NUMERICAL COMPUTATION

Gonzalo Galiano Casas *Esperanza García Gonzalo*

Department of Mathematics, University of Oviedo

Contents

Chapter 1

Finite arithmetic and error analysis

1 Introduction

While real numbers may have a representation using an infinite number of digits, the amount of memory available in a computer is finite. Thus, a restriction for representing and handling real numbers must apply. Numbers in computers are stored using two main formats:

- *Integer format*, which allows an exact storing of a finite set of integer numbers.
- *Floating point format*, allowing the exact storing of a finite set of rational numbers.

1.1 IEEE 754 standard

The standard floating point representation commonly implemented in today processors is the IEEE 754 format.

The first IEEE 754 norm dates to 1985, where only the binary representation was implemented. Its fundamental formats were single and double precision formats. In 2008, a second version was introduced, extending the previous one to deal with decimal representation and a further quadruple precision binary representation. These five basic formats, with their main parameters, are shown in Table 1.1. These parameters will be explained along the chapter.

Apart from these basic formats, other less commonly used are available such as the extended precision and the extensible precision format, allowing for further accuracy in number representation.

Before the establishment of IEEE 754 standard, FPU's (Floating Point Units) or math co-processors were optional integrated circuits added to the motherboard which, together with the main processors, were in charge of floating point operations. These operations were particular to each operative system and compilators.

After IEEE 754 arrival, the math co-processors became standard. Nowadays, these processors compute both basic operations, like summing, and more complex operations, such as trigonometric functions evaluation. However, most current processors implement only the 1985 standard, being the 2008 version implemented via software.

Apart from defining the storing format and rounding rules for floating point representation, the IEEE 754 standard also deals with the main arithmetic operations, the conversion between

	Binary formats			Decimal formats	
parameter	binary32	binary64	binary128	decimal64	decimal128
precision (p)	24	53	113	16	34
e_{max}	+127	+1023	+16383	+384	+6144

Table 1.1: Main parameters in the IEEE 754 basic formats

different formats, and the exception rules. IEEE 754 does not specify integer representation, but for its role as exponents in floating point representation.

1.2 Binary and decimal representation

Every real number has a decimal representation and a binary representation (and, indeed, a representation based on any positive integer greater than 1). Instead of representation, we sometimes use the word *expansion*.

In particular, the representation of integer numbers is straightforward, requiring an expansion in nonnegative powers of the base. For example, consider the number

$$(71)_{10} = 7 \times 10^1 + 1 \times 10^0,$$

and its binary equivalent

$$(1000111)_2 = 1 \times 2^6 + 0 \times 2^5 + 0 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0.$$

Non-integral real numbers have digits (or bits) to the right of the decimal (or binary) point. These expansions may be finite or nonterminating. For example, $11/2$ has the expansions

$$\frac{11}{2} = (5.5)_{10} = 5 \times 10^0 + 5 \times 10^{-1},$$

and

$$\frac{11}{2} = (101.1)_2 = 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + 1 \times 2^{-1}.$$

Both of these expansions terminate. However, the number $1/10$, which obviously has the finite decimal representation $(0.1)_{10}$, does not have a finite binary representation. Instead, it has the nonterminating expansion

$$\frac{1}{10} = (0.0001100110011\ldots)_2 = 1 \times 2^{-4} + 1 \times 2^{-5} + 0 \times 2^{-6} + 0 \times 2^{-7} + 1 \times 2^{-8} + 1 \times 2^{-9} + \cdots$$

Note that this representation, although nonterminating, is repeating. The fraction $1/3$ has nonterminating expansions in both binary and decimal:

$$\frac{1}{3} = (0.333\ldots)_{10} = (0.010101\ldots)_2.$$

Rational numbers always have either finite or periodic expansions. For example,

$$\frac{1}{7} = (0.142857142857\ldots)_{10}.$$

In fact, any finite expansion can also be expressed as a periodic expansion. For example, $1/10$ can be expressed as

$$\frac{1}{10} = (0.09999\ldots)_{10}.$$

However, we will use the finite expansion when it does exist. Irrational numbers always have nonterminating, non-repeating expansions. For example,

$$\sqrt{2} = (1.414213\ldots)_{10}, \quad \pi = (3.141592\ldots)_{10}, \quad e = (2.718281\ldots)_{10}.$$

Definition 1 *A **decimal floating point representation** of a nonzero real number, x, is a representation of the type*

$$x = \sigma \times (\bar{x})_{10} \times 10^n,$$

*where $\sigma = \pm 1$ is the **sign**, $\bar{x} \in \mathbb{R}$ is the **mantissa**, and $n \in \mathbb{Z}$ is the **exponent**. Similarly, a **binary floating point representation** of a nonzero real number, x, is a representation of the type*

$$x = \sigma \times (\bar{x})_2 \times 2^e.$$

*The representation is said to be **normalized** if*

- *In the decimal case, the mantissa, satisfies $(1)_{10} \leq \bar{x} < (10)_{10}$.*
- *In the binary case, the mantissa, satisfies $(1)_2 \leq \bar{x} < (10)_2$.*

*The **significant digits of a number** are the digits of the mantissa not counting leading zeros. Thus, for normalized numbers, the number of significant digits is the same that the number of digits in the mantissa.*

*The **precision of a representation** is the maximum number, p, of significant digits that can be represented. For a normalized representation, the precision coincides with the number of digits in the mantissa.*

*The precision may be **finite**, if $p < \infty$, or **infinite**, if there is no limit to the number of digits in the mantissa.*

Example 1.1 Normalization and significant digits. For the number $x = 314.15$, the normalized decimal floating point representation has

$$\sigma = +1, \quad \bar{x} = 3.1415, \quad n = 2,$$

so the representation has 5 significant digits. The binary number $x = (10101.11001)_2$ has the normalized representation $(1.010111001)_2 \times 2^4$, with 10 significant digits.

The number $x = (101.001101)_2 = (5.203125)_{10}$ has the normalized floating point decimal representation with

$$\sigma = +1, \quad \bar{x} = 5.203125, \quad n = 0,$$

while the normalized binary floating point representation has

$$\sigma = (1)_2, \quad \bar{x} = (1.01001101)_2, \quad e = (2)_{10} = (10)_2.$$

Thus, the number of significant digits is 7 for the decimal representation, and 9 for the binary representation. □

Example 1.2 Precision of a representation. Suppose that, for a binary representation, we use p digits in the mantissa. If the representation of a given number, x, can be normalized, then it will have the form

$$x = \pm 1.b_1 b_2 \ldots b_{p-1} \times 2^e.$$

Since it can not have leading zeros, the precision of the representation is p. Now, suppose that the representation of x can not be normalized, and that it is of the form

$$x = \pm 0.0 \ldots 0 b_j \ldots b_{p-1} \times 2^e.$$

where $b_j \neq 0$ and $j \leq p-1$. Then, the precision of the representation is $p-j$. □

1.3 Conversion from decimal to binary and vice versa

Binary to decimal conversion is straightforward, as we are so familiar with decimal representation. For example,

$$(1101011.101)_2 = 2^6 + 2^5 + 2^3 + 2^1 + 2^0 + 2^{-1} + 2^{-3} = (107.625)_{10}.$$

Decimal to binary conversion is performed in two steps. First, converting the integer part of the number. Second, converting its fractional part. The algortihm is as follows:

1. Integer part. We sequentially divide by 2 and keep the remainders as the digits in base 2. We first write the last quotient (1, in the example) and then the remainders, from right to left:

Quotients	107	53	26	13	6	3	1
Remainders	1	1	0	1	0	1	
						←	

2. Fractional part. We sequentially multiply by 2 and subtract the integer part. The binary digits are the remainders, written from left to right:

Fractional	0.625	0.25	0.5	0
Integer		1	0	1
		→		

The final result is $(107.625)_{10} = (1101011.101)_2$, as expected.

2 Integer representation

As already mentioned, the IEEE 754 standard does not specifically deal with integer representation. However, since the exponent of the floating point representation is an integer, we shall give some notions on their binary representation.

Binary	Unsigned	IEEE 754
0000	0	Reserved
0001	1	-6
0010	2	-5
0011	3	-4
0100	4	-3
0101	5	-2
0110	6	-1
0111	7	0
1000	8	1
1001	9	2
1010	10	3
1011	11	4
1100	12	5
1101	13	6
1110	14	7
1111	15	Reserved

Table 1.2: Four bits integer representations

For m-bits unsigned integers, we may represent 2^m numbers, those in the range of integer numbers between $(00\ldots00)_2 = (0)_{10}$ and $(11\ldots11)_2 = (2^m-1)_{10}$. Table 1.2 shows the example $m = 4$.

For m-bits signed integers, there are several representation estrategies. We shall focus in the representation used in the IEEE 754 standard.

This standard uses the *biased representation*: numbers are represented consecutively, running increasingly from the smallest negative number to the largest positive number. Since the first and the last exponent values are reserved for special cases (e.g. infinity or `NaN` symbols), there are 2^m-2 representable numbers, those in the range $\left[-2^{m-1}+2, 2^{m-1}-1\right]$.

Thus, the IEEE 754 representation of x is the same that the representation of the unsigned integer $x+b$, where $b = 2^{m-1}-1$ is the *bias* of the representation.

3 IEEE 754 floating point binary representation

The IEEE 754 floating point binary representation of a number $x \neq 0$ is given by

$$x = \sigma \times \bar{x} \times 2^e.$$

- The first bit is for the sign, σ, which stores 0 for positive numbers, and 1 for negative numbers.
- The exponent, e, is a signed integer following the IEEE 754 biased representation, in which the largest and the smallest exponents are reserved for special cases.

Figure 1.1: Single precision representation (32 bits).

- The mantissa is normalized[1], that is, $1 \leq \bar{x} < (10)_2$. In the binary representation the normalization implies that the first digit must be 1, and then it is unnecessary to store it. In this way, a bit is saved. This is known as the *hidden bit technique*.

Numbers may be stored in *bit-strings* of 32 bits (single precision), 64 bits (double precision), and 128 bits (quadruple precision).

3.1 Single precision (32 bits)

In single precision, numbers are stored as $x = \sigma \times (1.a_1a_2\ldots a_{23}) \times 2^e$. The 32 bits are distributed in the following way: 1 bit for the sign, 8 bits for the exponent, and 23 bits for the mantissa. Observe that, due to the hidden bit, the actual precision of this representation, for normalized numbers, is $p = 24$.

Since we have 8 bits for the exponent, this means that there is room for $2^8 = 256$ binary numbers. The smallest, $(00000000)_2$, is reserved to represent zero and other *denormalized numbers*. The largest, $(11111111)_2$, is reserved for the infinity (`Inf`) and Not-a-Number (`NaN`) symbols.

The exponent bias is $2^{m-1} - 1 = 127$, and thus the exponent take the integer values in $[-126, 127]$. Introducing the notation $e_{min} = -126$ and $e_{max} = 127$, we may check that one advantage of this technique is that the inverse of a normalized number having the minimum exponent is always smaller than the largest number,

$$\frac{1}{\bar{x} \times 2^{e_{min}}} = \frac{1}{\bar{x} \times 2^{-126}} = \frac{1}{\bar{x}} \times 2^{126} < 2^{127},$$

since $\bar{x} \geq 1$ due to the hidden bit. Thus, no overflow may take place.

Moreover, the biased representation is more efficient for number comparison. When comparison between two numbers take place, first the exponents are compared, and only in the case they coincide, their mantissas are compared too.

Example 1.3 Compute the single precision IEEE 754 binary representation of the number $(-118.625)_{10}$.

The mantissa. For the fractional part of the mantissa, we get

Fractional :	0.625	0.25	0.5	0
Integer :	1	0	1	

and therefore, we store $(0.101)_2$. For the integer part, we obtain

Quotients :	118	59	29	14	7	3	1
Remainders :	0	1	1	0	1	1	

and thus we store $(1110110)_2$. The complete mantissa is written as

$$(1110110.101)_2.$$

[1] There are exceptions, as we shall see.

Figure 1.2: Double precission precision representation (64 bits).

The result is easy to check:

$$1\times 2^6+1\times 2^5+1\times 2^4+0\times 2^3+1\times 2^2+1\times 2^1+0\times 2^0+1\times 2^{-1}+0\times 2^{-2}+1\times 2^{-3}=118.625.$$

Following the IEEE standard, we normalize the mantissa as

$$1110110.101 = 1.110110101\times 2^6,$$

which is stored as

`11011010100000000000000`.

Recall that due to the hidden bit technique, the first 1 is omitted.

The exponent. The bias is $2^{m-1}-1=127$. The base 10 biased exponent is then $6+bias=6+127=133$. Computing its binary representation

Quotients:	133	66	33	16	8	4	2	1
Remainders:	1	0	1	0	0	0	0	

we get $(10000101)_2$.

The sign. Since the number is negative, the sign bit is 1.

Therefore, the answer is

sign	exponent	mantissa
1	10000101	11011010100000000000000

□

3.2 Double precision (64 bit)

In this case, numbers are stored as $x=\sigma\times(1.a_1a_2\ldots a_{52})\times 2^e$. The 64 bits are distributed in the following way: 1 bit for the sign, 11 bits for the exponent, and 52 bits for the mantissa. It has therefore a precision $p=53$ for normalized numbers, taking into account the hidden bit.

The 11 bits for the exponent give room for $2^{11}=2048$ binary numbers, distributed in the interval $[-1022,1023]$, the bias being 1023. The smallest and the largest exponents are reserved for special cases, like in the single precision case.

3.3 Special values

We discuss here the single precision special values. The corresponding double precision exceptions are analogous. As mentioned in the previous subsections, the special values are stored in the largest and smallest exponent values.

- The largest exponent is $e=(11111111)_2$. This exponent is reserved for:

– *Infinity.* All the mantissa digits are zeros. It is due to *overflow*.

Value	sign	exponent	mantissa
$+\infty$	0	11111111	00000000000000000000000
$-\infty$	1	11111111	00000000000000000000000

– `NaN` (Not a Number). The mantissa is not identically zero. There are two kind: `QNaN` (Quiet `NaN`), meaning *indeterminate*, and `SNaN` (Signaling `NaN`) meaning *invalid operation*. Attempts to compute $0/0$, 0^0, or similar expressions result in `NaN`.

Value	sign	exponent	mantissa
SNaN	0	11111111	10000000000000001000000
QNaN	1	11111111	00000010000000010000000

- The smallest exponent is $e = (00000000)_2$. This exponent is reserved for:

  – *Zero.* Since the hidden bit takes the value 1, it is not possible to represent the zero as a normalized number. The following representations are used

Value	sign	exponent	mantissa
$+0$	0	00000000	00000000000000000000000
-0	1	00000000	00000000000000000000000

  – *Denormalized numbers*[2]. The hidden bit is assumed to be zero, and the exponent value is assumed to take the smallest possible value, that is $(00000001)_2$, although it is still represented with `00000000`. For example,

sign	exponent	mantissa
0	00000000	00001000010000000001000
1	00000000	01000100000000000010000

  The advantage of introducing denormalized numbers is that, since the hidden bit is zero, numbers smaller than the smallest normalized number may be represented, filling thus the gap between zero and the smallest normalized number. However, these numbers have less significant digits (lower precision) than the normalized numbers, since they have leading zeroes (at least, the hidden bit).

  Another difference with respect to normalized numbers is in number distribution: while normalized numbers have a logarithmic distribution, denormalized numbers have a linear distribution.

Example 1.4 Compute the base 10 value and the precision representation of the number

sign	exponent	mantissa
0	00000000	00010110000000000000000

Since the exponent is `00000000` and the mantissa is not identically zero, the number is denormalized. Thus, the exponent is $e_{min} = -126$, and the hidden bit is 0. Therefore, it represents the number

$$(0.0001011) \times 2^{-126},$$

[2] Also known as *subnormal* numbers, in IEEE 754-2008.

with precision $p = 24 - 4 = 20$. In decimal base, is given by

$$(2^{-4} + 2^{-6} + 2^{-7}) \times 2^{-126} \approx 1.0102 \times 10^{-39}.$$

□

Observe that the smallest single precision normalized number, R_{min}, is, in absolute value,

sign	exponent	mantissa
0	00000001	00000000000000000000000

that is, $(1.00\ldots00) \times 2^{-126}$, which is larger than the largest denormalized number $(0.11\ldots11) \times 2^{-126}$, writen as

sign	exponent	mantissa
0	00000000	11111111111111111111111

Example 1.5 Compute the smallest denormalized numbers in single and double precision.

In single precision, it is

sign	exponent	mantissa
0	00000000	00000000000000000000001

representing, in binary base

$$(0.00000000000000000000001) \times 2^{-126} = 2^{-23} \times 2^{-126} = 2^{-149} \approx 1.4013 \times 10^{-45},$$

which has a precision $p = 1$. Similarly, in double precision we get

$$(2^{-52}) \times 2^{-1022} = 2^{-1074} \approx 4.9407 \times 10^{-324}.$$

□

3.4 Accuracy

We have two main ways of measuring the accuracy of floating point arithmetics:

- *The machine epsilon*, ε, which is the difference between 1 and the next number, $x > 1$, which is representable.
- *The largest integer*, M, such that any other positive integer, $x \leq M$, is representable.

Machine epsilon in single and double precision. The single precision normalized representation of 1 is the 24 binary digits number

$$(1.0\underbrace{\ldots}_{22}0)_2 \times 2^0.$$

If we add a normalized number with exponent smaller than -23, then the resulting number will have a mantissa with more than the 24 permitted digits. Thus, the smallest normalized number, ε, such that $1 + \varepsilon > 1$ in single precision is $1. \times 2^{-23}$. Indeed, we have

$$1 + \varepsilon = (1.0\ldots0)_2 \times 2^0 + (1.0\ldots0)_2 \times 2^{-23} = (1.0\ldots01)_2 \times 2^0.$$

	Decimal represented	Binary 25 digits	Mantissa 1.+23 bits	Exp	Representation
	1	000...001	1.00...000	0	Exact
	2	000...010	1.00...000	1	Exact
	3	000...011	1.10...000	1	Exact
	4	000...100	1.00...000	2	Exact
	⋮	⋮	⋮	⋮	⋮
	16777215	011...111	1.11...111	23	Exact
$M = 2^{24} \rightarrow$	16777216	100...000	1.00...0000	24	Exact
	16777216	100...001	1.00...0001	24	**Rounded**
	16777218	100...010	1.00...0010	24	Exact
	16777220	100...011	1.00...0011	24	**Rounded**
	16777220	100...100	1.00...0100	24	Exact
	⋮	⋮	⋮	⋮	⋮

Table 1.3: Single precision floating point integer representation

That is, for single precision, we get $\varepsilon = 2^{-23} \approx 1.19 \times 10^{-7}$. In a similar way, we get for double precision $\varepsilon = 2^{-52} \approx 2.22 \times 10^{-16}$.

Largest integer. The largest integer is $M = 2^p$. Let us justify this statement using Table 1.3 for single precision. The arguments for double precision follow the same line.

As shown in Table 1.3, all numbers smaller than $M = 2^{24}$ admit a normalized exact representation in single precision.

For $M = 2^{24}$, the last digit may not be stored, but since this digit is zero, following the rounding rules, see Subsection 3.5, M is rounded to the closest number finishing in zero. Thus, in this case, there is no loss of digits and the representation is exact.

However, for the next number the last digit is one, and rounding leads to a cutting off of this digit, implying no exact representation. From this number on, some integers are represented in an exact form and some others are not. Since in decimal base we have

$$M = 2^{24} = 16777216,$$

we deduce that all the six-digits integers are stored exactly.

A similar argument for double precision representation shows that

$$M = 2^{53} = 9007199254740992$$

is the largest integer. Thus, integers up to 15 digits are stored exactly.

Overflow and underflow

Since for any given precision there are a maximum and a minimum storable positive numbers, some procedure must be followed if these barriers are violated. When operations lead to numbers larger than the maximum storable number, an *overflow* is produced. The IEEE 754 format may support this result assigning the symbols $\pm\infty$, and usually, aborting the execution.

On the contrary, if some operations lead to a number which is smaller than the minimum positive number, an *underflow* is produced. Then two results are possible. That the number lies in the range of denormalized numbers, so it is still representable (although with a loss of precision), or that it is even smaller than the smaller positive denormalized number. In this case, the number is rounded to zero. In both cases, execution continues.

3.5 Rounding

When operations lead to a number for which the mantissa contains more digits than the precision of the representation, the number must be approximated by another representable number. For instance, let us consider the base 10 number

$$x = \pm d_0.d_1d_2\ldots \times 10^n = \pm\Big(\sum_{k=0}^{\infty} d_k 10^{-k}\Big) \times 10^n, \tag{1.1}$$

with $d_k = 0, 1, ..., 9$, for all k, and $d_0 \neq 0$. For a precision p, the digits $d_p, d_{p+1}, \ldots$ must be dropped from the representation, possibly implying a modification of the last representable digit, d_{p-1}.

In the norm IEEE 754 we have four procedures to approximate x:

- Round up: taking the closest representable larger number.
- Round down: taking the closest representable smaller number.
- Round towards zero (*truncation*): replacing the non representable digits by zero.
- Round to nearest representable digit (*rounding*).

The most usual procedures are truncation and rounding. We explain them in some detail.

Decimal representation

In this case, x is given by formula (1.1). We have, for a precision of p digits,

- Truncation:
$$x^* = \pm d_0.d_1d_2\ldots d_{p-1} \times 10^n.$$
- Rounding:
$$x^* = \begin{cases} \pm d_0.d_1d_2\ldots d_{p-1} \times 10^n & \text{if } 0 \leq d_p \leq 4, \\ \pm\big(d_0.d_1d_2\ldots d_{p-1} + 10^{-(p-1)}\big) \times 10^n & \text{if } 5 < d_p \leq 9, \\ \pm\big(d_0.d_1d_2\ldots d_{p-1} + 10^{-(p-1)}\big) \times 10^n & \text{if } d_p = 5, \text{ and } d_{p+k} > 0 \text{ for some } k > 0, \\ \text{nearest number ending in even} & \text{if } d_p = 5, \text{ and } d_{p+k} = 0 \text{ for all } k > 0. \end{cases}$$

Example 1.6 Round the following numbers in decimal base:

number	precision	truncation	rounding
1.999953	5	1.9999	2.000
2.433309	4	2.433	2.433
2.433500	4	2.433	2.434
2.434500	4	2.434	2.434

□

Binary representation

In this case, the number takes the form

$$x = \pm 1.b_1 b_2 \ldots \times 2^e = \pm\Big(\sum_{k=0}^{\infty} b_k 2^{-k}\Big) \times 2^e,$$

with $b_k = 0, 1$ for all k. For a precision p (including the hidden bit), we have

- Truncation:
$$x^* = \pm 1.b_1 b_2 \ldots b_{p-1} \times 2^e.$$
- Rounding:
$$x^* = \begin{cases} \pm 1.b_1 b_2 \ldots b_{p-1} \times 2^e & \text{if } b_p = 0, \\ \pm\big(1.b_1 b_2 \ldots b_{p-1} + 2^{-(p-1)}\big) \times 2^e & \text{if } b_p = 1 \text{ and } b_{p+k} = 1 \text{ for some } k > 0, \\ \text{nearest number ending in 0} & \text{if } b_p = 1 \text{ and } b_{p+k} = 0 \text{ for all } k > 0. \end{cases}$$

Example 1.7 Round the following numbers in binary base:

number	precision	truncation	rounding
1.1111	3	1.11	10.0
1.1101	3	1.11	1.11
1.0010	3	1.00	1.00
1.0110	3	1.01	1.10

Let us explain the roundings of the last two numbers. For both, we have $b_p = b_3 = 1$, and $b_{p+k} = 0$ for all $k > 0$ (only $k = 1$, in this example). Then, we round both numbers to the nearest representable number ending in zero, that is, we look for the nearest number with $b_2 = 0$. For 1.0010 this is clearly 1.00. For 1.0110, the possibilities are $x_1^* = 1.00$ and $x_2^* = 1.10$, and we have

$$|x - x_1^*| = 1.0110 - 1.0000 = 0.0110, \quad |x - x_2^*| = 1.1000 - 1.0110 = 0.0010.$$

To convince yourself of the last substraction, write it as

$$2^0 + 2^{-1} - (2^0 + 2^{-2} + 2^{-3}) = \frac{1}{2} - \frac{1}{4} - \frac{1}{8} = \frac{1}{8} = 2^{-3}.$$

□

Let us finish this section by comparing the approximation results obtained by truncation and by rounding for the binary representation of precision p. If truncating, we have

$$|x - x_t^*| = \Big(\sum_{k=p}^{\infty} b_k 2^{-k}\Big) \times 2^e \leq 2^{-(p-1)} 2^e,$$

where we used the formula for summing a geometric series. For rounding to the nearest, we have an even better behavior since the rounded value, x, is always, at worst, halfway between the two nearest representable numbers. Thus,

$$|x - x_r^*| \leq \frac{1}{2} 2^{-(p-1)} 2^e = 2^{-p} 2^e. \tag{1.2}$$

Therefore, the largest error we may have by truncating is twice the largest error made by rounding.

Example 1.8 Let $x = (1.1001101)_2$. We approximate by

- Truncation to 5 binary digits, $x_t^* = (1.1001)_2$. Then

$$|x - x_t^*| = (0.0000101)_2 = 2^{-5} + 2^{-7} = 0.0390625.$$

- Rounding to 5 binary digits, $x_r^* = (1.1010)_2$. In this case

$$|x - x_r^*| = (0.0000011)_2 = 2^{-6} + 2^{-7} = 0.0234375.$$

□

4 Error

Rounding errors due to finite arithmetic are small in each operation. However, if we concatenate many operations these errors may aggregate and propagate along the code variables. The result can be a large error between the exact solution and the computed solution. This effect is known as *numerical instability*.

Example 1.9 For the sequence $s_k = 1 + 2 + \ldots + k$, for $k = 1, 2, \ldots$, if we compute

$$x_k = \frac{1}{s_k} + \frac{2}{s_k} + \ldots + \frac{k}{s_k},$$

the exact result is

$$x_k = 1 \text{ for all } k = 1, 2, \ldots$$

However, in single precision we get

k	x_k^*	$\lvert x_k - x_k^* \rvert$
10^1	1.000000	0.0
10^3	0.999999	1.0×10^{-7}
10^6	0.9998996	1.004×10^{-4}
10^7	1.002663	2.663×10^{-3}

□

Definition 2 *The **absolute error** due to approximating x by x^* is defined as $e_a = |x - x^*|$, while the **relative error** of the same approximation is given by*

$$e_r = \frac{|x - x^*|}{|x|}.$$

The relative error is scale-independent, and therefore more meaningful than the absolute error, as we may check in the following example.

