Scientific Computing for Mechanical Engineering
Fundamentals of Numerical Methods

Andrea Prosperetti

Department of Mechanical Engineering
University of Houston

June 17, 2021

Contents

Chapter 1

Numbers

Mathematics distinguishes several types of numbers, the most familiar ones of which are natural numbers, integers, rational, real and complex.[1] Natural numbers are the positive (or non-negative) integers: 1, 2, ... (or 0, 1, 2, ...). Integers are the positive and negative natural numbers: 0, ± 1, ± 2, ... Rationals are numbers that can be represented by a fraction in which both numerator and denominator are integers (with the denominator being non-zero of course). Any rational number can be converted to decimal form as a string of digits (e.g. $\frac{4}{5} = 0.8$) and, conversely, a decimal representation expressed as a finite string of digits can be converted to a rational number, e.g. $16.4 = \frac{164}{100} = \frac{82}{50} = \frac{41}{25}$. When converted to decimal form some rational numbers give rise to an unending sequence of digits which, however, exhibit a clear pattern, e.g. $\frac{1}{3} = 0.33333\ldots$ or $\frac{52}{99} = 0.52525252\ldots$. Irrational numbers cannot be represented as fractions and, when converted to decimal notation, exhibit an unending sequence of digits which do not have a regular pattern but must be calculated one after the other. The class of real numbers includes integers, rational and irrational numbers.

Ordinarily computers can represent, and deal with, integers, real (or, better, rationals – see below) and complex numbers. While an integer, e.g. 7, can be seen as a rational number, $\frac{7}{1} = 7.0$, integers have special features that can be exploited to advantage in a computer as explained below in section 1.2.3. For this reason, they are treated as a separate number type. Since the rules for the algebra of complex numbers are different from those for real numbers, computers need to recognize complex numbers as a special category.[2] Scientific computation mostly uses real numbers; integers are mostly used for array indexing.

1.1 Number representation and round-off error

As will be explained below, computers represent numbers in binary (base 2) rather than in the familiar decimal system (base 10) but the important issue of round-off error can be better understood by using the more familiar decimal system. Thus, for now, we will "pretend" that computers use the decimal system rather than the binary representation of numbers.

A real number can be represented in an infinity of equivalent ways, e.g. $715.3 = 7.153 \times 10^2 = 0.7153 \times 10^3 = 71533 \times 10^{-1}$ and so on.[3] It is useful to standardize this representation. We use the normalized scientific notation (see `https://en.wikipedia.org/wiki/Significand`)[4]

$$A = (S)\, 0.\underbrace{abcde\ldots f}_{N digits} \times 10^E \,. \tag{1.1.1}$$

Here S is the sign, and can be plus or minus; the following N digits, each represented here by a letter, are the *significand* (also called, somewhat improperly, mantissa) and E denotes the exponent; N depends on the number of digits carried in the calculation, such as single precision, double precision etc. as explained below. The exponent E is uniquely determined by requiring that the first digit a not be 0.

A consequence of using only N digits is that a rational number having more than N digits cannot be represented exactly: any rational number with more than N digits must be approximated by (rounded to)

[1] These are by no means the only types. The concept of "number" is much more general and includes, for example, quaternions (ordered quadruplets of ordinary numbers), cardinal numbers (used to quantify the "size" of sets) and many (actually, an infinity of) others.

[2] It is not always possible to "avoid" complex numbers. For example, in general the eigenvalues of a real matrix are complex numbers and the rule to calculate the roots of a cubic equation can involve complex numbers even when all three roots are real.

[3] This is called floating-point representation of numbers because the decimal point can "float", i.e., it can be placed anywhere by adjusting the exponent. The same thing is possible when the number is expressed in binary or any other form.

[4] Another normalized standard representation in use is $A = (S)\, a\,.\, bcde\ldots f \times 10^E$.

one with N digits; for example, $\frac{1}{3} = 0.333\ldots$ must be truncated after N 3's. For the same reason, irrational numbers, such as π, cannot be represented exactly either. Thus, *any* number in a computer is either a rational number or a rational approximation to an irrational number. In fact, therefore, of the vast class of real numbers a computer can only represent and handle the (relatively "tiny" subset of) rationals with up to N significant digits, although it is common in the computing literature to refer to these as reals.

The truncation of a number to N digits introduces the so-called *round-off error*, which is essentially always present in the calculation carried out by a computer. This error affects not only the representation of numbers, but also the result of operations. For example, when two numbers are multiplied, the exact result of the multiplication may end up having more than N digits, but this has to be approximated by an N-digits number:

$$\left(S_1\, 0.m_1 \times 10^{E_1}\right)\, \left(S_2\, 0.m_2 \times 10^{E_2}\right) \,=\, (S_1 S_2)\, \underbrace{(0.m_1 \times 0.m_2)}_{more\ than\ N\ digits} \times 10^{E_1+E_2} \rightarrow (S_1 S_2)\, \underbrace{(0.m_1 \times 0.m_2)}_{N\ digits} \times 10^{E}\,. \tag{1.1.2}$$

The largest number that can be represented is determined by the largest exponent that the computer can store. A calculation producing a number larger than this causes *overflow*. Similarly, the largest (in modulus) negative exponent determines the smallest number that the computer can recognize (other than 0). This smallest number is the *machine epsilon*. A calculation producing a result smaller than this minimum (in modulus) produces *underflow*. These largest and smallest numbers depend on whether single, double or quadruple precision (or `float`, `double` and `long double` in C) are adopted.[5] Table 1.1.1 shows the range of numerical values available for numbers of type `float` (32 bits, or 4 bytes) and `double` (64 bits or 8 bytes). The reason why the largest exponent for the `float` type, for example, is 38, rather than, say, 99, is that numbers are represented in the binary system as explained below; more on this topic will be found in section 1.2.

	32 bit (`float`)	64 bit (`double`)
minimum	$1.17549435 \times 10^{-38}$	$2.2250738585072014 \times 10^{-308}$
maximum	$3.40282347 \times 10^{38}$	$1.7976931348623157 \times 10^{308}$

Table 1.1.1: Maximum and minimum numbers available with 32 and 64 bits, (called `float` and `double` in the C language).

The same problem illustrated above in the case of multiplication occurs in case of division and also with addition and subtraction. Consider for example the addition of two numbers with very different orders of magnitude, such as

$$4000 + 0.00044 \,=\, (0.4 + 0.000000044) \times 10^4 \,=\, 0.400000044 \times 10^4\,. \tag{1.1.3}$$

If the computer can, for example, only carry significands with 7 or fewer digits (typical for single-precision calculations), the result would be truncated to 0.4×10^4. If, at the next step, we were to subtract 3999.995 we would then find 0.5×10^{-2} in place of the exact result 0.544×10^{-2}, an error of nearly 9%. Thus, the associative property of addition, namely the fact that $(a+b)+c = a+(b+c)$, may fail in certain conditions. A better way to carry out this calculation would be to re-arrange the order of the operations as

$$\begin{aligned} (4000 - 3999.995) + 0.00044 \;&=\; (0.4 - 0.3999995) \times 10^4 + 0.44 \times 10^{-3} \,=\, 0.5 \times 10^{-2} + 0.44 \times 10^{-3} \\ &=\; (0.5 + 0.044) \times 10^{-2} \,=\, 0.544 \times 10^{-2}\,. \end{aligned} \tag{1.1.4}$$

For the same reason, the distributive property $a(b+c) = ab+ac$ may also fail to be satisfied. These examples show that, in designing a calculation, it is important to keep an eye on the orders of magnitude of the various

[5] Intel processors internally use an 80-bit floating-point format for all operations but, unless variables are declared as `long double`, this is not usually apparent to the user.

quantities so that the order of operations can be chosen so as to maximize accuracy. Similar failures of the associative and other properties may affect multiplication and other elementary operations.

A computer can only execute the four elementary operations: multiplication, division, addition and subtraction. Operation such as the calculation of a square root or of a trigonometric function are reduced to a combination of the four elementary operations. Even if the methods used for these calculations are highly accurate, since π cannot be represented exactly, $\sin \pi$, for example, will not vanish, but approximately equal to 0.1225×10^{-15} in double precision, or -0.8742×10^{-7} in single precision. Thus it can be stated that, save perhaps for a very small number of very special situations, all numerical results produced by a computer will be affected by round-off error.

A calculation involving a large number of operations will be affected by this error at every step. It is therefore of crucial importance that the algorithms (that is, the procedures) adopted to carry out calculations be designed in such a way as to avoid the accumulation of errors. This is an important point on which we will return more than once.

1.1.1 Rounding rules

Rounding is fairly straightforward when the last digit that needs to be dropped is between 1 and 4 or between 5 and 9. In both cases the last digit is dropped and, in the former case, the previous digit is left unchanged while, in the latter case, it is augmented by one. For example, to four significant digits, 122.34 is rounded to 122.3 while 122.36 is rounded to 122.4. Similarly, if 122.338 needs to be rounded to 4 digits it becomes 122.3, while 122.581 will become 122.6; this is called *round to nearest*.

The many different rules proposed for rounding address the case in which the digit to be dropped is a 5. This difficulty only arises if there is no information beyond the 5 as, when this is available, the round-to-nearest rule provides a unique answer. For example, 122.3501, truncated to 4 digits, would become 122.4 because the last 1 pushes the number past the midpoint. The real issue is what to do when there is no information available on digits beyond the 5 that needs to be dropped. The problem arises because of the danger of biasing the result of a sequence of calculations: it must be avoided that, due to the adopted rounding procedure, the final result of a sequence of operations drifts up or down off the correct value.

To avoid the situation in which the same code run on different computers produces different results, for rounding as well as other operations, all computer manufacturers now adopt the IEEE 754 Standard for Floating-Point Arithmetic. The rounding rule adopted in this standard is called *round half to even.* The rule is best explained when the 5 is the only digit beyond the decimal point. In this case the number is rounded to the closest even integer so that, for example, +23.5 becomes +24, as does +24.5, while -23.5 becomes -24, as does -24.5. To see what to do in a more general case it is sufficient to rewrite the number in this form by using powers of ten. For example, to see how to round 1.2345 to four digits, we write $1.2345 = 1234.5 \times 10^{-3}$ and apply the previous rule to 1234.5 finding 1234. Thus $1.2345 = 1234.5 \times 10^{-3} \rightarrow 1234 \times 10^{-3} = 1.234$. Similarly $3.1335 = 3133.5 \times 10^{-3} \rightarrow 3134 \times 10^{-3} = 3.134$. A quicker way to describe this rule is to say that the rounding makes the significand end with an even digit (including zero).

Another option provided by the IEEE 754 Standard, which is optional for decimal calculations is round to nearest away from zero. With this rule +24.5 is rounded to +25, instead of +24, and -24.5 is rounded to -25, instead of -24. Other options are provided as well. One use of different rounding options is to test the robustness, or stability, of the components of a code: if the results are significantly affected by the rounding chosen, chances are that there is a coding error, or that the method chosen for the calculation is unstable, or that the problem is ill-conditioned as posed.

The IEEE 754 Standard requires correct rounding in the sense that the rounded result should be the same as if the rounded value was obtained by rounding the result of infinitely precise arithmetic; in practice this requirement can be fulfilled by carrying only three extra bits in the calculation.

1.1.2 Examples

1. The consequences of round-off errors can be illustrated with an interesting example. It is a known fact that[6]

$$\Sigma_n \;=\; 1+\frac{1}{2}+\frac{1}{3}+\frac{1}{4}+\ldots+\frac{1}{n} \;\simeq \log n+\gamma+\ldots\,, \tag{1.1.5}$$

where log is the logarithm to the base e and $\gamma = 0.57721\,56649\ldots$ is Euler's constant. The series is, therefore, divergent as $n \to \infty$. Suppose to have access to an extremely lousy computer that can only keep 1 digit. That computer can represent the first term $1 = 0.1 \times 10^1$ and the second term $0.5 = 0.5 \times 10^0$ separately, but when it comes to putting them together to form

$$\Sigma_2 \;=\; 1+0.5 \tag{1.1.6}$$

it will round half to even and return $2 = 0.2 \times 10^1$ instead of $1.5 = 0.15 \times 10^1$ since representing the exact result 1.5 needs 2 digits. If we add one more term we would have $2 + 1/3 = 2.33333\ldots$ which would be rounded to 2 again, as if we we had added 0. The same will happen as we add one by one all the following terms since our computer can only produce a sum correct to one digit. Thus, for all intents and purposes, things go as if we were adding just zeros.This is similar to what was illustrated before in connection with (1.1.3). If our computer were to carry 2 digits it could correctly calculate the sum for a few more terms, if it carried 3 digits yet a few more and so on but, no matter how many digits the computer can carry, from some point on, adding new terms will effectively have the same effect as adding zeros. This is remarkable: $\log n$ increases without bound as n increases, but no computer will ever be able to prove this just by naively adding terms to the series. More ingenious ways to proceed (or more mathematics) are needed.

h	$f(\pi/6+h)$, 6 digits	$f(\pi/6+h)$, 3 digits	$f'_{approx}(\pi/6)$	error
0.2	-0.749428	-0.749	0.540	17%
0.02	-0.855853	-0.856	0.500	0%
0.002	-0.865024	-0.865	0.500	0%
0.0002	-0.865925	-0.866	0.0	∞
0.00002	-0.866015	-0.866	0.0	∞

Table 1.1.2: the results of an attempt to calculate a derivative by taking the increment smaller and smaller in the presence of round-off to 3 significant digits; see Example 2.

2. As another example let us consider the calculation of a derivative which, mathematically, is defined as

$$f'(x) \;=\; \lim_{h\to 0}\frac{f(x+h)-f(x)}{h}\,. \tag{1.1.7}$$

What happens if we try to calculate the limit numerically by taking h smaller and smaller? A moment's thought immediately tells us that this would be a bad idea because, if h is so small that our computer is unable to represent $f(x+h)$ as being different from $f(x)$, the result will be $f' = 0$! Consider for example $f(x) = -\cos x$. The exact derivative at $\pi/6$ is $f'(\pi/6) = \sin \pi/6 = \frac{1}{2}$. Let us try to calculate it as

$$f'_{approx}(\pi/6) \;=\; \frac{f(\pi/6+h)-f(\pi/6)}{h} \tag{1.1.8}$$

as we take h smaller and smaller. Suppose that we have a very bad computer with only 3 significant digits. In this computer $f(\pi/6) = 0.866025\ldots$ will simply be represented as $f(\pi/6) = 0.866$. Then we find the results in the table. The first column shows $f(\pi/6+h)$ correct to 6 digits and the second

[6] We do not need to show this, just accept it. A consideration that makes it plausible is that the sum resembles the integral of $1/x$ which is, of course, $\log x$.

one $f(\pi/6+h)$ correct to 3 digits, which is all that our computer can represent. The table displays a typical trend: first the error decreases as h is taken smaller and smaller, but then it increases as the number of digits carried is not sufficient to calculate accurately the difference $f(\pi/6+h)-f(\pi/6)$.

1.2 Binary and hexadecimal representation of numbers

It is evident that it is much easier to detect whether, for example, a capacitor is charged or not than to measure the charge on it, or whether a noisy line (electrical or optical) is transmitting a signal or not, rather than decoding the signal, whether a gate is open or closed, and so on. These considerations point to the desirability of building a computer on a system of 'yes' and 'no' or, in digital form, 0's and 1's. This information can readily be stored with bi-stable (or "flip-flop") components as implemented in the CMOS (complementary metal-oxide-semiconductor) integrated circuits, circuit elements that can exist in two different states, with one state representing 0 and the other one representing 1.

A *bit* (*bi*nary digi*t*) can be a 0 or a 1 and is the smallest unit of information that can be stored in a computer; 8 bits constitute a *byte*, which is typically used to represent characters such as numerical digits, letters, punctuation marks etc. In acronyms bits are denoted by a lower case "b" and bytes by an upper case "B". Thus, for example, 10 MB of storage denotes an amount of storage equivalent to ten mega-bytes, or 10 million bytes, while a transmission line having a speed of 10 Mbit/s (or Mb/s, often abbreviated Mbps) is capable of carrying 10 million bits per second. By encoding/interpreting sets of bits in various ways, computers represent and manipulate numbers, sets, strings, etc. and determine what to do (instructions).

Since a bit can take only two states, it is necessary to use binary representation for numbers.[7] The way this can be done becomes clearer by re-considering the familiar decimal representation, or representation to the base (or radix) 10. The table shows how we can interpret a number written in the usual base-10 notation, e.g. 513: In words, 513 is constructed by taking 5 one-hundreds, 1 ten, and 3 ones. When we write 513,

...	$10^4=$	$10^3=$	$10^2=$	$10^1=$	$10^0=$	$10^{-1}=$	...
...	10,000	1000	100	10	1	0.1	...
	0	0	5	1	3	2	

Table 1.2.1: The number 513 "unpacked" in the decimal representation.

we place the number of hundreds (5) first, then the number of tens (1) and then the number of ones (3). If there was a decimal digit, e.g. 513.4, the 4 after the period is the number of 0.1's and so forth:

$$513.4_{10} = 5\times 10^2 + 1\times 10^1 + 3\times 10^0 + 4\times 10^{-1}\,. \tag{1.2.1}$$

This is the *positional notation* or *place-value notation* for the representation of a number. As opposed to other notations (e.g., roman numerals) this notation greatly simplifies arithmetic manipulations. A systematic way to construct this representation is the following:

1. Consider the number at hand, 513 in this case, and see what is the largest power of 10 smaller than it, in this case $10^2=100$. Count how many of these 10^2's "fit" in 513; in this case 5;

2. Calculate $513-5\times 10^2=13$. Count how many 10^1's fit in 13; the answer is 1;

3. Calculate $513-5\times 10^2-1\times 10=3$. Count how many 10^0's fit in 1; the answer is 3;

4. Write the results in the order 5, then 1, then 3, i.e., 513.

[7] The binary representation is also used in computer graphics, e.g., or games. Computers generate color pictures on a video screen or liquid crystal display by mixing the three fundamental colors red, green, and blue. Each *pixel* (*pic*ture *el*ement) is turned on or off depending on the color required.

If it is necessary to specify the base (or representation) used in writing a number, it is customary to add it as a subscript. So, in this case, we have written 513_{10}.

This seems an overkill since we are so accustomed to representing numbers to the base 10, but once we understand the procedure we can apply it to represent numbers to any base, such as the base 2 used for the binary representation of numbers. For example, let us see how to represent in base 2 a number given by 184_{10} to base 10; the construction is shown in Table 1.2.2; in words, the idea is the following:

1. Consider the number at hand, 184_{10} in this case, and see what is the largest power of 2 smaller than it, in this case $2^7 = 128$, which "fits" into 184_{10} 1 time;

2. Calculate $184 - 1 \times 2^7 = 56$. Count how many 2^6's fit in 56; the answer is 0;

3. Calculate $184 - 1 \times 2^7 - 0 \times 2^6 = 56$. Count how many 2^5's fit in 56; the answer is 1;

4. Calculate $184 - 1 \times 2^7 - 0 \times 2^6 - 1 \times 2^5 = 24$. Count how many 2^4's fit in 24; the answer is 1;

5. Calculate $184 - 1 \times 2^7 - 0 \times 2^6 - 1 \times 2^5 - 1 \times 2^4 = 8$. Count how many 2^3's fit in 8; the answer is 1;

6. Calculate $184 - 1 \times 2^7 - 0 \times 2^6 - 1 \times 2^5 - 1 \times 2^4 - 1 \times 2^3 = 0$. Count how many 2^2's fit in 0; the answer is 0, and so it is for 2^1, 2^0 as well;

7. Hence the binary representation of 184_{10} is $[10111000]_2$

Hence, a binary number $A_2 = a_2 a_1 a_0 \, . \, a_{-1} a_{-2}$, with the a_j's all 0's or 1's, corresponds to the decimal

$$A_{10} \; = \; \sum_{j=-2}^{2} a_j \times 2^j \; = \; a_{-2} \times 2^{-2} + a_{-1} \times 2^{-1} + a_0 \times 2^0 + a_1 \times 2^1 + a_2 \times 2^2 \, . \tag{1.2.2}$$

For example, $A_2 = [101.11]_2$ represents the number $1 \times 2^{-2} + 1 \times 2^{-1} + 1 \times 2^0 + 0 \times 2^1 + 1 \times 2^2 \; = \; \frac{1}{4} + \frac{1}{2} + 1 + 4 \; = \; 5 + \frac{3}{4} = 5.75_{10}$.

...	$2^7 =$	$2^6 =$	$2^5 =$	$2^4 =$	$2^3 =$	$2^2 =$	$2^1 =$	$2^0 =$	...
...	128	64	32	16	8	4	2	1	...
	1	0	1	1	1	0	0	0	

Table 1.2.2: Construction of the binary representation of the number 184_{10}.

Just as $\frac{1}{3}$ cannot be represented exactly with a finite number of digits in a decimal system, there are numbers that cannot be represented exactly with a finite number of digits in a binary system; an example is $1.1_{10} = [1.000110011\ldots]_2$. This fact has some consequences that at first sight are unexpected. Consider the piece of pseudocode

```
x = 1.1
print "x =", x
```

Implementing this in single precision, on some machines, and with a relatively small number of digits, will produce the output `x=1.10000002`. If `x` is declared `double precision`, printing a small number of digits, in exponential notation, produces `x=0.11000000E+01`, but printing more digits gives `x=0.110000002384E+01`. Hence numbers are rounded also in binary representation. The only numbers that have a finite binary representation are rationals with denominators that are powers of 2, such as 1/2 or 3/16. Any rational with a denominator that has a prime factor other than 2 will have an infinite binary expansion. This means that numbers which appear to be short and exact when written in decimal format may need to be approximated when converted to binary floating-point.

Precision	Total no. of bits	Sign	Exponent	Significand
Single	32	1	8	23+1
Double	64	1	11	52+1

Table 1.2.3: Bits assigned to represent the different parts that constitute the binary representation of a number in single or double precision according to the IEEE 754 Standard. The +1 is the implicit bit; see text.

1.2.1 Normalized binary numbers

We start by mentioning a relation extensively used in the following:

$$S_M(x) \,=\, 1 + x^1 + x^2 + \ldots + x^{M-1} \,=\, \sum_{j=0}^{M-1} x^j \,=\, \frac{1-x^M}{1-x}\,. \tag{1.2.3}$$

The proof is very simple and very elegant: we note that $xS_M \,=\, S_M - 1 + x^M$ and solve for S_M to find the result. In particular we have

$$S_M(2) \,=\, 2^M - 1\,, \qquad S_M(1/2) \,=\, \frac{1-1/2^M}{1-1/2} \,=\, 2\left(1 - \frac{1}{2^M}\right)\,. \tag{1.2.4}$$

It is said that a binary number is normalized when it has the form $1.abcde\ldots f \times 2^E$, with $a,\, b, \ldots,\, f$ 0's or 1's and E the exponent, also expressed to base 2.[8] This expression is to be interpreted as

$$\left(1 + \frac{a}{2} + \frac{b}{2^2} + \frac{c}{2^3} + \frac{d}{2^4} + \frac{e}{2^5} + \ldots\right) \times 2^E\,. \tag{1.2.5}$$

Since, with this normalized representation, the first digit is always a 1 (but see below for denormalized numbers), in order to save bits its explicit representation is omitted (this is the leading bit, or implicit bit, or hidden bit convention). For example, with 3+1 bits (with the +1 indicating omission of the implied 1) and omitting the leading 1, if $E = 0$ we have the numbers:

$$\begin{aligned}
[000] \,&=\, 1 & [100] \,&=\, 1 + \tfrac{1}{2} + \tfrac{0}{4} + \tfrac{0}{8} \,=\, 1 + \tfrac{4}{8} \\
[001] \,&=\, 1 + \tfrac{0}{2} + \tfrac{0}{4} + \tfrac{1}{8} \,=\, 1 + \tfrac{1}{8} & [101] \,&=\, 1 + \tfrac{1}{2} + \tfrac{0}{4} + \tfrac{1}{8} \,=\, 1 + \tfrac{5}{8} \\
[010] \,&=\, 1 + \tfrac{0}{2} + \tfrac{1}{4} + \tfrac{0}{8} \,=\, 1 + \tfrac{2}{8} & [110] \,&=\, 1 + \tfrac{1}{2} + \tfrac{1}{4} + \tfrac{0}{8} \,=\, 1 + \tfrac{6}{8} \\
[011] \,&=\, 1 + \tfrac{0}{2} + \tfrac{1}{4} + \tfrac{1}{8} \,=\, 1 + \tfrac{3}{8} & [111] \,&=\, 1 + \tfrac{1}{2} + \tfrac{1}{4} + \tfrac{1}{8} \,=\, 1 + \tfrac{7}{8}\,.
\end{aligned} \tag{1.2.6}$$

Thus, these numbers fill the range from 1 to 2 with a regular spacing of $2^{4-1} = 1/8$. If $E = 1$ the numbers will fill the range between 2 and 4 with a spacing of 1/4 while, for $E = -1$, they will fill the range between 1/2 and 1 with a spacing of 1/16. Thus the spacing between consecutive numbers changes every time the exponent changes.

According to the IEEE 754 Standard, as shown in Table 1.2.3, the normalized representation of a binary number in single precision uses one bit for the sign S (0 for plus and 1 for minus), 8 bits for the exponent and 23+1 bits for the significand. Thus, in single precision, a number would be represented as

$$\underbrace{S}_{1\text{ bit}} \quad \underbrace{E_0E_1E_2E_3E_4E_5E_6E_7}_{8\text{ bits}} \quad \underbrace{M_1M_1M_2M_3M_4M_5M_6M_7M_8M_9MMMMMMMMMMM_{20}M_{21}M_{22}M_{23}}_{23\text{ bits}}\,. \tag{1.2.7}$$

[8] Without loss of generality, the digit to the left of the period can always be taken to be 1 by adjusting the exponent. For example $[63/64]_{10} = [0.111111]_2 = [1.11111 \times 2^{-1}]_2$.

S, as well as all the E's and all the M's are 0 or 1. Depending on the number to base 10 being represented, this translates to 6-8 decimal digits of precision.

To base 10, the bits forming the exponent are to be interpreted as

$$E_7 2^7 + E_6 2^6 + \ldots + E_1 2^1 + E_0 2^0 \,. \tag{1.2.8}$$

Let us distinguish three cases:

1. When not all the E's vanish and they are not all equal to 1, the possible values of the exponent range from 1 to 254, as shown from the first one of (1.2.4) with $M = 8$. These values are converted to actual exponents by subtracting the *bias* $Bi = 2^7 - 1 = 127$. With this convention the true exponent ranges from the minimum value
$$2^0 - 127 \;=\; -126\,, \tag{1.2.9}$$
to the maximum value
$$2^7 + 2^6 + \ldots + 2^1 + 0 \times 2^0 - 127 \;=\; +127\,. \tag{1.2.10}$$
For double precision the bias is 1,023 and the range between -1,022 and +1,023; for quadruple precision the bias is 16,383 and the range between -16,382 and +16,383.

2. When the exponent field is all zeros, it is said that the represented number is *denormalized*. In this case, the real exponent value is taken as $E = 1 - Bi$, and the implied bit is set to 0, rather than 1. In this way it is possible to represent the number 0 (for which all bits are 0) and numbers that are very close to 0.

3. When all the E's equal 1, there are two cases, either $\pm\infty$ or NaN. The former case, which denotes overflow, occurs when all the M's equal 0, with the sign determined by the first bit. The latter case, in which NaN stands for "not a number," occurs when the result of an operation cannot be given as a real number or as infinity, e.g. when attempting to compute $0/0$ or $\sqrt{-1}$.

The digits indicated by M are used to form the normalized significand as in (1.2.5):

$$1 + \frac{M_1}{2^1} + \frac{M_2}{2^2} + \ldots + \frac{M_{22}}{2^{22}} + \frac{M_{23}}{2^{23}}\,; \tag{1.2.11}$$

the first 1 is the implicit digit as mentioned above. The largest such number is obtained when all the M's equal 1 and, from the second one of (1.2.4) with $M = 24$, is

$$1 + \frac{1}{2} + \frac{1}{2^2} + \ldots + \frac{1}{2^{23}} \;=\; 2\left(1 - \frac{1}{2^{24}}\right) \;=\; 2\left(1 - \frac{1}{16777216}\right)\,. \tag{1.2.12}$$

To base 10, the largest number that can be represented with 32 bits is therefore

$$2\left(1 - \frac{1}{16777216}\right) \times 2^{127} \;\simeq\; 3.40282347 \times 10^{38}\,, \tag{1.2.13}$$

while the smallest normalized number is

$$1 \times 2^{-126} \;\simeq\; 1.17549435 \times 10^{-38}\,. \tag{1.2.14}$$

A similar procedure gives the largest and smallest numbers for double precision, which has an approximate precision of 15 decimal digits. This is how the values displayed in Table 1.1.1 are calculated.

The smallest difference between two consecutive numbers representable to base 2 and having a 0 exponent is clearly $1/2^{N-1}$. For example, with $N = 24$, we have

$$\left(1 + \frac{1}{2} + \ldots + \frac{1}{2^{22}} + \frac{1}{2^{23}}\right) - \left(1 + \frac{1}{2} + \ldots + \frac{1}{2^{22}} + \frac{0}{2^{23}}\right) \;=\; \frac{1}{2^{23}} \;\simeq\; 1.19 \times 10^{-7}\,. \tag{1.2.15}$$

Similarly, in double precision, we find $1/2^{53} \simeq 1.11 \times 10^{-16}$. If the exponent is $E = 1$, in single precision this doubles to $1/2^{23} \times 2^1 \simeq (1.19 \times 10^{-7}) \times 2^1 \simeq 2.38 \times 10^{-7}$, and so on as the exponent increases: every time the exponent increases by 1, the spacing doubles. Conversely, the spacing of consecutive numbers decreases as the exponent becomes more and more negative although, of course, there is a smallest possible value for the exponent. Below this minimum value the spacing is dictated by the normalized numbers and can be shown to remain constant. It is therefore evident that on a computer the spacing of representable numbers is not constant.

The difference between two consecutive representable floating-point numbers with exponent 0 is clearly significant as all others are multiples or submultiples of it. One half of this quantity is called *machine epsilon*. The mandated behavior of IEEE-compliant hardware is that the result of rounding be always within machine epsilon multiplied by the relevant power of 2.

1.2.2 Hexadecimal representation

Rather than accessing individual bits in memory, most computers use blocks of eight bits, or bytes, as the smallest addressable memory unit.

Since a byte is a group of 8 bits, in binary representation a byte varies from $[00000000]_2$ to $[11111111]_2$ which, in decimal representation, is 0_{10} and 255_{10}. (In any base, of course, the representation of 0 is 0.) A convenient way to represent bytes is to use the *hexadecimal* (or *hex*) representation, namely a representation based on 16 rather than 10 (for decimal) or 2 (for binary). Just as the familiar decimal representation needs the ten symbols 0, 1, 2, ..., 9, the hexadecimal representation needs 16 symbols: in addition to the standard 10 digits 0, 1, 2, ..., 9, the additional 6 are denoted by the letters A, B, C, D, E, F which represent the decimal numbers from 10 to 15: $A = 10_{10}$, $B = 11_{10}$, $C = 12_{10}$, $D = 13_{10}$, $E = 14_{10}$, $F = 15_{10}$.

Hex digit	0	1	2	3	4	5	6	7
Decimal value	0	1	2	3	4	5	6	7
Binary value	0000	0001	0010	0011	0100	0101	0110	0111
Hex digit	8	9	A	B	C	D	E	F
Decimal value	8	9	10	11	12	13	14	15
Binary value	1000	1001	1010	1011	1100	1101	1110	1111

Table 1.2.4: Correspondence between hex digits, decimals and binaries.

Table 1.2.4 shows the conversion between the hex digits and the corresponding binary and decimal values. Using this table it is easy to convert from hexadecimal to binary; for example $[26BC]_{16} = [0010\,0110\,1011\,1100]_2$. Conversely, a binary number can be converted to hexadecimal by dividing its digits in groups of 4 starting from the right; if the last group has less than 4 digits, the missing ones are interpreted as 0. Thus, for example, $[110100101011010]_2 = [0110\ 1001\ 0101\ 1010]_2 = [695A]_{16}$.

...	$16^3 =$	$16^2 =$	$16^1 =$	$16^0 =$	...
...	4096	256	16	1	...
	0	0	$15 = F$	$15 = F$	

Table 1.2.5: Construction of the hexadecimal representation of the number 255_{10}.

We can construct the hexadecimal representation of 255, for example, just as before according to Table 1.2.5. The result is then FF_{16}. Thus the range of 0_{10} to 255_{10} is equivalent to 0_{16} to FF_{16} in hexadecimal representation. As another example $[3B2]_{16} = (3 \times 16^2) + 11 \times 16^1 + 2 \times 16^0 = 768 + 176 + 2 = 946_{10}$.

On a computer base 16 is primarily used to represent memory addresses because it can represent every byte as two consecutive hexadecimal digits instead of the eight digits that would be required by the binary representation.

1.2.3 Integers

Unlike reals, in a computer integers can be represented exactly. Unsigned integers are assumed to be positive and, since there is no need to specify the sign, their representation can take advantage of one bit more than for signed integers. The relation between the binary and the decimal representation is the same as in (1.2.8) so that, for example,

$$[1011]_2 \;=\; 1\times 2^3 + 0\times 2^2 + 1\times 2^1 + 1\times 2^0 \;=\; 8+0+2+1 \;=\; 11_{10}\,. \tag{1.2.16}$$

On a 32-bit machine the maximum integer value that can be represented is $\sum_{j=0}^{31} 2^j = 2^{32}-1 = 4,294,967,295$ or 4 GB, as can be deduced from (1.2.4).

The encoding used for signed integers is called *Two's-complement encoding*: the first bit, corresponding to the coefficient of the power 2^{N-1}, with N the number of bits, encodes the sign, plus if it is 0, minus if it is 1:

$$[0110]_2 \;=\; 0\times 2^3 + 1\times 2^2 + 1\times 2^1 + 0\times 2^0 \;=\; 6_{10}\,, \tag{1.2.17}$$

but

$$[1110]_2 \;=\; -1\times 2^3 + 1\times 2^2 + 1\times 2^1 + 0\times 2^0 \;=\; -2_{10} \tag{1.2.18}$$

Thus, on a 32-bit machine, the largest positive number is represented in binary as $[0111\ldots111]_2$ or, in decimal notation, $\sum_{j=0}^{30} 2^j = 2^{31}-1 = 2,147,483,647$, i.e., 2 GB, or half of the maximum unsigned integer value. The smallest number that can be represented is $[1000\ldots000]_2$ or $-2^{31} = -2,147,483,648$, the absolute value of which is greater by 1 than the maximum value.[9] It interesting to note that, since from the first one of (1.2.4),

$$\sum_{j=0}^{N-1} 2^j \;=\; 2^N - 1\,, \tag{1.2.19}$$

we have

$$-1 \;=\; -2^N + \sum_{j=0}^{N-1} 2^j \tag{1.2.20}$$

which corresponds to the binary number $[1111\ldots111]_2$.

1.3 Strings

A string is a sequences of characters such as digits, letters, punctuation marks, whitespaces and also control characters such as carriage return (or `tab`), as well as instructions to printers or other devices. Each character is represented according to some standard encoding, the most common one being the ASCII (American Standard Code for Information Interchange) character code. The need for special characters, common in languages other than English, has prompted the proposal of many different encodings until the Unicode Consortium developed the Unicode Standard, which includes over 130,000 characters and which is almost universally adopted today. Unicode provides a unique number for every character, no matter what platform,

[9] We can now understand the origin of the denomination "Two's-complement encoding," which derives from the fact that the binary representations of two numbers x and $-x$ add up to 2^N. For example, with $N=3$, $x=2$ is represented as $[010]_2$, while $-2 = -4+2+0$ is represented as $[110]_2$. If we now forget the special interpretation of the first bit and consider both numbers as positive we find $[010]_2 + [110]_2 \;=\; 2+(2^2+2) = 8 = 2^3$. Since $\sum_{j=0}^{N-1} 2^j \;=\; 2^N - 1$, it is evident that $\sum_{j=0}^{N-1} a_j 2^j + \sum_{j=0}^{N-1}(1-a_j)2^j + 1 = 2^N$. Thus the simple rule to find the binary representation of the negative number $-x$ is to write that of the positive number x, change every 1 to a 0 and every 0 to a 1, and add 1.

Code	Decimal	Description	Code	Decimal	Description
U+0030	48	Digit 0	U+0000	0	Null character
U+0031	49	Digit 1	U+0008	8	Backspace
U+0041	65	Capital *A*	U+000A	10	Line feed
U+0042	66	Capital *B*	U+000D	13	Carriage return
U+005A	90	Capital *Z*	U+0011	17	Device Control 1
U+0061	97	Lower-case *a*	U+0021	33	! Exclamation mark
U+007A	122	Lower-case *z*	U+0024	36	$ Dollar sign
U+007E	126	$\sim$ Tilde	U+0025	37	% Percent
U+00A8	0168	¨ Diaeresis	U+0028	40	(Left Parenthesis
U+00B1	0177	$\pm$	U+002F	47	/ Slash

Table 1.3.1: Unicode representation of some letters, numbers, symbols and control characters. The number following $U+$ is the hexadecimal representation of the byte corresponding to the character; note that, in the examples shown, only one byte (two digits) is used.

device, application or language. This permits data to be transported through many different platforms, devices and applications without corruption.

The base encoding, known as the "Universal Character Set" of Unicode, uses a 32-bit (4 bytes) representation of characters. The UTF-8 representation, based on Unicode (Unicode Transformation Format – 8-bit), encodes each character as a sequence of bytes, with common characters requiring just 1 or 2 bytes, while less common ones require more. For the 128 characters common to US-ASCII and UTF-8, the same single-byte encoding is used, so that a single-byte ASCII sequence has the same meaning in UTF-8. Some examples are shown in Table 1.3.1 (remember that the hexadecimal representation of one byte requires two hexadecimal digits). The next 1,920 characters, which cover the remainder of almost all Latin-script alphabets, diacritical marks and also Greek, Cyrillic and others, are encoded with two bytes. Less common characters and mathematical symbols are encoded with 4 bytes.

1.4 Information storage

Memory and its efficient use are crucial aspect of computing to which we will devote quite some attention later. For the time being let us just mention that, in essence, at the machine level, memory is a very large one-dimensional array of bytes, referred to as *virtual memory*. Every byte of memory is identified by a unique number, known as its *address* (or *pointer* in C), which is essentially the index of the corresponding array element. The addresses of successive words differ by 4 or 8 bytes, i.e., 32 or 64 bits.

The set of all possible addresses is known as the *virtual address space*. If an object (data, instruction, etc.) occupies more than one byte, in virtually all machines it is stored as a contiguous sequence of bytes, with the address of the object given by that of the first byte. At the machine level, everything – be it data, instructions, controls – is a sequence of bytes.

The operating system (OS) provides private address spaces to each *process* (instance of a computer program) that is being executed. In this way, different processes cannot interfere with one another.

The *word* is a fixed-sized piece of data handled as a unit by the instruction set or the hardware of the processor. The number of bits in a word is the *word size* which, in today's computers, is almost invariably 32 bits or 64 bits. In a machine having a word size of w bits the virtual addresses can range from 0 to $2^w - 1$ so that, at most, a program can access 2^w bits. As we have seen before, for a 32-bit word this amounts to 4 GB of virtual address space. A 64-bit word machine has a virtual address space of about 18.447×10^{18} bytes, or 18 EB (exabytes).

Machines allow the use of fractions and multiples of their word size, provided they correspond to an integral number of bytes. The number of bytes typically allocated to the representation of different data

C Data Type	MATLAB	Typical 32-bit	Typical 64-bit	x86-64
`char`	`char`	1	1	1
`short`		2	2	2
`int`	`int32`	4	4	4
`long`	`int64`	4	8	8
`float`	`single`	4	4	4
`double`	`double`	8	8	8
`long double`				10/16
`char *`		4	8	8

Table 1.3.2: Number of bytes allocated to the representation of different data types in C and MATLAB on 32-bit and 64-bit machines; x86-64 is the 64-bit version of the x86 instruction set (see `https://en.wikipedia.org/wiki/X86-64`); in C the data type `char *` corresponds to pointers.

types in C and MATLAB is shown in Table 1.3.2.[10] It will observed, for example, that the representation of pointers (`char *` for a pointer referring to a data type `char`), uses the entire word length, while reals of type `double` use the entire word length in 64-bit machines and two words in 32-bit machines.

The ordering of the bytes specifying the address of a multi-byte word can vary from machine to machine, but it is usually transparent to the programmer and therefore we do not dwell on this topic here.

[10] This is not standardized and may vary between compilers and installations; a careful programmer will always use `sizeof(T)` when in doubt; this function returns the size of an object of type `T`.

Chapter 2

Numerical Differentiation

2.1 The basic idea

A computer cannot "understand" or evaluate the writing $f'(x)$ to denote the first derivative of a function $f(x)$ at the point x. [11] How do we instruct it to produce at least a reasonably accurate numerical approximation to it?

Mathematically, the first derivative is defined as

$$f'(x) \;=\; \lim_{h\to 0} \frac{f(x+h)-f(x)}{h}\,. \tag{2.1.1}$$

The definition is unambiguous provided the limit has the same value however h approaches 0, e.g. through positive values, negative values, or jumping between positive an negative values in any way. [12] The computer cannot calculate the actual limit, as there is a finite smallest number that it can deal with. But we may hope that, if we take h small enough, we can find a decent approximation to $f'(x)$. How to test this hypothesis?

A reasonable expectation is that, if we take h "small enough" without passing to the limit we will have a "reasonable approximation" to $f'(x)$:

$$f'(x) \;\simeq\; \frac{f(x+h)-f(x)}{h}\,. \tag{2.1.2}$$

We can re-write this as

$$f(x+h) \;\simeq f(x) + hf'(x)\,. \tag{2.1.3}$$

Upon comparing with the exact formula

$$f(x+h) \;=\; f(x) + \int_x^{x+h} f'(\xi)d\xi\,, \tag{2.1.4}$$

we see that (2.1.3) is equivalent to approximating the integral by a rectangle having base h and height $f'(x)$. How good is this approximation? Can we control and hopefully decrease the error? These are obviously crucial questions for numerical computation.

2.2 The Taylor theorem

The key to answering these questions is provided by *Taylor's theorem*, one of the most useful ideas in analysis: If the function $f(x)$ has k derivatives at a point x then there is a function $R_k(x,h)$, the remainder, such that

$$f(x+h) \;=\; f(x) + hf'(x) + \frac{1}{2!}h^2f''(x) + \frac{1}{3!}h^3f'''(x) + \ldots + \frac{1}{k!}h^kf^{(k)}(x) + h^kR_k(x,h)\,. \tag{2.1.5}$$

[11] There are now software packages (such as MATHEMATICA) which can deal with derivatives in the proper sense of mathematical analysis. This software can, for example, find the analytic solution of a differential equation when the equation is solvable. Since, however, the vast majority of problems are not solvable analytically, one has to recur to numerical computation and this is what we are referring to here.

[12] For example, if $f(x) = |x|$ and we consider $x = 0$ taking $h = 1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \ldots$ the limit has the value 1, but if we take $h = -1, -\frac{1}{2}, -\frac{1}{3}, -\frac{1}{4}, \ldots$ the limit would be -1, and if we were to take $h = 1, -\frac{1}{2}, \frac{1}{3}, -\frac{1}{4}, \ldots$ it would oscillate between ± 1. Thus, the function $|x|$ is not differentiable at $x = 0$ (although it possesses one-sided derivatives at this point).

and

$$\lim_{h\to 0} R_k(x,h) \;=\; 0\,, \tag{2.1.6}$$

so that the error incurred by stopping at the k-the derivative decreases *faster* than h^k. If $f^{(k+1)}$ exists, it can be shown that

$$R_k(x,h) \;=\; \frac{h}{(k+1)!} f^{(k+1)}(x+\theta h)\,, \qquad 0\le\theta\le 1\,, \tag{2.1.7}$$

so that we are guaranteed that the remainder equals the $(k+1)$-th derivative at some intermediate point between x and $x+h$. The trouble is, of course, that, in general, θ, which depends on both x and h, is unknown. Note that the result in (2.1.5) and (2.1.7) is true even if one stops the sum before the maximum k. So, for example,

$$f(x+h) \;=\; f(x)+hf'(x)+\frac{h^2}{2!}f''(x+\theta_1 h)\,, \qquad f(x+h) \;=\; f(x)+hf'(x)+\frac{1}{2!}h^2f''(x)+\frac{1}{3!}h^3f'''(x+\theta_2 h)\,, \tag{2.1.8}$$

and so on. Notice that terms proportional to h^k are divided by $k!$, which is a very rapidly increasing function of k. Furthermore, as $h\to 0$, $h^k\to 0$ faster and faster as k increases. For these two reasons terms containing higher powers of h become less and less important in influencing the value of f at $x+h$ as h is taken smaller and smaller (unless, of course, successive derivatives become larger and larger).

If the function has an infinity of derivatives in a neighborhood of x, the sum can be extended all the way to infinity to form the *Taylor series*:

$$f(x+h) \;=\; \sum_{n=0}^{\infty}\frac{h^n}{n!}f^{(n)}(x) \;=\; f(x)+hf'(x)+\frac{1}{2!}h^2f''(x)+\frac{1}{3!}h^3f'''(x)+\dots\,. \tag{2.1.9}$$

This series may or may not converge to $f(x+h)$. If it does, it is said that the function is *analytic* near x.[13]

It can be shown that the series in the right-hand side of (2.1.9) can be differentiated term by term with respect to x. By doing so we find

$$f'(x+h) \;=\; f'(x)+hf''(x)+\frac{1}{2!}h^2f'''(x)+\frac{1}{3!}h^3f''''(x)+\dots\,, \tag{2.1.10}$$

which is just Taylor's formula (2.1.10) applied to f' rather than to f. Intuitively, we may say that the Taylor series is the result of the attempt to fit to $f(x+h)$ a polynomial in h of higher and higher order in a neighborhood of x. We would start by fitting the value at x, namely $f(x+h)\simeq f(x)$. Then we try to improve this crude approximation by fitting a straight line writing $f(x+h)\simeq f(x)+\alpha h$. We choose α in such a way that the derivative of the straight line is the same as the derivative of the function at $h=0$, which gives $\alpha=f'(x)$, and so on. [14] See `https://www.youtube.com/watch?v=3d6DsjIBzJ4&t=520s` for a nice illustration of the idea.

2.2.1 Examples

1. For what kind of functions f do the formulae (2.1.26) and (2.1.27) give an exact result for the derivative? To answer this question we note from (2.1.25) that the function f must have zero second (and higher) derivative, i.e., it has to be a linear function, e.g. $f(x)=ax+b$. Indeed, in this case (2.1.26) gives

$$\frac{f(x+h)-f(x)}{h} \;=\; \frac{[a(x+h)+b]-(ax+b)}{h} \;=\; a\,, \tag{2.1.11}$$

which is the exact result.

[13] It should be noted that infinite differentiability is not sufficient. For example, the function $\exp(-1/x^2)$ is infinitely differentiable at $x=0$, but all its derivatives vanish so that the sum of the Taylor series is also 0, while f does not vanish in a neighborhood of $x=0$.

[14] Note that we move along the straight line by varying h; x is the fixed point at which we want to approximate the derivative.

Let us apply the formula to $f(x) = x^2$:

$$\frac{f(x+h) - f(x)}{h} \;=\; \frac{(x+h)^2 - x^2}{h} \;=\; 2x + h\,. \tag{2.1.12}$$

The first term is the exact derivative, the second one the error, which we verify here to be of first order in h, i.e., proportional to h raised to the first power.

2. A very important Taylor series is that for the function $f(x) = 1/(1-x)$. We have

$$f'(x) \;=\; \frac{1}{(1-x)^2}\,, \qquad f''(x) \;=\; \frac{2}{(1-x)^3}\,, \tag{2.1.13}$$

etc. so that

$$\frac{1}{1-(x+h)} \;=\; \frac{1}{1-x} + \frac{h}{(1-x)^2} + \frac{1}{2}\frac{2h^2}{(1-x)^3} + \ldots\,. \tag{2.1.14}$$

In particular, with $x = 0$, this gives

$$\frac{1}{1-h} \;=\; 1 + h + h^2 + \ldots \tag{2.1.15}$$

which is known as the *geometric series*.

This series is quite useful in many applications. Suppose for example that we want to convert a temperature from °F to °C. The exact result is found in this way:

$$T(^\circ C) \;=\; \frac{T(^\circ F) - 32}{1.8}\,. \tag{2.1.16}$$

We rewrite this as

$$T(^\circ C) \;=\; \frac{T(^\circ F) - 32}{2 - 0.2} \;=\; \frac{T(^\circ F) - 32}{2(1-0.1)} \;=\; \frac{1}{2}[T(^\circ F) - 32]\frac{1}{1-0.1} \tag{2.1.17}$$

Now, from (2.1.15), $1/(1-0.1) \simeq 1 + 0.1$ so that

$$T(^\circ C) \;\simeq\; \frac{1}{2}[T(^\circ F) - 32] \times 1.1\,. \tag{2.1.18}$$

In words: to convert from °F to °C subtract 32, divide by 2 and add 10%. For example, for $T = 92$ °F, by applying this rule we find $T = 33$ °C, while the exact result is 33.33 °C. The same trick works if we want to convert a weight W expressed in kg to lb:

$$W(lb) \;=\; \frac{W(kg)}{0.454} \;\simeq\; \frac{W(kg)}{0.45} \;=\; \frac{W(kg)}{0.5 - 0.05} \;=\; \frac{W(kg)}{0.5(1-0.1)} \;\simeq\; 2W(kg) \times 1.1\,. \tag{2.1.19}$$

Again, double and add 10%.

3. The geometric series is a special case of the more general *binomial series*:

$$(1+h)^\beta \;\simeq 1 + \beta h + \ldots \tag{2.1.20}$$

for any real, positive or negative, β. If $\beta = 2$ or 3 we recover the leading terms of the expansions of $(1+h)^2$ and $(1+h)^3$.

