# Contents

1 Polynomial interpolation .......................... 2
   1 Preliminaries .................................. 3
      1.1 Polynomials ................................ 3
      1.2 Rolle's theorem ............................. 3
      1.3 Polynomial evaluation : Horner scheme .... 4
   2 Polynomial interpolation problem ............... 5
      2.1 Existence and uniqueness of a solution ...... 5
      2.2 Interpolation polynomial of a function .... 5
      2.3 Approaches for determining the interpolating polynomial ............................ 6
   3 Interpolation in the monomial basis .............. 6
   4 Interpolation in the Lagrange basis .............. 7
      4.1 Lagrange basis ................................ 8
      4.2 The interpolating polynomial .............. 9
      4.3 Example ..................................... 9
      4.4 More properties of Lagrange polynomials ... 10
   5 Interpolation in the Newton basis ................. 10
      5.1 Divided differences ........................... 11
         5.1.1 Triangular scheme ....................... 11
         5.1.2 Some properties .......................... 11
      5.2 Newton basis ................................ 12
      5.3 The interpolating polynomial .............. 13
      5.4 Example ..................................... 14
      5.5 Implementation of divided differences scheme ............................ 15
   6 Error bounds in Lagrange interpolation .......... 18
   7 Chebyshev interpolation points .................. 19
      7.1 An example ................................... 19
      7.2 Statement of the problem .................... 21
      7.3 Chebyshev polynomials ....................... 21
      7.4 Optimal distribution of points ............. 22
      7.5 Alternating property ......................... 24
   8 Convergence ................................... 24
      8.1 Runge’s phenomenon ........................... 25
      8.2 Some convergence results .................... 26
      8.3 Lebesgue constant ............................ 27
   9 Overview on parametric interpolation ............. 29
  10 An example with Scipy ............................ 30
  11 Exercises ..................................... 31
Chapter 1

Polynomial interpolation

Introduction

This chapter should be more precisely entitled Polynomial Lagrange interpolation. Lagrange interpolation consists in determining a curve which passes through predetermined positions (the interpolation points), while Hermite interpolation requires to satisfy additional constraints on the derivatives at the interpolating points. The mathematical model chosen to perform this interpolation depends in particular on the application context and the number of points.

Given a sequence of \( n + 1 \) points \((x_i, y_i), i = 0, 1, \ldots, n\), with \( a \leq x_0 < x_1 < \cdots < x_{n-1} < x_n \leq b\), a simple solution consists in constructing the unique continuous piecewise linear function passing through these points, as shown in figure 1.1-(a). This solution has the disadvantage of not being smooth, but it reproduces precisely the geometry of the initial data \((x_i, y_i)\).

Polynomial Lagrange interpolation consists in determining the unique polynomial \( p(x) \) of degree \( n \) satisfying \( p(x_i) = y_i \) for \( i = 0, 1, \ldots, n\), thus producing a solution of class \( C^\infty \), see Figure 1.1-(b). As the number of points increases, the polynomial interpolant can however “oscillate”, and thus does not reproduce faithfully the nature of the initial data, see also figure 1.2.

An alternative approach consists in joining “simple” curves (typically, polynomials of low degree) so as to form a smooth composite curve (of class \( C^k \), \( k = 1, 2, \ldots \) on \( [a, b] \)) and to require this composite curve to go through points \((x_i, y_i)\). Such a composite curve is called a spline curve. The \( C^2 \) cubic spline curves are probably the most common and used splines, see figure 1.1-(c). Notice that figure 1.1-(a) actually exhibits a \( C^0 \) spline curve of degree one. Chapter ?? is devoted to the study of some interpolating splines.

![Figure 1.1: Interpolation of the same data \((x_i, y_i)\) by three different mathematical models: (a) a continuous piecewise linear function, (b) a polynomial function of degree 10, (c) a \( C^2 \) cubic spline, see chapter ??]. For comparison, the polynomial interpolant (b) is plotted in dotted line in figure (c).

Interpolation can be performed with different kinds of function, e.g., trigonometric functions. However, in this chapter, we are only concerned with polynomial interpolation (i.e., polynomial Lagrange

\(^{1}\)Joseph-Louis Lagrange, 1736-1813, Italian mathematician
CHAPTER 1. POLYNOMIAL INTERPOLATION

1 Preliminaries

We recall some useful results for the rest of this chapter.

1.1 Polynomials

Let \( p(x) = a_0 + a_1x + a_2x^2 + \cdots + a_nx^n \) be a polynomial of degree \( n \) (which means that \( a_n \neq 0 \)).

1. \( x \in \mathbb{R} \) (or \( \mathbb{C} \)) is a root of multiplicity \( k \) of \( p(x) \) if and only if
   \[
   p(x) = p'(x) = p''(x) = \cdots = p^{(k-1)}(x) = 0 \quad \text{and} \quad p^{(k)}(x) \neq 0,
   \]
   or, equivalently, if \( p(x) \) can be factored by \( (x - x)^k \) and not by \( (x - x)^{k+1} \), which means that there exists a polynomial \( q(x) \) of degree \( n - k \) such that
   \[
   p(x) = (x - x)^k q(x) \quad \text{and} \quad q(x) \neq 0.
   \]
   \( x \) is called a \textit{simple} root for \( k = 1 \), and a \textit{multiple} root for \( k \geq 2 \).

2. Fundamental theorem of algebra: any polynomial of degree \( n \), with real or complex coefficients, admits \textit{exactly} \( n \) roots (real or complex) if they are counted with their multiplicity:
   \[
   p(x) = a_n (x - z_1)^{k_1} (x - z_2)^{k_2} \cdots (x - z_r)^{k_r} \quad \text{with} \quad k_1 + k_2 + \cdots + k_r = n,
   \]
   where \( z_1, z_2, \ldots, z_r \) are all the distinct roots of \( p(x) \).

3. Thus, if \( p(x) \) admits \( n + 1 \) roots counted with their multiplicity, then \( p(x) \) is the polynomial identically zero. Consequently, if two polynomials of degree \( n \) coincide in \( n + 1 \) distinct values, they must be identical.

1.2 Rolle’s theorem

1. \textit{Mean value theorem}.
   Given a real-valued function \( f \), continuous on an interval \([a, b]\) and differentiable on the open interval \(]a, b[\), then there exists a point \( c \in ]a, b[ \) such that
   \[
   f'(c) = \frac{f(b) - f(a)}{b - a},
   \]
   which means there exists at least one point at which the tangent to the graph of the function is parallel to the secant through its endpoints.

2. \textit{Rolle’s theorem}.
   Rolle’s theorem can be viewed as a particular case of the mean value theorem. It states that a differentiable function that attains equal values at two distinct points must have a stationary point between these two points.
   Precisely, given a real-valued function \( f \), continuous on an interval \([a, b]\), differentiable on the open interval \(]a, b[\) and with \( f(a) = f(b) \), then there exists a point \( c \in ]a, b[ \) such that
   \[
   f'(c) = 0.
   \]
1.3 Polynomial evaluation: Horner scheme

Let \( p(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n \).

Algorithms involving polynomials, and in particular plotting the graph of a polynomial function over an interval \([a, b]\), require the evaluation of \( p(t_i) \) for many values \( t_i = a + i * (b - a) / N, \) \( i = 0, 1, \ldots, N, \) which justify the interest of developing efficient algorithms for such evaluations.

1. Naïve algorithm: requires \( 2n \) multiplications for the evaluation of \( p(x) \).

   ```python
   #initialization data :
   # x = a real
   # a = (a[0],a[1],...,a[n]) = vector of polynomial coefficients
   xn = 1
   p = a[0]
   for i = 1 until n do  # n steps
       xn = xn * x      # 1 multiplication
       p = p + a[i] * xn # 1 multiplication
   endfor
   return p            # p = p(x)
   ``

2. Horner scheme: requires only \( n \) multiplications for the evaluation of \( p(x) \). The Horner scheme (also named the Ruffini-Horner’s method) is based on the following factorization, illustrated here with a polynomial of degree 4

\[
p(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + a_4 x^4
= a_0 + x \left( a_1 + x \left( a_2 + x \left( a_3 + x (a_4) \right) \right) \right)
\]

   ```python
   #initialization data :
   # x = a real
   # a = (a[0],a[1],...,a[n]) = vector of polynomial coefficients
   p = a[n]
   for i = n-1 until 0 with step = -1 do  # n steps
       p = a[i] + (x - t[i]) * p          # 1 multiplication
   endfor
   return p                            # p = p(x)
   ``

3. Horner scheme relative to Newton basis, see section 5: requires only \( n \) multiplications and is based on the following factorization.

\[
p(x) = d_0 + d_1 (x - t_0) + d_2 (x - t_0) (x - t_1) + d_3 (x - t_0) (x - t_1) (x - t_2)
= d_0 + (x - t_0) \left( d_1 + (x - t_1) \left( d_2 + (x - t_2) d_3 \right) \right)
\]

   ```python
   #initialization data :
   # x = a real
   # d = (d[0],d[1],...,d[n]) = vector of coefficients = divided differences
   # t = (t[0],t[1],...,t[n]) = interpolation points
   p = d[n]
   for i = n-1 until 0 with step = -1 do
       p = d[i] + (x - t[i]) * p
   endfor
   return p  # p = p(x)
   ```
2 Polynomial interpolation problem

Given \( n + 1 \) real distinct points \( x_0, x_1, \ldots, x_n \) and \( n + 1 \) real numbers \( y_0, y_1, \ldots, y_n \), we look for a polynomial \( p(x) \) satisfying:

\[
p(x_i) = y_i, \quad i = 0, 1, \ldots, n.
\]

The number of constraints (equal to \( n + 1 \)) leads naturally to search a polynomial of degree \( n \), that is an element in \( \mathbb{R}_n[x] \), which is a vector space of dimension \( n + 1 \).

2.1 Existence and uniqueness of a solution

Consider the linear map \( \Phi: \mathbb{R}_n[x] \rightarrow \mathbb{R}^{n+1} \):

\[
p \rightarrow \Phi(p) = (p(x_0), p(x_1), \ldots, p(x_n)).
\]

**Proposition 1**

If points \( x_0, x_1, \ldots, x_n \) are all distinct, the linear map \( \Phi \) is bijective.