Example 1.10 Compute the absolute and relative errors corresponding to approximating x by x^*:

x	x^*	e_a	e_r
0.3×10^1	0.31×10^1	0.1	$0.333... \times 10^{-1}$
0.3×10^{-3}	0.31×10^{-3}	0.1×10^{-4}	$0.333... \times 10^{-1}$
0.3×10^4	0.31×10^4	0.1×10^3	$0.333... \times 10^{-1}$

□

Example 1.11 Compute estimates for the relative errors of truncation and rounding approximations. We have

$$\frac{|x - x_t^*|}{|x|} = \frac{\left(\sum_{k=p}^{\infty} b_k 2^{-k}\right) \times 2^e}{\left(\sum_{k=0}^{\infty} b_k 2^{-k}\right) \times 2^e} = \frac{\sum_{k=p}^{\infty} b_k 2^{-k}}{\sum_{k=0}^{\infty} b_k 2^{-k}}.$$

Since $b_0 = 1$, the number in the denominator is larger than one. Thus,

$$\frac{|x - x_t^*|}{|x|} < \sum_{k=p}^{\infty} b_k 2^{-k} \leq 2^{-(p-1)} = \varepsilon,$$

where ε is the machine epsilon. Similarly to (1.2), and using the above argument, we get

$$\frac{|x - x_r^*|}{|x|} < 2^{-p} = \frac{\varepsilon}{2}.$$

□

Definition 3 *We say that x^** ***approximates*** *x* ***with*** *p* ***significant digits*** *if p is the largest nonnegative integer such that*

$$\frac{|x - x^*|}{|x|} \leq 5 \times 10^{-p}.$$

Example 1.12 Let us find the significant digits in the following cases:

$x^* = 124.45$ approximates $x = 123.45$ with $p = 2$ significant digits, since

$$\frac{|x - x^*|}{|x|} = \frac{1}{123.45} = 0.0081 \leq 0.05 = 5 \times 10^{-2}.$$

$x^* = 0.0012445$ approximates $x = 0.0012345$ with $p = 2$ significant digits, since

$$\frac{|x - x^*|}{|x|} = \frac{0.00001}{0.0012345} = 0.0081 \leq 0.05 = 5 \times 10^{-2}.$$

$x^* = 999.8$ approximates $x = 1000$ with $p = 4$ significant digits, since

$$\frac{|x - x^*|}{|x|} = \frac{0.2}{1000} = 0.0002 \leq 0.0005 = 5 \times 10^{-4}.$$

□

Chapter 2

Nonlinear equations

1 Introduction

In this chapter, we study numerical methods to compute approximations to the roots or zeros of nonlinear equations of the type

$$f(x) = 0, \tag{2.1}$$

where $f : \mathbb{R} \to \mathbb{R}$ is a continuous function. In general, solutions of (2.1) can not be expressed in explicit form. Moreover, even if this is possible, it is seldom useful due to the complexity of the expression involved.

The numerical methods we study are of iterative nature. Starting from an initial approximation, and using some algorithms, we produce a sequence of approximations that, hopefully, converge to the solution.

Iterative methods must be stopped at some point, after a finite number of iterations. Thus, in general, we only obtain approximations to the solutions we look for. In addition, the rounding errors generated by the evaluations of $f(x)$ also limit the precision of any numerical method of approximation.

With some methods, like bisection method, it is enough to know the initial interval containing the solution to ensure the convergence of the sequence generated by the algorithm. However, other methods, although faster, are more sensible to the initial guess for starting the algorithm. Thus, we normally use an hybrid method in which one starts, say, with the bisection method to locate the solution and the we apply a finer method, like Newton's method, to approximate further the solution.

1.1 Order of convergence and stopping criterion

In the lines above we introduced some concepts which deserve to be detailed. Numerical methods for root approximation are *iterative methods*, that is, by means of an algorithm we define a sequence

$$x_0, x_1, \ldots, x_k, \ldots$$

such that $\lim_{k\to\infty} x_k = \alpha$. Then, due to the continuity of f we may infer

$$\lim_{k\to\infty} f(x_k) = f(\alpha) = 0.$$

The order of convergence of a method is related to the intuitive idea of speed of convergence of the sequence with respect to k, which a useful concept for algorithm comparison.

Definition 4 *Let us suppose that the sequence x_k converges to $\alpha \in \mathbb{R}$. We say that x_k* converges to α with order of convergence *p if*

$$\lim_{k\to\infty} \frac{|x_k - \alpha|}{|x_{k-1} - \alpha|^p} \neq 0, \text{ and finite.}$$

In the particular cases

- $p = 1$, we say that the convergence is *linear*,
- $p = 2$, the convergence is *quadratic*.

A numerical method is said to be of *order p* if the corresponding sequence converges to the solution with order of convergence p.

The sequence generated by the algorithm is, in general, infinite. Thus, a stopping criterion (or test) is needed to break the sequence at some point. The most crude criterion is that of setting a maximum number of iterations. Such criterion does not provide any information about the accuracy of the approximation. Most usual criterion are based on, for some small *tolerance* $\varepsilon > 0$,

- The *absolute difference* between two consecutive iterations,

$$|x_k - x_{k-1}| < \varepsilon.$$

- The *relative difference* between two consecutive iterations,

$$\frac{|x_k - x_{k-1}|}{|x_k|} < \varepsilon.$$

- The *residual* at iteration k,

$$|f(x_k)| < \varepsilon.$$

In practice, a combination of these criterion may be used. For instance, a maximum number of iterations together with a difference test, in order to prevent infinite loops (because ε is too small) or, simply, too long execution times.

2 The bisection method

For root approximation, one usually starts collecting qualitative information like the number of roots or their approximate location. This information can be gathered inspecting the graph of $f(x)$, which is normally a very useful tool to determine the number of roots and to enclose them in some suitable intervals.

Example 2.1 Consider the equation

$$\frac{x^2}{4} = \sin(x).$$

In Figure 2.1 the graphs of $y = x^2/4$, and $y = \sin(x)$ are plotted. By inspection, we may determine that the unique *positive* root, α, lies in the interval $(1.8, 2)$, being $\alpha \approx 1.9$. □

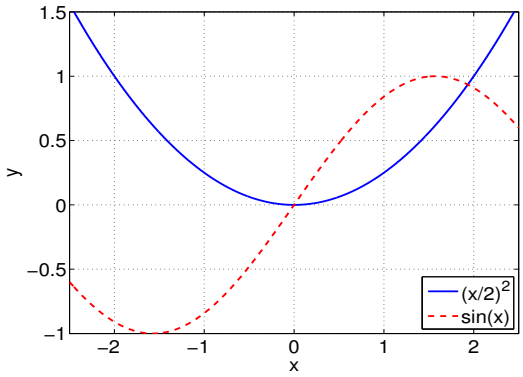


Figure 2.1: Plots of $y = x^2/4$, and $y = \sin(x)$.

The following theorem may be used to deduce whether the interval $[a,b]$ contains, at least, one root of the equation $f(x) = 0$.

Theorem 2.1 (Intermediate value) *Assume that the function $f(x)$ is continuous for all $x \in [a,b]$, with $f(a) \neq f(b)$, and that k is an intermediate value between $f(a)$ and $f(b)$. Then, there exists $\xi \in (a,b)$ such that $f(\xi) = k$.*

In particular, if $f(a)f(b) < 0$ then the equation $f(x) = 0$ has, at least, one root in the interval (a,b).

The bisection method makes a systematic use of the intermediate value theorem. Suppose that $f(x)$ is continuous in the interval $[a_0, b_0]$, and that $f(a_0)f(b_0) < 0$. In what follows, we shall determine a sequence of nested intervals $I_k = [a_k, b_k]$ such that

$$(a_0, b_0) \supset (a_1, b_1) \supset (a_2, b_2) \supset \cdots$$

all of them containing the root of the equation. These intervals are recursively determined as follows. Given $I_k = (a_k, b_k)$, we compute the middle point

$$m_k = \frac{a_k + b_k}{2} = a_k + \frac{1}{2}(b_k - a_k), \tag{2.2}$$

and $f(m_k)$. The way of expressing m_k by the right hand term in (2.2) has the advantage of minimizing the rounding error when computing the middle point.

We may assume that $f(m_k) \neq 0$ since, otherwise, we already found the root. The new interval is defined as

$$I_{k+1} = (a_{k+1}, b_{k+1}) = \begin{cases} (m_k, b_k) & \text{if } f(m_k)f(a_k) > 0, \\ (a_k, m_k) & \text{if } f(m_k)f(a_k) < 0. \end{cases}$$

From this definition it follows that $f(a_{k+1})f(b_{k+1}) < 0$, and therefore the interval I_{k+1} also contains a root of $f(x) = 0$.

After n iterations of the bisection method, the root lies in the interval (a_n, b_n), of length $2^{-n}(b_0 - a_0)$. That is, if we take m_n as an approximation to the root of $f(x)$, then we have an estimate for the absolute error

$$|\alpha - m_n| < 2^{-(n+1)}(b_0 - a_0). \tag{2.3}$$

In each step, a binary digit is gained in the accuracy of the approximation. Thus, finding an interval of length δ containing a root takes around $\log_2((b_0 - a_0)/\delta)$ evaluations of $f(x)$.

The expression (2.3) implies that the bisection method has a linear order of convergence. Clearly, the stopping criterion should be based on the absolute error between two iterations, which allows us to determine the number of iterations needed to achieve the prescribed tolerance, see Exercise **??**.

Example 2.2 The bisection method applied to the equation $f(x) = 0$, with $f(x) = x^2/4 - \sin(x)$, and $I_0 = (1.8, 2)$ gives the following sequence of intervals $[a_k, b_k]$,

k	a_k	b_k	m_k	$f(m_k)$
0	1.8	2	1.9	−0.0438
1	1.9	2	1.95	0.0217
2	1.9	1.95	1.925	−0.0115
3	1.925	1.95	1.9375	0.0050
4	1.925	1.9375	1.93125	−0.0033
5	1.93125	1.9375	1.934375	0.0008

Table 2.1:

Thus, after six iterations, we get $\alpha \in (1.93125, 1.934375)$, an interval of length $0.2 \times 2^{-6} \approx 0.003$. □

The execution time required by the bisection method is proportional to the number of evaluations of $f(x)$ and, therefore, the convergence is slow. But independent of the function smoothness. For smooth functions, for instance differentiable functions, other methods such as Newton's method give a faster convergence.

3 Newton's method

The only information used by the bisection method is the sign of $f(x)$ on the extremes of the intervals generated by the method. When the function is smooth, more efficient methods may be devised by taking advantage not only of the values of $f(x)$ in each iteration but also those of its derivatives.

Let $f : [a, b] \to \mathbb{R}$ be a differentiable function, and consider its approximation by the tangent line to f at the point $x_k \in (a, b)$, given by

$$y(x) = f(x_k) + f'(x_k)(x - x_k).$$

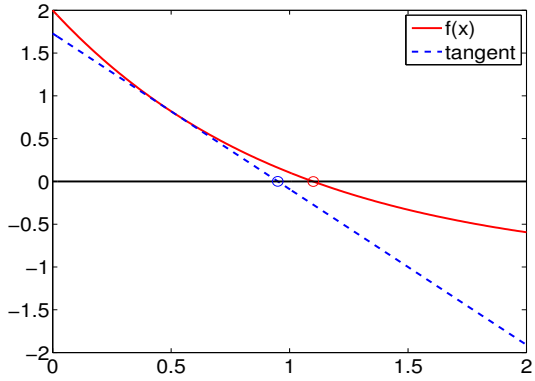


Figure 2.2: Geometric meaning of Newton's method. In each step, the root of the tangent is computed as an approximation to the root of the function.

If we fix x_{k+1} such that $y(x_{k+1}) = 0$, that is, such that it is an approximation to a root of $f(x)$, we get

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}, \quad k \geq 0, \tag{2.4}$$

whenever $f'(x_k) \neq 0$. The expression (2.4) is known as *method of Newton* and it corresponds to computing the zero of $f(x)$ locally replacing $f(x)$ by its tangent at x_k.

Note that to initialize Newton's method a first approximation or *guess*, x_0, is needed. This choice can be tricky since the method does not converge, in general. In practice, a initial guess may be obtained using the bisection method or by directly inspecting the graph of $f(x)$.

If x_0 is suitably chosen, and α is a *single* zero (i.e., $f'(\alpha) \neq 0$) then Newton's method is convergent. Moreover, if $f''(x)$ is continuous, it may be proven that the convergence is quadratic, see Exercise **??**.

The usual stopping criterium for Newton's method and, in general, for all fixed point based methods that we shall study in Section 4, is the absolute difference between two consecutive iterands

$$|x_{k+1} - x_k| < \varepsilon, \tag{2.5}$$

for a given tolerance $\varepsilon > 0$. Like in the bisection method, in practice, we also limit the maximum number of iterations to avoid infinite loops.

Newton's method can be easily extended to deal with systems of nonlinear equations. Thus, if $\mathbf{f} : \Omega \subset \mathbb{R}^N \to \mathbb{R}^N$ is given by

$$\begin{cases} f_1(x_1, x_2, \ldots, x_N) = 0, \\ f_2(x_1, x_2, \ldots, x_N) = 0, \\ \vdots \\ f_N(x_1, x_2, \ldots, x_N) = 0, \end{cases}$$

then the Newton's method to solve $\mathbf{f}(\mathbf{x}) = \mathbf{0}$, where $\mathbf{x} = (x_1, x_2, \ldots, x_N)$ and $\mathbf{f} = (f_1, \ldots, f_N)$, is as follows: given $\mathbf{x}_0 \in A$, for $k = 0, 1, \ldots$ and till convergence, we define

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \big(J_{\mathbf{f}}(\mathbf{x}_k)\big)^{-1} \mathbf{f}(\mathbf{x}_k),$$

where $J_{\mathbf{f}}(\mathbf{x}_k)$ is the Jacobian matrix of $\mathbf{f}(\mathbf{x})$ evaluated in $\mathbf{x}_k$, that is

$$\big(J_{\mathbf{f}}(\mathbf{x}_k)\big)_{ij} = \frac{\partial f_i}{\partial x_j}(\mathbf{x}_k).$$

As we already noticed for scalar functions, $f'(x_k)$ must be nonzero. Similarly, for vector functions the Jacobian matrix must have a well defined inverse, i.e. $\det\left(J_{\mathbf{f}}(\mathbf{x}_k)\right) \neq 0$ must hold. For the stopping criterium, we replace (2.5) by

$$\|\mathbf{x}_{k+1} - \mathbf{x}_k\| < \varepsilon,$$

where $\|\mathbf{y}\| = \left(\sum_{i=1}^{N} y_i\right)^{1/2}$ is the Euclidean norm of $\mathbf{y}$.

Example 2.3 Newton's method applied to the equation $f(x) = 0$, with $f(x) = x^2/4 - \sin(x)$, and $x_0 = 1.8$. Compare to Table 2.1 produced with the bisection method.

k	x_k	$f(x_k)$
0	1.8	-0.16384
1	1.94	0.01543
2	1.9338	9.e-05
3	1.933753765	3.e-09
4	1.933753762827021	-1.e-16

Table 2.2:

□

4 The fixed point method

In this section we introduce a general class of iterative methods used for root approximations as well as for other applications.

We say that a function $g : [a,b] \to \mathbb{R}$ has a *fixed point* α in the interval $[a,b]$ if $g(\alpha) = \alpha$. The fixed point method is based on the iteration

$$x_{k+1} = g(x_k), \quad k \geq 0, \tag{2.6}$$

where x_0 is an initial guess to be provided.

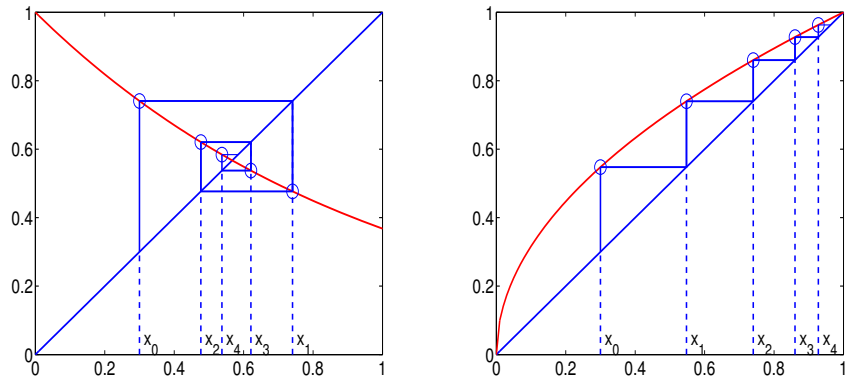
The fixed point method is of great generality and gives raise to the introduction of particular algorithms when the function g is specified. For instance, if we want to approximate a zero of $f : [a,b] \to \mathbb{R}$ using the fixed point method, we just have to define $g(x) = x + f(x)$, so if α is a fixed point of g then it is also a root of f. However, there is not a unique way to set this equivalence, as we show in the following example.

Example 2.4 The equation $x + \ln(x) = 0$ may be written, for example, as

$$(i)\ x = -\ln(x), \quad (ii)\ x = \mathrm{e}^{-x}, \quad (iii)\ x = \frac{x + \mathrm{e}^{-x}}{2}.$$

Notice that each of these equations lead to a different fixed point scheme, see Exercise **??**. □

A graphic interpretation of the fixed point method is shown in Figure 2.3. As it can be observed, in some cases the method is not convergent even for a initial guess arbitrarily close to the root. Therefore, we need to find some conditions which ensure the convergence of the method.



(a) $x_{k+1} = e^{-x_k}$, with $x_0 = 0.3$ (b) $x_{k+1} = \sqrt{x_k}$, with $x_0 = 0.01$

Figure 2.3: Examples of fixed point iterations: convergent (left), and divergent from the closest root (right)

Theorem 2.2 (Contractive map) *Let g be a function defined in the interval $[a,b] \subset \mathbb{R}$ and $x_0 \in [a,b]$ be an initial guess for the fixed point iteration defined in (2.6). Suppose that*

1. *$g(x) \in [a,b]$ for all $x \in [a,b]$,*
2. *g is differentiable in $[a,b]$,*
3. *There exists a positive constant $\gamma < 1$ such that $|g'(x)| \leq \gamma$ for all $x \in [a,b]$.*

Then g has a unique fixed point $\alpha \in [a,b]$, and the sequence x_k defined by (2.6) converges to α at least with linear order of convergence. More precisely,

$$\lim_{k\to\infty} \frac{|x_{k+1} - \alpha|}{|x_k - \alpha|} = |g'(\alpha)|.$$

Assume, in addition, that for some integer number $p > 1$, the function g is $p+1$ times continuously differentiable, and that $g^{(n)}(\alpha) = 0$ for $n = 1, \ldots, p-1$, and $g^{(p)}(\alpha) \neq 0$. Then, the order of convergence is p:

$$\lim_{k\to\infty} \frac{|x_{k+1} - \alpha|}{|x_k - \alpha|^p} = \frac{|g^{(p)}(\alpha)|}{p!}.$$

As already introduced for Newton's method, see (2.5), the stopping criterium for the fixed point method is usually based on the absolute difference between two consecutive iterations, plus the usual limitation in the maximum number of iterations.

Remark 2.1 *Newton's method can be deduced from the fixed point method by taking*

$$g(x) = x - \frac{f(x)}{f'(x)}.$$

Since Newton's method is quadratic, we may explore whether the result on the order of convergence stated in Theorem 2.2 may be improved. The answer is given in Exercise **??**.

Example 2.5 Fixed point method applied to the equation $g(x) = 0$, with $g(x) = x + f(x)$, $f(x) = x^2/4 - \sin(x)$, and $x_0 = 1.8$. Observe that function g is not contractive in the interval $(1.8, 2)$, but it is in an interval centered at zero. Thus, although farer away, the fixed point method converges to that root.

k	x_k	$f(x_k)$
0	1.8	-0.16384
1	1.6	-0.32861
2	1.3	-0.53813
3	0.7	-0.54771
4	0.2	-0.20759
5	0.01	-0.01404
6	0.00005	-5.e-05
7	0.0000000006	-6.e-10

Table 2.3:

However, with the simple change $g(x) = x - f(x)$, the fixed point method converges to the correct root:

k	x_k	$f(x_k)$
0	1.8	-0.16384
1	1.96	0.04042
2	1.923	-0.01358
3	0.937	0.00430
4	1.932	-0.00139
5	1.934	0.00044
6	1.9336	-0.00014
7	1.93378	0.00004
8	1.93374	-0.00001
9	1.933757	4.e-05
10	1.933752	1.e-05

Table 2.4:

Compare to Tables 2.1 and 2.2. □

5 The secant method

One of the main drawbacks of Newton's method is that we need to evaluate the derivative of the function in the points defined by the sequence of iterations. In some occasions, this is not possible due to the partial knowledge of the function, for instance at a finite number of points, as in a data sample of some physical magnitude.

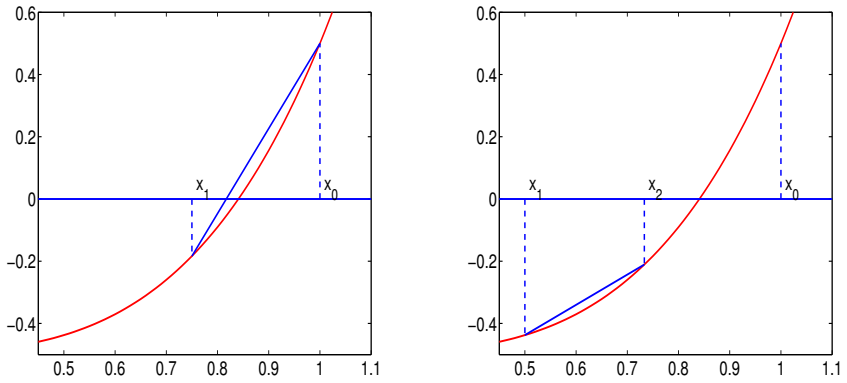


Figure 2.4: One iteration of Newton's method (left) and the secant method (right) for $f(x) = x^4 - 0.5$.

The secant method is a variant of Newton's method in which we approximate $f'(x)$ by the incremental quotient. Since

$$f'(x) = \lim_{y \to x} \frac{f(x) - f(y)}{x - y},$$

we may approximate $f'(x_{k-1})$ by

$$f'(x_{k-1}) \approx \frac{f(x_{k-1}) - f(x_{k-2})}{x_{k-1} - x_{k-2}}.$$

In this way, we obtain the following iterative scheme. Given *two* initial guesses x_0 and x_1, we take, for $k = 2, 3 \ldots$,

$$x_k = x_{k-1} - f(x_{k-1}) \frac{x_{k-1} - x_{k-2}}{f(x_{k-1}) - f(x_{k-2})}, \tag{2.7}$$

whenever $f(x_{k-1}) \neq f(x_{k-2})$.

When the secant method is convergent, the term $|x_{k-1} - x_{k-2}|$ becomes very small, and therefore the quotient $(x_{k-1} - x_{k-2})/(f(x_{k-1}) - f(x_{k-2}))$ will be determined with a poor numerical accuracy since if the approximations x_{k-1}, x_{k-2} are close to α, then the rounding error may be large.

However, an error analysis allows us to infer that, in general, the approximations satisfy $|x_{k-1} - x_{k-2}| \gg |x_{k-1} - \alpha|$ and, therefore, the main contribution to the rounding error comes from the term $f(x_{k-1})$.

Observe that formula (2.7) should not be simplified to

$$x_k = \frac{x_{k-2} f(x_{k-1}) - x_{k-1} f(x_{k-2})}{f(x_{k-1}) - f(x_{k-2})},$$

because this formula could lead to cancellation errors when $x_{k-1} \approx x_{k-2}$ and $f(x_{k-1}) f(x_{k-2}) > 0$. Even formula (2.7) may not be safe since, when $f(x_{k-1}) \approx f(x_{k-2})$, we could face division by zero or by numbers close to zero, leading to overflow. For these reasons, the most convenient form for the iterations is

$$s_{k-1} = \frac{f(x_{k-1})}{f(x_{k-2})}, \quad x_k = x_{k-1} + \frac{s_{k-1}}{1 - s_{k-1}} (x_{k-1} - x_{k-2}),$$

where the division by $1 - s_{k-1}$ takes place only if $1 - s_{k-1}$ is large enough.

Finally, it can be proven that the order of convergence of the secant method is lower than that of the Newton's method, and is given by $p = (1+\sqrt{5})/2 \approx 1.618$. The stopping criterion is similar to that introduced for Newton's method.

Example 2.6 Secant method applied to the equation $f(x) = 0$, with $f(x) = x^2/4 - \sin(x)$, $x_0 = 1.8$, and $x_1 = 2$. Compare to Tables 2.1, 2.2 and 2.4 produced with the other methods introduced in this chapter.

k	x_k	$f(x_k)$
0	1.8	-0.16384
1	2	0.09070
2	1.92	-0.00661
3	1.9335	-0.00022
4	1.933754	6.e-07
5	1.933753	-5.e-11

Table 2.5:

□

Chapter 3

Interpolation and approximation

In solving mathematical problems, we often need to evaluate a function in one or several points. However, there may arise drawbacks, such as:

- It can be expensive, in terms of processor use or time execution, to evaluate a complicated function.
- It may happen that we only have the value of a function at a finite set of points, like when sampling some physical magnitude.

A possible strategy to overcome these difficulties is to replace the complicate or partially unknown function by another, simpler function, which can be efficiently evaluated. These simpler functions are usually chosen among polynomials, trigonometric functions, rational functions, etc.

1 Interpolation

Definition 5 Interpolating *a given function, f, with another function, $\tilde{f}$, consists on, given the following data:*

- $n+1$ *different points* $x_0,\ x_1,\ldots,\ x_n$,
- $n+1$ *values of* f *at those points,* $f(x_0)=\omega_0,\ f(x_1)=\omega_1,\ldots,\ f(x_n)=,\omega_n$,

find a simple *function, $\tilde{f}$, such that $\tilde{f}(x_i)=\omega_i$, with $i=0,1,\ldots,n$.*

The points $x_0,\ x_1,\ldots,x_n$ are called nodes of interpolation*, and the function $\tilde{f}$ is called* interpolant of f in $x_0,\ x_1,\ldots,x_n$.

In what follows, we shall consider three types of interpolants:

- Polynomial interpolant, of the type

$$\tilde{f}(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_n x^n = \sum_{k=0}^{n} a_k x^k.$$

- Trigonometric interpolant, of the type

$$\tilde{f}(x) = a_{-M} e^{-iMx} + \ldots + a_0 + \ldots + a_M e^{iMx} = \sum_{k=-M}^{M} a_k e^{ikx},$$

  where $M = n/2$ if n is odd, and $M = (n-1)/2$ if n is even. Recall that i denotes the imaginary unit, and that $e^{ikx} = \cos(kx) + i\sin(kx)$.

- Piecewise polynomial interpolant, of the type

$$\tilde{f}(x) = \begin{cases} p_1(x) & \text{if } x \in (\tilde{x}_0, \tilde{x}_1) \\ p_2(x) & \text{if } x \in (\tilde{x}_1, \tilde{x}_2) \\ \cdots & \\ p_m(x) & \text{if } x \in (\tilde{x}_{m-1}, \tilde{x}_m) \end{cases}$$

  where $\tilde{x}_0, \ldots, \tilde{x}_m$ form a *partition* of the interval containing the interpolation nodes, (x_0, x_n), and $p_i(x)$ are polynomials.

2 Polynomial interpolation: the Lagrange polynomial

We seek for a polynomial interpolant (replacing the notation $\tilde{f}$ by P_n)

$$P_n(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n, \tag{3.1}$$

satisfying

$$P_n(x_0) = \omega_0, \quad P_n(x_1) = \omega_1, \quad P_n(x_2) = \omega_2, \quad \ldots \quad P_n(x_n) = \omega_n. \tag{3.2}$$

Evaluating the expression (3.1) in the nodes of interpolation and equating to the values ω_i, we get that the conditions (3.2) are equivalent to the polynomial coefficients being solution of the following system of linear equations

$$\begin{pmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^n \\ 1 & x_1 & x_1^2 & \cdots & x_1^n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^n \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{pmatrix} = \begin{pmatrix} \omega_0 \\ \omega_1 \\ \vdots \\ \omega_n \end{pmatrix}.$$

The coefficient matrix

$$A = \begin{pmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^n \\ 1 & x_1 & x_1^2 & \cdots & x_1^n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^n \end{pmatrix},$$

is of the *Vandermonde* type, with determinant given by

$$\det(A) = \prod_{0 \le l \le k \le n} (x_k - x_l).$$

Clearly, since the interpolation nodes are different, we have $\det(A) \neq 0$, and therefore the system has a unique solution, that is, there exists a unique polynomial P_n satisfying (3.2).

