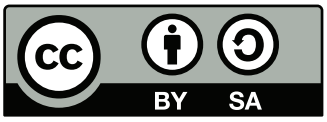
Universidad Complutense de Madrid

Laboratorio de Computación Científica
Laboratory for Scientific Computing
Interpolación y ajuste de funciones ※ Interpolation and function fitting

Juan Jiménez
Héctor García de Marina
Lía García

2 de septiembre de 2024



Índice general

Índice de figuras

Índice de cuadros

Capítulo/Chapter 1

Interpolación y ajuste de funciones

> Digo que ya tú sabes que la humildad es la basa y fundamento de todas las virtudes, y que sin ella no hay alguna que lo sea. Ella allana inconvenientes, vence dificultades, y es un medio que siempre a gloriosos fines nos conduce; de los enemigos hace amigos, templa la cólera de los airados y menoscaba la arrogancia de los soberbios; es madre de la modestia y hermana de la templanza; en fin, con ella no pueden atravesar triunfo que les sea de provecho los vicios, porque en su blandura y mansedumbre se embotan y despuntan las flechas de los pecados.
>
> El coloquio de los perros. Miguel de Cervantes

En este capítulo vamos a estudiar distintos métodos de aproximación polinómica. En términos generales el problema consiste en sustituir una función $f(x)$ por un polinomio,

In this chapter, we will explore different methods of polynomial approximation. We can define the problem in general terms as finding a polynomial to represent a function $f(x)$.

$$f(x) \approx p(x) = a_0 + a_1 \cdot x + a_2 \cdot x^2 + a_3 \cdot x^3 + \cdots + a_n \cdot x^n$$

Para obtener la aproximación podemos partir de la ecuación que define $f(x)$, por ejemplo la función error,

To obtain the approximation, we can start from the equation that defines $f(x)$, for instance, the error function,

$$erf(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$$

O bien, puede suceder que solo conozcamos algunos valores de la función, por ejemplo a

Nevertheless, it is possible that we only have some values of the function, such as when

través de una tabla de datos,

we only have a data table.

Tabla 1.1: $f(x) = erf(x)$

x	$f(x)$
0.0	0.0000
0.1	0.1125
0.2	0.2227
0.3	0.3286
0.4	0.4284
0.5	0.5205

La aproximación de una función por un polinomio, tiene ventajas e inconvenientes.

Probablemente la principal ventaja, desde el punto de vista del cómputo, es que un polinomio es fácil de evaluar mediante un ordenador ya que solo involucra operaciones aritméticas sencillas. Además, los polinomios son fáciles de derivar e integrar, dando lugar a otros polinomios.

En cuanto a los inconvenientes hay que citar el crecimiento hacia infinito o menos infinito de cualquier polinomio para valores de la variable independiente alejados del origen. Esto puede dar lugar en algunos casos a errores de redondeo difíciles de manejar, haciendo muy difícil la aproximación para funciones no crecientes.

Vamos a estudiar tres métodos distintos; en primer lugar veremos la aproximación mediante el polinomio de Taylor, útil para aproximar una función en las inmediaciones de un punto. A continuación, veremos la interpolación polinómica y, por último, estudiaremos el ajuste polinómico por mínimos cuadrados.

El uso de uno u otro de estos métodos esta asociado a la información disponible sobre la función que se desea aproximar y al uso que se pretenda hacer de la aproximación realizada.

To approximate a function using a polynomial has pros and cons.

Perhaps the main advantage, from a computing point of view, is that a polynomial is easy to evaluate using a computer as it only involves simple arithmetical operations. Besides, Polynomial integration and derivation are operations that are easy to carry out, yielding other polynomials.

Among their drawbacks we have to remark the polynomial growing towards infinity or minus infinity as the independent variable goes away zero. This can give rise in some cases to rounding errors that are difficult to handle, making the approximation for non-increasing functions very difficult.

We will study three different methods: first we will see the approximation using the Taylor polynomial, very useful to approximate a function close to a point for which we know the function value. Later, we will discus polynomial interpolation and finally we present mean square polynomial fitting.

We may relate the use of one or another method to the available information on the function we want to approximate and to the objectives we want to reach with the approximation.

1.1. El polinomio de Taylor.

Supongamos una función infinitamente derivable en un entorno de un punto x_0. Su expansión en serie de Taylor se define como,

1.1. The Taylor's polynomial.

Suppose we have an function infinitely derivable around a point x_0. We define the function taylor series expansion as,

$$f(x) = f(x_0)+f'(x_0)\cdot(x-x_0)+\frac{1}{2}f''(x_0)\cdot(x-x_0)^2+\cdots+\frac{1}{n!}f^{(n)}(x_0)\cdot(x-x_0)^n+\frac{1}{(n+1)!}f^{(n+1)}(z)\cdot(x-x_0)^{n+1}$$

Donde z es un punto sin determinar situado entre x y x_0. Si eliminamos el último término, la función puede aproximarse por un polinomio de grado n

Where z is an indeterminate point located between x and x_0. If we eliminate the last term the function would be approximated by a degree n polynomial.

$$f(x) \approx f(x_0) + f'(x_0) \cdot (x - x_0) + \frac{1}{2} f''(x_0) \cdot (x - x_0)^2 + \cdots + \frac{1}{n!} f^{(n)}(x_0) \cdot (x - x_0)^n$$

El error cometido al aproximar una función por un polinomio de Taylor de grado n, viene dado por el término,

The error we get when approximating a function by a degree n Taylor polynomial can be obtained as,

$$e(x) = |f(x) - p(x)| = \left| \frac{1}{(n+1)!} f^{(n+1)}(z) \cdot (x - x_0)^{n+1} \right|$$

Es fácil deducir de la ecuación que el error disminuye con el grado del polinomio empleado y aumenta con la distancia entre x y x_0. Además cuanto más suave es la función (derivadas pequeñas) mejor es la aproximación.

Por ejemplo para la función exponencial, el polinomio de Taylor de orden n desarrollado en torno al punto $x_0 = 0$ es,

Looking at the error function, it is easy to realize that decreases when the polynomial degree increases and increases when the distance between x and x_0 increases.

For instance, the Taylor polynomial of degree n for the exponential functions, around the point $x_0 = 0$ is,

$$e^x \approx 1 + x + \frac{1}{2} x^2 + \cdots + \frac{1}{n!} x^n = \sum_{i=0}^{n} \frac{1}{i!} x^i$$

y el del logaritmo natural, desarrollado en torno al punto $x_0 = 1$,

and the Taylor polynomial for the logarithm, around the point $x- == 1$ is,

$$\log(x) \approx (x-1) - \frac{1}{2}(x-1)^2 + \cdots + \frac{(-1)^{n+1}}{n}(x-1)^n = \sum_{i=1}^{n} \frac{(-1)^{i+1}}{i}(x-1)^i$$

La existencia de un término general para los desarrollos de Taylor de muchas funciones elementales lo hace particularmente atractivo para aproximar funciones mediante un ordenador. Así por ejemplo, la siguiente función escrita en Python, aproxima el valor del logaritmo natural en un punto, empleando un polinomio de Taylor del grado que se desee,

The existence of a general term for Taylor's expansion of many essential functions makes it highly interesting to approximate functions using a computer. For example, the following function, written in Python, approximates the natural logarithm using Taylor's polynomial of any degree we want.

series_de_taylor.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Fri Jul 12 15:02:07 2024


@author: juan
"""
import numpy as np
```

```
10  from matplotlib import pyplot as pl
11  def taylorln(x,n):
12      """
13      Esta función aproxima el valor del logaritmo natural de un numero
14      empleando para ello un polinomio de Taylor de grado n desarrollado en
15      torno a x=1. Las variables de entrada son: x, valor para el que se desea
16      calcular el logaritmo. n Grado del polinomio que se empleará en el
17       cálculo. La variable de salida y es el logaritmo de x.
18      This fuction computes (approx) the value of the natural logarithm for a number
19      using a degree n Taylor's polynomial, at x0=1. input variables are: x, value to
20      calculate its logarithm. n degree of the polynomial useto aporaches the log.
21      the funtion returns the locarithm computed.
22
23      Parameters
24      ----------
25      x : Real
26
27      n : int
28          Taylor's polinomial Degree
29
30      Returns
31      -------
32      y : real
33          log(x)
34
35      """
36      y = 0
37      for i in range(1,n+1):
38          y = y+(-1)**(i+1)*(x-1)**i/i
39
40      return(y)
41
42
43  def taylorcum(fun,n,x):
44      """
45
46
47      Parameters
48      ----------
49      fun : takes the taylor series of degree n, described in input function fun
50            and calculates and draws the result for an array of points, x
51          DESCRIPTION.
52      n : TYPE int
53          DESCRIPTION.
54          Taylor's polynomial degree'
55      x : TYPE array of real numbers
56          DESCRIPTION. point to calculate the Taylor series
57
58      Returns
59      -------
60      y : Type real
61          DESCRIPTION: array of values computed
```

