduce methods for how the objects can be drawn on the computer screen — this is the essence of the field of computer graphics.

A very similar kind of virtual world is used in machines like flight simulators and machines used for training purposes. In many cases it is both cheaper and safer to give professionals their initial training by using a computer simulator rather than letting them try 'the real thing'. This applies to pilots as well as

heart surgeons and requires that an adequate virtual world is built in terms of mathematical equations, with a realistic user interface.

In many machines this is taken a step further. A modern passenger jet has a number of computers that can even land the airplane. To do this the computer must have a mathematical model of itself as well as the equations that govern the plane. In addition the plane must be fitted with sensors that measure quantities like speed, height, and direction. These data are measured at regular time intervals and fed into the mathematical model. Instead of just producing a film of the landing on a computer screen, the computer can actually land the aircraft, based on the mathematical model and the data provided by the sensors.

In the same way surgeons may make use of medical imaging techniques to obtain different kinds of information about the interior of the patient. This information can then be combined to produce an image of the area undergoing surgery, which is much more informative to the surgeon than the information that is available during traditional surgery.

Similar virtual worlds can also be used to perform virtual scientific experiments. In fact a large part of scientific experiments are now performed by using a computer to solve the mathematical equations that govern an experiment rather than performing the experiment itself.

1.2.8 Summary

Via measuring devices (sensors), a wide range of information can be converted to digital form, i.e., to numbers. These numbers can be read by computers and processed according to mathematical equations or other logical rules. In this way both real and non-real phenomena can be investigated by computation. A computer can therefore be used to analyse an industrial object before it is built. For example, by making a detailed mathematical model of a car it is possible to compute its fuel consumption and other characteristics by simulation in a computer, without building a single car.

A computer can also be used to guide or run machines. Again the computer must have detailed information about the operation of the machine in the form of mathematical equations or a strict logical model from which it can compute how the machine should behave. The result of the computations must then be transferred to the devices that control the machine.

To build these kinds of models requires specialist knowledge about the phenomenon which is to be modelled as well as a good understanding of the basic tools used to solve the problems, namely mathematics, computing and computers.

1.3 Computation by hand and by computer

As a student of mathematics, it is reasonable to expect that you have at least a vague impression of what classical mathematics is. What I have in mind is the insistence on a strict logical foundation of all concepts like for instance differentiation and integration, logical derivation of properties of the concepts defined, and the development of symbolic computational techniques like symbolic integration and differentiation. This is all extremely important and should be well-known to any serious student of mathematics and the mathematical sciences. Not least is it important to be fluent in algebra and symbolic computations.

When computations are central to classical mathematics, what then is the *new* computational approach? To understand this we first need to reflect a bit on how we do our pencil-and-paper computations. Suppose you are to solve a system of three linear equations in three unknowns, like

$$\begin{aligned} 2x+4y-2z &= 2,\\ 3x-6z &= 3,\\ 4x-2y+4z &= 2. \end{aligned}$$

There are many different ways to solve this, but one approach is as follows. We observe that the middle equation does not contain y, so we can easily solve for one of x or z in terms of the other. If we solve for x we can avoid fractions so this seems like the best choice. From the second equation we then obtain $x = 1+2z$. Inserting this in the first and last equations gives

$$\begin{aligned} 2+4z+4y-2z &= 2,\\ 4+8z-2y+4z &= 2, \end{aligned}$$

or

$$\begin{aligned} 4y+2z &= 0,\\ -2y+12z &= -2. \end{aligned}$$

Using either of these equations we can express y or z in terms of one another. In the first equation, however, the right-hand side is 0 and we know that this will lead to simpler arithmetic. And if we express z in terms of y we avoid fractions. From the first equation we then obtain $z = -2y$. When this is inserted in the last equation we end up with $-2y+12(-2y) = -2$ or $-26y = -2$ from which we see that $y = 1/13$. We then find $z = -2y = -2/13$ and $x = 1+2z = 9/13$. This illustrates how an experienced equation solver typically works, always looking for shortcuts and simple numbers that simplify the calculations.