Such polynomial, P_n, is called the *Lagrange interpolation polynomial* in the points $x_0, x_1, \ldots, x_n$ relative to the values $\omega_0, \omega_1, \ldots, \omega_n$.

If the number of nodes, n, is large, solving the linear system may be expensive. However, there exist alternative methods which allows us to compute the Lagrange polynomial in a more efficient way. Among them, those using the *Lagrange fundamental polynomials*, and the *divided differences*.

2.1 Lagrange fundamental polynomials

It is a fundamental result that for each $i = 0, 1, \ldots, n$, there exists a unique polynomial ℓ_i of degree up to n such that $\ell_i(x_k) = \delta_{ik}$, where δ_{ik} denotes the *Kronecker's delta*[1]. Such polynomial is given by

$$\ell_i(x) = \prod_{\substack{j=0 \\ j \neq i}}^{n} \frac{x - x_j}{x_i - x_j}. \tag{3.3}$$

The polynomials $\ell_0, \ell_1, \ldots, \ell_n$ are called *Lagrange fundamental polynomials* of degree n. Observe that these polynomials only depend upon the interpolation nodes, x_i, and not on the values, ω_i. That is, the fundamental polynomials are not interpolants, but a useful tool to build them.

Definition 6 *The* Lagrange polynomial interpolant *in* $x_0, x_1, \ldots, x_n$ *relative to* $\omega_0, \omega_1, \ldots, \omega_n$ *is given by*

$$P_n(x) = \omega_0 \ell_0(x) + \omega_1 \ell_1(x) + \cdots + \omega_n \ell_n(x). \tag{3.4}$$

Clearly, since in the node x_i the only nonzero fundamental polynomial is $\ell_i(x)$ (taking the value one in x_i), we have

$$P_n(x_i) = \omega_i,$$

for $i = 0, \ldots, n$, and then $P_n(x)$ satisfies the interpolation conditions (3.2).

Example 3.1 Consider, for $i = 0, 1, 2$, the nodes $x_i = i$ and the values $\omega_i = f(x_i)$, with $f(x) = 1/(x+1)$. We have

$$\ell_0(x) = \frac{x - x_1}{x_0 - x_1} \frac{x - x_2}{x_0 - x_2} = \frac{x-1}{-1} \frac{x-2}{-2} = \frac{1}{2}(x-1)(x-2),$$

and, similarly, we obtain

$$\ell_1(x) = -x(x-2), \quad \ell_2(x) = \frac{1}{2}x(x-1).$$

Therefore

$$P_2(x) = \frac{1}{2}(x-1)(x-2) - \frac{1}{2}x(x-2) + \frac{1}{6}x(x-1).$$

□

[1] $\delta_{ik} = 0$ if $i \neq k$, $\delta_{ik} = 1$ if $i = k$.

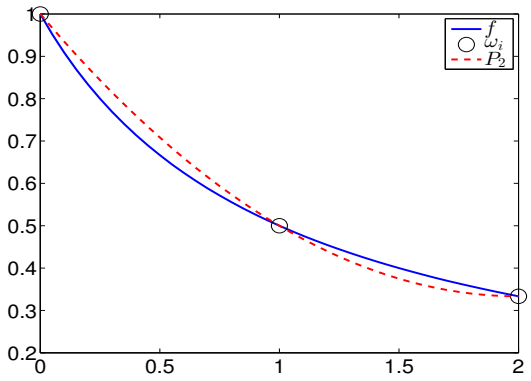


Figure 3.1: $f(x) = 1/(x+1)$, and its degree two Lagrange interpolant.

Computing the Lagrange polynomial in this way has a drawback: once the degree n polynomial is obtained, if the approximation is not good enough and we need to increase de degree of the interpolant, we have to redo all the computations again. To circumvect this difficulty, we shall use Newton's *method of divided differences*.

2.2 Divided differences

We may rewrite the Lagrange interpolation polynomial as

$$P_n(x) = c_0 + c_1(x-x_0) + c_2(x-x_0)(x-x_1) + \cdots + c_n(x-x_0)\cdots(x-x_n), \tag{3.5}$$

where $c_0, \ldots, c_n$ are constants to be determined. For $x = x_0$ we have $P_n(x_0) = c_0$, and also, due to the interpolation conditions, $P_n(x_0) = \omega_0$. Therefore, $c_0 = \omega_0$.

Dividing the expression (3.5) by $(x-x_0)$ and taking into account that $c_0 = \omega_0$, we get

$$\frac{P_n(x) - \omega_0}{x - x_0} = c_1 + c_2(x-x_1) + \cdots + c_n(x-x_1)\cdots(x-x_n), \tag{3.6}$$

and evaluating in $x = x_1$ we deduce

$$c_1 = \frac{P_n(x_1) - \omega_0}{x_1 - x_0} = \frac{\omega_1 - \omega_0}{x_1 - x_0}.$$

Following this idea, we divide the expression (3.6) by $(x-x_1)$ to get

$$\frac{1}{x-x_1}\left(\frac{P_n(x) - \omega_0}{x-x_0} - \frac{\omega_1 - \omega_0}{x_1 - x_0}\right) = c_2 + c_3(x-x_2) + \cdots + c_n(x-x_2)\cdots(x-x_n),$$

and, evaluating in $x = x_2$, we deduce

$$c_2 = \frac{1}{x_2 - x_1}\left(\frac{\omega_2 - \omega_0}{x_2 - x_0} - \frac{\omega_1 - \omega_0}{x_1 - x_0}\right).$$

Simple arithmetics lead us to write

$$c_2 = \frac{\dfrac{\omega_2 - \omega_1}{x_2 - x_1} - \dfrac{\omega_1 - \omega_0}{x_1 - x_0}}{x_2 - x_0}.$$

Summarizing, and introducing the usual divided differences notation, we have

$$\begin{aligned} c_0 &= [\omega_0] = \omega_0, \\ c_1 &= [\omega_0, \omega_1] = \frac{\omega_1 - \omega_0}{x_1 - x_0}, \\ c_2 &= [\omega_0, \omega_1, \omega_2] = \frac{\dfrac{\omega_2 - \omega_1}{x_2 - x_1} - \dfrac{\omega_1 - \omega_0}{x_1 - x_0}}{x_2 - x_0}. \end{aligned}$$

The key observation is that we may write the second order divided differences , $[\omega_0, \omega_1, \omega_2]$, using only the first order divided differences, $[\omega_1, \omega_2]$ and $[\omega_0, \omega_1]$. Indeed,

$$[\omega_0, \omega_1, \omega_2] = \frac{[\omega_1, \omega_2] - [\omega_0, \omega_1]}{x_2 - x_0}.$$

From these observations, we define the

- Divided differences of order 0,

$$[\omega_i] = \omega_i \quad \text{for } i = 0, 1, \ldots, n. \tag{3.7}$$

- Divided differences of order k ($k = 1, \ldots, n$),

$$[\omega_i, \omega_{i+1}, \ldots, \omega_{i+k}] = \frac{[\omega_{i+1}, \ldots, \omega_{i+k}] - [\omega_i, \omega_{i+1}, \ldots, \omega_{i+k-1}]}{x_{i+k} - x_i}, \tag{3.8}$$

  for $i = 0, 1, \ldots, n-k$.

Once the divided differences corresponding to some interpolation problem have been computed, the Lagrange interpolation polynomial of degree n is computed as follows.

Formula of Newton. The Lagrange interpolant polynomial of degree n is given by

$$\begin{aligned} P_n(x) = &[\omega_0] + [\omega_0, \omega_1](x - x_0) + [\omega_0, \omega_1, \omega_2](x - x_0)(x - x_1) + \cdots + \\ &+ [\omega_0, \omega_1, \ldots, \omega_n](x - x_0)(x - x_1) \cdots (x - x_{n-1}). \end{aligned} \tag{3.9}$$

The main advantage of this formulation is that the Lagrange polynomials of successive order may be computed recursively,

$$P_n(x) = P_{n-1}(x) + [\omega_0, \omega_1, \ldots, \omega_n](x - x_0)(x - x_1) \cdots (x - x_{n-1}).$$

Remark 3.1 *If the interpolation values $\omega_0, \omega_1, \ldots, \omega_n$ are obtained from a function, f, the notation $f[x_0, x_1, \ldots, x_n]$ is often used in place of $[\omega_0, \omega_1, \ldots, \omega_n]$. In such case, Newton's formula is written as*

$$\begin{aligned} P_n(x) = &f[x_0] + f[x_0, x_1](x - x_0) + f[x_0, x_1, x_2](x - x_0)(x - x_1) + \cdots + \\ &+ f[x_0, x_1, \ldots, x_n](x - x_0)(x - x_1) \cdots (x - x_{n-1}). \end{aligned} \tag{3.10}$$

Example 3.2 Consider again the data of Example 3.1, that is, for $i = 0,1,2$, the nodes $x_i = i$ and the values $\omega_i = 1/(i+1)$. We have

$$
\begin{aligned}
[\omega_i] &= \frac{1}{i+1}, \\
[\omega_0,\omega_1] &= \frac{\omega_1-\omega_0}{x_1-x_0} = \frac{\frac{1}{2}-1}{1-0} = -\frac{1}{2}, \\
[\omega_1,\omega_2] &= \frac{\omega_2-\omega_1}{x_2-x_1} = \frac{\frac{1}{3}-\frac{1}{2}}{1-0} = -\frac{1}{6}, \\
[\omega_0,\omega_1,\omega_2] &= \frac{[\omega_1,\omega_2]-[\omega_0,\omega_1]}{x_2-x_0} = \frac{-\frac{1}{6}+\frac{1}{2}}{2} = \frac{1}{6}.
\end{aligned}
$$

Then, the Lagrange polynomial is

$$P_2(x) = 1 - \frac{1}{2}x + \frac{1}{6}x(x-1).$$

If we add new data at the point $x_3 = 3$, with value $\omega_3 = 1/4$, we only have to compute the divided differences

$$
\begin{aligned}
[\omega_2,\omega_3] &= \frac{\omega_3-\omega_2}{x_3-x_2} = \frac{\frac{1}{4}-\frac{1}{3}}{1-0} = -\frac{1}{12}, \\
[\omega_1,\omega_2,\omega_3] &= \frac{[\omega_2,\omega_3]-[\omega_1,\omega_2]}{x_3-x_1} = \frac{-\frac{1}{12}+\frac{1}{6}}{2} = \frac{1}{24}, \\
[\omega_0,\omega_1,\omega_2,\omega_3] &= \frac{[\omega_1,\omega_2,\omega_3]-[\omega_0,\omega_1,\omega_2]}{x_3-x_0} = \frac{\frac{1}{24}-\frac{1}{6}}{3} = -\frac{1}{24},
\end{aligned}
$$

to obtain the Lagrange polynomial of degree 3,

$$P_3(x) = 1 - \frac{1}{2}x + \frac{1}{6}x(x-1) - \frac{1}{24}x(x-1)(x-2).$$

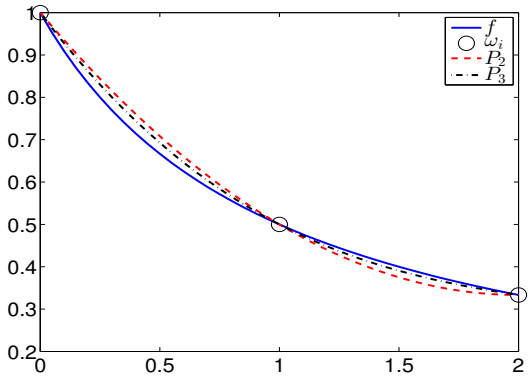


Figure 3.2: $f(x) = 1/(x+1)$, and two of its Lagrange interpolants.

□

2.3 Error estimation

The next result allows us to estimate the error made when replacing f by its Lagrange polynomial interpolant, P_n.

Theorem 3.1 *Suppose that*

- $f : [a,b] \to \mathbb{R}$ *is* $n+1$ *times continuously differentiable in* $[a,b]$.
- $x_0, x_1, \ldots, x_n \in [a,b]$
- $\omega_i = f(x_i)$, *for* $i = 0, 1, \ldots, n$.

Then, for all $x \in [a,b]$ *we have*

$$|f(x) - P_n(x)| \leq \max_{y \in [a,b]} \left| f^{(n+1)}(y) \right| \frac{|(x-x_0)(x-x_1)\cdots(x-x_n)|}{(n+1)!}.$$

In the most usual case in which the nodes are equi-spaced, that is, $x_i = x_{i-1} + h$, for some constant $h > 0$, the error estimate is simplified to

$$\max_{x \in [a,b]} |f(x) - P_n(x)| \leq \frac{\max_{x \in [a,b]} \left| f^{(n+1)}(x) \right|}{4(n+1)} h^{n+1},$$

where we used the estimate (see Exercise **??**)

$$|\Pi_{i=0}^{n}(x - x_i)| \leq \frac{h^{n+1}}{4} n! \tag{3.11}$$

Unfortunately, we can not deduce from this estimate that the error tends to zero when the polynomial degree tends to infinity, even if $h^{n+1}/(4(n+1))$ tends to 0, since the derivatives $f^{(n)}(x)$ could tend to infinity at some points. In fact, there exist examples showing that the limit could be even infinite.

3 Piecewise polynomial interpolation

As shown in the previous section, when the number of nodes for the Lagrange interpolation increases, the following happens:

- The degree of the polynomial interpolant increases, involving the formation of oscillations.
- The approximation does not necessary improves. For improvement, all the derivatives of the interpolated function must be uniformly bounded.

One way to avoide this situation is introducing the so-called *piecewise polynomial functions*. Although some regularity is lost with this technique, we ensure that the error will decrease as the number of interpolation nodes increases.

A degree n polynomial is uniquely determined by its values at $n+1$ different points. Thus, the interpolation by degree zero piecewise polynomials (*constantwise polynomials*) is that in which the polynomials, in this case constants, are determined in each node by, for instance,

$$\tilde{f}(x) = \begin{cases} \omega_0 & \text{if } x \in [x_0, x_1), \\ \omega_1 & \text{if } x \in [x_1, x_2), \\ \dots & \\ \omega_{n-1} & \text{if } x \in [x_{n-1}, x_n), \\ \omega_0 & \text{if } x = x_n. \end{cases}$$

Observe that if $\omega_i \neq \omega_{i+1}$ then $\tilde{f}$ is discontinuous at x_{i+1}.

Similarly, the degree one piecewise polynomial interpolation (*linearwise polynomials*) is that in which the polynomials, in this case straight lines, are determined by two consecutive nodes,

$$\tilde{f}(x) = \omega_i + (\omega_{i+1} - \omega_i)\frac{x - x_i}{x_{i+1} - x_i} \quad \text{if } x \in [x_i, x_{i+1}],$$

for $i = 0, \dots, n-1$. In this case, $\tilde{f}$ is continuous, but its first derivative is, in general, discontinuous at the nodes.

Together with the constantwise and linearwise interpolation, the interpolation with piecewise polynomials of order three (*cubic splines*) are the most important in this family of interpolants.

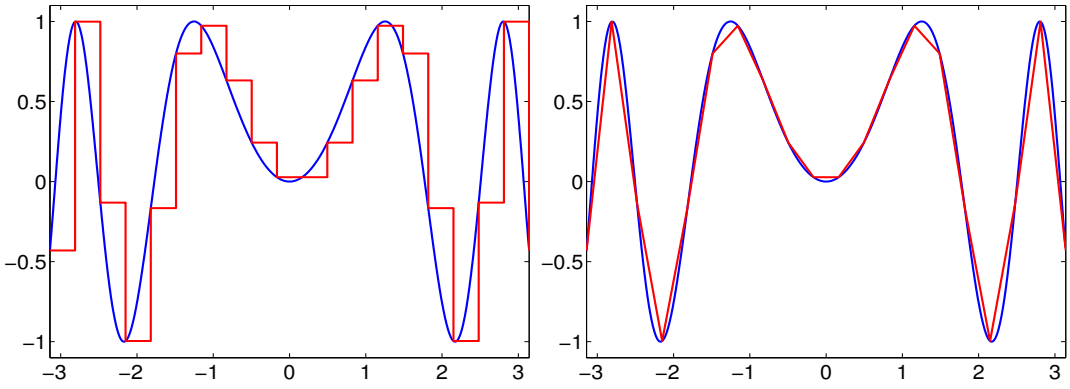


Figure 3.3: Left: constantwise interpolation. Right: linearwise interpolation.

3.1 Spline interpolation

The problem of interpolation by splines of order p (or degree p) consists on finding a function $\tilde{f}$ such that:

1. $\tilde{f}$ is $p-1$ times continuously differentiable in $[x_0, x_n]$.

2. $\tilde{f}$ is a piecewise function given by the polynomials $\tilde{f}_0, \tilde{f}_1, \dots, \tilde{f}_{n-1}$ defined, respectively, in $[x_0, x_1], [x_1, x_2], \dots, [x_{n-1}, x_n]$, and of degree lower or equal to p.

3. The polynomials satisfy the interpolation condition: $\tilde{f}_0(x_0) = \omega_0$, …, $\tilde{f}_n(x_n) = \omega_n$.

It can be proven that, for each $p \geq 1$, this problem has, at least, one solution. These solutions, $\tilde{f}$, are called *spline interpolant of degree p in the points* $x_{0,}x_1,\ldots,x_n$ *relative to the values* $\omega_{0,}\omega_1,\ldots,\omega_n$. The most common spline is the degree $p=3$ spline, also known as *cubic spline*.

Particularizing the above conditions to the case $p=3$ we see that the cubic spline must satisfy:

1. $\tilde{f}$ is twice continuously differentiable in $[x_0,x_n]$.
2. Each polynomial $\tilde{f}_0,\tilde{f}_1,\ldots,\tilde{f}_{n-1}$ defining the pieces of $\tilde{f}$ are of degree ≤ 3.
3. The polynomials satisfy the interpolation condition: $\tilde{f}(x_0)=\omega_0,\ldots,\tilde{f}(x_n)=\omega_n$.

Let us see how to calculate these polynomials. We do it in five steps.

Step 1: Since the second order derivative of $\tilde{f}$ is continuous in $[x_0,x_n]$ we have, in particular,

$$\begin{array}{lcl}
\omega_0'' & = & \tilde{f}_0''(x_0), \\
\omega_1'' & = & \tilde{f}_0''(x_1)=\tilde{f}_1''(x_1), \\
\omega_2'' & = & \tilde{f}_1''(x_2)=\tilde{f}_2''(x_2), \\
\ldots & \ldots & \ldots \\
\omega_{n-1}'' & = & \tilde{f}_{n-2}''(x_{n-1})=\tilde{f}_{n-1}''(x_{n-1}), \\
\omega_n'' & = & \tilde{f}_{n-1}''(x_n),
\end{array}$$

where ω_i'' denotes the unknown value of $\tilde{f}''(x_i)$.

Step 2: The polynomials $\tilde{f}_i$ are of degree ≤ 3. Hence, $\tilde{f}_i''$ are of degree ≤ 1, that is, straight lines or constants, with values ω_i'' and ω_{i+1}'' at the extremes of the interval $[x_i,x_{i+1}]$, respectively. Therefore, we have for $i=0,\ldots,n-1$,

$$\tilde{f}_i''(x)=\omega_i''\frac{x_{i+1}-x}{h_i}+\omega_{i+1}''\frac{x-x_i}{h_i}, \quad \text{with } h_i=x_{i+1}-x_i.$$

Step 3: Integrating each of these polynomials with respect to x, we get

$$\tilde{f}_i'(x)=-\omega_i''\frac{(x_{i+1}-x)^2}{2h_i}+\omega_{i+1}''\frac{(x-x_i)^2}{2h_i}+c_i,$$

where c_i is an unknown integration constant. A new integration leads to

$$\tilde{f}_i(x)=\omega_i''\frac{(x_{i+1}-x)^3}{6h_i}+\omega_{i+1}''\frac{(x-x_i)^3}{6h_i}+a_i(x_{i+1}-x)+b_i(x-x_i), \tag{3.12}$$

where a_i and b_i are unknown integration constants such that $c_i=-a_i+b_i$.

Step 4: We determine the constants a_i and b_i using the interpolation conditions:

$$\tilde{f}_i(x_i)=\omega_i \quad \tilde{f}_i(x_{i+1})=\omega_{i+1}.$$

For $i=0,\ldots,n-1$, we have

$$a_i=\frac{\omega_i}{h_i}-\omega_i''\frac{h_i}{6}, \qquad b_i=\frac{\omega_{i+1}}{h_i}-\omega_{i+1}''\frac{h_i}{6}. \tag{3.13}$$

Step 5: If we plug the expressions (3.13) of a_i and b_i in formula (3.12), we see that the only quantities which need to be determined are the values ω_i'', for $i=0,\ldots,n$. Using that the interpolant $\tilde{f}$ is twice continuously differentiable in $[x_0,x_n]$, we have that at the interior nodes it must hold

$$\tilde{f}_i'(x_{i+1})=\tilde{f}_{i+1}'(x_{i+1}), \quad i=0,\ldots,n-2,$$

giving us the following $n-1$ linear equations

$$\frac{h_i}{6}\omega_i'' + \frac{h_{i+1}+h_i}{3}\omega_{i+1}'' + \frac{h_{i+1}}{6}\omega_{i+2}'' = \frac{\omega_i}{h_i} - \big(\frac{1}{h_{i+1}} + \frac{1}{h_i}\big)\omega_{i+1} + \frac{\omega_{i+2}}{h_{i+1}}.$$

For the full determination of the $n+1$ values ω_i'' we still need two additional equations.

There are several strategies to determinate this system of equations, leading each of them to different variants of cubic splines. For instance, if we fix the value of two unknowns, let us say $\omega_0'' = \omega_n'' = 0$, the variant is known as *natural spline*, and the rest of values ω_i'', $i = 1,\ldots,n-1$ are the unique solution of the linear system

$$\mathbf{H}\omega_{in}'' = 6\mathbf{d}, \tag{3.14}$$

where $\omega_{in}'' = (\omega_1,\ldots,\omega_n)$, $\mathbf{d} = (\Delta_1 - \Delta_0,\ldots,\Delta_{n-1} - \Delta_{n-2})$, with $\Delta_i = (\omega_{i+1} - \omega_i)/h_i$, and

$$\mathbf{H} = \begin{pmatrix} 2(h_0+h_1) & h_1 & 0 & \cdots & 0 & 0 \\ h_1 & 2(h_1+h_2) & h_2 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 2(h_{n-3}+h_{n-2}) & h_{n-2} \\ 0 & 0 & 0 & \cdots & h_{n-2} & 2(h_{n-2}+h_{n-1}) \end{pmatrix}. \tag{3.15}$$

Step 6: Finally, once the value of ω'' is determined, we use formula (3.12) together with (3.13) to define the splines in each subinterval $[x_i, x_{i+1}]$, for $i = 0,\ldots,n-1$.

Example 3.3 We compute the natural cubic splines corresponding to the nodes $x_i = i$, and to the values $\omega_i = i^3$, for $i = 0,1,2,3,4$. The node step size is constant, $h_i = 1$. Thus,

$$\Delta_{i+1} - \Delta_i = \omega_{i+1} - 2\omega_i + \omega_{i-1} = \begin{cases} 6 & \text{for } i = 1, \\ 12 & \text{for } i = 2, \\ 18 & \text{for } i = 3. \end{cases}$$

The matrix $\mathbf{H}$ is given by

$$\mathbf{H} = \begin{pmatrix} 4 & 1 & 0 \\ 1 & 4 & 1 \\ 0 & 1 & 4 \end{pmatrix},$$

and solving the system $\mathbf{H}\omega_{in}'' = 6\mathbf{d}$ and imposing the natural conditions, we obtain (rounding)

$$\omega'' = (0, 6.4286, 10.2857, 24.4286, 0).$$

Now we find a_i and b_i from (3.13), and plug these values in (3.12). Expanding the result in the powers of x, we get

$$\begin{aligned} \tilde{f}_0(x) &= \frac{15x^3}{14} - \frac{x}{14}, \\ \tilde{f}_1(x) &= \frac{9x^3}{14} + \frac{9x^2}{7} - \frac{19x}{14} + \frac{3}{7}, \\ \tilde{f}_2(x) &= \frac{33x^3}{14} - 9x^2 + \frac{269x}{14} - \frac{93}{7}, \\ \tilde{f}_3(x) &= -\frac{57x^3}{14} + \frac{342x^2}{7} - \frac{2161x}{14} + \frac{1122}{7}. \end{aligned}$$

In Figure 3.4 we may visualize the result. □

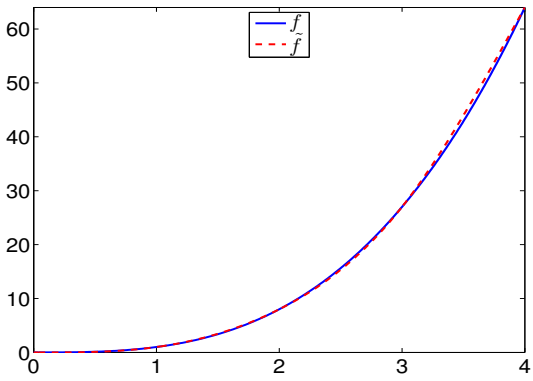


Figure 3.4: The function $f(x) = x^3$ and its natural spline interpolant.

3.2 Error estimation

The next result provides us with an error estimate for piecewise polynomial interpolation. Observe that, independently of the polynomial degree, we can narrow the error as much as we want by choosing the distance between consecutive nodes small enough.

Theorem 3.2 *Suppose that*

- $f : [a,b] \to \mathbb{R}$ *is* $p+1$ *times continuously differentiable in* $[a,b]$.
- $x_0, x_1, \ldots, x_n \in [a,b]$.
- $\omega_i = f(x_i)$, *for* $i = 0, 1, \ldots, n$.

Let $\tilde{h} = \max_{i=0,\ldots,n} h_i$. *Then, for all* $x \in [a,b]$ *we have*

$$\left| f(x) - \tilde{f}(x) \right| \leq c\tilde{h}^{p+1} \max_{y \in [a,b]} \left| f^{(p+1)}(y) \right|,$$

where c *is a constant independent of* f, x *and* $\tilde{h}$.

Example 3.4 Consider the function $f : [0, 2\pi] \to \mathbb{R}$, $f(x) = \sin(x)$, and the nodes $x_j = 2\pi j/N$, with $j = 0, 1, \ldots, N$. Then, $\tilde{h} = 2\pi/N$, and

$$\max_{y \in [0,2\pi]} \left| f^{(p+1)}(y) \right| \leq 1.$$

We deduce that the absolute error is bounded as

$$\left| \sin(x) - \tilde{f}(x) \right| \leq \frac{c}{N^{p+1}},$$

and therefore the order of convergence is $p+1$. □

4 Interpolation with trigonometric polynomials

The usual objective of interpolating with trigonometric polynomials is periodic functions interpolation, that is, interpolation of functions $f : [a,b] \to \mathbb{R}$ such that $f(a) = f(b)$. For simplicity, and without loss of generality[2], we consider the interval $[a,b] = [0, 2\pi]$.