```
62
63      """
64
65      y =np.array([fun(i,n) for i in x])
66      pl.plot(x,y)
```

La aproximación funciona razonablemente bien para puntos comprendidos en el intervalo $0 < x < 2$. La figura 8.1 muestra los resultados obtenidos en dicho intervalo para polinomios de Taylor del logaritmo natural de grados 2, 5, 10 20. La linea continua azul representa el valor del logaritmo obtenido con la función de Numpy `log`.

The approximation works reasonably well for points inside the interval $0 < x < 2$. Figure 8.1 shows the results achieved inside such interval, using Taylor's polynomial with degrees 2, 5, 10 and 20. The blue line represent the logarithm values we get using the numpy function `log`.

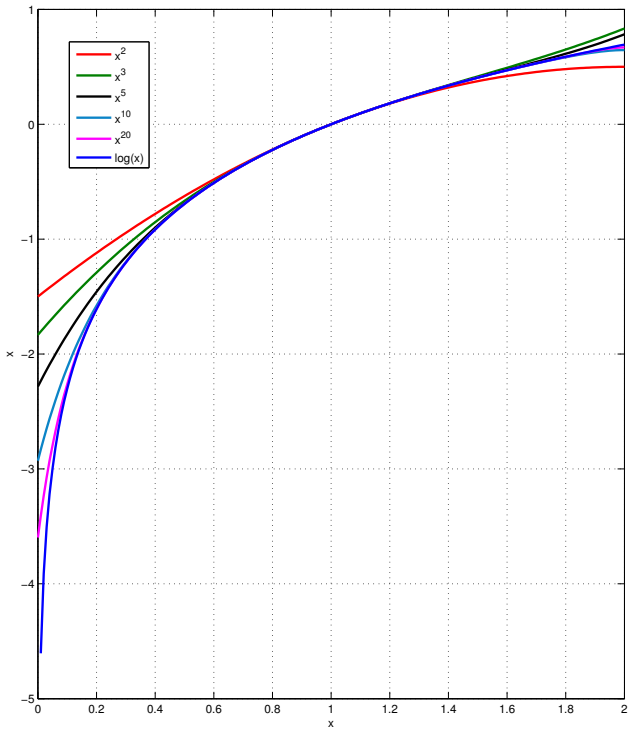


Figura 1.1: Comparación entre resultados obtenidos para polinomios de Taylor del logaritmo natural. (grados 2, 3, 5, 10, 20)
Figure 1.1: A comparison among the results achieved using Taylor polynomials to approach the logarithm. (2, 3, 5, 10, 20 degrees)

Las funciones $\sin(x)$ y $\cos(x)$, son también simples de aproximar mediante polinomios de

Functions $\sin(x)$ and $\cos(x)$, are also easy to approach using Taylor's polynomials. If we

Taylor. Si desarrollamos en torno a $x_0 = 0$, la serie del coseno solo tendrá potencias pares mientras que la del seno solo tendrá potencias impares,

expand around $x_0 = 0$, the cosine series will have only even powers and the sine series will only have odd powers,

$$\cos(x) \approx \sum_{i=0}^{n} \frac{(-1)^i}{(2i)!} x^{2i}$$
$$\sin(x) \approx \sum_{i=0}^{n} \frac{(-1)^i}{(2i+1)!} x^{2i+1}$$

En las figuras 8.2(a) y 8.2(b) Se muestran las aproximaciones mediante polinomios de Taylor de las funciones coseno y seno. Para el coseno se han empleado polinomios hasta grado 8 y para el seno hasta grado 9. En ambos casos se dan los resultados correspondientes a un periodo $(-\pi, \pi)$. Si se comparan los resultados con las funciones `cos` y `sin`, suministradas por Numpy, puede observarse que la aproximación es bastante buena para los polinomios de mayor grado empleados en cada caso.

Figures 8.2(a) and 8.2(b) show aproximations to sine and consine functions using Taylor's polynomials. We have use polynomias up to 8 degree for the cosine function and up to degree 9 for the sine function. In both cases we have calculated the results inside the interval $(-\pi, \pi)$. When we compare these results with those yielded by Numpy functions `cos` and `sin`, we see that the approximation is quite good for the higher degree polynomials we have used in each case.

1.2. Interpolación polinómica.

Se entiende por interpolación el proceso por el cual, dado un conjunto de pares de puntos $(x_0, y_0), (x_1, y_1), \cdots (x_n, y_n)$ se obtiene una función $f(x)$, tal que, $y_i = f(x_i)$, para cada par de puntos (x_i, y_i) del conjunto. Si, en particular, la función empleada es un polinomio $f(x) \equiv p(x)$, entonces se trata de interpolación polinómica.

Teorema de unicidad. Dado un conjunto $(x_0, y_0), (x_1, y_1), \cdots (x_n, y_n)$ de $n+1$ pares de puntos, tales que todos los valores x_i de dicho conjuntos son diferentes entre sí, solo existe un polinomio $p(x)$ de grado n, tal que $y_i = p(x_i)$ para todos los pares de puntos del conjunto.

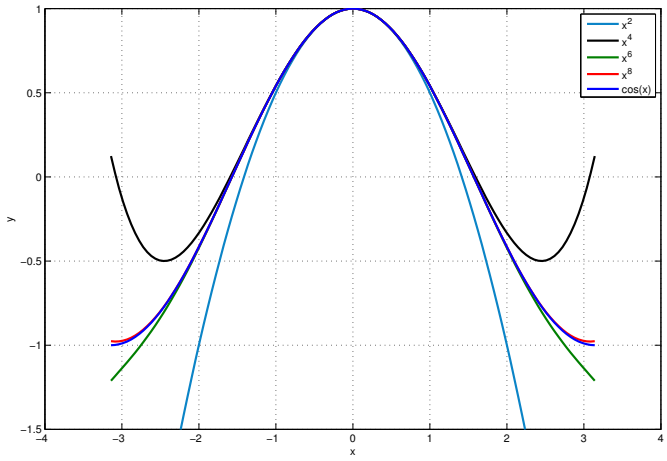
Si tratamos de interpolar los puntos con un polinomio de grado menor que n, es posible que no encontremos ninguno que pase por todos los puntos. Si, por el contrario empleamos un polinomio de grado mayor que n, nos encontramos con que no es único. Por último

1.2. Polynomial interpolation.

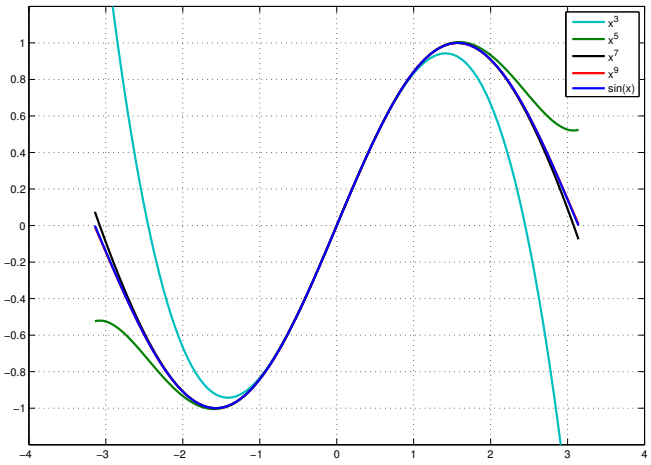
We define the interpolation as a process that, departing from a set of data pairs $(x_0, y_0), (x_1, y_1), \cdots (x_n, y_n)$, allows us to find a function $f(x)$ such that $y_i = f(x_i)$ for all pairs (x_i, y_i) of the set. In the case we use a polynomial as the interpolating function $f(x) \equiv p(x)$, we denote it as polynomial Interpolation.

The interpolation theorem. For any set of $n+1$ pairs of data points $(x_0, y_0), (x_1, y_1), \cdots (x_n, y_n)$, where no two x_i are the same, there is one and only one polynomial $p(x)$ of degree n that interpolates these points, i.e, it satisfies that $y_i = p(x_i)$ for all pairs on the dataset.

If we try to interpolate the points with a polynomial whose degree is less than n, it is possible that we will not find one that fits all the pairs in the dataset. Conversely, if we try to use a polynomial with a degree greater than n, it will not unique. Eventually, if we use a



(a) $\cos(x)$, polinomios 2, 4, 6 y 8 grados ※ polynomials 2, 4, 6 and 8 degrees



(b) $\sin(x)$, polinomios 3, 5, 7 y 9 grados ※polynomials 2, 4, 6 and 8 degrees

Figura 1.2: Polynomios de Taylor para las funciones coseno y seno
Figure 1.2: Taylor polynomial for cosine and sine functions

si el polinomio empleado es de grado n, entonces será siempre el mismo con independencia del método que empleemos para construirlo.

1.2.1. La matriz de Vandermonde

Supongamos que tenemos un conjunto de pares de puntos $\mathcal{A}$,

x	$f(x)$
x_0	y_0
x_1	y_1
x_2	y_2
$\vdots$	$\vdots$
x_n	y_n

Para que un polinomio de orden n,

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

pase por todos los pares de $\mathcal{A}$ debe cumplir,

$$y_i = a_0 + a_1 x_i + a_2 x_i^2 + \cdots + a_n x_i^n,\ \forall (x_i, y_i) \in \mathcal{A}$$

Es decir, obtendríamos un sistema de n ecuaciones lineales, una para cada par de valores, en la que las incógnitas son precisamente los $n+1$ coeficientes a_i del polinomio.