**Proof:** The kernel of \( \Phi \) consists of polynomials of degree less than or equal to \( n \) which cancel for each one of the \( n + 1 \) points \( x_i \). By our preliminaries (item 3) in section 1.1 we deduce that this kernel only contains the zero polynomial, so that \( \Phi \) is injective. Finally, by the rank-nullity theorem

\[
\dim(\mathbb{R}_n[x]) = \dim \ker(\Phi) + \dim \text{Im}(\Phi).
\]

the linear map \( \Phi \) is bijective. \( \blacksquare \)

**Corollary 1**

For any family of \( n + 1 \) real numbers \( y_0, y_1, \ldots, y_n \), there exists a **unique** polynomial \( p \in \mathbb{R}_n[x] \) satisfying the constraints

\[
p(x_i) = y_i, \quad i = 0, 1, \ldots, n.
\]

This polynomial is called the Lagrange interpolating polynomial (or simply, the interpolating polynomial) of the data \( (x_i, y_i) \).

2.2 Interpolation polynomial of a function

Given a sufficiently smooth function \( f \) defined on an interval \([a, b]\), and \( n+1 \) distinct points \( x_0, x_1, \ldots, x_n \) in \([a, b]\), the **interpolating polynomial of \( f \)** at the \( n + 1 \) distinct points \( x_i \) is the Lagrange interpolating polynomial associated with the data \((x_i, f(x_i))\), for \( i = 0, 1, \ldots, n \). This polynomial is denoted \( P(x, f) \) or more precisely by \( P_n(x, f) \). In other words, the ordinates \( y_i \) come here from the sampling of a function at \( n + 1 \) distinct points.

Figure 1.2 shows the interpolating polynomials \( P(x, f) \) for the two functions \( f_1(x) = 10 \cos(x/2) \) and \( f_2(x) = -5 + 15/(1 + x^2) \), at interpolation points \( x_i \) uniformly distributed in the interval \([a, b] = [-10, 10]\). Notice the difference in behaviour when the number of interpolation points increases: it seems to converge in one case and we have oscillations in the other case.

**Proposition 2**

The Lagrange interpolation process is exact over the set of polynomial of degree less than or equal to \( n \)

\[
\forall p(x) \in \mathbb{R}_n[x], \quad P(x, p) = p(x).
\]

**Proof:** This is a direct consequence of item 3 in section 1.1 (preliminaries). \( \blacksquare \)
Figure 1.2: Graph of functions $f_1$ (first row) and $f_2$ (second row) are plotted in dotted line and their interpolating polynomials are plotted in solid line.

2.3 Approaches for determining the interpolating polynomial

The method for determining the interpolating polynomial depends essentially on the basis selected to express this polynomial. We first consider the determination of this polynomial relative to the monomial (or canonical) basis. We then consider the determination of the interpolating polynomial relative to two basis specifically adapted to the Lagrange interpolation problem: the Lagrange basis and the Newton basis.

3 Interpolation in the monomial basis

Let $p(x)$ be a $n$-degree polynomial expressed in the monomial basis (canonical basis)

$$p(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n,$$

so that interpolation constraints lead to the $(n+1) \times (n+1)$ linear system

$$
\begin{align*}
    a_0 + a_1 x_0 + a_2 x_0^2 + \cdots + a_n x_0^n &= y_0 \\
    a_0 + a_1 x_1 + a_2 x_1^2 + \cdots + a_n x_1^n &= y_1 \\
    a_0 + a_1 x_2 + a_2 x_2^2 + \cdots + a_n x_2^n &= y_2 \\
    \vdots & \vdots \vdots \vdots \vdots \\
    a_0 + a_1 x_n + a_2 x_n^2 + \cdots + a_n x_n^n &= y_n
\end{align*}
$$

which can be written in matrix form

$$
\begin{pmatrix}
1 & x_0 & x_0^2 & \cdots & x_0^n \\
1 & x_1 & x_1^2 & \cdots & x_1^n \\
1 & x_2 & x_2^2 & \cdots & x_2^n \\
\vdots & \vdots & \vdots & \cdots & \vdots \\
1 & x_n & x_n^2 & \cdots & x_n^n \\
\end{pmatrix}
\begin{pmatrix}
a_0 \\
a_1 \\
a_2 \\
\vdots \\
a_n \\
\end{pmatrix} =
\begin{pmatrix}
y_0 \\
y_1 \\
y_2 \\
\vdots \\
y_n \\
\end{pmatrix}
$$
The matrix of this system is called the Vandermonde matrix. Its determinant (the Vandermonde determinant), see exercise ??, is

$$\prod_{0\leq i<j\leq n} (x_j - x_i)$$

which is obviously non zero if and only if the points $x_i$ are all distinct.

Unfortunately, the numerical resolution of this Vandermonde system is intricate, and even more... as this linear system is ill-conditioned. This means that a small error on the coefficients or on the second member leads to a major error on the solution of the system. In conclusion, this method is not recommended for the calculation of the interpolating polynomial, it is numerically unstable and costly.

Anyway, the evaluation of a polynomial expressed in the canonical basis has to be performed by means of the Horner scheme, see section 1.3.

```python
def InterpolVand(xi, yi):
    """ Polynomial interpolation of data (xi, yi) in the canonical basis
    (==> Vandermonde system)
    Input:
    xi = vector of interpolation abscissa
    yi = vector of ordinates
    output : cf = monomial coefficients of the interpolating polynomial
    ""
    N = np.size(xi)
    V = np.ones((N,N)) # Vandermonde matrix
    for k in range(1,N):
        V[:,k] = xi**k
    cf = np.linalg.solve(V, yi) # calculation of monomial coefficients
    return cf
```

**Exercise 1**

We propose to calculate the determinant of the Vandermonde matrix

$$V(x_0, x_1, \ldots, x_n) = \begin{pmatrix}
1 & x_0 & x_0^2 & \cdots & x_0^{n-1} & x_0^n \\
1 & x_1 & x_1^2 & \cdots & x_1^{n-1} & x_1^n \\
1 & x_2 & x_2^2 & \cdots & x_2^{n-1} & x_2^n \\
\vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\
1 & x_n & x_n^2 & \cdots & x_n^{n-1} & x_n^n 
\end{pmatrix}$$

where the $x_i$ are distinct values. With the following operations on columns $C_j$:

$$C_j := C_j - x_0 C_{j-1}, \quad \text{for } j = n+1, n, n-1, \ldots, 2,$$

deduce a relation between $\det(V(x_0, x_1, \ldots, x_n))$ and $\det(V(x_1, \ldots, x_n))$.

Then conclude by recurrence.

**4 Interpolation in the Lagrange basis**

Lagrange basis associated with interpolation points $x_i$ leads to a very simple expression of the interpolating polynomial.
4.1 Lagrange basis

Precisely, Lagrange polynomials $L_i(x)$ associated with the $n + 1$ distinct points

$$x_0, x_1, x_2, \ldots, x_n,$$

are defined by

$$L_i(x) = \prod_{\substack{0 \leq j \leq n \ j \neq i}} \frac{x - x_j}{x_i - x_j}, \quad 0 \leq i \leq n.$$

For $n = 0$, the unique Lagrange polynomial $L_0(x) = 1$, $x \in \mathbb{R}$, associated with the unique point $x_0$ is in that case independent of that point $x_0$.

![Figure 1.3: Lagrange polynomials $L_k(x)$ of degree 5, associated with points uniformly distributed in the interval $[-2, 4]$.](image)

The following properties are readily verified.

**Proposition 3**

For $0 \leq i, j \leq n$ we have

$$L_i(x_j) = \delta_{ij}.$$

**Proposition 4**

Lagrange polynomials $L_i(x)$, $0 \leq i \leq n$, form a basis of $\mathbb{R}_n[x]$, referred as the Lagrange basis (relative to points $x_i$ or associated with points $x_i$).

**Proof**: Since $\dim(\mathbb{R}_n[x]) = n + 1$, we just need to prove that the $n + 1$ polynomials $L_i(x)$ are linearly independents. Consider thus a linear combination equal to zero:

$$\sum_{i=0}^{n} \alpha_i L_i(x) = 0, \forall x \in \mathbb{R}.$$

For $x = x_j$ ($0 \leq j \leq n$), we have

$$\sum_{i=0}^{n} \alpha_i L_i(x_j) = \sum_{i=0}^{n} \alpha_i \delta_{ij} = \alpha_j = 0$$

We thus deduce that $\alpha_0 = \alpha_1 = \cdots = \alpha_n = 0$, which proves that the family of polynomials $L_i(x)$ is linearly independent and thus a basis of $\mathbb{R}_n[x]$.■
4.2 The interpolating polynomial

Proposition 5

The interpolating polynomial associated with data \((x_i, y_i)\), i.e., the unique polynomial \(p \in \mathbb{R}_n[x]\) such that \(p(x_i) = y_i\) for \(i = 0, 1, \ldots, n\), is a linear combination of polynomial \(L_i(x)\)

\[
p(x) = \sum_{i=0}^{i=n} y_i L_i(x).
\]

This polynomial is the interpolating polynomial in the Lagrange form.

Thus, the coefficients of the interpolating polynomial in the Lagrange basis associated with points \(x_i\) are simply the ordinates \(y_i\). In spite of its simple writing, this expression of the polynomial of interpolation is not always the best adapted to calculations. Indeed, each polynomial \(L_k(x)\) depends on the set of all interpolation points \(x_i\). See example below.

The use of Lagrange basis is relevant when several interpolations associated with the same data set of interpolation \(x_i\) have to be performed (which means that only data \(y_i\) change). However, if an interpolation point datum \(x_i\) is added or modified, all calculations must be fully resumed.

4.3 Example

Consider the interpolation of the polynomial function \(f(x) = 4 - x - 4x^2 + x^3\) at 4 points \(x_i\) given, together with their associate ordinates \(y_i = f(x_i)\), in the following table.