The interpolant, $\tilde{f}$, must satisfy

$$\tilde{f}(x_j) = f(x_j), \quad \text{where } x_j = \frac{2\pi j}{n+1}, \quad \text{for } j = 0, \ldots, n,$$

and have the form, if n is even,

$$\tilde{f}(x) = \frac{a_0}{2} + \sum_{k=1}^{M} \Big(a_k \cos(kx) + b_k \sin(kx) \Big), \tag{3.16}$$

with $M = n/2$, while if n is odd

$$\tilde{f}(x) = \frac{a_0}{2} + \sum_{k=1}^{M} \Big(a_k \cos(kx) + b_k \sin(kx) \Big) + a_{M+1} \cos((M+1)x), \tag{3.17}$$

with $M = (n-1)/2$. Using the identity $e^{ikx} = \cos(kx) + i \sin(kx)$ we may rewrite (3.16) and (3.17) as

$$\tilde{f}(x) = \sum_{k=-M}^{M} c_k e^{ikx} \quad \text{if } n \text{ is even}, \qquad \tilde{f}(x) = \sum_{k=-(M+1)}^{M+1} c_k e^{ikx} \quad \text{if } n \text{ is odd},$$

where

$$a_k = c_k + c_{-k}, \quad b_k = i(c_k - c_{-k}), \quad \text{for } k = 0, \ldots, M, \qquad c_{M+1} = c_{-(M+1)} = a_{M+1}/2.$$

Using the notation

$$\tilde{f}(x) = \sum_{k=-(M+\mu)}^{M+\mu} c_k e^{ikx},$$

with $\mu = 0$ if n is even and $\mu = 1$ if n is odd, the interpolation conditions are

$$\tilde{f}(x_j) = \sum_{k=-(M+\mu)}^{M+\mu} c_k e^{ikjh} = f(x_j), \quad j = 0, \ldots, n,$$

where $h = 2\pi/(n+1)$.

To compute the coefficients c_k we multiply (3.19) by $e^{-imx_j} = e^{-imjh}$, with $m \in \mathbb{Z}$, and sum with respect to j,

$$\sum_{j=0}^{n} \sum_{k=-(M+\mu)}^{M+\mu} c_k e^{ikjhe^{-imjh}} = \sum_{j=0}^{n} f(x_j) e^{-imjh}. \tag{3.18}$$

Using the identity

$$\sum_{j=0}^{n} e^{ijh(k-m)} = (n+1)\delta_{km},$$

[2] If the period is different, for instance T, the change of variable $x = 2\pi t/T$ renders the function to 2π-periodic.

we get

$$\sum_{j=0}^{n}\sum_{k=-(M+\mu)}^{M+\mu} c_k e^{ikjhe^{-imjh}} = \sum_{k=-(M+\mu)}^{M+\mu} c_k(n+1)\delta_{km} = (n+1)c_m.$$

Finally, from (3.18) we deduce (replacing m by k)

$$c_k = \frac{1}{n+1}\sum_{j=0}^{n} f(x_j)e^{-ikjh}, \quad k = -(M+\mu),\ldots,M+\mu.$$

We summarize these computations in the following definition.

Definition 7 *Given $f:[0,2\pi]\to\mathbb{R}$, we define its* discrete Fourier series *in the nodes $x_j = jh$, with $h = 2\pi/(n+1)$ and $j = 0,\ldots,n$ by*

$$\tilde{f}(x) = \sum_{k=-(M+\mu)}^{M+\mu} c_k e^{ikx}, \tag{3.19}$$

where $c_k = \frac{1}{n+1}\sum_{j=0}^{n} f(x_j)e^{-ikjh}$ and with $M = n/2$ and $\mu = 0$ if n is even, or $M = (n-1)/2$ and $\mu = 1$ if n is odd.

Example 3.5 Let $f(x)$ be any function and consider the nodes $x_j = jh$ with $h = 2\pi/3$, for $j = 0,1,2$. That is, $x_0 = 0$, $x_1 = 2\pi/3$, $x_2 = 4\pi/3$ and $n = 2$. Then $\mu = 0$ and $k = -1,0,1$,

$$c_k = \frac{1}{3}\big(f(0) + f(\frac{2\pi}{3})e^{-ik\frac{2\pi}{3}} + f(\frac{4\pi}{3})e^{-ik\frac{4\pi}{3}}\big),$$

therefore

$$\begin{aligned}
c_{-1} &= \frac{1}{3}\big(f(0) + f(\frac{2\pi}{3})e^{i\frac{2\pi}{3}} + f(\frac{4\pi}{3})e^{i\frac{4\pi}{3}}\big)\\
c_0 &= \frac{1}{3}\big(f(0) + f(\frac{2\pi}{3}) + f(\frac{4\pi}{3})\big),\\
c_1 &= \frac{1}{3}\big(f(0) + f(\frac{2\pi}{3})e^{-i\frac{2\pi}{3}} + f(\frac{4\pi}{3})e^{-i\frac{4\pi}{3}}\big)
\end{aligned}$$

Hence,

$$\begin{aligned}
\tilde{f}(x) = \sum_{k=-1}^{1} c_k e^{ikx} &= \frac{1}{3}\Big[\big(f(0) + f(\frac{2\pi}{3})e^{i\frac{2\pi}{3}} + f(\frac{4\pi}{3})e^{i\frac{4\pi}{3}}\big)e^{-ix} + \big(f(0) + f(\frac{2\pi}{3}) + f(\frac{4\pi}{3})\big)\\
&\quad + \big(f(0) + f(\frac{2\pi}{3})e^{-i\frac{2\pi}{3}} + f(\frac{4\pi}{3})e^{-i\frac{4\pi}{3}}\big)e^{ix}\Big]\\
&= \frac{1}{3}\Big[f(0)\big(1 + e^{-ix} + e^{ix}\big) + f(\frac{2\pi}{3})\big(1 + e^{-i(x-\frac{2\pi}{3})} + e^{i(x-\frac{2\pi}{3})}\big)\\
&\quad + f(\frac{4\pi}{3})\big(1 + e^{-i(x-\frac{4\pi}{3})} + e^{i(x-\frac{4\pi}{3})}\big)\Big].
\end{aligned}$$

Using the trigonometric identities, we finally deduce

$$\tilde{f}(x) = \frac{1}{3}\Big[f(0)\big(1 + 2\cos(x)\big) + f(\frac{2\pi}{3})\big(1 + 2\cos(x - \frac{2\pi}{3})\big) + f(\frac{4\pi}{3})\big(1 + 2\cos((x - \frac{4\pi}{3})\big)\Big].$$

□

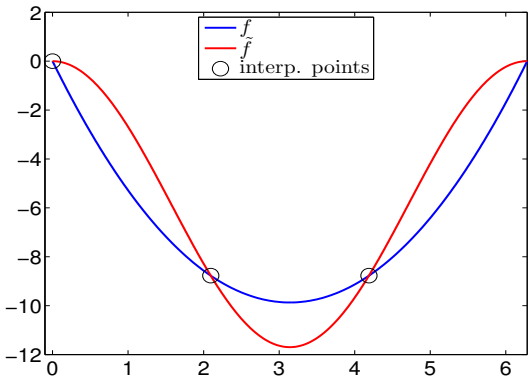


Figure 3.5: The function $f(x) = x(x-2\pi)$ and its interpolant.

5 Approximation by the least squares method

We have seen that the Lagrange interpolation does not guarantee a better approximation when the degree of the polynomial interpolant increases. This problem may be addressed by the composed interpolation, such as piecewise polynomial interpolation. However, none of them is useful to extrapolate information from the given data, that is, to generate new data value in points outside the interval to which the interpolation nodes belong.

For this task, we shall use the approximation methods, in which the interpolation condition $\tilde{f}(x_j) = f(x_j)$ is not necessarily satisfied.

Let us suppose that some data $\{(x_i, y_i), i = 0, \ldots, n\}$ is given, where y_i could represent the values $f(x_i)$ of some function f in the nodes x_i. For a given integer number $m \geq 1$ (usually, $m \ll n$) we look for a polynomial $\tilde{f}$ of degree m (and write $\tilde{f} \in \mathcal{P}_m$) satisfying the inequality

$$\sum_{i=0}^{n} |y_i - \tilde{f}(x_i)|^2 \leq \sum_{i=0}^{n} |y_i - p_m|^2,$$

for all polynomial $p_m \in \mathcal{P}_m$. If it does exist, $\tilde{f}$ is called the *least squares approximation* in $\mathcal{P}_m$ of the data set $\{(x_i, y_i), i = 0, \ldots, n\}$. Observe that, unless $m \geq n$, it is not possible to guarantee that $\tilde{f}(x_i) = y_i$ for all $i = 0, \ldots, n$.

Setting

$$\tilde{f}(x) = a_0 + a_1 x + \ldots + a_m x^m,$$

where the coefficients $a_0, \ldots, a_m$ are unknown, the problem may be formulated as follows: find $a_0, a_1, \ldots, a_m$ such that

$$\Phi(a_0, a_1, \ldots, a_m) = \min_{\{b_i, i=0,\ldots,m\}} \Phi(b_0, b_1, \ldots, b_m),$$

where

$$\Phi(b_0, b_1, \ldots, b_m) = \sum_{i=0}^{n} |y_i - (b_0 + b_1 x_i + \ldots + b_m x_i^m)|^2,$$

which is a minimization problem that can be handled by the usual techniques of differential calculus.

Let us solve the problem for the case $m = 1$, i. e., for a linear approximation polynomial (linear regression, in Statistics terminology). In this case, we have

$$\Phi(b_0, b_1) = \sum_{i=0}^{n} \Big(y_i^2 + b_0^2 + b_1^2 x_i^2 + 2b_0 b_1 x_i - 2b_0 y_i - 2b_1 x_i y_i \Big).$$

The point (a_0, a_1) in which Φ attains its minimum is determined by

$$\frac{\partial \Phi}{\partial b_0}(a_0, a_1) = 0, \quad \frac{\partial \Phi}{\partial b_1}(a_0, a_1) = 0.$$

Computing these partial derivatives we obtain the conditions

$$\sum_{i=0}^{n} \big(a_0 + a_1 x_i - y_i \big) = 0, \quad \sum_{i=0}^{n} \big(a_0 x_i + a_1 x_i^2 - x_i y_i \big) = 0,$$

which can be reordered as

$$\begin{aligned} a_0 (n+1) + a_1 \sum_{i=0}^{n} x_i &= \sum_{i=0}^{n} y_i, \\ a_0 \sum_{i=0}^{n} x_i + a_1 \sum_{i=0}^{n} x_i^2 &= \sum_{i=0}^{n} x_i y_i. \end{aligned}$$

This linear system of two equations with two unknowns has the solution

$$\begin{aligned} a_0 &= \frac{1}{D} \Big(\sum_{i=0}^{n} y_i \sum_{j=0}^{n} x_j^2 - \sum_{j=0}^{n} x_j \sum_{i=0}^{n} x_i y_i \Big), \\ a_1 &= \frac{1}{D} \Big((n+1) \sum_{i=0}^{n} x_i y_i - \sum_{j=0}^{n} x_j \sum_{i=0}^{n} y_i \Big), \end{aligned}$$

where $D = (n+1) \sum_{i=0}^{n} x_i^2 - \Big(\sum_{i=0}^{n} x_i \Big)^2$. This is the *least squares line* or *regression line*, $\tilde{f}(x) = a_0 + a_1 x$, which is the best approximation by a straight line, in the least squares sense, of the given data.

Example 3.6 Suppose that the execution time, t, of a code depends on an input parameter, j. Running the code, we obtain the following data:

j	10	15	25	50	100
t	1	1.2	2	3.5	6

Applying the above calculations, we obtain the regression line

$$\tilde{f}(t) = 0.5015 + 0.056t,$$

which allows us to extrapolate the execution times for other j-values.

□

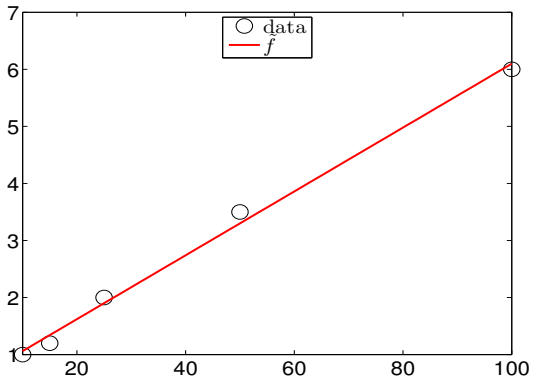


Figure 3.6: Regression line of the experimental data.

6 Approximation by orthogonal basis

In this section we shall deal with the case in which the function to approximate, f, is known in the whole interval $[a,b]$, and not simply in some of its points. Our aim is, given a function f which could have a complicated expression, produce another *similar* function $\tilde{f}$ with a simpler expression, such as a polynomial or a trigonometric function.

Like in Linear Algebra, in the theory of functions we may introduce spaces of functions, scalar products (and hence distances and orthogonality relations), basis for such spaces, etc. In this context, given two functions $f,g:[a,b]\to\mathbb{R}$, we shall use the scalar product

$$< f,g >= \int_a^b f(x)g(x)dx.$$

6.1 Approximation with Legendre polynomials

Let us start with an example. The space of polynomials of degree up to two defined in the interval $[-1,1]$ is

$$\mathcal{P}_2 = \{p(x) = a_0 + a_1x + a_2x^2 : a_0,a_1,a_2 \in \mathbb{R}, \quad x \in [-1,1]\}.$$

Obviously, any of these polynomials may be written as a unique linear combination of the polynomials

$$p_0(x) = 1, \quad p_1(x) = x, \quad p_2(x) = x^2.$$

Indeed, we just write $p(x) = a_0p_0(x) + a_1p_1(x) + a_2p_2(x)$ for whatever the values of a_0, a_1, and a_2. As a consequence,

$$\mathcal{B}_2 = \{p_0(x), p_1(x), p_2(x)\}$$

is a basis of $\mathcal{P}_2$. Like in Linear Algebra, when using orthogonal basis, we would like to find a decomposition of the type

$$p(x) = \frac{< p,p_0 >}{< p_0,p_0 >}p_0(x) + \frac{< p,p_1 >}{< p_1,p_1 >}p_1(x) + \frac{< p,p_2 >}{< p_2,p_2 >}p_2(x), \tag{3.20}$$

which, by now, is not possible since the basis $\mathcal{B}_2$ is not orthogonal. For example, we have

$$< p_0,p_2 >= \int_{-1}^1 x^2dx = \frac{2}{3} \neq 0.$$

However, we may orthogonalize [3] the basis $\mathcal{B}_2$, getting in our example

$$\{p_0(x) = 1, p_1(x) = x, p_2(x) = \frac{3x^2-1}{2}\}, \tag{3.21}$$

so, now, the decomposition (3.20) applies. Let us check it. One one hand,

$$\begin{aligned}
< p, p_0 >&= \int_{-1}^{1} (a_0 + a_1 x + a_2 x^2) dx = 2a_0 + \frac{2a_2}{3},\\
< p, p_1 >&= \int_{-1}^{1} (a_0 + a_1 x + a_2 x^2) x dx = \frac{2a_1}{3},\\
< p, p_2 >&= \int_{-1}^{1} (a_0 + a_1 x + a_2 x^2) \frac{3x^2-1}{2} dx = \frac{8a_2}{30}.
\end{aligned}$$

On the other hand, it is easy to see that

$$< p_0, p_0 >= 2, \quad < p_1, p_1 >= \frac{2}{3}, \quad < p_2, p_2 >= \frac{2}{5},$$

and therefore

$$\begin{aligned}
\frac{< p, p_0 >}{< p_0, p_0 >} p_0(x) + \frac{< p, p_1 >}{< p_1, p_1 >} p_1(x) + \frac{< p, p_2 >}{< p_2, p_2 >} p_2(x) =& a_0 + \frac{a_2}{3} + a_1 x\\
&+ \frac{2a_2}{3} \frac{3x^2-1}{2} = p(x).
\end{aligned}$$

Orthogonal polynomials of the basis given in (3.21) are called *Legendre polynomials* of order two. In general, the degree n Legendre polynomials are defined by the formula

$$L_n(x) = (-1)^n \frac{1}{n! 2^n} \frac{d^n}{dx^n} (x^2-1)^n, \quad n = 1, 2, \ldots,$$

with $L_0(x) = 1$, and satisfy

$$< L_n, L_n >= \int_{-1}^{1} L_n(x)^2 dx = \frac{2}{2n+1}.$$

Moreover, they can be recursively obtained by means of the formula

$$L_{n+1}(x) = \frac{2n+1}{n+1} x L_n(x) - \frac{n}{n+1} L_{n-1}(x), \quad n = 1, 2, \ldots,$$

with $L_0(x) = 1$ and $L_1(x) = x$.

Summarizing, any polynomial $p(x)$, of degree lower or equal than n and defined in the interval $[-1, 1]$ admits a decomposition in terms of the basis

$$\mathcal{L}_n = \{L_0(x), L_1(x), \ldots, L_n(x)\}$$

through the formula

$$p(x) = \sum_{j=0}^{n} \frac{< p, L_j >}{< L_j, L_j >} L_j(x).$$

[3] A basis may be always orthogonalized by the Gram-Schmidt procedure.

Similarly, any function $f:[-1,1]\to\mathbb{R}$ may be *approximated* in terms of Legendre polynomials by means of the expression

$$f(x)\approx\tilde{f}(x)=\sum_{j=0}^{n}\frac{<f,L_j>}{<L_j,L_j>}L_j(x), \tag{3.22}$$

where $\tilde{f}(x)$is the polynomial approximating $f(x)$.

In fact, if the function f satisfies certain regularity conditions, the infinite polynomial series is an alternative representation of such function, that is

$$f(x)=\lim_{n\to\infty}\sum_{j=0}^{n}\frac{<f,L_j>}{<L_j,L_j>}L_j(x).$$

Finally, let us observe that if the function to be approximated is defined in an interval different to $[-1,1]$, we may always introduce a change of variables to move it to such interval. Indeed, if $f:[a,b]\to\mathbb{R}$, and $x\in[a,b]$, we introduce the change

$$t=-1+2\frac{x-a}{b-a}\to x=a+\frac{b-a}{2}(t+1),$$

so now the corresponding function $g(t)=f(a+\frac{b-a}{2}(t+1))$ is defined in $[-1,1]$. Then, if the Legendre approximation is given by $\tilde{g}(t)$, that of f is given by $\tilde{f}(x)=\tilde{g}(-1+2\frac{x-a}{b-a})$.

Example 3.7 Consider the exponential function, $f(x)=e^x$ and let us find its approximation by Legendre polynomials of degree two. We have

$$\begin{aligned}
<f,L_0>&=\int_{-1}^{1}e^x dx=e-\frac{1}{e},\\
<f,L_1>&=\int_{-1}^{1}e^x x dx=\frac{2}{e},\\
<f,L_2>&=\int_{-1}^{1}e^x\frac{3x^2-1}{2}dx=e-\frac{7}{e}.
\end{aligned}$$

Then

$$\begin{aligned}
e^x\approx\frac{e-\frac{1}{e}}{2}L_0(x)+\frac{3}{e}L_1(x)+\left(e-\frac{7}{e}\right)\frac{5}{2}L_2(x)&=\frac{e^2-1}{2e}+\frac{3}{e}x+\frac{5(e^2-7)}{2e}\frac{3x^2-1}{2}\\
&=\frac{33-3e^2}{4e}+\frac{3}{e}x+\frac{15(e^2-7)}{4e}x^2.
\end{aligned}$$

□

6.2 Approximation with Fourier series

The idea of the previous section of approximating complicated functions by a linear combination of simpler functions is not limited to the consideration of polynomials. The most important example of non-polynomial functions defining a orthogonal basis are the trigonometric functions.

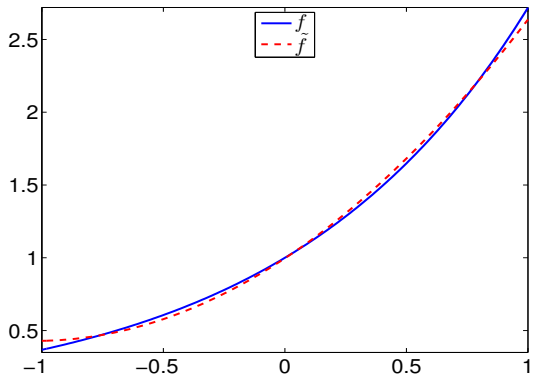


Figure 3.7: Function f and its approximation.

The *Fourier basis* of functions defined in the interval $[0,2\pi]$ is given by

$$\mathcal{F} = \{1, \sin(x), \cos(x), \sin(2x), \cos(2x), \ldots, \sin(nx), \cos(nx), \ldots\},$$

which can be written, using the exponential notation, as

$$\mathcal{F} = \{e^{inx}\}_{n=-\infty}^{n=\infty}.$$

It is easy to see that this basis is orthogonal with respect to the scalar product

$$< f, g >= \int_0^{2\pi} f(x)\bar{g}(x)dx,$$

where $\bar{z}$ denotes the conjugate[4] of the complex number z. Indeed, let us introduce the notation $\phi_n(x) = e^{inx}$ and compute the scalar product of two different elements of the basis ($n \neq m$)

$$\begin{aligned} < \phi_n, \phi_m >= \int_0^{2\pi} e^{inx}e^{-imx}dx = \int_0^{2\pi} e^{i(n-m)x}dx = \frac{1}{i(n-m)}\, e^{i(n-m)x}\Big|_0^{2\pi} \\ = \frac{1}{i(n-m)}\big(\cos((n-m)2\pi) + i\sin((n-m)2\pi) - \cos(0) + i\sin(0)\big) \\ = \frac{1}{i(n-m)}(1-1) = 0. \end{aligned}$$

On the other hand, if $n = m$, we have

$$< \phi_n, \phi_n >= \int_0^{2\pi} e^{inx}e^{-inx}dx = \int_0^{2\pi} 1dx = 2\pi.$$

Therefore, given a periodic functions of period [5] 2π, $f : [0,2\pi] \to \mathbb{R}$, we may consider an expression similar to (3.22) for the first $2M+1$ elements of the basis $\mathcal{F}$,

$$\tilde{f}(x) = \frac{1}{2\pi}\sum_{n=-M}^{M} < f, \phi_n > \phi_n(x),$$

[4] Recall that if $z = a + bi$, then $\bar{z} = a - bi$, and if $z = e^{ai}$ then $\bar{z} = e^{-ai}$.

[5] If the period is different, for instance T, the change of variable $x = 2\pi t/T$ renders the function to 2π-periodic.

where we used that $< \phi_n, \phi_n >= 2\pi$. Like for the Legendre polynomials, the function f may be represented as the infinite series

$$f(x) = \frac{1}{2\pi} \lim_{M\to\infty} \sum_{n=-M}^{M} < f, \phi_n > \phi_n(x),$$

which is the so-called *Fourier series* of f. The coefficients

$$\hat{f}_n = \frac{1}{2\pi} < f, \phi_n >= \frac{1}{2\pi} \int_0^{2\pi} f(x) e^{-inx} dx$$

are called *Fourier coefficients* of f, so that the series may be written as

$$f(x) = \sum_{n=-\infty}^{\infty} \hat{f}_n e^{inx}.$$

Using trigonometric identities, it is also common to express this series in terms of sines and cosines

$$f(x) = \frac{a_0}{2} + \sum_{n=1}^{\infty} a_n cos(nx) + b_n \sin(nx), \tag{3.23}$$

where, for $n = 0, 1, \ldots$

$$a_n = \hat{f}_n + \hat{f}_{-n} = \frac{1}{\pi} \int_0^{2\pi} f(x) \cos(nx) dx, \tag{3.24}$$

$$b_n = i(\hat{f}_n - \hat{f}_{-n}) = \frac{1}{\pi} \int_0^{2\pi} f(x) \sin(nx) dx. \tag{3.25}$$

Example 3.8 Let us consider again the situation of the Example 3.5 (see Figure 3.5) and let us use the Fourier approximation, instead of the trigonometric interpolation, as we did in that example. We have, for $f(x) = x(x-2\pi)$

$$\hat{f}_{-1} = \frac{1}{2\pi} \int_0^{2\pi} x(x-2\pi) e^{-ix} dx = 2,$$
$$\hat{f}_0 = \frac{1}{2\pi} \int_0^{2\pi} x(x-2\pi) dx = -\frac{2\pi^2}{3},$$
$$\hat{f}_1 = \frac{1}{2\pi} \int_0^{2\pi} x(x-2\pi) e^{ix} dx = 2,$$

so

$$\tilde{f}(x) = 2(e^{-ix} + e^{ix}) - \frac{2\pi^2}{3} = 4\cos(x) - \frac{2\pi^2}{3}.$$

□

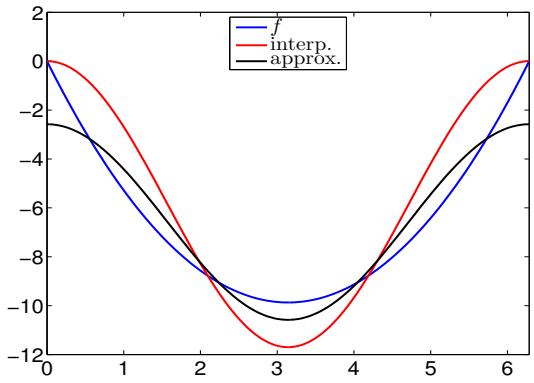


Figure 3.8: $f(x) = x(x-2\pi)$, and its trigonometric interpolant and Fourier series.

Chapter 4

Numerical differentiation and integration

In this chapter we introduce some methods for the numerical approximation of derivatives and integrals of functions. Concerning the integration, as it is well known, there exist functions which do not have an explicit representation of their primitives, while for many others the primitive have a so complicated explicit expression that their exact evaluation is not practical.

Another usual situation is when the function to be differentiated or integrated is known only at a finite number of points (not a whole interval), for instance, when the function is obtained through experimental data sampling.

In both situations it is necessary to consider numerical methods to approximate these operations, independently of the complicated form the function may have.

1 Numerical differentiation

For a function $f : (a,b) \subset \mathbb{R} \to \mathbb{R}$ continuously differentiable at a point $x \in (a,b)$, the derivative may be computed using the lateral limits

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

with $h > 0$. These expressions lead to the most basic approximations to the derivative: the *forward finite differences*, given by

$$(\delta_+ f)(x) = \frac{f(x+h)-f(x)}{h},$$

and the *backward finite differences*, given by

$$(\delta_- f)(x) = \frac{f(x)-f(x-h)}{h},$$

where $h > 0$ is a small number.

For obtaining an error estimate, we just consider the Taylor's expansion of f. If $f \in C^2(a,b)$ then

$$f(x+h) = f(x) + f'(x)h + \frac{f''(\xi)}{2}h^2,$$

where $\xi \in (x, x+h)$. We then have

$$|(\delta_+ f)(x) - f'(x)| \leq ch,$$

for some constant $c > 0$ independent of h, and therefore, the forward finite differences approximation has a first order of convergence. A similar argument gives the same result for the backward scheme.