4. As a final example let us take $f(x) = \cos \alpha x$. Then

$$f'(x) \;=\; -\alpha \sin \alpha x\,, \qquad f''(x) \;=\; -\alpha^2 \cos \alpha x\,. \tag{2.1.21}$$

Then

$$\cos[\alpha(x+h)] \;=\; \cos \alpha x + h\,(-\alpha \sin \alpha x) + \frac{1}{2!} h^2 \left(-\alpha^2 \cos \alpha x\right) + \ldots \tag{2.1.22}$$

Let us apply this at the point $x = 0$. Then we find

$$\cos \alpha h \;\simeq\; 1 - \frac{1}{2!} h^2 \alpha^2\,. \tag{2.1.23}$$

We see that this amounts to approximating the cosine function near the origin by a parabola. It is evident that the accuracy of the approximation will be very strongly dependent on the magnitude of α. A small α may give a reasonable result even with a fairy large h, while a large α will be unforgiving unless h is small enough. How good is this approximation to calculate the first point where $\cos \alpha x$ vanishes? The parabola in (2.1.23) vanishes for $h = \sqrt{2}/\alpha$ while $\cos \alpha h$ vanishes for $h = \pi/2\alpha$. How big is the error? Let's see:

$$\frac{\pi/2\alpha - \sqrt{2}/\alpha}{\pi/2\alpha} \;=\; \frac{\pi/2 - \sqrt{2}}{\pi/2} \;=\; 0.0997 \;\simeq 10\%\,. \tag{2.1.24}$$

Not too good, but not too bad either!

2.2.2 Back to $f'(x)$

By using the Taylor formula in (2.1.2) we find

$$\frac{f(x+h) - f(x)}{h} \;=\; \frac{f(x) + hf'(x) + \frac{1}{2!}h^2 f''(x) + \frac{1}{3!}h^3 f'''(x) + \ldots - f(x)}{h} \;=\; f'(x) + \frac{1}{2} h f''(x) + \frac{1}{3!} h^2 f'''(x) + \ldots\,. \tag{2.1.25}$$

This result supports our original expectation that, if h is taken small enough, (2.1.2) will give a decent approximation to $f'(x)$. Indeed, for h sufficiently small, all terms in the right-hand side other than $f'(x)$ will be small, and they will become smaller and smaller with decreasing h. How small is small depends, of course, on the function f. If $f''(x)$ is large, for example, h must be taken very small so that the product $hf''(x)$ is not so large as to destroy the desired numerical accuracy of the formula. The function f may change slowly in a certain range of values of x, where f'' is therefore small and one can use a reasonably large value of h, but the same function may vary rapidly in a different x-interval and, there, the formula must be used with a smaller h. With these caveats, we write

$$f'(x) \;\simeq\; \frac{f(x+h) - f(x)}{h}\,. \tag{2.1.26}$$

If the increment is chosen negative, let us write it as $-h$ for clarity; then we have

$$f'(x) \;\simeq\; \frac{f(x + (-h)) - f(x)}{(-h)} \;=\; \frac{f(x) - f(x-h)}{h}\,. \tag{2.1.27}$$

Proceeding as for (2.1.25) we find

$$\frac{f(x) - f(x-h)}{h} \;=\; f'(x) - \frac{1}{2} h f''(x) + \frac{1}{3!} h^2 f'''(x) + \ldots\,. \tag{2.1.28}$$

The formula (2.1.26) is the *forward* approximations to the first derivative since it uses a point ahead of x. Similarly, (2.1.27) is a *backward* approximation, since it uses a point behind x. These are one-sided

difference formulas as they approximate the derivative using only points to one side of x. It is evident from (2.1.25) that they are both affected by an error of order $O(h)$ and, therefore, they are *first-order accurate*. The difference between the true derivative and its approximation is a typical example of *truncation error*.

We note that the right-hand side of (2.1.26) may also be written as

$$\frac{f(x+h) - f\left((x+h) - h\right)}{h}\,, \tag{2.1.29}$$

which is exactly what we would write if we were to approximate $f'(x+h)$ by the backward formula (2.1.27). Thus, we may conclude that (2.1.26) approximates both $f'(x)$ and $f'(x+h)$ with the same error of order h.

An obvious question now is: how can we decrease the $O(h)$ error with which $f'(x)$ is estimated? By simply adding (2.1.25) and (2.1.28) we see that

$$\frac{1}{2}\left(\frac{f(x+h)-f(x)}{h} + \frac{f(x)-f(x-h)}{h}\right) \;=\; \frac{f(x+h)-f(x-h)}{2h} \;=\; f'(x) + \frac{1}{3}h^2 f'''(x)\,. \tag{2.1.30}$$

By using the three points x and $x \pm h$ we have succeeded in reducing the error from $O(h)$ to $O(h^2)$. How can we find a systematic way to derive high-order formulae in general without having to be so "smart" as to see this by inspection?

2.3 General method for the approximation of derivatives

We can see a path toward the systematic derivation of derivative approximations if we recast the derivation of the simple forward formula (2.1.26) as the following problem: To find constants α and β such that

$$\frac{\alpha f(x+h) + \beta f(x)}{h} \;\simeq\; f'(x)\,, \tag{2.3.1}$$

with as small an error as possible. To solve this problem we expand $f(x+h)$ in a Taylor series to find

$$\frac{\alpha f(x+h) + \beta f(x)}{h} \;\simeq\; \frac{\alpha[f(x) + hf'(x) + \frac{1}{2}h^2 f''(x) + \ldots] + \beta f(x)}{h} \;=\; \frac{\alpha+\beta}{h}f(x) + \frac{\alpha h}{h}f'(x) + \frac{\alpha h^2}{2h}f''(x) + \ldots \tag{2.3.2}$$

If the result is to be as close as possible to $f'(x)$ it is evidently necessary to choose

$$\alpha + \beta \;=\; 0\,, \qquad \alpha \;=\; 1\,. \tag{2.3.3}$$

Solving this linear system we find $\alpha = 1$, $\beta = -1$ and we recover (2.1.26). Evidently we must live with the error embodied in the last term of (2.3.2): there is nothing we can do about it because we have "spent" our free constants (or degrees of freedom) α and β to satisify the necessary relations (2.3.3).

The conclusion of this little analysis is that, if we want to cancel the f'' term, we need to introduce another degree of freedom. The way to do it is to invoke a third point, in addition to x and $x+h$, to approximate $f'(x)$. For example, let's try

$$\begin{aligned}
\frac{\alpha f(x+h) + \beta f(x) + \gamma f(x-h)}{h} \;&=\; \frac{1}{h}\Big\{\alpha[f(x) + hf'(x) + \frac{1}{2}h^2 f''(x) + \frac{1}{3!}h^3 f'''(x) + \ldots] + \beta f(x) \\
&\qquad +\gamma[f(x) + (-h)f'(x) + \frac{1}{2}(-h)^2 f''(x) + \frac{1}{3!}(-h)^3 f'''(x) + \ldots]\Big\} \\
&=\; \frac{\alpha+\beta+\gamma}{h}f(x) + \frac{\alpha h - \gamma h}{h}f'(x) + \frac{\alpha h^2 + \gamma h^2}{2h}f''(x) + \frac{\alpha h^3 - \gamma h^3}{3!h}f'''(x) + \ldots \\
&=\; \frac{\alpha+\beta+\gamma}{h}f(x) + (\alpha-\gamma)f'(x) + \frac{h(\alpha+\gamma)}{2}f''(x) + \frac{h^2(\alpha-\gamma)}{3!}f'''(x) + \ldots
\end{aligned} \tag{2.3.4}$$

To approximate the first derivative we must evidently remove $f(x)$ from the right-hand side and require that the coefficient of $f'(x)$ be 1. Furthermore, to improve the accuracy of the formula, we should remove the next term so that we can increasing the power of h to which the error is proportional.[15] Thus we must require that

$$\alpha + \beta + \gamma \,=\, 0\,, \qquad \alpha - \gamma \,=\, 1\,, \qquad \alpha + \gamma \,=\, 0\,. \tag{2.3.5}$$

This is a system of three algebraic equations in the three unknowns α, β and γ which is readily solved to find

$$\alpha \,=\, \frac{1}{2}\,, \qquad \beta \,=\, 0\,, \qquad \gamma \,=\, -\frac{1}{2}\,, \tag{2.3.6}$$

so that

$$f'(x) \,=\, \frac{f(x+h) - f(x-h)}{2h} + \frac{h^2}{12} f'''(x) + \dots \tag{2.3.7}$$

This is the same formula (2.1.30) that we had found by inspection, but which we have now derived as the result of a systematic method which can be applied to other cases as well. The fraction in (2.3.7) is the *centered difference approximation* to the first derivative.

An interesting comment on this result is the following: Suppose we want to apply this formula to approximating the derivative of f at the point $x + \frac{1}{2}h$ by using $\frac{1}{2}h$ as the increment in place of h. We would write

$$f'(x + \frac{1}{2}h) \,\simeq\, \frac{f(x + \frac{1}{2}h + \frac{1}{2}h) - f(x + \frac{1}{2}h - \frac{1}{2}h)}{2(\frac{1}{2}h)} \,=\, \frac{f(x+h) - f(x)}{h} + O(h^2)\,. \tag{2.3.8}$$

This shows that our relatively crude approximation (2.1.26) gives $f'(x)$ with an error proportional to h at any point of the interval between x and $x + h$ but, as shown by (2.3.7), it gives a smaller error proportional to h^2 at a special point, the midpoint! Otherwise said, the formula (2.1.2) approximates f' at the mid-point $x + \frac{1}{2}h$ much better than at either x or $x + h$. Similarly, the formula (2.1.28) approximates f' at the mid-point $x - \frac{1}{2}h$ with an error of order h^2.

2.3.1 A generalization

To further illustrate how the method works let us try to find an approximation to $f'(x)$ again by using 3 points, without choosing the third one at the special position $x - h$. We keep the position of the three points general by choosing them as x, $x + h_1$ and $x + h_2$. If $h_1 < 0$ the second point is to the left of x, while it is to the right if $h_1 > 0$; the same is true for the second point. Of course we should expect that, if we choose $h_2 = -h_1$, we would recover the previous result (2.3.7).

Again, what we need to do is to find the right coefficients of the linear combination

$$\alpha f(x) + \beta f(x + h_1) + \gamma f(x + h_2)\,, \tag{2.3.9}$$

that will produce such an improved approximation. In the previous case there was only one h and the mathematical definition of derivative suggested that it was efficient (in the sense of simplifying the algebra) to divide the linear combination by h as in (2.3.4). Now we have two h's and it is not obvious what to use but, since we have a systematic method, we should be confident that it will lead us to the right result irrespective of whether we include a denominator with h_1, h_2 or anything else. Our problem then is to find α, β and γ in such a way that the relation

$$\alpha f(x) + \beta f(x + h_1) + \gamma f(x + h_2) \,\simeq\, f'(x)\,, \tag{2.3.10}$$

holds with an error (or remainder) having as high a power of h_1 and h_2 as possible.

For this purpose we use the Taylor series on the second and third terms writing

$$f(x + h_1) \,=\, f(x) + h_1 f'(x) + \frac{1}{2} h_1^2 f''(x) + \frac{1}{3!} h_1^3 f'''(x) + \dots\,, \tag{2.3.11}$$

[15] Remember that we are always thinking of the limit $h \to 0$, i.e., of taking h smaller and smaller.

$$f(x+h_2) \;=\; f(x) + h_2 f'(x) + \frac{1}{2}h_2^2 f''(x) + \frac{1}{3!}h_2^3 f'''(x) + \dots \,. \tag{2.3.12}$$

Upon substitution into the left-hand side of (2.3.10) we have

$$\begin{aligned} \alpha f(x) + \qquad & \beta\left[f(x) + h_1 f'(x) + \frac{1}{2}h_1^2 f''(x) + \frac{1}{3!}h_1^3 f''' + \dots\right] + \\ & \gamma\left[f(x) + h_2 f'(x) + \frac{1}{2}h_2^2 f''(x) + \frac{1}{3!}h_2^3 f''' + \dots\right] \;\simeq\; f'(x)\,. \end{aligned} \tag{2.3.13}$$

Collecting terms we find

$$(\alpha+\beta+\gamma)\, f(x) + (\beta h_1 + \gamma h_2)\, f'(x) + \frac{1}{2}\left(\beta h_1^2 + \gamma h_2^2\right) f''(x) + \frac{1}{3!}\left(\beta h_1^3 + \gamma h_2^3\right) f''' + \dots \;\simeq\; f'(x)\,. \tag{2.3.14}$$

Upon comparing left- and right-hand sides we see that, if we we choose α, β and γ in such a way that

$$\alpha + \beta + \gamma \;=\; 0\,, \tag{2.3.15}$$

$$\beta h_1 + \gamma h_2 \;=\; 1\,, \tag{2.3.16}$$

$$\frac{1}{2}\left(\beta h_1^2 + \gamma h_2^2\right) \;=\; 0\,, \tag{2.3.17}$$

$$\frac{1}{3!}\left(\beta h_1^3 + \gamma h_2^3\right) \;=\; 0\,, \tag{2.3.18}$$

the left-hand side would approximate $f'(x)$ up to terms containing the fourth and higher powers of the increments h_1 and h_2. Since we only have three parameters, in general we can only satisfy three of these four conditions. Which ones to choose becomes evident when we realize that, in order to have a local approximation to $f'(x)$, the points $x+h_1$ and $x+h_2$ must be chosen close to x, so that the increments will be small (in a suitable sense to be specified later) and their powers will get smaller and smaller with increasing order. For this reason we choose to satisfy the first three equations of the system since the powers h_1^3 and h_2^3 will be smaller than h_1^2 and h_2^2. Satisfying the fourth one would require adding another parameter, which can be achieved by considering f at a fourth point $x+h_3$, and so on if we wish to remove still higher powers of the increments. With three points only, the best that we can do (at least in general) is to reduce the difference between the two sides of (2.3.10) to terms containing the third powers of the increments.

The system constituted by the first three equations of (2.3.16) is readily solved to find

$$\alpha \;=\; -\frac{h_1+h_2}{h_1 h_2}\,, \qquad \beta \;=\; \frac{h_2}{h_1(h_2-h_1)}\,, \qquad \gamma \;=\; -\frac{h_1}{h_2(h_2-h_1)} \tag{2.3.19}$$

so that we have the general formula

$$f'(x) \;\simeq\; -\frac{h_1+h_2}{h_1 h_2} f(x) + \frac{h_2}{h_1(h_2-h_1)} f(x+h_1) - \frac{h_1}{h_2(h_2-h_1)} f(x+h_2)\,. \tag{2.3.20}$$

The coefficient of the third derivative f''' is readily found to be

$$-\frac{1}{3!}h_1 h_2 \tag{2.3.21}$$

so that no choice of h_1 and/or h_2 (other than, trivially, 0) can cause it to vanish. The difference between the two sides of (2.3.10) is then, to leading order,

$$\alpha f(x) + \beta f(x+h_1) + \gamma f(x+h_2) - f'(x) \;=\; -\frac{1}{3!}h_1 h_2 f'''(x) + \dots \,. \tag{2.3.22}$$

This relation shows that the general formula (2.3.20) is accurate to second order: h_1 and h_2 are both of first order, and their product $h_1 h_2$ is therefore of second order.

Let us now look at some special cases

- Take $h_2 = h$, $h_1 = -h$; then we find

$$\alpha \,=\, 0\,, \qquad \beta \,=\, -\frac{1}{2h}\,, \qquad \gamma \,=\, \frac{1}{2h}\,, \tag{2.3.23}$$

so that

$$f'(x) \,=\, \frac{f(x+h) - f(x-h)}{2h} + O(h^3)\,. \tag{2.3.24}$$

In this case the two points $x + h_1$ and $x + h_2$ happen to be on either side of x and equally distant from it, which justifies the denomination *centered difference* given to this formula. We have already noted in the previous section that (2.3.24) can also be deduced by taking the average of the forward and backward formulae (2.1.26) and (2.1.27).

- Take $h_1 = h$, $h_2 = 2h$ to find a formula for the one-sided approximation from the right (also called *forward formula*, because it uses points ahead of the point of interest x):

$$f'(x) \,=\, \frac{4f(x+h) - 3f(x) - f(x+2h)}{2h} + O(h^2)\,. \tag{2.3.25}$$

- If $h_1 = -h$, $h_2 = -2h$ we have the formula for the one-sided approximation from the left (also called *backward formula*, because it uses points behind the point of interest x):

$$f'(x) \,=\, \frac{3f(x) - 4f(x-h) + f(x-2h)}{2h} + O(h^2)\,. \tag{2.3.26}$$

2.4 Higher-order derivatives

The general approach outlined in the previous section can also be applied to higher-order derivatives. Let us work out the second derivative, for example, for the special case in which we consider the point x and the two points $x \pm h$. As in (2.3.10) we write

$$\alpha f(x) + \beta f(x+h) + \gamma f(x-h) \,\simeq\, f''(x)\,, \tag{2.4.1}$$

and, proceeding as in (2.3.13), we have

$$\begin{aligned} \alpha f(x) + \qquad & \beta \left[f(x) + hf'(x) + \frac{1}{2}h^2 f''(x) + \frac{1}{3!}h^3 f''' + \frac{1}{4!}h^4 f^{iv} + \ldots \right] + \\ & \gamma \left[f(x) - hf'(x) + \frac{1}{2}h^2 f''(x) - \frac{1}{3!}h^3 f''' + \frac{1}{4!}h^4 f^{iv} + \ldots \right] \,\simeq\, f''(x)\,, \end{aligned} \tag{2.4.2}$$

or, upon collecting pieces,

$$(\alpha+\beta+\gamma) f(x) + h(\beta-\gamma) f'(x) + \frac{h^2}{2}(\beta+\gamma) f''(x) + \frac{h^3}{3!}(\beta-\gamma) f''' + \frac{h^4}{4!}(\beta-\gamma) f^{iv} + \ldots \,\simeq\, f''(x)\,. \tag{2.4.3}$$

In order for the left-hand side to equal the right-hand side as $h \to 0$ it is necessary that

$$\alpha + \beta + \gamma \;=\; 0 \tag{2.4.4}$$

$$h(\beta - \gamma) \;=\; 0 \tag{2.4.5}$$

$$\frac{h^2}{2}(\beta + \gamma) \;=\; 1\,. \tag{2.4.6}$$

Upon solving this system we find

$$\alpha \,=\, -\frac{2}{h^2}\,, \qquad \beta \,=\, \gamma = \frac{1}{h^2} \tag{2.4.7}$$

so that

$$f''(x) = \frac{f(x+h) - 2f(x) + f(x-h)}{h^2} + \frac{h^2}{12} f^{iv} + \dots . \tag{2.4.8}$$

This relation shows that the discretization formula embodied by the first term in the right-hand side is second-order accurate. In this special case something remarkable has happened: normally, with 3 points, we can only have 3 equations. Killing the coefficient of the term in f''' would amount to a fourth equation which normally[16] would not be satisfied. But, with this special choice of the three points, this fourth equation is identical to the second one and is therefore "accidentally" satisfied! Note that, while for the first derivative we need a minimum of 2 points and, adding a third one, buys us more accuracy, there would be no way to estimate the second derivative with fewer than 3 points because we have three requirements: "knocking out" the terms containing f and f' in the left-hand side of (2.4.3) and be left with a coefficient 1 for the third term f''. There is no way we can satisfy these three requirements with fewer than three coefficients, and this requires the use of three points minimum. However, just as in the case of the first derivative, we can obtain a formula with better accuracy by adding points.

There is another way in which we can deduce (2.4.8) which is interesting. The second derivative is the derivative of the first derivative and, therefore, we can apply (2.1.30); we do so by using $h/2$ rather than h. In this way we find

$$f''(x) \simeq \frac{f'(x+h/2) - f'(x-h/2)}{2(h/2)} . \tag{2.4.9}$$

Now we apply again (2.1.30), this time to approximate $f'(x \pm h/2)$:

$$f'(x+h/2) \simeq \frac{f([x+h/2]+h/2) - f([x+h/2]-h/2)}{2(h/2)} = \frac{f(x+h) - f(x)}{h} \tag{2.4.10}$$

$$f'(x-h/2) \simeq \frac{f([x-h/2]+h/2) - f'([x-h/2]-h/2)}{2(h/2)} = \frac{f(x) - f(x-h)}{h} \tag{2.4.11}$$

Upon substituting into (2.4.9) we find

$$f''(x) \simeq \frac{1}{h} \left[\frac{f(x+h) - f(x)}{h} - \frac{f(x)) - f(x-h)}{h} \right] \tag{2.4.12}$$

which is identical to (2.4.8).

How many points would we need to approximate, for example, the third derivative? When we carry out a Taylor series expansion of $f(x+h)$, for example, we have to write down three terms before we see f'''. By using the same approach as before, we need to kill the coefficients of $f(x)$, $f'(x)$ and $f''(x)$ and set equal to 1 the coefficient of f'''. Thus we need to impose 4 conditions, and this (except possibly for special cases, in which one equation is automatically satisfied as we saw before happening in (2.4.3) would require 4 points. As in the previous cases, we might buy some better accuracy by adding more points to the formula but, at a minimum, we would need the value of $f(x)$ at four points. By the same argument we would need a minimum of five points for the fourth derivative, and so on.

2.5 Partial derivatives

Differential equations in more than one space dimension contain partial derivatives along the same coordinate, or mixed derivatives. A possible approach in the former case is a straightforward extension of the one-dimensional case, as we now show. In two dimensions, let us write $x_j = j\,\Delta x$, $y_k = k\,\Delta y$ and $u_{jk} = u(x_j, y_k)$. Let us consider, for example, the Laplacian operator:

$$\nabla^2 u = \left(\frac{\partial^2 u}{\partial x^2} \right)_y + \left(\frac{\partial^2 u}{\partial y^2} \right)_x \tag{2.5.1}$$

[16] "Normally" means that, if in place of $x \pm h$ we had taken two general points $x + h_1$ and $x + h_2$, the coefficient of f''' would not vanish.

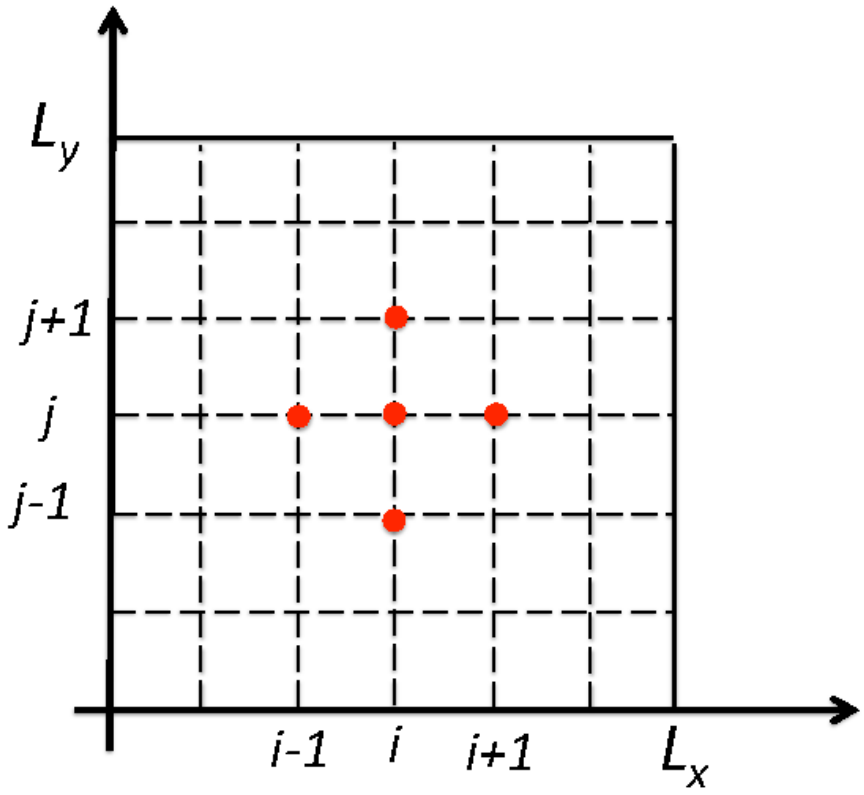


Figure 2.5.1: The stencil for the discretization (2.5.2) of the Laplacian operator in two dimensions.

where we use the usual notation $(\partial^2 u/\partial x^2)_y$ and $(\partial^2 u/\partial y^2)_x$ to indicate that the first derivative is with respect to x keeping y constant and conversely for the second one. One way to discretize this operator is as (see figure 2.5.1):

$$[\nabla^2 u]_{jk} \simeq \frac{u_{j-1,k} - 2u_{jk} + u_{j+1,k}}{\Delta x^2} + \frac{u_{j,k-1} - 2u_{jk} + u_{j,k+1}}{\Delta y^2} \tag{2.5.2}$$

In other words, the derivative with respect to x is calculated keeping $y = y_k$ constant (as indicated by the notation in (2.5.1)) using the usual formula for a second derivative and similarly for the derivative with respect to y.

Other possibilities exist, however. To explain them it is necessary to show the form of the Taylor series in two variables. This is readily derived starting from the familiar one in a single variable as follows:

$$f(x+h, y+k) = f(x, y+k) + h\left[\frac{\partial f}{\partial x}\right]_{x,y+k} + \frac{1}{2!}h^2\left[\frac{\partial^2 f}{\partial x^2}\right]_{x,y+k} + \dots \tag{2.5.3}$$

Now we apply the single-variable formula to each term in the right-hand side. For the first one we have

$$f(x, y+k) = f(x, y) + k\left[\frac{\partial f}{\partial y}\right]_{x,y} + \frac{1}{2!}k^2\left[\frac{\partial^2 f}{\partial y^2}\right]_{x,y} + \dots \tag{2.5.4}$$

For the second one it is convenient to set

$$g(x, y+k) \;=\; \left[\frac{\partial f}{\partial x}\right]_{x,y+k} \tag{2.5.5}$$

because then we find

$$g(x, y+k) \;=\; g(x,y) + k\frac{\partial g}{\partial y} + \dots \tag{2.5.6}$$

If we are interested in second-order accuracy it is sufficient to keep only one term since this is multiplied by h in (2.5.3); thus we have

$$\left[\frac{\partial f}{\partial x}\right]_{x,y+k} \;=\; \left[\frac{\partial f}{\partial x}\right]_{x,y} + k\left[\frac{\partial}{\partial y}\frac{\partial f}{\partial x}\right]_{x,y} \;=\; \frac{\partial f}{\partial x} + k\frac{\partial^2 f}{\partial y \partial x} + \dots \tag{2.5.7}$$

and, similarly, again to second order,

$$\left[\frac{\partial^2 f}{\partial x^2}\right]_{x,y+k} \;=\; \left[\frac{\partial^2 f}{\partial x^2}\right]_{x,y} + \dots \tag{2.5.8}$$

Upon substituting back into (2.5.3) we then find

$$\begin{aligned} f(x+h, y+k) \;&=\; f(x,y) + k\frac{\partial f}{\partial y} + \frac{1}{2!}k^2\frac{\partial^2 f}{\partial y^2} \\ &\qquad + h\left(\frac{\partial f}{\partial x} + k\frac{\partial^2 f}{\partial y \partial x}\right) + \frac{1}{2!}h^2\frac{\partial^2 f}{\partial x^2} + \dots \\ &=\; f(x,y) + h\frac{\partial f}{\partial x} + k\frac{\partial f}{\partial y} + \frac{1}{2!}\left[h^2\frac{\partial^2 f}{\partial x^2} + 2hk\frac{\partial^2 f}{\partial y \partial x} + k^2\frac{\partial^2 f}{\partial y^2}\right] + \dots \end{aligned} \tag{2.5.9}$$

With this formula we can consider, for example,

$$\frac{1}{4\Delta^2}\left[u_{j+1,k+1} + u_{j+1,k-1} + u_{j-1,k-1} + u_{j-1,k+1} - 4u_{jk}\right], \tag{2.5.10}$$

in which we have taken $\Delta x = \Delta = \Delta$ for simplicity. Then we find

$$u_{j+1,k\pm1} \;=\; u_{jk} + \Delta\left(\frac{\partial u}{\partial x} \pm \frac{\partial u}{\partial y}\right) + \frac{1}{2!}\Delta^2\left[\frac{\partial^2 u}{\partial x^2} \pm 2\frac{\partial^2 u}{\partial y \partial x} + \frac{\partial^2 u}{\partial y^2}\right] + \dots \tag{2.5.11}$$

$$u_{j-1,k\pm1} \;=\; u_{jk} - \Delta\left(\frac{\partial u}{\partial x} \mp \frac{\partial u}{\partial y}\right) + \frac{1}{2!}\Delta^2\left[\frac{\partial^2 u}{\partial x^2} \mp 2\frac{\partial^2 u}{\partial y \partial x} + \frac{\partial^2 u}{\partial y^2}\right] + \dots \tag{2.5.12}$$

Upon collecting terms we then find

$$u_{j+1,k+1} + u_{j+1,k-1} + u_{j-1,k-1} + u_{j-1,k+1} \;=\; 4u_{jk} + 4\Delta^2\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) + \dots \tag{2.5.13}$$

We thus conclude that

$$\frac{1}{4\Delta^2}\left[u_{j+1,k+1} + u_{j+1,k-1} + u_{j-1,k-1} + u_{j-1,k+1} - 4u_{jk}\right] \;=\; [\nabla^2 u]_{jk} + O(\Delta^2)\,. \tag{2.5.14}$$

so that we have found a different way to approximate the Laplacian operator.

Proceeding as in the case of ordinary derivatives, we notice that when we apply the Taylor series to $u(x_{j\pm1}, y_{k\pm1})$ we end up with $u_{j,k}$, $\partial_x u$, $\partial_y u$, $\partial_x^2 u$, $\partial_y^2 u$ and $\partial_x \partial_y$, in addition to the higher order terms. To form the Laplacian we need to delete $u(x_j, y_k)$, the first-order partial derivatives and the mixed second-order derivative, and set to 1 the coefficients of $\partial_x^2 u$, $\partial_y^2 u$. These are 5 conditions and they therefore require a minimum of 5 points to be satisfied.

Chapter 3

Elliptic Equations I

3.1 Elliptic equations

The two most famous examples of elliptic equations are the Poisson equation

$$\nabla^2 u = f(x)\,, \tag{3.1.1}$$

with its special case, the Laplace equation,

$$\nabla^2 u = 0\,, \tag{3.1.2}$$

and the Helmholtz equation

$$\nabla^2 u + \lambda u = f(x)\,. \tag{3.1.3}$$

Here ∇^2 is the Laplacian operator

$$\nabla^2 u = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2}\,. \tag{3.1.4}$$

u is the unknown function, f a given function and the parameter λ, which can be positive or negative (or even complex), also given and known. These equations are to be solved subject to appropriate boundary conditions, an important point to which we return later.

A simple problem giving rise to the one-dimensional version of the Helmholtz equation is heat conduction in a cylindrical rod subject to convective cooling on its surface. If we consider a slice of the rod of thickness dx, at steady state we have a heat balance

$$[\text{heat into slice at } x] = [\text{heat out of slice at } (x+dx)] + [\text{heat lost by convection}] \tag{3.1.5}$$

or, in formulae,

$$S\left[-k\frac{dT}{dx}\right]_x = S\left[-k\frac{dT}{dx}\right]_{x+dx} + P\,dx\,h(T-T_\infty)\,. \tag{3.1.6}$$

Here S and P are the area and perimeter of the rod cross section, k is the thermal conductivity of the rod material, h is the convective heat transfer coefficient, T is the temperature of the rod cross section and T_∞ is the temperature of the fluid cooling the rod. Since, by Taylor's theorem,

$$\left[\frac{dT}{dx}\right]_{x+dx} = \left[\frac{dT}{dx}\right]_x + dx\left[\frac{d^2T}{dx^2}\right]_x + O(dx^2)\,, \tag{3.1.7}$$

dividing through by dx we find

$$kS\frac{d^2T}{dx^2} - hP(T-T_\infty) = 0\,. \tag{3.1.8}$$

If we let

$$u = T - T_\infty\,, \qquad \lambda = -\frac{hP}{kS}\,, \tag{3.1.9}$$

we have an example of the one-dimensional version of the Helmholtz equation (3.1.3).

Generally speaking, both the Poisson and Helmholtz equations describe equilibrium situations, that is, the state of a system long after all motion (or time dependence) has faded away and all agents affecting the system balance. For example, the Poisson equation describes the steady-state temperature field in a

solid in which heat is transported by conduction (and possibly, as we have just seen, steady convection on its boundary). As we have seen, when a steady state has been reached, the addition of convection leads us to the Helmholtz equation. In these systems, f would correspond to a source of heat in the solid due to electrical current or radioactive decay. The Poisson equation also describes the electrical potential in a system with motionless charges, or the deformation of an elastic membrane. Many other examples could be added to this list.

It should be stressed that the mathematical formulation of a problem does not consist only of the differential equation: boundary conditions are of equal importance and, in general, cannot be treated as an "afterthought" once the differential equation has been solved. This attitude can be often applied when working with ordinary differential equations, but it is really misleading as boundary conditions are just as important as the differential equation itself as is often found with partial differential equations. Typical boundary conditions for elliptic problems may be of the

- *Dirichlet type*: The value of the unknown u is prescribed on (part of) the boundary. For example, the temperature at the root and at the end of the rod in the previous example is known;
- *Neumann type*: The derivative of the unknown u is prescribed on (part of) the boundary. For example, the rod is insulated at one of the ends, so that the heat flux $q = -kdT/dx = 0$;
- *Mixed*, or *Robin type*: A linear combination of u and its derivative, namely $\alpha u + \beta dT/dx$ with known values for α and β, is prescribed on (part of) the boundary. For example, if there is convective cooling at one end of the rod we would write $q = q_{conv}$ or $-kdT/dx = h(T - T_\infty)$.

A specific features of elliptic problems is that boundary conditions are specified over the entire boundary. They can be of one type on part of the boundary and of a differen type on the remainder. For a one-dimensional problem the boundary reduces to the two end-points of the interval of interest. In a two-dimensional problem it would be the perimeter of the domain of interest and, in three dimensions, the surface bounding the domain of interest.

The fact that we impose conditions over the entire boundary makes this a *boundary value problem* as opposed to a situation in which we would impose more than one boundary condition on part of the domain boundary leaving the boundary conditions unspecified on the remaining part. Typically problems of this type involve time so that the domain of interest is actually a region of space-time with the inital instant $t = 0$ part of the "boundary" of this space-time region. A typical example in mechanics might be the elastic waves on a taut string. In this case the domain of interest is a strip in the (x, t) plane in which x lies between the left and right end-points of the string while t ranges from 0 to the final time of interest, which could be infinite. In this case we would specify consitions at the end-points of the string (e.g., the end-points cannot move) and, since this is a mechanics problem, we would also need to specify two conditions, initial shape and velocity of the string, at the "base" of the strip $t = 0$. (Problems in which future conditions at some later are specified are somewhat unusual.) Problems of this type are usually referred to as *initial-boundary value problems* We will see how such problems are handled numerically later on.

How do we solve an equation such as (3.1.1) or (3.1.3) numerically? The simplest case is that of a one-dimensional problem with Dirichlet conditions, with which we begin. We consider the Poisson equation since the procedure for the Helmholtz equation then follows in a straightforward way.

3.2 Set-up for numerical solution

We consider the one-dimensional version of (3.1.1) in a domain $0 \leq x \leq L$:

$$\frac{d^2u}{dx^2} = f(x)\,, \tag{3.2.1}$$

with the function $f(x)$ prescribed. We assume that the boundary conditions are of the Dirichlet type: $u(x = 0) = u_0$, $u(x = L) = u_L$ with u_0, u_L both prescribed quantities which will depend on the particular problem that is being solved.

We start by dividing the domain of interest into $N+1$ equal segments, each of length h, by introducing N equi-spaced points so that $h = L/(N+1)$(figure 3.2.1).[17] Let us denote by $x_j = jh$ the positions of these points and let us write $u_j = u(x_j)$, $f_j = f(x_j)$ for brevity. Evidently $x_0 = 0$ and $x_{N+1} = L$. We wish to write an approximate form of the equation (3.2.1) at the generic point x_j, with $j = 1, 2, \ldots, N$:[18]

$$\left[\frac{d^2u}{dx^2} \;=\; f(x)\right]_{x=x_j} . \tag{3.2.2}$$

We know how to approximate the left-hand side:

$$\left[\frac{d^2u}{dx^2}\right]_{x=x_j} \simeq \frac{u_{j+1} - 2u_j + u_{j-1}}{h^2} , \tag{3.2.3}$$

and, evidently, $[f(x)]_{x=x_j} = f(x_j) = f_j$. Thus we find

$$\frac{u_{j+1} - 2u_j + u_{j-1}}{h^2} \;=\; f_j + O(h^2) . \tag{3.2.4}$$

Here we have included $O(h^2)$ in the right-hand side as a reminder of the fact that, in going from (3.2.2) to (3.2.4), we have introduced a *truncation error* ϵ_T, which is the difference between the finite-difference approximation and the original derivative; as we know

$$\epsilon_T \;=\; \frac{u_{j+1} - 2u_j + u_{j-1}}{h^2} - \left[\frac{d^2u}{dx^2}\right]_{x=x_j} \;=\; \frac{h^2}{12}\left[\frac{d^4u}{dx^4}\right]_{x=x_j} + \ldots , \tag{3.2.5}$$

is proportional to h^2.

Upon multiplying (3.2.4) by h^2 we have

$$u_{j-1} - 2u_j + u_{j+1} \;=\; h^2 f_j + O(h^4) , \tag{3.2.6}$$

in which we write $h^2 O(h^2) = O(h^4)$; we omit the specific indication of this term in the following. For $j = 1$ we take $u_{j-1} = u_0$ to the right-hand side since it is known:

$$-2u_1 + u_2 \;=\; h^2 f_1 - u_0 , \tag{3.2.7}$$

and, similarly, for $j = N$ we take $u_{j+1} = u_L$ to the right-hand side

$$u_{N-1} - 2u_N \;=\; h^2 f_N - u_L . \tag{3.2.8}$$

[17] Using only one point, $N = 1$, gives us two segments each one of length $L/2$; using 2 points, $N = 2$, gives us three segments of length $L/3$ each, and so on.

[18] Since the values of u_0 and u_{N+1} are given by the boundary conditions, we do not need to write equations for the end-points of the domain.

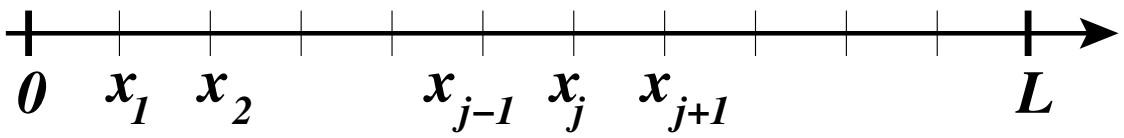


Figure 3.2.1: Discretization of the computational domain $0 < x < L$.

In this way we have generated the linear algebraic system

$$-2u_1 + u_2 = h^2 f_1 - u_0\,, \tag{3.2.9a}$$
$$u_1 - 2u_2 + u_3 = h^2 f_2\,, \tag{3.2.9b}$$
$$u_2 - 2u_3 + u_4 = h^2 f_3\,, \tag{3.2.9c}$$
$$\cdots$$
$$u_{N-2} - 2u_{N-1} + u_N = h^2 f_3\,, \tag{3.2.9d}$$
$$u_{N-1} - 2u_N = h^2 f_N - u_L\,. \tag{3.2.9e}$$

The problem of solving the Poisson equation has then been reduced to the problem of solving this linear system. This is a simple example of one of the many ways in which linear systems become associated with the numerical solution of differential equations.

For the the one-dimensional Helmholtz equation, in place of (3.2.4) we would have

$$\frac{u_{j-1} - 2u_j + u_{j+1}}{h^2} + \lambda u_j = f_j\,, \tag{3.2.10}$$

Upon rearranging this becomes

$$u_{j-1} - (2 - \lambda h^2)u_j + u_{j+1} = h^2 f_j\,, \tag{3.2.11}$$

which differs only slightly from (3.2.6). Boundary conditions at both end-points are also needed and they are handled in the same way if they are of the Dirichlet type.

Since (3.2.3) is not exactly equal to d^2u/dx^2, the discretized equations (3.2.4) and (3.2.10) are not exactly equal to the original differential equations. This difference is called the *truncation error* and is an unavoidable consequence of the transition from the continuum to the discrete formulation of the problem on which the numerical solution is built. In general, the discretized version of a problem is said to be *consistent* with the original differential formulation when the truncation error becomes smaller and smaller as the nodes are made closer to each other and their number therefore increases. In the present cases, since (3.2.3) explicitly shows that the truncation error decreases proportionally to h^2, consistency is obviously verified.

Another important issue is that of *convergence*: does the solution of the discretized problem tend to the solution of the original differential problem as the grid spacing tends to zero? It should be stressed that convergence is not a necessary consequence of consistency because it may happen that the solution of the discretized problem is unstable. This point will be better appreciated in connection with the discussion of iterative methods and time-dependent problems to be undertaken later.

3.3 Numerical solution of tri-diagonal linear systems

We consider a linear algebraic system of the form

$$a_1 u_1 + c_1 u_2 = f_1 \tag{3.3.1a}$$
$$b_2 u_1 + a_2 u_2 + c_2 u_3 = f_2 \tag{3.3.1b}$$
$$\cdots$$
$$b_{N-1} u_{N-2} + a_{N-1} u_{N-1} + c_{N-1} u_N = f_{N-1} \tag{3.3.1c}$$
$$b_N u_{N-1} + a_N u_N = f_N\,. \tag{3.3.1d}$$

We re-write this system in matrix form as

$$\left|\begin{array}{cccccccc} a_1 & c_1 & 0 & 0 & 0 & \dots & 0 & 0 \\ b_2 & a_2 & c_2 & 0 & 0 & \dots & 0 & 0 \\ 0 & b_3 & a_3 & c_3 & 0 & \dots & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & 0 & 0 & b_{N-1} & a_{N-1} & c_{N-1} \\ 0 & 0 & 0 & 0 & 0 & 0 & b_N & a_N \end{array}\right|\left|\begin{array}{c} u_1 \\ u_2 \\ u_3 \\ \dots \\ \dots \\ \dots \\ u_{N-1} \\ u_N \end{array}\right| = \left|\begin{array}{c} f_1 \\ f_2 \\ f_3 \\ \dots \\ \dots \\ \dots \\ f_{N-1} \\ f_N \end{array}\right| \tag{3.3.2}$$

It will be noticed that the matrix has non-zero elements only on the main diagonal and on the two adjacent subdiagonals. This is a very special form which greatly facilitates the solution as we will see. Matrices of this special type are called *tri-diagonal matrices*.

To explain the method let us consider a 3×3 system:

$$\left|\begin{array}{ccc} a_1 & c_1 & 0 \\ b_2 & a_2 & c_2 \\ 0 & b_3 & a_3 \end{array}\right|\left|\begin{array}{c} u_1 \\ u_2 \\ u_3 \end{array}\right| = \left|\begin{array}{c} f_1 \\ f_2 \\ f_3 \end{array}\right| \tag{3.3.3}$$

i.e.

$$a_1u_1 + c_1u_2 = f_1 \tag{3.3.4}$$
$$b_2u_1 + a_2u_2 + c_2u_3 = f_2 \tag{3.3.5}$$
$$b_3u_2 + a_3u_3 = f_3 \tag{3.3.6}$$

To eliminate u_1 between the first two equations we form the combination (3.3.5) $-$ $(b_2/a_1) \times$ (3.3.4) to find

$$(b_2u_1 + a_2u_2 + c_2u_3) - \frac{b_2}{a_1}(a_1u_1 + c_2u_2) = f_2 - \frac{b_2}{a_1}f_1 \tag{3.3.7}$$

which reduces to

$$\left(a_2 - \frac{b_2c_1}{a_1}\right)u_2 + c_2u_3 = f_2 - \frac{b_2}{a_1}f_1 \tag{3.3.8}$$

Define

$$\alpha_2 = a_2 - \frac{b_2c_1}{a_1}, \qquad g_2 = f_2 - \frac{b_2}{a_1}f_1 \tag{3.3.9}$$

Then we are left with the two equations

$$\alpha_2u_2 + c_2u_3 = g_2 \tag{3.3.10}$$
$$b_3u_2 + a_3u_3 = f_3 \tag{3.3.11}$$

We operate as before forming (3.3.11) $-$ $(b_3/\alpha_2) \times$ (3.3.10) to find

$$(b_3u_2 + a_3u_3) - \frac{b_3}{\alpha_2}(\alpha_2u_2 + c_2u_3) = f_3 - \frac{b_3}{\alpha_2}g_2 \tag{3.3.12}$$

from which

$$\left(a_3 - \frac{b_3}{\alpha_2}c_2\right)u_3 = f_3 - \frac{b_3}{\alpha_2}g_2 \tag{3.3.13}$$

which, with

$$\alpha_3 = a_3 - \frac{b_3}{\alpha_2}c_2, \qquad g_3 = f_3 - \frac{b_3}{\alpha_2}g_2 \tag{3.3.14}$$

we can write as

$$\alpha_3 u_3 \;=\; g_3 \tag{3.3.15}$$

which is easily solved to give g_3/α_3. At this point we go back to (3.3.10) to find

$$u_2 \;=\; \frac{g_2 - c_2 u_3}{\alpha_2} \tag{3.3.16}$$

and to (3.3.4) to find

$$a_1 u_1 + c_1 u_2 \;=\; f_1 \tag{3.3.17}$$

from which we can calculate u_1.

In summary, if we set

$$\alpha_1 \;=\; a_1\,, \qquad g_1 \;=\; f_1\,, \tag{3.3.18}$$

we see that

$$\alpha_2 \;=\; a_2 - \frac{b_2 c_1}{\alpha_1}\,, \qquad g_2 \;=\; f_2 - \frac{b_2}{\alpha_1} g_1\,, \tag{3.3.19}$$

$$\alpha_3 \;=\; a_3 - \frac{b_3}{\alpha_2} c_2\,, \qquad g_3 \;=\; f_3 - \frac{b_3}{\alpha_2} g_2\,, \tag{3.3.20}$$

after which we find the solution as

$$u_3 \;=\; \frac{g_3}{\alpha_3}\,, \tag{3.3.21}$$

$$u_2 \;=\; \frac{g_2 - c_2 u_3}{\alpha_2} \tag{3.3.22}$$

$$u_1 \;=\; \frac{g_1 - c_1 u_2}{\alpha_1} \tag{3.3.23}$$

There is a clear pattern here. For the general case of an $N \times N$ system it goes like this:

$$\alpha_1 \;=\; a_1\,, \qquad g_1 \;=\; f_1\,, \tag{3.3.24}$$

$$\alpha_j \;=\; a_j - \frac{b_j c_{j-1}}{\alpha_{j-1}}\,, \qquad g_j \;=\; f_j - \frac{b_j}{\alpha_{j-1}} g_{j-1} \qquad \text{for} \qquad j \;=\; 2,\, 3,\, \ldots,\, N \tag{3.3.25}$$

after which

$$u_N \;=\; \frac{g_N}{\alpha_N}\,, \tag{3.3.26}$$

and

$$u_j \;=\; \frac{g_j - c_j u_{j+1}}{\alpha_j} \qquad \text{for} \qquad j \;=\; N-1,\, N-2,\, \ldots,\, 1\,. \tag{3.3.27}$$

This is the so-called *Thomas* or *tri-diagonal algorithm*. A pseudo-code for this algorithm looks as follows:

$\alpha_1 = a_1$
$g_1 = f_1$

for $j = 2, N$
$\alpha_j = a_j - (b_j/\alpha_{j-1})c_{j-1}$
$g_j = f_j - (b_j/\alpha_{j-1})g_{j-1}$
end

$u_N = g_N/\alpha_N$

for $k = 1, N-1$
$u_{N-k} = (g_{N-k} - c_{N-k}u_{N-k+1})/\alpha_{N-k}$
end

It may be noted that, since a_j and f_j are never re-used after they are used to calculate α_j and g_j, they can actually be over-written. Thus, in place of $\alpha_j = a_j - (b_j/\alpha_{j-1})c_{j-1}$ and $g_j = f_j - (b_j/\alpha_{j-1})g_{j-1}$ we can simply save memory by writing $a_j = a_j - (b_j/a_{j-1})c_{j-1}$ and $f_j = f_j - (b_j/a_{j-1})f_{j-1}$. (Note that the equal sign here is used to assign a value as in most programming languages. It does not mean equality; so for example, the meaning of $a_j = a_j - (b_j/a_{j-1})c_{j-1}$ is that the result of the calculation $a_j - (b_j/a_{j-1})c_{j-1}$ is stored in the same memory location that contained a_j.)

A careful consideration of the method just described identifies some possible problems that can arise in its application:

1. Since each step of both the direct and inverse procedures requires division by α_j, to apply the method it is necessary that none of the α_j's vanish. There are theorems that give sufficient conditions for this not to happen. These theorems point to a very desirable feature of the matrix called *diagonal dominance*. The matrix of the system (3.3.2) is diagonally dominant if

$$|a_j| \geq |b_j| + |c_j|\,, \qquad j = 1,\, 2,\, \ldots,\, N\,, \tag{3.3.28}$$

   with at least one of the inequalities holding in the strict sense (i.e., with at least one the "$\geq$" replaced by a "$>$"). For the Poisson equation, i.e., (3.2.10) with $\lambda = 0$, $a_j = -2$, $b_j = 1$, $c_j = 1$ and this condition holds with the equal sign for $2 \leq j \leq N-1$. For $j = 1$ $b_1 = 0$, and for $j = N$ and $c_N = 0$, so that for these values of j the inequality holds in the strict sense. The situation is more delicate for the Helmholtz equation (i.e., $\lambda \neq 0$), where a more careful analysis may be necessary.

2. The backward recurrence relation (3.3.27) can cause problems because of round-off errors. For example, suppose that u_N as determined by (3.3.26) is affected by an error ϵ_N, so that the quantity that is actually calculated is $\tilde{u}_N = u_N + \epsilon_N$ in place of the "true value" u_N.[19] Then (3.3.27) gives

$$\tilde{u}_{N-1} \;=\; u_{N-1} + \epsilon_{N-1} \;=\; \frac{g_{N-1} - c_{N-1}\tilde{u}_N}{\alpha_{N-1}} \;=\; \frac{g_{N-1} - c_{N-1}u_N}{\alpha_{N-1}} - \frac{c_{N-1}\epsilon_N}{\alpha_{N-1}}\,, \tag{3.3.29}$$

   which shows that the error induced in the determination of u_{N-1} is given by

$$\epsilon_{N-1} \;=\; -\frac{c_{N-1}}{\alpha_{N-1}}\epsilon_N\,. \tag{3.3.30}$$

   At the next step we find an error

$$\epsilon_{N-2} \;=\; -\frac{c_{N-2}}{\alpha_{N-2}}\epsilon_{N-1} \;=\; \left(-\frac{c_{N-2}}{\alpha_{N-2}}\right)\left(-\frac{c_{N-1}}{\alpha_{N-1}}\right)\epsilon_N\,, \tag{3.3.31}$$

   and so forth. It is evident that, if $c_j/\alpha_j > 1$, the error is amplified at each step and, with a large system of equations, can grow large enough to destroy the calculation. Sufficient conditions to avoid this problem is that none of the b_j's and c_j's vanish and, once again, diagonal dominance.