This is quite different from how a computer operates. A computer works according to a very detailed procedure which states exactly how the calculations are to be done. The procedure can tell the computer to look for simple numbers and shortcuts, but this is usually a waste of time since most computers handle fractions just as well as integers.

Another, more complicated example, is computation of symbolic integrals. For most of us this is a bag of isolated techniques and tricks. In fact the Norwegian mathematician Viggo Brun once said that *differentiation is a craft; integration is an art*. If you have some experience with differentiation you will understand what Brun meant by it being a craft; you arrive at the answer by following fairly simple rules. Many integrals can also be solved by definite rules, but the more complicated ones require both intuition and experience. And in fact most indefinite integrals cannot be solved at all. It may therefore come as a surprise to many that computers can be programmed to perform symbolic integration. In fact, Brun was wrong. There is a precise procedure which will always give the answer to the integral if it exists, or say that it does not exist if this is the case. For a human the problem is of course that the procedure requires so much work that for most integrals it is useless, and integration therefore appears to be an art. For computers, which work so must faster, this is less of a problem, see Figure 1.1. Still there are plenty of integrals (most!) that require so many calculations that even the most powerful computers are not fast enough. Not least would the result require so much space to print that it would be incomprehensible to humans!

These simple examples illustrate that when (experienced) humans do computations they try to find shortcuts, look for patterns and do whatever they can to simplify the work; in short they tend to improvise. In contrast, computations on a computer must follow a strict, predetermined algorithm. A computer may appear to improvise, but such improvisation must necessarily be planned in advance and built into the procedure that governs the calculations.

1.4 Algorithms

In the previous section we repeatedly talked about the 'procedure' that governs a calculation. This procedure is simply a sequence of detailed instructions for how the quantity in question can be computed; such procedures are usually referred to as *algorithms*. Algorithms have always been important in mathematics as they specify how calculations should be done. In the past, algorithms were usually intended to be performed manually by humans, but today many algorithms are designed to work well on digital computers.