Por ejemplo para los puntos,

x	$f(x)$
1	2
2	1
3	-2

Obtendríamos,

$$\begin{aligned} a_0 + a_1 \cdot 1 + a_2 \cdot 1^2 &= 2 \\ a_0 + a_1 \cdot 2 + a_2 \cdot 2^2 &= 1 \\ a_0 + a_1 \cdot 3 + a_2 \cdot 3^2 &= -2 \end{aligned}$$

que podríamos expresar en forma matricial como,

$$\begin{pmatrix} 1 & 1 & 1^2 \\ 1 & 2 & 2^2 \\ 1 & 3 & 3^2 \end{pmatrix} \cdot \begin{pmatrix} a_0 \\ a_1 \\ a_2 \end{pmatrix} = \begin{pmatrix} 2 \\ 1 \\ -2 \end{pmatrix}$$

degree n polynomial, this polynomial is always the same regardless method used to build it.

1.2.1. The Vandermonde's matrix

Suppose we have a set $\mathcal{A}$ of data points.

For a polynomial of degree n,

to go through all pairs in $\mathcal{A}$, it must satisfy,

So, we would have a system of n linear equations, one for each pair of data points, where the unknowns are the $n+1$ coefficients a_i of the polynomial.

for example, for the dataset,

we get,

which we can express in matrix form as,

Y en general, para n pares de datos,

And, in general, for n data pairs,

$$\begin{pmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^n \\ 1 & x_1 & x_1^2 & \cdots & x_1^n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^n \end{pmatrix} \cdot \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{pmatrix}$$

La matriz de coeficientes del sistema resultante recibe el nombre de matriz de Vandermonde. Está formada por la n primeras potencias de cada uno de los valores de la variable independiente, colocados por filas. Es evidente que cuanto mayor es el número de datos, mayor tenderá a ser la diferencia de tamaño entre los elementos de cada fila. Por ello, en la mayoría de los casos, resulta ser una matriz mal condicionada para resolver el sistema numéricamente. En la práctica, para obtener el polinomio interpolador, se emplean otros métodos alternativos,

The coefficient matrix of the resulting system is called the Vandermonde's matrix. Its elements are hte n first powers of the independent variable values, allocated by rows. It is easy to notice that when the number of data increases the difference among the elements of a row will tend to increase also. For this reason in most cases the Vandermonde's matrix is a poor conditioned matrix and so not suitable for solving the system numerically. For this reason, in practice,the interpolation polynomial is computed using other alternative methods.

1.2.2. El polinomio interpolador de Lagrange.

A partir de los valores $x_0, x_1, \cdots, x_n$, se construye el siguiente conjunto de $n+1$ polinomios de grado n

1.2.2. Lagrange Interpolating Polynomial

Departing from the values $x_0, x_1, \cdots, x_n$, We build the following set of $n+1$ polynomial of degree n

$$l_j(x) = \prod_{\substack{k=0 \\ k \neq j}}^{n} \frac{x - x_k}{x_j - x_k} = \frac{(x-x_0)(x-x_1)\cdots(x-x_{j-1})(x-x_{j+1})\cdots(x-x_n)}{(x_j-x_0)(x_j-x_1)\cdots(x_j-x_{j-1})(x_j-x_{j+1})\cdots(x_j-x_n)}$$

Los polinomios así definidos cumplen una interesante propiedad en relación con los valores $x_0, x_1, \cdots, x_n$, empleados para construirlos,

These polynomial exhibit an interesting property when evaluated at the points $x_0, x_1, \cdots, x_n$, we have used for building them.

$$l_j(x_i) = \begin{cases} 1, & i = j \\ 0, & i \neq j \end{cases}$$

A partir de estos polinomios podemos construir ahora el siguiente polinomio de interpolación empleando las imágenes $y_0, y_1 \cdots, y_n$ correspondientes a los valores $x_0, x_1, \cdots, x_n$,

From this polynomials we can build a interporlation polynomial using the codomain values $y_0, y_1 \cdots, y_n$ corresponding to the domain values $x_0, x_1, \cdots, x_n$,

$$p(x) = \sum_{j=0}^{n} l_j(x) \cdot y_j$$

Efectivamente, es fácil comprobar que, tal y como se ha construido, este polinomio pasa por los pares de puntos x_i, y_i, puesto que

It is ease to check that, using this method for building the interpolation polynomial, it passes through all pairs of points x_i, y_i, be-

$p(x_i) = y_i$.

El siguiente código de en Python calcula el valor en un punto x cualquiera del polinomio de interpolación de Lagrange construido a partir un conjunto de puntos $\mathcal{A} \equiv \{(x_i, y_i)\}$.

cause $p(x_i) = y_i$.

The following python function `lagrang` calculates, at any point x, the Lagrange interpolating polynomial value built from a set of pairs of points $\mathcal{A} \equiv \{(x_i, y_i)\}$.

lgr_pol.py

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Tue Jul 23 10:40:23 2024

@author: juan
A code to implement the Lagrange polynomial
"""
import numpy as np

def lagrang(x,y,x1):
    """
    Function to calculate the Lagrange interpolating polynomial at point x1

    Parameters
    ----------
    x : TYPE real np array
        DESCRIPTION. Table Values x to be interpolated
    y : TYPE real np array
        DESCRIPTION. Table values y to be interpolated
    x1 : TYPE real
        DESCRIPTION. Point a which the polynomial is evaluated

    Returns
    -------
    y1 : TYPE real
        DESCRIPTION. Lagrange polynomial Value at x1. y1 = lagrang(x1)

    """
    y1 = 0
    n = x.shape[0]
    for j in range(0,n):
        lj = 1
        for i in range(0,j):
            lj = lj*(x1-x[i])/(x[j]-x[i])
        for i in range(j+1,n):
            lj = lj*(x1-x[i])/(x[j]-x[i])
        y1 = y1 +lj*y[j]
    return(y1)

def langrmult(x,y,x1):
    """
    Takes the funtion lagrang and calculates the values of the Lagrange
    polinomial at any point in array x1