It is possible to deduce a second order approximation having the same computational cost that the backward and forward approximations. This is the so-called *centered finite differences*, given by

$$(\delta f)(x) = \frac{f(x+h) - f(x-h)}{2h}.$$

Taylor's expansion of order three give us the identities

$$f(x+h) = f(x) + f'(x)h + \frac{f''(x)}{2}h^2 + \frac{f'''(\xi_+)}{6}h^3, \tag{4.1}$$

$$f(x-h) = f(x) - f'(x)h + \frac{f''(x)}{2}h^2 - \frac{f'''(\xi_-)}{6}h^3, \tag{4.2}$$

where $\xi_+ \in (x, x+h)$ and $\xi_- \in (x-h, x)$. Subtracting both expressions we obtain,

$$(\delta f)(x) - f'(x) = \frac{f'''(\xi_+) + f'''(\xi_-)}{12}h^2,$$

from where we deduce

$$|(\delta f)(x) - f'(x)| \leq ch^2,$$

for some constant $c > 0$ independent of h.

Normally, the numerical differentiation of a function is implemented in a uniform mesh of an interval, that is, for $x_i = a + ih$, with $h = (b-a)/n$ and i running the indices $i = 0, \ldots, n$. In this case, and for all the above schemes, the *edge problem* arises, due to the fact that the finite differences can not be computed at one or both of the interval borders. Indeed, the forward differences may not be evaluated at x_n, since we need an additional node "x_{n+1}" which, in general, is not available. Similarly, the backward differences may not be computed at x_0. Neither the centered differences at x_0 and x_n.

We resort to interpolation to solve this problem. For instance, for centered differences, which give an approximation of second order, we consider the Lagrange polynomial interpolant of degree 2 defined in the points x_0, x_1, x_2, (see Newton's formula (3.10), Chapter 3)

$$p(x) = f[x_0] + f[x_0, x_1](x - x_0) + f[x_0, x_1, x_2](x - x_0)(x - x_1).$$

Differentiating and evaluating in $x = x_0$ we obtain

$$f'(x_0) \approx p'(x_0) = f[x_0, x_1] + f[x_0, x_1, x_2](x_0 - x_1).$$

Taking into account that the mesh is uniform and replacing the divided differences expression, we deduce

$$f'(x_0) \approx \frac{f(x_1) - f(x_0)}{h} - \frac{f(x_2) - 2f(x_1) + f(x_0)}{2h} = \frac{1}{2h}\big(-3f(x_0) + 4f(x_1) - f(x_2)\big).$$

A similar argument gives

$$f'(x_n) \approx \frac{1}{2h}\big(3f(x_n) - 4f(x_{n-1}) + f(x_{n-2})\big).$$

1.1 Higher order derivatives

Computing the second derivative, or higher order derivatives, is achieved composing the previous schemes. For instance, a usual scheme for the second derivative is

$$f''(x) \approx \big(\delta_+(\delta_- f)\big)(x) = \frac{f(x+h) - 2f(x) + f(x-h)}{h^2}.$$

Error estimates for the approximation are again obtained through the Taylor's expansions given in (4.1) and (4.2), but now adding those expressions. We obtain

$$f''(x) = \frac{f(x+h) - 2f(x) + f(x-h)}{h^2} - \frac{f'''(\xi_+) - f'''(\xi_-)}{6}h,$$

from where

$$|\big(\delta_+(\delta_- f)\big)(x) - f''(x)| \leq ch,$$

that is, the approximation is linear.

1.2 Numerical differentiation of functions of several variables

The previous procedure for approximating derivatives of functions of one variables may be extended naturally to functions of several variables. Let $f : \Omega \subset \mathbb{R}^2 \to \mathbb{R}$ a continuously differentiable function and denote by (x,y) a point of Ω. The partial derivatives of f are given by

$$\frac{\partial f}{\partial x}(x,y) = \lim_{h\to 0} \frac{f(x+h,y) - f(x,y)}{h},$$
$$\frac{\partial f}{\partial y}(x,y) = \lim_{h\to 0} \frac{f(x,y+h) - f(x,y)}{h},$$

to which we may apply any of the previous finite differences schemes.

Through the partial derivatives, we define the *gradient* of f

$$\nabla f(x,y) = \Big(\frac{\partial f}{\partial x}(x,y), \frac{\partial f}{\partial y}(x,y)\Big),$$

which provides the geometrical information of steepest increase and decrease directions of f.

For a vector field, $\mathbf{F} = (F_1, F_2) : \Omega \subset \mathbb{R}^2 \to \mathbb{R}^2$, we define the *divergence* of $\mathbf{F}$ by

$$\operatorname{div} \mathbf{F}(x,y) = \frac{\partial F_1}{\partial x}(x,y) + \frac{\partial F_2}{\partial y}(x,y).$$

Here, the physical interpretation is related to the measure of the difference between the outwards and inwards flow trough the surface enclosing a control volume. Therefore, if the vector field has *sources* the divergence is positive, and if it has *sinks* the divergence is negative.

Finally, the composition of the gradient and the divergence gives a second order operator -since it has second order derivatives-, the *Laplacian*, given by

$$\Delta f(x,y) = \operatorname{div} \nabla f(x,y) = \frac{\partial^2 f}{\partial x^2}(x,y) + \frac{\partial^2 f}{\partial y^2}(x,y).$$

Let us show with an example how to compute the numerical approximations of these differential operators. Let $\Omega = (a,b) \times (c,d)$, and consider the meshes of the intervals (a,b) and (c,d) given by, respectively,

$$x_i = a + ih, \quad \text{with } h = \frac{b-a}{n}, \quad i = 0, \ldots, n$$
$$y_j = c + jh, \quad \text{with } h = \frac{d-c}{m}, \quad j = 0, \ldots, m.$$

Observe that, for simplicity, we assumed $(b-a)/n = (d-c)/m$. In general, the mesh step lengths, denoted by h_x and h_y, may be different.

From these one-dimensional meshes we build a two-dimensional mesh for the rectangle Ω, given simply by the points (x_i, y_j), $i = 0, \ldots, n$, $j = 0, \ldots, m$.

Now, the forward finite differences approximation is

$$\nabla f(x_i, y_j) \approx \frac{1}{h}\big(f(x_{i+1}, y_j) - f(x_i, y_j), f(x_i, y_{j+1}) - f(x_i, y_j)\big),$$
$$\operatorname{div} \mathbf{F}(x_i, y_j) \approx \frac{1}{h}\big(F_1(x_{i+1}, y_j) - F_1(x_i, y_j) + F_2(x_i, y_{j+1}) - F_2(x_i, y_j)\big).$$

Observe the border problem at the upper border. A combination of forward and backward differences lead us to

$$\Delta f(x_i, y_j) = \frac{1}{h^2}\big(f(x_{i+1}, y_j) + f(x_{i-1}, y_j) + f(x_i, y_{j+1}) + f(x_i, y_{j-1}) - 4f(x_i, y_j)\big),$$

with a border problem in all the borders.

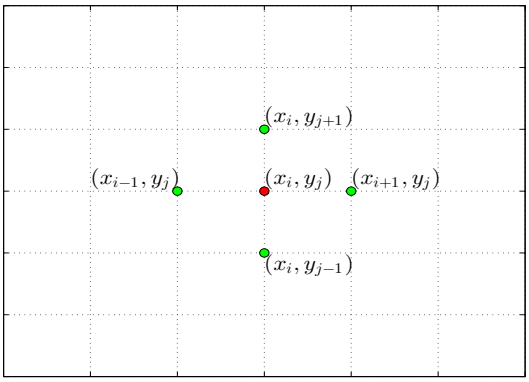


Figure 4.1: Nodes involved in the Laplacian discretization.

1.3 Approximation of differential equations

A *differential equation* is an equation involving derivatives of a function. In the usual situation, the function itself is unknow, and the equation is used to determine it. For instance,

$$f'(t) = 3t^2 + 1, \quad t > 0, \tag{4.3}$$

has the solution $f(t) = t^3 + t + c$, where c is a constant of integration. To fully determine f, a *initial condition* is normally provided, e.g. $f(0) = 2$. With this condition, the constant of integration is fixed $c = 2$.

For a differential equation like (4.3), finding the solution $f(t)$ is equivalent to finding a primitive of its right hand side. Since, on one hand, primitives of functions are hard to compute, and on the other, differential equations more complicated than (4.3) usually arise in applications, approximated methods are needed to solve this type of problems.

The usual approach is to use forward or backward finite differences to discretize the problem. If using forward differences, equation (4.3) is replaced by the so-called *explicit Euler method*,

$$f(t_{i+1}) = f(t_i) + \tau(3t_i^2 + 1), \quad i = 0, 1, \ldots$$

where $t_i = i\tau$, and $\tau > 0$ is small. While, f using backward differences, equation (4.3) is replaced by the *implicit Euler method*,

$$f(t_i) = f(t_{i-1}) + \tau(3t_i^2 + 1), \quad i = 1, 2, \ldots$$

Example 4.1 Linear differential equation.

The problem is: Given $a \in \mathbb{R}$, $g : [0,T] \to \mathbb{R}$, and $f(0) = f_0 \in \mathbb{R}$, find $f : [0,T] \to \mathbb{R}$ such that

$$f'(t) = af(t) + g(t), \quad t \in (0,T].$$

For instance, if $g \equiv 0$, then $f(t) = e^{at} f_0$. The explicit Euler method for this equation takes the form, for $i = 0, 1, \ldots$

$$\begin{aligned} f(t_{i+1}) =& f(t_i) + \tau(af(t_i) + g(t_i)), \\ =& (1 + a\tau) f(t_i) + \tau g(t_i). \end{aligned}$$

□

Example 4.2 General differential equation of first order.

The problem is: Given $F : [0,T] \times \mathbb{R} \to \mathbb{R}$, and $f(0) = f_0 \in \mathbb{R}$, find $f : [0,T] \to \mathbb{R}$ such that

$$f'(t) = F(t, f(t)), \quad t \in (0,T].$$

For instance, if $F(t,x) = ax$, then, like in the previous example, $f(t) = e^{at} f_0$. The explicit Euler scheme is, for $i = 0, 1, \ldots$

$$f(t_{i+1}) = f(t_i) + \tau F(t_i, f(t_i)). \tag{4.4}$$

□

2 Numerical integration

In this section we introduce some classical formulas for the numerical integration of one-dimensional continuous functions, $f : (a,b) \to \mathbb{R}$. For the sake of brevity, we shall write

$$I(f) = \int_a^b f(x)dx.$$

Integration formulas for approximating $I(f)$ are called *simple* if the approximation takes place in the whole interval (a,b), and *composite* if, before the application of the formula, we split the interval (a,b) in a given number, n, of subintervals

$$I_i = [x_i, x_{i+1}], \quad \text{with } i = 0, \ldots, n-1,$$

where $x_i = a + ih$, for $i = 0, \ldots, n$, and $h = \frac{b-a}{n}$. We use that

$$I(f) = \sum_{i=0}^{n-1} \int_{I_i} f(x)dx,$$

and then we apply the approximation formula in each subinterval.

Two criterion are used to measure the approximation quality. If the formula is simple, we say that its *degree of accuracy* is r if for any polynomial of degree r, $p_r(x)$, the result of using the approximation formula is the exact value of $I(p_r)$.

For composite formulas, the usual criterion of order of convergence (also termed *approximation order*) is used, taken with respect to the subintervals size.

2.1 Middle point formula

The middle point formula is the simplest formula. We approximate the value of f in (a,b) by its middle point value,

$$I_{mp}(f) = (b-a) f\Big(\frac{a+b}{2}\Big),$$

where *mp* stands for *middle point*.

For an error estimate, we use Taylor's expansion. Assuming that f is once continuously differentiable in (a,b), we get

$$f(x) = f\Big(\frac{a+b}{2}\Big) + f'\Big(\frac{a+b}{2}\Big)\Big(x - \frac{a+b}{2}\Big) + \frac{f''(\xi)}{2}\Big(x - \frac{a+b}{2}\Big)^2,$$

with $\xi \in (a,b)$. Then

$$\begin{aligned} I(f) &= I_{mp}(f) + f'\Big(\frac{a+b}{2}\Big)\int_a^b \Big(x - \frac{a+b}{2}\Big)dx + \frac{f''(\xi)}{2}\int_a^b \Big(x - \frac{a+b}{2}\Big)^2 dx \\ &= I_{mp}(f) + \frac{f''(\xi)}{24}(b-a)^3. \end{aligned} \tag{4.5}$$

Therefore, since the estimate depends upon the second derivative of f, we deduce that the formula has an accuracy degree $r = 1$.

The corresponding composite formula is

$$I^c_{mp}(f) = h \sum_{i=0}^{n-1} f\Big(\frac{x_i + x_{i+1}}{2}\Big), \tag{4.6}$$

where c means *composite*. Using an argument like (4.5) we deduce

$$I(f) - I^c_{mp}(f) = \frac{b-a}{24} f''(\xi) h^2,$$

where $\xi \in (a,b)$, and therefore, the approximation order is quadratic.

2.2 Trapezoidal formula

It is obtained approximating the function by the Lagrange polynomial interpolant of order 1. Thus,

$$I_t(f) = \int_a^b \Big(f(a) + \frac{f(b)-f(a)}{b-a}(x-a) \Big) dx = \frac{b-a}{2}\big(f(a)+f(b)\big).$$

The error is

$$I(f) - I_t(f) = -\frac{(b-a)^3}{12} f''(\xi),$$

where $\xi \in (a,b)$. The degree of accuracy is then $r = 1$, like for the middle point formula.

The corresponding composite formula is given by

$$I_t^c(f) = \frac{h}{2} \sum_{i=0}^{n-1} \big(f(x_i) + f(x_{i+1}) \big), \tag{4.7}$$

and like for the middle point formula, the approximation order is quadratic:

$$I(f) - I_t^c(f) = -\frac{b-a}{12} f''(\xi) h^2,$$

where $\xi \in (a,b)$.

2.3 Formula of Simpson

It is obtained approximating the function by the Lagrange polynomial interpolant of order 2. The formula is

$$I_s(f) = \frac{b-a}{6}\big(f(a) + 4f\Big(\frac{a+b}{2}\Big) + f(b)\big).$$

The error is

$$I(f) - I_s(f) = -\frac{1}{16}\frac{(b-a)^5}{180} f^{(4)}(\xi),$$

where $\xi \in (a,b)$. Thus, the degree of accuracy of Simpson's formula is $r = 3$

The corresponding composite formula is given by

$$I_s^c(f) = \frac{h}{6} \sum_{i=0}^{n-1} \big(f(x_i) + 4f\Big(\frac{x_i + x_{i+1}}{2}\Big) + f(x_{i+1}) \big), \tag{4.8}$$

and using Taylor's expansion we readily see that the approximation order if four:

$$I(f) - I_s^c(f) = -\frac{b-a}{2880} f^{(4)}(\xi) h^4,$$

where $\xi \in (a,b)$.

2.4 Higher order formulas

The previous formulas for numerical integration to approximate $I(f)$ use Lagrange polynomial interpolants of different degree to approximate the function, and then integrate exactly these polynomials.

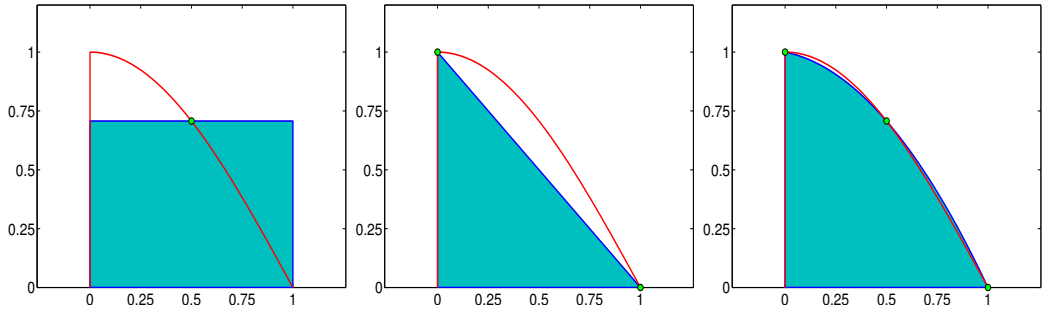


Figure 4.2: Middle point (left), trapezoidal (center), and Simpson (right).

n	$\{\bar{x}_i\}$	$\{\bar{\alpha}_i\}$
1	$\{\pm 1/\sqrt{3}\}$	$\{1\}$
2	$\{\pm\sqrt{15}/5, 0\}$	$\{5/9, 8/9\}$
3	$\left\{\pm(1/35)\sqrt{525-70\sqrt{30}},\right.$ $\left.\pm(1/35)\sqrt{525+70\sqrt{30}}\right\}$	$\left\{(1/36)(18+\sqrt{30}),\right.$ $\left.(1/36)(18-\sqrt{30}\right\}$
4	$\left\{0, \pm(1/21)\sqrt{245-14\sqrt{70}},\right.$ $\left.\pm(1/21)\sqrt{245+14\sqrt{70}}\right\}$	$\left\{128/225, (1/900)(322+13\sqrt{70},\right.$ $\left.(1/900)(322-13\sqrt{70}\right\}$

Table 4.1: Nodes and weights for the Gauss formula for the first values of n.

In general, we may define the approximation

$$I_{app}(f) = \int_a^b \Pi_n f(x)dx,$$

where $\Pi_n f$ is the Lagrange polynomial interpolant of degree n in the nodes of a given mesh , x_i, $i = 0, \ldots, n-1$. Computing this integral, we obtain

$$I_{app}(f) = \sum_{i=0}^{n} \alpha_i f(x_i),$$

where

$$\alpha_i = \int_a^b \ell_i(x)dx, \quad i = 0, \ldots, n,$$

being ℓ_i the $i-$th Lagrange fundamental polynomial of degree n, as introduced in (3.3). Thus, the approximation will have an accuracy degree of, at least, $r = n$.

2.5 Formula of Gauss

Inspired by the expression

$$I_{app}(f) = \sum_{i=0}^{n} \alpha_i f(x_i), \tag{4.9}$$

we may inquire if there exist choices of the *weights*, α_i, and of the nodes, x_i, such that the corresponding accuracy degree is higher than the given by Lagrange interpolants.

To simplify the exposition, we shall restrict ourselves to the interval $(-1,1)$, having on mind that, once the nodes $\bar{x}_i$ and the weights $\bar{\alpha}_i$ are found relative to this interval, we may change to a generic interval (a,b) by means of the change of variables

$$x_i = \frac{a+b}{2} + \frac{b-a}{2}\bar{x}_i, \quad \alpha_i = \frac{b-a}{2}\bar{\alpha}_i.$$

The answer to the above question is provided by Legendre polynomials of degree up to $n+1$, already introduced in Subsection 6.1 of Chapter 3.

It may be proven that the highest accuracy degree for the approximation (4.9) is $r = 2n+1$, and that it can be obtained by the *formula of Gauss*, with nodes and weights determined as follows

$$\begin{cases} \bar{x}_i = \text{zeros of } L_{n+1}(x), \\ \\ \bar{\alpha}_i = \dfrac{2}{(1-\bar{x}_i^2)\big(L'_{n+1}(\bar{x}_i)\big)^2}, \quad i = 0,\ldots,n. \end{cases}$$

The weights are all positive, and the nodes belong to the interval $(-1,1)$. Table 2.5 gives these nodes and weights for the cases $n = 1,2,3,4$.

If f is $2n+2$ times continuously differentiable, then the error of the approximation is given by

$$I(f) - I_g(f) = \frac{2^{2n+3}((n+1)!)^4}{(2n+3)((2n+2)!)^3} f^{(2n+2)}(\xi),$$

where $\xi \in (-1,1)$.

Example 4.3 We integrate the function $f(x) = \sin(x)$ in the interval $[0,\pi]$, whose exact results is $I(f) = 2$. For the middle point, trapezoidal and Simpson's formula, we use the composite versions, with $n = 20$. For the Gauss formula, we just take five points, corresponding to the zeros of the Legendre polynomial of degree 5 ($n = 4$, in Table 2.5). The following table shows the absolute error of each approximation.

Method	Middle point	Trapezoidal	Simpson	Gauss
Abs. error	2.0576e-03	4.1140e-03	4.2309e-07	1.1028e-07

□

Chapter 5

Systems of linear equations

Our objective in this chapter is to devise methods, exact or approximate, to find the solutions to linear systems of equations having the same number of equations than of unknowns. The problem is, given the numbers a_{ij} and b_j for $i, j = 1, 2, \ldots, n$ find the numbers $x_1, x_2, \ldots, x_n$ satisfying the n linear equations

$$\begin{array}{rcl} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n & = & b_1, \\ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n & = & b_2, \\ \vdots & & \vdots \\ a_{n1}x_1 + a_{n2}x_2 + \ldots + a_{nn}x_n & = & b_n. \end{array}$$

Here, $A = (a_{ij})_{i,j=1}^n$ is the coefficient matrix, $\mathbf{b} = (b_i)_{i=1}^n$ is the independent term, and $\mathbf{x} = (x_i)_{i=1}^n$ is the vector of unknowns. Using matrix notation, the system takes the form

$$A\mathbf{x} = \mathbf{b}.$$

Numerical methods to solve linear systems may be classified in two main classes: direct methods and iterative methods.

Direct methods compute the solution in a finite number of steps, if an infinite precision arithmetic is used. In practice, a finite precision arithmetic is normally used, introducing rounding errors which may greatly affect to the solution. Direct methods are useful to solve small systems of equations or large unstructured systems. The basic methods of this type are Gauss method, Gauss-Jordan method and the related LU factorization.

Iterative methods define a sequence of approximate solutions converging to the exact solution. In this case, in addition to rounding errors, truncation errors due to the realization of a finite number of iterations, arise. These methods are specially useful when the system is large and the coefficient matrix has a suitable structure allowing to certain simplifications or approximations. The basic methods of this type are the method of Jacobi and the method of Gauss-Seidel.

1 Direct methods

1.1 The method of Gauss

Gauss method consists on transforming the original system to obtain another in which the coefficient matrix is upper triangular. This is done by suitable linear combinations of the system

equations, which do not alter the solution of the system.

In this transformation, only the coefficient matrix and the independent vector play a role. We introduce the *extended* matrix

$$[A|b] = \begin{pmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1n} & b_1 \\ a_{21} & a_{22} & a_{23} & \dots & a_{2n} & b_2 \\ a_{31} & a_{32} & a_{33} & \dots & a_{3n} & b_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \dots & a_{nn} & b_n \end{pmatrix}$$

The method has two main steps:

1. *Triangulation.* The equivalent system is obtained operating on the rows to produce zeros under the main diagonal, by the linear combinations

   $$r_i \to r_i + \lambda r_j, \quad j \neq i,$$

   where r_i is the i-th row. A variant of the method uses the so-called *pivoting* technique, in which the position of rows may be also interchanged,

   $$r_i \leftrightarrow r_j.$$

   Once the matrix has been rendered to the upper triangular form, we get a system of the type

   $$U\mathbf{x} = \mathbf{b}'$$

   where U has the form

   $$U = \begin{pmatrix} u_{11} & u_{12} & u_{13} & \dots & u_{1n} \\ 0 & u_{22} & u_{23} & \dots & u_{2n} \\ 0 & 0 & u_{33} & \dots & u_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & u_{nn} \end{pmatrix}.$$

2. *Backward substitution.* The i-th system equation, for $i = 1, 2, \dots, n$, is given by

   $$u_{ii}x_i + u_{ii+1}x_{i+1} + \cdots + u_{in}x_n = b'_i.$$

   Since we are assuming $\det(A) \neq 0$, and we know that the linear transformations leave the determinant invariant, we have that

   $$\Pi_{i=1}^{n} u_{ii} = \det(U) = \det(A) \neq 0,$$

   implying that $u_{ii} \neq 0$ for all $i = 1, \dots, n$. Then the equivalent system is easyly solved starting from the last row and proceeding upwards, that is, for $i = n, n-1, \dots, 1$, we set

   $$x_i = \frac{b'_i - u_{ii+1}x_{i+1} - \cdots - u_{in}x_n}{u_{ii}} = \frac{1}{u_{ii}}\left(b'_i - \sum_{j=i+1}^{n} u_{ij}x_j\right). \tag{5.1}$$

Example 5.1 Solve, using Gauss method, the linear system:

$$\begin{array}{rrrcl} 2x & +3y & -z & = & 5, \\ 4x & +4y & -3z & = & 3, \\ -2x & +3y & -z & = & 1. \end{array}$$

First, we triangulate the extended matrix. We start producing zeros in the first column, below the pivot $\mathbf{2}$.

$$\begin{array}{c} r_1 \\ r_2 \\ r_3 \end{array} \left(\begin{array}{cccc} \mathbf{2} & 3 & -1 & 5 \\ \mathbf{4} & 4 & -3 & 3 \\ \mathbf{-2} & 3 & -1 & 1 \end{array} \right) \begin{array}{rcl} r_1' & = & r_1 \\ r_2' & = & r_2 - \frac{\mathbf{4}}{\mathbf{2}} r_1 \\ r_3' & = & r_3 - \frac{\mathbf{-2}}{\mathbf{2}} r_1 \end{array}$$

In the next step we produce zeros in the second column, below the pivot $\mathbf{-2}$,

$$\begin{array}{c} r_1' \\ r_2' \\ r_3' \end{array} \left(\begin{array}{cccc} 2 & 3 & -1 & 5 \\ 0 & \mathbf{-2} & -1 & -7 \\ 0 & \mathbf{6} & -2 & 6 \end{array} \right) \begin{array}{rcl} r_1'' & = & r_1' \\ r_2'' & = & r_2' \\ r_3'' & = & r_3 - \frac{\mathbf{6}}{\mathbf{-2}} r_2' \end{array}$$

Thus, we obtained the upper triangular matrix

$$\begin{array}{c} r_1'' \\ r_2'' \\ r_3'' \end{array} \left(\begin{array}{cccc} 2 & 3 & -1 & 5 \\ 0 & -2 & -1 & -7 \\ 0 & 0 & -5 & -15 \end{array} \right).$$

Once the extended matrix is triangular, we apply the backward substitution to solve the system, i.e., we start solving from the last equation up. In equation form, we have

$$\begin{array}{rrrcr} 2x & +3y & -z & = & 5, \\ & -2y & -z & = & -7, \\ & & -5z & = & -15, \end{array}$$

and computing the solution is already straightforward. □

Pivoting

When triangulating, in the first transformation, we produce zeros below a_{11}. In the second step, we repeat the operation below a_{22}', and so on. These elements, a_{ii}, are the *pivots*. There are two variants of the Gauss method, according to how we deal with pivots:

- *Gauss partial pivoting*, in which rows are interchanged so as to get the element with maximum absolute value as pivot.
- *Gauss total pivoting*, where both rows and columns may be interchanged. In this case, we must pay attention to columns interchange, since it also involves the interchanging of the corresponding unknowns.

Using partial pivoting is compulsory when some element of the diagonal, a_{ii}, vanishes or is small in absolute value. The reason is that in the triangulating process we divide by the pivot some of the coefficient matrix elemnts. Of course, division by zero is undefined. But also, division by a small number should be avoided, since it may cause large rounding errors.