<table>
<thead>
<tr>
<th>(x_0 = 1)</th>
<th>(x_1 = 3)</th>
<th>(x_2 = -1)</th>
<th>(x_3 = 2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(y_0 = f(x_0) = 0)</td>
<td>(y_1 = f(x_1) = -8)</td>
<td>(y_2 = f(x_2) = 0)</td>
<td>(y_3 = f(x_3) = -6)</td>
</tr>
</tbody>
</table>

More precisely, we first interpolate \(f\) at points \(x_0\) and \(x_1\), and then we add successively the interpolation points \(x_2\) and \(x_3\). We thus get the three interpolating polynomials \(P_k(x, f) (k = 1, 2, 3)\), associated with data \((x_i, f(x_i))\), \(0 \leq i \leq k\). See figure 1.4 below.

These interpolating polynomials are determined in each of the appropriate Lagrange basis associated with points \(x_i\) and also in the canonical basis.

\[
P_1(x, f) = y_0 \frac{x - x_1}{x_0 - x_1} + y_1 \frac{x - x_0}{x_1 - x_0} = 0 \frac{x - 3}{1 - 3} + (-8) \frac{x - 1}{3 - 1} = 4 - 4x
\]

\[
P_2(x, f) = y_0 \frac{(x - x_1)(x - x_2)}{(x_0 - x_1)(x_0 - x_2)} + y_1 \frac{(x - x_0)(x - x_2)}{(x_1 - x_0)(x_1 - x_2)} + y_2 \frac{(x - x_0)(x - x_1)}{(x_2 - x_0)(x_2 - x_1)}
\]

\[
= 0 \frac{(x - 3)(x + 1)}{-4} + (-8) \frac{(x - 1)(x + 1)}{8} + 0 \frac{(x - 1)(x - 3)}{8} = 1 - x^2
\]

\[
P_3(x, f) = y_0 \frac{(x - x_1)(x - x_2)(x - x_3)}{(x_0 - x_1)(x_0 - x_2)(x_0 - x_3)} + y_1 \frac{(x - x_0)(x - x_2)(x - x_3)}{(x_1 - x_0)(x_1 - x_2)(x_1 - x_3)}
\]

\[
+ y_2 \frac{(x - x_0)(x - x_1)(x - x_3)}{(x_2 - x_0)(x_2 - x_1)(x_2 - x_3)} + y_3 \frac{(x - x_0)(x - x_1)(x - x_2)}{(x_3 - x_0)(x_3 - x_1)(x_3 - x_2)}
\]

\[= 4 - x - 4x^2 + x^3\]

We remark that \(P_3(x, f) = f(x)\), which is natural since these two polynomials of degree 3 coincide in 4 distinct values, see item (3) in section 1.1. Finally, it should be noticed that there is no apparent connection in the expressions of polynomials \(P_k(x, f)\) and \(P_{k+1}(x, f)\). The situation will be different in the Newton basis.
4.4 More properties of Lagrange polynomials

The following exercise gives some basic (but important) properties of Lagrange basis polynomials.

**Exercise 2**

With the previous notations for Lagrange polynomials, prove the following properties.

1. **Partition of unity**:
   \[ \sum_{i=0}^{i=n} L_i(x) = 1 \]

2. For all \( j \in \mathbb{N}, 0 \leq j \leq n, \)
   \[ \sum_{i=0}^{i=n} x_i^{j} L_i(x) = x^j \]

3. For all \( j \in \mathbb{N}, 1 \leq j \leq n, \)
   \[ \sum_{i=0}^{i=n} (x - x_i)^j L_i(x) = 0 \]

4. **Derivatives**:
   \[ L'_i(x) = L_i(x) \sum_{j=0}^{n} \frac{1}{x - x_j}, \quad 0 \leq i \leq n. \]  
   \[ (1.1) \]

Check that this expression is valid for each real value \( x \). Give a more detailed formula which does not involve a possible division by zero.

**Solution**:

4. Derivatives at point \( x_i \) and points \( x_j \) with \( j \neq i \):

\[ L'_i(x_i) = \sum_{j=0}^{n} \frac{1}{x_i - x_j} \quad \text{and} \quad L'_i(x_j) = \sum_{j=0}^{n} \left( \frac{1}{x_i - x_j} \prod_{k=0}^{n} \frac{x_j - x_k}{x_i - x_k} \right). \]

5 Interpolation in the Newton basis

Newton basis improves the previous process: each polynomial \( N_k(x) \) of the Newton basis only depends on interpolation points \( x_0, \ldots, x_{k-1} \). We first introduce the divided differences.
5.1 Divided differences

Given a set of real data \((x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n)\), the associated divided differences (or forward divided differences) are defined recursively as follows.

\[
\delta[x_j] := y_j \quad 0 \leq j \leq n \quad \text{(order 0)}
\]

\[
\delta[x_j, x_{j+1}, \ldots, x_{j+p}] := \frac{\delta[x_{j+1}, \ldots, x_{j+p}] - \delta[x_j, x_{j+1}, \ldots, x_{j+p-1}]}{x_{j+p} - x_j} \quad 1 \leq p \leq n \quad 0 \leq j \leq n - p \quad \text{(order p)}
\]

If the set of data comes from a function \(f\), i.e., \(y_j = f(x_j)\), \(0 \leq j \leq n\), the following notations are commonly preferred

\[
f[x_j] := f(x_j) \quad 0 \leq j \leq n \quad \text{(order 0)}
\]

\[
f[x_j, x_{j+1}, \ldots, x_{j+p}] := \frac{f[x_{j+1}, \ldots, x_{j+p}] - f[x_j, x_{j+1}, \ldots, x_{j+p-1}]}{x_{j+p} - x_j} \quad 1 \leq p \leq n \quad 0 \leq j \leq n - p \quad \text{(order p)}
\]

5.1.1 Triangular scheme

Let us determine the first divided differences.

\[
\delta[x_0] := y_0
\]

\[
\delta[x_0, x_1] := \frac{\delta[x_1] - \delta[x_0]}{x_1 - x_0} = \frac{y_1 - y_0}{x_1 - x_0}
\]

\[
\delta[x_0, x_1, x_2] := \frac{\delta[x_{1,2}] - \delta[x_0, x_1]}{x_2 - x_0} = \frac{y_2 - y_1}{x_2 - x_1} - \frac{y_1 - y_0}{x_1 - x_0} = \frac{y_2 - y_1}{(x_2 - x_1)(x_2 - x_0)} - \frac{y_1 - y_0}{(x_1 - x_0)(x_2 - x_0)}
\]

In practice, the divided are calculated according a triangular scheme.

<table>
<thead>
<tr>
<th>(x_0)</th>
<th>(x_1)</th>
<th>(x_2)</th>
<th>(x_3)</th>
<th>(\vdots)</th>
<th>(x_{n-1})</th>
<th>(x_n)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(y_0)</td>
<td>(y_1)</td>
<td>(y_2)</td>
<td>(y_3)</td>
<td>(\vdots)</td>
<td>(y_{n-1})</td>
<td>(y_n)</td>
</tr>
</tbody>
</table>

\[
\delta[x_0, \ldots, x_n] = \delta[x_{n-1}, x_n] \quad \delta[x_0, \ldots, x_{n-1}] \quad \delta[x_1, \ldots, x_n] \quad \delta[x_0, \ldots, x_n]
\]

5.1.2 Some properties

The following exercise gives some basic (but important) properties of divided differences.

Exercise 3

With the previous notations for divided differences, prove the following properties.

1. Expanded form :

\[
\delta[x_0, x_1, \ldots, x_k] = \sum_{j=0}^{k} \frac{y_j}{\prod_{i=0, i \neq j}^{k} (x_j - x_i)}, \quad 0 \leq k \leq n.
\]

2. Symmetry.
Deduce that the divided differences do not depend on the order of the data, that is

\[ \delta[x_{\sigma(0)}, x_{\sigma(1)}, \ldots, x_{\sigma(k)}] = \delta[x_0, x_1, \ldots, x_k] \]

for all permutation \( \sigma \) of the set \( \{0, 1, \ldots, k\} \), \( 0 \leq k \leq n \).

3. Linearity.
Deduce that if \( f = \lambda g + \mu h \), where \( g \) and \( h \) are two functions defined on an interval \([a, b]\) which contains all the points \( x_i \), then

\[ f[x_0, x_1, \ldots, x_k] = \lambda g[x_0, x_1, \ldots, x_k] + \mu h[x_0, x_1, \ldots, x_k] \quad 0 \leq k \leq n. \]

4. Derivatives and mean value theorem.
Assuming that the function \( f \) is sufficiently smooth, we have

\[ f[x_0, x_1, \ldots, x_k] = \frac{f^{(k)}(\xi)}{k!} \quad \text{where} \quad \xi \in (\min(x_i), \max(x_i)) \quad 0 \leq k \leq n. \]

The proof of this last property should be postpone to the end of the next section, as this result is a direct consequence of propositions 7 and 9.

5.2 Newton basis
Newton polynomials associated with the \( n + 1 \) distincts points \( x_0, x_1, x_2, \ldots, x_n \), are defined in the following way:

- \( N_0(x) = 1 \),
- \( N_1(x) = x - x_0 \),
- \( N_2(x) = (x - x_0)(x - x_1) \),
- \( \vdots \)
- \( N_n(x) = (x - x_0)(x - x_1) \cdots (x - x_{n-1}) \).

These polynomials are thus determined by the polynomial \( N_0(x) \) and the recurrence relation

\[ N_{k+1}(x) = (x - x_k) N_k(x), \quad k = 0, 1, \ldots, n - 1. \]

Notice that definition of polynomials \( N_k(x) \) does not involve the “last interpolation point” \( x_n \). However, point \( x_n \) will participate to the calculation of the interpolating polynomial relative to the Newton basis.

**Proposition 6**

Newton polynomials \( N_i(x) \), \( 0 \leq i \leq n \), form a basis of \( \mathbb{R}_n[x] \), referred as the Newton basis relative to (or associated with) points \( x_i \).