3. The previous conditions are only sufficient. Suppose for example that $b_K = 0$ for a certain index K. In this case the unknown u_{K-1} drops out of the K-th equation and, since this is the only link between the K-th equation and the preceding ones, the system reduces to two smaller systems, one with unknowns $u_1,\, u_2,\, \ldots,\, u_{K-1}$ and one with unknowns $u_K,\, u_{K+1},\, \ldots,\, u_N$, which can be solved separately one from the other.

[19] Here we do not refer to the true value in the sense of the true solution of the differential equation. At this point we have introduced disretization errors that make that "truest" value irretrivable. We refer to the fact that, due to round-off error, u_N will not even be the exact solution of the discretized linear system.

The same algorithm can be extended to more complicated problems in which the matrix is *block-tri-diagonal*, i.e., it has the form

$$\left|\begin{array}{cccccccc} A_1 & C_1 & 0 & 0 & 0 & \dots & 0 & 0 \\ B_2 & A_2 & C_2 & 0 & 0 & \dots & 0 & 0 \\ 0 & B_3 & A_3 & C_3 & 0 & \dots & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & 0 & 0 & B_{N-1} & A_{N-1} & C_{N-1} \\ 0 & 0 & 0 & 0 & 0 & 0 & B_N & A_N \end{array}\right| , \tag{3.3.32}$$

in which the A_j, B_j and C_j are all matrices. The sequence of operations is the same except that divisions are to be interpreted as multiplications by the suitable inverse matrix. The big advantage is due to the fact that the computational cost of linear system solution increases rapidly with the size of the matrix (the precise rate of increase depending on the specific method). Hence, having to deal with smaller matrices accrues significant savings.

3.4 Neumann boundary conditions

In the case of boundary conditions of the Neumann type, it is the derivative of u at the boundary point(s) that is prescribed. For example, for $x = 0$, the mathematical statement of the problem prescribes that

$$\left.\frac{du}{dx}\right|_{x=0} = v\,, \tag{3.3.33}$$

with v given, but u_0 itself unknown. Now the system of equations written at the nodes $j = 1, 2, \dots, N$ needs to be augmented by an equation for the node $j = 0$ and, if a Neumann condition is imposed at $x = L$, also by an equation for the node $j = N + 1$.

A simple idea to formulate an equation for $j = 0$ is to approximate (3.3.33) by a forward formula:

$$\frac{u_1 - u_0}{h} \simeq v + O(h) \tag{3.3.34}$$

from which

$$-u_0 + u_1 = hv + O(h^2)\,. \tag{3.3.35}$$

This can be taken as the first equation of the system to which all the others of the form (3.2.6) with $j = 1, 2, \dots$ would be added; for the Helmholtz problem the second equation will be then

$$u_0 - (2 - \lambda h^2)u_1 + u_2 = h^2 f_1\,, \tag{3.3.36}$$

and so on. The disadvantage of this procedure is that, while the formula for the second derivative that we use in (3.2.4) and the analogous Helmholtz equation is highly accurate so that the error in the generic equation of the system is $O(h^4)$, as shown in (3.2.6), the error in (3.3.36) is much greater due to the use of the low-accuracy formula (3.3.33). Many problems are critically sensitive to boundary conditions, and this procedure runs the risk of injecting a significant error in the entire calculation unless a very fine grid is used. A better procedure might be to use, in place of (3.3.34), the more accurate three-point forward formula

$$\frac{4u_2 - 3u_1 - u_0}{2h} \simeq v + O(h^2) \tag{3.3.37}$$

with which the first equation of the system would be

$$-u_0 - 3u_1 + 4u_2 = 2hv + O(h^3)\,. \tag{3.3.38}$$

But, if we do this, we have destroyed the tri-diagonal structure of the system as the first equation now has 3 unknowns rather than 2. A smarter way is to introduce what is called a *ghost node* at $x = x_{-1} = -h$. Now, in place of (3.3.34) we have the higher-accuracy formula

$$\frac{u_1 - u_{-1}}{2h} \simeq v + O(h^2) \tag{3.3.39}$$

from which $u_{-1} = u_1 - 2hv + O(h^3)$. If we now write (3.2.11) for $j = 0$ we have

$$u_{-1} - (2 - \lambda h^2)u_0 + u_1 = h^2 f_0 \tag{3.3.40}$$

and, upon expressing u_{-1} using (3.3.39), we find

$$-(2 - \lambda h^2)u_0 + 2u_1 = h^2 f_0 + 2hv\,, \tag{3.3.41}$$

which has the desired form and an error of order $h\,O(h^2) = O(h^3)$, only slightly larger than the $O(h^4)$ error affecting the other equations of the system. The same procedure can be applied if the Neumann condition is at $x = L$.

Chapter 4

Linear Systems

4.1 Introduction

Generally sepaking, the presence of unavoidable truncation and round-off errors makes it impossible to calculate the exact solution of a problem even when the method employed is capable, in principle, of doing so. (This is reason why occasionally write "exact" in quotation marks.) Recognizing this, we might as well use methods capable of producing approximate solutions if they lead to a faster execution. A prime class of such methods are iterative methods which produce a sequence of approximations which converges to the exact solution as the number of iterations increases. Ideally, we might want to iterate so many times that the difference between the exact and the analytical solution is smaller than the truncation and round-off errors, although this is actually seldom necessary.

Although our focus are linear algebraic systems, let's get started with a simple example. It is known that, for any positive real number s, [20]

$$x_{n+1} = \frac{1}{2}\left(x_n + \frac{s}{x_n}\right) \to \sqrt{s}\,. \tag{4.1.1}$$

For $s = 2$, upon taking $x_0 = 1$, we generate these results:

$$\begin{aligned} x_1 &= 1.5 \\ x_2 &= 1.416666666666 \\ x_3 &= 1.41425686274510 \\ x_4 &= 1.414213562374690 \\ x_5 &= 1.414213562373095 \\ \text{exact} &= 1.414213562373095\ldots \end{aligned} \tag{4.1.2}$$

If one wanted only, say, 4 accurate digits, one could stop at x_3.

In computational science this general idea finds major applications in the solution of linear systems (among others). As noted before, a major motivation is the operation count. For example, Gauss elimination (section 4.7 below) requires $O(N^3/3)$ operations for a system of order N, and this can be substantially decreased by using a rapidly converging iterative method.

4.2 The Jacobi method

Jacobi's method is perhaps the oldest iterative method for linear systems. To explain it we consider a simple 3×3 system:

$$a_{11}u_1 + a_{12}u_2 + a_{13}u_3 = f_1 \tag{4.2.1a}$$
$$a_{21}u_1 + a_{22}u_2 + a_{23}u_3 = f_2 \tag{4.2.1b}$$
$$a_{31}u_1 + a_{32}u_2 + a_{33}u_3 = f_3 \tag{4.2.1c}$$

[20] This is actually the Newton-Raphson method applied to the equation $x^2 - s = 0$; this method was known to the ancient Babylonians.

Start from an arbitrary initial "guess" (u_1^0, u_2^0, u_3^0) (for example, one could take $(u_1^0 = u_2^0 = u_3^0 = 0)$ and generate the recurrence

$$a_{11}u_1^{(n+1)} = f_1 - (a_{12}u_2^{(n)} + a_{13}u_3^{(n)}) \tag{4.2.2a}$$
$$a_{22}u_2^{(n+1)} = f_2 - (a_{21}u_1^{(n)} + a_{23}u_3^{(n)}) \tag{4.2.2b}$$
$$a_{33}u_3^{(n+1)} = f_3 - (a_{31}u_1^{(n)} + a_{32}u_2^{(n)}) \tag{4.2.2c}$$

If the matrix A of the original system is decomposed as the sum of a diagonal matrix, a lower-triangular matrix and an upper triangulat matrix:

$$\mathsf{A} = \mathsf{D} + \mathsf{L} + \mathsf{U}\,, \tag{4.2.3}$$

as follows

$$\begin{vmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{vmatrix} = \begin{vmatrix} a_{11} & 0 & 0 \\ 0 & a_{22} & 0 \\ 0 & 0 & a_{33} \end{vmatrix} + \begin{vmatrix} 0 & 0 & 0 \\ a_{21} & 0 & 0 \\ a_{31} & a_{32} & 0 \end{vmatrix} + \begin{vmatrix} 0 & a_{12} & a_{13} \\ 0 & 0 & a_{23} \\ 0 & 0 & 0 \end{vmatrix}, \tag{4.2.4}$$

the Jacobi method can be written in a compact way as

$$\mathsf{D}\mathbf{u}^{(n+1)} = \mathbf{f} - [\mathsf{L}\mathbf{u}^{(n)} + \mathsf{U}\mathbf{u}^{(n)}]\,. \tag{4.2.5}$$

Since D is diagonal, it is readily inverted:

$$\mathsf{D}^{-1} = \begin{vmatrix} 1/a_{11} & 0 & 0 \\ 0 & 1/a_{22} & 0 \\ 0 & 0 & 1/a_{33} \end{vmatrix}, \tag{4.2.6}$$

so that (4.2.5) is very easily solved:

$$\mathbf{u}^{(n+1)} = \mathsf{D}^{-1}\left[\mathbf{f} - (\mathsf{L}\mathbf{u}^{(n)} + \mathsf{U}\mathbf{u}^{(n)})\right]. \tag{4.2.7}$$

4.3 The Gauss-Seidel Method

In the Jacobi method all the variables are updated before moving on to the next step. But if $u_1^{(n+1)}$ is closer to the "exact" value of u_1 than $u_1^{(n)}$, it would make sense to use it in the update of $u_2^{(n+1)}$ writing, in place of (4.2.2b),

$$a_{22}u_2^{(n+1)} = f_2 - (a_{21}u_1^{(n+1)} + a_{23}u_3^{(n)})\,, \tag{4.3.1}$$

and, for the same reason, to use both $u_1^{(n+1)}$ and $u_2^{(n+1)}$ in the update of $u_3^{(n+1)}$ writing

$$a_{33}u_3^{(n+1)} = f_3 - (a_{31}u_1^{(n+1)} + a_{32}u_2^{(n+1)})\,. \tag{4.3.2}$$

For a system of N equations these relations have the form

$$a_{jj}u_j^{(n+1)} = f_j - \left(a_{j1}u_1^{(n+1)} + \ldots + a_{j,j-1}u_{j-1}^{(n+1)} + a_{j,j+1}u_{j+1}^{(n)} + \ldots + a_{jN}u_N^{(n)}\right). \tag{4.3.3}$$

This is the *Gauss-Seidel method*, one of the basic iterative methods for the solution of linear systems. It converges typically twice as fast as the Jacobi method. In matrix form we can write the previous equations as

$$\mathsf{D}\mathbf{u}^{(n+1)} = \mathbf{f} - \left[\mathsf{U}\mathbf{u}^{(n)} + \mathsf{L}\mathbf{u}^{(n+1)}\right]. \tag{4.3.4}$$

We thus see that the Gauss-Seidel method corresponds to replacing (4.2.5) by

$$(\mathsf{D} + \mathsf{L})\,\mathbf{u}^{(n+1)} = \mathbf{f} - \mathsf{U}\mathbf{u}^{(n)}\,. \tag{4.3.5}$$

The matrix in the left-hand side is not diagonal, but its special structure makes the solution of this system easy as is made evident by the explicit expression (4.3.3).

4.4 Successive over-relaxation (SOR)

The Gauss-Seidel method for a 3×3 system was

$$a_{11}u_1^{(n+1)} = f_1 - \left(a_{12}u_2^{(n)} + a_{13}u_3^{(n)}\right), \tag{4.4.1a}$$

$$a_{22}u_2^{(n+1)} = f_2 - \left(a_{21}u_1^{(n+1)} + a_{23}u_3^{(n)}\right), \tag{4.4.1b}$$

$$a_{33}u_3^{(n+1)} = f_3 - \left(a_{31}u_1^{(n+1)} + a_{32}u_2^{(n+1)}\right). \tag{4.4.1c}$$

Let us re-write these by adding and subtracting $a_{jj}u_j^{(n)}$ to the right-hand side of each equation:

$$a_{11}u_1^{(n+1)} = a_{11}u_1^{(n)} + \left[f_1 - \left(a_{11}u_1^{(n)} + a_{12}u_2^{(n)} + a_{13}u_3^{(n)}\right)\right], \tag{4.4.2a}$$

$$a_{22}u_2^{(n+1)} = a_{22}u_2^{(n)} + \left[f_2 - \left(a_{21}u_1^{(n+1)} + a_{22}u_2^{(n)} + a_{23}u_3^{(n)}\right)\right], \tag{4.4.2b}$$

$$a_{33}u_3^{(n+1)} = a_{33}u_3^{(n)} + \left[f_3 - \left(a_{31}u_1^{(n+1)} + a_{32}u_2^{(n+1)} + a_{33}u_3^{()}\right)\right]. \tag{4.4.2c}$$

After division by a_{jj}, the terms in square brackets in these expresssions can be seen as "corrections" which, when added to $u_j^{(n)}$, generate a better estimate of $u_j^{(n+1)}$. If these "corrections" all have the same sign, as it often happens for elliptic problems, it may be expected that increasing somewhat the magnitude of the "correction" will result in a faster convergence. This observation suggests to set, in place of (4.4.1) or (4.4.2),

$$a_{11}u_1^{(n+1)} = a_{11}u_1^{(n)} + \omega\left[f_1 - \left(a_{11}u_1^{(n)} + a_{12}u_2^{(n)} + a_{13}u_3^{(n)}\right)\right], \tag{4.4.3a}$$

$$a_{22}u_2^{(n+1)} = a_{22}u_2^{(n)} + \omega\left[f_2 - \left(a_{21}u_1^{(n+1)} + a_{22}u_2^{(n)} + a_{23}u_3^{(n)}\right)\right], \tag{4.4.3b}$$

$$a_{33}u_3^{(n+1)} = a_{33}u_3^{(n)} + \omega\left[f_3 - \left(a_{31}u_1^{(n+1)} + a_{32}u_2^{(n+1)} + a_{33}u_3^{(n)}\right)\right], \tag{4.4.3c}$$

with $\omega > 1$. In matrix form we can re-write these relations as

$$\mathsf{D}\mathbf{u}^{(n+1)} = \mathsf{D}\mathbf{u}^{(n)} + \omega\left[\mathbf{f} - \mathsf{D}\mathbf{u}^{(n)} - \mathsf{L}\mathbf{u}^{(n+1)} - \mathsf{U}\mathbf{u}^{(n)}\right], \tag{4.4.4}$$

or

$$\frac{\mathsf{D} + \omega\mathsf{L}}{\omega}\mathbf{u}^{(n+1)} = \left(\frac{1-\omega}{\omega}\mathsf{D} - \mathsf{U}\right)\mathbf{u}^{(n)} + \mathbf{f}. \tag{4.4.5}$$

Once written in this form, the method becomes applicable to a general $N \times N$ system. This relations defines the method of *successive over-relaxation*, or *SOR*. A simple way to remember this method is to note that, if $\mathbf{u}_{GS}^{(n)}$ denotes the result of the Gauss-Seidel method at iteration n, (4.4.5) is simply[21]

$$\mathbf{u}^{(n+1)} = \omega\mathbf{u}_{GS}^{(n)} + (1-\omega)\mathbf{u}^{(n)}. \tag{4.4.6}$$

as is evident upon comparison with (4.3.5).

A slightly different variant may be formulated by re-writing (4.4.5) identically as

$$\frac{\mathsf{D} + \mathsf{L} - (1-\omega)\mathsf{L}}{\omega}\mathbf{u}^{(n+1)} = \left(\frac{1-\omega}{\omega}\mathsf{D} - \mathsf{U}\right)\mathbf{u}^{(n)} + \mathbf{f}. \tag{4.4.7}$$

and then moving the term $(1-\omega)\mathsf{L}\mathbf{u}^{(n+1)}$ to the right-hand side as, approximately, $(1-\omega)\mathsf{L}\mathbf{u}^{(n)}$. In this way the method becomes

$$\frac{\mathsf{D} + \mathsf{L}}{\omega}\mathbf{u}^{(n+1)} = \left[\frac{1-\omega}{\omega}(\mathsf{D} + \mathsf{L}) - \mathsf{U}\right]\mathbf{u}^{(n)} + \mathbf{f}. \tag{4.4.8}$$

[21] One way to prove this relation is to calculate $(\mathsf{D} + \mathsf{L})\mathbf{u}^{n+1} - \omega\mathbf{u}_{GS}^{n+1}$ using (4.3.5) for $\mathbf{u}_{GS}^{n+1}$. In this way one finds $[(1-\omega)/\omega](\mathsf{D} + \mathsf{L})\mathbf{u}^n$, from which (4.4.6) follows.

The question of optimizing the choice of the parameter ω to achieve as fast a convergence rate as possible is a complex matter with a large literature. We limit ourselves to noting that it can be shown that in most cases (and, in particular, when the matrix derives from the discretization of elliptic equations), for convergence it is necessary to take $\omega < 2$. Thus, in most cases, the practical range for the parameter ω is $1 < \omega < 2$. In some special cases values of ω in the range $0 < \omega < 1$ are used and, in this case, the method is called "under-relaxation".

A practical point to keep in mind is that, with the SOR method, the error often grows with the first few iterations before convergence sets in and the error starts decreasing.

4.5 When to stop iterating?

An obvious question that arises with all iterative methods is when to stop the iterative process. An obvious – and superficial – answer is that we should stop when the solution that we have found is "close enough" to the "exact" solution, but the real question is how to judge this closeness. The exact solution of the linear system (4.2.1) is such that

$$a_{11}u_1 + a_{12}u_2 + a_{13}u_3 \quad - f_1 \;=\; 0\,, \tag{4.5.1a}$$
$$a_{21}u_1 + a_{22}u_2 + a_{23}u_3 \quad - f_2 \;=\; 0\,, \tag{4.5.1b}$$
$$a_{31}u_1 + a_{32}u_2 + a_{33}u_3 \quad - f_3 \;=\; 0\,. \tag{4.5.1c}$$

An approximate solution will be such that the *residual*, namely the right-hand side of these equations is not zero but, if the solution is accurate, is "small". Since "small" is never an absolute quantity, but it is always relative to something else, what should we use to gauge the smallness of the residual? A good measure is the following

$$\begin{aligned}
\epsilon_1 &= \frac{|a_{11}u_1 + a_{12}u_2 + a_{13}u_3 - f_1|}{|a_{11}u_1| + |a_{12}u_2| + |a_{13}u_3| + |f_1|}\,,\\
\epsilon_2 &= \frac{|a_{21}u_1 + a_{22}u_2 + a_{23}u_3 - f_2|}{|a_{21}u_1| + |a_{22}u_2| + |a_{23}u_3| + |f_2|}\,,\\
\epsilon_3 &= \frac{|a_{31}u_1 + a_{32}u_2 + a_{33}u_3 - f_3|}{|a_{31}u_1| + |a_{32}u_2| + |a_{33}u_3| + |f_3|}\,.
\end{aligned} \tag{4.5.2}$$

We should stop the iterative process when the *largest* one of these errors is smaller than some prescribed tolerance, for example 10^{-4} or 10^{-6} or some other small number dependent on the accuracy that we need.

The extension of this criterion to a general system of the form

$$\sum_{k=1}^{N} a_{jk}u_k \;=\; f_j\,, \qquad j \;=\; 1,\, 2,\, 3,\, \ldots,\, N\,, \tag{4.5.3}$$

is immediate. We define

$$\epsilon_j \;=\; \frac{|\sum_{k=1}^{N} a_{jk}u_k - f_j|}{\sum_{k=1}^{N} |a_{jk}u_k| + |f_j|}\,, \tag{4.5.4}$$

and we focus on the magnitude of the *largest* one of the ϵ_j's.

4.6 Convergence

For obvious reasons the issue of convergence is central in the development and applications of iterative nethods. A very general way to describe such methods is to say that the matrix is decomposed as

$$\mathsf{A} \;=\; \mathsf{N} - \mathsf{P}\,, \tag{4.6.1}$$

and that the method consists in generating the recursion

$$\mathsf{N}\mathbf{u}^{(n+1)} \;=\; \mathbf{f} + \mathsf{P}\mathbf{u}^{(n)}\,, \tag{4.6.2}$$

with the decomposition chosen in such a way that the system in the left-hand side of this equation is, in some sense, "easy" to solve. The splittings corresponding to the methods described in the previous sections are summarized in Table 4.6.1.

Method	N	P
Jacobi, Eq. (4.2.5)	D	$-(\mathsf{L}+\mathsf{U})$
Gauss-Seidel, Eq. (4.3.5)	$\mathsf{D}+\mathsf{L}$	$-\mathsf{U}$
SOR no. 1, Eq. (4.4.5)	$\omega^{-1}\mathsf{D}+\mathsf{L}$	$\omega^{-1}(1-\omega)\mathsf{D}-\mathsf{U}$
SOR no. 2, Eq. (4.4.8)	$\omega^{-1}(\mathsf{D}+\mathsf{L})$	$\omega^{-1}(1-\omega)(\mathsf{D}+\mathsf{L})-\mathsf{U}$

Table 4.6.1: Splitting $\mathsf{A} = \mathsf{N} - \mathsf{P}$ for various iterative methods.

At each step of the iteration the error is

$$\epsilon_k \;=\; \mathbf{u}^{(n)} - \mathbf{u}_{ex}\,, \tag{4.6.3}$$

with the "exact" solution satisfying $\mathsf{N}\mathbf{u}_{ex} = \mathbf{f} + \mathsf{P}\mathbf{u}_{ex}$. Using this relation we find

$$\begin{aligned} \mathsf{N}\epsilon_{k+1} &= \mathsf{N}\mathbf{u}^{(n+1)} - \mathsf{N}\mathbf{u}_{ex} \;=\; (\mathbf{f} + \mathsf{P}\mathbf{u}^{(n)}) - (\mathbf{f} + \mathsf{P}\mathbf{u}_{ex}) \\ &= \mathsf{P}\mathbf{u}^{(n)} - \mathsf{P}\mathbf{u}_{ex} \;=\; \mathsf{P}\epsilon_k\,. \end{aligned} \tag{4.6.4}$$

We therefore conclude that

$$\epsilon_{k+1} = \mathsf{N}^{-1}\mathsf{P}\epsilon_k\,, \tag{4.6.5}$$

i.e.

$$\epsilon_1 = \mathsf{N}^{-1}\mathsf{P}\epsilon_0\,, \qquad \epsilon_2 = \mathsf{N}^{-1}\mathsf{P}\epsilon_1 \;=\; (\mathsf{N}^{-1}\mathsf{P})(\mathsf{N}^{-1}\mathsf{P})\epsilon_0 \;=\; (\mathsf{N}^{-1}\mathsf{P})^2\epsilon_0 \tag{4.6.6}$$

and, more generally,

$$\epsilon_k = (\mathsf{N}^{-1}\mathsf{P})^k\epsilon_0\,. \tag{4.6.7}$$

We see that, for the error to tend to zero irrespective of the value of ϵ_0, i.e., irrespective of the initial "guess" to the solution, it is necessary that, in some sense,

$$\text{“}\lim_{k\to\infty}(\mathsf{N}^{-1}\mathsf{P})^k \;=\; 0''\,. \tag{4.6.8}$$

Here we have used quotation marks because, while this relation would have an obvious meaning for numbers, it is not immediately clear how we should understand it for matrices. To address this point we need to introduce the *spectral radius* ρ of the matrix $\mathsf{N}^{-1}\mathsf{P}$, denoted by $\rho(\mathsf{N}^{-1}\mathsf{P})$. The spectral radius is the modulus of the largest eigenvalue of the matrix. The error decreases and the method converges provided $\rho < 1$. At each iteration, the error decreases by a factor ρ so that, after k iterations, it has decreased by a factor ρ^k and, therefore, the faster the smaller ρ. If we want an error smaller than 10^{-m}, for some positive integer m, we need k iterations such that

$$\rho^k < 10^{-m}\,, \tag{4.6.9}$$

and, upon taking logs to the base 10,

$$k\log_{10}\rho \;<\; -m \qquad\Longrightarrow\qquad k(-\log_{10}\rho) \;>\; m\,, \tag{4.6.10}$$

i.e.

$$k \;>\; \frac{m}{-\log_{10}\rho}\,. \tag{4.6.11}$$

Unfortunately it is often impractical to calculate the spectral radius. For this reason use is made of approximations to ρ which rely on the *norm* of the matrix, which we now define.

4.6.1 Matrix norm

There are several ways in which we can define the *norm* of vectors. In a two-dimensional space the Euclidean, or L^2 norm, is just

$$\| \mathbf{x} \|_2 = \sqrt{x_1^2 + x_2^2}\,, \tag{4.6.12}$$

and has the well-known meaning of (geometric) distance from the origin. The L^1 norm is defined by

$$\| \mathbf{x} \|_1 = |x_1| + |x_2|\,. \tag{4.6.13}$$

These are just two examples of many possible definitions. A general definition (but, by no means, the most general one) that includes both of these is the L^p norm:

$$\| \mathbf{x} \|_p = \left(|x_1|^p + |x_2|^p\right)^{1/p}\,. \tag{4.6.14}$$

In a space with N dimensions, the sums in these expressions will include N terms. Generally speaking, the norm of a vector may be thought of as a generalized "length" of the vector, or a generalized "distance from the origin" of the end-point of the vector.

It is readily seen from these defintions that, for any constant c,

$$\| c\mathbf{x} \| = |c|\ \| \mathbf{x} \|\,, \tag{4.6.15}$$

a property that is generally valid for any norm.

Consider now a 2×2 matrix B operating on a vector $\mathbf{x} = (x_1,\, x_2)$ to produce a new vector $\mathbf{y} = (y_1,\, y_2)$:

$$\mathsf{B}\mathbf{x} \;=\; \mathbf{y}\,, \tag{4.6.16}$$

or, explicitly,

$$\begin{vmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{vmatrix} \begin{vmatrix} x_1 \\ x_2 \end{vmatrix} = \begin{vmatrix} y_1 \\ y_2 \end{vmatrix}\,. \tag{4.6.17}$$

We define the norm of the matrix B by

$$\| \mathsf{B} \| = \max_{\|\mathbf{x}\|=1} \| \mathsf{B}\mathbf{x} \|\ . \tag{4.6.18}$$

Thus, the norm of a matrix is the maximum (generalized) "stretch" that the matrix can produce on a vector of unit norm. If $\| \mathsf{B} \| < 1$, unit vectors are actually "shrunk" rather than "stretched". It is evident that the definition of matrix norm depends upon the definition used for the norm of vectors. We say that a vector norm induces the corresponding matrix norm.

Since the norm is the maximum lengthening possible for a vector of norm 1, it is evident that, for any other vector $\mathbf{y}$ of norm 1,

$$\| \mathsf{B}\mathbf{y} \| \le \| \mathsf{B} \|\ . \tag{4.6.19}$$

For any vector $\mathbf{w}$ and any norm, the vector $\mathbf{w}/\| \mathbf{w} \|$ has unit norm because, by (4.6.15),

$$\| (\mathbf{w}/\| \mathbf{w} \|) \| = (1/\| \mathbf{w} \|)\ \| \mathbf{w} \| = 1\,. \tag{4.6.20}$$

Therefore, according to (4.6.19),

$$\| \mathsf{B}(\mathbf{w}/\| \mathbf{w} \|) \| \le \| \mathsf{B} \|\,, \tag{4.6.21}$$

from which, by (4.6.15) and multiplying by $\| \mathbf{w} \|$, we deduce that

$$\| \mathsf{B}\mathbf{w} \| \le \| \mathsf{B} \|\, \| \mathbf{w} \|\ . \tag{4.6.22}$$

If we set $\mathsf{B} = \mathsf{N}^{-1}\mathsf{P}$, (4.6.7) becomes

$$\epsilon_k = \mathsf{B}^k \epsilon_0\,. \tag{4.6.23}$$

From (4.6.22) we have

$$\| \epsilon_k \| = \| \mathsf{B}\epsilon_{k-1} \| = \| \mathsf{B} \| \, \| \epsilon_{k-1} \| \leq (\| \mathsf{B} \|)^2 \, \| \epsilon_{k-2} \| , \tag{4.6.24}$$

and so on, so that

$$\| \epsilon_k \| \leq (\| \mathsf{B} \|)^k \, \| \epsilon_0 \| . \tag{4.6.25}$$

This relation shows that a *sufficient* condition for the decrease of the error and, therefore, convergence of the method, is that $\| \mathsf{B} \| < 1$ so that the matrix $\mathsf{N}^{-1}\mathsf{P}$ "shrinks" rather than "stretches" the vectors to which it is applied. This condition, however, is only sufficient for two reasons. The first one is the presence of the "$\leq$" sign in (4.6.25): if the true upper limit is much smaller than $(\| \mathsf{B} \|)^k$, we might have convergence even though $\| \mathsf{B} \| > 1$. Secondly, as mentioned before, the true convergence criterion should be formulated in terms of the spectral radius ρ. It is a general property of all matrix norms that $\rho \leq \| \mathsf{B} \|$. Thus, if any norm of the matrix is less than 1, we are guaranteed that the spectral radius is also less than 1 and the iterative method converges, but it may also happen that $\| \mathsf{B} \| > 1$, while $\rho < 1$ and the method would still converge. In spite of these caveats, the calculation of some norms is relatively straightforward and herein lies their usefulness for the estimate of convergence.

It can be shown that the norm of an $N \times N$ matrix induced by the L^1 vector norm is given by

$$\| B \|_1 = \max_{1 \leq j \leq N} \sum_{i=1}^{N} |b_{ij}| . \tag{4.6.26}$$

A similar norm, which is also easy to calculate, is the "infinity norm", or L^∞ norm, defined by

$$\| B \|_\infty = \max_{1 \leq i \leq N} \sum_{j=1}^{N} |b_{ij}| . \tag{4.6.27}$$

Clearly, the L^∞ norm of B is the L^1 norm of the transpose of B. The norm induced by the L^2 vector norm "should be"

$$\| B \|_F = \sqrt{\sum_{i,j=1}^{N} a_{ij}^2} , \tag{4.6.28}$$

but, actually, it can be shown that $\| B \|_2 \leq \| B \|_F$; the norm (4.6.28) is called the *Frobenius norm*. The calculation of the actual L^2 norm is quite involved, which is very unfortunate because, in the case of symmetric matrices (which are often encountered) $\| \mathsf{B} \|_2 = \rho$ so that knowledge of $\| \mathsf{B} \|_2$ would directly tell us what ρ is. As before, if the Frobenius norm is less than 1, we have a sufficient condition for convergence.

For the Jacobi method $\mathsf{B} = -\mathsf{D}^{-1}(\mathsf{L} + \mathsf{U})$:

$$\begin{vmatrix} -1/a_{11} & 0 & 0 \\ 0 & -1/a_{22} & 0 \\ 0 & 0 & -1/a_{33} \end{vmatrix} \begin{vmatrix} 0 & a_{12} & a_{13} \\ a_{21} & 0 & a_{23} \\ a_{31} & a_{32} & 0 \end{vmatrix} = \begin{vmatrix} 0 & -a_{12}/a_{11} & -a_{13}/a_{11} \\ -a_{21}/a_{22} & 0 & -a_{23}/a_{22} \\ -a_{31}/a_{33} & -_{32}/a_{33} & 0 \end{vmatrix} . \tag{4.6.29}$$

Hence

$$\| \mathsf{B} \|_\infty = \max_{1 \leq i \leq N} \sum_{j=1, j \neq i}^{N} \left| \frac{a_{ij}}{a_{ii}} \right| < 1 \tag{4.6.30}$$

implies

$$|a_{ii}| > \sum_{j=1, j \neq i}^{N} |a_{ij}| \qquad \text{for all } i , \tag{4.6.31}$$

i.e., diagonal dominance. A similar conclusion holds for the Gauss-Seidel method. For the latter, it can also be shown that, if the matrix of the linear system is positive definite, convergence is always ensured. For the second form of the SOR method, Eq. (4.4.8), it can be shown that, if the eigenvalues of $-(\mathsf{D} + \mathsf{L})^{-1}\mathsf{U}$ are λ_i, with $|\lambda_i| < 1$, then those of the corresponding matrices of the SOR method are given by $\omega\lambda_i + 1 - \omega$ and these are smaller than $|\lambda_i|$, which implies a faster convergence, if $\omega > 1$.

4.6.2 Condition number

Another important consideration is the degree to which the solution is affected by errors in the right-hand side $\mathbf{f}$ due, e.g., to round-off. So suppose that we consider

$$\mathsf{A}\mathbf{u} = \mathbf{f} \qquad \text{and} \qquad \mathsf{A}(\mathbf{u}+\delta\mathbf{u}) = \mathbf{f}+\delta\mathbf{f}\,, \tag{4.6.32}$$

in which the first equation is the one we would really like to solve, while the second one is the one we can solve in practice given that the right-hand side is affected by some error $\delta\mathbf{f}$. In order to have confidence that the solution that we *can* calculate, namely $\mathbf{u}+\delta\mathbf{u}$, is a good approximation to the one we *would like* to have, namely $\mathbf{u}$, it is evidently necessary that a small error $\delta\mathbf{f}$ result in only a small difference $\delta\mathbf{u}$ in $\mathbf{u}$. Subtracting the first of (4.6.32) from the second we have

$$\mathsf{A}\delta\mathbf{u} = \delta\mathbf{f} \qquad \text{from which} \qquad \delta\mathbf{u} = \mathsf{A}^{-1}\delta\mathbf{f}\,. \tag{4.6.33}$$

Using (4.6.22) we conclude that

$$\| \delta\mathbf{u} \| = \| \mathsf{A}^{-1}\delta\mathbf{f} \| \leq \| \mathsf{A}^{-1} \| \, \| \delta\mathbf{f} \| \ . \tag{4.6.34}$$

On the other hand, from the first of (4.6.32), we have

$$\| \mathbf{f} \| = \| \mathsf{A}\mathbf{u} \| \leq \| \mathsf{A} \| \, \| \mathbf{u} \| \tag{4.6.35}$$

from which

$$\| \mathbf{u} \| \geq \frac{\| \mathbf{f} \|}{\| \mathsf{A} \|}\,. \tag{4.6.36}$$

For the ratio $\| \delta\mathbf{u} \| \ / \ \| \mathbf{u} \|$ we therefore have

$$\frac{\| \delta\mathbf{u} \|}{\| \mathbf{u} \|} \leq \| \mathsf{A}^{-1} \| \, \frac{\| \delta\mathbf{f} \|}{\| \mathbf{u} \|} \leq \| \mathsf{A} \| \, \| \mathsf{A}^{-1} \| \ \frac{\| \delta\mathbf{f} \|}{\| \mathbf{f} \|}\,. \tag{4.6.37}$$

Proceeding similarly, we deduce from (4.6.33) that $\| \delta\mathbf{f} \| \leq \| \mathsf{A} \| \, \| \delta\mathbf{u} \|$ and, from the first of (4.6.32), that $\| \mathbf{u} \| \leq \| \mathsf{A}^{-1} \| \, \| \mathbf{f} \|$ so that

$$\frac{\| \delta\mathbf{u} \|}{\| \mathbf{u} \|} \ \geq \ \frac{\| \delta\mathbf{f} \|}{\| \mathsf{A} \| \, \| \mathbf{u} \|} \ \geq \ \frac{1}{\| \mathsf{A} \| \, \| \mathsf{A}^{-1} \|} \, \frac{\| \delta\mathbf{f} \|}{\| \mathbf{f} \|}\,. \tag{4.6.38}$$

The quantity

$$K = \| \mathsf{A} \| \, \| \mathsf{A}^{-1} \| \tag{4.6.39}$$

is the *condition number* of the matrix A, and it can be thought of essentially as the modulus of the ratio of the largest to the smallest eigenvalue of A; K is therefore always larger than, or equal to, 1. Combining the previous results we can write that

$$\frac{1}{K} \frac{\| \delta\mathbf{f} \|}{\| \mathbf{f} \|} \ \leq \ \frac{\| \delta\mathbf{u} \|}{\| \mathbf{u} \|} \leq K \frac{\| \delta\mathbf{f} \|}{\| \mathbf{f} \|}\,. \tag{4.6.40}$$

This relation bounds the possible range of the magnitude of $\delta\mathbf{u}$, the error in $\mathbf{u}$, resulting from an inaccuracy $\delta\mathbf{f}$ in $\mathbf{f}$. If K is not too much larger than 1, we can have confidence that a small $\delta\mathbf{f}$ cannot have a catastrophic effect on the accuracy of our solution, but if K is much larger than 1, all we can say is that it is possible for $\| \delta\mathbf{u} \| \ / \ \| \mathbf{u} \|$ to be either much smaller or much larger than $\| \delta\mathbf{f} \| \ / \ \| \mathbf{f} \|$ so that the smallness of $\| \delta\mathbf{f} \| \ / \ \| \mathbf{f} \|$ in no way guarantees that of $\| \delta\mathbf{u} \| \ / \ \| \mathbf{u} \|$. In this case therefore our purported solution may in fact be very far from the one we would like to have. Since, in practice, we have neither information nor control on $\delta\mathbf{f}$, on the basis of this relation we should be at least suspicious of the solution that we have found. As a matter of fact, it can be shown that it is always possible to find a $\delta\mathbf{f}$ such that (4.6.37) holds with the equal sign. If we are unlucky enough that our $\delta\mathbf{f}$ is, or is close to, this particular vector, our solution may be quite different from the correct one that we want. Matrices such that $K \gg 1$ are labelled *ill-conditioned*. They are almost singular[22] and, since the solution – by whatever means – of a linear system ultimately is equivalent to dealing with the inverse of the matrix, the process is prone to large numerical errors.

[22] A singular matrix is not invertible and its condition number is infinity.

4.7 Gauss elimination

The methods described so far are iterative. There is a famous direct method to produce (in principle) an exact solution without iteration called *Gauss elimination*. In principle, this method is applicable to any (solvable) linear system whatever the structure of its matrix, and it reduces to the tri-diagonal algorithm when the matrix is tri-diagonal. To demonstrate the procedure let us consider once again the simple case of a 3×3 linear system:

$$\begin{vmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{vmatrix} \begin{vmatrix} u_1 \\ u_2 \\ u_3 \end{vmatrix} = \begin{vmatrix} f_1 \\ f_2 \\ f_3 \end{vmatrix} \tag{4.7.1}$$

or

$$a_{11}u_1 + a_{12}u_2 + a_{13}u_3 = f_1\,, \tag{4.7.2a}$$
$$a_{21}u_1 + a_{22}u_2 + a_{23}u_3 = f_2\,, \tag{4.7.2b}$$
$$a_{31}u_1 + a_{32}u_2 + a_{33}u_3 = f_3\,. \tag{4.7.2c}$$

We form the linear combination [(4.7.2a) – a_{21}/a_{11}(4.7.2b)]:

$$a_{21}u_1 + a_{22}u_2 + a_{23}u_3 - \frac{a_{21}}{a_{11}}\left(a_{11}u_1 + a_{12}u_2 + a_{13}u_3\right) = f_2 - \frac{a_{21}}{a_{11}}f_1 \tag{4.7.3}$$

and [(4.7.2c) – a_{31}/a_{11}(4.7.2a)]:

$$a_{31}u_1 + a_{32}u_2 + a_{33}u_3 - \frac{a_{31}}{a_{11}}\left(a_{11}u_1 + a_{12}u_2 + a_{13}u_3\right) = f_3 - \frac{a_{31}}{a_{11}}f_1 \tag{4.7.4}$$

Upon simplification these become

$$\left(a_{22} - \frac{a_{21}}{a_{11}}a_{12}\right)u_2 + \left(a_{23} - \frac{a_{21}}{a_{11}}a_{13}\right)u_3 = f_2 - \frac{a_{21}}{a_{11}}f_1 \tag{4.7.5}$$
$$\left(a_{32} - \frac{a_{31}}{a_{11}}a_{12}\right)u_2 + \left(a_{33} - \frac{a_{31}}{a_{11}}a_{13}\right)u_3 = f_3 - \frac{a_{31}}{a_{11}}f_1 \tag{4.7.6}$$

If we set

$$a_{22}^{(1)} = a_{22} - \frac{a_{21}}{a_{11}}a_{12}\,, \qquad a_{23}^{(1)} = a_{23} - \frac{a_{21}}{a_{11}}a_{13}\,, \qquad f_2^{(1)} = f_2 - \frac{a_{21}}{a_{11}}f_1\,, \tag{4.7.7}$$
$$a_{32}^{(1)} = a_{32} - \frac{a_{31}}{a_{11}}a_{12}\,, \qquad a_{33}^{(1)} = a_{33} - \frac{a_{31}}{a_{11}}a_{13}\,, \qquad f_3^{(1)} = f_3 - \frac{a_{31}}{a_{11}}f_1\,, \tag{4.7.8}$$

our system has been reduced to the form

$$a_{11}u_1 + a_{12}u_2 + a_{13}u_3 = f_1 \tag{4.7.9a}$$
$$a_{22}^{(1)}u_2 + a_{23}^{(1)}u_3 = f_2^{(1)} \tag{4.7.9b}$$
$$a_{32}^{(1)}u_2 + a_{33}^{(1)}u_3 = f_3^{(1)}\,. \tag{4.7.9c}$$

Now the procedure can be repeated for the last two equations: form [(4.7.9b) – $a_{32}^{(1)}/a_{22}^{(1)}$(4.7.9c)] to find

$$\left(a_{33}^{(1)} - \frac{a_{32}^{(1)}}{a_{22}^{(1)}}a_{23}^{(1)}\right)u_3 = f_3^{(1)} - \frac{a_{32}^{(1)}}{a_{22}^{(1)}}f_2^{(1)}\,. \tag{4.7.10}$$

Upon setting

$$a_{33}^{(2)} = a_{33}^{(1)} - \frac{a_{32}^{(1)}}{a_{22}^{(1)}}a_{23}^{(1)}\,, \qquad f_3^{(2)} = f_3^{(1)} - \frac{a_{32}^{(1)}}{a_{22}^{(1)}}f_2^{(1)}\,, \tag{4.7.11}$$

we have the three equations

$$\begin{aligned} a_{11}u_1 + a_{12}u_2 + a_{13}u_3 &= f_1 && \text{(4.7.12a)} \\ a_{22}^{(1)}u_2 + a_{23}^{(1)}u_3 &= f_2^{(1)} && \text{(4.7.12b)} \\ a_{33}^{(2)}u_3 &= f_3^{(2)}\,. && \text{(4.7.12c)} \end{aligned}$$

The last equation is readily solved:

$$u_3 \;=\; \frac{f_3^{(2)}}{a_{33}^{(2)}}\,. \tag{4.7.13}$$

Then, from (4.7.12b):

$$u_2 \;=\; \frac{f_2^{(1)} - a_{23}^{(1)}u_3}{a_{22}^{(1)}}\,, \tag{4.7.14}$$

and, from (4.7.12a):

$$u_1 \;=\; \frac{f_1 - (a_{12}u_2 + a_{13}u_3)}{a_{11}}\,. \tag{4.7.15}$$

This simple 3×3 example shows the nature of the procedure which consists of two stages. In the first stage, called *forward elimination*, the unknowns u_1, u_2, …, u_{N-1} are successively eliminated leaving a system with the structure:

$$\begin{aligned} a_{11}u_1 + a_{12}u_2 + a_{13}u_3 + \ldots a_{1N}u_N &= f_1 && \text{(4.7.16a)} \\ a_{22}^{(1)}u_2 + a_{23}^{(1)}u_3 + \ldots + a_{2N}^{(1)}u_N &= f_2^{(1)} && \text{(4.7.16b)} \\ a_{33}^{(2)}u_3 + \ldots + a_{3N}^{(2)}u_N &= f_3^{(2)} && \text{(4.7.16c)} \\ \ldots &\quad \ldots && \text{(4.7.16d)} \\ a_{NN}^{(N-1)}u_N &= f_N^{(N-1)}\,, && \text{(4.7.16e)} \end{aligned}$$

in which

$$a_{ij}^{(k)} \;=\; a_{ij}^{(k-1)} - \frac{a_{ik}^{(k-1)}}{a_{kk}^{(k-1)}}a_{kj}^{(k-1)}\,, \qquad f_i^{(k)} \;=\; f_i^{(k-1)} - \frac{a_{ik}^{(k-1)}}{a_{kk}^{(k-1)}} - f_{i-1}^{(k-1)}\,, \tag{4.7.17}$$

The second stage, called *back substitution*, consists in solving the equations in the opposite order starting from

$$u_N \;=\; \frac{f_N^{(N-1)}}{a_{NN}^{(N-1)}}\,, \tag{4.7.18}$$

and, for general i:

$$u_i \;=\; \frac{1}{a_{ii}^{(i-1)}}\left(f_i^{(i-1)} - \sum_{j=i+1}^{N} a_{ij}^{(i-1)}u_j\right)\,. \tag{4.7.19}$$

Applying this formula for $i = 1$ requires $f_1^{(0)}$ and $a_{1j}^{(0)}$, which are just f_1 and a_{1j}.

The procedure can run into trouble if one of the terms by which the equations are divided (which are called the *pivot elements*) is zero or very small. A procedure which proves useful (although there is no proof that it is optimal) is *pivoting*: The order of the equations is rearranged in such a way that the diagonal element is the largest available in modulus. So, for example, the coefficients multiplying u_1 in all the equations are examined and the equation with the largest such coefficient (in modulus) is chosen to be the first equation of the system; the second equation is then taken to be that which, among the remaining equations, has the largest coefficient (in modulus) multiplying u_2, and so on. This procedure should always be used with Gaussian elimination in the case of large systems as it can be shown that, otherwise, the procedure is unstable in the sense that small changes in the right-hand sides (e.g., due to round-off error) can result in spurious big changes in the solution.

4.8 Pivoting

We have seen the importance of diagonal dominance in ensuring the convergence of the Jacobi and Gauss-Seidel methods. Furthermore, in both methods, at some point it is necessary to divide by a_{ii} and, if this matrix element is zero or very small, we would encounter stability or overflow problems. The same difficulty arises in the Gauss elimination method when we divide by $a_{kk}^{(k-1)}$ as we have already noted.

To handle these situations we observe that the solution of the algebraic system is the same whatever the order in which the equations that constitute it are written. Since the coefficients in each equation are the entries of the system's matrix, changing the order of the equations is equivalent to changing the order in which the rows of the matrix are arranged. So, before proceeding with the algorithms, we identify in each equation the matrix element with the largest modulus, the so-called pivot element. Suppose that, for the j-the equation, this happens to be the element a_{jK} multiplying u_K. Then we change the order of the equations (or of the rows of the matrix) in such a way that this equation becomes the K-the equation of the algebraic system. For Gauss elimination this reordering of the equations is effected every time an unknown is eliminated. This process is called partial *pivoting*, to distinguish it from complete pivoting in which, in addition to rows, colums are also interchanged. In practice complete pivoting offers limited benefits and is more complicated to implement and is, therefore, seldom used.

To see the problem and how pivoting remedies it let us look at a specific example of two equations in two unknowns:

$$a_{11}u_1 + a_{12}u_2 = f_1 \tag{4.8.1a}$$

$$a_{21}u_1 + a_{22}u_2 = f_2\,. \tag{4.8.1b}$$

Proceeding as in the derivation of (4.7) we eliminate u_1 between the first and the second equation so that the system is reduced to the form

$$a_{11}u_1 + a_{12}u_2 = f_1 \tag{4.8.2a}$$

$$\left(a_{22} - \frac{a_{21}}{a_{11}}a_{12}\right)u_2 = f_2 - \tfrac{a_{21}}{a_{11}}f_1\,. \tag{4.8.2b}$$

Consider the specific case

$$a_{11} = 0.3\times 10^{-6}\,, \qquad a_{12} = 1\,, \qquad f_1 = 0.7 \tag{4.8.3}$$

$$a_{21} = 1\,, \qquad a_{22} = 1\,, \qquad f_2 = 0.9\,, \tag{4.8.4}$$

the solution of which, correct to 5 significant digits, is $u_1 = 0.20000$, $u_2 = 0.70000$. If we carry out arithmetic operations with 5 digits accuracy the second equation becomes

$$-0.33333\times 10^{7}u_2 = -0.23333\times 10^{7}\,, \tag{4.8.5}$$

from which $u_2 = 0.70000$ and therefore, upon substituting into the first equation, $u_1 = 0.00000$. With pivoting we would invert the order of the equations so that, in place of of (4.8.1), we would have

$$a_{21}u_1 + a_{22}u_2 = f_2 \tag{4.8.6a}$$

$$a_{11}u_1 + a_{12}u_2 = f_1\,. \tag{4.8.6b}$$

Upon eliminating u_1 as before we now have

$$a_{21}u_1 + a_{22}u_2 = f_2 \tag{4.8.7a}$$

$$\left(a_{12} - \frac{a_{22}}{a_{21}}a_{11}\right)u_2 = f_1 - \tfrac{a_{11}}{a_{21}}f_2\,, \tag{4.8.7b}$$

or, retaining 5 digits with proper rounding off,

$$u_1 + u_2 = 0.9 \tag{4.8.8a}$$

$$1.0000\,u_2 = 0.70000\,, \tag{4.8.8b}$$

from which we obtain the correct solution to the accuracy retained.

Chapter 5

Elliptic Equations II

5.1 Multi-dimensional elliptic equations

In two space dimensions referred to Cartesian coordinates the Poisson equation is

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \;=\; f(x,y)\,, \tag{5.1.1}$$

and the Helmholtz equation

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \lambda u \;=\; f(x,y)\,. \tag{5.1.2}$$

A specific features of elliptic problems is that boundary conditions need to be specified over the entire boundary. For a two-dimensional problem the boundary is the contour of the domain of interest. For example, in the case of an elastic membrane, the boundary condition around the periphery of the membrane would describe the deformation of the frame to which the membrane is attached. In this case u would be the elevation of the membrane above the planar configuration that it takes when the frame is flat.