If we want an algorithm to be performed by a computer, it must be expressed

$$\int \frac{\sin(x)}{\cos(6x)}\,dx = \left(\frac{1}{6}+\frac{i}{6}\right)(-1)^{1/4}\,\text{ArcTan}\left[\left(\frac{1}{2}+\frac{i}{2}\right)(-1)^{1/4}\,\text{Sec}\left[\frac{x}{2}\right]\left(\text{Cos}\left[\frac{x}{2}\right]+\text{Sin}\left[\frac{x}{2}\right]\right)\right] -$$
$$\left(\frac{1}{6}+\frac{i}{6}\right)(-1)^{3/4}\,\text{ArcTanh}\left[\left(\frac{1}{2}+\frac{i}{2}\right)(-1)^{3/4}\,\text{Sec}\left[\frac{x}{2}\right]\left(\text{Cos}\left[\frac{x}{2}\right]-\text{Sin}\left[\frac{x}{2}\right]\right)\right] +$$
$$\frac{1}{12\left(2+\sqrt{2}\right)}\left(1+\sqrt{2}\right)\left(x+2\sqrt{3}\,\text{ArcTanh}\left[\frac{2+\left(2+\sqrt{2}\right)\text{Tan}\left[\frac{x}{2}\right]}{\sqrt{6}}\right]-\text{Log}\left[\text{Sec}\left[\frac{x}{2}\right]^2\right]+\right.$$
$$\left.\text{Log}\left[\text{Sec}\left[\frac{x}{2}\right]^2\left(\sqrt{2}-2\,\text{Cos}[x]+2\,\text{Sin}[x]\right)\right]\right)+\left(\left(2\left(\sqrt{2}+\sqrt{3}\right)\text{ArcTanh}\left[\frac{2+\left(2+\sqrt{6}\right)\text{Tan}\left[\frac{x}{2}\right]}{\sqrt{2}}\right]+\right.\right.$$
$$\left.\left.\left(3+\sqrt{6}\right)\left(x-\text{Log}\left[\text{Sec}\left[\frac{x}{2}\right]^2\right]+\text{Log}\left[\text{Sec}\left[\frac{x}{2}\right]^2\left(\sqrt{6}-2\,\text{Cos}[x]+2\,\text{Sin}[x]\right)\right]\right)\right)\right.$$
$$\left.\left(1+\sqrt{6}\,\text{Sin}[x]\right)\left(3+\sqrt{6}-\left(2+\sqrt{6}\right)\text{Cos}[x]+\left(2+\sqrt{6}\right)\text{Sin}[x]\right)\right)\Big/$$
$$\left(12\left(\left(12+5\sqrt{6}\right)\text{Cos}[2x]+2\,\text{Cos}[x]\left(5+2\sqrt{6}+5\sqrt{6}\,\text{Sin}[x]\right)-\right.\right.$$
$$\left.\left.2\left(12+5\sqrt{6}+4\left(5+2\sqrt{6}\right)\text{Sin}[x]-6\,\text{Sin}[2x]\right)\right)\right)+$$
$$\left(\left(x-2\sqrt{3}\,\text{ArcTanh}\left[\frac{\sqrt{2}+\left(-1+\sqrt{2}\right)\text{Tan}\left[\frac{x}{2}\right]}{\sqrt{3}}\right]-\text{Log}\left[\text{Sec}\left[\frac{x}{2}\right]^2\right]+\right.\right.$$
$$\left.\left.\text{Log}\left[-\text{Sec}\left[\frac{x}{2}\right]^2\left(1+\sqrt{2}\,\text{Cos}[x]-\sqrt{2}\,\text{Sin}[x]\right)\right]\right)$$
$$\left.\left(\sqrt{2}+2\,\text{Sin}[x]\right)\left(-1+\sqrt{2}-\left(-2+\sqrt{2}\right)\text{Cos}[x]+\left(-2+\sqrt{2}\right)\text{Sin}[x]\right)\right)\Big/$$
$$\left(24\left(\left(-2+\sqrt{2}\right)\text{Cos}[x]-\left(-1+\sqrt{2}\right)(\text{Cos}[2x]+\text{Sin}[2x])\right)\right)+$$
$$\left(\left(-2\left(-2+\sqrt{6}\right)\text{ArcTanh}\left[\sqrt{2}+\left(\sqrt{2}-\sqrt{3}\right)\text{Tan}\left[\frac{x}{2}\right]\right]+\right.\right.$$
$$\left.\left.\left(3\sqrt{2}-2\sqrt{3}\right)\left(x-\text{Log}\left[\text{Sec}\left[\frac{x}{2}\right]^2\right]+\text{Log}\left[-\text{Sec}\left[\frac{x}{2}\right]^2\left(\sqrt{3}+\sqrt{2}\,\text{Cos}[x]-\sqrt{2}\,\text{Sin}[x]\right)\right]\right)\right)\right.$$
$$\left.\left(\sqrt{2}-2\sqrt{3}\,\text{Sin}[x]\right)\left(-3+\sqrt{6}-\left(-2+\sqrt{6}\right)\text{Cos}[x]+\left(-2+\sqrt{6}\right)\text{Sin}[x]\right)\right)\Big/$$
$$\left(24\left(\left(-12+5\sqrt{6}\right)\text{Cos}[2x]+2\,\text{Cos}[x]\left(-5+2\sqrt{6}+5\sqrt{6}\,\text{Sin}[x]\right)-\right.\right.$$
$$\left.\left.2\left(-12+5\sqrt{6}+4\left(-5+2\sqrt{6}\right)\text{Sin}[x]+6\,\text{Sin}[2x]\right)\right)\right)$$

Figure 1.1. An integral and its solution as computed by the computer program Mathematica. The function $\sec(x)$ is given by $\sec(x) = 1/\cos(x)$.

in a form that the computer understands. Various languages, such as C++, Java, Python, Matlab etc., have been developed for this purpose, and a computer program is nothing but an algorithm translated into such a language. Programming therefore requires both an understanding of the relevant algorithms and knowledge of the programming language to be used.