**Proof:** Since dim \( (\mathbb{R}_n[x]) = n + 1 \), we just need to prove that the \( n + 1 \) polynomials \( N_i(x) \) are linearly independents. Consider thus a linear combination equal to zero for all \( x \):

\[ \sum_{i=0}^{n} \alpha_i N_i(x) = 0, \quad \forall x \in \mathbb{R}. \]

For \( x = x_0 \), we get \( \sum_{i=0}^{n} \alpha_i N_i(x_0) = \alpha_0 = 0 \). Then, considering successively values \( x_1, x_2, \ldots, x_n \), we deduce that \( \alpha_1 = \alpha_2 = \cdots = \alpha_n = 0 \), which proves that the family of polynomials \( N_i(x) \) is linearly independent and thus a basis of \( \mathbb{R}_n[x] \).
Notation: Given a polynomial \( p(x) \), let us denote by \( \mu_k[p(x)] \) the coefficient of its term of degree \( k \) (monomial of degree \( k \)). As an example, for \( p(x) = 2 - x + 3x^2 - 3x^4 \), we have \( \mu_3[p(x)] = 0 \) and \( \mu_4[p(x)] = -3 \).

Each Newton polynomial \( N_k(x) \) is exactly of degree \( k \) and its leading term (highest degree monomial) is exactly \( x^k \), which means that \( \mu_k[N_k(x)] = 1 \).

Thus, for any polynomial of degree \( k \), expressed in the Newton basis relative to points \( x_i \), we have
\[
\mu_k \left[ d_0 N_0(x) + d_1 N_1(x) + \cdots + d_{k-1} N_{k-1}(x) + d_k N_k(x) \right] = d_k.
\] (1.2)

### 5.3 The interpolating polynomial

For any \( k \geq 0 \), let \( P_k(x) \) be the interpolating polynomial of the data \( (x_i, y_i), \ i = 0, 1, \ldots, k \). The two following propositions allows to determine each interpolating polynomial \( P_k(x) \) in the Newton basis.

**Proposition 7**

For any \( k \geq 1 \), we have
\[
P_k(x) = P_{k-1}(x) + \delta_k \ N_k(x)
\]
with \( \delta_k \) the coefficient of \( P_k(x) \) in the Newton basis polynomial of degree \( k \).

**Proof:** Polynomial \( P_k(x) - P_{k-1}(x) \) is of degree less than or equal to \( k \) and vanishes at points \( x_0, x_1, \ldots, x_{k-1} \), since polynomials \( P_k(x) \) and \( P_{k-1}(x) \) coincide at these points. Consequently, it exists \( \delta_k \in \mathbb{R} \), such that
\[
P_k(x) - P_{k-1}(x) = \delta_k \ (x-x_0)(x-x_1)\cdots(x-x_{k-1}) = \delta_k \ N_k(x).
\]
Then, \( \mu_k [P_{k-1}(x) + \delta_k \ N_k(x)] = \delta_k \) by relation (1.2).

As a result, the interpolating polynomial \( P_k(x) \) at points \( x_0, \ldots, x_{k-1}, x_k \) is deduced from the interpolation polynomial \( P_{k-1}(x) \) by the determination of a single additional coefficient, namely the coefficient \( \delta_k \).

\[
\frac{\delta_0 \ N_0(x) + \delta_1 \ N_1(x) + \cdots + \delta_{k-1} \ N_{k-1}(x) + \delta_k \ N_k(x)}{P_k(x)} = \frac{P_0(x)}{P_1(x)}
\]

Let us determine the first two coefficients \( \delta_0 \) and \( \delta_1 \).

<table>
<thead>
<tr>
<th>Interpolation polynomial</th>
<th>Additional interpolation constraints</th>
<th>coefficients ( \delta_k )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( P_0(x) = \delta_0 \ N_0(x) )</td>
<td>( P_0(x_0) = y_0 )</td>
<td>( \delta_0 = y_0 )</td>
</tr>
<tr>
<td>( P_1(x) = \delta_0 \ N_0(x) + \delta_1 \ N_1(x) )</td>
<td>( P_1(x_1) = y_1 )</td>
<td>( \delta_1 = \frac{y_1 - y_0}{x_1 - x_0} )</td>
</tr>
</tbody>
</table>

**Proposition 8**

Coefficient \( \delta_k \) of proposition 7 is the divided difference associated with points \( x_0, \ldots, x_k \):
\[
\delta_k = \delta[x_0, x_1, \ldots, x_k], \quad 0 \leq k \leq n.
\]

**Proof:** We proceed by induction on the degree of the interpolating polynomial.

- For the degree 0, the result is clear: the interpolating polynomial of the single data \( (x_0, y_0) \) is the constant polynomial \( q_0(x) = y_0 \ N_0(x) = \delta[x_0] \ N_0(x) \), by definition of the divided difference of order 0.
– Assume now that the property holds for any interpolating polynomial of degree $k - 1$. Let $P_{1,k}(x)$ be the interpolating polynomial at points $x_1, \ldots, x_k$ and consider the following polynomial
\[ p(x) = \frac{x - x_0}{x_k - x_0} P_{1,k}(x) - \frac{x - x_k}{x_k - x_0} P_{k-1}(x). \]

Polynomial $p(x)$ is of degree less than or equal to $k$ and $p(x_i) = y_i$ for $i = 0, \ldots, k$, so that, by uniqueness of the interpolation polynomial, we have $p(x) = P_k(x)$. Then, with relation [1.2]
\[ \delta_k = \mu_k [P_k(x)] = \mu_k [p(x)] = \frac{1}{x_k - x_0} \mu_{k-1} [P_{1,k}(x)] - \frac{1}{x_k - x_0} \mu_{k-1} [P_{k-1}(x)], \]
\[ = \frac{1}{x_k - x_0} \delta[x_1, \ldots, x_k] - \frac{1}{x_k - x_0} \delta[x_0, \ldots, x_{k-1}], \]
\[ = \delta[x_0, x_1, \ldots, x_{k-1}, x_k], \]
from the definition of divided differences, which concludes the proof.

Finally, the interpolating polynomial $P_n(x)$ of the data $(x_i, y_i)$, $0 \leq i \leq n$, is expressed in the Newton basis as follows
\[ P_n(x) = \delta[x_0] + \sum_{k=1}^{n} \delta[x_0, \ldots, x_k] \prod_{i=x_0}^{x_k} (x - x_i) \]
where divided differences $\delta_k = \delta[x_0, \ldots, x_k]$ come from the main downward diagonal of the triangular scheme given in section 5.1. Then in practice, polynomial $P_n(x)$ is evaluated with the Horner algorithm described in item 3 of section 1.3.

This Newton approach benefits of some attractive advantages. The coefficients (divided differences) in the Newton basis are easily computable according to a triangular scheme, the Newton basis allows an Horner evaluation unlike the Lagrange basis, but above all this algorithm allows to add simply new interpolation points dynamically. We give an example in next section.

### 5.4 Example

Consider again the interpolation of the polynomial function $f(x) = 4 - x - 4x^2 + x^3$ at the same 4 points $x_i$ given in section 1.3. We first evaluate (with details) the complete triangular table of divided differences for these 4 data.

<table>
<thead>
<tr>
<th>$x_k$</th>
<th>$y_k = \delta [x_k] = \hat{f}(x_k)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_0 = 1$</td>
<td>$y_0 = \delta [x_0] = 0$</td>
</tr>
<tr>
<td>$x_1 = 3$</td>
<td>$y_1 = \delta [x_1] = -8$</td>
</tr>
<tr>
<td>$x_2 = -1$</td>
<td>$y_2 = \delta [x_2] = 0$</td>
</tr>
<tr>
<td>$x_3 = 2$</td>
<td>$y_3 = \delta [x_3] = -6$</td>
</tr>
</tbody>
</table>

from which we deduce the interpolating polynomial factorized according Horner scheme as mentioned...
in section 1.3

\[ P_3(x, f) = \delta[x_0] N_0(x) + \delta[x_0, x_1] N_1(x) + \delta[x_0, x_1, x_2] N_2(x) + \delta[x_0, x_1, x_2, x_3] N_3(x) \]

\[ = 0.1 + (-4). (x - x_0) + (-1). (x - x_0)(x - x_1) + 1. (x - x_0)(x - x_1)(x - x_2) \]

\[ = 0 + (x - x_0) \left( -4 + (x - x_1) \left( -1 + (x - x_2)(1) \right) \right) \]

\[ = 0 + (x - 1) \left( -4 + (x - 3) \left( -1 + (x + 1)(1) \right) \right) \]

\[ = 4 - x - 4x^2 + x^3 \]

Now, so as to highlight the benefit of propositions 7 and 8, assume that interpolating data \((x_i, y_i)\) are provided dynamically. We thus calculate successively the interpolating polynomials \(P_1(x, f), P_2(x, f), P_3(x, f)\) as in section 4.3

- \(P_1(x, f)\): we start with data \((x_0, y_0)\) and \((x_1, y_1)\) and complete the triangular table of divided differences.

<table>
<thead>
<tr>
<th>(x_k)</th>
<th>(y_k = f(x_k))</th>
</tr>
</thead>
<tbody>
<tr>
<td>(x_0 = 1)</td>
<td>(y_0 = 0)</td>
</tr>
<tr>
<td>(x_1 = 3)</td>
<td>(y_1 = -8)</td>
</tr>
</tbody>
</table>

\[ P_1(x, f) = 0 - 4x - 1 \]

- \(P_2(x, f)\): we add the data \((x_2, y_2)\) and have just to update the third row of the triangular table of divided differences.

<table>
<thead>
<tr>
<th>(x_k)</th>
<th>(y_k = f(x_k))</th>
</tr>
</thead>
<tbody>
<tr>
<td>(x_0 = 1)</td>
<td>(y_0 = 0)</td>
</tr>
<tr>
<td>(x_1 = 3)</td>
<td>(y_1 = -8)</td>
</tr>
<tr>
<td>(x_2 = -1)</td>
<td>(y_2 = 0)</td>
</tr>
</tbody>
</table>

\[ P_2(x, f) = P_1(x, f) - (x - 1)(x - 3) \]

- \(P_3(x, f)\): we add the data \((x_3, y_3)\) and have just to update the fourth row of the triangular table of divided differences.