Example 5.2 Solve, using partial pivoting, the system

$$\begin{array}{rrrcr} x & +y & -z & = & 0, \\ 2x & +y & +z & = & 7, \\ 3x & -2y & -z & = & -4. \end{array}$$

We choose the pivot in the first column by selecting the element with largest absolute value, and produce zeros below it:

$$\begin{pmatrix} \mathbf{1} & 1 & -1 & 0 \\ \mathbf{2} & 1 & 1 & 7 \\ \mathbf{3} & -2 & -1 & -4 \end{pmatrix} \Leftrightarrow \begin{matrix} r_1 \\ r_2 \\ r_3 \end{matrix} \begin{pmatrix} \mathbf{3} & -2 & -1 & -4 \\ \mathbf{2} & 1 & 1 & 7 \\ \mathbf{1} & 1 & -1 & 0 \end{pmatrix} \begin{array}{rcl} r_1' & = & r_1 \\ r_2' & = & r_2 - \frac{\mathbf{2}}{\mathbf{3}} r_1 \\ r_3' & = & r_3 - \frac{\mathbf{1}}{\mathbf{3}} r_1 \end{array}$$

In the next step, we see that the maximum of the pivot and of the elements below it, $\max(7/3, 5/3)$, is just the pivot $\mathbf{7/3}$, so we do not need to interchange rows.

$$\begin{matrix} r_1' \\ r_2' \\ r_3' \end{matrix} \begin{pmatrix} 3 & -2 & -1 & -4 \\ 0 & \frac{\mathbf{7}}{\mathbf{3}} & \frac{5}{3} & \frac{29}{3} \\ 0 & \frac{\mathbf{5}}{\mathbf{3}} & -\frac{2}{3} & \frac{4}{3} \end{pmatrix} \begin{array}{rcl} r_1'' & = & r_1' \\ r_2'' & = & r_2' \\ r_3'' & = & r_3 - \frac{\mathbf{5/3}}{\mathbf{7/3}} r_2' \end{array}$$

Thus, we obtained the upper triangular matrix

$$\begin{matrix} r_1'' \\ r_2'' \\ r_3'' \end{matrix} \begin{pmatrix} 3 & -2 & -1 & -4 \\ 0 & \frac{7}{3} & \frac{5}{3} & \frac{29}{3} \\ 0 & 0 & -\frac{13}{7} & -\frac{39}{7} \end{pmatrix},$$

and we finish applying backward substitution. □

1.2 The method of Gauss-Jordan

We use the same ideas than in the Gauss method, but to get a diagonal system, instead of a triangular system. To do this, the same kind of operations are performed on the extended matrix. We beguin with an example.

Example 5.3 Solve the following system by the Gauss-Jordan method.

$$\begin{array}{rrrcr} 2x & +3y & -z & = & 5, \\ 4x & +4y & -3z & = & 3, \\ -2x & +3y & -z & = & 1. \end{array}$$

We write the extended matrix and divide the first row by the pivot $\mathbf{2}$.

$$\begin{matrix} r_1 \\ r_2 \\ r_3 \end{matrix} \begin{pmatrix} \mathbf{2} & 3 & -1 & 5 \\ 4 & 4 & -3 & 3 \\ -2 & 3 & -1 & 1 \end{pmatrix} \begin{array}{rcl} r_1' & = & r_1/\mathbf{2} \end{array}$$

Then we produce zeros below the pivot of the first column,

$$\begin{matrix} r_1' \\ r_2 \\ r_3 \end{matrix} \begin{pmatrix} 1 & \frac{3}{2} & -\frac{1}{2} & \frac{5}{2} \\ 4 & 4 & -3 & 3 \\ -2 & 3 & -1 & 1 \end{pmatrix} \begin{array}{rcl} r_1' & & \\ r_2' & = & r_2 - 4r_1' \\ r_3' & = & r_3 - (-2)r_1' \end{array}$$

We repeat the operation with the second row, dividing by the pivot -2,

$$\begin{array}{c} r_1' \\ r_2' \\ r_3' \end{array} \left(\begin{array}{cccc} 1 & \frac{3}{2} & -\frac{1}{2} & \frac{5}{2} \\ 0 & -2 & -1 & -7 \\ 0 & 6 & -2 & 6 \end{array} \right) \begin{array}{l} r_2'' \\ \\ \\ \end{array} = r_2'/(-2)$$

and then produce zeros below and above the pivot,

$$\begin{array}{c} r_1' \\ r_2'' \\ r_3' \end{array} \left(\begin{array}{cccc} 1 & \frac{3}{2} & -\frac{1}{2} & \frac{5}{2} \\ 0 & 1 & \frac{1}{2} & \frac{7}{2} \\ 0 & 6 & -2 & 6 \end{array} \right) \begin{array}{l} r_1'' = r_1' - (3/2) r_2'' \\ r_2'' \\ r_3'' = r_3' - 6 r_2'' \end{array}$$

Finally, we repeat these operations with the third row, dividing now by -5.

$$\begin{array}{c} r_1'' \\ r_2'' \\ r_3'' \end{array} \left(\begin{array}{cccc} 1 & 0 & -\frac{5}{4} & -\frac{11}{4} \\ 0 & 1 & \frac{1}{2} & \frac{7}{2} \\ 0 & 0 & -5 & -15 \end{array} \right) \begin{array}{l} \\ \\ r_3''' = r_3''/(-5) \end{array}$$

and produce the corresponding zeros above the pivot,

$$\begin{array}{c} r_1'' \\ r_2'' \\ r_3''' \end{array} \left(\begin{array}{cccc} 1 & 0 & -\frac{5}{4} & -\frac{11}{4} \\ 0 & 1 & \frac{1}{2} & \frac{7}{2} \\ 0 & 0 & 1 & 3 \end{array} \right) \begin{array}{l} r_1''' = r_1'' - (-5/4) r_3''' \\ r_2''' = r_2'' - (1/2) r_3''' \\ r_3''' \end{array}$$

The equivalent system is then

$$\left(\begin{array}{cccc} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 1 & 3 \end{array} \right),$$

and the solution is given by the independent term.

□

The method of Gauss-Jordan also admits the partial and total pivoting strategies. This method is specially useful for solving many systems which share the same coefficient matrix but that have different independent terms. Therefore, it is also adequate to compute the inverse of a matrix.

Computing the inverse of a matrix by the Gauss-Jordan method

If it does exist, the inverse of a square matrix, A, of order n, is another square matrix of order n, denoted by A^{-1}, which satisfies $AA^{-1} = A^{-1}A = I$, where I denotes the identity matrix (of order n, in this case).

If we denote the columns of A^{-1} by $c_1, c_2, \ldots, c_n$, and those of the identity matrix as $e_1, e_2, \ldots, e_n$, then we may write

$$A^{-1} = (c_1 c_2 \ldots c_n), \quad I = (e_1 e_2 \ldots e_n).$$

Since $AA^{-1} = I$, we have

$$A(c_1 c_2 \ldots c_n) = (e_1 e_2 \ldots e_n),$$

and rewritting as

$$Ac_1 = e_1,\ Ac_2 = e_2, \ldots,\ Ac_n = e_n$$

we see that the columns of A^{-1} are the solutions to n systems having A as the coefficient matrix, and the columns of I as independent terms. If we solve simultaneously these n systems, the solutions will be the columns of A^{-1}. We apply the Gauss-Jordan method to accomplish this task.

The procedure has the following steps:

1. Consider the matrix $n \times 2n$ given by $[A|I]$, i. e., the row concatenation of A and I.
2. Operating by rows, transform A to get I in the left hand side of the matrix $[A|I]$. Then, the resulting right hand side matrix is the inverse of A, that is, we get after the transformation the matrix $\left[I|A^{-1}\right]$.
3. Check that $AA^{-1} = I = A^{-1}A$.

Example 5.4 Compute the inverse of

$$A = \begin{pmatrix} 3 & 2 & 3 \\ 2 & 1 & 1 \\ 3 & 1 & 1 \end{pmatrix}.$$

We start writing the extended matrix $[A|I]$ and dividing the first row by the pivot **3**,

$$\begin{matrix} r_1 \\ r_2 \\ r_3 \end{matrix} \begin{pmatrix} \mathbf{3} & 2 & 3 & 1 & 0 & 0 \\ 2 & 1 & 1 & 0 & 1 & 0 \\ 3 & 1 & 1 & 0 & 0 & 1 \end{pmatrix} \begin{matrix} r'_1 \\ \\ \\ \end{matrix} \begin{matrix} = \\ \\ \\ \end{matrix} \begin{matrix} r_1/\mathbf{3} \\ \\ \\ \end{matrix}$$

Next, produce zeros below the pivot,

$$\begin{matrix} r'_1 \\ r_2 \\ r_3 \end{matrix} \begin{pmatrix} 1 & \frac{2}{3} & 1 & \frac{1}{3} & 0 & 0 \\ 2 & 1 & 1 & 0 & 1 & 0 \\ 3 & 1 & 1 & 0 & 0 & 1 \end{pmatrix} \begin{matrix} r'_1 \\ r'_2 \\ r'_3 \end{matrix} \begin{matrix} \\ = \\ = \end{matrix} \begin{matrix} \\ r_2 - 2r'_1 \\ r_3 - 3r'_1 \end{matrix}$$

Repeat for the second row, dividing by the pivot $-\frac{\mathbf{1}}{\mathbf{3}}$.

$$\begin{matrix} r'_1 \\ r'_2 \\ r'_3 \end{matrix} \begin{pmatrix} 1 & \frac{2}{3} & 1 & \frac{1}{3} & 0 & 0 \\ 0 & -\frac{\mathbf{1}}{\mathbf{3}} & -1 & -\frac{2}{3} & 1 & 0 \\ 0 & -1 & -2 & -1 & 0 & 1 \end{pmatrix} \begin{matrix} r''_2 \\ \\ \\ \end{matrix} \begin{matrix} = \\ \\ \\ \end{matrix} \begin{matrix} r'_2/(-\frac{\mathbf{1}}{\mathbf{3}}) \\ \\ \\ \end{matrix}$$

And produce zeros,

$$\begin{matrix} r'_1 \\ r''_2 \\ r'_3 \end{matrix} \begin{pmatrix} 1 & \frac{2}{3} & 1 & \frac{1}{3} & 0 & 0 \\ 0 & 1 & 3 & 2 & -3 & 0 \\ 0 & -1 & -2 & -1 & 0 & 1 \end{pmatrix} \begin{matrix} r''_1 \\ r''_2 \\ r''_3 \end{matrix} \begin{matrix} = \\ \\ = \end{matrix} \begin{matrix} r'_1 - (2/3)r''_2 \\ \\ r'_3 - (-1)r''_2 \end{matrix}$$

Repeat with the third row, producing zeros above the pivot

$$\begin{matrix} r''_1 \\ r''_2 \\ r''_3 \end{matrix} \begin{pmatrix} 1 & 0 & -1 & -1 & 2 & 0 \\ 0 & 1 & 3 & 2 & -3 & 0 \\ 0 & 0 & 1 & 1 & -3 & 1 \end{pmatrix} \begin{matrix} r'''_1 \\ r'''_2 \\ r'''_3 \end{matrix} \begin{matrix} = \\ = \\ = \end{matrix} \begin{matrix} r''_1 - (-1)r'''_3 \\ r''_2 - 3r'''_3 \\ r''_3 \end{matrix}$$

Since the left sub-matrix is the identity matrix, the procedure finishes. The resulting right sub-matrix is A^{-1}.

$$\left[I|A^{-1}\right] = \begin{pmatrix} 1 & 0 & 0 & 0 & -1 & 1 \\ 0 & 1 & 0 & -1 & 6 & -3 \\ 0 & 0 & 1 & 1 & -3 & 1 \end{pmatrix}$$

We check it,

$$AA^{-1} = \begin{pmatrix} 3 & 2 & 3 \\ 2 & 1 & 1 \\ 3 & 1 & 1 \end{pmatrix} \begin{pmatrix} 0 & -1 & 1 \\ -1 & 6 & -3 \\ 1 & -3 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} = I,$$

and

$$A^{-1}A = \begin{pmatrix} 0 & -1 & 1 \\ -1 & 6 & -3 \\ 1 & -3 & 1 \end{pmatrix} \begin{pmatrix} 3 & 2 & 3 \\ 2 & 1 & 1 \\ 3 & 1 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} = I.$$

□

1.3 LU factorization

In the LU factorization method the objective is to decompose the original coefficients matrix A into a product of an upper triangular matrix, U, and a lower triangular matrix, L, so we get

$$A = LU.$$

For some matrices, this decomposition is not possible unless we permute its rows. Then, in general, there always exists a decomposition of the type

$$PA = LU,$$

where P is a permutation matrix. In addition, if A is invertible, then it admits an LU factorization if and only if all its leading principal minors are nonzero.

Since, if it does exist, the LU factorization is not unique, the following additional condition is assumed,

$$l_{ii} = 1 \quad \text{for } i = 1, 2, \ldots, n.$$

Let us consider the system of equations

$$A\mathbf{x} = \mathbf{b},$$

and assume that A admits an LU factorization. The steps to solve this system by LU factorization are the following

1. Compute the factorization $A = LU$. Since $A\mathbf{x} = \mathbf{b}$, we get $LU\mathbf{x} = \mathbf{b}$.
2. Solve $L\mathbf{y} = \mathbf{b}$ by forward substitution, to obtain $\mathbf{y}$.
3. Solve $U\mathbf{x} = \mathbf{y}$ by backward substitution, to obtain $\mathbf{x}$.

Backward substituion was introduced in the formula (5.1) as a final step for the Gauss method. Forward substituion is a similar procedure to solve a system with a lower triangular matrix, $L = (l_{ij})$. In this case, the solution is given by

$$x_i = \frac{b_i - l_{i1}x_1 - \cdots - l_{ii-1}x_{i-1}}{l_{ii}} = \frac{1}{l_{ii}}\left(b_i - \sum_{j=1}^{i-1} l_{ij}x_j\right).$$

Example 5.5 Solve the following linear system by LU factorization.

$$\begin{array}{rrrcr} x & +y & +z & = & 1, \\ -x & +y & & = & 0, \\ & -2y & +2z & = & -4. \end{array}$$

1. *Factorization.* We use the method of Gauss. In the first step, we produce zeros below a_{11}.

$$\begin{array}{c} r_1 \\ r_2 \\ r_3 \end{array} \left(\begin{array}{rrr} \mathbf{1} & 1 & 1 \\ -1 & 1 & 0 \\ 0 & -2 & 2 \end{array}\right) \quad \begin{array}{rcl} r_1' &=& r_1 \\ r_2' &=& r_2 - (\mathbf{-1/1})\ r_1 \\ r_3' &=& r_3 - \mathbf{0/1}\ r_1 \end{array}$$

   The *multipliers* (in this example $-\mathbf{1}$ and $\mathbf{0}$), written in bold face, are the elements of L. In the new matrix we construct, we place the multipliers replacing the zeros we created in the step before. We Repeat the procedure producing zeros below the next pivot

$$\begin{array}{c} r_1' \\ r_2' \\ r_3' \end{array} \left(\begin{array}{rrr} 1 & 1 & 1 \\ \mathbf{-1} & \mathbf{2} & 1 \\ \mathbf{0} & -2 & 2 \end{array}\right) \quad \begin{array}{rcl} r_1'' &=& r_1' \\ r_2'' &=& r_2' \\ r_3'' &=& r_3' - (\mathbf{-2/2})\ r_2' \end{array}$$

   And we obtain the matrix storing simultaneously L and U.

$$\left(\begin{array}{rrr} 1 & 1 & 1 \\ -1 & 2 & 1 \\ 0 & -1 & 3 \end{array}\right).$$

   The matrices L and U are

$$L = \left(\begin{array}{rrr} 1 & 0 & 0 \\ \mathbf{-1} & 1 & 0 \\ \mathbf{0} & \mathbf{-1} & 1 \end{array}\right) \quad U = \left(\begin{array}{rrr} 1 & 1 & 1 \\ 0 & 2 & 1 \\ 0 & 0 & 3 \end{array}\right)$$

2. *Forward substitution.* We solve the system $Ly = b$, being $b = (1, 0, -4)$ the independent term of the system. We easily get $y = (1, 1, -3)$.

3. *Backward substitution* We solve the system $Ux = y$ to get the final solution, x. The result is $x = (1, 1, -1)$.

□

2 Iterative methods

Like for other iterative methods already introduced in previous chapters, iterative methods for solving linear systems of equations define a sequence of vectors, $\mathbf{x}^{(k)}$, which are expected to converge to the solution, $\mathbf{x}$, of the given linear system, i.e.

$$\lim_{k\to\infty} \mathbf{x}^{(k)} = \mathbf{x},$$

with $\mathbf{x}$ satisfying $A\mathbf{x} = \mathbf{b}$.

These methods are, in general, more efficient than direct methods for solving large systems of equations with sparse[1] coefficient matrices. The reason is that they are based just on matrix-vector multiplication, and that only the nonzero elements of the coefficient matrix need to be stored. In normal situations, iterative methods give acceptable approximations with few iterations, and have the advantage of being more robust to rounding errors than direct methods .

However, unlike direct methods, it is in general not possible to know in advance the number of operations needed to attain the solution (up to a prescribed error bound), and thus to know the execution time needed to get an approximation with a prescribed error tolerance. In addition, they also need some parameter prescription which is not present in direct methods.

Given an initial guess, $\mathbf{x}^{(0)}$, an iterative method produce a sequence of approximations, $\mathbf{x}^{(k)}$, for $k = 1,2,\ldots$, by some predefined algorithm, which is stopped when some criterion based on, for instance, the absolute difference between two iterations, is reached.

The classic linear iterative methods are based on rewriting the problem $A\mathbf{x} = \mathbf{b}$ as

$$\mathbf{x} = B\mathbf{x} + \mathbf{c},$$

where B is an $n \times n$ matrix and $\mathbf{c}$ is a column vector of dimension n. Taking $\mathbf{x}^{(0)}$ as an initial guess, we produce the sequence by the recursive formula

$$\mathbf{x}^{(k)} = B\mathbf{x}^{(k-1)} + \mathbf{c}$$

for $k = 1,2,\ldots$ The matrix B is called the *iteration matrix*, and must satisfy

$$\det(I - B) \neq 0, \tag{5.2}$$

that is, $I - B$ must be invertible. The vector $\mathbf{c}$ is called the *iteration vector*.

2.1 Method of Jacobi

In this method, to deduce the matrix B, we consider the decomposition $A = L + D + U$, where

$$L = \begin{pmatrix} 0 & 0 & \cdots & 0 \\ a_{21} & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & 0 \end{pmatrix}, \quad D = \begin{pmatrix} a_{11} & 0 & \cdots & 0 \\ 0 & a_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & a_{nn} \end{pmatrix}, \quad U = \begin{pmatrix} 0 & a_{12} & \cdots & a_{1n} \\ 0 & 0 & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{pmatrix},$$

and we assume that D is invertible, i.e. $a_{ii} \neq 0$ for all $i = 1,\ldots,n$. We deduce, after some algebra,

$$\mathbf{x} = -D^{-1}(L+U)\mathbf{x} + D^{-1}\mathbf{b} = (I - D^{-1}A)\mathbf{x} + D^{-1}\mathbf{b}.$$

[1] A sparse matrix is a matrix in which most of the elements are zero.

This formula motivates the Jordan's iterative scheme

$$\mathbf{x}^{(k)} = B_J \mathbf{x}^{(k-1)} + \mathbf{c}_J \tag{5.3}$$

with the iteration matrix and vector given by, respectively,

$$B_J = I - D^{-1}A, \quad \mathbf{c}_J = D^{-1}\mathbf{b}.$$

Observe that we have $D(I - B_J) = A$, implying $\det(D)\det(I - B_J) = \det(A) \neq 0$, and therefore neither D or $I - B_J$ may be singular (have zero determinant), explaining condition (5.2). The component-wise expression of formula (5.3) is simply

$$x_i^{(k)} = \frac{1}{a_{ii}}\Big(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k-1)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(k-1)}\Big).$$

Example 5.6 Solve the system

$$\begin{array}{rrrrcl} 10x_1 & -x_2 & +2x_3 & & = & 6, \\ -x_1 & +11x_2 & -x_3 & +3x_4 & = & 6, \\ 2x_1 & -x_2 & +10x_3 & -x_4 & = & 11, \\ & 3x_2 & -x_3 & +8x_4 & = & 15, \end{array} \tag{5.4}$$

by Jacobi's method, starting at $\mathbf{x}^{(0)} = \mathbf{0}$. Use the stopping criterion $\|\mathbf{x}^{(k)} - \mathbf{x}^{(k-1)}\|_\infty < 0.01$.

We first check that $\det(A) \neq 0$ (left to the reader). Then we compute the iteration matrix and vector:

$$B_J = I_{4\times 4} - \begin{pmatrix} \frac{1}{10} & 0 & 0 & 0 \\ 0 & \frac{1}{11} & 0 & 0 \\ 0 & 0 & \frac{1}{10} & 0 \\ 0 & 0 & 0 & \frac{1}{8} \end{pmatrix} \begin{pmatrix} 10 & -1 & 2 & 0 \\ -1 & 11 & -1 & 3 \\ 2 & -1 & 10 & -1 \\ 0 & 3 & -1 & 8 \end{pmatrix} = \begin{pmatrix} 0 & \frac{1}{10} & \frac{-2}{10} & 0 \\ \frac{1}{11} & 0 & \frac{1}{11} & \frac{-3}{11} \\ \frac{-2}{10} & \frac{1}{10} & 0 & \frac{1}{10} \\ 0 & \frac{-3}{8} & \frac{1}{8} & 0 \end{pmatrix},$$

$$\mathbf{c}_J = \begin{pmatrix} \frac{1}{10} & 0 & 0 & 0 \\ 0 & \frac{1}{11} & 0 & 0 \\ 0 & 0 & \frac{1}{10} & 0 \\ 0 & 0 & 0 & \frac{1}{8} \end{pmatrix} \begin{pmatrix} 6 \\ 6 \\ 11 \\ 15 \end{pmatrix} = \begin{pmatrix} \frac{6}{10} \\ \frac{6}{11} \\ \frac{11}{10} \\ \frac{15}{8} \end{pmatrix}.$$

Finally, we implement the iterations (5.3).

A more graphical way to deduce and implement Jacobi's method is the following. First, rewrite the system solving for x_1 the first equation, for x_2 the second, etc. We get

$$\begin{array}{rcl} x_1 & = & (6 + x_2 - 2x_3)/10, \\ x_2 & = & (6 + x_1 + x_3 - 3x_4)/11, \\ x_3 & = & (11 - 2x_1 + x_2 + x_4)/10, \\ x_4 & = & (15 - 3x_2 + x_3)/8, \end{array} \tag{5.5}$$

Then, for $k \geq 1$, Jacobi's method is just

$$\begin{array}{rcl} x_1^{(k)} & = & (6 + x_2^{(k-1)} - 2x_3^{(k-1)})/10, \\ x_2^{(k)} & = & (6 + x_1^{(k-1)} + x_3^{(k-1)} - 3x_4^{(k-1)})/11, \\ x_3^{(k)} & = & (11 - 2x_1^{(k-1)} + x_2^{(k-1)} + x_4^{(k-1)})/10, \\ x_4^{(k)} & = & (15 - 3x_2^{(k-1)} + x_3^{(k-1)})/8. \end{array}$$

For the first iteration, we have

$$\begin{aligned} x_1^{(1)} &= (6+x_2^{(0)}-2x_3^{(0)})/10 = 0.6, \\ x_2^{(1)} &= (6+x_1^{(0)}+x_3^{(0)}-3x_4^{(0)})/11 = 0.545, \\ x_3^{(1)} &= (11-2x_1^{(0)}+x_2^{(0)}+x_4^{(0)})/10 = 1.1, \\ x_4^{(1)} &= (15-3x_2^{(0)}+x_3^{(0)})/8 = 1.875. \end{aligned}$$

We check the stopping criterion,

$$\|\mathbf{x}^{(1)}-\mathbf{x}^{(0)}\|_\infty = \max_{1\le i\le 4}\left(\|\mathbf{x}_i^{(1)}-\mathbf{x}_i^{(0)}\|_\infty\right) = \max(0.6, 0.545, 1.1, 1.875) = 1.875 > 0.01.$$

Since the stopping criterion is not satisfied, we proceed with further iterations until the sixth iteration, when the stopping criterion is satisfied. We have

$$\mathbf{x}^{(6)} = (0.369, 0.153, 1.240, 1.979)$$

with $\|\mathbf{x}^{(6)}-\mathbf{x}^{(5)}\|_\infty = 0.007 < 0.01$. Thus, $\mathbf{x}^{(6)}$ is our approximate solution, that we may compare with the exact solution

$$\mathbf{x} = (0.368, 0.154, 1.239, 1.972).$$

□

2.2 Method of Gauss-Seidel

In this case, to deduce the iteration matrix, B, we use the same decomposition than for the Jacobi method, but from $(L+D+U)\mathbf{x} = \mathbf{b}$, we write $(L+D)\mathbf{x} = -U\mathbf{x}+\mathbf{b}$, and then

$$\mathbf{x} = -(L+D)^{-1}U\mathbf{x} + (L+D)^{-1}\mathbf{b}.$$

Thus, we define the iterative scheme

$$\mathbf{x}^{(k)} = B_{GS}\mathbf{x}^{(k-1)} + \mathbf{c}_{GS}, \tag{5.6}$$

with

$$B_{GS} = -(L+D)^{-1}U, \quad \mathbf{c}_{GS} = (L+D)^{-1}\mathbf{b}. \tag{5.7}$$

Observe that, in this method, both $L+D$ and $I-B_{GS}$ must be non-singular. The component-wise expression is now

$$x_i^{(k)} = \frac{1}{a_{ii}}\Big(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(k-1)}\Big).$$

Example 5.7 Solve the system (5.4) by the Gauss-Seidel's method, starting at $\mathbf{x}^{(0)} = \mathbf{0}$, and using the stopping criterion $\|\mathbf{x}^{(k)}-\mathbf{x}^{(k-1)}\|_\infty < 0.01$.

Like in the Jacobi's method, we must first check that both $L+D$ and $I-B_{GS}$ are invertible. This task is left to the reader. For using the vector form (5.6) of the method, we must first compute $(L+D)^{-1}$ and then use it for defining the matrix and vector iterators (5.7). Instead, we do it by showing the *graphical* form.

We start rewriting the system as in (5.5). The difference with Jacobi's method is that once one of the components has been computed, it enters in the computation of the next component, without waiting till the next iteration

$$\begin{aligned} x_1^{(k+1)} &= (6+x_2^{(k)}-2x_3^{(k)})/10, \\ x_2^{(k+1)} &= (6+x_1^{(\mathbf{k+1})}+x_3^{(k)}-3x_4^{(k)})/11, \\ x_3^{(k+1)} &= (11-2x_1^{(\mathbf{k+1})}+x_2^{(\mathbf{k+1})}+x_4^{(k)})/10, \\ x_4^{(k+1)} &= (15-3x_2^{(\mathbf{k+1})}+x_3^{(\mathbf{k+1})})/8. \end{aligned}$$

Performing the iterations, we see that at the fourth the stopping criterion is satisfied. We get

$$\mathbf{x}^{(4)} = (0.369, 0.154, 1.239, 1.972)$$

with $\|\mathbf{x}^{(4)} - \mathbf{x}^{(3)}\|_\infty = 0.009 < 0.01$. Observe that, compared to the Jacobi's method, the Gauss-Seidel's method has saved two iterations. □

2.3 Convergence of iterative methods

We present two ways for checking the convergence. The first is particular to the Jordan's and Gauss-Seidel's methods. The second applies to any iterative method of the form $\mathbf{x}^{(k)} = B\mathbf{x}^{(k-1)} + \mathbf{c}$. We start with some difinitions.