```

```
46      Parameters
47      ----------
48      x : TYPE real np array
49          DESCRIPTION. Table Values x to be interpolated
50      y : TYPE real np array
51          DESCRIPTION. Table values y to be interpolated
52      x1 : TYPE real
53          DESCRIPTION. array  of point a which the polynomial is evaluated
54
55      Returns
56      -------
57      y1 : TYPE real
58          DESCRIPTION. Lagrange polynomial Valueat x1. y1 = PLagrange(x1)
59
60      """
61      yaux = [] #fast way to create an array same size as x
62      for i in x1:
63          yaux.append(lagrang(x,y,i))
64      y1 = np.array(yaux)
65      return(y1)
66
67
```

1.2.3. Diferencias divididas.

Tanto el método de la matriz de Vandermonde como el de los polinomios de Lagrange, presentan el inconveniente de que si se añade un dato más (x_{n+1}, y_{n+1}) a la colección de datos ya existentes, es preciso recalcular el polinomio de interpolación desde el principio.

El método de las diferencias divididas, permite obtener el polinomio de interpolación en un número menor de operaciones que en el caso del polinomio de Lagrange y además, el cálculo se hace escalonadamente, aprovechando todos los resultados anteriores cuando se añade al polinomio la contribución de un nuevo dato.

El polinomio de orden n de diferencias divididas se construye de la siguiente manera,

1.2.3. Divided diferences.

The two method we have studied so far, Vadermonde matrix and Lagrange polynomial, have a common drawback. If we add a new pair of data (x_{n+1}, y_{n+1}) to the existing data collection, it is necessary to recalculate the interpolation polynomial from scratch.

The divided differences algorithm allows us to build the interpolation polynomial performing less operations than in the case of Lagrange polynomial. Besides, the computing is carried out stepwise, making use of all previous results when a new data pair is added to calculate the interpolation polynomial.

We build the divided differences polynomial of degree n as follows,

$$p_n(x) = a_0 + (x - x_0)\cdot a_1 + (x - x_0)\cdot(x - x_1)\cdot a_2 + \cdots + (x - x_0)\cdot(x - x_1)\cdots(x - x_{n-2})\cdot(x - x_{n-1})\cdot a_n$$

Donde, $(x_0, y_0), (x_1, y_1), \cdots (x_n, y_n)$, representan los datos para los que se quiere calcular el polinomio interpolador de grado n. Si sustituimos las datos en el polinomio, llegamos a un sistema de ecuaciones, triangular inferior,

Where $(x_0, y_0), (x_1, y_1), \cdots (x_n, y_n)$, represent the dataset for which we want to calculate the interpolation polynomial of degree n. If we evaluate the polynomial using the dataset we arrive to a lower triangular system of

en el que las incógnitas son los coeficientes del polinomio.

linear equation, where the polynomial coefficients are the unknowns.

$$\begin{aligned}
&a_0 && = y_0\\
&a_0 + (x_1 - x_0)a_1 && = y_1\\
&a_0 + (x_2 - x_0)a_1 + (x_2 - x_0)(x_2 - x_1)a_2 && = y_2\\
&\cdots\\
&a_0 + (x_n - x_0)a_1 + \cdots + (x_n - x_0)(x_n - x_1)\cdots(x_n - x_{n-2})(x_n - x_{n-1})a_n && = y_n
\end{aligned}$$

Este sistema se resuelve explícitamente empleando un esquema de diferencias divididas.

La diferencia divida de primer orden entre dos puntos (x_0, y_0) y (x_1, y_1) se define como,

This system can be solved using the divided differences algorithm.

We define the first-order two-point divided difference (x_0, y_0) y (x_1, y_1) as,

$$f[x_0, x_1] = \frac{y_1 - y_0}{x_1 - x_0}$$

Para tres puntos, (x_0, y_0), (x_1, y_1) y (x_2, y_2), se define la diferencia dividida de segundo orden como,

For three point, (x_0, y_0), (x_1, y_1) y (x_2, y_2), we define the second-order divided difference as,

$$f[x_0, x_1, x_2] = \frac{f[x_1, x_2] - f[x_0, x_1]}{x_2 - x_0}$$

y, en general definiremos la diferencia dividida de orden i para $i + 1$ puntos como,

and eventually, we can define the $i-$order divide diference for $i + 1$ points as,

$$f[x_0, x_1, \cdots, x_i] = \frac{f[x_1, x_2, \cdots, x_i] - f[x_0, x_1, \cdots, x_{i-1}]}{x_i - x_0}$$

Si despejamos por sustitución progresiva los coeficientes del polinomio de interpolación del sistema triangular inferior obtenido, cada coeficiente puede asociarse a una diferencia dividida,

We can now get the polynomial coefficients applying progressive substitutions to the lower triangular system defined above, then, each coefficient may be related with a divided difference,

$$\begin{aligned}
a_0 &= f[x_0] = y_0\\
a_1 &= f[x_0, x_1]\\
&\vdots\\
a_i &= f[x_0, x_1, \cdots, x_i]\\
&\vdots\\
a_n &= f[x_0, x_1, \cdots, x_n]
\end{aligned}$$

Por tanto, podemos obtener directamente los coeficientes del polinomio calculando las diferencias divididas. Veamos un ejemplo em-

Therefore we can get the polynomial coefficients straightforwardly if we compute the divided differences. Let's see an example using

pleando el siguiente conjunto de cuatro datos,

the following four data set.

x	0	1	3	4
y	1	-1	2	3

Habitualmente, se construye a partir de los datos una tabla, como la 8.2, de diferencias divididas. Las primera columna contiene los valores de la variable x, la siguiente los valores de las diferencias divididas de orden cero (valores de y). A partir de la segunda, las siguientes columnas contienen las diferencias dividas de los elementos de la columna anterior, calculados entre los elementos que ocupan filas consecutivas. La tabla va perdiendo cada vez una fila, hasta llegar a la diferencia dividida de orden $n-1$ de todos los datos iniciales.

Usually, we build the divided difference polynomial, using a table such as table 8.2. the first column contains the values of variable x the second the values of the cero-order divided differences (values of variable y). From the second one on, the following columns contain the divided differences of the elements held in the previous column. These differences are calculated using the elements located on consecutive rows. The table lost a row each time we advance a column. When we arrive to the $n-1$-order divided difference, we have a single value that depends on all initial data.

Tabla 1.2: Tabla de diferencia divididas para cuatro datos
Table 1.2: four data divided difference table

x_i	y_i	$f[x_i, x_{i+1}]$	$f[x_i, x_{i+1}, x_{i+2}]$	$f[x_i, x_{i+1}, x_{i+2}, x_{i+3}]$
$x_0 = 0$	$y_0 = 1$	$f[x_0, x_1] = -2$	$f[x_0, x_1, x_2] = 7/6$	$f[x_0, x_1, x_2, x_3] = -1/3$
$x_1 = 1$	$y_1 = -1$	$f[x_1, x_2] = 3/2$	$f[x_1, x_2, x_3] = -1/6$	
$x_2 = 3$	$y_2 = 2$	$f[x_2, x_3] = 1$		
$x_3 = 4$	$y_3 = 3$			

Los coeficientes del polinomio de diferencias divididas se corresponden con los elementos de la primera fila de la tabla. Por lo que en nuestro ejemplo el polinomio resultante sería,

The divided differences polynomial coefficients are the elements held in the first row of the table. So, for our example, we obtain the following interpolation polynomial,

$$p_3(x) = 1 - 2x + \frac{7}{6}x(x-1) - \frac{1}{3}x(x-1)(x-3)$$

Es importante hacer notar que el polinomio de interpolación obtenido por diferencias divididas siempre aparece representado como suma de productos de binomios $(x-x_0)(x-x_1)\cdots$ y los coeficientes obtenidos corresponden a esta representación y no a la representación habitual de un polinomio como suma de potencias de la variable x.

El siguiente código permite calcular el polinomio de diferencias divididas a partir de un conjunto de n datos. Como el polinomio de diferencias divididas toma una forma especial, es preciso tenerlo en cuenta a la hora de calcular su valor en un punto x determinado. La función `difdiv` permite obtener los coefi-

Notice that the interpolation polynomial, built using divided differences, it always represented as a sum of binomials products $(x-x_0)(x-x_1)\cdots$ and the coefficients computed belong to this representation a not to the standard polynomial representation as a sum of variable x powers.

The following code implement the divided differences polynomial from a set of n data. We must take into account the special form of the polynomial to calculate the value ot takes in an specific x point. Function `difdiv` computes the polynomiañ coefficients departing from a set x, y of data. Function `evdif` calculates the value of the polynomial in a point

cientes del polinomio de diferencias divididas a partir de una coleccion de datos x, y. La función `evdif` permite calcular el valor que toma el polinomio en un punto cualquiera x_i, a partir de los coeficientes calculados y los valores x de la collección de datos.

whatsoever, using the computed coefficientes and the x values of the data set.

dif_div.py

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Wed Jul 24 15:22:01 2024
This file define two funtion. one to calculate the divided differences
polynomial coefficients and the second one to evaluate the divided differences
polynomial any point
@author: juan
"""
import numpy as np
import matplotlib.pyplot as pl
def difdiv(x,y):
    """
    This functions computes the coefficient of a the divided differences
    polynomial using thw data arrays x,y

    Parameters
    ----------
    x : TYPE double
        DESCRIPTION. numpy array with data x
    y : TYPE double
        DESCRIPTION. numpy array with data y

    Returns
    -------
    a : TYPE double
        DESCRIPTION: coefficientes of the polynomial
        p_n(x)=a_0+(x-x_0)a_1+(x-x_0)(x-x_1)a_2+...
        ...+(x-x_0)(x-x_1)...(x-x_{n-2})(x-x_{n-1})a_n
    """
    n = x.shape[0] #number of data
    #we use the y variable to inisialise the vector of coefficients
    a = y.copy() #warning I need to copy to avoid overwrite the values of y
    #we must compute n differences
    for j in range(1,n): #start in 1 we alredy have the order zero differences
        for i in range(j,n):
            a[i] = (a[i]-y[i-1])/(x[i]-x[i-j])
        y = a.copy()
    return(a)

def evdif(a,x,xi):
    """
    Evaluates the divided diferences polynomial from points x and coefficients
    y.
```