<table>
<thead>
<tr>
<th>(x_k)</th>
<th>(y_k = f(x_k))</th>
</tr>
</thead>
<tbody>
<tr>
<td>(x_0 = 1)</td>
<td>(y_0 = 0)</td>
</tr>
<tr>
<td>(x_1 = 3)</td>
<td>(y_1 = -8)</td>
</tr>
<tr>
<td>(x_2 = -1)</td>
<td>(y_2 = 0)</td>
</tr>
<tr>
<td>(x_3 = 2)</td>
<td>(y_3 = -6)</td>
</tr>
</tbody>
</table>

\[ P_3(x, f) = P_2(x, f) + (x - 1)(x - 3)(x + 1) \]

5.5 Implementation of divided differences scheme

The triangular scheme of divided differences can be easily implemented with a two dimensional array. We propose in the following exercise to implement this scheme for divided differences with a one dimensional vector, and allowing to add dynamically new data points.

**Exercise 4**

1) Write a Python function `diffdiv(xi, yi)` enabling the computation of divided differences with a one-dimensional vector `delta`.

**Input data:**

- \(xi = (x_0, x_1, \ldots, x_n)\) : the \(n + 1\) interpolation x-coordinates
- \(yi = (y_0, y_1, \ldots, y_n)\) : typically \(y_i = f(x_i)\)

**Output data:**
• $\delta = (\delta[x_0], \delta[x_0, x_1], \ldots, \delta[x_0, x_1, \ldots, x_n], \ldots, \delta[x_{n-1}, x_n], \delta[x_n])$: vector of size $2n+1$ containing in its first half, the divided differences of the main downward diagonal of the triangular scheme, and in its second half, the other divided differences enabling the addition of new data points $(x_{n+1}, y_{n+1})$.

2) Write a Python function `updateDD(xi, yi, delta, xnew, ynew)` enabling the update of the vector `delta` of divided differences in case of a new data $(x_{new}, y_{new})$.

Hints:
(a) duplicate the vector $y_i$ of size $n+1$ into the vector $\delta$ by inserting a zero between each consecutive element $y_i$ and $y_{i+1}$, so as to get a vector of size $2n+1$.
(b) take inspiration from the table below where each calculated divided difference is put in the slot of the $\delta$ vector associated to its line index.

<table>
<thead>
<tr>
<th>$\delta$</th>
<th>$y_0 = \delta[x_0]$</th>
<th>$y_1 = \delta[x_1]$</th>
<th>$y_2 = \delta[x_2]$</th>
<th>$y_3 = \delta[x_3]$</th>
<th>$\vdots$</th>
<th>$y_{n-1} = \delta[x_{n-1}]$</th>
<th>$y_n = \delta[x_n]$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0$</td>
<td>$\delta[x_0, x_1]$</td>
<td>$\delta[x_1, x_2]$</td>
<td>$\delta[x_2, x_3]$</td>
<td>$\delta[x_3, x_4]$</td>
<td>$\vdots$</td>
<td>$\delta[x_{n-2}, x_{n-1}, x_n]$</td>
<td>$\delta[x_{n-1}, x_n]$</td>
</tr>
</tbody>
</table>

Solution: 1)
```python
def diffdiv(xi, yi):
    """ Calculation of the vector delta of divided differences (DD)
    for interpolation data (xi, yi)
    Input:
        xi, yi = two vectors of same size n+1 (n = degree)
    Output:
        delta = vector of size 2n+1, precisely:
        delta(j) for j=0,1,...,n are the DD (first half)
        delta(j) for j=n+1,...,2n+1 are the other DD for updates
    """
    n = np.size(xi)
    delta = np.zeros(2*n-1)
    for i in range(n):
        delta[2*i] = yi[i]
    for k in range(1, n):
        for j in range(n-k):
            delta[k+2*j] = (delta[k+2*j+1] - delta[k+2*j-1]) / (xi[k+j] - xi[j])
    return delta
```

Solution: 2)
```python
def updateDD(xi, yi, delta, xnew, ynew):
    """ Update the vector delta of divided differences (DD)
    when a new interpolation datum (xnew, ynew)
    is added to the data xi, yi.
    The first half of the vector delta contains the DD
```
The second half contains the necessary data for subsequent updates

-> Return:
   - vectors of updated data xi and yi (xnew, ynew at the end)
   - updated vector delta

```
n = np.size(xi)
xi = np.append(xi,xnew)
yi = np.append(yi,ynew)
delta = np.append(delta,[0, ynew])
for j in range(1,n+1):
  k = 2*n-j
  delta[k] = (delta[k+1]-delta[k-1]) / (xi[n]-xi[n-j])
return xi, yi, delta
```

Exercise 5

3) Implement the Newton interpolation in Python with the Horner algorithm. Apply this program to the uniform interpolation (evenly spaced points) of the function $x \mapsto \cos(1-t^2) \exp(-t^2 + 3t - 2)$, $x\in[0,4]$.

Solution: 3) We first give the Newton interpolation function and then the main program.

```
def ftest(t):
    return (np.cos(1-t**2)) * np.exp(-t**2 + 3*t - 2)

def NewtonInterpol(xi,yi,a,b,nbEvalPts):
    """ Polynomial interpolation of the data (xi,yi) in the Newton basis ,
    with Horner evaluation .
    Returns a sampling of the interpolating polynomial over
    the interval [a,b] with nbEvalPts points
    Input :
        xi , yi = two vectors of same size n+1 (n = degree)
        a,b = two real numbers with a < b
        nbEvalPts = integer = number of sampling points
    Output :
        py = vector of nbEvalPts reals
          (the sampling of the interpolating polynomial over [a,b])
    """
    degree = np.size(xi) - 1
    t = np.linspace(a,b,nbEvalPts)
    delta = diffdiv(xi,yi)
    py = delta[degree] * np.ones(nbEvalPts)
    for k in range(degree-1,-1,-1):
        py = py * (t - xi[k]) + delta[k]
    return py
```

```python
import numpy as np
import matplotlib.pyplot as plt
plt.cla()

a = 0 ; b = 4 ; nb = 300
t = np.linspace(a,b,nb)
y = ftest(t)
plt.plot(t, y, 'r--')

# uniformly distributed interpolation points xi:
    degree = 8
    xi = np.linspace(a,b,degree+1)
    yi = ftest(xi)
```
CHAPTER 1. POLYNOMIAL INTERPOLATION

0.0 0.5 1.0 1.5 2.0 2.5 3.0 3.5 4.0
0.0 0.5 1.0 1.5 2.0 2.5 3.0

test function
equally spaced points
interpolating polynomial

Figure 1.5: Interpolation of the function \( f(x) = \cos(1 - t^2) \exp(-t^2 + 3t - 2) \): degree 8 (left) and degree 14 (right).

```python
plt.plot(xi, yi, 'or', ms=8)

# Newton interpolation
y = NewtonInterpol(xi, yi, a, b, nbt)
plt.plot(t, y, 'b', lw=1)
plt.legend(["'test function'", "'equally spaced points'", "'interpolating polynomial'"], loc='best')
```

6 Error bounds in Lagrange interpolation

Consider a function \( f \in C^{n+1}[a,b] \), and a set of \( n+1 \) distinct points \((x_i)\) in the interval \([a,b]\). We propose to estimate an error bounds between the function \( f \) and its interpolating polynomial associated with the data \((x_i, y_i = f(x_i))\).

Let \( x \) be a real value fixed in the interval \([a,b]\), different from each of the points \( x_i \), and consider the function

\[
\phi_x(u) = f(u) - P_n(u, f) - (f(x) - P_n(x, f)) \prod_{i=0}^{n}(u - x_i) / \prod_{i=0}^{n}(x - x_i), \quad u \in [a,b],
\]

which is clearly of class \( C^{n+1} \) on \([a,b]\). This function \( \phi_x \) vanishes at each point \( x_i \) and also at point \( x \) and thus at \( n+2 \) points in the interval \([a,b]\). Consequently, by Rolle’s theorem, its derivative vanishes at \( n+1 \) distinct points in \([a,b]\). This derivative \( \phi'_x \in C^n[a,b] \) and admits \( n+1 \) distinct points in \([a,b]\), which allows to apply again Rolle’s theorem. Finally, we deduce by recurrence that the function \( \phi^{(n+1)} \) vanishes at one point \( \xi \) in \([a,b]\) : \( \exists \xi \in [a,b], \phi^{(n+1)}(\xi) = 0. \)

The \((n+1)\)-th derivative of the polynomial \( P_n(u, f) \) is identically zero and the \((n+1)\)-th derivative of the \((n+1)\)-degree polynomial \( \prod_{i=0}^{n}(u - x_i) \) is constant and equal to \((n+1)!\), so that

\[
\phi^{(n+1)}(\xi) = f^{(n+1)}(\xi) - (f(x) - P_n(x, f)) \frac{(n+1)!}{\prod_{i=0}^{n}(x - x_i)} = 0,
\]

which leads to

\[
f(x) - P_n(x, f) = \frac{\prod_{i=0}^{n}(x - x_i)}{(n+1)!} f^{(n+1)}(\xi).
\]

This relation is also valid for \( x = x_i, 0 \leq i \leq n \), which allows to set up the following result.
CHAPTER 1. POLYNOMIAL INTERPOLATION

Proposition 9
Let \( f \in C^{n+1}[a,b] \), \( n + 1 \) distinct points \( x_0, x_1, \ldots, x_n \) in \([a,b]\) and \( P_n(x,f) \) the interpolating polynomial of \( f \) associated with these data points. Then,

\[
\forall x \in [a,b], \exists \xi_x \in ]a,b[ , \quad f(x) - P_n(x,f) = \prod_{i=0}^{i=n} \frac{(x - x_i)}{(n+1)!} f^{(n+1)}(\xi_x).
\]

Thus,

\[
\forall x \in [a,b] , \quad |f(x) - P_n(x,f)| \leq \left| \prod_{i=0}^{i=n} \frac{(x - x_i)}{(n+1)!} \right| \max_{a \leq \xi \leq b} |f^{(n+1)}(\xi)|. \tag{1.3}
\]

Finally, with \( \Pi_{n+1}(x) = \prod_{i=0}^{i=n} (x - x_i) \), we get

\[
\left| |f - P_n(., f)| \right| = \max_{x \in [a,b]} |f(x) - P_n(x,f)| \leq \frac{||\Pi_{n+1}||}{(n+1)!} ||f^{(n+1)}|| \tag{1.4}
\]

This formula shows that the error depends on the function \( f \) and on the norm \( ||\Pi_{n+1}|| \) which is related to the repartition of the interpolation points \( x_i \) in the interval \([a,b]\). If we have no information on the distribution of points \( x_i \), the best estimation is

\[
||\Pi_{n+1}|| \leq (b-a)^{n+1}.
\]

In case of evenly spaced points \( x_i \) in \([a,b]\), one can prove that

\[
||\Pi_{n+1}|| \leq \left( \frac{b-a}{e} \right)^{n+1}.
\]

The best possible distribution of the interpolation points consists in the Chebyshev points, introduced in section 7 and leads to the optimal estimation

\[
||\Pi_{n+1}|| \leq 2 \left( \frac{b-a}{4} \right)^{n+1},
\]

which represents a substantial gain. For example, for \( n = 20 \) the ratio \( 2 \left( \frac{b-a}{4} \right)^{n+1} / (b-a)^{n+1} < 6.10^{-4} \), so that interpolation at Chebyshev points significantly improves the accuracy compared with interpolation at evenly spaced points.