We will express the algorithms we encounter in a language close to standard mathematics which should be quite easy to understand. This means that if you want to test an algorithm on a computer, it must be translated to your preferred programming language. For the simple algorithms we encounter, this process should be straightforward, provided you know your programming language well enough.

1.4.1 Statements

The building blocks of algorithms are *statements*, and statements are simple operations that form the basis for more complex computations.

Definition 1.1. *An algorithm is a finite sequence of statements. In these notes there are only five different kinds of statements:*

1. Assignments
2. For-loops
3. If-tests
4. While-loops
5. Print statement

Statements may involve expressions*, which are combinations of mathematical operations, just like in general mathematics.*

The first four types of statements are the important ones as they cause calculations to be done and control how the calculations are done. As the name indicates, the print statement is just a tool for communicating to the user the results of the computations.

Below, we are going to be more precise about what we mean by the five kinds of statements, but let us also ensure that we agree what expressions are. The most common expressions will be formulas like $a + bc$, $\sin(a + b)$, or $e^{x/y}$. But an expression could also be a bit less formal, like *"the list of numbers* $\boldsymbol{x}$ *sorted in increasing order"*. Usually expressions only involve the basic operations in the

mathematical area we are currently studying and which the algorithm at hand relates to.

1.4.2 Variables and assignment

Mathematics is in general known for being precise, but its notation sometimes borders on being ambiguous. An example is the use of the equals sign, '='. When we are solving equations, like $x+2=3$, the equals sign basically tests equality of the two sides, and the equation is either true or false, depending on the value of x. On the other hand, in an expression like $f(x)=x^2$, the equals sign acts like a kind of definition or *assignment* in that we assign the value x^2 to $f(x)$. In most situations the interpretation can be deduced by the context, but there are situations where confusion may arise as we will see in section 2.3.1.

Computers are not very good at judging this kind of context, and therefore most programming languages differentiate between the two different uses of '='. For this reason it is also convenient to make the same kind of distinction when we describe algorithms. We do this by introducing the operator := for assignment and retaining = for comparison.

When we do computations, we may need to store the results and intermediate values for later use, and for this we use variables. Based on the discussion above, to store the number 2 in a variable a, we will use the notation $a := 2$; we say that the variable a is *assigned* the value 2. Similarly, to store the sum of the numbers b and c in a, we write $a := b+c$. One important feature of assignments is that we can write something like $s := s+2$. This means: Take the value of s, add 2, and store the result back in$s$. This does of course mean that the original value of s is lost.

Definition 1.2 (Assignment)**.** *The formulation*

var := *expression*;

means that the expression on the right is to be calculated, and the result stored in the variable var. For clarity the expression is often terminated by a semicolon.

Note that the assignment $a := b+c$ is different from the mathematical equation $a = b+c$. The latter basically tests equality: It is true if a and $b+c$ denote the same quantities, and false otherwise. The assignment is more like a command: Calculate the the right-hand side and store the result in the variable on the right.

1.4.3 For-loops

Very often in algorithms it is necessary to repeat essentially the same thing many times. A common example is calculation of a sum. An expression like

$$s = \sum_{i=1}^{100} i$$

in mathematics means that the first 100 integers should be added together. In an algorithm we may need to be a bit more precise since a computer can really only add two numbers at a time. One way to do this is

$s := 0;$
for $i := 1, 2, \ldots, 100$
    $s := s + i;$

The sum will be accumulated in the variable s, and before we start the computations we make sure s has the value 0. The for-statement means that the variable i will take on all the values from 1 to 100, and each time we add i to s and store the result in s. After the for-loop is finished, the total sum will then be stored in s.

Definition 1.3 (For-loop)**.** *The notation*

for *var* := *list of values*
    sequence of statements;

means that the variable var *will take on the values given by* list of values*. For each such value, the indicated* sequence of statements *will be performed. These may include expressions that involve the loop-variable* var*.*

A slightly more complicated example than the one above is

$s := 0;$
for $i := 1, 2, \ldots, 100$
    $x := \sin(i);$
    $s := s + x;$
$s := 2s;$

which calculates the sum $s = 2\sum_{i=1}^{100} \sin i$. Note that the two indented statements are both performed for each iteration of the for-loop, while the non-indented statement is performed after the for-loop has finished.