As we have seen in the one-dimensional case, the simplest situation to consider is that of Dirichlet conditions, in which the function u itself is assigned on the boundary. The Neumann case, in which the normal derivative of u is assigned on the boundary, is handled by introducing ghost nodes as in the one-dimensional case.

5.2 Set-up for the numerical solution

We consider (5.1.1) in a rectangular domain $0 \le x \le L_x$, $0 \le y \le L_y$. The boundary conditions are

$$u(x, y=0) \;=\; U_0(x)\,, \quad u(x, y=L_y) \;=\; U_L(x)\,, \quad u(x=0, y) \;=\; V_0(y)\,, \quad u(x=L_x, y) \;=\; V_L(y)\,, \tag{5.2.1}$$

in which the four functions $U_{0,L}(x)$, $V_{0,L}(y)$ are prescribed. The right-hand side of the equation, $f(x,y)$, is also prescribed.

We divide the x axis into $M+1$ equal segments, each of length Δx, by introducing M equi-spaced points so that $\Delta x = L_x/(M+1)$. We index these points by j so that $x_j = j\,\Delta x$; evidently $x_0 = 0$ and $x_{M+1} = L_x$. We proceed similarly with the y axis by introducing N equi-spaced points generating equal segments of length $\Delta y = L_y/(N+1)$; the generic point is $y_k = k\,\Delta y$ with $y_0 = 0$ and $y_{N+1} = L_y$. In this way the domain of interest is covered by a grid of points (x_j, y_k). Our objective is to find an approximate solution of the equation at each one of these points. For simplicity of writing we use the notation

$$u_{j,k} \;=\; u(x_j, y_k)\,. \tag{5.2.2}$$

As in the one-dimensional case, we need to approximate the Poisson equation (5.1.1) at the point x_j, y_k:

$$\left[\left(\frac{\partial^2 u}{\partial x^2}\right) + \left(\frac{\partial^2 u}{\partial y^2}\right) - f(x,y)\right]_{x=x_j, y=y_k} \;=\; 0\,. \tag{5.2.3}$$

There are several ways in which we can discretize the differential operator. The simplest one, on which we focus first, consists in using the results for the derivatives in one dimension. In so doing, we interpret, e.g.,

$\partial^2 u/\partial x^2$ as $(\partial^2 u/\partial x^2)_y$, namely as the derivative with respect to x taken keeping the other variable y fixed, and conversely for $\partial^2 u/\partial y^2$. If we follow this procedure, we can use directly the familiar second-derivative formula derived before. For the first one we have

$$\left[\left(\frac{\partial^2 u}{\partial x^2}\right)_y\right]_{x=x_j,y=y_k} = \frac{u_{j-1,k} - 2u_{jk} + u_{j+1,k}}{\Delta x^2} + O(\Delta x^2)\,. \tag{5.2.4}$$

Since y is kept constant, the second index remains fixed at k. Similarly, for the other term,

$$\left[\left(\frac{\partial^2 u}{\partial y^2}\right)_x\right]_{x=x_j,y=y_k} = \frac{u_{j,k-1} - 2u_{jk} + u_{j,k+1}}{\Delta y^2} + O(\Delta y^2)\,. \tag{5.2.5}$$

We will see later in section 5.4 that other alternatives to this procedure exist. With these steps the equation becomes

$$\frac{u_{j-1,k} - 2u_{jk} + u_{j+1,k}}{\Delta x^2} + \frac{u_{j,k-1} - 2u_{jk} + u_{j,k+1}}{\Delta y^2} = f_{jk}\,. \tag{5.2.6}$$

This discretization of the Laplace operator, which has an accuracy of the second order, is the one most widely used. In the special case $\Delta x = \Delta y = \Delta$ this becomes

$$u_{j-1,k} + u_{j+1,k} + u_{j,k-1} + u_{j,k+1} - 4u_{jk} = \Delta^2 f_{jk}\,. \tag{5.2.7}$$

Approximating the equation relies therefore on the five-node *stencil* shown in figure 5.2.1.

If we want to write these equations in matrix form we must arrange the u_{jk}'s in a single vector $\mathbf{u}$. There are different ways to do this. One way (used in MATLAB by default) is to stack all the columns of the matrix u_{jk} "one on top" of the other. With reference to figure 5.2.1), this means starting from the node in the lower left corner, moving horizontally to the right and, when the last node of that line is reached, going to the first node of the line above and so forth. In this way the matrix gets stored into the one-dimensional vector (of which we write the transpose to save space)

$$\begin{aligned}\mathbf{u}^T = \Big| \; & u_{11} \quad u_{21} \quad u_{31} \quad \dots \quad u_{M1} \quad u_{12} \quad u_{22} \quad u_{32} \quad \dots \quad u_{M2} \\ & \dots \quad u_{j-2,k-1} \quad u_{j-1,k-1} \quad u_{j,k-1} \quad u_{j+1,k-1} \quad \dots \quad u_{M,k-1} \\ & \dots \quad u_{j-1,k} \quad u_{j,k} \quad u_{j+1,k} \quad u_{j+2,k} \quad \dots \quad u_{M,k} \\ & \dots \quad u_{j-1,k+1} \quad u_{j,k+1} \quad u_{j+1,k+1} \quad \dots \quad u_{M,k+1} \quad \dots \Big|\,.\end{aligned} \tag{5.2.8}$$

$C++$ uses the opposite convention, with the rows, rather than the columns, stacked "on top of each other":

$$\begin{aligned}\mathbf{u}^T = \Big| \; & u_{11} \quad u_{12} \quad u_{13} \quad \dots \quad u_{1N} \quad u_{21} \quad u_{22} \quad u_{23} \quad \dots \quad u_{2N} \\ & \dots \quad u_{j-1,k-2} \quad u_{j-1,k-1} \quad u_{j-1,k} \quad u_{j-1,k+1} \quad \dots \quad u_{j-1,N} \\ & \dots \quad u_{j,k-2} \quad u_{j,k-1} \quad u_{j,k} \quad u_{j,k+1} \quad \dots \quad u_{j,N} \\ & \dots \quad u_{j+1,k-2} \quad u_{j+1,k-1} \quad u_{j+1,k} \quad \dots \quad u_{j+1,N} \quad \dots \Big|\,.\end{aligned} \tag{5.2.9}$$

Let us consider the corresponding form of the matrix for the linear system (5.2.6) if we use the MATLAB convention (5.2.8). Let us assume, for simplicity of writing, that $\Delta x = \Delta y = \Delta$. After some rearrangement, (5.2.6) written for the point (j,k) may be written as

$$\begin{aligned}&\dots + 0u_{j-1,k-1} + u_{j,k-1} + 0u_{j+1,k-1} + \dots \\ &\quad \dots + u_{j-1,k} - 4u_{jk} + u_{j+1,k} + 0u_{j+2,k} + \dots \\ &\quad\quad \dots + 0u_{j-1,k+1} + u_{j,k+1} + 0u_{j+1,k+1} + \dots = \Delta^2 f_{jk}\,,\end{aligned} \tag{5.2.10}$$

where we have inserted some terms with a 0 coefficient and some dots to signify the absence of other terms with y-index k and $k \pm 1$. Written in a similar way, the equation for the point $(j+1,k)$, which is found from

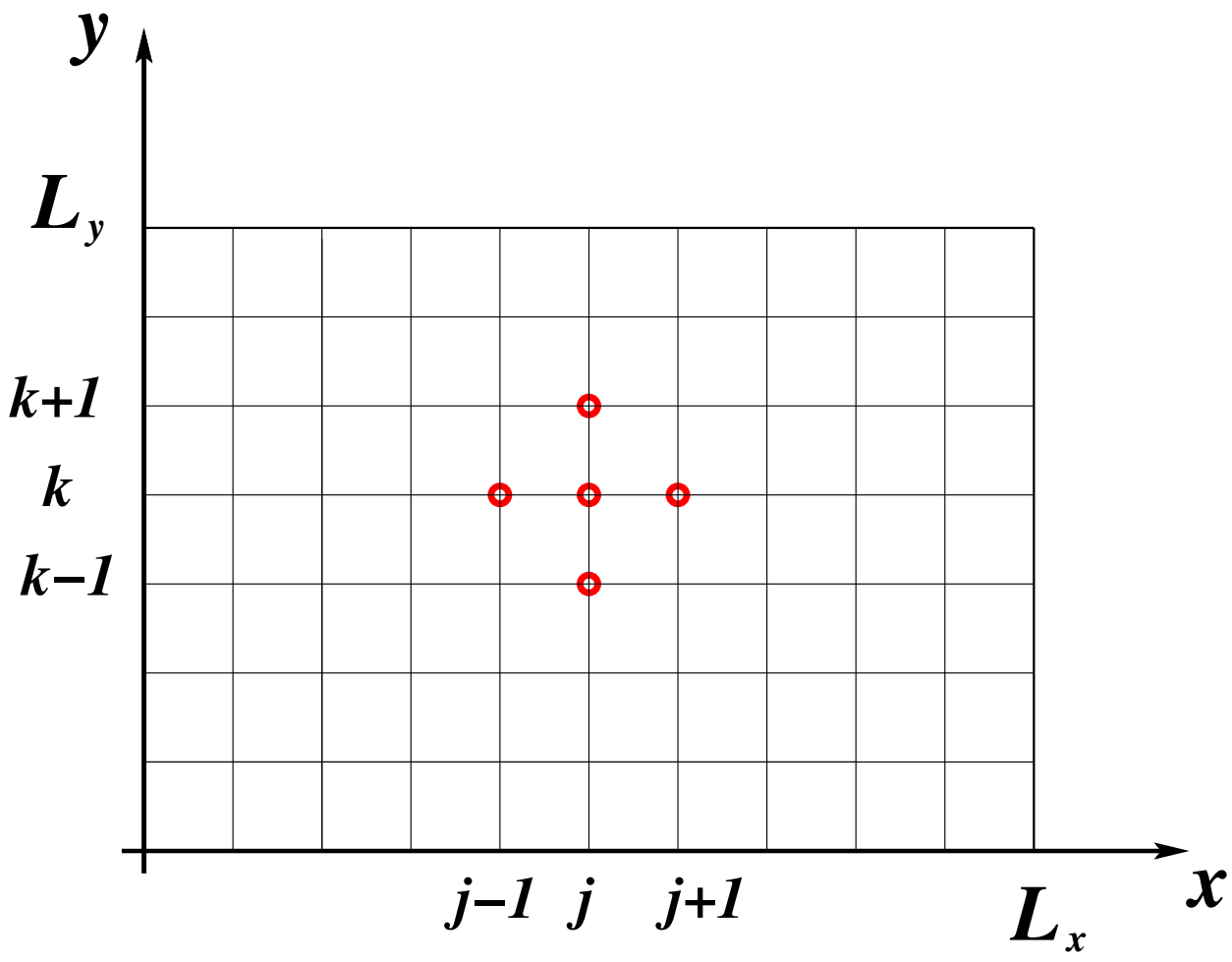


Figure 5.2.1: Discretization of the Laplacian operator in two dimensions.

the previous one by simply incrementing the index j by 1, is

$$\begin{aligned} &\ldots 0u_{j,k-1} + u_{j+1,k-1} + 0u_{j+2,k-1} + \ldots \\ &\quad \ldots + u_{j,k} - 4u_{j+1,k} + u_{j+2,k} + 0u_{j+3,k} + \ldots \\ &\qquad \ldots + 0u_{j,k+1} + u_{j+1,k+1} + 0u_{j+2,k+1} + \ldots = \Delta^2 f_{j+1,k}\,. \end{aligned} \tag{5.2.11}$$

By inspection we then see the structure of the linear system in matrix form:

$$\begin{vmatrix} \ldots & 0 & 1 & 0 & 0 & \ldots & 0 & 0 & 1 & -4 & 1 & 0 & \ldots & 0 & 1 & 0 & 0 & \ldots \\ \ldots & 0 & 0 & 1 & 0 & 0 & \ldots & 0 & 0 & 1 & -4 & 1 & 0 & \ldots & 0 & 1 & 0 & \ldots \end{vmatrix} \begin{vmatrix} \ldots \\ u_{j-1,k-1} \\ u_{j,k-1} \\ u_{j+1,k-1} \\ \ldots \\ u_{j-1,k} \\ u_{jk} \\ u_{j+1,k} \\ u_{j+2,k} \\ \ldots \\ u_{j-1,k+1} \\ u_{j,k+1} \\ u_{j+1,k+1} \\ \ldots \end{vmatrix} = \Delta^2 \begin{vmatrix} \ldots \\ f_{j-1,k-1} \\ f_{j,k-1} \\ f_{j+1,k-1} \\ \ldots \\ f_{j-1,k} \\ f_{jk} \\ f_{j+1,k} \\ f_{j+2,k} \\ \ldots \\ f_{j-1,k+1} \\ f_{j,k+1} \\ f_{j+1,k+1} \\ \ldots \end{vmatrix}\,. \tag{5.2.12}$$

This matrix is built by placing on the main diagonal and the two adjacent ones the terms involved in the derivative of u along the x direction (except that the diagonal coefficient is -4 instead of -2). There is also one non-zero element to the left, corresponding to the coefficient of $u_{j,k-1}$, and one to the right, corresponding to the coefficient of $u_{j,k+1}$. In the next line the three terms corresponding to the x-derivative of $u_{j+1,k}$ are on the main diagonal and the two adjacent ones shifted by one place to the right, and so are the non-zero coefficients of $u_{j+1,k-1}$ and $u_{j+1,k+1}$. Thus, the matrix consists of the main diagonal, the diagonals above and below, and two other diagonals separated from the three central ones by diagonals of zeros, i.e., the matrix is *penta-diagonal* because it has five diagonals. A matrix such as this one in which many elements vanish is called *sparse*.

It is evident that only the diagonals of this matrix should be stored to avoid a useless waste of memory space. For even a simple discretization with 100 points in each direction there are 10,000 unknowns, and the matrix would consist of 10,000×10,000 = 10^8 elements.

5.3 Numerical solution

There are two basic methods of solution for a system of the form (5.2.12), iterative and direct. A third method is based on the use of the Fourier series and is only applicable to matrices with a special form. Actually, the matrix in (5.2.12) has this special form and such methods are therefore applicable to it, but we will not pursue this topic further.

Iterative methods such as Jacobi, Gauss-Seidel and successive over-relaxation are nearly always applicable, even though their rate of convergence may not be optimal. The Jacobi method applied to (5.2.6) with equal spacings in the two directions is

$$u_{j,k}^{(n+1)} = \frac{1}{4}\left[u_{j-1,k}^{(n)} + u_{j+1,k}^{(n)} + u_{j,k-1}^{(n)} + u_{j,k+1}^{(n)}\right] - \frac{\Delta^2}{4} f_{jk}\,. \tag{5.3.1}$$

It is interestng to note that the first term is just the average of u at the four points neighboring (x_j, y_k).[23] The Gauss-Seidel method uses in the right-hand side the values of u as soon as they become available and,

[23] This is the discrete version of the mean-value theorem applicable to functions that satisfy the Laplace equation.

therefore, its detailed form depends on how the nodes are traversed. If, for example, the nodes are traversed along horizontal lines starting from the bottom of the domain and moving up in y, none of the nodes with index $k+1$ will have been updated when one is working on the nodes of the line k, while all the nodes of the line $k-1$ will have been updated. Moving from left to right along the k line of nodes, all the nodes with first index less than the index of interest will also have been updated so that the scheme becomes

$$u_{j,k}^{(n+1)} = \frac{1}{4}\left[u_{j-1,k}^{(n+1)} + u_{j+1,k}^{(n)} + u_{j,k-1}^{(n+1)} + u_{j,k+1}^{(n)}\right] - \frac{\Delta^2}{4} f_{jk}\,. \tag{5.3.2}$$

Similarly, application of the SOR method would lead to

$$u_{j,k}^{(n+1)} = (1-\omega)u_{j,k}^{(n)} + \frac{\omega}{4}\left[u_{j-1,k}^{(n+1)} + u_{j+1,k}^{(n)} + u_{j,k-1}^{(n+1)} + u_{j,k+1}^{(n)}\right] - \omega\frac{\Delta^2}{4} f_{jk}\,. \tag{5.3.3}$$

Another method with similar convergence properties is the *alternating direction* method. Each iteraton step consists of two sub-steps. In the first one we solve a tri-diagonal system in the x direction "pretending" that we know the solution at the other nodes along the y direction:

$$u_{j-1,k}^{n+1/2} - 2u_{jk}^{n+1/2} + u_{j+1,k}^{n+1/2} = \Delta^2 f_{jk} - [u_{j,k-1}^{n} - 2u_{jk}^{n} + u_{k,k+1}^{n}]\,, \tag{5.3.4}$$

and, in the second one, we solve a tri-diagonal system in the y direction using the just updated values for the nodes along the x direction:

$$u_{j,k-1}^{n+1} - 2u_{jk}^{n+1} + u_{j,k+1}^{n+1} = \Delta^2 f_{jk} - [u_{j-1,k}^{n+1/2} - 2u_{jk}^{n+1/2} + u_{j+1,k}^{n+1/2}]\,. \tag{5.3.5}$$

Here we have affixed the superscript $n+1/2$ to the values updated with the first step to signify that this is only one of the two sub-steps that constitute the complete iteration step.

A variation of this method that can improve the convergence rate is the following:[24]

$$u_{j-1,k}^{n+1/2} + u_{j+1,k}^{n+1/2} - (2+\rho)u_{jk}^{n+1/2} = \Delta^2 f_{jk} - [u_{j,k-1}^{n} - (2-\rho)u_{jk}^{n} + u_{k,k+1}^{n}]\,, \tag{5.3.6}$$

$$u_{j,k-1}^{n+1} + u_{j,k+1}^{n+1} - (2+\rho)u_{jk}^{n+1} = \Delta^2 f_{jk} - [u_{j-1,k}^{n+1/2} - (2-\rho)u_{jk}^{n+1/2} + u_{j+1,k}^{n+1/2}]\,. \tag{5.3.7}$$

When convergence has been reached, for all j and k $u_{jk}^{n} = u_{jk}^{n+1/2} = u_{jk}^{n+1}$ and the extra terms multiplied by ρ cancel. However, a suitable choice of the parameter ρ can speed up convergence, much in the same way as a suitable choice for the successive over-relaxation parameter can speed up convergence of the SOR method. We will see later how this method can be viewed as the result of solving the original equation by looking for the long-time limit of a diffusion equation.

It can be shown that, when the parameter ρ is chosen in an optimal way, the convergence rate of this method is the same as that of successive over-relaxation. The substeps of this method are however somewhat more complicated than for the SOR method and, therefore, this method is used only in special cases.

More advanced methods of solution based on the matrix structure exist, such as the conjugate gradient, the strongly implicit method and others. Another powerful method to deal with elliptic problems is the multigrid method, which uses repeated passes through a series of progressively coarser grids, followed by repeated passes through progressively finer grids and so on. All these methods require more advanced knowedge of matrix theory and we will not describe them.

5.4 The multi-dimensional Taylor series

In discretizing the Laplace operator $\partial^2 u/\partial x^2 + \partial^2 u/\partial y^2$ in section 3.2 we have treated the variables x and y independently of each other. This is not the only option. For example, it can be shown that, for $\Delta x = \Delta y$,

$$\left[\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right]_{x_j, y_k} = \frac{1}{4\Delta x^2}\left[u_{i+1,j+1} + u_{i+1,j-1} + u_{i-1,j+1} + u_{i-1,j-1} - 4u_{ij}\right] + O(\Delta x^2)\,, \tag{5.4.1}$$

[24] Sometimes called the Peaceman-Rachford method.

in which, as before, $u_{i+1,j-1} = u(x_i + \Delta x, y_j - \Delta y)$ etc. The way to show this is to make use of the Taylor series for a function dependent simultaneously on two variables. Another situation in which we need to have recourse to such an extension of the Taylor series is if, for example, the equation to be discretized contains a mixed derivative such as $\partial^2 u/\partial x \partial y$.

As always, the primary tool to prove a relation such as (5.4.1) is to use a Taylor series extended, in this case, to two variables. The derivation is simple and starts from the familiar form with a single variable as follows:

$$u(x+h, y+k) = u(x, y+k) + h \left[\frac{\partial u}{\partial x}\right]_{x,y+k} + \frac{1}{2!} h^2 \left[\frac{\partial^2 u}{\partial x^2}\right]_{x,y+k} + O(h^3)\,. \tag{5.4.2}$$

Now we apply the single-variable formula to each term in the right-hand side. For the first one we have

$$u(x, y+k) = u(x, y) + k \left[\frac{\partial u}{\partial y}\right]_{x,y} + \frac{1}{2!} k^2 \left[\frac{\partial^2 u}{\partial y^2}\right]_{x,y} + O(k^3)\,. \tag{5.4.3}$$

For the second one it is convenient to set

$$v(x, y+k) = \left[\frac{\partial u}{\partial x}\right]_{x,y+k}\,, \tag{5.4.4}$$

because then we find

$$v(x, y+k) = v(x, y) + k \frac{\partial v}{\partial y} + O(k^2)\,. \tag{5.4.5}$$

If we are interested in second-order accuracy it is sufficient to keep only one term since this is multiplied by h in (5.4.2); thus we have

$$v(x, y+k) = \left[\frac{\partial u}{\partial x}\right]_{x,y+k} = \left[\frac{\partial u}{\partial x}\right]_{x,y} + k \left[\frac{\partial}{\partial y}\left(\frac{\partial u}{\partial x}\right)\right]_{x,y} + O(k^2) = \left[\frac{\partial u}{\partial x}\right]_{x,y} + k \left[\frac{\partial^2 u}{\partial y\, \partial x}\right]_{x,y} + O(k^2)\,, \tag{5.4.6}$$

and similarly, again keeping only the term that will survive to second order when multiplied by h^2,

$$\left[\frac{\partial^2 u}{\partial x^2}\right]_{x,y+k} = \left[\frac{\partial^2 u}{\partial x^2}\right]_{x,y} + O(k)\,. \tag{5.4.7}$$

Upon substituting back into (5.4.2) we then find

$$\begin{aligned} u(x+h, y+k) &= u(x, y) + k \frac{\partial u}{\partial y} + \frac{1}{2!} k^2 \frac{\partial^2 u}{\partial y^2} + O(h^3, k^3) \\ &\quad + h \left(\frac{\partial u}{\partial x} + k \frac{\partial^2 u}{\partial y \partial x} + O(k^2)\right) + \frac{1}{2!} h^2 \left(\frac{\partial^2 u}{\partial x^2} + O(k)\right) \\ &= u(x, y) + h \frac{\partial u}{\partial x} + k \frac{\partial u}{\partial y} + \frac{1}{2!} \left[h^2 \frac{\partial^2 u}{\partial x^2} + 2hk \frac{\partial^2 u}{\partial y \partial x} + k^2 \frac{\partial^2 u}{\partial y^2}\right] \\ &\quad + O(h^3, k^3, h^2 k, h k^2)\,. \end{aligned} \tag{5.4.8}$$

It should be stressed that, just as in the case of a single variable, u istelf and the partial derivatives in this relation are evaluated at the fixed point $(x,\, y)$ which is used as a starting point for the expansion.

It is easy to remember this formula if we note that the second-order term has a structure reminiscent of a square. Indeed, in symbolic form, we can write it as

$$\frac{1}{2}\left(h^2 \frac{\partial^2 u}{\partial x^2} + 2hk \frac{\partial^2 u}{\partial x \partial y} + k^2 \frac{\partial^2 u}{\partial y^2}\right) = \frac{1}{2}\left(h \frac{\partial}{\partial x} + k \frac{\partial}{\partial y}\right)^2 u\,. \tag{5.4.9}$$

Extending this notation we can write the two-variable Taylor series beyond the second order as

$$u(x+h,y+k) \;=\; \sum_{\ell=0}^{N} \frac{1}{\ell!}\left(h\frac{\partial}{\partial x}+k\frac{\partial}{\partial y}\right)^{\ell} u + \text{remainder}\,, \tag{5.4.10}$$

in which the remainder contains a sum of terms proportional to $h^n k^{N+1-n}$ with $n = 0, 1, \ldots, N+1$ similarly to the $O(\ldots)$ terms in (5.4.8).

With this formula we can reconsider the numerator of (5.4.1):

$$u_{j+1,k+1} + u_{j+1,k-1} + u_{j-1,k-1} + u_{j-1,k+1} - 4u_{jk}\,. \tag{5.4.11}$$

Let us take $\Delta x = \Delta y$ for simplicity. Then we find

$$u_{j+1,k\pm1} \;=\; u_{jk} + \Delta x\left(\frac{\partial u}{\partial x} \pm \frac{\partial u}{\partial y}\right) + \frac{1}{2!}\Delta x^2\left[\frac{\partial^2 u}{\partial x^2} \pm 2\frac{\partial^2 u}{\partial y \partial x} + \frac{\partial^2 u}{\partial y^2}\right] + \ldots \tag{5.4.12}$$

$$u_{j-1,k\pm1} \;=\; u_{jk} - \Delta x\left(\frac{\partial u}{\partial x} \mp \frac{\partial u}{\partial y}\right) + \frac{1}{2!}\Delta x^2\left[\frac{\partial^2 u}{\partial x^2} \mp 2\frac{\partial^2 u}{\partial y \partial x} + \frac{\partial^2 u}{\partial y^2}\right] + \ldots \tag{5.4.13}$$

Upon collecting terms we have

$$u_{j+1,k+1} + u_{j+1,k-1} + u_{j-1,k-1} + u_{j-1,k+1} \;=\; 4u_{jk} + 4\Delta^2\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) + O(\Delta x^4)\,, \tag{5.4.14}$$

and therefore we conclude that

$$\frac{1}{4\Delta x^2}\left[u_{j+1,k+1} + u_{j+1,k-1} + u_{j-1,k-1} + u_{j-1,k+1} - 4u_{jk}\right] \;=\; [\nabla^2 u]_{jk} + O(\Delta x^2)\,. \tag{5.4.15}$$

Thus, we have found a different way to approximate the Laplacian operator. In detail, the error term in this expression is found to be

$$\frac{\Delta x^2}{12}\left(\frac{\partial^4 u}{\partial x^4} + \frac{\partial^4 u}{\partial y^4} + 6\frac{\partial^4 u}{\partial x^2 \partial y^2}\right)\,. \tag{5.4.16}$$

While the formula (5.4.15) has an error of the same order as the earlier discretization (5.2.6), it is affected by a feature that can cause a major problem. Unlike (5.2.6), the nodes appearing in the formula for the point (j,k) are all different from those appearing in the formula for the neighboring point $(j+1,k)$. For example, if j is even, the formula involves nodes with indices $j \pm 1$ which are odd, while the formula for the neighboring point $(j+1,k)$, in which $j+1$ will now be odd, involves points with inidices j and $j+2$ which are even. The same thing happens with the other neighboring points $(j-1,k)$, $(j,k\pm1)$. As you can readily see by drawing a grid and marking the nodes, this this even-odd decoupling has the consequence that all the grid nodes get divided into two families that never "talk" to each other. Thus, the error level for nodes belonging to one family could be completely different from that on noes belonging to the other one. This feature renders this scheme risky and, for this reason, it is not recommended.

Chapter 6

The Diffusion Equation

6.1 The diffusion equation

In one space dimension the diffusion equation is

$$\frac{\partial u}{\partial t} \;=\; D\frac{\partial^2 u}{\partial x^2}\,. \tag{6.1.1}$$

The parameter D is the diffusivity. In a fluid mechanics problem it would be the kinematic viscosity for a problem in which the diffusing quantity u is momentum (or vorticity). In a heat transfer problem it would be the thermal diffusivity $D = k/\rho c_p$, and so on. The mathematical statement of the problem needs to be completed by adding an initial condition:

$$u(x, t = 0) \;=\; f(x)\,, \tag{6.1.2}$$

and boundary conditions at the end-points, e.g. $x = 0$ and $x = L$. These boundary conditions can be of the Dirichlet type:

$$u(x = 0, t) \;=\; g_0(t)\,, \qquad u(x = L, t) \;=\; g_L(t)\,, \tag{6.1.3}$$

of the Neumann type:

$$\left.\frac{\partial u}{\partial x}\right|_{x=0} \;=\; h_0(t)\,, \qquad \left.\frac{\partial u}{\partial x}\right|_{x=L} \;=\; h_L(t)\,, \tag{6.1.4}$$

or of the Dirichlet-type at one end and of the Neumann type at the other, or other types as well. It may be noted that, upon dividing (6.1.1) by D we have

$$\frac{\partial u}{\partial (Dt)} \;=\; \frac{\partial^2 u}{\partial x^2}\,, \tag{6.1.5}$$

which, upon setting $t^* = Dt$, is

$$\frac{\partial u}{\partial t^*} \;=\; \frac{\partial^2 u}{\partial x^2}\,. \tag{6.1.6}$$

Thus, we can always assume $D = 1$ provided we remember that, upon so doing, the time coordinate is modified.

The simplest situation to keep in mind to aid intuition may be that of the diffusion of heat. In this case u would represent the temperature in a one-dimensional body situation, e.g. a rod or a slab in which x would be the coordinate normal to the surfaces. The initial condition would be the initial temperature of the system. Dirichlet boundary conditions would represent imposed temperatures at the boundary of the domain, while Neumann boundary conditions would represent heat fluxes at the boundary.

6.2 Some simple analytic solutions

To gain some insight into what to expect from the numerical solution of the diffusion equation it may help to consider some elementary analytic solutions of (6.1.1).

Consider for example homogeneous Dirichlet conditions at the end-points, $u(x = 0, t) = u(x = L, t) = 0$ and let $f(x) = G_0 \sin(M\pi x/L)$, with A_0 a constant and M an integer. This evidently satisfies the boundary

conditions and it represents an initial oscillatory distribution of u. If $M = 1$ there is one half of a full oscillation (ie., from 0 to a maximum and then back to 0 as the phase goes from 0 to π), if $M = 2$ there are two half-oscillations, i.e., a complete oscillation (from 0 to a maximum, back to 0, then to a minimum and to 0 again as the phase goes from 0 to 2π) and so on. The number of half-oscillations increases as M is taken larger and larger.[25] We look for a solution of the form $u(x,t) = G(t)\sin(M\pi x/L)$ with $G(t=0) = G_0$. Upon substitution into (6.1.1) and cancellation of the sine function that appears on both sides we find

$$\frac{dG}{dt} = -D\left(\frac{\pi M}{L}\right)^2 G\,, \tag{6.2.1}$$

the pertinent solution of which is $G(t) = G_0\exp[-(\pi M/L)^2 Dt]$ so that

$$u(x,t) = G_0\exp\left(-\frac{\pi^2 M^2 Dt}{L^2}\right)\sin\frac{\pi M x}{L}\,. \tag{6.2.2}$$

This solution shows that the temperature oscillations for the M-th mode decay in time over a time scale of the order of $(L/\pi M)^2/D$. The slowest temperature distribution to decay is that with $M = 1$, the decay time scale for which is $L^2/(\pi^2 D)$. The property of oscillations with larger M values decaying faster than ones with small M values becomes intuitively clear in the context of heat transfer: when M is large, the temperature oscillations occur over short distances and, therefore,heat from a region with positive temperature does not have to travel very far to heat up a region with negative temperature. The time scale τ_M for the decay of the temperature oscillations is the inverse of the quantity multiplying t in (6.2.2) and can be written as $\tau_M \sim (L/M)^2/(\pi D)$. This shows that, during a time τ_M, heat travels over a distance of the order of L/M, which is of the separation between "hot" and "cold" regions. This is a general property of the diffusion equation: during a time τ, the diffusing quantity propagates over a distance ℓ of the order of $\ell \sim \sqrt{\pi D\tau}$ and, conversely, the time needed to propagate over a distance ℓ is of the order of $\tau \sim \ell^2/(\pi D)$. These characteristic *diffusion length* and *diffusion time* are an important property of the physics modelled by the diffusion equation which also strongly influence the numerical solution methods as we will see.

If the boundary conditions are still homogeneous but of the Neumann type, namely $[\partial u/\partial x]_{x=0} = [\partial u/\partial x]_{x=L} = 0$, one finds solutions of a similar form proportional to $\cos(M\pi x/L)$ rather than $\sin(M\pi x/L)$.

These elementary solutions have a special relevance in view of the fact that, according to the Fourier series, a general function $f(x)$ which vanishes at $x = 0$, L, or the derivative of which vanishes at $x = 0$, L, can be expressed as a superposition of sine or cosine terms similar to the ones considered above. Each term of this superposition decays in time accoridng to the previous results. Thus, the fine-scale features of f, which are embodied in terms with large M, decay much faster than features with a longer spatial scale. The slowest decay is associated with the feature with the largest length scale, namely $M = 1$.

6.3 Explicit discretization of the diffusion equation

A very straightforward – indeed, obvious – way to discretize (6.1.1) is:

$$\frac{u_j^{n+1} - u_j^n}{\Delta t} = D\frac{u_{j-1}^n - 2u_j^n + u_{j+1}^n}{\Delta x^2}\,, \tag{6.3.1}$$

where we have written $u_j^n = u(j\Delta x,\, n\Delta t)$. This is called the explicit method because u_j^{n+1} is explicitly given by (figure 6.3.1)

$$u_j^{n+1} = u_j^n + \lambda\left(u_{j-1}^n - 2u_j^n + u_{j+1}^n\right) \qquad \text{where} \qquad \lambda = \frac{D\Delta t}{\Delta x^2}\,. \tag{6.3.2}$$

[25] Recall that $k = 2\pi/\ell$ is the *wave number*, namely the number of full oscillations over a distance of 2π length units; this is similar to the angular frequency ω, which is the number of oscillations over a time interval of 2π time units. Since the number of complete oscillations is $M/2$, the wavelength is $L/(M/2)$, so that the wave number is $k = 2\pi/[L/(M/2)] = \pi M/L$ and the argument of the sine is $M\pi x/L = 2\pi x/\ell = kx$, very similar to the time dependence of a simple oscillator $\sin\omega t = \sin 2\pi t/T$, with T the period of the oscillation.

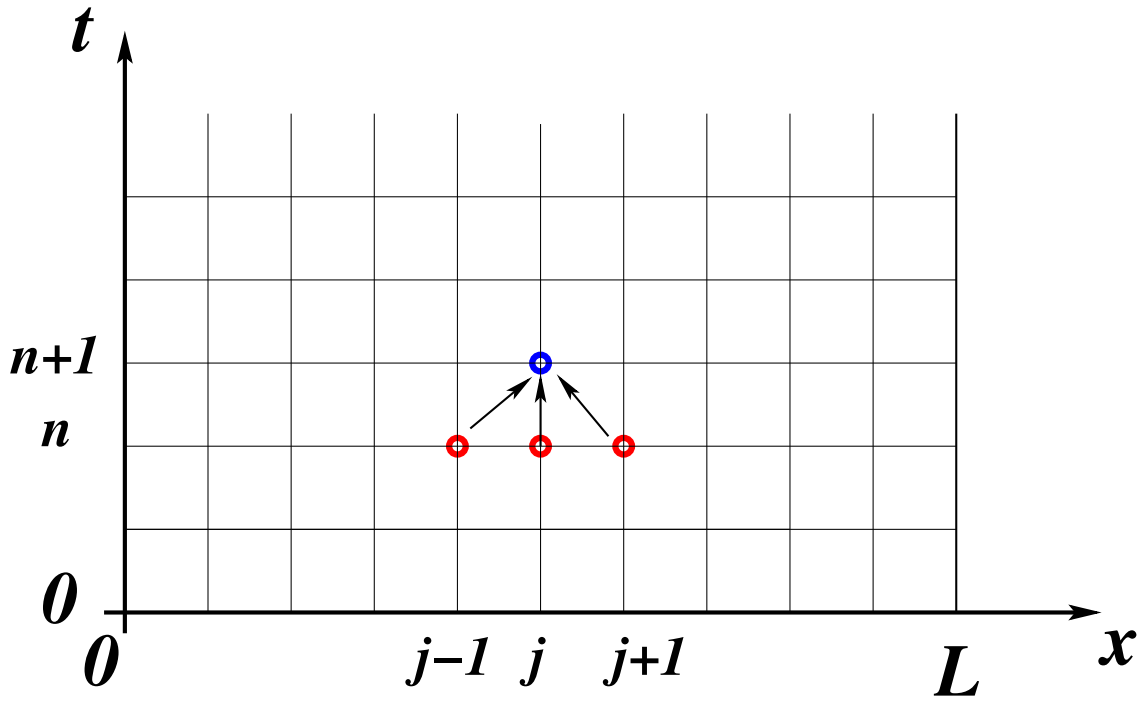


Figure 6.3.1: Schematic representation of the explicit method for the solution of the diffusion equation.

By keeping one extra term in the Taylor series expansions we find the order of magnitude of the truncation error:

$$\frac{\partial u}{\partial t} + \frac{\Delta t}{2}\frac{\partial^2 u}{\partial t^2} + \ldots = D\left(\frac{\partial^2 u}{\partial x^2} + \frac{\Delta x^2}{12}\frac{\partial^4 u}{\partial x^4} + \ldots\right). \tag{6.3.3}$$

By subtracting the original differential equation we find the truncation error

$$\epsilon_T = \left[\frac{u_j^{n+1} - u_j^n}{\Delta t} - D\frac{u_{j-1}^n - 2u_j^n + u_{j+1}^n}{\Delta x^2}\right] - \left[\frac{\partial u}{\partial t} - D\frac{\partial^2 u}{\partial x^2}\right]_j^n = \left[D\frac{\Delta x^2}{12}\frac{\partial^4 u}{\partial x^4} - \frac{\Delta t}{2}\frac{\partial^2 u}{\partial t^2} + \ldots\right]_j^n. \tag{6.3.4}$$

Thus we see that (6.3.1) approximates (6.1.1) with an error of order $O(\Delta x^2, \Delta t)$. Note however that if a relation is established between Δt and Δx when they both tend to zero, then the overall accuracy of the scheme might be different. For example, if $\Delta t/\Delta x$ is kept constant (i.e., whenever the time step is cut in half the space step is also cut in half), then the scheme has a global first-order accuracy, which would become second-order if $\Delta t/\Delta x^2$ were to be kept constant, which would require decreasing Δt by a factor of four whenever Δx is halved.

If Δt and Δx are reasonably small, we might say that our numerical solution is closer to the solution of the equation (6.3.3)

$$\frac{\partial u}{\partial t} = D\frac{\partial^2 u}{\partial x^2} + \frac{D\Delta x^2}{12}\frac{\partial^4 u}{\partial x^4} - \frac{\Delta t}{2}\frac{\partial^2 u}{\partial t^2}. \tag{6.3.5}$$

We can rewrite this equation by noting that, if u satisfies (6.1.1), then

$$\frac{\partial^2 u}{\partial t^2} = \frac{\partial}{\partial t}\left(\frac{\partial u}{\partial t}\right) = \frac{\partial}{\partial t}\left(D\frac{\partial^2 u}{\partial x^2}\right) = D\frac{\partial^2}{\partial x^2}\left(\frac{\partial u}{\partial t}\right) = D^2\frac{\partial^4 u}{\partial x^4}, \tag{6.3.6}$$

so that (6.3.5) approximatey becomes

$$\frac{\partial u}{\partial t} = D\frac{\partial^2 u}{\partial x^2} + \frac{D\Delta x^2}{2}\left(\frac{1}{6} - \lambda\right)\frac{\partial^4 u}{\partial x^4}. \tag{6.3.7}$$

Thus, when we apply the discretization shown above, what we are actually generating is a result closer to the solution of this equation rather than to that of the original diffusion equation. The difference between the two becomes smaller and smaller as Δx and Δt decrease, but is always present.[26]

An equation such as (6.3.7), which includes the effects of the leading terms of the truncation error, is called the *effective equation* because the numerical solution "effectively" solves this equation in the sense that it will be closer to the solution of this equation than to that of the original equation. Since this equation only contains the leading terms of the error, the numerical solution may be expected to be close to the solution of the effective equation only as long as Δx and Δt are sufficiently small.

Incidentally, we may observe that the special value $\lambda = \frac{1}{6}$ removes the error term shown leaving the terms indicated by dots in (6.3.3) to dominate the error. In this way, the order of accuracy of the discretization is increased. This is just a welcome accident happening in this particular case.

In order to see the consequences of the discretization error, let us solve (6.3.7) for the same problem considered in section 6.2. Proceeding as shown there we let $u(x,t) = G(t)\sin(\pi M x/L)$ and find

$$\frac{dG}{dt} = -D\left(\frac{\pi M}{L}\right)^2 (1-K)\, G\,, \tag{6.3.8}$$

where, to save writing, we have set

$$K = \frac{1}{2}\left(\frac{1}{6}-\lambda\right)\left(\frac{\pi M \Delta x}{L}\right)^2 . \tag{6.3.9}$$

The pertinent solution of (6.3.8) is

$$G(t) = G_0 \exp\left[-\left(\frac{\pi M}{L}\right)^2 (1-K)\, Dt\right] \tag{6.3.10}$$

and differs from the exact one (6.2.2) by the presence of the term K. If M is not large and $\Delta x/L$ is sufficiently small, K will be small. However, we see here the unwelcome possibility that, if $K > 1$, our numerical solution might grow instead of decaying! We have noted before that our interest in special solutions of this type is because a general initial condition can be decomposed into the sum of sines and/or cosines. Since this decomposition, in general, generates an infinite number of terms, there is no limit to how large M can be. It would appear therefore that the method will generate incorrect, exponentially growing solutions however Δx and Δt are chosen if M is large enough. Actually the situation is not as dire because, with increasing M, the higher order terms in the Taylor expansion become more and more important and it is not justified to stop the expansions at the level we have used in (6.3.3). Clearly, to investigate the question we need some more powerful tools.

6.4 The Lax equivalence theorem

We were led to the previous discretization formally by applying what we know about approximating first and second derivatives, but the remark at the end of the previous section casts some doubt on whether the solution (6.3.2) truly approximates the exact analytic solution of the original problem. What we would like is a guarantee that the difference between the analytic and numerical solutions at every point of the domain of interest can be arbitrarily small by choosing smaller and smaller time and space steps. This characteristic is referred to as *convergence* and it is evident that a method that does not enjoy it would be fairly useless. Thus, to take any numerical method seriously, we should establish whether it is convergent.

[26] The argument seems inconsistent because we have derived (6.3.6) by saying that u satisfies the original equation (6.1.1), but now we find that u satisfies something closer to (6.3.7). The proper way to think about this issue is to realize the successive approximation nature of the procedure. We have used (6.1.1) to estimate a term multiplied by Δt in (6.3.3). If we were to use (6.3.7) instead, the end result would be an additional term containing the factor Δt^2. Since some terms of this order have already been dropped in deriving (6.3.3), all terms of the same order should be dropped for consistency.

Equation (6.3.4) shows that the discretized differential equation can be made to differ as little as we please from the original differential equation by choosing a small Δx and Δt, but what we need is a guarantee that the *solutions* of the two equations become arbitrarily close. The reason this latter property does not follow directly from the closeness of the discretized and original equations is that, when we reduce Δx and Δt, the number of spatial points to fill the domain and the number of time steps to reach a certain finite time increase more and more, which might cause an accumulation of error and large deviations between the numerical and exact solutions.

Addressing this matter is generally difficult. For example, for the diffusion equation, the number of time steps necessary to reach a certain time t becomes larger and larger as Δt becomes smaller, and we should prove that the numerical error affecting each single step does not accumulate. For an elliptic problem, we should prove that the solution of an algebraic linear system the matrix of which becomes larger and larger as the discretization becomes finer and finer converges to the exact solution. The great usefulness of the Lax equivalence theorem is that it permits to bypass the difficulty associated with attempting to face the convergence question head on. The theorem can be stated as follows:

> LAX EQUIVALENCE THEOREM: Given a properly posed linear initial-value problem and a *consistent* finite-difference approximation to it, *stability* is the necessary and sufficient condition for convergence.

Properly posed means that the original statement of the problem (i.e., before discretization) is mathematically sound.[27] A *consistent discretization* is a discretization such that the difference between the discretized equation and the original differential equation tends to 0 when the discretization becomes finer and finer. This is usually relatively easy to check by carrying out a Taylor series expansion of the discretized formula as we have done before.[28] In the present case (6.3.4) shows directly that, as Δt and Δx are reduced, the original differential equation is recovered. This argument shows that the scheme is consistent.

According to the Lax theorem, therefore, in order to ensure that the scheme is convergent we need to check its *stability* properties, i.e., to ensure that the solution does not grow without bound as we use more and more time steps to extend the solution to later and later times. Generally speaking, stability can be conditional or unconditional. In the former case the scheme is stable only under certain conditions, e.g. on Δt and Δx. In the latter case the scheme is always stable irrespective of how Δt and Δx are chosen. Of course, unconditional stability does not necessarily imply accuracy – it only means that errors, whatever their magnitude, do not accumulate.

6.5 The von Neumann stability method

There is a very nice method to study the stability of linear schemes.[29] The foundation of the method lies in the theory of the Fourier series but, without going into detail, we just show how it is applied.

We look for solutions of (6.3.2) having a special form suggested by the elementary solutions of section 6.2. The algebra is much simpler if we use complex exponentials in place of sines and cosines, so that we look for solutions of the form

$$u(x_j, t) \;=\; G(t) \exp\left(iM\pi \frac{x_j}{L}\right) \;=\; G(t) \exp\left(iM\pi \frac{j\Delta x}{L}\right) . \tag{6.5.1}$$

[27] For example, the original mathematical problem has a unique solution that doesn't "jump around" when the data (i.e., the functions g and h, or the diffusivity D) are slightly changed; "initial-value problem" means that data are only assigned at the initial time.

[28] When there is more than one independent variable, the difference between the discretized formula and the original analytic formula must go to zero when the increments of each variable go to zero independently of each other. For example, if the difference has an expression like $(\ldots)\Delta t + (\ldots)\Delta x$, it will go to zero whatever the way in which Δt and Δx are made small. But if the difference has the form $(\ldots)(\Delta t/\Delta x)$, it would tend to zero only if $\Delta t \to 0$ faster than Δx. In this case the discretization would be inconsistent.

[29] An extension to non-linear schemes, possibly using some *ad hoc* tricks, is also sometimes possible.

Here $i = \sqrt{-1}$; by Euler's formula, $e^{i\alpha} = \cos\alpha + i\sin\alpha$. We define the (dimensionless) wavenumber k by (see footnote 25)

$$k_M \;=\; \frac{M\pi\Delta x}{L}\,, \qquad M \;=\; 0,\, \pm 1,\, \pm 2\,, \ldots\,, \tag{6.5.2}$$

and write

$$u(x_j, t) \;=\; G(t) e^{ik_M j}\,. \tag{6.5.3}$$

Note that k_M can be positive or negative and can range from zero all the way to infinity. Small values of M correspond to long-wavelength perturbations and large values to short-wavelength perturbations. The former oscillate slowly in space, the latter rapidly.

Upon substituting (6.5.3) into (6.3.2) and cancelling some common terms we find

$$G(t+\Delta t) \;=\; G(t)\left[1 + \lambda\left(e^{-ik_M} - 2 + e^{ik_M}\right)\right] \;=\; G(t)\left[1 - 2\lambda\left(1 - \cos k_M\right)\right] \;=\; G(t)\left[1 - 4\lambda\sin^2\frac{k_M}{2}\right]. \tag{6.5.4}$$

Thus

$$G(t+\Delta t) \;=\; G(t)\left[\ldots\right] \tag{6.5.5}$$

$$G(t+2\Delta t) \;=\; G(t+\Delta t)\left[\ldots\right] \;=\; G(t)\left[\ldots\right]^2\,, \tag{6.5.6}$$

and so on. It is therefore evident that $G(t)$ will become larger and larger if the *amplification factor*

$$V \;=\; \frac{G(t+\Delta t)}{G(t)} \;=\; 1 - 4\lambda\sin^2\frac{k}{2}\,, \tag{6.5.7}$$

is greater than 1 in modulus or, equivalently, if $|V| > 1$. If, on the other hand, $|V| \le 1$, $G(t)$ will remain bounded. So the stability condition is

$$\left|1 - 4\lambda\sin^2\frac{k}{2}\right| \;\le\; 1\,. \tag{6.5.8}$$

If the quantity in the modulus is positive, the inequality is satisfied. So what remains to be checked is whether the inequality is satisfied when it is negative, i.e. whether $4\lambda\sin^2\frac{k_M}{2} - 1 \le 1$ or

$$\lambda \;\le\; \frac{1}{2\sin^2\frac{k_M}{2}}\,. \tag{6.5.9}$$

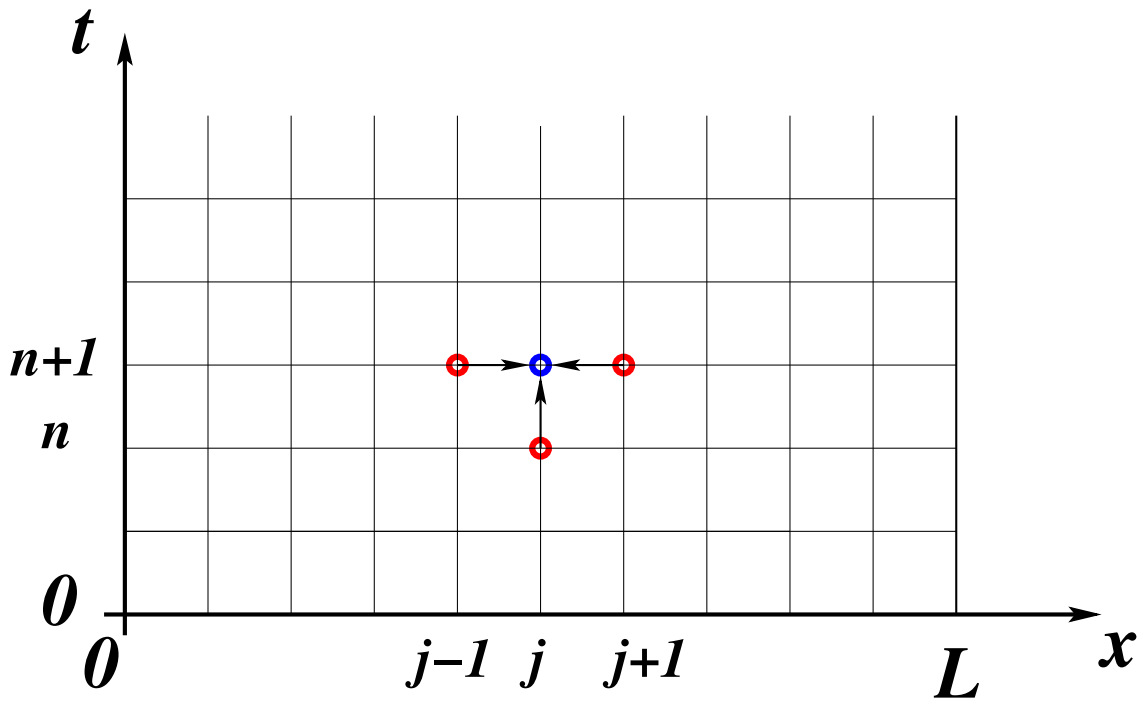
The minimum value of the left-hand side is $1/2$, which is attained for $k = \pi,\, 3\pi/2, \ldots$. If this relation needs to be satisfied for all values of k, so that no perturbation can get amplifeid as time passes, it is evidently necessary that $\lambda \le 1/2$. Thus we conclude that the explicit method is *conditionally stable*: the condition for stability is[30]

$$\lambda \;=\; \frac{D\Delta t}{\Delta x^2} \;\le\; \frac{1}{2}\,. \tag{6.5.10}$$

We can interpret this relation physically by noting that, from the results of section 6.2, we know that the time needed to diffuse the information contained in u over a distance Δx is of the order of $\Delta x^2/D$. This stability limit, therefore, states that we should not take Δt larger than the time it takes for u to diffuse over a distance Δx. Attempting to do so would be like attempting to predict u a Δx away without waiting long enough that for the information to have time to propagate over that distance.