7 Chebyshev interpolation points

In this section, we determine the best distribution of interpolation points in the interval \([a,b]\) so as to minimize the norm \( ||\Pi_{n+1}|| \). Notice that polynomial \( \Pi_{n+1}(x) \) is of order \( n + 1 \) and is monic: the coefficient of its highest monomial is 1.

We start with an example that can highlight a strategy for selecting data points, but which is not essential for the reading of this section.

7.1 An example

Figures 1.2 and 1.5 (where interpolating points are evenly spaced) exhibit oscillation effects on the edges of the interval. We propose to experiment a more appropriate distribution of data points \( x_i \) with a stronger concentration of points on the edges of the interval, where the oscillations are the largest.

Exercise 6
Experiment this strategy by modifying the function AcquisitionPolygone(color1,color2)
CHAPTER 1. POLYNOMIAL INTERPOLATION

given at the end of chapter ?? (Introduction to Python), so as to:
1) plot the graph of the function \( f(x) = \cos(1-t^2) \exp(-t^2+3t-2) \) on the interval \([a, b] = [0, 4]\),
2) determine and plot the (uniform) interpolating polynomial of the function \( f \) at \( N \) (\( N=\text{degree}+1 \)) evenly spaced points \( x_i \).
3) acquire with the mouse a set of \( N \) points \( x_m \) in \([a, b]\),
4) determine and plot the (mouse) interpolating polynomial of the function \( f \) at points \( x_m \).
An example is given in the following figure 1.6.

Figure 1.6: “Mouse” and uniform interpolation of the function \( f(x) = \cos(1-t^2) \exp(-t^2+3t-2) \). On this example, concentrating points on the edges of the interval leads to a better interpolating polynomial.

Solution: We just give the modified mouse acquisition function, since the rest of the script is similar as in the implementation section 5.5.

```python
def MousePointsV2(f, a, b, N, color1):
    """ V2 acquisition of N points 
    data points are plotted only on the function
    Acquisition of x-coordinates with the mouse in interval \([a, b]\) 
    and plotting of the associated values on the function \( f \)
    Output: 2 vectors \( x, y \) of same size (data to be interpolated)
    \( x \) = the x-coordinates acquired with the mouse
    \( y \) = \( f(x) \) image of the x’s in the function
    """
    x = []  # x, y are an empty lists
    y = []
    k = 0
    while k < N:
        coord = plt.ginput(1, mouse_add=1, mouse_stop=3, mouse_pop=2)
        # coord is a list of tuples : coord = [(x,y)]
        if coord != []:
            xx = coord[0][0]
            # the x-coordinates must be all distinct:
            test = 1
            if np.size(x) > 0:
                for j in range(np.size(x)):
                    if xx == x[j]:
                        test = 0
            if test == 1:  # xx is different from all the other points
                yy = f(xx)
                plt.plot(xx, yy, color1, markersize=6)
                x.append(xx)
```
7.2 Statement of the problem

We first study the problem on the symmetric interval $[-1, 1]$, from which we then deduce the solution over any interval $[a, b]$ by an affine transformation. So, we consider the problem $P$ of determining a strictly increasing sequence of $n + 1$ points $x_i$:

$$-1 \leq x_0 < x_1 < \cdots < x_{n-1} < x_n \leq 1$$

such that the norm $||\Pi_{n+1}||$ of polynomial $\Pi_{n+1}(x)$ is minimal, with

$$||\Pi_{n+1}|| = \max_{x \in [-1,1]} \left| (x-x_0)(x-x_1)\cdots(x-x_n) \right|.$$

Exercise 7

1) Solve this problem by hand for $n = 0$, $n = 1$ and $n = 2$, which provides monic polynomials respectively of degree 1, 2 and 3. Remark that the solution should be symmetric due to the symmetry of the interval $[-1, 1]$.

2) For each solution $\Pi_{n+1}(x)$ found, $n \in \{0, 1, 2\}$, determine the set

$$M_n = \left\{ u \in [-1,1], \left| \Pi_{n+1}(u) \right| = ||\Pi_{n+1}|| \right\}$$

and check that $\text{card}(M_n) = n + 2$.

The general solution is based on Chebyshev polynomials.

7.3 Chebyshev polynomials

The study of Chebyshev polynomials is presented in the form of a problem. Not all questions (such as the orthogonality property) are essential for the study of our problem.

Right are the graph of functions $\cos(x)$ and $\arccos(x)$.

The Chebyshev polynomial of degree $n$ is defined by

$$T_n(x) = \cos \left( n \arccos(x) \right), \quad x \in [-1,1].$$

1. Compute $T_0(x)$ and $T_1(x)$.

2. Compute $T_{n+1}(x) + T_{n-1}(x)$ and derive the following recursive relation

$$T_{n+1}(x) = 2x T_n(x) - T_{n-1}(x), \quad n \geq 1.$$

---

$^2$Pafnuty Lvovich Chebyshev (1821–1894), Russian mathematician.
3. Compute the Chebyshev polynomials $T_n(x)$ for $n \leq 5$ and plot their graphs on the interval $[-1, 1]$.

![Graph of Chebyshev polynomials](image)

4. Deduce that $T_n(x)$ is a polynomial of degree $n$ whose $n$-th degree coefficient is $2^{n-1}$, for $n \geq 1$.

5. Show that $T_n(x)$ is even if $n$ is even and is odd if $n$ is odd.

6. Prove that $T_n(x)$ has $n$ simple roots $r_k$ in the interval $[-1, 1]$ and specify these roots as functions of $n$ and $k$. — These roots are called Chebyshev points of order $n$ relative to the interval $[-1, 1]$.

7. Check that for all $x$ in $[-1, 1]$, $|T_n(x)| \leq 1$, and show that $|T_n(x)| = 1$ for $n + 1$ distinct values of $x$ in $[-1, 1]$. Specify these values.

8. Demonstrate that the Chebyshev polynomials are orthogonal relative to the following scalar product of weight $\frac{1}{\sqrt{1-x^2}}$. Specifically:

$$\int_{-1}^{1} T_n(x) T_p(x) \frac{1}{\sqrt{1-x^2}} \, dx = \begin{cases} 0 & \text{if } n \neq p \\ \pi & \text{if } n = p = 0 \\ \pi/2 & \text{if } n = p \neq 0 \end{cases}$$

Use the change of variable $x = \cos(u)$ and we recall that $\arccos(x) = \frac{-1}{\sqrt{1-x^2}}$.

### 7.4 Optimal distribution of points

From, the previous study of Chebyshev polynomials we have

$$T_n(x) = 2^{n-1} \prod_{k=0}^{n-1} (x - r_k) \quad \text{with} \quad r_k = \cos \left( \frac{(2k + 1)\pi}{2n} \right), \quad 0 \leq k \leq n - 1,$$

and

$$|T_n(x)| = 1 \quad \iff \quad x = m_k = \cos \left( \frac{k\pi}{n} \right), \quad 0 \leq k \leq n,$$

with

$$T_n(m_k) = \cos(k\pi) = (-1)^k.$$

As already mentioned in the previous section, the roots $r_k$ are called the Chebyshev points of order $n$ relative to the interval $[-1, 1]$.

**Proposition 10**

Let $h(x)$ be an $n$-degree polynomial with leading coefficient (coefficient of $x^n$) equal to $2^{n-1}$.
Prove that if \( h(x) \neq T_n(x) \) then

\[
\max_{x \in [-1,1]} |h(x)| > \max_{x \in [-1,1]} |T_n(x)| = 1.
\]

**Proof:** We proceed by contradiction. Assume that \( \max_{x \in [-1,1]} |h(x)| \leq 1 \), and consider the polynomial \( \gamma(x) = h(x) - T_n(x) \) which is of degree \( n - 1 \) as the highest terms of \( h(x) \) and \( T_n(x) \) are identical. We prove that \( \gamma(x) \) admits \( n \) roots in \([-1,1]\), precisely one root in each interval \([m_{k+1}, mk]\), from which we will deduce the result.

If \( k \) is an even integer, we have:

\[
\gamma(m_{k+1}) = h(m_{k+1}) - T_n(m_{k+1}) = h(m_{k+1}) - (-1)^{k+1} = h(m_{k+1}) + 1 \geq 0,
\]

by our assumption.

\[
\gamma(m_k) = h(m_k) - T_n(m_k) = h(m_k) - (-1)^k = h(m_{k+1}) - 1 \leq 0.
\]

The result is symmetric if \( k \) is an odd integer. Thus, \( \gamma(x) \) admits (at least) one root \( \xi_k \) on each close interval \([m_{k+1}, mk]\), \( 0 \leq k \leq n - 1 \).