1.4.4 If-tests

The third kind of statement lets us choose what to do based on whether or not a condition is true. The general form is as follows.

Definition 1.4 (If-statement)**.** *Consider the statement*

if *condition*
    sequence of statements;
else
    sequence of statements;

where condition *denotes an expression that is either true or false. The meaning of this is that the first group of statements will be performed if condition is true, and the second group of statements if condition is false.*

As an example, suppose we have two numbers a and b, and we want to find the largest and store this in c. This can be done with the if-statement

if $a < b$
    $c := b$;
else
    $c := a$;

The condition in the if-test can be any expression that evaluates to true or false. In particular it could be something like $a = b$ which tests whether a and b are equal. This should not be confused with the assignment $a := b$ which causes the value of b to be stored in a.

Our next example combines all the three different kinds of statements we have discussed so far. Many other examples can be found in later chapters.

Example 1.5. Suppose we have a sequence of real numbers $(a_k)_{k=1}^n$, and we want to compute the sum of the negative and the positive numbers in the sequence separately. For this we need to compute two sums which we will store in the variables $s1$ and $s2$: In $s1$ we will store the sum of the positive numbers, and in $s2$ the sum of the negative numbers. To determine these sums, we step through the whole sequence, and check whether an element a_k is positive or negative. If it is positive we add it to $s1$ otherwise we add it to $s2$. The following algorithm accomplishes this.

$s1 := 0$; $s2 := 0$;
for $k := 1, 2, \ldots, n$
    if $a_k > 0$
        $s1 := s1 + a_k$;

else
$s2 := s2 + a_k$;

After these statements have been performed, the two variables $s1$ and $s2$ should contain the sums of the positive and negative elements of the sequence, respectively. ∎

1.4.5 While-loops

The final type of statement that we need is the while-loop, which is a combination of a for-loop and an if-test.

Definition 1.6 (While-statement)**.** *Consider the statement*

while *condition*
sequence of statements;

This will repeat the sequence of statements as long as condition *is true.*

Note that unless the logical condition depends on the computations in the sequence of statements this loop will either not run at all or run forever. Note also that a for-loop can always be replaced by a while-loop.

Consider once again the example of adding the first 100 integers. With a while-loop this can be expressed as

$s := 0$; $i := 1$;
while $i \leq 100$
$s := s + 1$;
$i := i + i$;

This example is expressed better with a for-loop, but it illustrates the idea behind the while-loop. A typical situation where a while-loop is convenient is when we compute successive approximations to some quantity. In such situations we typically want to continue the computations until some measure of the error has become smaller than a given tolerance, and this is expressed best with a while-loop.

1.4.6 Print statement

Occasionally want our toy computer to print something. For this we use a print statement. As an example, we could print all the integers from 1 to 100 by writing

for $i = 1, 2, \ldots, 100$
print i;

Sometimes we may want to print more elaborate texts; the syntax for this will be introduced when it is needed.

1.5 Doing computations on a computer

So far, we have argued that computations are important in mathematics, and computers are good at doing computations. We have also seen that humans and computers do calculations in quite different ways. A natural question is then how you can make use of computers in your calculations. And once you know this, the next question is how you can learn to use computers in this way.

1.5.1 How can computers be used for calculations?

There are at least two essentially distinct ways in which you can use a computer to do calculations:

1. You can use software written by others; in other words you may use the computer as an advanced calculator.

2. You can develop your own algorithms and implement these in your own programs.

Anybody who uses a computer has to depend on software written by others, so if your are going to do mathematics by computer, you will certainly do so in the 'calculator style' sometimes. The simplest example is the use of a calculator for doing arithmetic. A calculator is nothing but a small computer, and we all know that calculators can be very useful. There are many programs available which you can use as advanced calculators for doing common computations like plotting, integration, algebra and a wide range of other mathematical routine tasks.