Let L be the extent of the spatial domain over which we want to solve the problem. A consequence of the analysis of section 6.2 is that, for most problems of interest, diffusion phenomena over a scale L require for their manifestation a time at least of the order of L^2/D, i.e., $L^2/(D\Delta t)$ time steps, and often more. Even using the maximum stable time step with $\lambda = \frac{1}{2}$, this requires about $(L/\Delta x)^2$ time steps. Since, for spatial

[30] Strictly speaking, the von Neumann method is only justified at interior nodes; boundary nodes may require a special treatment.



%endpicture

Figure 6.6.1: Schematic representation of the implicit method for the solution of the diffusion equation.

accuracy, we normally need to use a small Δx, it is seen that the number of time steps can become very, even prohibitively, large if we want to have a stable scheme. Thus, for practical applications, the stability condition (6.5.10) is very bad news.

As a concluding comment it may be noted that, in other, more general cases, the quantity $G(t+\Delta t)/G(t)$ could well be complex. The stability condition is still $|G(t+\Delta t)/G(t)| \leq 1$, but the modulus would then have to be interpreted in the sense of complex numbers.

6.6 Implicit discretization

Another possible way to discretize the diffusion equation is the implicit discretization (figure 6.6.1)

$$\frac{u_j^{n+1} - u_j^n}{\Delta t} = D\frac{u_{j-1}^{n+1} - 2u_j^{n+1} + u_{j+1}^{n+1}}{\Delta x^2}\,. \tag{6.6.1}$$

This is labelled implicit as it is not possible to obtain u_j^{n+1} independently from the values of u at all the other points, unlike (6.3.2). As a matter of fact, in this case we must solve a tri-diagonal system:

$$-\lambda u_{j-1}^{n+1} + (1+2\lambda)u_j^{n+1} - \lambda u_{j+1}^{n+1} = u_j^n\,. \tag{6.6.2}$$

An analysis similar to that leading to (6.3.7) leads to

$$\frac{\partial u}{\partial t} = D\frac{\partial^2 u}{\partial x^2} + \frac{D^2\Delta x^2}{2}\left(\frac{1}{6} + \lambda\right)\frac{\partial^4 u}{\partial x^4}\,. \tag{6.6.3}$$

Thus, we see that the error of the implicit discretization is the same as that of the explicit one.[31] However, no special value of λ can increase the order of accuracy of this formula.

The stability of the implicit method can be ascertained in the same way as shown before. In this case, in place of (6.5.4) we find

$$G(t+\Delta t) \;=\; G(t) - G(t+\Delta t)\left(4\sin^2 k/2\right)\,, \tag{6.6.4}$$

from which

$$V \;=\; \frac{G(t+\Delta t)}{G(t)} \;=\; \frac{1}{1+4\lambda\sin^2 k/2}\,. \tag{6.6.5}$$

We thus conclude that $|V|<1$ always, which makes the implicit method *unconditionally stable*.

While this is of course a welcome feature, it is obvious that one cannot use an arbitrarily large time step because the discretization is only first-order accurate in Δt. For example, if we are interested in resolving a certain spatial scale, we need a Δx small enough compared with that scale. If we take a Δt that is too large, λ will then be large and if we divide (6.6.2) by λ we find

$$-u_{j-1}^{n+1} + \left(2+\frac{1}{\lambda}\right)u_j^{n+1} - u_{j+1}^{n+1} \;=\; \frac{u_j^n}{\lambda}\,. \tag{6.6.6}$$

If λ is very large this equation becomes, approximately,

$$-u_{j-1}^{n+1} + 2u_j^{n+1} - u_{j+1}^{n+1} \;\simeq\; 0\,, \tag{6.6.7}$$

i.e., approximately, $\partial^2 u/\partial x^2 \simeq 0$, which is the form to which the diffusion equation reduces for times large enough that the system has reached a steady condition. This argument shows that, by taking Δt too large, we would only calculate the large-time state of the spatial scales we are interested in and be unable to follow their evolution in time.

6.7 Three-level time discretization

Both the explicit and the implicit discretization are affected by an error of order Δt in time which it would be good to improve so that longer time steps would be allowed without compromising accuracy. Since the $O(\Delta t)$ time error arises from the way the time derivative is approximated, a natural way to attempt to do this is to use a more accurate approximation for $\partial u/\partial t$. We might for example try the so-called Richardson discretization

$$\frac{u_j^{n+1}-u_j^{n-1}}{2\Delta t} \;=\; D\frac{u_{j-1}^n - 2u_j^n + u_{j+1}^n}{\Delta x^2}\,, \tag{6.7.1}$$

which can indeed be shown to have a formal error of order Δt^2 and Δx^2. The problem with this formula is that it is *unconditionally unstable*! In other words, no matter how one chooses the time and space steps, the solution will grow without bound. This is an interesting counter-example to Lax's theorem. If the right-hand side is evaluated at the advanced time t^{n+1}, on the other hand, the resulting formula is unconditionally stable.

Richardson's is a three-level method because it involves the unknown function at three time levels, t^{n-1}, t^n and t^{n+1}. There are other three-level methods which achieve $O(\Delta t^2)$ and are stable. One of the most used ones is the following:

$$\frac{3u_j^{n+1}-4u_j^n+u_j^{n-1}}{2\Delta t} \;=\; D\frac{u_{j-1}^{n+1} - 2u_j^{n+1} + u_{j+1}^{n+1}}{\Delta x^2}\,. \tag{6.7.2}$$

[31] The relation (6.3.7) shows that the error is of order Δx^2 but, since λ must be a finite number in order to progress in time, it follows that $\Delta t \sim \lambda \Delta x^2$ so that an error of order Δx^2 in space also entails an error of order Δt in time. This can be proven directly by deriving the analog of (6.3.3) for this case.

To study the consistency of this method it is easier to rewrite it identically as

$$\frac{3(u_j^{n+1}-u_j^n)-(u_j^n-u_j^{n-1})}{2\Delta t} \;=\; D\frac{u_{j-1}^{n+1}-2u_j^{n+1}+u_{j+1}^{n+1}}{\Delta x^2}\,. \tag{6.7.3}$$

By proceeding with a Taylor series expansion as usual we have

$$u_j^{n+1}-u_j^n \;=\; \Delta t\partial_t u+\frac{\Delta t^2}{2}\partial_t^2 u+\dots\,, \tag{6.7.4}$$

$$u_j^n-u_j^{n-1} \;=\; u_j^n-\left[u_j^n-\Delta t\partial_t u+\frac{\Delta t^2}{2}+\dots\right] \;=\; \Delta t\partial_t u-\frac{\Delta t^2}{2}\partial_t^2 u+\dots\,, \tag{6.7.5}$$

so that

$$\begin{aligned}\frac{3(u_j^{n+1}-u_j^n)-(u_j^n-u_j^{n-1})}{2\Delta t} &= \frac{1}{2\Delta t}\left[3\Delta t\partial_t u+\frac{3\Delta t^2}{2}-\Delta t\partial_t u+\frac{\Delta t^2}{2}\partial_t^2 u\right]+\dots\\ &= \partial_t u+\Delta t\partial_t^2 u+\dots \;=\; \partial_t u+D\Delta t\partial_t\partial_x^2 u+\dots\,,\end{aligned} \tag{6.7.6}$$

where we have used the fact that $\partial_t u = D\partial_x^2 u$ in the last step and all the terms are evaluated at the time level t^n. Repeating the calculation that leads to the effective equation (6.3.3) we find

$$D\frac{u_{j-1}^{n+1}-2u_j^{n+1}+u_{j+1}^{n+1}}{\Delta x^2} \;=\; D[\partial_x^2 u]^{n+1}+\dots \;=\; D[\partial_x^2 u]^n+D\Delta t\partial_t[\partial_x^2 u]^n+\dots\,. \tag{6.7.7}$$

We thus see that, when (6.7.6) and (6.7.7) are substituted into (6.7.3), the $O(\Delta t)$ terms cancel, showing that the method is of second-order accuracy in time.

The von Neumman stability analysis proves the method to be unconditionally stable. Furthermore, it is found that it damps out the spurious oscillations which arise in the case of stiff problems, i.e., of problems with two (or more) different time scales, one much shorter than the other. It may, however, produce oscillatory solutions when Δt is too large. Generally speaking, three-time-level methods have two shortcomings which it is worth while to point out out. The first one is the need to retain information at two time levels, which increases storage needs and may increase latency even when sufficient memory is available to accommodate the additional memory requirements. Secondly, these methods cannot evidently be applied for the first time step as the algorithm necessary to generate the solution at $t = \Delta t$ requires information at $t = 0$, which is available from the initial conditions, but also at an earlier time $t = -\Delta t$ which is not available. Thus, for the first time step, some other method, or the use of ghost nodes in time, would be required. In spite of these shortcomings these methods have features that make them suitable for specific cases, such as the one of stiff problems just mentioned.

6.8 The Crank-Nicolson scheme

For the reasons explained at the end of the previous section, it is usually preferable to decrease the time error by having recourse to other methods, the most widely used of which is the *Crank-Nicolson scheme*, which is similar to the use of the trapezoidal rule to approximate an integral; the scheme is

$$\frac{u_j^{n+1}-u_j^n}{\Delta t} \;=\; \frac{D}{2}\left(\frac{u_{j-1}^n-2u_j^n+u_{j+1}^n}{\Delta x^2}+\frac{u_{j-1}^{n+1}-2u_j^{n+1}+u_{j+1}^{n+1}}{\Delta x^2}\right)\,. \tag{6.8.1}$$

Advancement of the solution from t^n to t^{n+1} requires the solution of the tri-diagonal system

$$-\frac{\lambda}{2}u_{j-1}^{n+1}+(1+\lambda)u_j^{n+1}-\frac{\lambda}{2}u_{j+1}^{n+1} \;=\; \frac{\lambda}{2}u_{j-1}^n+(1-\lambda)u_j^n+\frac{\lambda}{2}u_{j+1}^n\,. \tag{6.8.2}$$

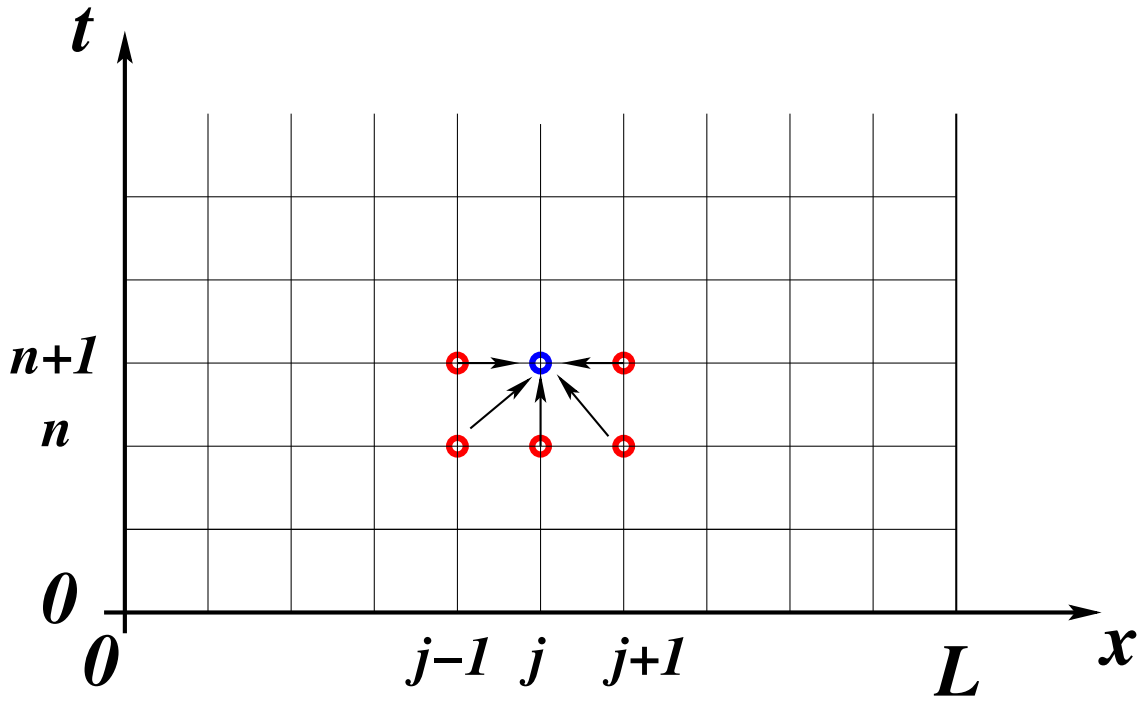


Figure 6.8.1: Schematic representation of the Crank-Nicolson method for the solution of the diffusion equation.

The necessary amount of work is similar to that for the implicit method, but this method is preferred due to its smaller truncation error. Indeed it can be checked by carrying out the usual Taylor series expansions that now the error is of order Δt^2 and Δx^2. The same conclusion can be reached more quickly in the following way. We expand the advanced-time solution to find

$$u_j^{n+1} \;=\; u(x_j, t^n + \Delta t) \;=\; u(x_j, t^n) + \Delta t[\partial_t u]^n + \frac{\Delta t^2}{2}[\partial_t^2 u]^n + O(\Delta t^3)\,, \tag{6.8.3}$$

where we have explicitly indicated the time level of the derivatives for clarity. By solving this relation for $[\partial_t u]^n$ we have

$$\begin{aligned}[\partial_t u]^n \;&=\; \frac{u_j^{n+1} - u_j^n}{\Delta t} - \frac{\Delta t}{2}[\partial_t^2 u]^n + O(\Delta t^2) \\ &=\; \frac{u_j^{n+1} - u_j^n}{\Delta t} - \frac{\Delta t}{2}[\partial_t (D\partial_x^2 u)]^n + O(\Delta t^2)\,.\end{aligned} \tag{6.8.4}$$

In the last step we have written $\partial_t^2 u \;=\; \partial_t(\partial_t u)$ and we have used the diffusion equation to replace the inner time derivative. We also have

$$[\partial_x^2 u]^{n+1} \;=\; [\partial_x^2 u]^n + \Delta t[\partial_t(\partial_x^2 u)]^n + O(\Delta t^2)\,. \tag{6.8.5}$$

The terms of order Δt can be eliminated between these two equations to find

$$[\partial_t u]^n \;=\; \frac{u_j^{n+1} - u_j^n}{\Delta t} - \frac{D}{2}\left([\partial_x^2 u]^{n+1} - [\partial_x^2 u]^n\right)\,, \tag{6.8.6}$$

or, using the diffusion equation in the left-hand side,

$$D[\partial_x^2 u]^n \;=\; \frac{u_j^{n+1} - u_j^n}{\Delta t} - \frac{D}{2}\left([\partial_x^2 u]^{n+1} - [\partial_x^2 u]^n\right)\,. \tag{6.8.7}$$

Upon rearranging we conclude that

$$\frac{u_j^{n+1} - u_j^n}{\Delta t} \;=\; \frac{D}{2}\left([\partial_x^2 u]^{n+1} + [\partial_x^2 u]^n\right)\,, \tag{6.8.8}$$

from which (6.8.1) follows by writing down explicitly the discretized form of the spatial derivatives. The final order of the error in time is therefore the error of the terms indicated by $O(\Delta t^2)$ in the previous relations.

The von Neumann stability analysis leads to

$$V \;=\; \frac{G(t+\Delta t)}{G(t)} \;=\; \frac{1 - 2\lambda \sin^2 k/2}{1 + 2\lambda \sin^2 k/2}\,, \tag{6.8.9}$$

from which it is seen that the Crank-Nicolson scheme is unconditionally stable. A more detailed consideration including the boundary conditions supports this conclusion for Dirichlet boundary conditions.

6.9 The multi-dimensional diffusion equation

In two space dimensions the diffusion equation is

$$\frac{\partial u}{\partial t} \;=\; D\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right)\,. \tag{6.9.1}$$

We now have two space variables and therefore we need to add a subscript; we write $u_{jk}^n = u(x_j, y_k, t^n)$.

In order to save writing let us introduce the following definitions:

$$\nabla_x^2 u_{jk} \;=\; \frac{u_{j-1,k} - 2u_{j,k} + u_{j+1,k}}{\Delta x^2}\,, \qquad \nabla_y^2 u_{jk} \;=\; \frac{u_{j,k-1} - 2u_{j,k} + u_{j,k+1}}{\Delta y^2}\,. \tag{6.9.2}$$

Thus, $\nabla_x^2 u_{jk}$ and $\nabla_y^2 u_{jk}$ stand for the standard second-order approximations to the second derivatives in the x and y directions, respectively. Note that, in general, Δx and Δy can be different.

With this notation, the explicit discretization of (6.9.1) reads

$$\frac{u_{jk}^{n+1} - u_{jk}^n}{\Delta t} \;=\; D\left(\nabla_x^2 u_{jk}^n + \nabla_y^2 u_{jk}^n\right)\,. \tag{6.9.3}$$

The order of the error is Δt in time and Δx^2 and Δy^2 in space. The von Neumann criterion now shows that, for stability,

$$\frac{D\Delta t}{\Delta x^2} + \frac{D\Delta t}{\Delta y^2} \le \frac{1}{2}\,. \tag{6.9.4}$$

This is more stringent than in the one-dimensional case; for example, if $\Delta x = \Delta y$, it would give $D\Delta t/\Delta x^2 \le \frac{1}{4}$ rather than $\frac{1}{2}$ as in the one-dimensional case.

The implicit discretization is

$$\frac{u_{jk}^{n+1} - u_{jk}^n}{\Delta t} \;=\; D\left(\nabla_x^2 u_{jk}^{n+1} + \nabla_y^2 u_{jk}^{n+1}\right)\,. \tag{6.9.5}$$

Now the node (j,k) is connected to the four surrounding nodes $j \pm 1, k$ and $j, k \pm 1$, so that the linear system corresponding to (6.9.5) has a matrix with 5 diagonals rather than 3. The solution procedure is harder, but the method is unconditionally stable, although with an error of order Δt in time and Δx^2 and Δy^2 in space as the explicit method.

The Crank-Nicolson discretization is

$$\frac{u_{jk}^{n+1} - u_{jk}^n}{\Delta t} \;=\; \frac{D}{2}\left(\nabla_x^2 u_{jk}^{n+1} + \nabla_y^2 u_{jk}^{n+1} + \nabla_x^2 u_{jk}^n + \nabla_y^2 u_{jk}^n\right)\,. \tag{6.9.6}$$

This is unconditionally stable for Dirichlet boundary conditions. As the implicit scheme, it gives rise to a penta-diagonal matrix, although it has a higher accuracy with an error of order Δt^2 in time (and, as before, Δx^2, Δy^2 in space).

Analogous considerations apply for the three-dimensional case; in particular, for both the implicit and Crank-Nicolson methods, the matrix will have seven non-zero diagonals.

6.9.1 The ADI scheme

The Alternating Direction Implicit scheme (Peaceman & Rachford) is an elegant way to replace the problem of dealing with a penta-diagonal matrix with two steps, each one involving a tri-diagonal matrix. The idea is to take two half-steps, of length $\frac{1}{2}\Delta t$ each, as follows:

$$\frac{u_{jk}^{n+1/2} - u_{jk}^{n}}{\Delta t/2} \;=\; D\left(\nabla_x^2 u_{jk}^{n+1/2} + \nabla_y^2 u_{jk}^{n}\right), \tag{6.9.1.1}$$

$$\frac{u_{jk}^{n+1} - u_{jk}^{n+1/2}}{\Delta t/2} \;=\; D\left(\nabla_x^2 u_{jk}^{n+1/2} + \nabla_y^2 u_{jk}^{n+1}\right), \tag{6.9.1.2}$$

The first step is explicit in y and implicit in x while, conversely, the second step is explicit in x and implicit in y. These features justify the denomination "alternating direction." Here we use a superscript $n+1/2$ not to indicate the correct value at $t^{n+1/2}$ but simply to note the fact that (6.9.1.1) is an intermediate value for u_{jk}^{n+1}. The method is second-order accurate in time and space and unconditionally stable. This statement can be verified by calculating the amplification factor for each substep; the amplification factor of the entie scheme is the product of the two.

One can prove consistency of the method by expanding everything in Taylor series as usual, but there is a neater way to justify it. We re-write the two equations (6.9.1.1), (6.9.1.2) as

$$\left(1 - \frac{D\Delta t}{2}\nabla_x^2\right) u_{jk}^{n+1/2} \;=\; \left(1 + \frac{D\Delta t}{2}\nabla_y^2\right) u_{jk}^{n}\,. \tag{6.9.1.3}$$

$$\left(1 - \frac{D\Delta t}{2}\nabla_y^2\right) u_{jk}^{n+1} \;=\; \left(1 + \frac{D\Delta t}{2}\nabla_x^2\right) u_{jk}^{n+1/2}\,. \tag{6.9.1.4}$$

Going back to the definition (6.9.2) of $\nabla_x^2 u_{jk}$, we see that this is a linear algebraic system the matrix of which we can write in a short-hand form as $(1 - \frac{1}{2}D\Delta t\nabla_x^2)$. The solution of the system can be written as

$$u_{jk}^{n+1/2} \;=\; \left(1 - \frac{D\Delta t}{2}\nabla_x^2\right)^{-1}\left(1 + \frac{D\Delta t}{2}\nabla_y^2\right) u_{jk}^{n}\,, \tag{6.9.1.5}$$

where the first factor in the right-hand side denotes the inverse of the operator $(1 - \frac{1}{2}D\Delta t\nabla_x^2)$. Upon substituting into (6.9.1.4) we have

$$\begin{aligned}\left(1 - \frac{D\Delta t}{2}\nabla_y^2\right) u_{jk}^{n+1} \;&=\; \left(1 + \frac{D\Delta t}{2}\nabla_x^2\right)\left(1 - \frac{D\Delta t}{2}\nabla_x^2\right)^{-1}\left(1 + \frac{D\Delta t}{2}\nabla_y^2\right) u_{jk}^{n} \\ &=\; \left(1 - \frac{D\Delta t}{2}\nabla_x^2\right)^{-1}\left(1 + \frac{D\Delta t}{2}\nabla_x^2\right)\left(1 + \frac{D\Delta t}{2}\nabla_y^2\right) u_{jk}^{n}\,. \end{aligned} \tag{6.9.1.6}$$

The switching of the order of the first two operators in the right-hand side is justified by the fact that they both contain the same operator ∇_x^2. Now we apply the operator $(1 - \frac{1}{2}D\Delta\nabla_x^2)$ to both sides to find

$$\left(1 - \frac{D\Delta t}{2}\nabla_x^2\right)\left(1 - \frac{D\Delta t}{2}\nabla_y^2\right) u_{jk}^{n+1} \;=\; \left(1 + \frac{D\Delta t}{2}\nabla_x^2\right)\left(1 + \frac{D\Delta t}{2}\nabla_y^2\right) u_{jk}^{n}\,. \tag{6.9.1.7}$$

But

$$\left(1-\frac{D\Delta t}{2}\nabla_x^2\right)\left(1-\frac{D\Delta t}{2}\nabla_y^2\right)u_{jk}^{n+1} \;=\; \left(1-\frac{D\Delta t}{2}\nabla_x^2-\frac{D\Delta t}{2}\nabla_y^2\right)u_{jk}^{n+1}+O(\Delta t^2)\,, \tag{6.9.1.8}$$

and similarly for the terms in the right-hand side of (6.9.1.7) so that this equation becomes

$$\left(1-\frac{D\Delta t}{2}\nabla_x^2-\frac{D\Delta t}{2}\nabla_y^2\right)u_{jk}^{n+1} \;=\; \left(1+\frac{D\Delta t}{2}\nabla_x^2+\frac{D\Delta t}{2}\nabla_y^2\right)u_{jk}^{n}+O(\Delta t^2)\,, \tag{6.9.1.9}$$

which is, evidently, the two-dimensional Crank-Nicolson discretization (6.9.6). Since the discretization error affecting the Crank-Nicolson discretization is also Δt^2, this "operator splitting" procedure does not decrease the order of accuracy of the overall scheme.

In three dimensions the method uses 2 intermediate sub-steps of length $\Delta t/3$, but is less stable requiring $D\Delta t/\Delta x^2$, $D\Delta t/\Delta y^2$ and $D\Delta t/\Delta z^2$ to be all less than, or equal to, $\frac{3}{2}$.

It is interesting to show how the modified alternating direction method explained for elliptic problems is rooted in the ADI method. The starting point is the observation that the solution of the elliptic problem

$$\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2} \;=\; f(x,y)\,, \tag{6.9.1.10}$$

with suitable boundary conditions, can be considered the long-time limit of the time-dependent diffusion equation (with $D=1$)

$$\frac{\partial u}{\partial t} \;=\; \frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}-f(x,y)\,, \tag{6.9.1.11}$$

with the same boundary conditions. For example, one may think of u as the temperature of a system forced by a source f with prescribed temperature values on the boundaries. If the system is started in any condition, as time passes it will reach a steady state and the solution of (6.9.1.11) will tend to that of (6.9.1.10). Since we are really intersted in the solution of (6.9.1.10), the variable t appearing in (6.9.1.11) is not a real physical time, but only a pseudo-time introduce to develop a numerical method for the solution of (6.9.1.10). To aid our intuition, however, there is no harm in thinking of it as a real time.

If we apply the ADI scheme (6.9.1.1), (6.9.1.2) to (6.9.1.11) we have

$$\frac{u_{jk}^{n+1/2}-u_{jk}^{n}}{\Delta t/2} \;=\; \nabla_x^2 u_{jk}^{n+1/2}+\nabla_y^2 u_{jk}^{n}-f_{jk}\,, \tag{6.9.1.12}$$

$$\frac{u_{jk}^{n+1}-u_{jk}^{n+1/2}}{\Delta t/2} \;=\; \nabla_x^2 u_{jk}^{n+1/2}+\nabla_y^2 u_{jk}^{n+1}-f_{jk}\,, \tag{6.9.1.13}$$

which can be rearranged in the form

$$\left(\nabla_x^2-\rho\right)u_{jk}^{n+1/2} \;=\; f_{jk}-\left(\nabla_y^2+\rho\right)u_{jk}^{n}\,, \tag{6.9.1.14}$$

$$\left(\nabla_y^2-\rho\right)u_{jk}^{n+1} \;=\; f_{jk}-\left(\nabla_x^2+\rho\right)u_{jk}^{n+1/2}\,, \tag{6.9.1.15}$$

in which $\rho = 2/\Delta t$. These are precisely the equations for the modified alternating direction method for elliptic problems described earlier. Since our interest is not in the time evolution of u, but only in the limit of u as the pseudo-time t tends to infinity, the only criterion for the choice of Δt is that the procedure be stable, not necessarily that Δt be such as to produce a faithful approximation to the time evolutio of u.

Chapter 7

The Wave Equation

7.1 The wave equation

The standard one-dimensional linear wave equation has the form

$$\frac{\partial^2 v}{\partial t^2} - c^2 \frac{\partial^2 v}{\partial x^2} \;=\; 0\,. \tag{7.1.1}$$

One can think of v, for example, as representing the displacement of an infinitesimal piece of a vibrating string from the equilibrium position. This mechanical interpretation makes it evident that suitable initial conditions will concern the initial position and the initial velocity of each infinitesimal piece of the string so that we must associate to (7.1.1) initial conditions of the form

$$v(x,0) \;=\; f(x)\,, \qquad \left.\frac{\partial v}{\partial t}\right|_{t=0} \;=\; g(x)\,, \tag{7.1.2}$$

with f and g given functions. There will also be problem-dependent boundary conditions, e.g.

$$v(x=0,t) \;=\; F_0(t)\,, \qquad v(x=L,t) \;=\; F_L(t)\,, \tag{7.1.3}$$

or similar conditions of the Neumann type, i.e., concerning the x-derivatives of v at the-end points rather than the function v itself.

We note that

$$\begin{aligned}\left(\frac{\partial}{\partial t} + c\frac{\partial}{\partial x}\right)\left(\frac{\partial v}{\partial t} - c\frac{\partial v}{\partial x}\right) \;&=\; \frac{\partial}{\partial t}\frac{\partial v}{\partial t} + \frac{\partial}{\partial t}\left(-c\frac{\partial v}{\partial x}\right) + c\frac{\partial}{\partial x}\frac{\partial v}{\partial t} + c\frac{\partial}{\partial x}\left(-c\frac{\partial v}{\partial x}\right)\\ &=\; \frac{\partial^2 v}{\partial t^2} - c^2\frac{\partial^2 v}{\partial x^2}\,.\end{aligned} \tag{7.1.4}$$

Thus, upon setting

$$\frac{\partial v}{\partial t} - c\frac{\partial v}{\partial x} \;=\; u\,, \tag{7.1.5}$$

we may write[32]

$$\frac{\partial u}{\partial t} + c\frac{\partial u}{\partial x} \;=\; 0\,. \tag{7.1.6}$$

The initial condition for v is given by the first one of (7.1.2). We obtain the initial condition for u from (7.1.5) evaluated at $t = 0$:

$$u(x,0) \;=\; \left.\frac{\partial v}{\partial t}\right|_{t=0} - c\frac{\partial}{\partial x}v(x,t=0) \;=\; g(x) - c\frac{df}{dx}\,. \tag{7.1.7}$$

In this way the original second-order wave equation has been written as an equivalent pair of first-order equations, (7.1.5) and (7.1.6). Since these are simpler, we will start by considering the numerical solution of these first-order equations. Before turning to numerics, however, it is useful to consider some analytical aspects of the problem.

[32] Equally well, of course, we could set

$$\frac{\partial v}{\partial t} + c\frac{\partial v}{\partial x} \;=\; w\,, \qquad \frac{\partial w}{\partial t} - c\frac{\partial w}{\partial x} \;=\; 0\,.$$

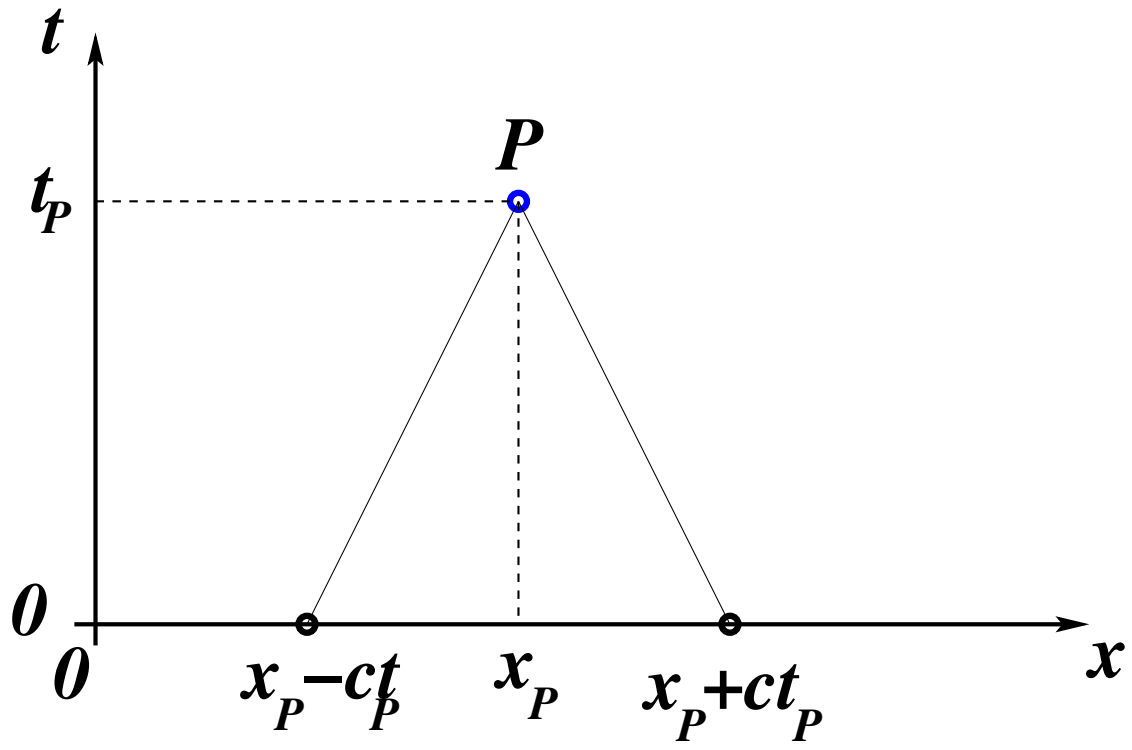


Figure 7.2.1: The wave equation describes the propagation of the information provided at the initial time $t = 0$ along the lines $x \pm ct = \text{const}$.

7.2 Some analytical solutions of the wave equations

The wave equation has a simple, exact and elegant general solution found by D'Alembert; it is

$$v(x,t) \;=\; F(x-ct) + G(x+ct)\,, \tag{7.2.1}$$

with the functions F and G depending on the initial conditions, as we show later. To prove that (7.2.1) is indeed a solution we note that $F(x-ct)$ is really a function of a *single* variable, the "package" $z(x,t) = x-ct$, $F = F(\,z(x,t))$. Thus, using the differentiation rule for a function of a function, we have

$$\frac{\partial F}{\partial x} \;=\; \frac{\partial z}{\partial x}\frac{dF}{dz} \;=\; \frac{dF}{dz}\,, \qquad \frac{\partial F}{\partial t} \;=\; \frac{\partial z}{\partial t}\frac{dF}{dz} \;=\; -c\frac{dF}{dz}\,, \tag{7.2.2}$$

in which dF/dz is the derivative of F with respect to the single argument on which it depends, i.e., the entire "package" $x - ct$. For example,

$$\frac{\partial}{\partial x}\sin(x-ct) \;=\; \cos(x-ct) \qquad \frac{\partial}{\partial t}\sin(x-ct) \;=\; -c\cos(x-ct)\,. \tag{7.2.3}$$

In the same way we find

$$\frac{\partial^2 F}{\partial x^2} \;=\; \frac{\partial z}{\partial x}\frac{d}{dz}\left(\frac{dF}{dz}\right) \;=\; \frac{d^2F}{dz^2}\,, \qquad \frac{\partial^2 F}{\partial t^2} \;=\; \frac{\partial}{\partial t}\left(\frac{\partial F}{\partial t}\right) \;=\; \frac{\partial}{\partial t}\left[-c\frac{dF}{dz}\right] \;=\; \frac{\partial z}{\partial t}\frac{d}{dz}\left[-c\frac{dF}{dz}\right] \;=\; c^2\frac{d^2F}{dz^2}\,, \tag{7.2.4}$$

so that, indeed, F satisfies (7.1.2). A similar calculation proves that G is also a solution. The proof that (7.2.1) is the most general solution is more difficult and we will not attempt to present it.

Before connecting F and G to the initial conditions, it is interesting to go a little deeper into the meaning of the special dependence of F and G on x and t, namely according to $F(x-ct)$ and $G(x+ct)$. Let us start with the function F. It is clear from (7.2.2) that, in addition to being a solution of the second-order wave equation, $F(x-ct)$ is also a solution of the first-order equation (7.1.6):

$$\frac{\partial F}{\partial t} + c\frac{\partial F}{\partial x} = 0\,. \tag{7.2.5}$$

We can think of an initial condition $F(x)$ which, at a later time, has become $F(x-ct)$. Consider a point x_0 where the initial conditions has the value $F(x_0)$ and, at a later time t_P, a point x_P. It is evident that, if $x_P - ct_P = x_0$, i.e. $x_P = x_0 + ct_P$, the solution F at (x_P, t_P) will have exactly the same value as at x_0 at the initial time. Thus we see that the value $F(x_0)$ has *propagated* with the velocity c to the point x_P at the later time t_P or, in other words, the "trajectory" of the initial value $F(x_0)$ is the line $x = x_0 + ct$. This argument shows that the function $F(x-ct)$ describes a perturbation propagating in the direction of increasing x with speed c (assumed positive) maintaining exactly the same shape for all times. A more formal way to prove this statement is to change the original reference frame to one moving with a velocity c. This change is effected by writing $x = x' + ct$, where x' is the spatial coordinate in the moving frame. Evidently $F(x-ct) = F(x')$, which says that, observed from the moving frame, the shape of the wave is independent of time so that it remains identical to the initial shape. A calculation similar to (7.2.2) applied to $G(x+ct)$ shows that this function solves

$$\frac{\partial G}{\partial t} - c\frac{\partial G}{\partial x} \;=\; 0\,, \tag{7.2.6}$$

which is the same as (7.1.6) and (7.2.5) except that now the speed is negative. We can apply to $G(x+ct) = G(x-(-c)t)$ an argument similar to the one used for F to show that $G(x+ct)$ represents a perturbation propagating with speed $-c$, i.e., in the direction of decreasing x (assuming $c > 0$) maintaining exactly the same shape for all times. In conclusion, (7.2.5) and its general solution $F(x-ct)$ describe waves propagating to the right with speed $c > 0$, while (7.2.6) and its general solution $G(x+ct)$ describe waves propagating to the left with speed $-c < 0$. We then find that the general solution of the second-order wave equation (7.1.1) consists of the sum of two perturbations propagating in opposite directions.

To complete these considerations we now show how F and G are related to the initial conditions (7.1.2) for v. For simplicity, let us take $g = 0$. Then we have

$$v(x, t=0) \;=\; f(x) \;=\; F(x) + G(x)\,, \qquad \left.\frac{\partial v}{\partial t}\right|_{t=0} \;=\; -cF'(x) + cG'(x) \;=\; 0\,. \tag{7.2.7}$$

From the second equation we have $G(x) = F(x) + K$ for some constant K and, substituting into the first one,

$$F(x) + F(x) + K \;=\; f(x)\,, \tag{7.2.8}$$

from which

$$F \;=\; \frac{1}{2}(f - K)\,, \qquad G \;=\; \frac{1}{2}(f-K) + K \;=\; \frac{1}{2}(f+K)\,. \tag{7.2.9}$$

Upon adding F and G to form D'Alembert solution (7.2.1) the constant K drops out (which implies that it has no physical meaning and could actually have been taken equal to zero or whatever other value at the outset) and we find

$$v(x,t) \;=\; \frac{1}{2}\left[f(x-ct) + f(x+ct)\right]\,. \tag{7.2.10}$$

Thus, the initial perturbation $f(x)$ splits into two equal waves, one, $\frac{1}{2}f(x-ct)$ propagating to the right, and the other, $\frac{1}{2}f(x+ct)$, to the left. A similar argument can be applied to the more general case in which $g \neq 0$. In this case one finds that $v(x_P, t_P)$, the solution at some space-time point (x_P, t_P), also depends on the integral of g between the two points $x_P - ct_P$ and $x_P + ct_P$ where the lines with slope $\pm 1/c$ passing

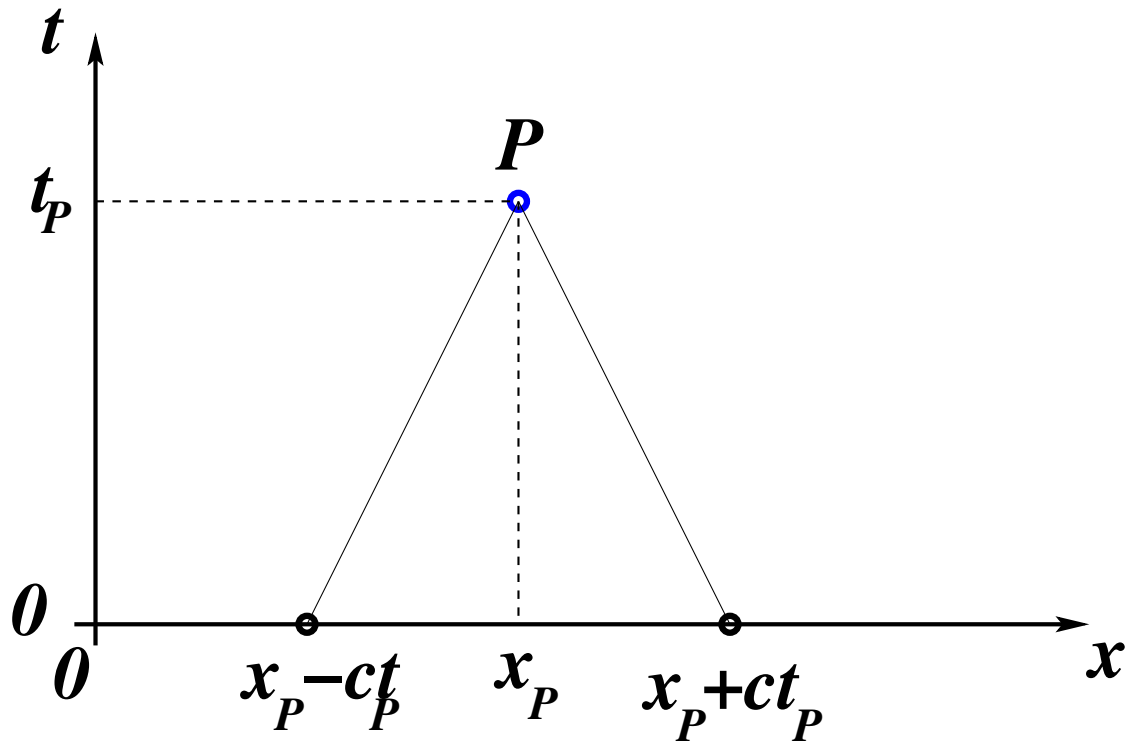


Figure 7.2.2: If there was a source of waves operating in space and time as in (7.2.11), only the portion of this source contained in the triangle would have any effect on the solution of the wave equation at the point P. The value of this source at a point x outside the triangle does not have enough time to travel to the point P given that the propagation velocity cannot exceed c.

through the space-time point (x_P, t_P) intersect the line $t = 0$ (figure 7.2.1). This interval is the *domain of dependence* of the solution at position x at time t. The reason the values of f or g outside this interval cannot have any effect on the solution at (x_P, t_P) is that, for this to happen, information would have to propagate with a velocity faster than c (i.e., in the (x, t) plane the slope of the lines should smaller in modulus than $1/c$), which is incompatible with the physics embodied in the wave equation.

If there was a mechanism that keep generating waves after the initial time, the right-hand side of the wave equation would be non-zero:

$$\frac{\partial^2 v}{\partial t^2} - c^2 \frac{\partial^2 v}{\partial x^2} \;=\; \Phi(x,t)\,. \tag{7.2.11}$$

In this case the domain of dependence of the solution at the point $(x_P,\, t_P)$ would be the entire triangle in figure 7.2.2.

If the domain of interest is limited to $0 < x < L$, as in the problem posed in the previous section, see (7.1.3), propagation cannot of course go beyond the points 0 and L: at these points the wave will be reflected and possibly modified, depending on the functions F_0 and F_L in (7.1.3). The solution (7.2.1) can be adapted to this more complex situation, but we will not dwell on these aspects as our focus here are the numerical aspects of the solution of the wave equation.

The first image that comes to mind thinking of a vibrating string is not that of propagating waves, but an oscillatory motion in which each point goes up and down oscillating with the same frequency but different amplitudes. To show how this situation is actually contained in (7.2.1) let us consider the special case

$$F(x - ct) \;=\; A_0 \cos[k(x - ct)]\,, \qquad G(x + ct) \;=\; A_0 \cos[k(x + ct)]\,. \tag{7.2.12}$$

Then

$$\begin{aligned} v(x,t) &= A_0\left(\cos[k(x-ct)]+\cos[k(x+ct)]\right) = 2A_0\cos\left(k\frac{x-ct+x+ct}{2}\right)\cos\left(k\frac{x-ct-x-ct}{2}\right) \\ &= 2A_0\,\cos(kx)\,\cos(kct)\,. \end{aligned} \tag{7.2.13}$$

What this relation shows is that a point x_1 oscillates with an angular frequency $\omega = kc$ and an amplitude $2A_0\,\cos(kx_1)$, while another point x_2 oscillates with *the same* angular frequency and a (generally different) amplitude $2A_0\,\cos(kx_2)$. A solution of this type is called a *standing wave* and, on the surface, the very notion of propagating waves seems alien to it. However, from its derivation, we clearly see that a standing wave is really the result of the superposition of counter-propagating waves as they are reflected by the end-points $x = 0$ and $x = L$.

As a final point we note a connection between the second-order wave equation and the Helmholtz equation that we have seen earlier. Suppose that we are interested in a wave phenomenon where all the waves have the same frequency ω. Then the solution of the wave equation will have the form $v(x,t) = u(x)e^{-i\omega t}$ so that, upon substituting into (7.1.1), we find

$$\frac{d^2u}{dx^2}+\frac{\omega^2}{c^2}u = 0\,. \tag{7.2.14}$$

7.3 Discretization of the first-order wave equation

As usual, we set $u_j^n = u(x_j,t^n)$. *A priori* we can imagine many different ways in which (7.1.6) can be discretized; limiting ourselves to first-order accuracy in time we may write, for example:

- Backward Euler, first-order in x, explicit in time

$$\frac{u_j^{n+1}-u_j^n}{\Delta t}+c\frac{u_j^n-u_{j-1}^n}{\Delta x} = 0\,, \tag{7.3.1}$$

- Backward Euler, first-order in x, implicit in time

$$\frac{u_j^{n+1}-u_j^n}{\Delta t}+c\frac{u_j^{n+1}-u_{j-1}^{n+1}}{\Delta x} = 0\,, \tag{7.3.2}$$

- Forward Euler, first-order in x, explicit in time

$$\frac{u_j^{n+1}-u_j^n}{\Delta t}+c\frac{u_{j+1}^n-u_j^n}{\Delta x} = 0 \tag{7.3.3}$$

- Forward Euler, first-order in x, implicit in time

$$\frac{u_j^{n+1}-u_j^n}{\Delta t}+c\frac{u_{j+1}^{n+1}-u_j^{n+1}}{\Delta x} = 0 \tag{7.3.4}$$

- Centered difference, second-order in x, explicit in time

$$\frac{u_j^{n+1}-u_j^n}{\Delta t}+c\frac{u_{j+1}^n-u_{j-1}^n}{2\Delta x} = 0\,, \tag{7.3.5}$$

- Centered difference, second-order in x, implicit in time

$$\frac{u_j^{n+1}-u_j^n}{\Delta t}+c\frac{u_{j+1}^{n+1}-u_{j-1}^{n+1}}{2\Delta x} = 0\,, \tag{7.3.6}$$

and others as well. If we wanted second-order accuracy in both space and time we might want to try

$$\frac{u_j^{n+1} - u_j^{n-1}}{2\Delta t} + c\frac{u_{j+1}^{n+1} - u_{j-1}^{n+1}}{2\Delta x} = 0\,, \tag{7.3.7}$$

which is implicit, or the explicit analog of this. Our task is now to see which of these methods is preferable in terms of accuracy and stability.

7.4 The method of effective equations

There is a very nice way to gain a deep insight into the properties of the various possible discretizations, which is called the method of effective equations. We have already had an introduction to this method in the case of the diffusion equation; we now apply it to the first-order wave equation. Let us focus on the backward Euler explicit scheme (7.3.1) which gives

$$u_j^{n+1} = u_j^n - \frac{c\Delta t}{\Delta x}(u_j^n - u_{j-1}^n)\,. \tag{7.4.1}$$

The idea of the method of effective equation is that, when we use this (or any one of the previous discretizations), what we are solving is not really (7.1.6) but an equation that differs from it because of the presence of errors that are a consequence of the discretization. By using the usual Taylor series argument, we see that

$$\frac{u_j^n - u_{j-1}^n}{\Delta x} = \frac{\partial u}{\partial x} - \frac{1}{2}\Delta x\frac{\partial^2 u}{\partial x^2} + \dots \tag{7.4.2}$$

and

$$\frac{u_j^{n+1} - u_j^n}{\Delta t} = \frac{\partial u}{\partial t} + \frac{1}{2}\Delta t\frac{\partial^2 u}{\partial t^2} + \dots \tag{7.4.3}$$

This latter result can be re-expressed noting that, from (7.1.6),

$$\frac{\partial^2 u}{\partial t^2} = \frac{\partial}{\partial t}\left(\frac{\partial u}{\partial t}\right) = \frac{\partial}{\partial t}\left(-c\frac{\partial u}{\partial x}\right) = -c\frac{\partial}{\partial x}\left(\frac{\partial u}{\partial t}\right) = c^2\frac{\partial^2 u}{\partial x^2}\,. \tag{7.4.4}$$

Upon substituting (7.4.2) and (7.4.4) into (7.3.1) and rearranging we see that (7.3.1) is not really the same as the original equation (7.1.6), but (assuming that the higher-order terms we have neglected in (7.4.2) and (7.4.3) are negligible) is closer to

$$\frac{\partial u}{\partial t} + c\frac{\partial u}{\partial x} = \frac{c\Delta x}{2}\left(1 - \frac{c\Delta t}{\Delta x}\right)\frac{\partial^2 u}{\partial x^2}\,. \tag{7.4.5}$$

Sure enough, when Δx and Δt tend to zero, the error vanishes, which proves that the discretization (7.3.1) is consistent with (7.1.6) but, for any finite Δx and Δt, what we are really solving is an equation closer to (7.4.5) than to (7.1.6). Alternatively put, we may say that what we find by numerically solving (7.3.1) according to (7.4.1) what we are really generating is something which is closer to the solution of (7.4.5) than to the solution of the original equation (7.1.6).[33] The use of the effective equation (7.4.5) to study the properties of the various possible discretizations of (7.1.6) gives this method its name.