Assume now that this root \( \xi_k \) is at an edge of the interval \([m_{k+1}, mk]\), let say \( m_k \), (with \( k \neq 0 \) and \( k \neq n \)). Then, \( h(m_k) = T_n(m_k) \) and \( m_k \) is an extremum of the polynomial \( h(x) \) and of course of \( T_n(x) \), so that \( \gamma'(m_k) = h'(m_k) - T'_n(m_k) = 0 \). Consequently, the polynomial \( \gamma(x) \) admits a double root in \( \xi_k = m_k \). The end of the proof consists in a technical counting (with multicity) of the distinct roots.

- As a result, the solution to problem \( P \) on the interval \([-1,1]\) is the sequence of Chebyshev points of order \( n + 1 \) relative to the interval \([-1,1]\) and defined by

\[
r_k = \cos \left( (2k + 1) \frac{\pi}{2n + 2} \right), \quad 0 \leq k \leq n,
\]

so that

\[
\Pi_{n+1}(x) = \frac{1}{2^n} T_{n+1}(x) = \prod_{k=0}^{n} (x - r_k)
\]

and \( ||\Pi_{n+1}|| = \frac{1}{2^n} \).

- The solution over a general interval \([a,b]\) is then deduced through the affine transformation

\[
x \in [-1,1] \mapsto u = \varphi(x) \in [a,b] \quad \text{with} \quad \varphi(x) = \frac{a + b}{2} + \frac{b - a}{2} x.
\]

Precisely, given a set of \( n + 1 \) distinct values \( u_i \) in the interval \([a,b]\), associated with \( n + 1 \) distinct values \( x_i = \varphi^{-1}(u_i) \) in \([-1,1]\), we have with the variable \( u \in [a,b] \) associated with the variable \( x = \varphi^{-1}(u) \) in \([-1,1]\):

\[
\prod_{i=0}^{n} (u - u_i) = \prod_{i=0}^{n} (\varphi(x) - \varphi(x_i)) = \prod_{i=0}^{n} \left( \frac{b - a}{2} (x - x_i) \right) = \left( \frac{b - a}{2} \right)^{n+1} \prod_{i=0}^{n} (x - x_i).
\]

Consequently, the best distribution of points on the interval \([a,b]\) is the sequence of *Chebyshev points of order* \( n + 1 \) *over the interval* \([a,b]\):

\[
\hat{r}_k = \frac{a + b}{2} + \frac{b - a}{2} \cos \left( (2k + 1) \frac{\pi}{2n + 2} \right), \quad 0 \leq k \leq n,
\]

and we have the norm on \([a,b]\)

\[
||\prod_{k=0}^{n} (x - \hat{r}_k)|| = \left( \frac{b - a}{2} \right)^{n+1} \frac{1}{2^n} = 2 \left( \frac{b - a}{4} \right)^{n+1}.
\]

**Exercise 8**

Consider the example developed in exercise \([\) and compare the uniform and Chebyshev interpo-
lations on the interval [0, 4] with different degrees.

Solution: With our previous Python scripts, we just need to give the calculation of Chebyshev points.

```python
a = 0; b = 4
degree = 8
# Chebyshev interpolation points rk
k = np.arange(0, degree + 1)
kk = (2*k+1)*np.pi / (2*degree + 2)
rk = (a+b)/2 + ((b-a)/2) * np.cos(kk)
yk = ftest(rk)
plt.plot(rk, yk, 'og', ms=6)
```

Figure 1.7: Uniform and Chebyshev interpolations of the function \( f(x) = \cos(1-t^2) \exp(-t^2+3t-2) \).

7.5 Alternating property

The optimal solution \( \Pi_{n+1}(x) = \frac{1}{2^n} T_{n+1}(x) = (x - r_0)/(x - r_1) \cdots (x - r_n) \) on the interval \([-1, 1]\) is characterized by the fact that the norm \( \| \Pi_{n+1} \| \) is reached exactly \( n+2 \) times on this interval (once between each point \( r_k \) and \( r_{k+1} \) and once at each edge of the interval). This is called the alternating or equioscillation property.

Exercise 9

Plot the graph of the Chebyshev polynomial \( T_{n+1}(x) \) (for example with \( n = 5 \)) on the interval \([-1, 1]\). Then, mildly disrupt the roots \( r_k \) and plot the graph of the associated disrupted polynomial

\[
\tilde{T}_{n+1}(x) = 2^n \prod_{i=0}^{n} (x - \tilde{r}_k)
\]

where \( \tilde{r}_k = r_k + \epsilon_k \) are the disrupted roots.

8 Convergence

Consider a function \( f \in C[a,b] \) and a sequence of \( n+1 \) distinct points \( x_{n,i} \)

\[ a \leq x_{n,0} < x_{n,1} < \cdots < x_{n,n} \leq b \]
for each \( n \in \mathbb{N}^* \). Denoting by \( P_n(., f) \) the interpolating polynomial of \( f \) at the \( n + 1 \) points \( x_{n,i}, 0 \leq i \leq n \), we consider the question of the convergence of the sequence of these interpolating polynomials \( P_n(., f) \) to \( f \) when \( n \) tends to \(+\infty\). Convergence may be understood pointwise or uniform.

We first consider a well-known example which shows that uniform interpolation is not recommended and that we definitely need to prefer the Chebyshev interpolation.

### 8.1 Runge’s phenomenon

Runge’s phenomenon\(^3\) is a problem of oscillations that occurs (for certain functions) with uniform interpolation when the degree increases.

Carl Runge noticed this behaviour of polynomial interpolation in 1901 when studying the approximation of certain functions by polynomials and in particular rational functions which are the simplest functions after polynomials. This discovery was surprising at this time because due to the Weierstrass theorem (see chapter ??) it was thought that going to higher degrees would improve accuracy.

Precisely, the classical example of this Runge phenomenon is provided by the rational function \( f_R(x) = \frac{1}{1 + 25x^2} \) defined on the interval \([-1, 1]\), that we interpolate at uniformly distributed data points in the intervall \([-1, 1]\). Figure 1.9 provides examples of such interpolating polynomials.

An analysis of this phenomenon can be found in the literature, e.g., Demailly 2006 (reference given at the end of this chapter). We give an overview of the main result concerning the classical example provided above. Considering more generally the function \( f_\alpha(x) = \frac{1}{\alpha^2 + x^2}, x \in [-1,1] \) where \( \alpha > 0 \) is a parameter, and denoting by \( P_n(x, f_\alpha) \) its uniform interpolating polynomial of degree \( n \), we have the following result.

\[
\alpha > \alpha_0 \simeq 0.526 : \text{the sequence } P_n(x, f_\alpha) \text{ converges uniformly to } f_\alpha \text{ on } [-1,1].
\]

\[
\alpha < \alpha_0 : \text{the sequence } P_n(x, f_\alpha) \text{ converges uniformly to } f_\alpha \text{ on any close interval contained in the open interval } ] - \bar{x}, \bar{x}[ \text{ and diverges in } [-1, -\bar{x}[ \cup ]\bar{x}, 1], \text{ where } \bar{x} \text{ is solution of the following equation}
\]

\[
(1 + x)^{\frac{1+x}{\alpha}} (1 - x)^{\frac{1-x}{\alpha}} = \sqrt{1 + \alpha^2} \exp\left(\alpha \arctan\left(\frac{1}{\alpha}\right)\right)
\]

The classical function \( f_R \) studied by C. Runge corresponds to parameter \( \alpha = 1/5 < \alpha_0 \), which allows to compute the value \( \bar{x} \). The interval \(( -\bar{x}, \bar{x} )\) has been plotted in bold green on the figure 1.9.

**Exercise 10**

Determine an approximation of the value \( \bar{x} \) with algorithms developed in chapter ??

**Solution** : \( \bar{x} \simeq 0.7266768 \)

\(^3\) Carl David Tolmé Runge, 1856-1927, German mathematician
CHAPTER 1. POLYNOMIAL INTERPOLATION

8.2 Some convergence results

From inequality (1.4) given in section 6 we have the immediate following result.

Proposition 11

Let \( f \in \mathcal{C}^\infty[a,b] \), such that

\[
\exists M \in \mathbb{R}^+, \quad \forall k \in \mathbb{N}, \quad \left\| f^{(k)} \right\| \leq M.
\]

Then, for any sequence of interpolation points \((x_{n,i})\), the sequence of interpolating polynomials \(P_n(.,f)\) converges uniformly to \(f\) on the interval \([a,b]\) when \(n\) tends to \(+\infty\).

The following result provides an interesting insight into the interpolation and convergence process, but it has not real practical interest. Indeed, one can easily understand that such an “algorithm” is rather hard to implemented numerically.

Proposition 12

For any function \( f \in \mathcal{C}[a,b] \) there exists a sequence of points \((x_{n,i})\), \( n \in \mathbb{N}_* \), \( 0 \leq i \leq n \) for which the sequence of interpolating polynomials \(P_n(x,f)\) converges uniformly to \(f\) on \([a,b]\).

Proof: The proof is a direct consequence of the Weierstrass approximation theorem which states that there exists a sequence of polynomials \(p_n(x)\) of best approximation (see chapter ??) which converges uniformly to \(f\). Then, due to the equioscillation property, the graph of each polynomial of best approximation \(p_n(x)\) intersects the graph of \(f\) for \(n + 1\) distinct values \(x_{n,i}\). Then, by uniqueness, the interpolating polynomial \(P_n(x,f)\) at these values coincides with \(p_n(x)\), which gives the result.

Analytic functions. Some results require technical developments. As an example, for an analytic function \(f\) defined by a power series centered at point \(\frac{a+b}{2}\), with radius of convergence \(R > \frac{3}{2}(b-a)\),

Figure 1.9: Polynomial interpolation of the function \(f(x) = \frac{1}{1 + 25x^2}\) with evenly spaced data points, over the interval \([-1, 1]\). The green bold bar represents the area where the sequence of interpolating polynomials “should converge”.

Figure 1.9:

- degree 8
- degree 10
- degree 20
- degree 30
- degree 40
- degree 60
the sequence of interpolating polynomials converges uniformly to $f$ for any sequence of points $(x_{n,i})$. For Chebyshev interpolating polynomials, we just need $R > \frac{3}{4}(b-a)$ to insure the uniform convergence.