Definition 8 *Let A be a square matrix.*

- *A is* diagonally strictly dominant by rows *if*

$$|a_{ii}| > \sum_{\substack{j=1 \\ j\neq i}}^{n} |a_{ij}|, \quad \textit{for } i = 1,2,\ldots,n.$$

- *A is* diagonally strictly dominant by columns *if*

$$|a_{ii}| > \sum_{\substack{i=1 \\ i\neq j}}^{n} |a_{ij}|, \quad \textit{for } j = 1,2,\ldots,n.$$

A is positive definite*, if it is* symmetric*, that is, $A = A^T$, and*

$$x^T A x > 0 \quad \textit{for all } x \neq 0.$$

Remark 5.1 Sylvester's criterion: *A symmetric matrix is definite positive if its leading principal minors are all positive. The k-th leading principal minor of a matrix is the determinant of its upper-left $k \times k$ sub-matrix. For instance, for*

$$A = \begin{pmatrix} 2 & 2 & 0 \\ 2 & 5 & -1 \\ 0 & -1 & 3 \end{pmatrix},$$

we have that A is symmetric, and

$$\det(a_{11}) = 2 > 0, \quad \det\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} = 6 > 0, \quad \textit{and} \quad \det(A) = 16 > 0.$$

Thus, A is definite positive.

We have the following convergence result.

Theorem 5.1 *Consider the system* $A\mathbf{x} = \mathbf{b}$.

- *If A is diagonally strictly dominant by rows or columns then the methods of Jacobi and Gauss-Seidel converge for any initial guess.*
- *If A is positive definite then the method of Gauss-Seidel converges for any initial guess.*

Example 5.8 Let us consider the matrix

$$A = \begin{pmatrix} 2 & 1 & 0 \\ 2 & 5 & -1 \\ 0 & -1 & 3 \end{pmatrix},$$

and check the assumption of Theorem 5.1. Since A is not symmetric, it can not be definite positive. Thus, we check if it is diagonally strictly dominant. For the first row, we have

$$|a_{11}| = |2| > |1| + |0| = |a_{12}| + |a_{13}|,$$

while for the first column we have

$$|a_{11}| = |2| = |2| + |0| = |a_{21}| + |a_{31}|.$$

Thus, the matrix is not diagonally strictly dominant neither for rows or for columns. However, notice that Theorem 5.1 gives sufficient conditions. The iterative schemes *could* converge for this matrix, but it does not *necessarily* converge. □

Observe that the general iterative scheme we are studying,

$$\mathbf{x}^{(k)} = B\mathbf{x}^{(k-1)} + \mathbf{c}, \tag{5.8}$$

is just a fixed point method like the studied in Chapter 2 for finding zeros of nonlinear functions. There, we defined the iterative scheme $x_k = g(x_{k-1})$, where g is a differentiable function, and stated several sufficient conditions for convergence, among which the *contractivity* of g, which is verified ig $g'(x) < 1$. In the context of the scheme (5.8), we have $g' = B$ (in an n-dimensional sense) and then, the contractivity is fulfilled if "$B < 1$" in some sense to be precised.

Definition 9 *The* spectral radius *of a square matrix, B, of oder n, is given by*

$$\rho_B = \max_{i=1,\ldots,n} |\lambda_i|,$$

where λ_i *are the eigenvalues of B.*

Now we may make precise the above idea of "$B < 1$".

Theorem 5.2 *Given a linear system in the form* $\mathbf{x} = B\mathbf{x} + \mathbf{c}$*, the corresponding iterative method (5.8) is convergent if and only if* $\rho_B < 1$.

Example 5.9 In this example we study the convergence of the Gauss-Seidel's method for the system $A\mathbf{x} = \mathbf{b}$, for any $\mathbf{b} \in \mathbb{R}^3$, and with

$$A = \begin{pmatrix} 3 & 1 & 1 \\ 1 & 2 & -1 \\ 3 & 1 & 3 \end{pmatrix}.$$

In the Gauss-Seidel's method we have $B_{GS} = -(L+D)^{-1}U$, which gives

$$B_{GS} = -(L+D)^{-1}U = -\begin{pmatrix} \frac{1}{3} & 0 & 0 \\ -\frac{1}{6} & \frac{1}{2} & 0 \\ -\frac{5}{18} & -\frac{1}{6} & \frac{1}{3} \end{pmatrix}\begin{pmatrix} 0 & 1 & 1 \\ 0 & 0 & -1 \\ 0 & 0 & 0 \end{pmatrix} = \begin{pmatrix} 0 & -\frac{1}{3} & -\frac{1}{3} \\ 0 & \frac{1}{6} & \frac{2}{3} \\ 0 & \frac{5}{18} & \frac{1}{9} \end{pmatrix}.$$

The eigenvalues, λ_i, for $i = 1,2,3$ of the matrix B_{GS} are determined as the roots of the *characteristic polynomial*, defined as $p(\lambda) = \det(B_{GS} - \lambda I)$. Thus, we have to solve

$$p(\lambda) = \begin{vmatrix} 0-\lambda & -\frac{1}{3} & -\frac{1}{3} \\ 0 & \frac{1}{6}-\lambda & \frac{2}{3} \\ 0 & \frac{5}{18} & \frac{1}{9}-\lambda \end{vmatrix} = 0$$

which is simplified to

$$p(\lambda) = \lambda\left(\frac{1}{6} + \frac{5}{18}\lambda - \lambda^2\right) = 0.$$

Therefore,

$$\lambda_1 = 0, \quad \lambda_2 = 0.57, \quad \lambda_3 = 0.29.$$

Since all the eigenvalues are smaller than one, we deduce from Theorem 5.2 that the Gauss-Seidel iterative scheme is convergent for this matrix. □

Theorem 5.3 (Speed of convergence) *Let B be the iteration matrix of a convergent iterative method, and* $\mathbf{x}$ *its exact solution. Suppose that* $\rho(B) < 1$*. Then,*

$$\|\mathbf{x}^{(k)} - \mathbf{x}\| \leq \frac{\rho(B)^k}{1-\rho(B)}\|\mathbf{x}^{(1)} - \mathbf{x}^{(0)}\|. \tag{5.9}$$

Remark 5.2 *The bound (5.9) provied us with an estimate for the number of iterations needed to get an error smaller than a specified tolerance. Indeed, we get this if*

$$\frac{\rho(B)^k}{1-\rho(B)}\|\mathbf{x}^{(1)} - \mathbf{x}^{(0)}\| < tol \implies k > \frac{1}{\log(\rho(B))}\log\left(\frac{(1-\rho(B))tol}{\|\mathbf{x}^{(1)} - \mathbf{x}^{(0)}\|}\right). \tag{5.10}$$

Example 5.10 For the matrix of Example 5.9, and $\mathbf{b} = (0,1,0)$, compute the number of iterations needed to get an error, in the Euclidean norm, smaller than $tol = 0.001$ when starting at $\mathbf{x}^{(0)} = \mathbf{0}$.

We have $\rho(B_{GS}) = 0.57$. The first iteration is

$$\mathbf{x}^{(1)} = \mathbf{c}_{GS} = (L+D)^{-1}\mathbf{b} = \big(0, \frac{1}{2}, -\frac{1}{6}\big).$$

Thus, $\|\mathbf{x}^{(1)} - \mathbf{x}^{(0)}\|_2 = 1/(2\sqrt{10})$. Using the bound (5.10), we get

$$k > \frac{1}{\log(\rho(B_{GS}))} \log\Big(\frac{(1-\rho(_{GS}))tol}{\|\mathbf{x}^{(1)} - \mathbf{x}^{(0)}\|}\Big) \approx 10.509.$$

Thus, 11 iterations are enough to obtain the specified tolerance.

□

3 Approximation of partial differential equations

A *partial differential equation* (PDE) is an equation involving derivatives of a function of two or more variables, for instance

$$\frac{\partial u(t,x)}{\partial t} - \frac{\partial^2 u(t,x)}{\partial x^2} = 0, \tag{5.11}$$

where $u : (0,T) \times (0,L) \to \mathbb{R}$. Like for differential equations, the problem is to find the function u which satisfies the PDE ((5.11) in our example) together with some conditions which allow to determine uniquely the solution to the problem. There are two type of conditions:

- The *initial condition*, that is, an equation establishing how $u(t,x)$ is at starting time,

$$u(0,x) = u_0(x), \quad \text{for all } x \in (0,L).$$

- The *boundary condition*. Since we have second order derivatives in x, we need two conditions to determine the corresponding constants of integration. There are many boundary conditions one can prescribe. The most common are:

$$u(t,0) = u(t,L) = 0, \quad \text{for all } t \in (0,T) \qquad \text{(Dirichlet boundary condition)},$$
$$\frac{\partial u(t,0)}{\partial x} = \frac{\partial u(t,L)}{\partial x} = 0, \quad \text{for all } t \in (0,T) \qquad \text{(Neumann boundary condition)}.$$

PDE's are a very important topic in Physics and Engineering, and most of the times approximated methods must be used to solve them.

Introducing backward differences in time in (5.11), we find the time discretization

$$u(t_{i+1},x) = u(t_i,x) + \tau \frac{\partial^2 u(t_{i+1},x)}{\partial x^2},$$

where τ is the time step of the time mesh, that is, $t_{i+1} - t_i = \tau$. Using centered differences for the second order derivative, we get

$$u(t_{i+1},x_j) = u(t_i,x_j) + \frac{\tau}{h^2}\big(u(t_{i+1},x_{j-1}) - 2u(t_{i+1},x_j) + u(t_{i+1},x_{j+1})\big).$$

Thus, if we want to determine u at time t_{i+1} from its previous state at time $t = t_i$, we must solve

$$-ru(t_{i+1},x_{j-1}) + (1+2r)u(t_{i+1},x_j) - ru(t_{i+1},x_{j+1}) = u(t_i,x_j), \tag{5.12}$$

where $r = \tau/h^2$. If the nodes of $(0,L)$ are $x_0, x_1, \ldots, x_m$, then the above equation (5.12) only can be evaluated for $x_1, \ldots, x_{m-1}$ (the interior nodes). In the boundary nodes, $x_0 = 0$ and $x_m = L$, we use the boundary conditions. For instance, for Dirichlet boundary conditions, we set $u(x_0, t_i) = u(x_m, t_i) = 0$ for all i.

Introducing the notation $b_j = u(t_i, x_j)$, and $y_j = u(t_{i+1}, x_j)$, we rewrite (5.12) together with the boundary conditions as the linear system of equations

$$\begin{pmatrix} 1 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & 0 \\ -r & 1+2r & -r & 0 & \cdots & 0 & 0 & 0 & 0 \\ 0 & -r & 1+2r & -r & \cdots & 0 & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \cdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & -r & 1+2r & -r & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & -r & 1+2r & -r \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_{m-2} \\ y_{m-1} \\ y_m \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_{m-2} \\ b_{m-1} \\ b_m \end{pmatrix} \tag{5.13}$$

It is possible to show that the matrix of this system is non-singular and, in fact, definite positive. This system can be solved with any of the methods we have described in this chapter.

Chapter 6

Optimization

The central problem in the theory of Optimization is the development of mathematical tools to, on one hand, deduce the existence of minima and/or maxima of applications $f : \Omega \subset \mathbb{R}^N \to \mathbb{R}$, and, on the other hand, to devise numerical algorithms providing approximations to such points.

The most fundamental classification of optimization problems divide them in *problems without constraints*, and *problems with constraints*. Constraints are usually formulated in terms of functional restrictions limiting the points belonging to Ω.

Observe that maximizing a function, f, is equivalent to minimizing the function $-f$. Thus, we shall only deal with the minimization problem, understanding that all the results we may obtain are directly translated to the maximization problem.

1 Definition of an optimization problem

An optimization problem may be sketched as follows: a physical or control variable must be chosen to optimize (minimize or maximize) a physical criterion, such as the energy, a technical criterion, like accuracy, duration, etc., or an economical criterion, like cost, productivity, etc., always considering the natural constraints affecting to the variable (must be positive, integer, ...)

We introduce in the following lines some terminology and notation commonly used in optimization problems. An *optimization problem* consists of:

1. A *criterion* (or *cost*, or *objective*), f, mapping the space of decision variables, $\Omega \subset \mathbb{R}^N$, to $\mathbb{R}$,
$$f : \Omega \to \mathbb{R}.$$

2. The *constraints*. In general, not all the elements of Ω are admissible as solutions since some constraints, determining the *space of solutions*, must be satisfied. These constraints arise in applications in different forms, which may be present simultaneously:

   (a) *Equality constraints*
   $$\phi(y) = 0, \tag{6.1}$$
   where $\phi : \Omega \subset \mathbb{R}^N \to \mathbb{R}^m$, with $m < n$. We say that the solution is subject to *m equality constraints*, $\phi_i(y) = 0$, for $i = 1, ..., m$.

(b) *Inequality constraints*

$$\psi(y) \leq 0, \tag{6.2}$$

where $\psi : \Omega \subset \mathbb{R}^N \to \mathbb{R}^p$, with $p < n$, and (6.2) means $\psi_j(y) \leq 0$ for $j = 1, ..., p$. We say that the solution is subject to *p inequality constraints*.

(c) *Set constraints.* Equality and inequality constraints are particular cases of constraints given as set constraints, of the type $y \in S$, where $S \subset \Omega$ is a given set.

In any case, the constraints determine a subset $U \subset \Omega$, called *set of admissible points*, given as $U = \{y : y \quad \text{satisfy the constraints}\}$. The minimization problem consists, then, in finding $u \in U$ such that

$$f(x) \leq f(y) \quad \text{for all } y \in U. \tag{6.3}$$

If such x does exist, we say that it is a *minimum* of f in U, and that $f(x)$ is the *minimum value* of the minimization problem.

In general, we have not on hand mathematical techniques for solving any minimization problem in the whole set U, i.e. for finding a *global minimum* of (6.3). Thus, we normally restrict ourselves to finding a *local minimum* $\bar{x} \in U$, i.e., to solve

$$f(\bar{x}) \leq f(y) \quad \text{for all } y \in U \cap B,$$

where B is a neighborhood of $\bar{x}$. Clearly, a global minimum is always a local minimum, being the reciprocal not true, in general.

Sometimes, we shall use the following short notation to refer to a minimization problem:

$$\begin{cases} \min f(x) \\ x \in C, \quad \phi(x) = 0, \quad \psi(x) \leq 0. \end{cases}$$

Example 6.1 Linear programming.

Important problems in Economy and Engineering are formulated in terms of a *linear programming problem*:

$$\begin{cases} \min f(x) = c^T x \\ x \in \mathbb{R}^n, \quad Ax \geq b, \end{cases}$$

where $c \in \mathbb{R}^N$ is a row vector, $x \in \mathbb{R}^N$ is a column vector, A is a $m \times n$ matrix, and $b \in \mathbb{R}^N$ is a column vector.

The first linear programming problem, dating to 1944, was introduced to formulate the diet problem. We have a stock of n types of food products $x_1, \ldots, x_n$, and m parameters related to quantities of vitamins, proteins, etc. contained in such food. We define

- a_{ij}, the quantity of parameter i contained in product j,
- b_j, the minimum necessary quantity of parameter j in each ration, and
- c_j, the unitary cost of product j.

Thus, the minimum cost ration, given by x_j units of product j and satisfying the constraints of minimum content of parameter i is the solution of

$$\begin{cases} \min \sum_{j=1}^{n} c_j x_j \\ x_j \geq 0, \quad j = 1, ..., n, \quad \sum_{j=1}^{n} a_{ij} x_j \geq b_i, \quad i = 1, ..., m. \end{cases}$$

□

2 Optimization without constraints

2.1 Necessary and sufficient conditions for a local minimum

Given a function $f : \mathbb{R}^N \to \mathbb{R}$ twice continuously differentiable, the procedure used in Differential Calculus to find points of minimum is the following:

1. Solve the system of equations for the *critical points*, i.e., find $x^* \in \mathbb{R}^N$ such that $\nabla f(x^*) = 0$, or in expanded form,
$$\frac{\partial f}{\partial x_1}(x^*) = 0, \ldots, \frac{\partial f}{\partial x_n}(x^*) = 0. \tag{6.4}$$
Equations (6.4) are the so-called *first order optimality conditions*.

2. Evaluate the Hessian matrix of f, H_f, at the critical points, and check whether the matrix is positive definite. That is, check if the symmetric matrix
$$H_f(x^*) = \begin{pmatrix} \frac{\partial^2 f(x^*)}{\partial x_1^2} & \frac{\partial^2 f(x^*)}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f(x^*)}{\partial x_1 x_N} \\ \frac{\partial^2 f(x^*)}{\partial x_2 x_1} & \frac{\partial^2 f(x^*)}{\partial^2 x_2} & \cdots & \frac{\partial^2 f(x^*)}{\partial x_2 x_N} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial^2 f(x^*)}{\partial x_N \partial x_1} & \frac{\partial^2 f(x^*)}{\partial x_2 \partial x_N} & \cdots & \frac{\partial^2 f(x^*)}{\partial^2 x_N} \end{pmatrix}$$
is definite positive.

If this is the case, then x^* is a point of local minimum for f, that is, there exists a radius $\rho > 0$ such that
$$f(x^*) \leq f(x) \quad \text{for all } x \in B_\rho(x^*).$$
Let us see why this program is justified.

Theorem 6.1 (Necessary conditions for local minimum) *Let f be a twice continuously differentiable function and assume that x^* is a local minimum. Then $\nabla f(x^*) = 0$ and $H_f(x^*)$ is positive semidefinite.*

Proof. Let $v \in \mathbb{R}^N$ be a given vector. Taylor's theorem implies
$$f(x^* + tv) = f(x^*) + t\nabla f(x^*)^T v + \frac{t^2}{2} v^T H_f(x) v + o(\|t\|^2).$$
Since x^* is a local minimum, we have $f(x^* + tu) \geq f(x^*)$, for t small enough. Then, dividing by t, we get
$$\nabla f(x^*)^T v + \frac{t}{2} v^T H_f(x) v + o(\|t\|) \geq 0. \tag{6.5}$$
Setting $t = 0$ and $v = -\nabla f(x^*)$ we deduce $\|\nabla f(x^*)\| = 0$, i.e., $\nabla f(x^*) = 0$. Now, using this identity in (6.5), dividing by t and taking $t = 0$, we obtain
$$\frac{1}{2} v^T H_f(x) v \geq 0.$$

□

Condition (6.4), although necessary, is not sufficient for x^* being a point of minimum of f. So it is to say, there exist critical points of f which are not minimum. To ensure that a critical point is actually a minimum we use the following result.

Theorem 6.2 (Sufficient conditions for a local minimum) *Let f be a twice continuously differentiable function and assume that x^* is a critical point of f and that $H_f(x^*)$ is positive definite. Then, x^* is a local minimum of f.*

Proof. Let $v \in \mathbb{R}^N$ be a nonzero given vector. For t small enough, Taylor's theorem implies

$$f(x^*+tv) = f(x^*) + \frac{t^2}{2} v^T H_f(x) v + o(\|t\|^2).$$

Since $H_f(x^*)$ is positive definite, there exists a number $\lambda > 0$ such that

$$v^T H_f(x) v > \lambda \|v\|^2 > 0.$$

Then

$$f(x^*+tv) - f(x^*) = \frac{t^2}{2} v^T H_f(x) v + o(\|t\|^2) > \lambda \frac{t^2}{2} \|v\|^2 + o(\|t\|^2) > 0,$$

for all $t \neq 0$ small enough. □

Example 6.2 Observe that Taylor's theorem tell us that a function with a local minimum in x^* is, in a neighborhood of x^*, bounded from below by a paraboloid. For instance, assume $x^* = 0$ is a minimum of a two-dimensional function ($n = 2$). Taking $e = (x_1, x_2)$ and neglecting the term $o(\|e\|^2)$, we get

$$\begin{aligned} f(x_1,x_2) \approx & f(0,0) + x_1 \frac{\partial f}{\partial x_1}(0,0) + x_2 \frac{\partial f}{\partial x_2}(0,0) + \frac{1}{2}\Big(\frac{\partial^2 f}{\partial x_1^2}(0,0) x_1^2 + \frac{\partial^2 f}{\partial x_2^2}(0,0) x_2^2 \\ & + 2 \frac{\partial^2 f}{\partial x_1 \partial x_2}(0,0) x_1 x_2 \Big) \\ = & f(0,0) + \frac{1}{2}\Big(\frac{\partial^2 f}{\partial x_1^2}(0,0) x_1^2 + \frac{\partial^2 f}{\partial x_2^2}(0,0) x_2^2 + 2 \frac{\partial^2 f}{\partial x_1 \partial x_2}(0,0) x_1 x_2 \Big) \\ > & f(0,0) + \lambda(x_1^2 + x_2^2), \end{aligned}$$

for some $\lambda > 0$, since $H_f(0)$ is positive definite. □

Although, in general, a function may have many local minima, and the differential method does not tell us which of them is the global minimum, there is an important exception: when the function is convex, and defined in a convex set.

Definition 10 *We say that a set $\Omega \subset \mathbb{R}^N$ is* convex *if for all $x,\ y \in \Omega$, and for all $\mu \in [0,1]$ we have*

$$\mu x + (1-\mu) y \in \Omega.$$

We say that a function $f : \Omega \subset \mathbb{R}^N \to \mathbb{R}$ is convex *if for all $x,\ y \in \Omega$ and for all $\mu \in [0,1]$ we have*

$$f(\mu x + (1-\mu) y) \leq \mu f(x) + (1-\mu) f(y).$$

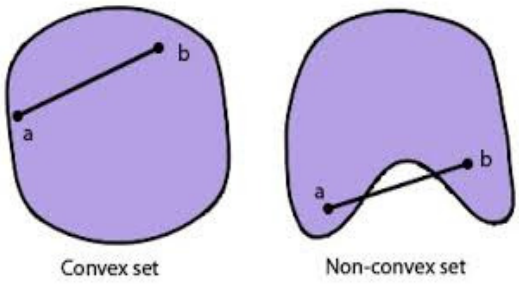


Figure 6.1: Example of convex and non-convex sets in $\mathbb{R}^2$.

It is not difficult to prove that if $\Omega \subset \mathbb{R}^N$ is convex and bounded, and if $f : \Omega \to \mathbb{R}$ is convex and differentiable, then f can have, at most, one critical point which, if it does exist, corresponds to a global minimum of f.

Recall that a function $f : \Omega \subset \mathbb{R}^N \to \mathbb{R}$ with the Hessian $H_f(x)$ positive definite for all $x \in \Omega$ is a convex function, see the Appendix.

Example 6.3 Let $\Omega = (-a,a) \subset \mathbb{R}$, an interval centered at $a > 0$, which is clearly a convex set, and $f(x) = x^2$, which is a convex function since $f''(x) > 0$. Thus, the unique critical point $0 \in (-a,a)$ is a global minimum.

In the same interval, the function $g(x) = e^{-x}$ is also convex, since $g''(x) > 0$. However, the are not critical points of g in $(-a,a)$, and the above statement does not give any clue about the minima of g. Observing the graph of g, we see that it has not minima in this interval, since it is a decreasing function. If the interval is redefined to $[-a,a]$, then it has a unique global minimum, attained at the border $x = a$, which is not a critical point. □

Finding the exact solution of the first order optimality conditions, (6.4), is not always possible. Thus, as in previous chapters, we consider iterative methods to approximate the solution.

Example 6.4 Let us consider a differentiable function, f, defined in $\mathbb{R}$. The optimality conditions of first order reduce to finding $x^* \in \mathbb{R}$ such that

$$f'(x^*) = 0.$$

Using Newton's method for approximating zeros of nonlinear functions, see formula (2.4) in Chapter 2, the approximation algorithm for the critical points of f is given by

$$x_{k+1} = x_k - \frac{f'(x_k)}{f''(x_k)}, \quad \text{for } k = 1,2,\ldots$$

where x_0 is an initial guess. Clearly, a necessary condition for convergence is $f''(x) \neq 0$ in the set of iterands. In fact, if we look for a minimum, we must have $f''(x) > 0$ in a neighborhood of the solution. Thus, convexity or positive definiteness. □

2.2 Method of Newton

Newton's method for finding minima of functions $f : \mathbb{R}^N \to \mathbb{R}$ is deduced from Taylor's expansion, given by formula (A.34). Let us consider the second order approximation, that is, neglect the term

$o(\|e\|^2)$. We get

$$f(x) \approx f(x_k) + \nabla f(x_k)^T (x - x_k) + \frac{1}{2}(x - x_k)^T H_f(x_k)(x - x_k), \tag{6.6}$$

where H_f is the Hessian matrix. To find an approximation of a critical point of f, we differentiate the right hand side term of (6.6) with respect to x_j, for $j = 1, \ldots, n$, and equate to zero. We obtain the system of linear equations

$$\nabla f(x_k) + H_f(x_k)(x - x_k) = 0,$$

where, writing x_{k+1} for the solution, we deduce

$$x_{k+1} = x_k - (H_f(x_k))^{-1} \nabla f(x_k). \tag{6.7}$$

Observe that in Newton's minimization method, like in the corresponding method to find zeros of nonlinear functions, the initial guess, x_0, must be close enough to the minimum to achieve convergence. Thus, we should initially check that the matrix $H_f(x_0)$ is positive definite.

Example 6.5 Let $f(x,y) = \frac{1}{m}(x^m + \eta y^m)$, where $m > 1$ is an integer number and $\eta \in \mathbb{R}$ is positive. Thus, $f(x,y) > 0$ for all $(x,y) \neq (0,0)$ and $f(0,0) = 0$, that is $(0,0)$ is a global minimum. We have

$$\nabla f(x,y) = (x^{m-1}, \eta y^{m-1}), \quad H_f(x,y) = (m-1) \begin{pmatrix} x^{m-2} & 0 \\ 0 & \eta y^{m-2} \end{pmatrix}.$$

Then,

$$\big(H_f(x,y)\big)^{-1} \nabla f(x,y) = \frac{1}{m-1} \begin{pmatrix} x^{2-m} & 0 \\ 0 & \frac{1}{\eta} y^{2-m} \end{pmatrix} \begin{pmatrix} x^{m-1} \\ \eta y^{m-1} \end{pmatrix} = \frac{1}{m-1} \begin{pmatrix} x \\ y \end{pmatrix}.$$

Therefore, using the notation $\mathbf{x} = (x,y)$, Newton's method gives the iterative formula

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \frac{1}{m-1}\mathbf{x}_k = \frac{m-2}{m-1}\mathbf{x}_k.$$

First, observe that if $m = 2$ and therefore f is a parabolid, Newton's method converges in the first step, since we directly get $\mathbf{x}_1 = 0$ for any initial guess $\mathbf{x}_0$ we may choose. If $m \neq 2$, we may solve the above iterative formula to get

$$\mathbf{x}_{k+1} = \Big(\frac{m-2}{m-1}\Big)^{k+1} \mathbf{x}_0 \to \mathbf{0} \quad \text{as } k \to \infty,$$

for any $\mathbf{x}_0 \in \mathbb{R}^2$, since $(m-2)/(m-1) < 1$. Therefore, the method converges for any power $m > 1$ if function f, and for any initial guess. However, observe that if m is very large then the quotient $(m-2)/(m-1)$ is very close to one, and the convergence will be slow. □

Since computing the inverse of a matrix is normally an expensive calculation, when using Newton's method we solve, instead of (6.7), the following system

$$H_f(x_k) y = \nabla f(x_k), \tag{6.8}$$

and then, we write $x_{k+1} = x_k - y$. An additional advantage of having a positive definite Hessian matrix is that it admits a *Cholesky factorization*, that is, there exists a lower triangular matrix, L, with positive diagonal, such that $H_f(x_k) = LL^T$. Then, once the factorization has been computed, we may solve the system (6.8) by forward substitution.