If we set

$$D = \frac{c\Delta x}{2}\left(1 - \frac{c\Delta t}{\Delta x}\right)\,, \tag{7.4.6}$$

[33] We have derived (7.4.4)) by saying that u satisfies the original equation (7.1.6), but now we find that u satisfies something closer to (7.4.5), which seems inconsistent. However, if we were to use (7.4.5) instead, the end result would be an additional term containing the factor Δt^2. Since some terms of this order have already been dropped in deriving (7.4.3) and, therefore, all terms of the same order must be dropped for consistency. This argument resolves the apparent contradiction of the procedure, which is an example of the very commonly used method of successive approximations. This issue will play a role when we include one more term in the expansion; see section 7.5 and footnote 37.

(7.4.5) becomes

$$\frac{\partial u}{\partial t} + c\frac{\partial u}{\partial x} \;=\; D\frac{\partial^2 u}{\partial x^2}\,. \tag{7.4.7}$$

We see that (7.4.5) contains a diffusive process of purely numerical origin (in the right-hand side), in addition to the original propagation process (in the left-hand side).[34]

In order to see what are the consequences of the non-zero right-hand side of (7.4.7), we again consider a special solution (i.e., a single Fourier mode) corresponding to the initial condition

$$f(x) \;=\; Ae^{ikx}\,, \tag{7.4.8}$$

with i the imaginary unit and k a constant.[35] The exact solution of (7.1.6) subject to this initial condition is simply

$$u_{ex}(x,t) \;=\; Ae^{ik(x-ct)}\,, \tag{7.4.9}$$

as follows from the analysis of section 7.2. To find the solution of the effective equation (7.4.7) we try

$$u(x,t) \;=\; G(t)e^{ikx}\,, \tag{7.4.10}$$

with $G(0) = A$ to so that the initial condition (7.4.9) is satisfied. Upon substitution into (7.4.7) we have

$$\frac{dG}{dt} + ikcG \;=\; D(-k^2)G\,, \tag{7.4.11}$$

which is readily integrated by separating the variables

$$\frac{dG}{G} \;=\; -(ikc + Dk^2)dt\,, \tag{7.4.12}$$

to find, after imposing the initial condition $G(0) = A$,

$$G(t) \;=\; Ae^{-k(ic+kD)t}\,. \tag{7.4.13}$$

Upon substituting into (7.4.10) we find then that the solution of (7.4.7) is given by

$$u(x,t) \;=\; Ae^{ik(x-ct)}e^{-Dk^2t}\,. \tag{7.4.14}$$

The first factor is the exact solution, but it is seen that its amplitude, instead of remaining A for all time as in (7.4.9), depends on time according to the second factor. If $D > 0$, the amplitude decreases, the slower the smaller Δx (cf. (7.4.6)), but it decreases nonetheless. This is a typical manifestation of the *numerical diffusion* or *artificial dissipation* affecting the scheme. The rate of decay is greater the larger k, i.e., short wavelengths get damped faster than long ones. The situation is much worse if $D < 0$ because, in this case, the amplitude grows without bound: the numerical method that we have used is unstable and produces a physically unacceptable result. For stability we must therefore require that $D \geq 0$. If $c > 0$ (waves moving toward increasing x) the stability condition is therefore

$$C \;\equiv\; \frac{c\Delta t}{\Delta x} \leq 1\,. \tag{7.4.15}$$

[34] If we effect the change of variables $x' = x + ct$ mentioned before in section 7.2, (7.4.7) becomes the diffusion equation in the form seen before, namely

$$\frac{\partial u}{\partial t} \;=\; D\frac{\partial^2 u}{\partial x'^2}\,.$$

As seen from the moving frame, therefore, the wave shape will not remain constant but diffuse away like, for example, a spot of high temperature in heat-conducting material.

[35] k is the wave number, related to the wave length λ by $k = 2\pi/\lambda$; the relation is similar to that between the angular frequency ω and the period T, which is given by $\omega = 2\pi/T$.

The quantity C in the left-hand side is known as the *Courant number* and this relation is known as the *CFL condition* (short for Courant, Friedrichs, Lewy). The definition (7.4.6) of D shows that $D = 0$ when the Courant number is exactly equal to 1. This feature is related to the special nature of the lines $x - ct = \text{const.}$ which are the *characteristic lines* of the equation. However exploiting this fact is only possible in the very simple case considered here in which c is a constant. In a real-life application the quantity analogous to c would be different from place to place and time to time and, therefore, it is usually impossible to choose a discretization that guarantees a unit Courant number in the entire domain of the calculation for all times.

Unsurprisingly, the condition (7.4.15) has a physical interpretation. To see this, we note that, since the exact solution of the equation is $F(x - ct) = \text{const.}$, the wave delivered by the equation at the point at $(x_j,\, t^{n+1})$, namely $F(x_j - ct^{n+1})$, must equal the value of F at the earlier time t^n at some point x^*. To find the position of this point we note that it must satisfy $F(x_j - ct^{n+1}) = F(x^* - ct^n)$, which requires that the arguments of F be the same, i.e. that

$$x^* - ct^n \;=\; x_j - c(t^n + \Delta t)\,. \tag{7.4.16}$$

Solving we find

$$x_* \;=\; x_j - c\Delta t \;=\; x_{j-1} + \Delta x - c\Delta t\,. \tag{7.4.17}$$

In words, the solution at $(x_j,\, t^{n+1})$ is obtained by propagating with velocity c the value of F which, at the earlier time t^n, resided at the point $x_j - c\Delta t$. Of course, since we only calculate the solution at the grid points, we do not know the value of $F(x^* - ct^n)$, but we can find it by interpolation using the values at x_{j-1} and x_j. This is precisely what the discretization (7.3.1) does. Indeed, by rewriting (7.4.1) as

$$u_j^{n+1} \;=\; u_j^n - \frac{c\Delta t}{\Delta x}\left(u_j^n - u_{j-1}^n\right) \;=\; \left(1 - \frac{c\Delta t}{\Delta x}\right) u_j^n + \frac{c\Delta t}{\Delta x} u_{j-1}^n\,, \tag{7.4.18}$$

we readily see that the right-hand side is precisely an estimate of the value of u^n at the point $x^* = x_j - c\Delta t$ obtained by a linear interpolation between the values u_{j-1}^n at x_{j-1} and u_j^n at x_j.

If $c\Delta t$ were greater than Δx, this point x^* would be outside the segment $x_{j-1} \le x \le x_j$ which would mean that u_{j-1}^n at the point x_{j-1} could not have been affected by the information carried by the wave and, therefore, it would not have valid information to contribute to u_j^{n+1}.[36]

The same analysis can be applied to the equation (7.2.6) for left-propagating waves. The result is the same as in (7.4.6) and (7.4.7) with c replaced by $-c$. We then see from (7.4.6) that, with this substitution, $D < 0$ becomes negative no matter how Δt and Δx are chosen, so that the discretization (7.3.1) is unconditionally unstable for left-propagating waves.

All the other discretizations shown in section 7.3 can be studied in the same way. In particular, if we consider the explicit forward Euler discretization (7.3.3), we find the same equation (7.4.7) in which D is given by an expression that we write in such a way as to make the comparison with (7.4.6) easier:

$$D \;=\; \frac{(-c)\Delta x}{2}\left(1 - \frac{(-c)\Delta t}{\Delta x}\right)\,. \tag{7.4.19}$$

If $c > 0$, D will be negative irrespective of the choice of Δt and Δx and therefore the metod will be unstable. However, if $c < 0$, the calculation is stable provided that

$$\frac{|c|\Delta t}{\Delta x} \le 1\,. \tag{7.4.20}$$

[36] It can be seen from the solution (7.2.10) of the second-order equation that, at t^{n+1}, the solution at x_j is built from the earlier values at t^n issuing from the points $x_1 - ct^n = x_j - c(t^n + \Delta t)$ and $x_2 + ct^n = x_j + c(t^n + \Delta t)$, i.e., the points $x_1 = x_j - c\Delta t$ and $x_2 = x_j + c\Delta t$. The complete solution shows that the integral of the values of v in the interval $x_1 \le x \le x_2$ is also required. Thus, this interval includes the *domain of dependence*, at time t^n, of the solution at x_j at the later time t^{n+1}. If $c\Delta t$ was greater than Δx, one would have effects, at t^{n+1}, dependent on a region larger than the domain of dependence and, therefore, propagating at a speed faster than c.

We thus have found something rather peculiar: for waves propagating to the right (i.e., toward increasing x) it is necessary to use the backward Euler method to obtain stable solutions, while the forward Euler method is necessary for waves propagating in the opposite direction; in either cases the Courant number cannot exceed 1. This conclusion is at the root of what is called *upwind differencing*: for stability the spatial derivative must be discretized using the node "upwind" (i.e., in the direction opposite to the "wind velocity" c) of the node of interest, namely u_{j-1} if $c > 0$ and u_{j+1} if $c < 0$. This is a standard procedure, for example, for the discretization of the convection terms in the solution of the Navier-Stokes equations.

7.5 Dispersive error

Let us keep one more term in the expansions (7.4.2) and (7.4.4) for $\partial u/\partial x$ and $\partial u/\partial t$:

$$\frac{u_j^{n+1} - u_j^n}{\Delta t} \;=\; \frac{\partial u}{\partial t} + \frac{1}{2}\Delta t \frac{\partial^2 u}{\partial t^2} + \frac{1}{6}\Delta t^2 \frac{\partial^3 u}{\partial t^3} + \dots \,, \tag{7.5.1}$$

$$\frac{u_j^{n} - u_{j-1}^n}{\Delta x} \;=\; \frac{\partial u}{\partial x} - \frac{1}{2}\Delta x \frac{\partial^2 u}{\partial x^2} + \frac{1}{6}\Delta x^2 \frac{\partial^3 u}{\partial x^3} + \dots \,. \tag{7.5.2}$$

We can re-express the time derivatives in the first equation in terms of space derivatives. [37] Upon substituting into the explicit forward-Euler discretization (7.4.1) we then find

$$\frac{\partial u}{\partial t} + c\frac{\partial u}{\partial x} \;=\; D\frac{\partial^2 u}{\partial x^2} - cE\frac{\partial^3 u}{\partial x^3} \,, \tag{7.5.3}$$

with D given by (7.4.6) and

$$E \;=\; \frac{\Delta x^2}{6}\left(1 - \frac{c\Delta t}{\Delta x}\right)\left(1 - \frac{2c\Delta t}{\Delta x}\right) . \tag{7.5.4}$$

Proceeding in the usual way we assume that $u(x,t) = G(t)e^{ikx}$ and find

$$\frac{dG}{dt} \;=\; -(ikc + k^2 D - ik^3 cE)G \,, \tag{7.5.5}$$

from which, with $G(0) = 1$, we find

$$G(t) \;=\; \exp\left[-ikc(1 - Ek^2)t\right] e^{-k^2 Dt} \,, \tag{7.5.6}$$

from which

$$u \;=\; \exp\left[ikx - ikc(1 - Ek^2)t\right] e^{-k^2 Dt} \,. \tag{7.5.7}$$

Upon comparison with the exact solution (7.4.9) we notice two significant differences. The first one is the damping of the wave proportionally to $\exp(-k^2 Dt)$ as found before. The new feature is the fact that the speed of propagation is not the constant c independent of k, but now has a k-dependence and has the effective value

$$c_{eff}(k) \;=\; (1 - Ek^2)c \,. \tag{7.5.8}$$

[37] The calculation in this case is not as strightforward as the step leading to (7.4.4). The reason is explained in footnote 33 and has to do with the fact that now we need a result correct to second order terms included. Thus, in calculating $\partial_t^2 u$, we should not use the original equation (7.1.6) but the effective equation (7.4.7) correct to first order. In so doing we find

$$\partial_t^2 u \simeq \partial_t\left(-c\partial_x u + D\partial_x^2 u\right) \;=\; \left(-c\partial_x u + D\partial_x^2 u\right)\partial_t u \;=\; \left(-c\partial_x + D\partial_x^2\right)^2 u \;=\; c^2\partial_x^2 u - 2cD\partial_x^3 u + (\text{2nd order terms}) \,.$$

With this result we have

$$\frac{u^{n+1} - u^n}{\Delta t} \;\simeq\; \partial_t u + \frac{\Delta t}{2}c^2\partial_x^2 u - \frac{\Delta t \Delta x}{6}c^2\left(3 - 2\frac{c\Delta t}{\Delta x}\right)\partial_x^3 u + (\text{3nd order terms}) \,.$$

Upon combining with (7.5.2) we find (7.5.3).

Thus, the solution of the finite-difference equation describes waves the propagation velocity of which is not constant, as in the original differential equation, but depends on the wavelength of the perturbation. Depending on the sign of E and the magnitude of k this effective speed of propagation can be larger or smaller than c, and it can also be negative. According to the theory of the. Fourier series, any initial form of the wave can be written as a (finite or infinite) superposition of sinusoidal waves:

$$u(x,0) \;=\; \sum_n c_n e^{ik_n x}\,. \tag{7.5.9}$$

The coefficients c_n are such that, once the various monochromatic components are added together, they reproduce the initial disturbance $u(x,0)$. Since each component propagates according to (7.5.7), at a later time $t > 0$ the wave will be given by

$$u(x,t) \;=\; \sum_n c_n e^{i[k_n x - c(1-Ek_n^2)t]} e^{-k_n^2 Dt}\,. \tag{7.5.10}$$

In addition to the damping effect due to numerical diffusion, we see that the various components propagate at different velocities and therefore acquire a phase difference with respect to one other. The end result is that the wave changes shape. This form of error is called *dispersive* because it leads to a "dispersion" of the wave shape. The effect is analogous to the effect of a prism which disperses the various components of white light due to the different speed of propagation of the individual monochromatic constituents.

7.6 Higher-order schemes

The diffusive behavior that we have encountered is due to the first-order error of the discretizations we have used for the derivatives. As far as the space derivative is concerned, we can improve the accuracy by using the centered formula. A way to improve the time discretization is the following.

From the Taylor series expansion

$$\frac{u_j^{n+1} - u_j^n}{\Delta t} \;=\; \frac{\partial u}{\partial t} + \frac{1}{2}\Delta t \frac{\partial^2 u}{\partial t^2} + O(\Delta t^2) \tag{7.6.1}$$

we see that, if we were to approximate $\partial u/\partial t$ as

$$\frac{\partial u}{\partial t} \;=\; \frac{u_j^{n+1} - u_j^n}{\Delta t} - \frac{1}{2}\Delta t \frac{\partial^2 u}{\partial t^2} + \ldots \tag{7.6.2}$$

the error would become of order Δt^2 rather than Δt. As we have already remarked after (7.4.4) this can also be written

$$\frac{\partial u}{\partial t} \;=\; \frac{u_j^{n+1} - u_j^n}{\Delta t} - \frac{1}{2}\Delta t c^2 \frac{\partial^2 u}{\partial x^2} + \ldots\,. \tag{7.6.3}$$

The idea now is to use the usual centered difference formula to express the second space derivative writing

$$\frac{\partial u}{\partial t} \;\simeq\; \frac{u_j^{n+1} - u_j^n}{\Delta t} - \frac{\Delta t c^2}{2}\,\frac{u_{j-1}^n - 2u_j^n + u_{j+1}^n}{\Delta x^2}\,. \tag{7.6.4}$$

Upon substituting this relation for $\partial u/\partial t$ and using the centered formula for $\partial u/\partial x$ in the equation (7.1.6) we then find

$$\frac{u_j^{n+1} - u_j^n}{\Delta t} - \frac{\Delta t c^2}{2}\,\frac{u_{j-1}^n - 2u_j^n + u_{j+1}^n}{\Delta x^2} + c\frac{u_{j+1}^n - u_{j-1}^n}{2\Delta x} \;=\; 0\,. \tag{7.6.5}$$

Upon reorganizing this relation we have the explicit *Lax-Wendroff* scheme:

$$u_j^{n+1} \;=\; u_j^n - \frac{c\Delta t}{2\Delta x}(u_{j+1}^n - u_{j-1}^n) + \frac{c^2\Delta t^2}{2\Delta x^2}\left(u_{j-1}^n - 2u_j^n + u_{j+1}^n\right)\,. \tag{7.6.6}$$

The usual procedure applied to this equation gives the modified equation

$$\frac{\partial u}{\partial t} + c\frac{\partial u}{\partial x} \;=\; \frac{c\Delta x^2}{6}\left(1 - \frac{c^2\Delta t^2}{\Delta x^2}\right)\frac{\partial^3 u}{\partial x^3}\,. \tag{7.6.7}$$

Remarkably, the damping effect is absent (of course, it is present at higher order, and therefore it is less troublesome) and the dominant error is dispersive. This equation gives no information on the stability of this discretization as, applying the same procedure used in section 7.5, we would find $|G(t)| = 1$. The stability criterion can be found by going one more order to include terms of the form $\partial^4 u/\partial x^4$ and, upon so doing, it is found that the method is stable under the same condition (7.4.15) found before.

7.7 The second-order wave equation

We have seen that the second-order wave equation (7.1.1) can be split into the system (7.1.5) and (7.1.6) and this system can form the basis for a numerical method. On the assumption that $c > 0$ (which is no restriction since only c^2 appears in the equation), a simple discretization would be

$$\frac{u_j^{n+1} - u_j^n}{\Delta t} + c\frac{u_j^n - u_{j-1}^n}{\Delta x} \;=\; 0\,, \qquad \frac{v_j^{n+1} - v_j^n}{\Delta t} - c\frac{v_{j+1}^n - v_j^n}{\Delta x} \;=\; u_j^n\,. \tag{7.7.1}$$

Notice that (because of the assumption that $c > 0$) we have used *upwind differencing* in both space derivatives. Stability of this method requires that $c\Delta t < \Delta x$.

The method can be improved by using the Lax-Wendroff idea based on (7.6.3):

$$\frac{u_j^{n+1} - u_j^n}{\Delta t} \;=\; \frac{\partial u}{\partial t} - \frac{1}{2}\Delta t\frac{\partial^2 u}{\partial t^2} + \dots\,, \qquad \frac{v_j^{n+1} - v_j^n}{\Delta t} \;=\; \frac{\partial v}{\partial t} - \frac{1}{2}\Delta t\frac{\partial^2 v}{\partial t^2} + \dots\,. \tag{7.7.2}$$

As before we see from the first one of (5) that $\partial_t^2 u = c^2\partial_x^2 u$ and we see directly from the wave equation that the same relation is satisfied by v, namely $\partial_t^2 v = c^2\partial_x^2 v$. Thus, as before, we have the scheme

$$u_j^{n+1} \;=\; u_j^n - \frac{c\Delta t}{2\Delta x}(u_{j+1}^n - u_{j-1}^n) + \frac{c^2\Delta t^2}{2\Delta x^2}\left(u_{j-1}^n - 2u_j^n + u_{j+1}^n\right)\,. \tag{7.7.3}$$

$$v_j^{n+1} \;=\; v_j^n + \frac{c\Delta t}{2\Delta x}(v_{j+1}^n - v_{j-1}^n) + \frac{c^2\Delta t^2}{2\Delta x^2}\left(v_{j-1}^n - 2v_j^n + u_{j+1}^n\right) + u_j^n\,. \tag{7.7.4}$$

The second-order equation can also be discretized directly. The simplest discretization is

$$\frac{v_j^{n+1} - 2v_j^n + v_j^{n-1}}{\Delta t^2} \;=\; c^2\frac{v_{j-1}^n - 2v_j^n + v_{j+1}^n}{\Delta x^2}\,. \tag{7.7.5}$$

This is an explicit method, in that it gives directly v_j^{n+1} if the solution at the two previous time levels is known. Not surprisingly, it is found that this scheme is stable only if $c\Delta t \le \Delta x$.

The method requires a special treatment for the first time step $t^1 = \Delta t$ as can be seen by setting $n = 0$ in (6):

$$\frac{v_j^1 - 2v_j^0 + v_j^{-1}}{\Delta t^2} \;=\; c^2\frac{v_{j-1}^0 - 2v_j^0 + v_{j+1}^0}{\Delta x^2}\,. \tag{7.7.6}$$

Here $v_j^0 = f(x_j)$ is just the initial condition from (7.1.2), but v_j^{-1}, i.e., v_j at time $t = -\Delta t$, is undefined. We can handle this problem by adapting to the case of the time variable the ghost node idea we that we used for the space variable in the case of Neumann type boundary conditions. We write

$$\frac{v_j^1 - v_j^{-1}}{2\Delta t} \;=\; g_j\,, \tag{7.7.7}$$

where the function $g_j = g(x_j)$ is known from the initial conditions (7.1.2). From here we have

$$v_j^{-1} \;=\; v_j^1 - 2\Delta t g_j \,. \tag{7.7.8}$$

In this way (7.7.6) becomes

$$\frac{v_j^1 - 2f_j + (v_j^1 - 2\Delta t g_j)}{\Delta t^2} \;=\; c^2 \frac{f_{j-1} - 2f_j + f_{j+1}}{\Delta x^2} \,, \tag{7.7.9}$$

or

$$\frac{2v_j^1 - 2f_j - 2\Delta t g_j}{\Delta t^2} \;=\; c^2 \frac{f_{j-1} - 2f_j + f_{j+1}}{\Delta x^2} \,. \tag{7.7.10}$$

The next time step with $n = 1$ will involve v_j^1, given by this relation, and $v_j^0 = f_j$. The relation (7.7.6) applies to all the following time steps without modification.

A more stable method can be based on an idea similar to the Crank-Nicolson discretization of the diffusion equation, namely

$$\frac{v_j^{n+1} - 2v_j^n + v_j^{n-1}}{\Delta t^2} \;=\; \frac{c^2}{2} \left(\frac{v_{j-1}^{n+1} - 2v_j^{n+1} + v_{j+1}^{n+1}}{\Delta x^2} + \frac{v_{j-1}^{n-1} - 2v_j^{n-1} + v_{j+1}^{n-1}}{\Delta x^2} \right) . \tag{7.7.11}$$

It is found that the modulus of the amplification factor for this scheme equals 1 whatever the choice of Δt and Δx, so that the method is never unstable. As before, when this equation is written for $n = 1$, the last three terms in the right-hand side introduce values of v at the unphysical time $-\Delta t$. This issue can be handled by means of ghost (time) nodes as before.

7.8 Supplement: Stability

As we have remarked in the case of the diffusion equation, the method of effective equations gives us an indication of the way in which the analytic solution may differ from the numerical one and is suggestive of the stability properties of the discretization scheme but, due to the neglect of the higher-order terms of the truncation error, which is only justifed for relatively large spatial scales, it does not give complete results as to the stability of the scheme. It is therefore of some interest to apply the von Neumann method to investigate the stability properties of various possible discretizations irrespective of the limitation on the scale of the perturbation. We thus set, as in (7.4.10)

$$u_j^n \;=\; G^n e^{ikx_j} \;=\; G^n e^{ikj\Delta x} \;=\; G^n e^{ik^* j} \,, \tag{7.8.1}$$

in which k^* is the dimensionless wave number which takes the values

$$k^* \;=\; k\Delta x \;=\; \frac{M\pi\Delta x}{L} \,, \qquad M \;=\; 0,\, \pm 1,\, \pm 2,\, \ldots \,. \tag{7.8.2}$$

Since

$$G^n \;=\; \frac{G^n}{G^{n-1}} G^{n-1} \;=\; VG^{n-1} \;=\; V\frac{G^{n-1}}{G^{n-2}} G^{n-2} \;=\; V^2 G^{n-2} \;=\; \ldots V^n G(0) \,, \tag{7.8.3}$$

stability requires that $|V|^n$ remain bounded for all n's and values of k^* which, in turn, requires $|V| \leq 1$.

Let us begin from the explicit centered-difference scheme (7.3.5) which has the attractive feature of being of second-order accuracy in x. Upon substituting (7.8.1) into the equation we find

$$V \;=\; \frac{G^{n+1}}{G^n} \;=\; 1 - i\frac{c\Delta t}{\Delta x} \sin k^* \,. \tag{7.8.4}$$

The modulus of this quantity

$$|V| \;=\; \sqrt{1 + \left(\frac{c\Delta t}{\Delta x}\right)^2 \sin^2 k^*} \,, \tag{7.8.5}$$

is always greater than 1, which makes the scheme unconditionally unstable. A similar analysis shows that (7.3.6), the implicit version of this scheme, is, on the other hand, unconditionally stable.

Turning now to the first-order explicit backward Euler scheme that we have studied in section 7.4 we find, in place of (7.8.4),

$$V \;=\; \frac{G^{n+1}}{G^n} \;=\; 1 - \frac{c\Delta t}{\Delta x}\left(1 - e^{-ik^*}\right). \tag{7.8.6}$$

Since, as we know, this method is conditionally stable, it pays to go a little deeper in the analysis of this result. By recalling Euler's formula, $e^{-i\alpha} = \cos\alpha - i\sin\alpha$, we can separate the modulus and the phase of V:

$$V \;=\; |V|e^{-i\phi} \;=\; |V|\cos\phi - i|V|\sin\phi\,, \tag{7.8.7}$$

finding

$$|V| \;=\; \sqrt{1 - 4\left(1 - \frac{c\Delta t}{\Delta x}\right)\frac{c\Delta t}{\Delta x}\sin^2\frac{k^*}{2}}\,, \tag{7.8.8}$$

$$\cos\phi \;=\; \frac{1 - 2(c\Delta t/\Delta x)\sin^2 k^*/2}{|V|}\,, \qquad \sin\phi \;=\; \frac{(c\Delta t/\Delta x)\sin k^*}{|V|}\,. \tag{7.8.9}$$

It is readily seen that, for $c > 0$, the modulus is smaller than 1 provided that $c\Delta t/\Delta x < 1$. This is the same CFL condition (7.4.15) found earlier and, therefore, the von Neumann method does not produce new information in this respect. It does however give a fuller picture of how the diffusive error depends on the wave number k^*. If we take $G(0) = 1$ we find from (7.8.1)

$$u_j^n \;=\; V^n e^{ik^* j} \;=\; |V|^n \exp(ik^* j - in\phi)\,. \tag{7.8.10}$$

Upon comparison with (7.4.14), we realize that $|V|^n$ plays the same role as $e^{-Dk^2 t}$ in that relation. However, we should appreciate that we found the effective equation (7.4.7) by truncating the Taylor series expansion, which is justified provided that the higher-order terms are small. With an exponential space dependence as in (7.8.1) this is equivalent to assuming that k^* is small, i.e., long waves. For this reason we should expect to recover the previous results of section 7.4 not for all k^*, but only for $k^* \ll 1$ and it is in this case that we would expect $|V|^n$ to reduce to $e^{-Dk^2 t}$. We can check this by expanding (7.8.8) for small k^* to find

$$|V| \;=\; 1 - \frac{1}{2}\left(1 - \frac{c\Delta t}{\Delta x}\right)\frac{c\Delta t}{\Delta x}k^{*2}\,, \tag{7.8.11}$$

so that, since $t = n\Delta t$,

$$|V|^n \;=\; \left[1 - \frac{1}{2}k^{*2}\left(1 - \frac{c\Delta t}{\Delta x}\right)\frac{ct}{n\Delta x}k^{*2}\right]^n. \tag{7.8.12}$$

For large n this becomes,[38]

$$|V|^n \;\simeq\; \exp\left[-\left(1 - \frac{c\Delta t}{\Delta x}\right)\frac{c}{2\Delta x}k^{*2}t\right] \;=\; \exp\left[-\left(1 - \frac{c\Delta t}{\Delta x}\right)\frac{c\Delta x}{2}k^{*2}t\right], \tag{7.8.13}$$

which indeed equals $e^{-Dk^{*2}t}$ with D given by (7.4.6).

Comparison of (7.8.10) with (7.5.7) suggests that the exponential factor $\exp(-in\phi)$ should include information on dispersion effects. As before, we start by looking at the case $k^* \ll 1$, where we expect to find

[38] Recall that

$$\lim_{n\to\infty}\left(1 - \frac{a}{n}\right)^n \;=\; e^{-a}\,.$$

$\exp(-in\phi) \simeq \exp\left[-ikc(1-Ek^2)t\right]$. For small k^*, from the second one of (7.8.9) we find[39]

$$\phi \simeq \frac{c\Delta t}{\Delta x} k^* \left[1 + \frac{1}{6}\left(1 - \frac{c\Delta t}{\Delta x}\right)\left(1 - \frac{2c\Delta t}{\Delta x}\right) k^{*2}\right], \tag{7.8.14}$$

so that, setting as before $n = t/\Delta t$ and $k^* = k\Delta x$, we have

$$n\phi \simeq kct \left[1 - \frac{\Delta x^2}{6}\left(1 - \frac{c\Delta t}{\Delta x}\right)\left(1 - \frac{2c\Delta t}{\Delta x}\right) k^2\right], \tag{7.8.15}$$

which gives precisely the factor $\exp\left[-ikc(1-Ek^2)t\right]$ in (7.5.7) with E given by (7.5.4).

[39] This is the calculation. The expression for $\sin\phi$ in (7.8.9) shows that ϕ is small when k^* is small. Thus we expand both sides of the equation:

$$\phi - \frac{1}{6}\phi^3 = \frac{c\Delta t}{\Delta x} k^* \left(1 - \frac{1}{6}k^{*2}\right)\left[1 - \frac{1}{2}\frac{c\Delta t}{\Delta x}\left(1 - \frac{c\Delta t}{\Delta x}\right)\right].$$

Since ϕ is small, ϕ^3 can be estimates using only the leading order of the right-hand side to find $\phi^3 \simeq k^3(c\Delta t/\Delta x)^3$. In this way we find the result (7.8.14).

Chapter 8

Ordinary Differential Equations

9.1 Simple methods for a single first-order ODE

Let us start with a very simple problem

$$\frac{du}{dt} \,=\, f(t)\,, \qquad u(t=0) \,=\, a\,. \tag{9.1.1}$$

The solution evidently is

$$u(t) \,=\, a + \int_0^t f(\tau) d\tau\,. \tag{9.1.2}$$

Our objective is to generate numerical approximations to this exact result. As usual we start by discretizing the independent variable by setting $t^n = n\Delta t$, $t^{n+1} = t^n + \Delta t$, with $n = 0,\, 1,\, 2,\, \ldots$, and we write $u(t^n) = u^n$, $u(t^{n+1}) = u^{n+1}$ to denote the values of u at times t^n and t^{n+1}.[40] For now we use a fixed value of Δt. We will see later that it is possible (and it may be actually necessary) to adjust Δt as the integration proceeds.

Suppose we have calculated u up to the time t^n:

$$u(t^n) \,=\, u^n \,=\, a + \int_0^{t^n} f(\tau) d\tau\,. \tag{9.1.3}$$

At the later time $t^{n+1} = t^n + \Delta t$ the exact value of u is given by

$$u(t^{n+1}) \,=\, u(t^n) + \int_{t^n}^{t^{n+1}} f(\tau) d\tau\,. \tag{9.1.4}$$

The problem is then to approximate the integral so that we can then proceed to u^{n+2} and so on. Let us consider a few options:

1. We can carry out the integration approximately by keeping f constant and equal to its value at t^n; in so doing we would find

$$u(t^{n+1}) \,\simeq\, u(t^n) + f(t^n)(t^{n+1} - t^n) \,=\, u(t^n) + \Delta t f(t^n)\,. \tag{9.1.5}$$

2. We can carry out the integration approximately by keeping f constant and equal to its value at t^{n+1}; in so doing we would find

$$u(t^{n+1}) \,\simeq\, u(t^n) + f(t^{n+1})(t^{n+1} - t^n) \,=\, u(t^n) + \Delta t f(t^{n+1})\,. \tag{9.1.6}$$

3. We can carry out the integration approximately by keeping f constant and equal to the average of its values at t^n and t^{n+1}; in so doing we would find

$$u(t^{n+1}) \,\simeq\, u(t^n) + \frac{1}{2}\left[f(t^{n+1}) + f(t^n)\right](t^{n+1} - t^n) \,=\, u(t^n) + \frac{\Delta t}{2}\left[f(t^{n+1}) + f(t^n)\right]\,. \tag{9.1.7}$$

[40] Note that here n is not an exponent, it is just an index. The convention frequently adopted is that spatial indices are written as subscripts, while time indices are written as superscripts. For example, u_j^n means $u(x_j, t^n)$.

Let us consider the errors affecting these approximations. Near t^n we may write

$$f(t) \,=\, f\Big(\, t^n + (t-t^n)\Big) \,=\, f(t^n) + (t-t^n)f'(t^n) + \frac{1}{2}(t-t^n)^2 f''(t^n) + \dots\,. \tag{9.1.8}$$

With this we have

$$\begin{aligned}\int_{t^n}^{t^{n+1}} f(\tau)d\tau \;&\simeq\; \int_{t^n}^{t^{n+1}} \left[f(t^n) + (\tau-t^n)f'(t^n) + \frac{1}{2}(\tau-t^n)^2 f''(t^n) + \dots\right] d\tau \\ &=\; (t^{n+1}-t^n)f(t^n) + \frac{1}{2}(t^{n+1}-t^n)^2 f'(t^n) + \dots \end{aligned} \tag{9.1.9}$$

We see therefore that, if we only keep the first term as in (9.1.5), the error resulting error is proportional to $\Delta t^2 \,=\, (t^{n+1}-t^n)^2$. We find the same result for (9.1.6). To prove it we write

$$f(t) \,=\, f\Big(\, t^{n+1} - (t^{n+1}-t)\Big) \,=\, f(t^{n+1}) - (t^{n+1}-t)f'(t^{n+1}) + \frac{1}{2}(t^{n+1}-t)^2 f''(t^{n+1}) + \dots\,, \tag{9.1.10}$$

and carry out the integration finding

$$\begin{aligned}\int_{t^n}^{t^{n+1}} f(\tau)d\tau \;&\simeq\; \int_{t^n}^{t^{n+1}} \left[f(t^{n+1}) + (t-t^{n+1})f'(t^{n+1}) + \frac{1}{2}(t-t^{n+1})^2 f''(t^{n+1}) + \dots\right] d\tau \\ &=\; (t^{n+1}-t^n)f(t^{n+1}) - \frac{1}{2}(t^{n+1}-t^n)^2 f'(t^{n+1}) + \dots \end{aligned} \tag{9.1.11}$$

By adding (9.1.9) and (9.1.11) and averaging we find

$$\int_{t^n}^{t^{n+1}} f(\tau)d\tau \;\simeq\; \frac{1}{2}\left[f(t^n) + f(t^{n+1})\right] - \frac{1}{4}\Delta t^2\left[f'(t^{n+1}) - f'(t^n)\right] + \dots \tag{9.1.12}$$

But we know that $f'(t^{n+1}) - f'(t^n) \,\simeq\, \Delta t f''(t^n)$ so that the previous result can be rewritten as

$$\int_{t^n}^{t^{n+1}} f(\tau)d\tau \,=\, \frac{1}{2}\left[f(t^n) + f(t^{n+1})\right] - \frac{1}{4}\Delta t^3 f''(t^n) + \dots \tag{9.1.13}$$

We thus conclude that the error in this case is proportional to Δt^3 rather than Δt^2 so that the *trapezoidal rule* (9.1.7) is more accurate than the other two.

For the particular case we have considered (9.1.5) and (9.1.6) have the same error and seem to be equivalent. There is however an important difference between these two formulae if we consider a more complicated equation than (9.1.1):

$$\frac{du}{dt} \,=\, f(t,u)\,, \qquad u(t=0) \,=\, a\,. \tag{9.1.14}$$

Note that here the unknown function u is also present in the function f in the right-hand side. The exact solution is now

$$u(t) \,=\, a + \int_0^t f\left(\tau, u(\tau)\right) d\tau\,, \tag{9.1.15}$$

and the analog of (9.1.4) would then be

$$u(t^{n+1}) \,=\, u(t^n) + \int_{t^n}^{t^{n+1}} f\left(\tau, u(\tau)\right) d\tau\,. \tag{9.1.16}$$

The approximation (9.1.5) extends very simply:

$$u(t^{n+1}) \,\simeq\, u(t^n) + (t^{n+1}-t^n) f\left(t^n, u(t^n)\right)\,. \tag{9.1.17}$$

The approximation (9.1.6) is superficially just as simple

$$u(t^{n+1}) \simeq u(t^n) + (t^{n+1} - t^n) f\left(t^n, u(t^{n+1})\right) . \tag{9.1.18}$$

However, on a closer inspection, we see that, while everything in the right-hand side of (9.1.17) is known, the unknown value $u(t^{n+1})$ is present in *both* the left- and right-hand sides of (9.1.18). For this reason, the formula (9.1.17) is called *explicit* while the formula (9.1.18) is called *implicit*.

EXAMPLE. Consider the problem

$$\frac{du}{dt} = \beta C e^{-\beta t} u^2 , \qquad u(0) = a . \tag{9.1.19}$$

The exact solution is easily found by separating the variables

$$\frac{du}{u^2} = \beta C e^{-\beta t} dt . \tag{9.1.20}$$

Integrate from $t = 0$, where $u = a$, to the generic t, where $u = u(t)$ to find

$$\int_a^{u(t)} \frac{du}{u^2} = \beta C \int_0^t e^{-\beta \tau} d\tau , \tag{9.1.21}$$

or

$$-\frac{1}{u(t)} + \frac{1}{a} = -C(e^{-\beta t} - 1) , \tag{9.1.22}$$

from which

$$u(t) = \frac{a}{1 + aC(e^{-bt} - 1)} . \tag{9.1.23}$$

It may be noted that, if $aC > 1$, the solution will diverge as t approaches

$$t = -\frac{1}{\beta} \log\left(1 - \frac{1}{aC}\right) . \tag{9.1.24}$$

Applied to (9.1.19) the explicit formula would give

$$u^{n+1} = u^n + \Delta t \beta C e^{-\beta t^n} (u^n)^2 , \tag{9.1.25}$$

but the implicit formula would give

$$u^{n+1} = u^n + \Delta t \beta C e^{-\beta t^{n+1}} (u^{n+1})^2 . \tag{9.1.26}$$

In order to find u^{n+1} we now need to solve the quadratic equation

$$\Delta t \beta C e^{-\beta t^{n+1}} (u^{n+1})^2 - u^{n+1} + u^n = 0 . \tag{9.1.27}$$

This is easily done in this case but if, for example, we had u^3 or u^4 or some other complicated function in place of u^2 we would have to use a numerical method to find the solution such as, for example, the Newton-Raphson method.

9.2 The Newton-Raphson method

The Newton-Raphson method is an efficient and widely used iterative method to approximate a zero of a function $F(x)$, i.e., a value x_* of x such that $F(x_*) = 0$. As described later, some care needs to be exercised when the function F has more than one zero; we address this point later.

The idea of the method is simple: Let x_κ be a provisional guess to a zero of F. To generate an improved approximation $x_{\kappa+1}$ to x_* so as to improve this guess we let $x_{\kappa+1} = x_\kappa + \Delta x$ and expand in Taylor series:

$$0 \simeq F(x_\kappa + \Delta x) = F(x_\kappa) + \Delta x F'(x_\kappa) + \dots . \tag{9.2.1}$$

We truncate the expansion to the first term and solve for Δx

$$\Delta x = x_{\kappa+1} - x_\kappa = -\frac{F(x_\kappa)}{F'(x_\kappa)} , \tag{9.2.2}$$

or

$$x_{\kappa+1} = x_\kappa - \frac{F(x_\kappa)}{F'(x_\kappa)} . \tag{9.2.3}$$

In words the method works as follows: start at x_κ and draw the tangent line to the function $F(x)$ at $x = x_\kappa$. The place $x_{\kappa+1}$ where this tangent intersects the x axis is the improved approximation required.[41]

As a simple example we can apply the method to the calculation of the n-th root of some number a. In this case what we want is to find the zero of the function $F(x) = x^n - a = 0$. The formula (9.2.3) gives

$$x_{\kappa+1} = x_\kappa - \frac{x_\kappa^n - a}{n x_\kappa^{n-1}} = \frac{(n-1)x_\kappa^n + a}{n x_\kappa^{n-1}} . \tag{9.2.4}$$

For example, for $n = 2$, in this way we find the algorithm already mentioned in section 4.1 for the iterative calculation of a square root.

If the function F has more than one zero, what happens is that the x axis breaks up into regions such that the method converges to one or the other zero depending on where the first guess x_0 is located. In practice (but not necessarily), to avoid converging to the "wrong" zero it is desirable to start with an initial guess "close enough" to the desired zero.[42] For example, the function $F(x) = x(1-x)$ has zeros at 0 and 1. In this case the Newton-Raphson formula (9.2.3) gives

$$x_{\kappa+1} = x_\kappa - \frac{x_\kappa - x_\kappa^2}{1 - 2x_\kappa} = \frac{x_\kappa^2}{2x_\kappa - 1} . \tag{9.2.5}$$

Clearly 0 and 1 are fixed points of this iteration, as they should be. If $|x_\kappa|$ is small, this becomes $x_{\kappa+1} \simeq -x_\kappa^2$ so that the iterations converge to 0. A simple graphical argument shows that the iterations converge to $x_* = 0$ if $x_0 < \frac{1}{2}$, while they converge to $x_* = 1$ if $x_0 > \frac{1}{2}$. As an example, Table 9.2.1 shows the sequence of iterations starting from $x_0 = -10$ and $x_0 = 10$. It is interesting to note that, due to the rounding caused by the finite number of digits used in the calculation, after a while the method produces the exact results rather than an infinite sequence of closer and closer approximations to them. In general, for the same reason, eventually x_κ would stop changing even though it may not quite coincide with the exact zero of F (for example, when the exact zero is an irrational number.)

The same idea applies directly to systems of equations. For example, if we want to approximate values (x_*, y_*) such that $F(x_*, y_*) = 0$ and $G(x_*, y_*) = 0$, implementing the same basic idea we would find the linear system

$$\frac{\partial F}{\partial x}\Delta x + \frac{\partial F}{\partial y}\Delta y = -F(x_\kappa, y_\kappa) , \tag{9.2.6}$$

$$\frac{\partial G}{\partial x}\Delta x + \frac{\partial G}{\partial y}\Delta y = -G(x_\kappa, y_\kappa) , \tag{9.2.7}$$

[41] The tangent to the curve $F(x)$ at the point $x = x_\kappa$ is $y = F(x_\kappa) + F'(x_\kappa)(x - x_\kappa)$. Setting $y = 0$ and $x = x_{\kappa+1}$ we find (9.2.3).

[42] How close is "close enough" will depend on the specific form of the function F, as shown in the example that follows.

x_k	$x_k(1-x_k)$	x_k	$x_k(1-x_k)$
-10.0000000	-110.000000	10.0000000	-90.0000000
-4.76190472	-27.4376411	5.26315784	-22.4376736
-2.15470791	-6.79747391	2.90782189	-5.54760647
-0.874440193	-1.63908589	1.75582504	-1.32709658
-0.278166175	-0.355542600	1.22744870	-0.279181600
-4.97171581E-02	-5.21889552E-02	1.03555775	-3.68220992E-02
-2.24824622E-03	-2.25330098E-03	1.03555775	-3.68220992E-02
-5.03170304E-06	-5.03172805E-06	1.00118041	-1.18180376E-03
-2.54658516E-11	-2.54658516E-11	1.00000143	-1.43051352E-06
0	0	1	0

Table 9.2.1: The Newton-Raphson iterative method applied to the function $F(x) = x(1-x)$.

where the partial derivatives are calculated at (x_κ, y_κ). We would then solve this system for $\Delta x = x_{\kappa+1} - x_\kappa$ and $\Delta y = y_{\kappa+1} - y_\kappa$. It may be noted that the determinant of the system is the Jacobian of F and G.

9.3 Heun's method

The same difficulty encountered with the implicit method affects the more accurate trapezoidal rule:

$$u^{n+1} = u^n + \frac{1}{2}\left[f(t^n, u^n) + f(t^{n+1}, u^{n+1})\right]\Delta t + O(\Delta t^3)\,. \tag{9.3.1}$$

However, the error analysis leading to (9.1.13) suggests a considerable simplification. Suppose that we can approximate $f(t^{n+1}, u^{n+1})$ with an error of order Δt^2. Since this term is multiplied by Δt, the final error introduced in the result will be of order Δt^3, which is the same as the order of the error of the entire formula (9.3.1) itself. This step therefore would not increase the error of the result, but it might considerably simplify the calculation. We exploit this remark by setting

$$f(t^{n+1}, u^{n+1}) \simeq f\left(t^{n+1}, u^n + \Delta t f(t^n, u^n)\right)\,. \tag{9.3.2}$$

The error affecting u in the right-hand side is proportional to Δt^2, but the great advantage is that now f^{n+1}, approximated in this way, is completely known.[43] This is called *Heun's method* and can be written as a two-step method:

$$\text{Step 1} \qquad u^* = u^n + \Delta t f(t^n, u^n)\,, \tag{9.3.3}$$

$$\text{Step 2} \qquad u^{n+1} = u^n + \frac{1}{2}\Delta t\left[f(t^n, u^n) + f(t^{n+1}, u^*)\right]\,. \tag{9.3.4}$$

The first step is called a *predictor step* for u^{n+1}, the second step is the *corrector step*. This is a simple example of a family of methods called predictor-corrector methods.

There is a whole class of similar methods which can be written as

$$(a) \qquad u^* = u^n + \alpha\Delta t f(t^n, u^n)\,, \tag{9.3.5}$$

$$(b) \qquad u^{n+1} = u^n + \frac{\Delta t}{2\alpha}\left[(2\alpha - 1)f(t^n, u^n) + f(t^n + \alpha\Delta t, u^*)\right]\,, \tag{9.3.6}$$

[43] The proof is as follows. Let us write for brevity $u^* = u^n + f(t^n, u^n)\Delta t$. Then we are interested in $f(t^{n+1}, u^{n+1}) - f(t^{n+1}, u^*)$. We can write

$$f(t^{n+1}, u^*) = f(t^{n+1}, u^* + (u^{n+1} - u^*)) \simeq f(t^{n+1}, u^{n+1}) + (u^{n+1} - u^*)\frac{\partial f}{\partial u}\,,$$

which, since $u^{n+1} - u^*$ is of order Δt^2, proves that $f(t^{n+1}, u^{n+1}) - f(t^{n+1}, u^*)$ is of order Δt^2.

where α is an arbitrary parameter. All these methods have the same accuracy of order Δt^2. Clearly, by taking $\alpha = 1$, we are back to Heun's method (9.3.5), (9.3.6). The choice $\alpha = \frac{1}{2}$ gives the so-called mid-point rule:

$$u^{n+1} \;=\; u^n + f\left(\frac{1}{2}(t^n + t^{n+1}), u^n + \frac{1}{2}\Delta t f(t^n, u^n)\right)\,\Delta t\,. \tag{9.3.7}$$

9.4 Systems of ODEs

The previous methods extend in a pretty straightforward way to systems of ODEs. For example, let us consider how Heun's method would be applied to the system

$$\frac{du_1}{dt} \;=\; f_1(t, u_1, u_2)\,, \qquad \frac{du_2}{dt} \;=\; f_2(t, u_1, u_2)\,. \tag{9.4.1}$$

The first step is:

$$u_1^* \;=\; u_1^n + \Delta t f_1(t^n, u_1^n, u_2^n)\,, \tag{9.4.2}$$
$$u_2^* \;=\; u_2^n + \Delta t f_2(t^n, u_1^n, u_2^n)\,, \tag{9.4.3}$$

and the second step

$$u_1^{n+1} \;=\; u_1^n + \frac{\Delta t}{2}\left[f_1(t^n, u_1^n, u_2^n) + f_1(t^{n+1}, u_1^*, u_2^*)\right]\,, \tag{9.4.4}$$
$$u_2^{n+1} \;=\; u_2^n + \frac{\Delta t}{2}\left[f_2(t^n, u_1^n, u_2^n) + f_2(t^{n+1}, u_1^*, u_2^*)\right]\,. \tag{9.4.5}$$

Although we have only specifically addressed first-order equations and systems, there is no loss of generality because any higher-order equation can be reduced to a system. For example, the equation

$$\frac{d^2u}{dt^2} + a(t)\frac{du}{dt} + b(t)u(t) \;=\; f(t)\,, \tag{9.4.6}$$

with initial conditions

$$u(t=0) \;=\; u_0\,, \qquad \left.\frac{du}{dt}\right|_{t=0} \;=\; v_0\,, \tag{9.4.7}$$

is evidently equivalent to the two-equations system

$$\frac{du}{dt} \;=\; v(t)\,, \tag{9.4.8}$$
$$\frac{dv}{dt} \;=\; f(t) - a(t)v(t) - b(t)u(t)\,, \tag{9.4.9}$$

with initial conditions

$$u(t=0) \;=\; u_0\,, \qquad v(t=0) \;=\; v_0\,. \tag{9.4.10}$$

Example 1. Let us consider a simple harmonic oscillator with damping; after dividing through by the body's mass the equation of motion is

$$\frac{d^2x}{dt^2} \;=\; -\omega_0^2 x - 2b\frac{dx}{dt}\,, \tag{9.4.11}$$

in which ω_0 has the physical meaning of the angular frequency of the oscillator and b is the damping parameter. We associate to this equation the initial conditions

$$x(0) \;=\; X_0\,, \qquad \left.\frac{dx}{dt}\right|_{t=0} \;=\; V_0\,. \tag{9.4.12}$$

The exact solution of the problem is

$$x(t) \;=\; e^{-bt}\,(A\cos\omega t + B\sin\omega t)\;, \qquad \omega \;=\; \sqrt{\omega_0^2 - b^2}\,. \tag{9.4.13}$$

The constants A and B are found from the initial conditions. For example, $x(0) = A = X_0$, and similarly for B by imposing the second condition in (9.4.12).