![Cosine function uniform interpolant](image)

**Figure 1.10**: Polynomial interpolation of the function cosine on the interval $[-2\pi, 3\pi]$, but the evenly spaced data points are only sampled in the sub-interval $[0, \pi]$.

### 8.3 Lebesgue constant

Given a sequence of $n+1$ distinct points $x_i$ in the interval $[a, b]$ we consider the Lagrange interpolation operator

$$L_n : f \in C[a,b] \mapsto P_n(x,f) \in \mathbb{R}_n[x],$$

where $P_n(x,f)$ is the interpolating polynomial of $f$ at points $x_i$. With Lagrange basis polynomials we have

$$P_n(x,f) = \sum_{i=0}^{n} f(x_i) L_i(x)$$

so that

$$|P_n(x,f)| \leq \left( \max_{x \in [a,b]} |f(x)| \right) \sum_{i=0}^{n} |L_i(x)|, \quad \forall x \in [a,b]$$

and

$$||L_n(f)|| \leq \left( \max_{x \in [a,b]} \sum_{i=0}^{n} |L_i(x)| \right) \||f||.$$

**Definition 1**

The Lebesgue constant associated with the data points $x_0, x_1, \ldots, x_n$ is defined by

$$\Lambda_n = \max_{x \in [a,b]} \left( \sum_{i=0}^{n} |L_i(x)| \right).$$
We thus have $\|L_n(f)\| \leq \Lambda_n \|f\|$ and one can prove that $\Lambda_n$ is the norm of the Lagrange interpolation operator $L_n$, i.e., $\Lambda_n = \|L_n\|$. The Lebesgue constant can be viewed as an “amplification” factor of the error in the interpolation process.

This constant is also related to the convergence of interpolating polynomials and leads to the following result that we admit.

**Proposition 13**

Assume that the function $f$ is $K$-Lipschitz, which means that for all $x, y \in [a, b]$, we have $|f(x) - f(y)| \leq K |x - y|$. Then, the sequence of Chebyshev interpolating polynomials converges uniformly to $f$ on $[a, b]$.

Since $||x| - |y|| \leq |x - y|$ for all $x, y \in \mathbb{R}$, we deduce that the absolute function is 1-Lipschitz, so that the previous proposition [13] applies. See figure 1.11 and notice the behaviour of the sequence of interpolating polynomials at the non differentiable point.

![Figure 1.11: Uniform and Chebyshev interpolation of the absolute function which is only $C^0$ at the origin. Clearly, Chebyshev interpolation has a better behaviour and avoids oscillations on the edges of the interval. But the zoom on the Chebyshev interpolants of degrees 20, 30 and 40 shows the difficulty for a $C^\infty$ polynomial to fit with the absolute value at its non differentiable value.](image-url)
9 Overview on parametric interpolation

Given a sequence of points \( M_i = (x_i, y_i), 0 \leq i \leq n \), we look for a polynomial parametric curve of degree \( n \) which interpolates these data points, that is which goes through each point \((x_i, y_i)\) for some parameter \( t_i \). The main question consists in the choice of the interpolation parameters \( t_i \), which are also called the interpolation nodes or simply the nodes.

Essentially, we consider three choices for the interpolation parameters. The first one is the uniform parametrization: parameters \( t_i \) are evenly spaced in the parameters domain. The second one is the Chebyshev parametrization as introduced in section \( \Box \). The last one is the chordal parametrization: parameters are chosen in such a way that distances between successive parameters \( t_i \) are proportional to the distances between associate successive data points \( M_i \). The chordal parametrization is more reliable and faithful with respect to the geometry of the initial data.

Precisely, we look for a \( n \)-degree polynomial parametric curve

\[
m : [a, b] \in \mathbb{R} \longrightarrow \mathbb{R}^2 \quad t \longmapsto m(t) = \begin{pmatrix} m_x(t) \\ m_y(t) \end{pmatrix}
\]

such that

\[
m(t_i) = M_i \iff \begin{cases} m_x(t_i) = x_i \\ m_y(t_i) = y_i \end{cases}, \quad 0 \leq i \leq n
\]

with a sequence of interpolation nodes \( a \leq t_0 < t_1 < \cdots < t_n \leq b \), and where \( m_x(t) \) and \( m_y(t) \) are polynomials of degree \( n \). As a result, we are reduced to solving two separate polynomial interpolation problems.

Considering the parameters domain \([a, b] = [0, 1]\) (that does not affect the method), the choices for the interpolation nodes \( t_i \) are the following.

- Uniform parameterization: \( t_{i+1} - t_i = 1/n \).
- Chebyshev parameterization: \( \hat{t}_i = a + \frac{b-a}{2} \cos \left( (2i + 1) \frac{\pi}{2n+2} \right) \).
- Chordal parameterization: \( t_{i+1} - t_i = \frac{d_i}{\sum_{k=0}^{n-1} d_k} \) with \( d_k = \text{dist}(M_k, M_{k+1}) \).

Example: parametric interpolation of a smooth curve

![Example](image)

Example: parametric interpolation of mouse data

We consider mouse acquired data (precisely polygons) and we compare the parametric interpolation with the uniform, Chebyshev and chordal parameterizations through different examples. See figure \( \Box \). Notice that using the Chebyshev parameterization, the first and last Chebyshev parameters do not coincide with extremities of the parameters domain \((a \text{ and } b)\).
Figure 1.12: We consider the parametric interpolation of the Lissajous curve \( t \mapsto (x = \sin(2t), y = \sin(3t)), \ t \in [0, 2\pi] \). We first choose \( n + 1 \) points on the curve through a uniform sampling in the parameter domain. We then interpolate this sampling by polynomial parametric curves of different degrees with the same (uniform) parameters. As the sine function is analytic, the convergence is insured.

Finally, the increase of the number of data points increases the degree of the interpolating polynomials and leads to oscillations and numerical errors. We should prefer approximation or spline interpolation as introduced in the following chapters.

10 An example with Scipy

We use here the method \texttt{scipy.interpolate.lagrange} from the module \texttt{scipy.interpolate}. However, I am not sure I would recommend this method as it works with the monomial basis. As already mentioned, spline interpolation is more appropriate in most situations.
11 Exercises

Exercise 11

Consider the 7 distinct points :

\[x_0 = 0, \ x_1 = 1, \ x_2 = 2, \ x_3 = 3, \ x_4 = 4, \ x_5 = 5, \ x_6 = 6,\]

and a sufficiently smooth function \( f \) which satisfies

\[
\begin{align*}
  f(x_0) &= 0, \\
  f(x_1) &= \sum_{i=0}^{1} i^2 = 0^2 + 1^2, \\
  f(x_2) &= \sum_{i=0}^{2} i^2 = 0^2 + 1^2 + 2^2, \\
  f(x_3) &= \sum_{i=0}^{3} i^2 = 0^2 + 1^2 + 2^2 + 3^2, \\
  f(x_4) &= \sum_{i=0}^{4} i^2 = 0^2 + 1^2 + 2^2 + 3^2 + 4^2, \\
  f(x_5) &= \sum_{i=0}^{5} i^2 = 0^2 + 1^2 + 2^2 + 3^2 + 4^2 + 5^2, \\
  f(x_6) &= \sum_{i=0}^{6} i^2 = 0^2 + 1^2 + 2^2 + 3^2 + 4^2 + 5^2 + 6^2.
\end{align*}
\]

1. Determine the divided differences

\[f[x_0], f[x_0, x_1], f[x_0, x_1, x_2], \ldots, f[x_0, x_1, x_2, x_3, x_4, x_5, x_6].\]

2. Deduce the interpolating polynomial \( p_6(x) \) of the function \( f \) at points \( x_i \).
3. Prove by induction that for all \( n \in \mathbb{N} \),
\[
p_6(n) = \sum_{i=0}^{n} i^2.
\]

4. Is the proof by induction necessary if we know that \( \sum_{i=0}^{n} i^2 \) is a polynomial function of degree 3 of the variable \( n \).

Exercise 12

**Neville-Aitken algorithm.**

Let \( f \in C[a, b] \). Consider a set of \( n + 1 \) distinct points
\[
a = x_0 < x_1 < x_2 < \cdots < x_{n-1} < x_n = b,
\]
and \( P_{i,k}(x, f) \) the interpolating polynomial of \( f \) at points \( x_i, x_{i+1}, \ldots, x_{i+k} \).

1. Specify polynomials
\[
P_{i,0}(x, f), \quad i = 0, 1, \ldots, n.
\]

2. Prove that
\[
P_{i,k+1}(x, f) = \frac{x-x_i}{x_{i+k+1}-x_i} P_{i+1,k}(x, f) + \frac{x_{i+k+1}-x}{x_{i+k+1}-x_i} P_{i,k}(x, f) \quad \text{for} \quad 0 \leq k \leq n-1 \quad 0 \leq i \leq n-k-1
\]

3. Specify the previous formula for \( k = 0 \) and \( k = 1 \).

Then use this approach to construct the interpolating polynomial associated with data
\[
x_0 = -1, \quad x_1 = 0, \quad x_2 = 1,
\]
for a function \( f \) satisfying
\[
f(x_0) = 1, \quad f(x_1) = 0, \quad f(x_2) = 2.
\]

Exercise 13

*Error* — Consider the function \( f(x) = 2^x \) and \( p_n(x) \) the interpolating polynomial of \( f \) at \( n + 1 \) data points uniformly distributed on \([0, 1]\). Find an upper bound for the error
\[
|p_n(x) - 2^x|, \quad x \in [0, 1].
\]

Exercise 14

*Error* — With which precision can we compute \( \sqrt{115} \) using the polynomial interpolation based on the points \( x_0 = 100, \quad x_1 = 121, \quad x_2 = 144 \).

Exercise 15

*Error* — Consider the function \( f(x) = 2 \sin(x) - 3 \cos(x) \) defined on the interval \([-\pi, 3\pi]\) and \( P_n(x, f) \) its interpolating polynomial at \( n + 1 \) distinct points in \([-\pi, 3\pi]\).

Determine \( n \) such that \( ||f - P_n(. , f)|| < 10^{-5} \).

References:
Jean-Pierre Demailly, Analyse numérique et équations différentielles, Presses universitaires de Grenoble, 2006
And several Internet examples