The calculator style of computing can be very useful and may help you solve a variety of problems. The goal of these notes however, is to help you learn to develop your own algorithms which you can then implement in your own computer programs. This will enable you to deduce new computer methods and solve problems which are beyond the reach of existing algorithms.

When you develop new algorithms, you usually want to implement the algorithms in a computer program and run the program. To do this you need to know a programming language, i.e., an environment in which you can express your algorithm in such a way that a computer can execute the algorithm. It is therefore assumed that you are familiar with a suitable programming language already, or that you are learning one while you are working with these notes. Virtually any programming language like Java, C++, Python, Matlab, Mathematica,

..., will do. The algorithms in these notes will be written in a form that should make it quite simple to translate them to your choice of programming language. Note however that it will usually not work to just type the text literally into C++ or Python; you need to know the syntax (grammar) of the language you are using and translate the algorithm accordingly.

1.5.2 What do you need to know?

There are a number of things you need to learn in order to become able to deduce efficient algorithms and computer programs:

- You must learn to recognise when a computer calculation is appropriate, and when formal methods or calculations by hand are more suitable
- You must learn to translate your informal mathematical ideas into detailed algorithms that are suitable for computers
- You must understand the characteristics of the computing environment defined by your computer and the programming language you use

Let us consider each of these points in turn. Even if the power of a computer is available to you, you should not forget your other mathematical skills. Sometimes your intuition, computation by hand or logical reasoning will serve you best. On the other hand, with good algorithmic skills you can often use the computer to answer questions that would otherwise be impossible even to consider. You should therefore aim to gain an intuitive understanding for when a mathematical problem is suitable for computer calculation. It is difficult to say exactly when this is the case; a good learning strategy is to read these notes and see how algorithms are developed in different situations. As you do this you should gradually develop an algorithmic thinking yourself.

Once you have decided that some computation is suitable for computer implementation you need to formulate the calculation as a precise algorithm that only uses operations available in your computing environment. This is also best learnt through practical experience, and you will see many examples of this process in these notes.

An important prerequisite for both of the first points is to have a good understanding of the characteristics of the computing environment where you intend to do your computations. At the most basic level, you need to understand the general principles of how computers work. This may sound a bit overwhelming, but at a high level, these principles are not so difficult, and we will consider most of them in later chapters.

1.5.3 Different computing environments

One interesting fact is that as your programming skills increase, you will begin to operate in a number of different computing environments. We will not consider this in any detail here, but a few examples may illustrate this point.

Sequential computing As you begin using a computer for calculations it is natural to make the assumption that the computer works sequentially and does one operation at a time, just like we tend to do when we perform computations manually.

Suppose for example that you are to compute the sum

$$s = \sum_{i=1}^{100} a_i$$

where each a_i is a real number. Most of us would then first add a_1 and a_2, remember the result, then add a_3 to this result and remember the new result, then add a_4 to this and so on until all numbers have been added. The good news is that you can do exactly the same on a computer! This is called sequential computing and is definitely the most common computing environment.

Parallel computing If a group of people work together it is possible to add numbers faster than a single person can. The key observation is that the numbers can be summed in many different ways. We may for example sum the numbers in the order indicated by

$$s = \sum_{i=1}^{100} a_i = \underbrace{a_1 + a_2}_{a_1^1} + \underbrace{a_3 + a_4}_{a_2^1} + \underbrace{a_5 + a_6}_{a_3^1} + \cdots + \underbrace{a_{97} + a_{98}}_{a_{49}^1} + \underbrace{a_{99} + a_{100}}_{a_{50}^1}.$$

The key is that these sums are independent of each other. In other words we may hire 50 people, give them two numbers each and tell them to compute the sum. And when everybody is finished, we can of course repeat this and ask 25 people to add the 50 results,

$$s = \sum_{i=1}^{50} a_i^1 = \underbrace{a_1^1 + a_2^1}_{a_1^2} + \underbrace{a_3^1 + a_4^1}_{a_2^2} + \underbrace{a_5^1 + a_6^1}_{a_3^2} + \cdots + \underbrace{a_{47}^1 + a_{48}^1}_{a_{24}^2} + \underbrace{a_{49}^1 + a_{50}^1}_{a_{25}^2}.$$