Stopping criterion and error estimation

Since Newton's method searches for a critical point, a reasonable criterion for stopping the iterations could be

$$\|\nabla f(x_k)\| \leq \tau_r \|\nabla f(x_0)\|, \tag{6.9}$$

with $\tau_r \in (0,1)$, capturing in this way the gradient norm decrease. However, if $\|\nabla f(x_0)\|$ is small, it could be not possible to satisfy (6.9) in the floating point arithmetics, and therefore the iterations would not terminate. A more exigent criterion, and also safer, is based on a combination of the absolute and relative errors, i.e.

$$\|\nabla f(x_k)\| \leq \tau_r \|\nabla f(x_0)\| + \tau_a,$$

where τ_a is a tolerance for the absolute error. Of course, in addition to these criterion, one also adds a limit to the maximum number of iterations.

We finish this section with a convergence result.

Theorem 6.3 *Assume the following conditions:*

- *f is three times continuously differentiable*
- *x^* is a critical point of f*
- *$H_f(x^*)$ is positive definite.*

Then, if x_0 is close enough to x^*, the iterations of Newton's method (6.7) converge quadratically to x^*, i.e., for some constant $\lambda > 0$,

$$\|x_{k+1} - x^*\| \leq \lambda \|x_k - x^*\|^2.$$

2.3 The gradient method

In the gradient method, also known as *descent method*, we search for directions for which, when passing from iterand x_k to x_{k+1}, the value of f decreases, i.e. we have $f(x_{k+1}) < f(x_k)$.

We define the iterative scheme

$$x_{k+1} = x_k + \alpha_k d_k, \tag{6.10}$$

where d_k is the direction in the step k and $\alpha_k > 0$ is the length of the corresponding step. From Taylor's expansion of first order, we get

$$f(x_{k+1}) = f(x_k + \alpha_k d_k) \approx f(x_k) + \alpha_k \langle \nabla f(x_k), d_k \rangle,$$

and therefore, to get the steepest descent, we take the opposite direction to $\nabla f(x_k)$, that is

$$d_k = -\nabla f(x_k), \tag{6.11}$$

and then

$$f(x_{k+1}) \approx f(x_k) - \alpha_k \|\nabla f(x_k)\|^2 \leq f(x_k),$$

since $\alpha_k > 0$. Therefore, from (6.10) we obtain

$$x_{k+1} = x_k - \alpha_k \nabla f(x_k). \tag{6.12}$$

For choosing the step length, we define the function $\phi : \mathbb{R} \to \mathbb{R}$ given by $\phi(\alpha) = f(x_k + \alpha d_k)$ and search for α_k minimizing ϕ. Observe that we have reduced the n-dimensional minimization problem to a one-dimensional problem, which can be solved, for instance, by Newton's method.

In practice, instead of minimizing ϕ, it is often preferred to minimize an interpolator of ϕ. For instance, since we have the data

$$\phi(0) = f(x_k), \quad \phi(1) = f(x_k + d_k), \quad \text{and } \phi'(0) = \langle -d_k, d_k \rangle < 0,$$

we can take an approximation of $\phi(\alpha)$, for $\alpha \in [0,1]$, by the quadratic polynomial

$$q(\alpha) = \phi(0) + \phi'(0)\alpha + (\phi(1) - \phi(0) - \phi'(0))\alpha^2,$$

whose global minimum may be easily computed. On one hand, if $\phi(1) - \phi(0) - \phi'(0) < 0$, then the minimum of q belongs to the border of the interval $[0,1]$, and we take $\alpha = 1$ ($\alpha = 0$ is not allowed, since then the iterations stop, see (6.10)).

On the other hand, if $\phi(1) - \phi(0) - \phi'(0) > 0$, then ϕ has the local minimum given by

$$\alpha_L = \frac{-\phi'(0)}{2(\phi(1) - \phi(0) - \phi'(0))} > 0,$$

so we take $\alpha = \min\{1, \alpha_L\}$.

An inherent property of the gradient method is that the trajectory followed by the iterands is zig-zagging. Indeed, if α_k is the exact minimum of $\phi(\alpha)$ then, using the chain rule, we obtain

$$0 = \phi'(\alpha_k) = \langle \nabla f(x_k + \alpha_k d_k), d_k \rangle = -\langle \nabla f(x_{k+1}), \nabla f(x_k) \rangle,$$

where we used (6.10) and (6.11). Thus, $\nabla f(x_k)$ and $\nabla f(x_{k+1})$ are orthogonal.

Stopping criterion and error estimation

Like for Newton's method, a reasonable stopping criterion is obtained by combining the absolute and relative errors of ∇f,

$$\|\nabla f(x_k)\| \leq \tau_r \|\nabla f(x_0)\| + \tau_a,$$

where $\tau_r \in (0,1)$ is a tolerance for the relative error and τ_a is a tolerance for the absolute error.

In general, the gradient method has not good convergence properties. Depending on the function, the method can be very slow. We illustrate this fact with an example.

Example 6.6 Consider the function $f(x) = \frac{a}{2}x^2$, with $a \in (0,1)$, having the unique critical point at $x^* = 0$. An easy computation for the step $\alpha = \min\{1, \alpha_L\}$ shows that $\alpha_L = 1/a$, so we must take $\alpha = 1$. Then, the iterations (6.12) take the form

$$x_{k+1} = x_k - f'(x_k) = (1-a)x_k,$$

so we can expect only linear convergence:

$$|x_{k+1} - x_k| = a|x_k - x^*|.$$

Moreover, we obtain by recursion that $x_k = (1-a)^k x_0$, and therefore, if a is close to zero, the convergence is extremely slow. □

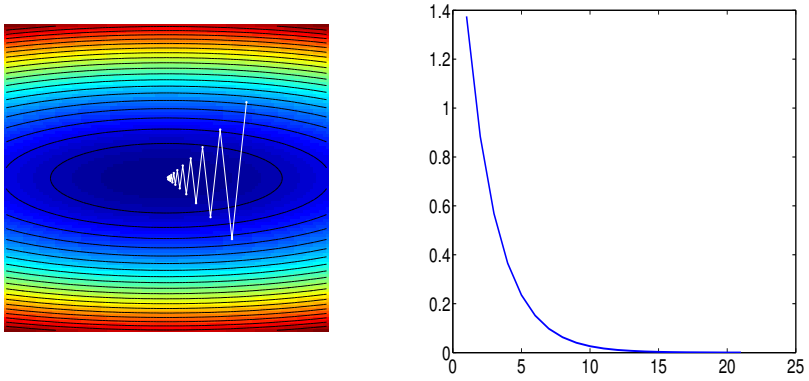


Figure 6.2: Descent trajectories for x_k and $f(x_k)$.

3 Constrained optimization

The choice of a method to solve a constrained optimization problem depends on the type of constraints operating in the problem: equality, inequality, or set restrictions.

In this section we shall introduce two methods which are particularly important. The *method of Lagrange multipliers* and the *penalty method*. The first is used for equality and inequality constraints, while the second operates for any kind of restriction.

3.1 Lagrange multipliers. Equality constraints

The Lagrange multipliers method allows us to use the optimization techniques already studied for problems without constraints to problems with constraints. Let as recall the problem formulation.

Given a differentiable objective function $f : \Omega \subset \mathbb{R}^N \to \mathbb{R}$, and a set of differentiable functions $\phi_i : \Omega \subset \mathbb{R}^N \to \mathbb{R}$, for $i = 1, \ldots, m$, with $m < n$, find a minimum x^* of f in Ω satisfying the equality constraints $\phi_i(x^*) = 0$ for all $i = 1, \ldots, m$. We have the following result.

Theorem 6.4 (Necessary conditions for constrained problems) *Suppose that x^* is a point of the set*

$$U = \{x \in \Omega : \phi_i(x) = 0, \quad 1 \leq i \leq m\} \subset \Omega, \tag{6.13}$$

such that the m vectors $\nabla\phi_i(x^*) \in \mathbb{R}^N$, with $i = 1, \ldots, m$, are linearly independent. Then, if f has a local minimum at x^* relative to the set U, there exist m numbers $\lambda_i(x^*)$, such that

$$\nabla f(x^*) + \lambda_1(x^*)\nabla\phi_1(x^*) + \ldots + \lambda_m(x^*)\nabla\phi_m(x^*) = 0. \tag{6.14}$$

The numbers $\lambda_i(x^*)$ are called *Lagrange multipliers*.

Although Theorem 6.4 provide us with a criterion to decide if a point x^* may be a constrained minimum, it does not give any idea of how to calculate it.

The most common tool used to find such a point is the *Lagrangian function*. Let us denote by λ to the vector $(\lambda_1, \ldots, \lambda_m)$, by $\phi : \mathbb{R}^N \to \mathbb{R}^m$ to the function $\phi(x) = (\phi_1(x), \ldots, \phi_m(x))$, and consider the function $L : \mathbb{R}^N \times \mathbb{R}^m \to \mathbb{R}$ given by

$$L(x, \lambda) = f(x) + \lambda^T \phi(x) = f(x) + \sum_{i=1}^{m} \lambda_i \phi_i(x).$$

If (x^*,λ^*) is a minimum of L (without constraints) then $\nabla_{(x,\lambda)}L(x^*,\lambda^*)=0$, i.e., the optimality conditions with respect to x

$$\nabla f(x^*)+\sum_{i=1}^{m}\lambda_i^*\nabla\phi_i(x^*)=0, \tag{6.15}$$

and with respect to λ

$$\phi_i(x^*)=0,\quad i=1,\dots,m, \tag{6.16}$$

must hold. Observe that (6.16) is, precisely, the constraint condition (6.13), and that (6.15) is the condition (6.14). We deduce that any x^* such that (x^*,λ^*) is a critical point of $L(x,\lambda)$ is a candidate to be a minimum for the constrained problem.

Example 6.7 Let $f(x_1,x_2)=-x_2$ and $\phi(x_1,x_2)=x_1^2+x_2^2-1$ ($n=2$, $m=1$). The set of constraints is, then, the circumference

$$U=\{(x_1,x_2)\in\mathbb{R}^2 : x_1^2+x_2^2=1\}.$$

The Lagrangian function is given by

$$L(x_1,x_2,\lambda)=-x_2+\lambda(x_1^2+x_2^2-1).$$

The critical points are determined by

$$\begin{aligned}
0&=\frac{\partial L}{\partial x_1}(x^*,\lambda^*)=2\lambda x_1,\\
0&=\frac{\partial L}{\partial x_2}(x^*,\lambda^*)=-1+2\lambda x_2,\\
0&=\frac{\partial L}{\partial \lambda}(x^*,\lambda^*)=x_1^2+x_2^2-1.
\end{aligned}$$

Solving, we get $x_1^*=0$, $x_2^*=\pm 1$ and $\lambda^*=1/2x_2^*$. □

We finish this section making explicit the sufficient conditions of second order for a constrained minimum with equality restrictions.

Theorem 6.5 (Sufficient conditions for constrained problems) *Let $x^*\in U$, with U the set of constraints given by (6.13) and $\lambda\in\mathbb{R}^m$ such that (6.14) holds. Suppose that the Hessian matrix of L, with respect to x, given by*

$$H(x^*)=H_f(x^*)+\lambda^T H_\phi(x^*)$$

is positive definite in the set $M=\{y\in\mathbb{R}^m : \nabla\phi(x^*)^T y=0\}$. Then x^* is a constrained minimum of f in the set U.

Observe that in the previous example, we have

$$H(x^*)=\begin{pmatrix}0&0\\0&0\end{pmatrix}+\lambda^*\begin{pmatrix}2&0\\0&2\end{pmatrix},\quad M=\{(y_1,y_2)\in\mathbb{R}^2 : x_2^*y_2=0\}.$$

Therefore, $H(x^*)$ is positive definite only for $x^*=(0,1)$. The other critical point of the Lagrangian, $(0,-1)$, corresponds to a constrained maximum.

3.2 The penalty method

Like in the Lagrange multipliers method, the penalty method consists on transformin a constrained problem to a problem without constraints. However, in this case the constraints may be far more general than just of equality. According to the notation given in the introduction, the problem is stated as

$$\min_{x \in S} f(x). \tag{6.17}$$

The idea of the penalty method is replacing the objective function, $f(x)$, by another function

$$f(x) + cP(x) \tag{6.18}$$

and solving the unconstrained problem for the new function. To do this, we take c as a positive constant and a function P satisfying the conditions (P):

1. P is continuous in Ω,
2. $P(x) \geq 0$ for $x \in \Omega$, and
3. $P(x) = 0$ if and only if $x \in S$.

Example 6.8 Suppose that S is given by m inequality constraints,

$$S = \{x \in \mathbb{R}^N : \phi_i(x) \leq 0, \quad i = 1, \ldots, m\}.$$

An example of penalty function is

$$P(x) = \frac{1}{2} \sum_{i=1}^{m} \max(0, \phi_i(x))^2.$$

In Figure 6.3 we can see an example of function $cP(x)$ in the one-dimensional case, with $\phi_1(x) = x - b$ and $\phi_2(x) = a - x$. For c large, the minimum of function (6.18) must lie in a region where P is small. Thus, by increasing c we expect that the corresponding points of minimum will approximate the set S and, if they are close to each other, they will also minimize f. Ideally, when $c \to \infty$, the solution to the penalty problem converges to the solution of the constrained problem (6.17). □

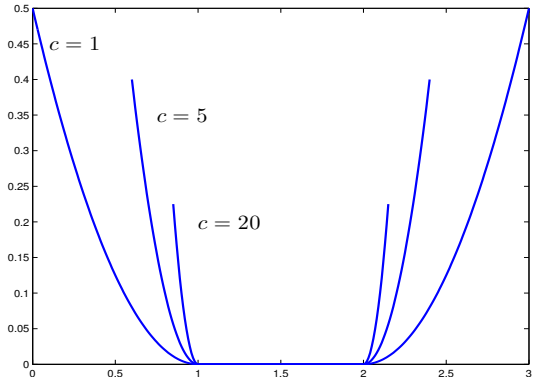


Figure 6.3: Function $cP(x)$ for several values of c.

The procedure to solve the constrained problem (6.17) by the penalty method is as follows: Let c_k be a sequence such that, for all $k = 1, 2, \ldots$, satisfy the conditions (C):

- $c_k \geq 0$
- $c_{k+1} > c_k$,
- $\lim_{k\to\infty} c_k = \infty$.

Define the functions

$$q(c,x) = f(x) + cP(x). \tag{6.19}$$

For each k, assume that the problem $\min q(c_k,x)$ has a solution, x_k. We have the following result.

Theorem 6.6 *Let x_k be a sequence generated by the penalty method. Then, any limit point of the sequence is the solution of the constrained minimization problem (6.17).*

Observe that the problem (6.19) may be solved, for instance, by Newton's method. In the proof of this theorem we shall use the following auxiliary result.

Lemma 1 *For all $k = 1,2,\ldots$, we have*

$$q(c_k,x_k) \leq q(c_{k+1},x_{k+1}), \tag{6.20}$$
$$P(x_k) \geq P(x_{k+1}), \tag{6.21}$$
$$f(x_k) \leq f(x_{k+1}). \tag{6.22}$$

In addition, if x^* is a solution of the constrained problem (6.17) then

$$f(x^*) \geq q(c_k,x_k) \geq f(x_k). \tag{6.23}$$

Proof. We have

$$\begin{aligned} q(c_{k+1},x_{k+1}) &= f(x_{k+1}) + c_{k+1}P(x_{k+1}) \geq f(x_{k+1}) + c_kP(x_{k+1}) \\ &\geq f(x_k) + c_kP(x_k) = q(c_k,x_k), \end{aligned}$$

proving (6.20). We also have

$$f(x_k) + c_kP(x_k) \leq f(x_{k+1}) + c_{k+1}P(x_{k+1}), \tag{6.24}$$
$$f(x_{k+1}) + c_{k+1}P(x_{k+1}) \leq f(x_k) + c_{k+1}P(x_k). \tag{6.25}$$

Adding (6.24) to (6.25) we get

$$(c_{k+1} - c_k)P(x_{k+1}) \leq (c_{k+1} - c_k)P(x_k),$$

proving (6.21). Moreover,

$$f(x_k) + c_kP(x_k) \leq f(x_{k+1}) + c_kP(x_{k+1}),$$

and using (6.21) we get (6.22). Finally, if x^* is solution of (6.17) then $P(x^*) = 0$, and therefore

$$f(x^*) = f(x^*) + c_kP(x^*) \geq f(x_k) + c_kP(x_k) \geq f(x_k),$$

proving (6.23).□

Proof of Theorem 6.6. Suppose that $\bar{x}$ is a limit point of some subsequence of x_k, denoted by $\bar{x}_k$. By continuity, we have

$$\lim_{k\to\infty} f(\bar{x}_k) = f(\bar{x}). \tag{6.26}$$

Let M the minimum value corresponding to problem (6.17). According to Lemma 1, the sequence of values $q(c_k, x_k)$ is not decreasing and bounded by M. Therefore, there exists a $q^* \in \mathbb{R}$ such that

$$\lim_{k\to\infty} q(c_k, \bar{x}_k) = q^* \leq M. \tag{6.27}$$

Subtracting (6.26) from (6.27) we get

$$\lim_{k\to\infty} c_k P(\bar{x}_k) = q^* - f(\bar{x}). \tag{6.28}$$

Since $P(\bar{x}_k) \geq 0$ and $c_k \to \infty$, (6.28) implies $\lim_{k\to\infty} P(\bar{x}_k) = 0$. Using the continuity of P, this implies $P(\bar{x}) = 0$, and hence $\bar{x}$ satisfies the constraint $\bar{x} \in S$. Finally, using (6.23) we deduce $f(\bar{x}_k) \leq M$, and then $f(\bar{x}) = \lim_{k\to\infty} f(\bar{x}_k) \leq M$. □

Example 6.9 Minimize $f(x,y) = x^2 + 2y^2$ in the set $S = \{(x,y) \in \mathbb{R}^2 : x + y \geq 1\}$. We define the differentiable penalty function

$$P(x,y) = \begin{cases} 0 & \text{if } (x,y) \in S, \\ (x+y-1)^2 & \text{if } (x,y) \in \mathbb{R}^2 \backslash S, \end{cases}$$

$c_k = k$, and $q_k(x,y) = f(x,y) + c_k P(x,y)$. Observe that function P satisfies conditions (P), and that the sequence c_k satisifes conditions (C). In practice, we would apply a numerical method such as the gradient method to solve the unconstrained minimization of q_k. In this example, we shall compute the exact solution. We start computing the critical points.

If $(x,y) \in S$ is a critical point of q_k then

$$\nabla q_k(x,y) = (2x, 4y) = (0,0).$$

However, the unique solution to this equation is $(0,0) \notin S$. Therefore, we disregard this point. If $(x,y) \in \mathbb{R}^2 \backslash S$ is a critical point of q_k then

$$\nabla q_k(x,y) = (2(1+k)x + 2ky - 2k, 2kx + 2(2+k)y - 2k) = (0,0),$$

with the unique solution given by

$$(x_k^*, y_k^*) = \Big(\frac{2k}{3k+2}, \frac{k}{3k+2}\Big),$$

and since $x_k^* + y_k^* = 3k/(3k+2) < 1$, we have indeed $(x_k^*, y_k^*) \in \mathbb{R}^2 \backslash S$, for any $k = 1, 2, \ldots$ Finally, the exact minimum of f is obtained taking the limit $k \to \infty$, which gives $(x^*, y^*) = (2/3, 1/3) \in S$.

□

Appendix. Some fundamental definitions and results

Let $x \in \mathbb{R}^n$. The *Euclidean norm of* x is defined as

$$\|x\| = \Big(\sum_{i=1}^{n} x_i^2\Big)^{1/2},$$

and the ℓ^∞ *norm of* x is given by

$$\|x\|_\infty = \max_{i=1,\dots,n} |x_i|.$$

A *square matrix*, A, *of order* n is and ordered collection of numbers, $a_{ij} \in \mathbb{R}$, for $i, j = 1, \dots, n$,

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix}.$$

We often use the notation $A = (a_{ij})$, when the order of the matrix is clear from the context.

The *transpose* of A, denoted by A^T, is another matrix obtained interchanging the rows and columns of A, that is

$$A^T = \begin{pmatrix} a_{11} & a_{21} & \cdots & a_{n1} \\ a_{12} & a_{22} & \cdots & a_{n2} \\ \vdots & \vdots & \vdots & \vdots \\ a_{1n} & a_{2n} & \cdots & a_{nn} \end{pmatrix}.$$

A square matrix, A, is *symmetric* if $A = A^T$. A square matrix, A, is *positive definite* if A is symmetric and

$$x^T A x > 0 \quad \text{for all } x \in \mathbb{R}^n, \quad x \neq 0.$$

If the inequality is not strict, A is said to be *positive semidefinite*.

Vector norms induce matrix norms in the following way:

$$\|A\| = \max_{x \neq 0} \frac{\|Ax\|}{\|x\|}.$$

The *characteristic polynomial* of a square matrix of order n is a polynomial, P, of degree n, which is invariant under matrix similarity (linear combinations of rows and columns),

$$P(\lambda) = \det(A - \lambda I_n),$$

where I_n is the identity matrix of order n. The n roots of the characteristic polynomial, λ_i, for $i = 1,\ldots,n$, are called the *eigenvalues* of A, which may be real or complex numbers. If A is symmetric, then $\lambda_i \in \mathbb{R}$, for all $i = 1,\ldots,n$. In addition, if A is definite positive then $\lambda_i > 0$, for all $i = 1,\ldots,n$. The *spectral radius*, ρ, of A is given by the maximum eigenvalue in absolute value, that is,

$$\rho = \max_{i=1,\ldots,n} |\lambda_i|.$$

Let $\Omega \subset \mathbb{R}^n$ be an open set, and $f : \Omega \to \mathbb{R}$ be twice continuously differentiable. The *partial derivative of f with respect to x_i*, evaluated at a point $x \in \Omega$, is denoted as

$$\frac{\partial f}{\partial x_i}(x).$$

Partial derivatives of higher order are defined by composition of partial derivatives of first order. For instance

$$\frac{\partial^2 f}{\partial x_i \partial x_j}(x)$$

is the second partial derivative of f with respect to x_i and x_j, evaluated in x. An important property of second partial derivatives is that they are independent of the order of derivation, i.e.

$$\frac{\partial^2 f}{\partial x_i \partial x_j}(x) = \frac{\partial^2 f}{\partial x_j \partial x_i}(x). \tag{A.29}$$

The *gradient* of f in x is the vector

$$\nabla f(x) = \Big(\frac{\partial f}{\partial x_1}(x), \ldots, \frac{\partial f}{\partial x_n}(x)\Big).$$

The second order partial derivatives of f are often collected into a matrix, called the *Hessian of f*,

$$H_f(x) = \begin{pmatrix} \frac{\partial^2 f}{\partial x_1^2}(x) & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n}(x) \\ \frac{\partial^2 f}{\partial x_2 \partial x_1}(x) & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n}(x) \\ \vdots & \vdots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1}(x) & \cdots & \frac{\partial^2 f}{\partial x_n^2}(x) \end{pmatrix}. \tag{A.30}$$

Due to (A.29), the Hessian matrix is symmetric. The trace of the Hessian of f, i.e. the sum of the elements of the main diagonal, is called the *Laplacian of f* in x, and denoted as $\Delta f(x)$. That is,

$$\Delta f(x) = \sum_{i=1}^{n} \frac{\partial^2 f}{\partial x_i^2}(x). \tag{A.31}$$

We say that a set $\Omega \subset \mathbb{R}^n$ is *convex* if for all $x,\ y \in \Omega$, and for all $t \in [0,1]$

$$tx + (1-t)y \in \Omega.$$

A function $f : \Omega \subset \mathbb{R}^n \to \mathbb{R}$ is *convex in the convex set* Ω if, for all $x, y \in \Omega$, and for all $t \in [0,1]$,

$$f(tx + (1-t)y) \leq tf(x) + (1-t)f(y).$$

f is called *strictly convex* if the above inequality is strict for all $x \neq y$ and $t \in (0,1)$.

If f is twice continuously differentiable then it is convex in the convex set, Ω, if and only if $H_f(x)$ is positive semidefinite for all $x \in \Omega$.

Let $\Omega \subset \mathbb{R}^n$ be an open set, and $f : \Omega \to \mathbb{R}^m$ be a vector function, $\mathbf{f} = (f_1, \ldots, f_m)$, continuously differentiable. The *Jacobian matrix* of $\mathbf{f}$ is the $m \times n$ matrix given by

$$J_f(x) = \begin{pmatrix} \frac{\partial f_1}{\partial x_1}(x) & \ldots & \frac{\partial f_1}{\partial \partial x_n}(x) \\ \frac{\partial f_2}{\partial x_1}(x) & \ldots & \frac{\partial f_2}{\partial x_n}(x) \\ \vdots & \vdots & \vdots \\ \frac{\partial f_m}{\partial x_1}(x) & \ldots & \frac{\partial f_m}{\partial x_n}(x) \end{pmatrix}. \tag{A.32}$$

If $m = n$ then the Jacobian of $\mathbf{f}$ is a square matrix, whose determinant is called the *Jacobian determinant* of $\mathbf{f}$ in x, usually denoted as $|J_f(x)|$. Also, in the case $m = n$, the trace of $J_f(x)$ has a name, *the divergence of* $\mathbf{f}(x)$, denoted by $\operatorname{div} f(x)$. That is,

$$\operatorname{div} \mathbf{f}(x) = \sum_{i=1}^{n} \frac{\partial f_i}{\partial x_i}(x). \tag{A.33}$$

For a real function $f : \Omega \to \mathbb{R}$, the composition of the gradient and the divergence gives the Laplacian,

$$\Delta f(x) = \operatorname{div}\big(\nabla f(x)\big).$$

Taylor's expansion is an useful tool we shall often use.

Theorem 1.7 (Taylor) *Let f be twice continuously differentiable in a neighborhood of a point $x^* \in \mathbb{R}^n$. Then , for all $e \in \mathbb{R}^n$ with $\|e\|$ small enough, we have*

$$f(x^* + e) = f(x^*) + \nabla f(x^*)^T e + \frac{1}{2} e^T H_f(x) e + o(\|e\|^2). \tag{A.34}$$

Recall that a *neighborhood of radius* ρ *centered in* x^* is the set $B_\rho(x^*) = \{x \in \mathbb{R}^n : \|x - x^*\| < \rho\}$ (an $n-$dimensional ball). The notation $o(t^2)$ (*small o*) means

$$\lim_{t \to 0} \frac{o(t^2)}{t^2} = 0.$$

Bibliography

[1] R. Burden, J. D. Faires, Numerical methods, Brooks/Cole Cengage Learning, Boston, 2010.

[2] S. C. Chapra, R. P. Canale, Numerical methods for engineers, McGraw Hill, 2009 .

[3] D. G. Luenberger, Linear and nonlinear programming, Kluwer, Norwell, 2003.

[4] A. Quarteroni, F. Saleri, P. Gervasio, Scientific computing with Matlab and Octave, Springer-Verlag, Berlin, 2010.