To apply Heun's method we start by reducing the second-order equation to a system of two first order equations. Evidently we can write this system as

$$\frac{dx}{dt} \;=\; v\,, \qquad \frac{dv}{dt} \;=\; -\omega_0^2 x - bv\,. \tag{9.4.14}$$

Upon comparing with (9.4.1) we see that

$$x \to u_1\,, \qquad u_2 \to v\,, \qquad f_1(t,u_1,u_2) \to f_1(t,x,v) \;=\; v\,, \qquad f_2(t,u_1,u_2) \to f_2(t,x,v) \;=\; -\omega_0^2 x - bv\,. \tag{9.4.15}$$

Then the first step of Heun's method is

$$x^* \;=\; x^n + \Delta t v^n\,, \tag{9.4.16}$$

$$v^* \;=\; v^n + \Delta t\left[-\omega_0^2 x^n - bv^n\right]\,, \tag{9.4.17}$$

and the second step

$$x^{n+1} \;=\; x^n + \frac{\Delta t}{2}\left[v^n + v^*\right]\,, \tag{9.4.18}$$

$$v^{n+1} \;=\; v^n + \frac{\Delta t}{2}\left[-\omega_0^2 x^n - bv^n - \omega^2 x^* - bv^*\right]\,. \tag{9.4.19}$$

Example 2. Let us consider the orbit of a planet around the sun. The orbit is planar and, in its plane, we choose a Cartesian system of coordinates (x, y) with the sun at the origin (because of the sun's huge mass, we can neglect the motion of its center of mass as the planet goes around it). After dividing through by the mass of the planet, the equations that govern its motion are

$$\frac{d^2x}{dt^2} \;=\; -\frac{GM_s x}{(x^2+y^2)^{3/2}}\,, \tag{9.4.20}$$

$$\frac{d^2y}{dt^2} \;=\; -\frac{GM_s y}{(x^2+y^2)^{3/2}}\,, \tag{9.4.21}$$

in which G is the gravitational constant and M_s the mass of the sun. We decompose this system of equations into four first-order equations:

$$\frac{dx}{dt} \;=\; v_x\,, \qquad \frac{dv_x}{dt} \;=\; -\frac{GM_s x}{(x^2+y^2)^{3/2}}\,, \tag{9.4.22}$$

$$\frac{dy}{dt} \;=\; v_y\,, \qquad \frac{dv_y}{dt} \;=\; -\frac{GM_s y}{(x^2+y^2)^{3/2}}\,, \tag{9.4.23}$$

so that

$$f_1(t,x,v_x,y,v_y) \;=\; v_x\,, \qquad f_2(t,x,v_x,y,v_y) \;=\; -\frac{GM_s x}{(x^2+y^2)^{3/2}}\,, \tag{9.4.24}$$

$$f_3(t,x,v_x,y,v_y) \;=\; v_y\,, \qquad f_4(t,x,v_x,y,v_y) \;=\; -\frac{GM_s y}{(x^2+y^2)^{3/2}}\,. \tag{9.4.25}$$

Then the first step would be

$$x^* = x^n + \Delta t v_x^n\,, \qquad y^* = y^n + \Delta t v_y^n\,, \tag{9.4.26}$$

$$v_x^* = v_x^n - \frac{GM_s \Delta t x^n}{[(x^n)^2 + (y^n)^2]^{3/2}}\,, \qquad v_y^* = v_y^n - \frac{GM_s \Delta t y^n}{[(x^n)^2 + (y^n)^2]^{3/2}}\,, \tag{9.4.27}$$

and the second step

$$x^{n+1} = x^n + \frac{1}{2}\Delta t\,[v_x^n + v_x^*]\,, \qquad v_x^{n+1} = v_x^n - \frac{1}{2} GM_s \Delta t \left[\frac{x^n}{[(x^n)^2 + (y^n)^2]^{3/2}} + \frac{x^*}{[(x^*)^2 + (y^*)^2]^{3/2}}\right] \tag{9.4.28}$$

$$y^{n+1} = y^n + \frac{1}{2}\Delta t\,\left[v_y^n + v_y^*\right]\,, \qquad v_y^{n+1} = v_y^n - \frac{1}{2} GM_s \Delta t \left[\frac{y^n}{[(x^n)^2 + (y^n)^2]^{3/2}} + \frac{y^*}{[(x^*)^2 + (y^*)^2]^{3/2}}\right]. \tag{9.4.29}$$

9.5 Runge-Kutta methods

The most famous class of methods for the initial-value problem associated with ordinary differential equations (or ordinary differential equation systems) goes under the name of *Runge-Kutta methods* of which Heun's method mey be considered as a low-order example. Just as for Heun's method, for each order of accuracy, there is an infinity of variants. The most well known are:

1. *Third-order Runge-Kutta method* (or simply Kutta method) is a three-step method:

$$(a) \qquad u^* = u^n + \frac{\Delta t}{2} f(t^n, u^n)\,, \tag{9.5.1}$$

$$(b) \qquad u^{**} = u^n + \Delta t \left[2f(t^n + \frac{1}{2}\Delta t, u^*) - f(t^n, u^n)\right]\,, \tag{9.5.2}$$

$$(c) \qquad u^{n+1} = u^n + \frac{\Delta t}{6}\left[f(t^n, u^n) + 4f(t_n + \frac{1}{2}\Delta t, u^*) + f(t^n + \Delta t, u^{**})\right]\,. \tag{9.5.3}$$

   The u^* given by the first step is evidently an estimate of u at the midpoint as given by the forward Euler method. The second step can be written as

$$u^{**} = u^n + \Delta t f(t^n + \frac{1}{2}\Delta t, u^*) + \Delta t \left[f(t^n + \frac{1}{2}\Delta t, u^*) - f(t^n, u^n)\right]\,. \tag{9.5.4}$$

   The first two terms are an estimate of u^{n+1} given by the midpoint rule, and the last term is a correction to it. Finally, u^{n+1} is the result of the weighted average of the slopes du/dt at the three points t^n, $t^n + \frac{1}{2}\Delta t$, $t^n + \Delta t$, with the central one given more weight than the first and the last on account of its somewhat greater accuracy. This method has a third-order accuracy meaning that the error is of order Δt^4.

2. The *Fourth-order Runge-Kutta method* is a four-step method:

$$(a) \qquad u^* = u^n + \frac{\Delta t}{2} f(t^n, u^n)\,, \tag{9.5.5}$$

$$(b) \qquad u^{**} = u^n + \frac{\Delta t}{2} f(t^n + \frac{1}{2}\Delta t, u^*)\,, \tag{9.5.6}$$

$$(c) \qquad u^{***} = u^n + \Delta t f(t^n, u^{**}) \tag{9.5.7}$$

$$(d) \qquad u^{n+1} = u^n + \frac{\Delta t}{6}\left[f(t^n, u^n) + 2f(t_n + \frac{1}{2}\Delta t, u^*) + 2f(t^n + \frac{1}{2}\Delta t, u^{**}) + f(t^n, u^{***})\right] \tag{9.5.8}$$

   This is the most famous of the Runge-Kutta methods and it is widely used for the integration of ordinary differential equations; it has an accuracy of order Δt^4 with the error being of order Δt^5.

9.6 Adaptive integration

In order to save computation, in solving a specific problem it is desirable to take time steps as long as possible, but not so long as to compromise accuracy. Since, with the methods described, each time step is calculated indepently from the preceding ones,[44] it is possible to adjust the time step in response to the needs of accuracy and efficiency as the integration proceeds; this process is termed *adaptive integration.*

The key ingredient is to have a way to "sense" how accurate a solution the integration process is producing "in real time," i.e., as the integration progresses. A standard way to achieve this objective is to take one step of length Δt starting at t^n, and two half-steps of length $\frac{1}{2}\Delta t$, also starting at t^n, and to compare the two solutions thus found, which are two different approximations to u^{n+1}:

$$\Delta \;=\; \left| u^{n+1}_{two\ steps} - u^{n+1}_{one\ step} \right| . \tag{9.6.1}$$

In the case of a system of equations we would be looking at the maximum of this difference over all the dependent variables. The general idea is that, if Δ, as computed, is very very small, we can safely increase the time step, for example doubling it for the next step. If, on the other hand, Δ is not sufficiently small, that signals that the time step currently being used is too long and needs to be reduced, e.g. halved.

Of course, "large" and "small" must also be related to some scale specific to the problem at hand, so that the quantity to focus on really is not Δ but some normalized form of Δ:

$$\frac{1}{\phi} \;=\; \frac{\Delta}{\Delta_{desired}} , \tag{9.6.2}$$

where $\Delta_{desired}$ is a measure of the maximum acceptable difference between the two solutions $u^{n+1}_{two\ steps}$ and $u^{n+1}_{one\ step}$. The time step can be increased if ϕ is larger than some limit (e.g., one or a few powers of 10), but it should be decreased if ϕ is smaller than 1. There are several possibilities:

- If, for example, the solution is oscillating with a characteristic amplitude A, one may want to take $\Delta_{desired} = \epsilon A$, with ϵ some small number, e.g., 10^{-6}.
- Another possibility is to use u^{n+1} itself as a scale by taking $\Delta_{desired} = \epsilon |u^{n+1}|$. In this way, the criterion requires that the difference should be a small fraction of the current value of the dependent variable.
- An empirical prescription founded on actual practice is:

  $$\Delta t_{new} \;=\; \Delta t\, \phi^{\alpha} , \tag{9.6.3}$$

  where Δt_{new} is the time step to be used for the next step. Here $\alpha = 0.2$ when $\Delta < \Delta_{desired}$ so that $\phi > 1$; in this case the time step is increased by this rule. When $\Delta > \Delta_{desired}$, $\phi < 1$ and α takes the value $\alpha = 0.25$. In this case, the rule leads to a decrease of the time step. The quantity to be used for $\Delta_{desired}$ when using this formula is

  $$\Delta_{desired} \;=\; \epsilon \left(|u| + \Delta t \left| \frac{du}{dt} \right| \right) . \tag{9.6.4}$$

9.7 The Method of Lines or Semi-Discretization

Let us go back to the diffusion equation:

$$\frac{\partial u}{\partial t} \;=\; D \frac{\partial^2 u}{\partial x^2} , \tag{9.7.1}$$

[44] For this reason these are called *one-step methods.* There are other classes of methods, called *multi-step methods*, in which taking the next time step involves a few of the preceding ones.

with D the diffusivity. Associated with this equation there will be an initial condition

$$u(x, t = 0) \;=\; F(x)\,, \tag{9.7.2}$$

and boundary conditions at the ends of the interval of interest.

Let us discretize the space dimension with equally spaced nodes x_0, x_1, x_2, …, x_j, …, x_{N+1}, where x_0 is the left boundary, at which u is known from one of the boundary conditions, and u_{N+1} is the right boundary where the other boundary condition gives information on u. Upon setting

$$u(x_j, t) \;=\; u_j(t)\,, \tag{9.7.3}$$

and using the standard centered-difference approximation for $\partial^2 u/\partial x^2$ we find

$$\frac{du_j(t)}{dt} \;=\; D\frac{u_{j-1}(t) - 2u_j(t) + u_{j+1}(t)}{\Delta x^2}\,. \tag{9.7.4}$$

These equations written for $j = 1, 2, \ldots$ constitute a systems of ordinary differential equations (ODEs) in which the unknowns are $u_1(t)$, $u_2(t)$, …, each one subject to an initial condition $u_j(t = 0) = F(x_j)$. We can write the system in matrix form as

$$\frac{d\mathbf{U}(t)}{dt} \;=\; \frac{D}{\Delta x^2}\mathsf{A}\mathbf{U}(t)\,, \tag{9.7.5}$$

in which

$$\mathbf{U}(t) \;=\; \left|\begin{array}{c} u_1(t) \\ u_2(t) \\ \ldots u_{j-1}(t) \\ u_j(t) \\ u_{j+1}(t) \\ \ldots \\ u_N(t) \end{array}\right|\,, \tag{9.7.6}$$

and A is the tri-diagonal matrix that we have encountered before:

$$\mathsf{A} \;=\; \left|\begin{array}{ccccccc} -2 & 1 & 0 & 0 & 0 & 0 & \ldots \\ 1 & -2 & 1 & 0 & 0 & 0 & \ldots \\ 0 & 1 & -2 & 1 & 0 & 0 & \ldots \\ 0 & 0 & 1 & -2 & 1 & 0 & \ldots \\ & & & \ldots & & & \\ 0 & 0 & 0 & 0 & \ldots & 1 & -2 \end{array}\right|\,. \tag{9.7.7}$$

By this procedure, we have reduced the partial differential equation (9.7.1) to a system of ordinary differential equations which can be integrated numerically by one of the methods introduced in this chapter. This is the so-called *method of lines*.

Chapter 9

Weighted Residuals Methods

8.1 Introduction

The first step in the numerical solution of differential (or integral) equations is always to transform the original equation into an algebraic system. The only methods to achieve this objective that we have seen so far have been based on the discretization of derivatives, but there are many others, for example a whole class of methods that go under the heading of *weighted residual methods*. Consider an equation that we write in symbolic form

$$\mathcal{N}(u) \,=\, F(x,t) \qquad a < x < b\,, \tag{8.1.1}$$

where $F(x,t)$ is prescribed and $\mathcal{N}(u)$ stands for a set of derivatives and/or other operations applied to the unknown function u ($\mathcal{N}$ is not necessarily linear). For definiteness we have shown a single space variable x but, more generally, there may be more than one spatial variable. We have also included a time variable which may or may not be present.

The general idea is to approximate the unknown function u by a linear combinations of known (prescribed) *trial functions* $\phi_j(x)$:[45]

$$u(x,t) \,\simeq\, \sum_{j=1}^{N} u_j(t)\phi_j(x)\,. \tag{8.1.2}$$

The difference

$$\mathcal{N}\left(\sum_{j=1}^{N} u_j(t)\phi_j(x)\right) - \mathcal{N}(u) = \mathcal{N}\left(\sum_{j=1}^{N} u_j(t)\phi_j(x)\right) - F(x,t)\,, \tag{8.1.3}$$

is the residual ("left-over") between the exact solution of the problem and the approximation (8.1.2). We now impose that the integral of this quantity weighted by a set of N prescribed *test functions* $\psi_k(x)$ vanish:[46]

$$\int_a^b \psi_k(x)\mathcal{N}\left(\sum_{j=1}^{N} u_j(t)\phi_j(x)\right) dx \,\simeq\, \int_a^b \psi_k(x)F(x,t)dx\,. \tag{8.1.4}$$

Upon taking in turn $k = 1, 2, \ldots N$ these integrals generate N equations from which the unknown coefficients u_j can be determined. This general idea can be implemented in many different ways.

8.2 The finite element method

One way, with which you are familiar, is the finite-element method. As a reminder, let us consider the very simple problem

$$\frac{d^2u}{dx^2} + u \,=\, F(x) \qquad 0 < x < 1 \tag{8.2.1}$$

with boundary conditions

$$u(x=0) \,=\, 0\,, \qquad \left.\frac{du}{dx}\right|_{x=1} \,=\, 0\,. \tag{8.2.2}$$

[45] In this example the coefficients u_j of the linear combination are functions of time. In other cases, they may be just constants or functions of other variables.

[46] If there is more than one spatial variable, the spatial integral may be on a multi-dimensional spatial domain.

We divide the interval $0 < x < 1$ into equal sub-intervals of length Δx and approximate the solution of the differential equation in the form (8.1.2) with the trial functions chosen as

$$\phi_j(x) \;=\; \begin{cases} 0 & x < x_{j-1} \\ \frac{x-x_{j-1}}{x_j-x_{j-1}} & x_{j-1} \le x \le x_j \\ \frac{x_{j+1}-x}{x_{j+1}-x_j} & x_j \le x \le x_{j+1} \\ 0 & x_{j+1} \le x \end{cases} \tag{8.2.3}$$

These are "tent functions" with the shape of triangles extending between the points x_{j-1} and x_{j+1}, where they vanish, and peaking at x_j, where they have the value 1. In some versions of the method polynomials of order higher than the first or other functions are used. The characteristic of the method that we want to emphasize is that all the functions of the family are non-zero only on a part of the domain. For this reason, for each k, only a few of the the integrals (8.1.4) are non-zero as we will soon see.

In the *Galerkin method* the test functions are taken to be the same as the trial functions. However, before proceeding by setting up the integrals as in (8.1.4), we need to avoid having to take the second derivative of u because, if we approximate u as in (8.1.2) with the previous trial functions, we cannot take second derivatives since the ϕ_j's cannot be differentiated twice. We can avoid the need for this step by recasting the problem in the *weak form*, which can be done by integrating by parts:

$$\int_0^1 \phi_k(x) \left(\frac{d^2u}{dx^2} + u - F \right) dx \;=\; \left[\phi_k \frac{du}{dx} \right]_0^1 + \int_0^1 \left[-\frac{d\phi_k}{dx}\frac{du}{dx} + \phi_k(u-F) \right] dx \;=\; 0\,. \tag{8.2.4}$$

At $x = 0$ all the ϕ_k vanish, including the first one and, at $x = 1$, even if one of the ϕ_k is non-zero, $du/dx = 0$. So the integrated term vanishes and now we can proceed as in (8.1.2) finding

$$\sum_j \left[\int_0^1 \left(-\frac{d\phi_k}{dx}\frac{d\phi_j}{dx} + \phi_k\phi_j \right) dx \right] u_j \;=\; \int_0^1 \phi_k F dx\,. \tag{8.2.5}$$

We now define a matrix A having elements

$$\mathsf{A}_{kj} \;=\; \int_0^1 \left(-\frac{d\phi_k}{dx}\frac{d\phi_j}{dx} + \phi_k\phi_j \right) dx\,, \tag{8.2.6}$$

and note that, due to the fact that the trial functions are non-zero only over parts of the domain, only a few of these integrals are non-zero. In terms of A, (8.2.5) becomes

$$\sum_j \mathsf{A}_{kj} u_j \;=\; \int_0^1 \phi_k F dx\,, \qquad k = 1, 2, 3, \ldots \tag{8.2.7}$$

i.e., a linear system which determines the u_j. It is interesting to note that, with the particular ϕ_j used here, the explicit form of this linear system is

$$\frac{u_{j-1} - 2u_j + u_{j+1}}{\Delta x^2} + \left(\frac{1}{6}u_{j-1} + \frac{2}{3}u_j + \frac{1}{6}u_{j+1} \right) \;=\; \int_0^1 \phi_j F dx\,, \tag{8.2.8}$$

and has therefore the familiar tri-diagonal form. The first term turns out to have exactly the same form as if we had used the centered difference formula to approximate the second derivative. This is of course a consequence of the specific functions ϕ_j chosen here and might not happen with a different set of trial/test functions. Had we used the finite-difference method, the second term in the left-hand side would just be u_j. Here we find instead the weighted average of u_j and $u_{j\pm1}$. By carrying out a Taylor series expansion we find

$$\begin{aligned} \frac{1}{6}u_{j-1} + \frac{2}{3}u_j + \frac{1}{6}u_{j+1} &= \frac{1}{6}\left(u_j - \Delta x\partial_x u + \frac{1}{2}\Delta x^2\partial_x^2 u + \ldots \right) + \frac{2}{3}u_j + \frac{1}{6}\left(u_j + \Delta x\partial_x u + \frac{1}{2}\Delta x^2\partial_x^2 u + \ldots \right) \\ &= u_j + \frac{1}{6}\Delta x^2\partial_x^2 u\,. \end{aligned} \tag{8.2.9}$$

The error is therefore of order Δx^2, which is the same as the error affecting the approximation of the second derivative. Hence, purely on this basis, we have no grounds to decide whether the finite-element method is more accurate than the finite difference method. This will depend on the specific problem and a choice between the two will probably be based on other considerations such ease of coding etc. Note that $\frac{1}{6} + \frac{2}{3} + \frac{1}{6} = 1$. If the three weights did not sum up to 1, the first term in the right-hand side of (8.2.9) would be a constant different from 1 and, in this case, it would be impossible to reduce the error by taking Δx smaller and smaller. The method would then be inconsistent with the equation to be solved.

8.3 Orthogonal functions

A family of functions $\phi_j(x)$ defined over some interval $a < x < b$ is said to be orthogonal when

$$\int_a^b \phi_k(x)\phi_j(x)s(x)dx \begin{cases} = 0 & k \neq j \\ \neq 0 & k = j \end{cases} \tag{8.3.1}$$

The function s, called the *weight function* is fixed for all functions of the family. There are many families of orthogonal functions which differ in the interval where the property is satisfied and in the weight function s.

- The trigonometric functions $1/\sqrt{2}$, $\cos k\pi x/L, \sin j\pi x/L$ considered over the interval $0 < x < L$ are all orthogonal to each other:

$$\int_0^L \sin\frac{j\pi x}{L}\,\sin\frac{k\pi x}{L}dx \,=\, \frac{L}{2}\delta_{jk}\,, \qquad \int_0^L \cos\frac{j\pi x}{L}\,\cos\frac{k\pi x}{L}dx \,=\, \frac{L}{2}\delta_{jk}\,, \qquad \int_0^L \left(\frac{1}{\sqrt{2}}\right)^2 dx \,=\, \frac{L}{2} \tag{8.3.2}$$

$$\int_0^L \sin\frac{j\pi x}{L}\,\cos\frac{k\pi x}{L}dx \,=\, 0\,, \qquad \int_0^L \frac{1}{\sqrt{2}}\,\sin\frac{j\pi x}{L}dx \,=\, 0\,, \qquad \int_0^L \frac{1}{\sqrt{2}}\,\cos\frac{j\pi x}{L}dx \,=\, 0 \tag{8.3.3}$$

- The functions $\phi_k \,=\, e^{ikx}$ satisfy a slightly modified version of (8.3.1) similar to the scalar product between complex vectors, namely

$$\int_0^{2\pi} \overline{e^{ikx}}e^{ijx}dx \,=\, \int_0^{2\pi} e^{i(j-k)x}dx \,=\, 2\pi\delta_{jk}\,, \tag{8.3.4}$$

  with the overline denoting the complex conjugate.

- The Legendre polynomials $P_k(x)$ satsify

$$\int_{-1}^1 P_j(x)P_k(x)dx \,=\, \frac{2}{2k+1}\delta_{jk} \tag{8.3.5}$$

  The first few are

$$P_0 \,=\, 1, \qquad P_1 \,=\, x\,, \qquad P_2 \,=\, \frac{1}{2}(3x^2-1) \tag{8.3.6}$$

  etc.

- The Chebyshev polynomials T_k satisfy

$$\int_{-1}^1 T_j(x)T_k(x)\frac{dx}{\sqrt{1-x^2}} \,=\, \frac{1}{2}(1+\delta_{k0})\delta_{jk} \tag{8.3.7}$$

  The first few are

$$T_0 \,=\, 1\,, \qquad T_1 \,=\, x\,, \qquad T_2 \,=\, 2x^2-1\,, \tag{8.3.8}$$

  and, in general

$$T_k(x) \,=\, \cos[k\cos^{-1}x] \tag{8.3.9}$$

A major difference with the finite-element functions is that these function are non-zero over the entire range, not just sub-intervals.

8.4 The spectral method

The spectral method consists in expanding the unknown function into a sum of orthogonal functions. While this method is less flexible than the finite-element method, when it can be used it has the advantage of a faster convergence, in the sense that the same error level can be achieved by using a smaller number of terms. We illustrate this method with some examples.

- EXAMPLE 1. Consider the one-dimensional wave equation

$$\frac{\partial u}{\partial t} + \frac{\partial u}{\partial x} \;=\; 0\,, \qquad 0 < x < 2\pi \tag{8.4.1}$$

with periodicity boundary conditions

$$u(x=0,t) \;=\; u(x=2\pi,t) \tag{8.4.2}$$

and an initial condition

$$u(x,0) \;=\; \sin(\pi\cos x)\,. \tag{8.4.3}$$

The exact solution is of course $u(x,t) \;=\; \sin[\pi\cos(x-t)]$. To solve the problem by the spectral method we set

$$u(x,t) \;\simeq\; \sum_{k=-N/2}^{N/2} u_k(t) e^{ikx}\,, \tag{8.4.4}$$

and substitute into the equation to find

$$\sum_{k=-N/2}^{N/2} \left(\frac{du_k}{dt} + iku_k\right) e^{ikx} \;\simeq\; 0\,. \tag{8.4.5}$$

Now, for $-\frac{N}{2} \le l \le \frac{N}{2}$, form the $N+1$ integrals

$$\int_0^{2\pi} e^{-ilx} \left[\sum_{k=-N/2}^{N/2} \left(\frac{du_k}{dt} + iku_k\right) e^{ikx} dx\right] \;\simeq\; 0\,. \tag{8.4.6}$$

By applying the orthogonality relation (8.3.4) we find the set of $N+1$ equations

$$\frac{du_l}{dt} + ilu_l \;=\; 0\,, \qquad l = -\frac{N}{2}, -\frac{N}{2}+1, \ldots, -1,\, 0,\, 1, \ldots,\, \frac{N}{2}-1,\, \frac{N}{2}\,. \tag{8.4.7}$$

In general the equations found in this way are too complicated to be solved analytically, and a numerical method (e.g., Runge-Kutta) is necessary. In this case however the solution is very simple:

$$u_l(t) \;=\; u_l(0)\,\exp(-ilt)\,. \tag{8.4.8}$$

To find $u_l(0)$ we use the same method applying it to (8.4.4) written for $t = 0$:

$$u(x,0) \;=\; \sin(\pi\cos x) \;=\; \sum_{l=-N/2}^{N/2} u_l(0) e^{ilx}\,. \tag{8.4.9}$$

Multiply by e^{-ikx} and integrate to find

$$2\pi u_k(0) \;=\; \int_0^{2\pi} e^{-ikx} \sin(\pi\cos x) dx\,. \tag{8.4.10}$$

N	spectral	2nd order FD	4th order FD
8	9.87×10^{-2}	1.11	0.962
16	2.55×10^{-4}	0.613	0.236
32	1.05×10^{-11}	0. 199	0.0267
64	6.22×10^{-13}	0.0542	1.85×10^{-3}

Table 8.4.1: N is the number of terms in the expansion and the number of points in the finite-difference (FD) schemes.

The integral can be calculated explicitly in this case and gives $u_k(0) = J_k(\pi)\sin(k\pi/2)$, where J_k is the Bessel function of the first kind of order k. In more complicated situations the integrations can be carried out numerically.

It will be observed that, in this case, the functions used for ϕ_k satisfy the boundary conditions exactly. This results in a very rapid decrease in the numerical error as shown in Table 8.4.1. How to handle more general boundary conditions?

- Example 2. Consider

$$\frac{\partial u}{\partial t} - D\frac{\partial^2 u}{\partial x^2} = 0\,, \qquad 0 < x < 1\,, \tag{8.4.11}$$

subject to

$$u(0,t) = 0\,, \qquad u(1,t) = 1\,, \qquad u(x,0) = f(x)\,. \tag{8.4.12}$$

To deal with the boundary conditions we set

$$u(x,t) = x + v(x,t)\,, \tag{8.4.13}$$

so that v to vanishes at both end points. Now the initial condition for v is

$$v(x,t=0) = f(x) - x\,. \tag{8.4.14}$$

and v satisfies the same equation as u:

$$\frac{\partial v}{\partial t} - D\frac{\partial^2 v}{\partial x^2} = 0\,, \qquad 0 < x < 1\,. \tag{8.4.15}$$

At this point we can set

$$v(x,t) \simeq \sum_{j=1}^{N} v_j(t)\sin j\pi t \tag{8.4.16}$$

which satisfies the boundary conditions. Upon substituting into the equation (8.4.15) we find

$$\sum_{j=1}^{N}\left[\frac{dv_j}{dt} + j^2\pi^2 D v_j\right]\sin j\pi x \simeq 0\,. \tag{8.4.17}$$

Multiply by $\sin k\pi x$ and integrate:

$$\sum_{j=1}^{N}\left(\int_0^1 \sin k\pi x\, \sin j\pi x\, dx\right)\left[\frac{dv_j}{dt} + j^2\pi^2 D v_j\right] \simeq 0\,. \tag{8.4.18}$$

Recalling (8.3.2) with $L = 1$ we find

$$\frac{dv_k}{dt} + k^2\pi^2 D v_k = 0\,, \tag{8.4.19}$$

the solution of which is

$$v_k(t) \;=\; v_k(0)e^{-k^2\pi^2 Dt}\,. \tag{8.4.20}$$

The initial condition is found from (8.4.14):

$$\int_0^1 \sin k\pi x[f(x)-x]dx \;=\; \sum_j \left(\int_0^1 \sin k\pi x\, \sin j\pi x\, dx\right) v_j \;=\; \frac{1}{2}v_k(0)\,. \tag{8.4.21}$$

- Example 3: Collocation or pseudo-spectral method. We again consider the diffusion equation over a different interval:

$$\frac{\partial u}{\partial t} - D\frac{\partial^2 u}{\partial x^2} \;=\; 0\,, \qquad -1 < x < 1\,, \tag{8.4.22}$$

subject to

$$u(x=-1,t) \;=\; u(x=1,t) \;=\; 0\,, \tag{8.4.23}$$

with a suitable initial condition. We expand u in a truncated series of Chebyshev polynomials:

$$u \;\simeq\; u^N \;=\; \sum_{j=0}^{N} u_j(t)T_j(x)\,, \tag{8.4.24}$$

but this time we choose the test functions differently:

$$\psi_k(x) \;=\; \delta(x-x_k)\,, \tag{8.4.25}$$

with δ the so-called delta function.[47] Now the integrals (8.1.4) are

$$\int_{-1}^{1} \delta(x-x_k)\left[\frac{\partial u^N}{\partial t} - D\frac{\partial^2 u^N}{\partial x^2}\right]dx \;=\; \left[\frac{\partial u^N}{\partial t} - D\frac{\partial^2 u^N}{\partial x^2}\right]_{x=x_k} \simeq 0\,. \tag{8.4.26}$$

Thus, the equation gets "collocated" at the points x_k. In this way we find the system of equations

$$\sum_{j=0^N} \left[T_j(x_k)\frac{du_j}{dt} - T''(x_k)u_j\right] \;\simeq\; 0\,, \qquad k = 0,1,2,\ldots,N\,, \tag{8.4.27}$$

which is a system of ordinary differential equations for the u_j. When the expansion is in terms of the Chebyshev polynomials as here, a good choice for the collocation points is $x_k \;=\; \cos \pi k/N$.

8.5 The τ method

A more general procedure to deal with boundary conditions is the τ method, which counts the boundary conditions as two of the equations determining the expansion coefficients. Again we consider an example based on the diffusion equation:

$$\frac{\partial u}{\partial t} - D\frac{\partial^2 u}{\partial x^2} \;=\; 0\,, \qquad 0 < x < 2\pi\,, \tag{8.4.28}$$

subject to

$$\left.\frac{\partial u}{\partial x}\right|_{x=0} \;=\; -2\,, \qquad u(2\pi,t) \;=\; 1\,. \tag{8.4.29}$$

[47] The key property of the delta function is that, for any function $f(x)$ defined at a point x_0, we have

$$\int_a^b f(x)\delta(x-x_0)dx = \begin{cases} f(x_0) & a < x_0 < b \\ 0 & x_0 < a \quad \text{or} \quad b < x_0 \end{cases}$$

There is no "normal" function that can satisfy this relation; δ is an important member of the class of *generalized functions*.

If we were to write $u = -2x + 1 + 4\pi + v(x,t)$, v would satisfy homogeneous conditions and we could proceed similarly to the second example of section 8.4,[48] but let us set instead

$$u(x,t) \simeq v_0(t) + \sum_{j=1}^{N} [u_j(t) \sin jx + v_j(t) \cos jx] \,. \tag{8.4.30}$$

Here we must allow for the presence of both sines an cosines as including only one family would violate one or the other of the boundary conditions. To satisfy the condition at $x = 0$ we must have

$$\sum_{j=1}^{N} j u_j(t) = -2 \,, \tag{8.4.31}$$

while, to satisfy the condition at $x = 2\pi$,

$$v_0(t) + \sum_{j=1}^{N} v_j(t) = 1 \,. \tag{8.4.32}$$

We substitute (8.4.30) into the diffusion equation (8.4.28) and find

$$\frac{dv_0}{dt} + \sum_{j=1}^{N} \left[\frac{du_j}{dt} \sin jx + \frac{dv_j}{dt} \cos jx \right] + D \sum_{j=1}^{N} j^2 [u_j(t) \sin jx + v_j(t) \cos jx] \simeq 0 \,. \tag{8.4.33}$$

We multiply by $\sin kx$ and integrate from 0 to 2π; upon using (8.3.2) we have

$$\frac{du_k}{dt} + Dk^2 u_k = 0 \,, \qquad k = 1, 2, \ldots, N-1 \,. \tag{8.4.34}$$

We multiply by $\cos kx$ and integrate from 0 to 2π. Again using one of the relations (8.3.2) we have

$$\frac{dv_k}{dt} + Dk^2 v_k = 0 \,, \qquad k = 1, 2, \ldots, N \,. \tag{8.4.35}$$

Equations (8.4.34) and (8.4.35) are $N - 1 + N = 2N - 1$ equations for the $2N + 1$ unknowns $u_1, \ldots u_N$ and $v_0, v_1, \ldots v_N$. We complete the set of these $2N - 1$ equations by adding (8.4.31) and (8.4.32). Now we have as many equations as we have unknowns and the problem can in principle be solved. In practice it is inconvenient to deal with a system the equations of which are in part differential equations and in part algebraic relations. Hence, as a practical matter, it may be preferable to differentiate (8.4.31) and (8.4.32):

$$\sum_{j=1}^{N} j \frac{du_j}{dt} = 0 \,, \qquad \frac{dv_0}{dt} + \sum_{j=1}^{N} \frac{dv_j}{dt} = 0 \,. \tag{8.4.36}$$

[48] Note however that, while the boundary conditions are homogeneous, they are different. In order to satisfy them we would use the expansion

$$v(x,t) = \sum_{j=1}^{N} u_j(t) \cos\left(j - \frac{1}{2}\right) x \,.$$

The cosinusoidal functions have been chosen so that they have a zero derivative at $x = 0$ and vanish at $x = \pi$; they are readily shown to be orthogonal to each other so that the previous method can be used.

Chapter 10

Verification and Validation

10.1 Introduction

You have written a new code. After debugging a bit it compiles and produces numbers. But is the code correct? Does it do what you wanted it to do?

- Does the code *solve the right equations*? i.e., is the mathematical model that is solved adequate to address the question at hand? An example: if the mathematical model assumes an incompressible fluid, is compressibility truly negligible? Addressing this question is the *validation* of the code.
- Does the code *solve the equations right*? i.e., are the numerical method and its implementation correct? Addressing this question is the *verification* of the code.

The first question has to do with the physical model implemented in the code. We will not concern ourselves with this question here focusing instead on the second one. The problems is serious. It is estimated that programmers make between 10 and 50 errors per 1000 lines of code. Careful checking can push this down to perhaps 0.5 per 1000. Google manages about 2×10^9 lines of code, Linux has 20.3×10^6, Windows 50×10^6, Android 12×10^6. How to get rid of the remaining bugs?

10.2 Verification

Even if a code compiles, it may contain a large variety of errors (bugs) that compilation cannot reveal. Examples are un-initialized or un-declared variables, incorrect number of parameters or parameter types passed on to a subroutine, mis-match of array dimensions and many others. Software now exists which can identify bugs that are overlooked by the compiler, but even these tools are not completely foolproof. For these reasons verification is an essential step in code development.

The process of verification is complex because a code can usually solve many variants of the same problem such as, for example, steady or transient cases, situations in which different parts of the mathematical model are active or important (e.g., radiation heat transfer may be negligible with certain parameter choices but not with others, certain geometries may "stress" part of the code more than others), and so on. It is usually wise to keep in mind an adaptation of a quote by Einstein: "No seemingly acceptable result can prove the code right, but a single unacceptable one can prove it wrong." Thus, verification has several facets:

- Grid independence: At the most basic level, before any result can be "believed," it must be proven to be insensitive to the time and spatial steps used to obtain it. In other words, the results obtained e.g. by halving the space step or the time step or both (of course, always respecting the stability conditions if any) must be "reasonably close" to the original ones. We have seen that, for excessively large steps, the numerical error is dominated by truncation errors (e.g., coarse discretization of derivatives), while for excessively small steps, it is dominated by round-off errors (i.e., due to the truncation of irrational or long rational numbers to the maximum length allowed by the computer). In between these two domains there is a "sweet spot" in which the numerical error is fairly insensitive to the time and/or space step used. For very large problems, the use a much smaller step might be impossible due to computer limitations or execution times. In such cases a possibility would be to use larger, rather than smaller steps. The problem is that, if the original result is close to the transition from acceptable to truncation-dominated, the difference found with the use or larger steps may be due to truncation

rather than to lack of grid convergence. Whenever possible, decreasing the steps (perhaps not so much as halving them) is the best strategy.

- There may be the temptation to compare results, for example, with an experiment or an independent code and to choose the discretization so as to achieve a match. For example, one may find that there is agreement with a certain discretization, but not if the discretization is refined. Accepting the discretization that gives a good match is a terrible idea for several reasons. In the first place, the comparison may just happen to be be favorable for the particular case(s) studied but, for different parameter values, different parts of the code may be activated or become important and we would have no basis to choose the discretization for these other cases. Secondly, but more importantly, the mere fact that the results change by changing the discretization is an indication of a serious problem. In the best of hypotheses, the change is due to truncation error. However it may very well be due to bugs or errors in the implementation of the original mathematical model which happen to compensate (or to become small for some other reason) for a specific discretization with specific parameter values, but may behave in a quite different way in other cases. For all these reasons, *grid independence is a cornerstone of computing* and must be adhered to whenever accuracy is important.

- KNOW THY ERROR! It may happen that, for some case for which information is available (e.g., experimental data), the results of the code are not as accurate as one would like, but are "accurate enough" for one's specific purposes. An understandable impulse would then be to decide to "live with" the inaccuracy, but this would be quite irresponsible: the error may be small in the cases that can be checked, but it may be substantial in other cases with different input parameters for which no information is available. To avoid unpleasant surprises it is necessary to *understand the origin of the error*. If it is due to a bug that has only a limited effect in the cases tested, but which might cause major problems in others, finding the bug and fixing the code would be a priority. In other cases, the error may be due, e.g., to excessively large time or space steps, and it may be impractical to use a finer discretization. Once this has been proven and understood, accepting the error might be justified. A related problem might arise if it is found that the results of the code look suspicious for some parameter values, but seem o.k. for other parameter values. In these cases one may think that it would be ok to live with the problem using the code staying away the zone of "bad" parameters. In fact, this would be a very dangerous policy until the cause of the code misbehavior is fully understood for the same reasons just described. An iterative method may be very slow to converge in some cases, but not in others. Just on the face of this observed behavior, it is impossible to tell if there is an error or the slow convergence is justified, and it would be imperative to give an answer to this question. To reiterate, while it may be o.k. to use a code that gives "good enough" results, it would be irresponsible to do it until the origin of the errors is fully understood. Sweeping them under the rug is a very dangerous tactics.

- Symptoms of problems with the code may be more subtle than an obviously impossible result such as the violation of basic physical laws. For example, the mathematical model may possess symmetries (e.g., a plane or an axis of symmetry) which should be reflected in the code results. Some violation of symmetry may be introduced by the discretization or by numerical error but, once again, it should not be dismissed until its origin is fully understood. Other useful tests involve conservation laws (e.g. is mass conserved? is the total momentum balanced?), Galilean invariance and others.

- It may happen that a grid fine enough to guarantee grid independence proves to be too expensive. If one needs to carry out many simulations this may be a significant problem and one may be forced to use a coarser grid than would be appropriate. This decision is less than optimal, but can be pursued provided it has been ascertained beyond reasonable doubt that the origin of the differences between the results obtained with two different grids is solely due to truncation error rather than some other cause.

- A more subtle symptom of problems with the code involves the order of accuracy of the algorithm.

Suppose, for example, that the discretization used has a formal spatial accuracy of second order. This implies that, halving the step, should decrease the error by a factor of 4, halving again by a factor of 16 and so on. If the code behavior does not conform with this expectation, there may well be a bug somewhere. Testing this feature may be subtle. In the first place, we need to define a suitable metric for the error. Consider a three-dimensional discretization of a spatial region and let $u(x_i, y_j, z_k)$ be the solution produced by the code at the grid nodes and $u_{ref}(x_i, y_j, z_k)$ be a "reference solution" (to be further specified below). The L_1 absolute error is defined as

$$\epsilon_1 = \| u - u_{ref} \|_{L_1} = \frac{1}{N_x N_y N_z} \sum_{ijk} |u(x_i, y_j, z_k) - u_{ref}(x_i, y_j, z_k)| , \tag{10.2.1}$$

where N_x, N_y and N_z are the number of nodes in the three directions. Another definition in use is the mean-square, or L_2 error:

$$\epsilon_2 = \| u - u_{ref} \|_{L_2} = \sqrt{\frac{1}{N_x N_y N_z} \sum_{ijk} |u(x_i, y_j, z_k) - u_{ref}(x_i, y_j, z_k)|^2} , \tag{10.2.2}$$

The previous definitions average over the errors and may therefore hide localized problems. A more sensitive error definition is the L_∞ error:

$$\epsilon_\infty = \| u - u_{ref} \|_{L_\infty} = \max_{ijk} |u(x_i, y_j, z_k) - u_{ref}(x_i, y_j, z_k)| . \tag{10.2.3}$$

Relative errors are defined by dividing the absolute errors by a suitable scale, such as the average of u_{ref} over the nodes, a quantity suggested by physical considerations or others.

If the discretization used has an order of accuracy N, then we would expect that, for some constant C, any one of these errors should satisfy a relation of the form

$$\epsilon \simeq C h^N , \tag{10.2.4}$$

so that the ratio of the errors given by two different discretizations with steps h_1 and h_2 would be expected to satisfy, approximately,

$$\frac{\epsilon_1}{\epsilon_2} \simeq \frac{h_1^N}{h_2}^N . \tag{10.2.5}$$

Upon taking the logarithm of (10.2.4) we find

$$\log \epsilon \simeq N \log h + \log C . \tag{10.2.6}$$

Here we use the $\simeq$ sign because, in principle, this expected scaling is asymptotic as $h \to 0$ and may or may not be exactly satisfied if h is not sufficiently small (for simplicity here we have assumed that the grid size is the same in all directions). A good procedure to be used to ascertain whether the expected order of accuracy is attained is to calculate the error (absolute or relative) for different grid sizes and to fit a straight line to its logarithm in dependence of the spatial step h. The slope of the line is the power of h with which the error decreases as h is reduced. The slope may not be the expected one if h is too large but, in the range of grid independence, one would expect to observe a slope close (if not exactly equal) to the order of accuracy of the discretization. Failing this, the code should be carefully analyzed. In addition to the errors of the dependent variables, one may want to study also the error of their derivatives.

It sometimes happens that different terms in the equation(s) are discretized with different orders of accuracy. For example, in a fluid mechanics code, convection may be first-order accurate and viscous terms second-order accurate. In this case the overall accuracy would be first order and the type of test described would not be able to spot an error in the viscous terms. To sidestep this problem it may be useful to switch off the terms one is not interested in testing retaining only the others. For example, to test the accuracy of the viscous terms alone one can remove the convection terms by commenting them out, or by multiplying them by a parameter that is set to zero for the test.

- Similar remarks can be made about errors associated to the temporal discretization. The more common situation in which both time and space are discretized is more complex. Often the spatial error can be investigated directly by studying the results of simulations of time-independent, steady state situations. After this step, the temporal error can be investigated. A possible procedure is the following.

  Suppose for example that the discretization is such that one would expect second-order accuracy in both space and time. The error would then be expected to have the form

  $$\epsilon \,=\, A(\Delta t)^2 + Bh^2\,, \tag{10.2.7}$$

  where A and B are positive constants (since they are the result of taking absolute values according to the previous definitions) related to derivatives of the quantities being calculated (which are the left-over terms in the Taylor series expansions approximating derivatives).[49] Carry out N simulations with the same h and decreasing time steps Δt_1, Δt_2, ..., Δt_N, with Δt_N small enough that the error is dominated by the spatial discretization. Let $\epsilon^{(k)}$ be the error associated to the use of Δt_k. Then

  $$\delta_k \,=\, \epsilon^{(k)} - \epsilon^{(N)} \,=\, A(\Delta t_k)^2 + Bh^2 - [A(\Delta t_N)^2 + Bh^2] \,\simeq\, A(\Delta t_k)^2\,, \tag{10.2.8}$$

  since Δt_N is so small that $A(\Delta t_N)^2 \ll Bh^2$. A plot of $\log \delta_k$ vs. Δt_k should then approximate a straight line with slope 2 if the temporal accuracy is indeed of second order.

  A more general, but more expensive, method is to carry out many simulations including a very fine one to be considered as a reference. The errors may be expected to have the form

  $$\epsilon \,=\, A(\Delta t)^\alpha + Bh^\beta + C(\Delta t)^\gamma h^\kappa\,. \tag{10.2.9}$$

  The errors associated to the available simulations can then be fitted to this expression to determine the 7 unknowns $A, B, C, \alpha, \beta, \gamma, \kappa$.

- Application of the previous analysis presupposes the availability of a suitable reference solution. Ideally, this would be an exact solution of the mathematical model, which may be a "manufactured solution" as explained below. Another possibility is to use for u_{ref} the solution found with the finest grid, assuming that this is a close approximation to the exact solution.

- Comparison with exact solutions. A very stringent comparison of the reliability of a code is the comparison with exact analytical solutions when available. Such a comparison is best carried out over a range of parameter values that provides a reasonable degree of what is referred to as *code coverage*, namely a test of as many components of the code as possible. In some cases analytic solutions are available only in special limits, e.g. when the non-linearities in the mathematical model solved by the code are very small (e.g, very small-amplitude oscillations). In such cases one may expect differences between the numerical and analytical results due to the fact that the simulation is not run close enough to the limit used for testing (e.g., the oscillations may be small but not so small that non-linear terms are completely negligible). Once again, it is necessary to understand the origin of these differences before declaring oneself satisfied that the code has passed the test.

- Code-to-code comparison. The relationship of code results to experiment falls under the heading of "validation" rather than verification.Usually a more stringent and more compelling verification method is to compare the code results with those of an independent code which can be adapted to solve the same or closely related problems. The power of this comparison lies in the possibility of comparing *actual numbers*, rather than trends or other superficial features. Looking at numbers is a much more reliable comparison than, for example, comparing color maps of the results of the two codes. A somewhat less reliable, but still useful, possibility is to compare the code results with graphs and tables found in the literature. In both cases, it could be that the other code or the published results are not exactly comparable because the problems solved, though similar, are not exactly equal. Once again, differences can be accepted, but only if their origin is clearly understood.

[49] More generally, the error may contain a contribution proportional to $h\Delta t$. Whether this happens or not can be ascertained by studying the truncation error.

- DEBUGGING: All the previous tests are directed at identifying the presence of errors in the code, but usually give little indication of where precisely the errors may lie. To find them one needs to embark into the crucial task of debugging the code, which is greatly facilitated by good programming practices. For example, it is recommended to build the code as a set of separate units, or subprograms, each one of which can and should be separately tested before being inserted into the code (*unit testing*). Combinations of these separate units, short of the complete code, can then be tested together (*component testing*) before the entire code is tested (*system testing*). It is also useful to build into the code parameters that can easily delete particular terms (e.g., the non-linear terms of the model so that the linear version of the problem can be tested with known linear solutions) or bypass certain sections of the calculation. These parameters should be easily accessible and grouped in an easy-to-find part of the code, such as near the beginning of the main program or in a separate subprogram. This strategy helps avoid the common error that, for example, when a code is set up for a test by setting certain parameters, they are not properly adjusted in all the places where they occur after the test has been completed.

  Unfortunately, however, ultimately debugging is an art. It consists in "asking" the code those crucial questions that reveal the bug. Physical intuition can be a powerful aid: under what conditions would this particular phenomenon be dominant? under what conditions would the interaction between two different phenomena result in some expected result? Running the code with parameter values that bring these phenomena into the forefront is a powerful way to find the errors.

10.3 The method of "manufactured solutions"

In most cases of interest, exact or approximate analytical solutions of the mathematical model implemented by the code are not available. In such cases the so-called "method of manufactured solutions" is useful. The idea is so simple that it hardly deserves a name: Any mathematical expression inserted into the original equations of the mathematical model will produce a "left-over piece" because, if this were not so, the mathematical expression would be an exact analytic solution of the equation(s). This left-over piece may be considered as a forcing term and the analytical expression that generates it would then be an exact solution of the original equation augmented by the forcing term.

Let us illustrate the idea with some simple examples. Consider the one-dimensional diffusion equation:

$$\frac{\partial u}{\partial t} - D\frac{\partial^2 u}{\partial x^2} \;=\; 0\,. \tag{10.3.1}$$

Set

$$v(x,t) \;=\; e^{t/\tau}\sin kx\,, \tag{10.3.2}$$

and substitute into the equation to find

$$\frac{\partial v}{\partial t} - D\frac{\partial^2 v}{\partial x^2} \;=\; \frac{e^{t/\tau}}{\tau}\sin kx + Dk^2 e^{t/\tau}\sin kx\,. \tag{10.3.3}$$

It follows that $v(x,t)$ as taken in (10.3.2) is an exact solution of this augmented equation subject to the initial condition $v(x,0) = \sin kx$. If the domain of interest extends for $a \le x \le b$, one would use the boundary conditions $v(a,t) \;=\; e^{t/\tau}\sin ka$ and $v(b,t) \;=\; e^{t/\tau}\sin kb$. The solution method for an equation of this form differs very little from that of the original equation because the numerical method is primarily determined by the differential terms in the equation rather than by the forcing terms. For example, a simple explicit discretization of (10.3.3) would be

$$\frac{u_j^{n+1} - u_j^n}{\Delta t} + D\frac{u_{j-1}^n - 2u_j^n + u_{j+1}^n}{\Delta x^2} \;=\; f(x_j, t^n)\,, \tag{10.3.4}$$

in which the function f represents the right-hand side of (10.3.3).

As another example consider the Poisson equation

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \;=\; \phi(x,y)\,, \tag{10.3.5}$$

subject to certain boundary conditions along the sides of the square $0 \le x \le L$, $0 \le y \le L$. Take $v(x,y) = x^m(L-x)^n + y^p(L-y)^q$ and substitute to find

$$\begin{aligned}\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2} \quad &= \quad \left[m(m-1)x^{m-2}(L-x)^n + n(n-1)x^m(L-x)^{n-2}\right] y^p(L-y)^q \\ &\quad + x^n(L-x)^m\left[p(p-1)y^{p-2}(L-y)^q + q(q-1)y^p(L-y)^{q-2}\right]\,. \end{aligned} \tag{10.3.6}$$

The right-hand side of this expression can be inserted in place of ϕ in the original equation (10.3.5) and the numerical solution compared with $v(x,t)$.