At the next step we ask 13 people to the compute the 13 sums

$$s = \sum_{i=1}^{25} a_i^2 = \underbrace{a_1^2 + a_2^2}_{a_1^3} + \underbrace{a_3^2 + a_4^2}_{a_2^3} + \underbrace{a_5^2 + a_6^2}_{a_3^3} + \cdots + \underbrace{a_{21}^2 + a_{22}^2}_{a_{11}^2} + \underbrace{a_{23}^2 + a_{24}^2}_{a_{12}^3} + \underbrace{a_{25}^2}_{a_{13}^3}.$$

Note that the last person has an easy job; since the total number of terms in this sum is an odd number she just needs to remember the result.

The structure should now be clear. At the next step we ask 7 people to compute pairs in the sum $s = \sum_{i=1}^{13} a_i^3$ in a similar way. The result is the 7 numbers a_1^4, a_2^4,..., a_7^4. We then ask 4 people to compute the pairs in the sum $s = \sum_{i=1}^{7} a_i^4$ which results in the 4 numbers a_1^5, a_2^5, a_3^5 and a_4^5. Two people can then add pairs in the sum $s = \sum_{i=1}^{4} a_i^5$ and obtain the two numbers a_1^6 and a_2^6. Finally one person computes the final sum as $s = a_1^6 + a_2^6$.

Note that at each step, everybody can work independently. At the first step we therefore compute 25 sums in the time that it takes one person to compute one sum. The same is true at each step and the whole sum is computed in 6 steps. If one step takes 10 seconds, we have computed the sum of 100 numbers in one minute, while a single person would have taken 990 seconds or 16 minutes and 30 seconds.

Our simple example illustrates the concept of parallel computing. Instead of making use of just one computing unit, we may attack a problem with several units. Supercomputers, which are specifically designed for number crunching, work on this principle. Today's (July 2007) most powerful computer has 65 536 computing units, each of which again has two processors for a total of 131 072 computing units.

An alternative to expensive supercomputers is to let standard PCs work in parallel. They can either be connected in a specialised network or can communicate via the Internet. [1] In fact, modern PCs themselves are so-called multicore computers which consist of several computing units or CPUs, although at present, the norm is at most 4 or 8 cores.

One challenge with parallel computing that we have overlooked here is the need for communication between the different computing units. Once the 25 persons have completed their sums, they must communicate their results to the 12 people who are going to do the next sums. This time is significant and supercomputers have very sophisticated communication channels. At the other end of the scale, the Internet is in most respects a relatively slow communication channel for parallel computing.

Other computing environments Computing environments are characterised by many other features than whether or not calculations can be done in parallel. Other characteristics are the number of digits used in numerical computations, how numbers are represented internally in the machine, whether symbolic cal-

[1] There is a project aimed at computing large prime numbers that make use of the internet in this way, see `www.mersenne.org`.

culations are possible, and so on. It is not necessary to know the details of how these issues are handled on your computer, but if you want to use the computer efficiently, you need to understand the basic principles. After having studied these notes you should have a decent knowledge of the most common computing environments.

Exercises

1.1 The algorithm in example 1.5 calculates the sums of the positve and negative numbers in a sequence $(a_k)_{k=1}^n$. In this exercise you are going to adjust this algorithm.

- **a)** Change the algorithm so that it computes the sum of the positive numbers and the absolute value of the sum of the negative numbers.
- **b)** Change the algorithm so that it also determines the number of positive and negative elements in the sequence.

1.2 Write down an algorithm for summing two three-digit numbers. You may assume that it is known how to sum one-digit numbers.

1.3 Write down an algorithm which describes how you multiply together two three-digit numbers. You may assume that it is known how to sum numbers .

Part I

Numbers