```
45
46      Parameters
47      ----------
48      y : TYPE double
49          DESCRIPTION. numpu arrray with divided differences polynomial coefficients
50      x : TYPE double
51          DESCRIPTION. numpy array with x data from the table to be interpolated
52      xi : TYPE double
53          DESCRIPTION. point to calculate the polynomial at. It can be also an array
54
55      Returns
56      -------
57      y1: TYPE: double
58          DESCRIPTION: computed value of the polynomial at x1
59      """
60      n = a.shape[0]
61      yi = a[0] #copy the first coefficient into the result
62      for k in range(1,n):
63          #product of binomials to be multiplied for the coefficients
64          binprod = 1
65          for j in range(k):
66              binprod = binprod*(xi-x[j])
67          yi = yi +a[k]*binprod
68      return(yi)
```

1.2.4. El polinomio de Newton-Gregory

Supone una simplificación al cálculo del polinomio de diferencias divididas para el caso particular en que los datos se encuentran equiespaciados y dispuestos en orden creciente con respecto a los valores de la coordenada x.

En este caso, calcular los valores de las diferencias es mucho mas sencillo. Si pensamos en las diferencias de primer orden, los denominadores de todas ellas son iguales, puesto que los datos están equiespaciados,

1.2.4. The Newton-Gregory polynomial

This polynomial is a simplification of the divided difference polynomial for the case in which the x data of the data set are equispaced and increasingly ordered.

For this case, it is much easier to compute the values of the differences. Think, for instance, in the first-order differences, as far as the data are equispaced ll denominators are equal,

$$\Delta x \equiv x_i - x_{i-1} = h$$

En cuanto a los numeradores, se calcularían de modo análogo al de las diferencias divididas normales,

Concerning the numerators, they are computed as in the standard case of divided differences,

$$\Delta y_0 = y_1 - y_0, \Delta y_1 = y_2 - y_1, \cdots, \Delta y_i = y_{i+1} - y_i, \cdots, \Delta y_{n-1} = y_n - y_{n-1}$$

Las diferencias de orden superior para los numeradores se pueden obtener de modo re-

Higher order differences can be computed recursively from the first-order differences be-

cursivo, a partir de las de orden uno, puesto que los denominadores de todas ellas h, son iguales.

cause the denominator of them all, h are equal.

$$\Delta^2 y_0 = \Delta\left(\Delta y_0\right) = \left(y_2 - y_1\right) - \left(y_1 - y_0\right) = \left(y_2 - 2y_1 + y_0\right)$$

En este caso, el denominador de la diferencia sería $x_2 - x_0 = 2h$, y la diferencia tomaría la forma,

In this case, the difference denominator would be $x_2 - x_0 = 2h$, and the difference would take the form,

$$f[x_0, x_1, x_2] = \frac{\Delta^2 y_0}{2h^2}$$

En general, para la diferencias de orden n tendríamos,

In genral, for the order-n differences we obtain,

$$\Delta^n y_0 = y_n - \binom{n}{1} \cdot y_{n-1} + \binom{n}{2} \cdot y_{n-2} - \cdots + (-1)^n \cdot y_0$$

Donde se ha hecho uso de la expresión binomial,

Where we have used the binomial expression,

$$\binom{k}{l} = \frac{k!}{l! \cdot (k-l)!}$$

Para obtener la diferencia dividida de orden n, bastaría ahora dividir por $n! \cdot h^n$.

And, eventually, we obtain the order-n divided difference just dividing by $n! \cdot h^n$.

$$f\left[x_0, x_1, \cdots, x_n\right] = \frac{\Delta^n y_0}{n! \cdot h^n}$$

A partir de las diferencias, podemos representar el polinomio de diferencias divididas resultante como,

Once we have got the differences we can write the divided differences polynomial as,

$$p_n(x) = y_0 + \frac{x - x_0}{h}\Delta y_0 + \frac{(x - x_1) \cdot (x - x_0)}{2 \cdot h^2}\Delta^2 y_0 + \cdots + \frac{(x - x_{n-1}) \cdots (x - x_1) \cdot (x - x_0)}{n! \cdot h^n}\Delta^n y_0$$

Este polinomio se conoce como el polinomio de Newton-Gregory, y podría considerarse como una aproximación numérica al polinomio de Taylor de orden n de la posible función asociada a los datos empleados.

En este caso, podríamos construir la tabla para obtener los coeficientes del polinomio, calculando en cada columna simplemente las diferencias de los elementos de la columna anterior. Por ejemplo,

Una vez calculadas las diferencias, basta dividir por $n! \cdot h^n$ los elementos de la primera fila de la tabla,

This polynomial is known as the Newton-Gregory polynomial, and it could be considered as a numerical approximation to the n-degree Taylor polynomial of the (possible) function associated to the dataset.

In this case, we can build the table to obtain the polynomial coefficients, computing in each column just the differences of the previous column. For example,

Once we have computed the differences, it is enough to divide by $n! \cdot h^n$ the elements of the table first row,

$$a_0 = 1, a_1 = \frac{-2}{1}, a_2 = \frac{5}{2 \cdot 1^2}, a_3 = \frac{-7}{6 \cdot 1^3}$$

Tabla 1.3: Tabla de diferencias para el polinomio de Newton-Gregory de cuatro datos
Table 1.3: Table of differences for a Newton-Gregory polynomial of four data

x_i	y_i	Δy_i	$\Delta^2 y_i$	$\Delta^3 y_i$
$x_0 = 0$	$y_0 = 1$	-2	5	-7
$x_1 = 1$	$y_1 = -1$	3	-2	
$x_2 = 2$	$y_2 = 2$	1		
$x_3 = 3$	$y_3 = 3$			

El siguiente código muestra un ejemplo de implementación en Python del polinomio de Newton-Gregory

The following code shows an example of Python implementation for the Newton-Gregory polynomial

newton_gregory.py

```
# -*- coding: utf-8 -*-
"""
Created on Thu Jul 25 20:12:49 2024
A gruby version of Newton-Gregory polynomial
@author: abierto
"""
import numpy as np
from dif_div import evdif
def newgre(x,y,x1=0):
    """


    Parameters
    ----------
    x : TYPE numpy array of data
        DESCRIPTION.
    y : TYPE numpy array of data
        DESCRIPTION.
    x1 : TYPE a point to calculate the value the polynomial
         takes
        DESCRIPTION.
        This function takes a data set, represented by the
        x and y array and obtain the corresponding
        newton-gregory polynomial.Besides it calculates the
        value of the polypolimial at the point x1. if no x1
        value is supplied then it takes x1 =0 by default.
        The program returns the polynomial coeficients and
        the value calculated at point x1

    Returns
    -------
    a : TYPE numpy array of polynomial coeficients
        DESCRIPTION.
    y1: TYPE real value
        DESCRIPTION. The value the polynomial takes at x1