It is evident that, in principle, the trial functions that can be used do not have to represent real physics as the idea is simply to investigate whether the implementation of the mathematical model is correct. However, the trial functions can actually be chosen so as to test particularly demanding numerical aspects, such a spatial region with rapid variation, small parameters multiplying high-order derivatives and others.

In carrying out this process one will find differences between the exact and the numerical solutions. As always, it is of critical importance to understand the origin of these differences. This process is aided by an understanding of the numerical errors affecting the discretization method used. For example, we know that the explicit method applied to the diffusion equation introduces an artificial damping in the numerical solution. Thus, some damping is expected, but it is important to assure oneself that it is not more or less than what would be expected.

Chapter 11

Problems

11.1 Theoretical problems

11.1.1 Number representation and roundoff

1. What is normalized scientific notation? What are its consequences?

2. Apply a suitable rounding to 5 digits to the numbers (a) 32.5601; (b) 1.81996; (c) 121.365; (d) 8456.95; (e) -54.7675

3. Write the following numbers in scientific notation for a 16-bit computer (half-precision) using IEEE 754 standards:

   (a) 0.0010245

   (b) $100\pi = 314.15926535897932384626433$

4. Consider the linear system

$$\begin{vmatrix} 3\lambda+\mu & \sqrt{3}(\lambda-\mu) \\ \sqrt{3}(\lambda-\mu) & \lambda+3\mu \end{vmatrix} \begin{vmatrix} u_1 \\ u_2 \end{vmatrix} = \begin{vmatrix} 0.1 \\ 0.6225 \end{vmatrix},$$

   with $\lambda = 10$ and $\mu = 0.1$. The solution accurate to 15 digits is:

$$u_1 \;=\; -0.602762257146565\,, \qquad u_2 \;=\; 1.063907481281669\,.$$

   (a) Rewrite the coefficients of the linear system and the right-hand side in the normalized scientific notation;

   (b) Round the coefficients and the right-hand side to 3 significant digits.

11.1.2 Approximation of derivatives

1. Determine the values of the parameters α, β, γ such that

$$\frac{\alpha f'(x+h) + \beta f'(x) + \gamma f'(x-h)}{h} \;\simeq\; f''(x)$$

   Determine the order of magnitude of the error as a power of the increment h.

2. We want to approximate the third derivative of a function $f(x)$ by using values of f at x itself and at nearby points. What is the minimum number of points that is needed? Explain the argument leading to your answer.

3. If the Taylor series expansion for a function $f(x)$ is

$$f(x+h) \;=\; f(x) + hf'(x) + \frac{1}{2}h^2 f''(x) + \frac{1}{3!}h^3 f''' + \ldots$$

   what is the Taylor series expansion for $f'(x+h) = [df/dx]_{x+h}$?

4. Determine the values of the parameters α, β, γ such that

$$\alpha f'(x+h) + \beta f'(x) \simeq \frac{f(x) + \gamma f(x+2h)}{h} .$$

What is the minimum possible error affecting this relation expressed as a power of the increment h?

5. (i) Explain *how* you would determine the values of the constants a, b, c, d, e in such a way that the following relation holds:

$$a \left[\frac{du}{dx}\right]_{x-h} + \left[\frac{du}{dx}\right]_{x} + b \left[\frac{du}{dx}\right]_{x+h} = \frac{1}{h} \left[cu(x-h) - du(x) + eu(x+h)\right]$$

(ii) What is the maximum order of accuracy that a relation of this type can achieve? (iii) Knowing that $a = b$ and $c + e = 0$, determine all the constants.

6. Consider the relation

$$A \left.\frac{du}{dx}\right|_{x-h} + B \left.\frac{du}{dx}\right|_{x} + C \left.\frac{du}{dx}\right|_{x+h} = \frac{1}{h} \left[Pu(x-h) + Qu(x) + Ru(x+h)\right] .$$

(a) Without doing any calculation, explain how you would determine the constants A, B, C, P, Q, R in such a way that this is a valid relation (up to some order in h);
(b) Without doing any explicit calculation, what is the best accuracy (i.e., the highest possible power of h) at which this relation can be valid?

7. How would you implement the Neumann condition

$$\left.\frac{\partial u}{\partial x}\right|_{x=0} = 1$$

in the finite-difference discretization of a problem in which the variable x ranges between 0 and 1?

8. Find the most accurate possible approximation to d^2u/dx^2 using the values of u at x, $x - h_1$ and $x + h_2$ with $h_1 \neq h_2$.

9. Differentiation formulae in the case of two space variables can also be defined by introducing an interaction between the two space directions, for instance through a semi-implicit form on one of the two space coordinates. A representative example for a first x-derivative is a weighted average of the central difference formulas on the lines $j-1$, j, $j+1$ as

$$\left(\frac{\partial u}{\partial x}\right)_{ij} = \frac{1}{6} \left[\frac{u_{i+1,j+1} - u_{i-1,j+1}}{\Delta x} + 4\frac{u_{i+1,j} - u_{i-1,j}}{\Delta x} + \frac{u_{i+1,j-1} - u_{i-1,j-1}}{\Delta x}\right] .$$

Show that the error is of order Δx^2.

10. Show that

$$\frac{1}{4\Delta^2} \left[u_{i+1,j+1} + u_{i+1,j-1} + u_{i-1,j-1} + u_{i-1,j+1} - 4u_{ij}\right] ,$$

approximates the two-dimensional Laplace operator with an error of order Δ^2 when $\Delta x = \Delta y = \Delta$. Show that the leading terms of the error are

$$\frac{\Delta^2}{12} \left(\frac{\partial^4 u}{\partial x^4} + \frac{\partial^4 u}{\partial y^4}\right) + \frac{\Delta^2}{2} \frac{\partial^4 u}{\partial x^2 \partial y^2} .$$

11. Develop a 9-point approximation to the Laplace operator

12. Show that
$$\frac{1}{4\Delta x\Delta y}\left[u_{i+1,j+1}-u_{i-1,j+1}-u_{i+1,j-1}+u_{i-1,j-1}\right](*)$$
approximates $\partial^2 u/\partial x\partial y$ with an error of order $\Delta x^2, \Delta y^2$.

13. Show that
$$\frac{1}{2\Delta x\Delta y}\left[u_{i+1,j+1}-u_{i-1,j+1}-u_{i+1,j}+u_{i-1,j}\right]$$
approximates $\partial^2 u/\partial x\partial y$ with an error of order $\Delta x^2, \Delta y$.

14. Similar formulas can be obtained by applying a backward difference in y, instead, or by changing the roles of x and y, leading to a formula which is second order in Δy and first order in Δx

15. Show that
$$\frac{1}{\Delta x\Delta y}\left[u_{i+1,j+1}-u_{i,j+1}-u_{i+1,j}+u_{i,j}\right]$$
approximates $\partial^2 u/\partial x\partial y$ with an error of order $\Delta x, \Delta y$.

16. Same with backward differences

17. By averaging the two 2nd order formulae for the mixed derivative
$$\frac{1}{2\Delta x\Delta y}\left[u_{i+1,j+1}-u_{i,j+1}-u_{i+1,j}+u_{i-1,j-1}-u_{i,j-1}-u_{i-1,j}+2u_{ij}\right]$$
approximates $\partial^2 u/\partial x\partial y$ with an error of 2nd order

18. An alternative to the last is
$$\frac{1}{2\Delta x\Delta y}\left[u_{i+1,j}-u_{i+1,j-1}+u_{i,j+1}+u_{i,j-1}-u_{i-1,j+1}+u_{i-1,j}-2u_{ij}\right]$$
By adding the last two, back to (*)

19. Another popular choice for a finite volume discretization consists in selecting the mesh points as the cell 'faces', which are then labeled at half-integer index values ($i \pm 1/2$), and the function values are defined at the centers of the cells, typical of finite volume approaches

20. 4th order derivative
$$\frac{1}{\Delta x^4}\left(u_{i+2}-4u_{i+1}+6u_i-4u_{i-1}+u_{i-2}\right) = \frac{\partial^4 u}{\partial x^4}-\frac{\Delta x^2}{6}\frac{\partial^6 u}{\partial x^6}$$

21. The Gear scheme uses
$$\frac{du}{dx} \simeq \frac{3u_{j+1}-4u_j+u_{j-1}}{2\Delta x}$$
(acually for du/dt not du/dx)

22. Is $2f(x_j)-f(x_{j-1})$ an approximation to $f(x_{j+1}$? and, if so, what is the order of the error? [Here $x_{j\pm1}=x_j\pm\Delta x$.]

23. (a) What is the maximum accuracy in principle achievable for the numerical approximation of $f'(x)$ using the five points x, $x\pm h_1$ and $x\pm h_2$ (with $h_1\neq h_2$)? (b) Find constants α, β and γ such that the equality
$$f'(x) \simeq \frac{\alpha[f(x+h)-f(x-h)]+\beta f(x)+\gamma[f(x+2h)-f(x-2h)]}{h}$$
holds with the maximum possible accuracy. Verify that the order of the error agrees with the estimate in part (a).

11.1.3 Linear Systems

1. Consider the linear system

$$\begin{vmatrix} a & -a \\ \pm c & c+\epsilon \end{vmatrix} \begin{vmatrix} u_1 \\ u_2 \end{vmatrix} = \begin{vmatrix} f_1 \\ f_2 \end{vmatrix},$$

with a, c, ϵ, f_1, f_2 given, $c > 0$, $|\epsilon|$ small and $af_2 + cf_1 \neq 0$. (a) Under what conditions is the matrix diagonally dominant? (b) Solve the system by the tri-diagonal method and show that difficulties can – but do not necessarily – arise when diagonal dominance is lost.

2. Can the tri-diagonal method be applied to a linear system having a matrix A of the form

$$\mathsf{A} = \begin{vmatrix} a_1 & c_1 & d_1 & 0 \\ b_2 & a_2 & c_2 & d_2 \\ 0 & b_3 & a_3 & c_3 \\ 0 & 0 & b_4 & a_4 \end{vmatrix}$$

If not by the tri-diagonal method, what numerical method would you use to solve a linear system having a matrix of this form?

3. Consider a linear system having a matrix of the form

$$\begin{vmatrix} a_1 & a_2 & a_3 & 0 \\ 0 & b_1 & b_2 & b_3 \\ 0 & 0 & c_1 & c_2 \\ 0 & 0 & 0 & d_1 \end{vmatrix}$$

Can this system be solved by the tri-diagonal algorithm? If not, how would you solve the problem numerically?

4. A linear system of three equations in three unknowns has a matrix of the form

$$\begin{vmatrix} a & 0 & 0 \\ b & 0 & c \\ d & e & f \end{vmatrix}$$

(i) Can this system be solved by the tri-diagonal algorithm if a, b, c, d, e, f are all non-zero? (ii) Is the matrix diagonally dominant? (iii) Can the system be solved by the tri-diagonal algorithm if $d = 0$ while a, b, c, e, f are non-zero?

5. Consider the linear system

$$(1-\lambda)u_1 + 2u_2 = f_1\,, \qquad u_1 + (2-\lambda)u_2 = f_2\,.$$

Determine the value(s) of λ that would make it impossible for the system to be solved by Gauss elimination (or any other method, for that matter, but focus on Gauss elimination) for general values of f_1 and f_2. When this condition is satisfied, however, there is a special relation between f_1 and f_2 which makes the system solvable. Determine it. Is the solution unique in this case? Can you give an interpretation of your results, i.e., explain why there is a potential difficulty with this system etc.?

6. If, in principle, the Gauss-Seidel method requires an infinity of iterations to produce the exact solution of a linear system, why is it used in place of other methods (e.g., Gauss elimination) that can produce exact solutions without iterations?

7. What would be an efficient numerical method to solve a linear algebraic system having a matrix of the form

$$\left|\begin{array}{ccccc} a_{11} & a_{12} & 0 & 0 & 0 \\ a_{21} & a_{22} & a_{21} & 0 & 0 \\ 0 & a_{32} & a_{33} & a_{34} & 0 \\ 0 & 0 & a_{43} & a_{44} & a_{45} \\ 0 & 0 & 0 & a_{54} & a_{55} \end{array}\right| \; ?$$

What other method could you use, and why is the one that you mentioned answering the previous question preferable?

11.1.4 Elliptic equations

1. (*i*) Describe how you would discretize the problem

$$\frac{d^2u}{dx^2} + 2a(x)\frac{du}{dx} + b(x)u \;=\; f(x)\,,$$

over the interval $0 \le x \le L$, with boundary conditions $u(x=0) = u_0$, $[du/dx]_{x=L} = V_L$. The functions $a(x)$, $b(x)$ and $f(x)$ are known. Make sure that you use a second-order accurate discretization. (*ii*) What would be a good method to calculate numerically the solution of the linear system generated by your discretization? (*iii*) If you had a code that implements the method you propose, how would you check numerically that its accuracy is indeed second-order? (*iv*) Show how you would apply the method of manufactured solutions to verify your code.

2. (*a*) Discretize the equation

$$\frac{du}{dx} + f(x)u(x) \;=\; 0$$

over the interval $0 \le x \le L$ by using N equi-spaced nodes with an error smaller than $O(h)$ (where $h = L/(N+1)$) at every point. The boundary condition is $du/dx|_{x=0} = V_0$. (Note that, since this is a first-order equation, you can impose only one boundary condition; this circumstance requires discretizing the derivative at the last node differently from the way it is discretized at the interior nodes.)
(*b*) Set up the resulting linear system in matrix form.
(*c*) Show how you would apply the Gauss-Seidel iterative method to solve it.

3. In order to solve the problem

$$\frac{d^2u}{dx^2} - \Lambda u(x) \;=\; 0\,,$$

(with Λ a given constant) in the range $0 \le x \le L$, subject to the conditions $u(x=0) = U_0$, $du/dx|_{x=L} = V_L$, the interval $0 \le x \le L$ is discretized using N equally spaced points with $x_1 = h$, $x_2 = 2h$, $\ldots$, $x_{N+1} = (N+1)h = L$. Write down the equation for the first node, the generic node, and the last node if an accuracy of order h^2 is required at each node.

4. In the square domain $0 < x, y < L$ consider the two-dimensional Poisson equation

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = f(x,y)\,,$$

subject to the boundary conditions $u(x=0,y) = U_0(y)$, $u(x=L,y) = U_L(y)$, $u(x,y=0) = V_0(x)$, $u(x,y=L) = V_L(x)$. For simplicity use only 4 internal nodes with $\Delta x = \Delta y = \frac{1}{3}L$. Discretize the equation and show how you would apply the method of alternating directions to solve the problem numerically.

11.1.5 Diffusion equation

1. What do the symbols u_{jk} and u_j^n that we frequently use mean?

2. Apply the von Neumann stability analysis method to the Richardson discretization of the one-dimensional diffusion equation and prove that it is unconditionally unstable. The Richardson discretization is a three-level scheme. You can assume that

$$\Gamma \;=\; \frac{G(t+\Delta t)}{G(t)} \;=\; \frac{G(t)}{G(t-\Delta t)}\,.$$

3. Consider the following discretization proposed for the one-dimensional diffusion equation:

$$\frac{u_j^{n+1}-u_j^n}{\Delta t} \;=\; \frac{D}{2}\left(\frac{u_{j-1}^n-2u_j^n+u_{j+1}^n}{\Delta x^2}+\frac{u_{j-1}^{n-1}-2u_j^{n-1}+u_{j+1}^{n-1}}{\Delta x^2}\right).$$

   Use the von Neumann method to determine whether this proposed discretization is stable for suitable conditions imposed on Δt and Δx. Assume that $\Gamma \;=\; G(t^{n+1})/G(t^n) \;=\; G(t^n)/G(t^{n-1})$.

4. Develop a finite-difference discretization suitable for the solution of the problem

$$\frac{\partial u}{\partial t} \;=\; D\frac{\partial^2 u}{\partial x^2}+v(t)\,, \qquad \frac{dv}{dt} \;=\; -\lambda v+f(t)$$

   in which λ and $f(t)$ are given. The initial conditions are

$$u(x,t=0) \;=\; F(x)\,, \qquad v(0) \;=\; V\,,$$

   with $F(x)$ and V given. The boundary conditions are

$$u(x=0,t) \;=\; G(t)\,, \qquad \left.\frac{\partial u}{\partial_x}\right|_{x=L} \;=\; 0\,.$$

5. Is the implicit discretization of the one-dimensional diffusion equation affected by a dispersion error?

6. Consider the following possible discretization for the one-dimensional diffusion equation:

$$\frac{u_j^{n+1}-u_j^n}{\Delta t} \;=\; D\left[(1-\alpha)\frac{u_{j-1}^n-2u_j^n+u_{j+1}^n}{\Delta x^2}+\alpha\frac{u_{j-1}^{n+1}-2u_j^{n+1}+u_{j+1}^{n+1}}{\Delta x^2}\right],$$

   with $0 \le \alpha \le 1$. Use the von Neumann method to determine the stability features of this discretization. Check your results with the special cases $\alpha = 0$, $\frac{1}{2}$ and 1 for which you know the answer.

7. It is stated in the notes that the Crank-Nicolson method for the one-dimensional diffusion equation:

$$\frac{u_j^{n+1}-u_j^n}{\Delta t} \;=\; \frac{D}{2}\left(\frac{u_{j-1}^n-2u_j^n+u_{j+1}^n}{\Delta x^2}+\frac{u_{j-1}^{n+1}-2u_j^{n+1}+u_{j+1}^{n+1}}{\Delta x^2}\right)$$

   is unconditionally stable. Prove this fact.

8. The formula for the explicit discretization of the diffusion equation is

$$\frac{u_j^{n+1}-u_j^n}{\Delta t} \;=\; D\frac{u_{j-1}^n-2u_j^n+u_{j+1}^n}{\Delta x^2}\,.$$

If, for fixed n, this is applied increasing j in sequential order j, when it comes to the j-node the $(j-1)$-node has already been updated. Following an idea similar to the Gauss-Seidel modification of the Jacobi method one might propose to use the updated value as soon as it is available, which would amount to using the following formula:

$$\frac{u_j^{n+1} - u_j^n}{\Delta t} \;=\; D\frac{u_{j-1}^{n+1} - 2u_j^n + u_{j+1}^n}{\Delta x^2}\,.$$

Is this discretization consistent with the original equation? Is it stable?

9. A modification of the Richardson scheme was introduced by DuFort & Frankel:

$$\frac{u_j^{n+1} - u_j^{n-1}}{2\Delta t} \;=\; D\frac{u_{j-1}^{n} - \frac{1}{2}(u_j^{n-1} + u_j^{n+1}) + u_{j+1}^n}{\Delta x^2}\,.$$

This differs from the Richardson scheme by the substitution of u_j^n in the right-hand side by $\frac{1}{2}\left(u_j^{n-1} + u_j^{n+1}\right)$. ($a$) Determine the truncatuon error. (b) Study the stability of this scheme.

10. The Crank-Nicolson method applied to the diffusion equation in one space dimension

$$\frac{\partial u}{\partial t} \;=\; D\frac{\partial^2 u}{\partial x^2}\,,$$

requires the solution of a tri-diagonal system. One may hope that the use of a method of the form

$$\frac{u_j^{n+1} - u_j^n}{\Delta t} \;=\; \frac{1}{2}D\left[\frac{u_{j-1}^{n} - 2u_j^n + u_{j+1}^n}{\Delta x^2} + \frac{u_{j-1}^{n-1} - 2u_j^{n-1} + u_{j+1}^{n-1}}{\Delta x^2}\right]\,,$$

might give the same advatanges in terms of (i) stability, and (ii) higher-order error without the need for a tri-diagonal system. Does it? [Hint: to study stability use the von Neumann method.]

11.1.6 Wave equation

1. Consider the first-order wave equation

$$\frac{\partial u}{\partial t} + c\frac{\partial u}{\partial x} \;=\; 0\,,$$

subject to the initial condition

$$u(x, t=0) \;=\; \begin{cases} 0 & -\infty < x \le -1 \\ 1 - x^2 & -1 \le x \le 1 \\ 0 & 1 \le x < \infty \end{cases}$$

Sketch the initial condition and the solution of the equation at $t = 1$ and at $t = 2$ if $c = 1$.

2. By finding the corresponding effective equation, determine whether the following proposed discretization of the first-order wave equation of the previous problem

$$\frac{u_j^{n+1} - u_j^n}{\Delta t} + c\frac{u_j^{n+1} - u_{j-1}^n}{\Delta x} \;=\; 0$$

is consistent with the original differential equation.

3. What is the exact solution of the equation

$$\frac{\partial u}{\partial t} + c\frac{\partial u}{\partial x} \;=\; 0 \qquad -\infty < x < \infty$$

subject to the initial condition $u(x, 0) = \exp(-|x|)$?

4. Use the method of effective equations to investigate whether the discretization

$$\frac{u_j^{n+1} - u_j^n}{\Delta t} + c\frac{u_{j+1}^n - u_{j-1}^n}{2\Delta x} \;=\; 0$$

is a suitable method for the solution of the first-order wave equation.

5. A way to solve the second-order wave equation is to break it up into two first-order equations resulting in

$$\frac{u_j^{n+1} - u_j^n}{\Delta t} + c\frac{u_j^n - u_{j-1}^n}{\Delta x} \;=\; 0\,, \qquad \frac{v_j^{n+1} - v_j^n}{\Delta t} - c\frac{v_{j+1}^n - v_j^n}{\Delta x} \;=\; u_j^n\,.$$

Study the stability properties of this scheme in the following way:

   (a) Let $u_j^n = G^n e^{ikj}$, $u_j^{n+1} = G^{n+1}e^{ikj}$ etc., and $v_j^n = H^n e^{ikj}$, $v_j^{n+1} = H^{n+1}e^{ikj}$ etc. with k an arbitrary real number.

   (b) Express G^n in terms of H^{n+1} and H^n from the second equation.

   (c) Substitute the result into the first equation finding a relation involving H^{n+2}, H^{n+1} and H^n. Let $H^{n+1} = \Lambda H^n$, $H^{n+2} = \Lambda H^{n+1}$, solve for Λ and find the condition(s) which ensure that $|\Lambda| < 1$.

   (d) Use this result to calculate $|G^{n+1}/G^n|$ and see whether imposing that this be less than 1 requires additional condition(s).

6. We have seen that a possible approach for the solution of the wave equation is

$$\frac{v_j^{n+1} - 2v_j^n + v_j^{n-1}}{\Delta t^2} \;=\; c^2\frac{v_{j-1}^n - 2v_j^n + v_{j+1}^n}{\Delta x^2}\,.$$

Let $v_j^n = H^n e^{ikj}$, $v_j^{n+1} = H^{n+1}e^{ikj}$ etc., substitute, assume $H^{n+1} = \Lambda H^n$, $H^n = \Lambda H^{n-1}$, solve for Λ and find the condition(s) that ensure that $|\Lambda| < 1$

7. Study the stability properties of the so-called *leap-frog scheme* for the one-dimensional wave equation

$$\frac{u_j^{n+1} - u_j^{n-1}}{2\Delta t} + c\frac{u_{j+1}^n - u_{j-1}^n}{2\Delta x}\,.$$

8. Study the stability properties of the so-called *Lax-Friedrichs scheme* for the one-dimensional wave equation

$$u_j^{n+1} \;=\; \frac{1}{2}\left(u_{j+1}^n + u_{j-1}^n\right) - \frac{c\Delta t}{2\Delta x}\left(u_{j+1}^n - u_{j-1}^n\right)\,.$$

9. Study the stability of the Lax-Wendroff scheme by the von Neumann method.

10. In order to solve the first-order linear wave equation

$$\frac{\partial u}{\partial t} + c\frac{\partial u}{\partial x} \;=\; 0\,,$$

inspired by the Crank-Nicolson method, one may propose the following discretization:

$$\frac{u_j^{n+1} - u_j^n}{\Delta t} + \frac{c}{2}\left(\frac{u_j^{n+1} - u_{j-1}^{n+1}}{\Delta x} + \frac{u_j^n - u_{j-1}^n}{\Delta x}\right) \;=\; 0\,.$$

(*i*) Is this method stable? (*ii*) Is this discretization consistent with the equation? (*iiI*) What is the order of the round-off error? [Hint: Recall that the modulus squared $|A/B|^2$ of the ratio of two complex numbers A and B is $AA^*/(BB^*)$, where the asterisk denotes the complex conjugate.]

11.1.7 Ordinary differential equations

1. Write down the first three steps for the integration of the differential equation

$$\frac{du}{dx} = x^2 \qquad u(0) = 0$$

by the midpoint rule. Use a constant step h.

2. How would you proceed to integrate numerically the ordinary differential equation

$$\frac{d^3u}{dt^3} + a(t)\frac{du}{dt} + b(t)u = 0\,,$$

with suitable initial conditions at $t = 0$?

11.1.8 Verification

1. What is the so-called "Method of manufactured solutions"? What is it used for? Why?

2. The L_1 norm of the error for the solution of a problem using a discretization with a certain spatial step h is 2.1×10^{-2}, using a spatial step $h/2$ is 5.5×10^{-3} and using a spatial step $h/4$ is 1.4×10^{-3}. How would you determine the approximate order of accuracy of the scheme? Calculate it.

3. How would you apply the method of manufactured solutions to verify the correctness of a code intended to solve the non-homogeneous first-order equation

$$\frac{\partial u}{\partial t} + c\frac{\partial u}{\partial x} = f(x,t)\,,$$

subject to $u(x, t = 0) = F(x)$, $u(x = 0, t) = G(t)$, with $f(x,t)$, $F(x)$ and $G(t)$ given?

11.2 Computational problems

11.2.1 Finite-precision arithmetic

1. It is known that, for large n,

$$S(n) = 1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \ldots \frac{1}{n} \simeq \log n + \gamma$$

where $\gamma = 0.577215665\ldots$ is Euler's constant. This implies that the series of which $S(n)$ is the partial sum is in fact divergent. Do you think that you show this by a direct calculation of $S(n)$ for larger ands larger n?

Write a computer code to evaluate $S(n)$ numerically and calculate the difference $(\log n + \gamma)/S(n) - 1$. Does this difference go to 0 as n increases as it should? Why not? Do the calculation using a different number of significant digits starting with a small number, e.g., 3, and then increasing it. If you use MATLAB you can achieve this objective as follows:

```
digits(d) sets the precision used by vpa command to 'd' significant digits.
example:
>> digits(3)
>> pi3 = vpa(pi)
pi3 =
3.14
```

For C use the following instructions:

```
float a; // a must have more than 3 decimal places for this to work
a = roundf(a*1000)/1000; // if you want the number to have 3 decimal places
```

If you want four significant places replace 1000 by 10000 etc.

Do not be surprised if you end up using millions of terms.

2. Use the first-order fomula for the derivative

$$f'_{approx}(x;h) \simeq \frac{f(x+h)-f(x)}{h}$$

to calculate as a function of h the error

$$\epsilon(h) = \left|\frac{f'_{exact}(x) - f'_{approx}(x;h)}{f'_{exact}(x)}\right|$$

for the function $f(x) = \cos\alpha x$, for $x = \frac{\pi}{4}$ and $\alpha = 1$, 10 and 100. Show that the error decreases proportionally to h once h is sufficiently small. You can submit tables and/or graphs. What do you deduce by comparing results for different values of α? Warning: α =100 is hard!
A good way to do the problem is the following:

(a) Establish a range for the variable $v = \log 1/h$, for example $-1 \leq v \leq 20$. To each value of v corresponds a value of $h = \exp(-v)$. Thus, the error that we may expect for $v = -1$, i.e., $h = e$ should be large and then decrease;

(b) Divide the previous v range into N intervals with endpoints $v_0, v_1, \ldots, v_N$, with corresponding $h_0, h_1, \ldots, h_N$;

(c) For each one of these values of h evaluate $f(x+h)$ [and $f(x-h)$ for problem 2]; calculate the error

$$\epsilon(h) = \left|\frac{f'_{exact} - f'_{approx}}{f'_{exact}}\right|$$

Graph $\log\epsilon$ vs. $\log 1/h$

11.2.2 Approximation of derivatives

1. The derivative of a function $f(x)$ can be approximated in a number of different ways:

- Forward difference (1st order accuracy)

$$f'(x) \simeq \frac{f(x+h)-f(x)}{h}$$

- Backward difference (1st oder)

$$f'(x) \simeq \frac{f(x)-f(x-h)}{h}$$

- Centered difference (2nd order)

$$f'(x) \simeq \frac{f(x+h)-f(x-h)}{2h}$$

- One-sided forward difference (2nd order)

$$f'(x) \simeq \frac{4f(x+h)-3f(x)-f(x+2h)}{2h}$$

- One-sided backward difference (2nd order)

$$f'(x) \simeq \frac{3f(x) - 4f(x-h) + f(x-2h)}{2h}$$

For the second derivative we saw the centered difference formula

$$f''(x) \simeq \frac{f(x+h) - 2f(x) + f(x-h)}{h^2}$$

The point of this assignement is to see how these formulae work in practice in some cases. For this purpose consider the family of functions

$$f_k(x) = \sin kx\,, \qquad 0 \le x \le 1$$

where k is a positive integer. Divide the interval $0 \le x \le 1$ into N equal parts, so that the h to be used in the previous formuale is $h = 1/N$. In comparing the exact derivatives to their approximate values we will use the following definition of the error (this is known as the L_1 norm):

$$E'_k = \frac{1}{(N-1)k} \sum_{j=1}^{N-1} |(f'_k)_{exact}(x_j) - (f'_k)_{approx}(x_j)|$$

where $x_j = jh$ so that $x_1 = h$, $x_2 = 2h$, ..., $x_{N-1} = (N-1)h$; $(f'_k)_{exact}(x_j)$ is the exact analytic value of f'_k evaluated for $x = x_j$ and $(f'_k)_{approx}(x_j)$ is the approximate value given by the previous formulae.
[**Note:** A more standard (and better) way to define the error would be as

$$E'_k = \frac{1}{N-1} \sum_{j=1}^{N-1} \left| \frac{(f'_k)_{exact}(x_j) - (f'_k)_{approx}(x_j)}{(f'_k)_{exact}(x_j)} \right| .$$

We do not do this here as $(f'_k)_{exact}(x_j)$ may become zero. Instead, since $(f'_k)_{exact} = k \cos kx$, its order of magnitude is k and, for this reason, we divide by k rather than the exact value of f'_k.]
The error E''_k for the second derivative is defined in the same way:

$$E''_k = \frac{1}{(N-1)k^2} \sum_{j=1}^{N-1} |(f''_k)_{exact}(x_j) - (f''_k)_{approx}(x_j)| \ .$$

Note that you are not asked to evaluate the derivatives at the end points $x_0 = 0$ and $x_N = 1$. While using the one-sided formulae, you will need the values of f_k at the left of $x = 0$ and to the right of $x = 1$. For simplicity you may use the exact values for these two points.

Using MATLAB (or any other language of your choice) write a computer program to calculate the errors. Consider the following cases:

(a) Take $N = 10$ and calculate the error incurred by the preceding formulae for $k = 1$;

(b) Repeat by taking $k = 10$ and 20 with $N = 10$;

(c) Repeat by taking $k = 20$ and $N = 100$ and $N = 500$;

(d) Repeat by taking $k = 1$ and $N = 100$, 10^3, $10^4, 10^6$, 10^9; for this question consider only the first derivative f' calculated according to the 1st-order forward formula and to the 2nd-order centered difference formula.

Show your results for E'_k and E''_k in tabular form separating those of the 1st-order formulae from those of the 2nd-order formulae. Comment on the results: what happens to the error when you increase k keeping h constant? Why? How do the 1st-order errors compare with the 2nd-order ones? What happens when you go from $N = 100$ to $N = 500$? Why? In the last question, what happens as N is increased all the way to 10^9? Why? For the last question show also graphs of the error as a function of $|\log h|$.

It is important that you learn to attach words to computational results. Just "throwing numbers at us" does not cut it!

11.2.3 Elliptic equations

1. Discretize the interval $0 \le x \le L$ using N equally spaced internal points $x_1 = h$, $x_2 = 2h, \ldots, x_N = Nh$, with $h = L/(N+1)$; the end poins are $x_0 = 0$ and $x_{N+1} = L$. Write a code using the tri-diagonal algorithm to solve the Helmholtz equation

$$\frac{d^2u}{dx^2} - k^2 u(x) \;=\; f(x)\,,$$

in the range $0 \le x \le L$. Consider two cases in which the boundary conditions are:

   (a) Dirichlet type: $u(x = 0) = U_0$, $u(x = L) = 0$; the exact solution in this case is, for $f(x) = A$, a constant,

$$u(x) \;=\; \left(\frac{\sinh[k(L-x)] + \sinh(kx)}{\sinh(kL)} - 1\right)\frac{A}{k^2} + U_0\frac{\sinh[k(L-x)]}{\sinh(kL)}$$

   (b) Neumann type: $du/dx|_{x=0} = v$, $u(x = L) = 0$; the exact solution is, for $f(x) = A$, a constant,

$$u(x) \;=\; \left(\frac{\cosh(kx)}{\cosh(kL)} - 1\right)\frac{A}{k^2} - \frac{v}{k}\frac{\sinh[k(L-x)]}{\cosh(kL)}$$

Do not start with a very large number of nodes N. Start with a reasonable number (e.g. $N = 10$) and then increase it if necessary. A good way to see if more nodes are needed is to use a certain value of N, and then repeat the calculation with $2N$ nodes. If the results of the two calculations are different, it means that N nodes are too few. In this case you should compare the results for $2N$ and $4N$ and so forth until the results of two successive discretizations are about the same; this is called a *grid convergence study* and is an essential procedure for reliable computations. We have given you the exact solutions so that, *after* the grid convergence study, you can check whether your results are indeed good.

Additionally to have some idea whether the finite difference scheme has the desired order of accuracy, you can find the formal order of accuracy. To do that you have to calculate the maximum absolute errors in N and $2N$ nodes using the analytical solution. Then use following formula to find the formal order of accuracy.

$$\frac{\log\left(\frac{Error_N}{Error_{2N}}\right)}{\log(2)}$$

You should expect that, once the grid convergence criterion has been satisfied, the numerical solution and the exact solutions should more or less superpose within the thickness of the line. A significantly larger difference is a symptom of coding errors.

For numerical purposes use two values of k, namely $k = 1$ and $k = 10$ and, for each one of them, solve both problems 1 and 2. For all cases you can take $L = 1$, $U_0 = 1$, $v = 1$, $A = 1$.

Submit graphs comparing the exact and numerical solutions and also tables including some (only some!) numerical values to permit a better comparison between the two solutions.

All the code writing history should be logged using git version control and the *.git* folder should be submitted along with the code.

2. Solve a linear system $\mathsf{A}\mathbf{u} = \mathbf{f}$ of order $N \times N$ if the matrix A is tri-diagonal of the form

$$\mathsf{A} = \begin{vmatrix} 1 & -1 & 0 & 0 & 0 & \dots & 0 \\ -1 & 2 & -1 & 0 & 0 & \dots & 0 \\ 0 & -1 & 2 & -1 & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & 0 & -1 & 2 & -1 \\ 0 & 0 & \dots & \dots & \dots & -1 & 2 \end{vmatrix}$$

3. Use the Gauss elimination method to solve a linear system $\mathsf{A}\mathbf{u} = \mathbf{f}$ of order $N \times N$ if the matrix A is given by

$$\mathsf{A}^{-1} = \begin{vmatrix} N & N-1 & N-2 & \dots & 2 & 1 \\ N-1 & N-1 & N-2 & \dots & 2 & 1 \\ N-2 & N-2 & N-2 & \dots & 2 & 1 \\ \dots & \dots & \dots & \dots & \dots & \dots \\ 2 & 2 & 2 & \dots & 2 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{vmatrix}$$

4. Write a computer code based on the Gauss-Seidel method to solve the linear algebraic system

$$\mathsf{A}\mathbf{u} = \mathbf{f}$$

The matrix A is square $N \times N$; the diagonal elements are given by $a_{ii} = C/i$ ($i = 1, 2, 3, \dots, N$). The off-diagonal elements are given by $a_{ij} = 1/(i+j)$ ($i, j = 1, 2, 3, \dots, N$, $i \neq j$). The i-th element of $\mathbf{f}$ is $f_i = \sin[i\pi/(N+1)]$. Take $N = 10$ and solve the problem with $C = 4$ and $C = 0.2$. Then take $N = 100$ and try $C = 30$ and $C = 1$. (Remember the role of diagonal dominance!) A criterion to stop the iterations is the following: At each step form the ratio

$$\epsilon_i = \frac{\left| \sum_j a_{ij} u_j - f_i \right|}{\sum_j |a_{ij} u_j| + |f_i|} \,,$$

and calculate the maximum of all the ϵ_i's. Stop the iterations when this maximum is less than some tolerance criterion ϵ, i.e., when

$$\max_i \epsilon_i \leq \epsilon \,.$$

For ϵ you can choose 10^{-3}, 10^{-4} etc. depending on the degree of accuracy that you wish. You should turn in the code and a table showing the result of your Gauss-Seidel calculation alongside the exact solution (you can use the left division operator of MATLAB to calculate it) for $N = 10$. Describe in words what you find for $N = 100$ and report the number of iterations that you used in the attempt to meet the convergence criterion for each case.

5. Write a computer code to solve the two-dimensional Poisson equation

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = f(x, y) \,,$$

in the rectangle $0 \leq x \leq L_x$, $0 \leq y \leq L_y$ subject to Dirichlet boundary conditions (i.e., u prescribed) on all four sides:

$$u(x, y = 0) = u_0(x) \,, \quad u(x, y = L_y) = u_L(x) \,, \quad u(x = 0, y) = v_0(y) \,, \quad u(x = L_x, y) = v_L(y) \,.$$

In order to solve the linear algebraic system obtained from the discretization you can use e.g. the Gauss-Seidel or successive over-relaxation method.

In order to test your code you can take $L_x = L_y = \pi$,

$$f(x,y) \;=\; -2M \sin(Mx)\, \cosh(My)\;,$$

with M an integer, and also $u_0 = u_L = v_0 = v_L = 0$. The exact solution for this case is

$$u(x,y) \;=\; (L-y) \sin(Mx)\, \sinh(My)\,.$$

Calculate the mean L^1 error at all internal points of your discretization. Make sure that the grid that you use is adequate for a good solution of the problem by conducting a grid refinement study.

Submit graphs comparing the exact and numerical solutions, e.g. $u(x,y)$ as a function of x at a few values of y, for example $y = L_y/4$, $L_y/2$, $3L_y/4$, and as a function of y at a few values of x, e.g. at $x = L_x/4$, $L_x/2$, $3L_x/4$, and also tables including some (only some!) numerical values to permit a better comparison between the two solutions.

All the code writing history should be logged using `git` version control and the *.git* folder should be submitted along with the code.

Some pieces of advice:

- Try to keep your code as flexible as possible so that you can reuse it later for a different problem changing it as little as possible. For example, leave L_x, L_y, f and the boundary conditions unspecified in the body of the code – just give them a name. When you want to run a specific case (e.g., the test case mentioned above) you will assign specific values/functions to the names you used for L_x, L_y, f, u_0, u_L, v_0, v_L at the beginning of the code, with no need to change anything else in the body of it. The same applies for the number of points in the two directions and the related loops. The general rule is: Do not "hard wire" information in the code if you can help it!
- For the test problem start with $M = 1$ and use a moderate amount of nodes, e.g. 10×10, increasing them if needed. When you try a larger M you may need to increase the number of nodes.
- It is a good practice to break up the code into separate pieces. This makes it much easier to debug it. For the present problem you may have a piece where the problem is set up, i.e., among others, where L_x, L_y, f and u_0, u_L, v_0, v_L are assigned, a piece where $f(x,y)$ is calculated at the nodes, and a separate piece implementing the iterative solution method of the linear system. A similar structure is demosntrated in the code for Assignment 4P that was uploaded on Blackboard.

6. Write a computer code to solve the two-dimensional Poisson equation

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \;=\; -F(x,y) \tag{10.3.7}$$

The domain of interest is the rectangle

$$a_x < x < b_x\,, \qquad a_y < y < b_y \tag{10.3.8}$$

and the boundary conditions

$$u(x, y = b_y) \;=\; f_b(x)\,, \qquad u(x, y = a_y) \;=\; g_b(x)\,, \tag{10.3.9}$$

$$\left.\frac{\partial u}{\partial x}\right|_{x=b_x} \;=\; 0\,, \qquad u(x = a_x, y) \;=\; g_b(a_x) + \frac{y - a_y}{b_y - a_y}[f_b(a_x) - g_b(a_x)] \tag{10.3.10}$$

$$a_x = a_y = 0\,, \qquad b_x = b_y = 2\pi \tag{10.3.11}$$

$$f_b(x) = (b_x - x)^2 \cos\frac{\pi x}{b_x}\,, \qquad g_b(x) = x(b_x - x)^2 \tag{10.3.12}$$

$$F(x,y) = \cos\left[\frac{\pi}{2}\left(2\frac{x-a_x}{b_x-a_x}+1\right)\right]\sin\left[\pi\frac{y-a_y}{b_y-a_y}\right] \tag{10.3.13}$$

Use ghost node(s) for Neumann condition(s).
After carrying out all the simulations needed for the report, run one last simulation with $F = 0$ and include the results in the report.

7. Write a computer code to solve the two-dimensional Poisson equation

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = -F(x,y) \tag{10.3.14}$$

The domain of interest is the rectangle

$$a_x < x < b_x\,, \qquad a_y < y < b_y \tag{10.3.15}$$

and the boundary conditions

$$u(x, y = b_y) = f_a(x)\,, \qquad u(x, y = a_y) = g_a(x)\,, \tag{10.3.16}$$

$$\left.\frac{\partial u}{\partial x}\right|_{x=a_x} = 0\,, \qquad u(x = b_x, y) = g_a(b_x) + \frac{y-a_y}{b_y-a_y}[f_a(b_x) - g_a(b_x)] \tag{10.3.17}$$

$$a_x = a_y = -\pi\,, \qquad b_x = b_y = \pi \tag{10.3.18}$$

$$f_a(x) = (x - a_x)^2 \cos\frac{\pi x}{a_x}\,, \qquad g_a(x) = x(x - a_x)^2 \tag{10.3.19}$$

$$F(x,y) = \cos\left[\frac{\pi}{2}\left(2\frac{x-a_x}{b_x-a_x}+1\right)\right]\sin\left[\pi\frac{y-a_y}{b_y-a_y}\right] \tag{10.3.20}$$

Use ghost node(s) for Neumann condition(s).
After carrying out all the simulations needed for the report, run one last simulation with $F = 0$ and include the results in the report.

11.2.4 Diffusion equation

1. Write a computer code to solve by the Crank-Nicolson method over the time interval $0 \le t \le T$ the one-dimensional diffusion equation

$$\frac{\partial u}{\partial t} = D\frac{\partial^2 u}{\partial x^2} + F(x,t)\,,$$

with $0 \le x \le L$ and the constant D and the function $F(x,t)$ prescribed. The initial condition is

$$u(x, t = 0) = f(x)\,, \tag{10.3.21}$$

and the boundary conditions

$$u(x = 0, t) = g_0(t)\,, \qquad u(x = L, t) = g_L(t)\,, \tag{10.3.22}$$

with $f(x)$, $g_0(t)$ and $g_L(t)$ also prescribed.

To test your program you can consider the following situations:

(a) $L = \pi$, $D = 0.1$, $T = 10$, $F = 0$, $g_0 = g_L = 0$, $f(x) = \sin kx$ with k an integer. The exact solution in this case is $u_{exact}(x,t) = \exp(-Dk^2 t)\,\sin kx$.

(b) $L = \pi$, $D = 0.1$, $g_0 = \sin \omega t$, $g_L(t) = \sin \omega t\, \cos kL$, $f(x) = 0$, $F(x,t) = (\omega \cos \omega t + Dk^2 \sin \omega t) \cos kx$, with k and integer. The exact solution in this case is $u_{exact}(x,t) = \sin \omega t\, \cos kx$. For a fixed Δt try different increasing values of ω, from $\omega \Delta t = 0.1$ on up. What happens to the error?

In each case calculate the average error

$$\epsilon \;=\; \frac{1}{N} \sum_{\ell=1}^{N} \left| \frac{u(x_\ell, T) - u_{exact}(x_\ell, T)}{u_{exact}(x_\ell, T)} \right| \tag{10.3.23}$$

where N is the number of interior nodes. Examine grid convergence and make sure that the results that you show are grid independent.

Submit a report with some graphs comparing the exact and numerical solution, $u(x,t)$ as a function of x at a few values of t, for example $t = T/5,\, T/2,\, T$, and also some tables including some (only some!) numerical values to permit a better comparison between the two solutions. Remember that you are supposed to write a report explaining what you have done and what conclusions you have drawn. The report should include only enough material to support these conclusions. Don't just "throw" graphs and tables to us – this is not what you are asked to do and this will not improve your grade!

All the code writing history should be logged using `git` version control and the *.git* folder should be submitted along with the code.

2. In the interval $0 \le x \le \pi$ consider the diffusion equation

$$\frac{\partial u}{\partial t} \;=\; D \frac{\partial^2 u}{\partial x^2}\,,$$

with $u = 0$ at $x = 0$ and $x = \pi$ and $u(x,0) = f(x)$ with $f(x) = \sin Nx$, with N an integer; take $D = 1$ and use $N = 1$ and $N = 5$. The exact solution is $u(x,t) = \exp(-DN^2 t)\,\sin Nx$.
(a) Write a computer program to solve this equation by using the explicit method. Convince yourselves that, if the stability condition is violated, the numerical solution blows up;
(b) Repeat using the implicit method; use the tri-diagonal program that you wrote for an earlier assignment to solve the linear system;
(c) And then repeat with the Crank-Nicolson method. Compare the Crank-Nicolson results with those of the implicit method and use a time step that demonstrates the superior accuracy of the Crank-Nicolson method.
Turn in your codes and graphs that compare the solution at different instants of time with the exact solution. For each figure, do not forget to supply information on the t to which it refers, the number of spatial nodes used, the value used for Δt and the value of the all-important quantity $D\Delta t/\Delta x^2$.

Write a computer code to solve the two-dimensional diffusion equation

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \;=\; \frac{\partial u}{\partial t} \tag{10.3.24}$$

The domain of interest is the rectangle

$$a_x < x < b_x\,, \qquad a_y < y < b_y \tag{10.3.25}$$

and the boundary conditions

$$u(x, y = b_y, t) \;=\; (1 - e^{-\lambda t}) f_b(x)\,, \qquad u(x, y = a_y, t) \;=\; (1 - e^{-\lambda t}) g_b(x)\,, \tag{10.3.26}$$

$$\left.\frac{\partial u}{\partial x}\right|_{x = b_x} \;=\; 0\,, \qquad u(x = a_x, y, t) \;=\; (1 - e^{-\lambda t}) \left[g_b(a_x) + \frac{y - a_y}{b_y - a_y} [f_b(a_x) - g_b(a_x)] \right] \tag{10.3.27}$$

$$a_x \,=\, a_y \,=\, 0\,, \qquad b_x \,=\, b_y \,=\, 2\pi \tag{10.3.28}$$

$$f_b(x) \,=\, (b_x - x)^2 \cos\frac{\pi x}{b_x}\,, \qquad g_b(x) \,=\, x(b_x - x)^2 \tag{10.3.29}$$

Use a value between 0.05 and 0.5 for λ. The initial condition is

$$u(x, y, t = 0) = 0\,. \tag{10.3.30}$$

Use ghost node(s) for Neumann condition(s).
Carry out the time integration to steady state, i.e., until the result becomes independent of time.

3. Write a computer code to solve the two-dimensional diffusion equation

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \,=\, \frac{\partial u}{\partial t} \tag{10.3.31}$$

The domain of interest is the rectangle

$$a_x < x < b_x\,, \qquad a_y < y < b_y \tag{10.3.32}$$

and the boundary conditions

$$u(x, y = a_y, t) \,=\, (1 - e^{-\lambda t})\phi_{ab}(x)\,, \qquad u(x, y = b_y, t) \,=\, (1 - e^{-\lambda t})\psi_{ab}(x)\,, \tag{10.3.33}$$

$$\left.\frac{\partial u}{\partial x}\right|_{x=a_x} \,=\, 0\,, \qquad \left.\frac{\partial u}{\partial x}\right|_{x=b_x} \,=\, 0\,, \tag{10.3.34}$$

$$a_x \,=\, a_y \,=\, -\pi\,, \qquad b_x \,=\, b_y \,=\, \pi \tag{10.3.35}$$

$$\phi_{ab}(x) \,=\, \{\cos[\pi(x - a_x)] - 1\}\,\cosh(b_x - x)\,, \qquad \psi_{ab}(x) \,=\, (x - a_x)^2\,\sin\frac{\pi(x - a_x)}{2(b_x - a_x)} \tag{10.3.36}$$

For λ use a value between 0.05 and 0.5. The initial condition is

$$u(x, y, t = 0) = 0 \tag{10.3.37}$$

Use ghost node(s) for Neumann condition(s).
Carry out the time integration to steady state, i.e., until the result becomes independent of time.

11.2.5 Wave equation

1. In principle, two possible discretizations of the one-dimensional wave equation

$$\frac{\partial u}{\partial t} + \frac{\partial u}{\partial x} = 0\,,$$

utilize, for the spatial derivative, the backward formula

$$\frac{u_j^{n+1} - u_j^n}{\Delta t} + \frac{u_j^n - u_{j-1}^n}{\Delta x} = 0\,,$$

or the forward formula

$$\frac{u_j^{n+1} - u_j^n}{\Delta t} + \frac{u_{j+1}^n - u_j^n}{\Delta x} = 0\,.$$

Write a code to solve the equation with both formulae in the range $0 < x$ subject to the initial condition

$$u(x, 0) = \begin{cases} x(1 - x) & \text{for } 0 < x < 1 \\ 0 & \text{for } 1 < x \end{cases}$$

- Find the analytic solution of the problem
- Show that the backward formula is stable if the Courant number is less then or at most equal to 1, but it produces a damped solution; compare the numerical with the exact solution; examine the amount of damping as you change Δx;
- Show that the backward formula is unstable if the Courant number is greater than 1;
- Show that the forward formula is unconditionally unstable.