    """
    n = x.shape[0]
```

```
39      a = y.copy() #we start the coefficient with the 0-order differences,i.e.
40                   #y values. remember use a copy to avoid overwrite the arrays
41      h = x[1] -x[0]
42      #here start the loop to calculate the differences
43      for j in range(1,n):
44          #for each iteration we calculate the differences
45          #of higher order. But we only need the first difference. So we restart
46          #the inner loop in the value j of the outer one
47          for i in range(j,n):
48              #now it is enough to divide for the distance h betwwen x point
49              #multiplied by order j of the diference
50              a[i] = (a[i]-y[i-1])/(j*h)
51          y = a.copy()
52      #now we calculate the value(s) of the polynomial using the
53      #function built to evaluate divided diferences polynomials
54      # if type(x1) != np.ndarray:
55      #     x1 = np.array([x1])
56      # y1 = []
57      # for i in x1:
58      #     y1.append(evdif(a,x,i))
59      # y1 = np.array(y1)
60      y1 = evdif(a,x,x1)
61      return(a,y1)
```

1.3. Interpolación por intervalos.

Hasta ahora, hemos visto cómo interpolar un conjunto de $n+1$ datos mediante un polinomio de grado n. En muchos casos, especialmente cuando el número de datos es suficientemente alto, los resultados de dicha interpolación pueden no ser satisfactorios. La razón es que el grado del polinomio de interpolación crece linealmente con el número de puntos a interpolar, así por ejemplo para interpolar 11 datos necesitamos un polinomio de grado 10. Desde un punto de vista numérico, este tipo de polinomios pueden dar grandes errores debido al redondeo. Por otro lado, y dependiendo de la disposición de los datos para los que se realiza la interpolación, puede resultar que el polinomio obtenido tome una forma demasiado complicada para los valores comprendidos entres los datos interpolados..

La figura 8.3 muestra el polinomio de interpolación de grado nueve para un conjunto de 10 datos. Es fácil darse cuenta, simplemen-

1.3. Piecewise interpolation

So far, we have seen how to interpolate a set of $n+1$ data using a degree n polynomial. In many case, in particular when the number of data is high, the results of such interpolation could be very poor. The problem comes from the linear increasing of the polynomial degree with the number of data. So, if we want to build the interpolation polynomial for a 11 data, we come up with a 10-degree polynomial. From a numerical point of view, this kinda polynomial are prone to cast large errors due to the rounding process. On the other hand, depending on how data used to compute the interpolation are distributed, the polynomial could take a too complicate shape that hardly can be related with the information held in the dataset.

Figure 8.3 shows a interpolating polynomial of degree nine obtained from a set of ten data. It is easy to realise, just for simple inspection, that there in not reason to justify the polynomial curvature between points 1 and 2 or between the points 9 and 10.

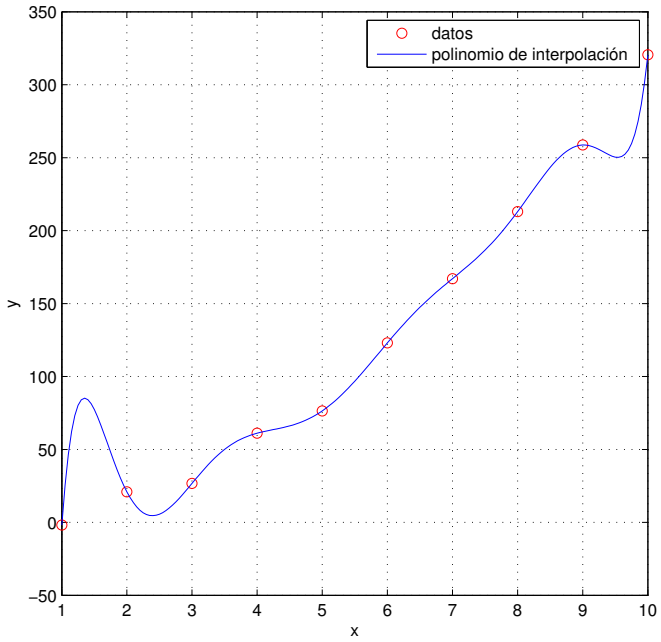


Figura 1.3: Polinomio de interpolación de grado nueve obtenido a partir de un conjunto de diez datos
Figure 1.3: Nine degree interpoling polynomial obtained using a set of ten data

te observando los datos, que no hay ninguna razón que justifique las curvas que traza el polinomio entre los puntos 1 y 2 o los puntos 9 y 10, por ejemplo.

En muchos casos es preferible no emplear todos los datos disponibles para obtener un único polinomio de interpolación. En su lugar, lo que se hace es dividir el conjunto de datos en varios grupos —normalmente se agrupan formando intervalos de datos consecutivos— y obtener varios polinomios de menor grado, de modo que cada uno interpole los datos de un grupo distinto.

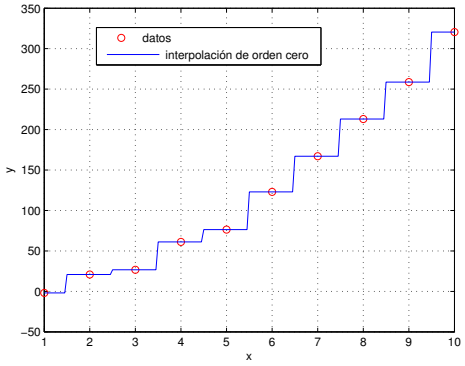
El grado de los polinomios empleados deberá estar, en principio, relacionado con los datos contenidos en cada tramo.

interpolación de orden cero si hacemos que cada intervalo contenga un solo dato, obtendríamos polinomios de interpolación de grado cero, $a_{0i} = y_i$. El resultado, es un conjunto

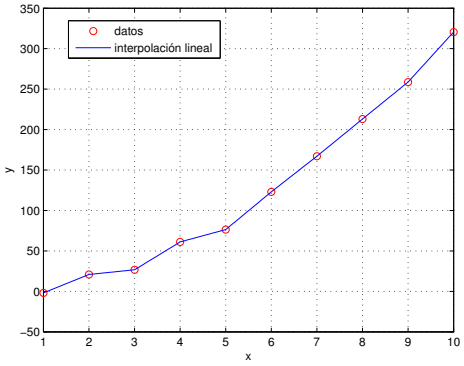
For these reasons, in many cases it is better not to use the whole dataset to build a single interpolation polynomial of maximum degree. Instead, a common practice is to divide the dataset in several groups of data —usually they are gathered using interval of consecutive data– and build several polynomial of lower degree, each one interpolating the data of a different group.

The degree of the polynomials should be related with the number of data allocated in each interval.

Zero-order interpolation. If we get interval which only hold a single data pair X, y, then we get zero-order interpolating polynomials, $a_{0i} = y_i$. The result is a stepwise inter-



(a) Interpolación de orden cero / Zero-order interpolation



(b) Interpolación lineal Linear interpolation

Figura 1.4: Interpolaciones de orden cero y lineal para los datos de la figura 8.3
Figure 1.4: Zero-order and linear interpolation for the figure data

de escalones cuya valor varía de un intervalo a otro de acuerdo con el dato representativo contenido en cada tramo. La figura 8.4(a) muestra el resultado de la interpolación de orden cero para los mismos diez datos de la figura 8.3.

interpolación lineal. En este caso, se dividen los datos en grupos de dos. Cada par de datos consecutivos se interpola calculando la recta que pasa por ellos. La interpolación lineal se emplea en muchas aplicaciones debido a su sencillez de cálculo. La figura 8.4(b), muestra el resultado de aproximar linealmente los mismos datos contenidos en los ejemplos anteriores.

Siguiendo el mismo procedimiento, aumentando el número de datos contenidos en cada intervalo, podríamos definir una interpolación cuadrática, con polinomios de segundo grado, tomando intervalos que contengan tres puntos, una interpolación cúbica, para intervalos de cuatro puntos etc.

1.3.1. Interpolación mediante splines cúbicos

Hemos descrito antes cómo el polinomio interpolador de orden n para un conjunto de $n+1$ datos puede presentar el inconveniente

polation which values changes from one interval to the next, taken the data value defined for each interval according to the dataset. Figure 8.4(a) shows the zero-order interpolation result for the same ten data of figure 8.3.

Linear interpolation. In this case, we divide the dataset in groups of two data. Each two consecutive data are interpolated calculating the line that pass through them. The linear interpolation is very commonly used due to its computing simplicity. Figure 8.4(b) shows the result of interpolating the same data utilised in previous examples, using linear interpolation.

Following the same procedure, we can increase the number of data include in each interval and define quadratic interpolation, using second-degree polynomials and taking three points in each interval; cubic interpolation,using third-degree polynomial and four point intervals. etc.

1.3.1. Cubic spline interpolation

WE have seen how the n-degree interpolating polynomial for a set of $n+1$ data, could present a quite complex shape that does not represent well the information supplied by the dataset. The piecewise interpolation that we

de complicar excesivamente la forma de la curva obtenida entre los puntos interpolados. La interpolación a tramos que acabamos de describir, simplifica la forma de la curva entre los puntos pero presenta el problemas de la continuidad en las uniones entre tramos sucesivos. Sería deseable encontrar métodos de interpolación que fueran capaces de solucionar ambos problemas simultáneamente. Una buena aproximación a dicha solución la proporcionan los *splines*.

Una función *spline* está formada por un conjunto de polinomios, cada uno definido en un intervalo, que se unen entre sí obedeciendo a ciertas condiciones de continuidad.

Supongamos que tenemos una tabla de datos cualquiera,

have described so far, simplifies the shape of the interpolating curve but have in turn the problem that it loses the continuity in the union between consecutive intervals. It could be valuable to find interpolation methods that may be able to cope with both problems simultaneously. A good approach to solve these problem is supplied by *spline* interpolation.

A *spline* is function built using a set of polynomials, any one of them defined in an interval. The spline polynomial are connected in the ends of the intervals meeting certain continuity conditions.

Suposse we have a data table whatsoever,

x	x_0	x_1	$\cdots$	x_n
y	y_0	y_1	$\cdots$	y_n

Para construir una función *spline* S de orden m, que interpole los datos de la tabla, se definen intervalos tomando como extremos dos puntos consecutivos de la tabla y un polinomio de grado m para cada uno de los intervalos,

To build a m order *spline* function S for interpolating the table data, we define intervals taking two consecutive point onn the table as limits. Then, we define an m degree polynomial for each interval.

$$S = \begin{cases} S_0(x),\ x \in [x_0, x_1] \\ S_1(x),\ x \in [x_1, x_2] \\ \vdots \\ S_i(x),\ x \in [x_i, x_{i+1}] \\ \vdots \\ S_{n-1}(x),\ x \in [x_{n-1}, x_n] \end{cases}$$

Para que S sea una función Spline de orden m debe cumplir que sea continua y tenga $m-1$ derivadas continuas en el intervalo $[x_0, x_n]$ en que se desean interpolar los datos.

Para asegurar la continuidad, los polinomios que forman S deben cumplir las siguientes condiciones en sus extremos;

For s to be a order m spline function it should be continue and it should have $m-1$ continue derivatives in the interval $[x_0, x_n]$ in which we want to interpolate the data.

To ensure the continuity of the polynomials that build S they must satisfy the following condition at their ends,

$$\begin{aligned} S_i(x_{i+1}) &= S_{i+1}(x_{i+1}),\ (1 \le i \le n-1) \\ S_i'(x_{i+1}) &= S_{i+1}'(x_{i+1}),\ (1 \le i \le n-1) \\ S_i''(x_{i+1}) &= S_{i+1}''(x_{i+1}),\ (1 \le i \le n-1) \\ &\vdots \\ S_i^{m-1}(x_{i+1}) &= S_{i+1}^{m-1}(x_{i+1}),\ (1 \le i \le n-1) \end{aligned}$$

Es decir, dos polinomios consecutivos del spline y sus $m-1$ primeras derivadas, deben tomar los mismos valores en el extremo común.

Una consecuencia inmediata de las condiciones de continuidad exigidas a los splines es que sus derivadas sucesivas, S', $S'', \cdots$ son a su vez funciones spline de orden $m-1$, $m-2, \cdots$. Por otro lado, las condiciones de continuidad suministran $(n-1)\cdot m$ ecuaciones que, unidas a las $n+1$ condiciones de interpolación —cada polinomio debe pasar por los datos que constituyen los extremos de su intervalo de definición—, suministran un total de $n\cdot(m+1)-(m-1)$ ecuaciones. Este número es insuficiente para determinar los $(m+1)\cdot n$ parámetros correspondientes a los n polinomios de grado m empleados en la interpolación. Las $m-1$ ecuaciones que faltan se obtienen imponiendo a los splines condiciones adicionales.

Splines cúbicos. Los splines más empleados son los formados por polinomios de tercer grado. En total, tendremos que determinar $(m+1)\cdot n = 4\cdot n$ coeficientes para obtener todos los polinomios que componen el spline. Las condiciones de continuidad más la de interpolación suministran en total $3\cdot(n-1)+n+1 = 4\cdot n-2$ ecuaciones. Necesitamos imponer al spline dos condiciones más. Algunas típicas son,

1. Splines naturales $S''(x_0) = S''(x_n) = 0$
2. Splines con valor conocido en la primera derivada de los extremos $S'(x_0) = y'_0, S'(x_n) = y'_n$
3. Splines periódicos,

$$\begin{cases} S(x_0) = S(x_n) \\ S'(x_0) = S'(x_n) \\ S''(x_0) = S''(x_n) \end{cases}$$

Intentar construir un sistema de ecuaciones para obtener a la vez todos los coeficientes de todos los polinomios es una tarea excesivamente compleja porque hay demasiados parámetros. Para abordar el problema partimos del hecho de que $S''(x)$ es también un

That is, two consecutive polynomial belonging to the spline and their $m-1$ first derivatives must take the same values in the common end.

A straightforward consequence of the continuity conditions impose to spline functions is that their successive derivatives S'. $S'', \cdots$ are in turn order $m-1$, $m-2\cdots$ spline functions too. Besides, the continuity conditions supply $(n-1)\cdot m$ equations that, together with the interpolation conditions —each polynomial should pass through the two data point that defines the ends of its definition interval—, sum up $n\cdot(m+1)-(m-1)$ equations. This number is insufficient for obtaining the $(m+1\cdot n)$ parameters belonging to the the n polynomial of degree m used in the spline interpolation. We need $m-1$ equations more that are defined imposing to the polynomials additional conditions.

Cubic Spline. Probably, the most used splines are those composed of third-degree polynomials. We must need to determine $(m+1)\dot{n} = 4\cdot n$ coefficients to obtain all the polynomials that compose the spline. The continuity plus the interpolation conditions supply $3\cdot(n-1)+n+1 = 4\cdot n-2$ equations. We need to impose two more conditions to the spline. Some frecuently used conditions are,

1. natural Spline $S''(x_0) = S''(x_n) = 0$
2. Spline with a known value for the derivatives at the ends of the interval $S'(x_0) = y'_0, S'(x_n) = y'_n$
3. Periodic Spline,

Try to build a system of equations to obtain all the coefficients of all the polynomial at a time, is an arduous task. There are too many parameters. We can address the problem starting with $S''(x)$, which it is also a order-1 spline for the points we want to interpolate. If we

spline de orden 1 para los puntos interpolados. Si los definimos como,

define it as,

$$S_i''(x) = -M_i \frac{x - x_{i+1}}{h_i} + M_{i+1} \frac{x - x_i}{h_i},\ i = 0, \cdots, n-1$$

donde $h_i = x_{i+1} - x_i$ representa el ancho de cada intervalo y donde cada valor $M_i = S''(x_i)$ será una de las incógnitas que deberemos resolver.

Si integramos dos veces la expresión anterior,

Where $h_i = x_{i+1} - x_i$ stand for each interval width and where the value $M_i = S''(x_i)$ will be the unknown we have to solve.

Now, we integrate two times this expression, $S_i''(x)$, to obtain,

$$S_i'(x) = -M_i \frac{(x - x_{i+1})^2}{2 \cdot h_i} + M_{i+1} \frac{(x - x_i)^2}{2 \cdot h_i} + A_i,\ i = 0, \cdots, n-1$$
$$S_i(x) = -M_i \frac{(x - x_{i+1})^3}{6 \cdot h_i} + M_{i+1} \frac{(x - x_i)^3}{6 \cdot h_i} + A_i(x - x_i) + B_i,\ i = 0, \cdots, n-1$$

Empezamos por imponer las condiciones de interpolación: el polinomio S_i debe pasar por el punto (x_i, y_i),

Let's start imposing the interpolation conditions: the polynomial S_i must to pass through the point (x_i, y_i),

$$S_i(x_i) = -M_i \frac{(x_i - x_{i+1})^3}{6 \cdot h_i} + B_i = y_i \Rightarrow B_i = y_i - \frac{M_i \cdot h_i^2}{6},\ i = 0, \cdots, n-1$$

A continuación imponemos continuidad del spline en los nodos comunes: El polinomio S_{i-1} también debe pasar por el punto (x_i, y_i),

Then, we impose the continuity of the spline in the points common to two polynomials: Polynomial S_{i-1} must also pass through the point (x_i, y_i),

$$S_{i-1}(x_i) = M_i \frac{(x_i - x_{i-1})^3}{6 \cdot h_i} + A_{i-1}(x_i - x_{i-1}) + \overbrace{y_{i-1} - \frac{M_{i-1} \cdot h_{i-1}^2}{6}}^{B_{i-1}} = y_i \Rightarrow$$
$$\Rightarrow A_{i-1} = \frac{y_i - y_{i-1}}{h_{i-1}} - \frac{M_i - M_{i-1}}{6} \cdot h_{i-1},\ i = 1, \cdots, n$$

Y por tanto,

and thus,

$$A_i = \frac{y_{i+1} - y_i}{h_i} - \frac{M_{i+1} - M_i}{6} \cdot h_i,\ i = 0, \cdots, n-1$$

En tercer lugar imponemos la condición de que las derivadas también sean continuas en los nodos comunes,

Thirdly, we impose that continuity condition to the derivatives, in the common end of two consecutive polynomials,

$$S_i'(x_i) = -M_i \frac{(x_i - x_{i+1})^2}{2 \cdot h_i} + M_{i+1} \frac{(x_i - x_i)^2}{2 \cdot h_i} + \frac{y_{i+1} - y_i}{h_i} - \frac{M_{i+1} - M_i}{6} \cdot h_i,\ i = 0, \cdots, n-1$$
$$S_{i-1}'(x_i) = -M_{i-1} \frac{(x_i - x_i)^2}{2 \cdot h_{i-1}} + M_i \frac{(x_i - x_{i-1})^2}{2 \cdot h_{i-1}} + \frac{y_i - y_{i-1}}{h_{i-1}} - \frac{M_i - M_{i-1}}{6} \cdot h_{i-1},\ i = 1, \cdots, n$$
$$S_i'(x_i) = S_{i-1}'(x_i),\ i = 1, \cdots, n-1 \Rightarrow$$
$$\Rightarrow -M_i \frac{h_i}{2} + \frac{y_{i+1} - y_i}{h_i} - \frac{M_{i+1} - M_i}{6} \cdot h_i = M_i \frac{h_{i-1}}{2} + \frac{y_i - y_{i-1}}{h_{i-1}} - \frac{M_i - M_{i-1}}{6} \cdot h_{i-1}$$

Si agrupamos a un lado los valores M_{i-1}, M_i, M_{i+1},

$$h_{i-1} \cdot M_{i-1} + 2 \cdot (h_{i-1} + h_i) \cdot M_i + h_i \cdot M_{i+1} = 6 \cdot \left(\frac{y_{i+1} - y_i}{h_i} - \frac{y_i - y_{i-1}}{h_{i-1}} \right)$$
$$i = 1, \cdots, n-1$$

En total tenemos $M_0, \cdots, M_n$, $n+1$ incógnitas y la expresión anterior, solo nos suministra $n-1$ ecuaciones. Necesitamos dos ecuaciones más. Si imponemos la condición de splines naturales, para el extremo de la izquierda del primer polinomio y para el extremo de la derecha del último,

If we group at one side the values M_{i-1}, M_i, M_{i+1},

We have in total $M_0, \cdots, M_n$, $n+1$ unknowns and the above expression only supply $n-1$ equations. We need two more equations. If we impose the conditions for a natural spline for the left end of the first polynomial and for the right end of the last one,

$$M_0 = S''(x_0) = 0$$
$$M_n = S''(x_n) = 0$$

Con estas condiciones y la expresión obtenida para el resto de los M_i, podemos construir un sistema de ecuaciones tridiagonal

With these last conditions and the expression obtained for the remaining M_i, we can build a tridiagonal system of equations

$$\begin{pmatrix} 2(h_0+h_1) & h_1 & 0 & 0 & \cdots & 0 & 0 \\ h_1 & 2(h_1+h_2) & h_2 & 0 & \cdots & 0 & 0 \\ 0 & h_2 & 2(h_2+h_3) & h_3 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & 2(h_{n-3}+h_{n-2}) & h_{n-2} \\ 0 & 0 & 0 & 0 & \cdots & h_{n-2} & 2(h_{n-2}+h_{n-1}) \end{pmatrix} \cdot \begin{pmatrix} M_1 \\ M_2 \\ M_3 \\ \vdots \\ M_{n-1} \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_{n-1} \end{pmatrix}$$

Donde hemos hecho,

Where,

$$b_i = 6 \cdot \left(\frac{y_{i+1} - y_i}{h_i} - \frac{y_i - y_{i-1}}{h_{i-1}} \right)$$

Tenemos un sistema de ecuaciones en el que la matriz de coeficientes es tridiagonal y además diagonal dominante, por lo que podríamos emplear cualquiera de los métodos vistos en el capítulo **??**. Una vez resuelto el sistema y obtenidos los valores de M_i, obtenemos los valores de A_i y B_i a Partir de las ecuaciones obtenidas más arriba.

Por último, la forma habitual de definir el polinomio de grado 3 S_i, empleado para interpolar los valores del intervalo $[x_i, x_{i+1}]$, mediante splines cúbicos se define como,

So, we have a system of equations for which the coefficients matrix is tridiagonal and, besides, it is a dominant diagonal matrix. For this reason, we can solve it with anyone of the methods described in chapter **??**. Once the system is solved, we get the values of M_i and the values of A_i and B_i using the equations described above.

A last remark: we usually define the 3-degree polynomial, S_i used for interpolating the values inside the interval $[x_i, x_{i+1}]$ as follows,

$$S_i(x) = \alpha_i + \beta_i(x - x_i) + \gamma_i(x - x_i)^2 + \delta_i(x - x_i)^3, \ x \in [x_i, x_{i+1}], \ (i = 0, 1, \cdots, n-1)$$

Donde,

Where,

$$\alpha_i = y_i$$
$$\beta_i = \frac{y_{i+1} - y_i}{h_i} - \frac{M_i \cdot h_i}{3} - \frac{M_{i+1} \cdot h_i}{6}$$
$$\gamma_i = \frac{M_i}{2}$$
$$\delta_i = \frac{M_{i+1} - M_i}{6 \cdot h_i}$$

La siguiente función permite obtener los coeficientes y el resultado de interpolar un conjunto de puntos mediante splines cúbicos,

The following Python code computes the coefficients of the cubic spline defined using a dataset and the value taken by the spline at any point.

cubic_spline.py

```python
# -*- coding: utf-8 -*-
"""
Created on Wed Jul 31 11:01:10 2024
This script implemets some functions to compute Cubic spline
@author: abierto
"""
import numpy as np

def spcubic(x,y):
    """
    Function to calculate the spline coefficients
    Parameters
    ----------
    x : TYPE real np array
        DESCRIPTION. Table Values x to be interpolated
    y : TYPE real np array
        DESCRIPTION. Table values y to be interpolated
    Returns
    -------
    h: Type array of diferecences betwee x consecutive data
    M,A,B: arrays with coefficientes for the polinomial that
           that compose the spline expressed as differences
           (see manual)
    C : TYPE numpy array size [I,4]
        I->intervals i.e number of data - 1
        4->Each 3-degree polinomial has 4 coefficients
        DESCRIPTION.spline coeficients in standard polynomial representation
    """
    l = x.shape[0] #number of data
    dy = np.zeros(l-1)
    h = dy.copy()

    for i in range(0,l-1):
        h[i] = x[i+1] - x[i]
        dy[i] = y[i+1]-y[i]
```

```
36      CSPp = np.zeros(l-2)
37      b = CSPp.copy()
38      for i in range(l-2):
39          CSPp[i] = 2*(h[i+1]+h[i]) #ppal diagonal
40          b[i] = 6*(dy[i+1]/h[i+1]-dy[i]/h[i])
41      #building the system matrix (a pro. would use a spare matrix here)
42      CSP = np.diag(CSPp)+np.diag(h[1:-1],1)+np.diag(h[1:-1],-1)
43      M = np.linalg.solve(CSP,b)
44      M = np.insert(M,0,0)
45      M= np.append(M,0)
46      A = np.zeros(l-1)
47      B = np.zeros(l-1)
48      for i in range(l-1):
49          A[i] = dy[i]/h[i] - (M[i+1]-M[i])*h[i]/6
50          B[i]=y[i]-M[i]*h[i]**2/6
51      C = np.zeros([l-1,4])
52      for i in range(l-1):
53          C[i,0]=y[i];
54          C[i,1]=dy[i]/h[i]-M[i]*h[i]/3-M[i+1]*h[i]/6;
55          C[i,2]=M[i]/2;
56          C[i,3]=(M[i+1]-M[i])/(6*h[i]);
57      return M,A,B,C,h
58
59  def evspc(M,A,B,x,xi):
60      """
61      This function calculates the value of a interpolating spline in point xi
62
63      Parameters
64      ----------
65      M : TYPE nummpy array
66          DESCRIPTION. Coefficients M of the spline
67      A : TYPE numpy array cofficients A of the spline
68          DESCRIPTION.
69      B : TYPE
70          DESCRIPTION.
71      x : TYPE numpy array
72          DESCRIPTION. x data of the datatable interpolated
73      xi : TYPE double
74          DESCRIPTION. point to calculate the spline value
75
76      Returns
77      -------
78      yi : TYPE double
79          DESCRIPTION. value calculate for the spline yi =s(xi)
80
81      """
82      l = x.shape[0]
83      h = np.zeros(l-1)
84      for i in range(0,l-1):
85          h[i] = x[i+1] - x[i]
86      j = 0
87      while xi > x[j]:
```

```
88          j = j+1
89      if j > l - 1:
90         j = l - 1
91      elif j < 1:
92          j = 1
93      yi=-M[j-1]*(xi-x[j])**3/(6*h[j-1])+ M[j]*(xi-x[j-1])**3/(6*h[j-1])\
94          +A[j-1]*(xi-x[j-1])+B[j-1]
95      return yi
```

La figura, 8.5 muestra el resultado de interpolar mediante un spline cúbico, los datos contenidos en la figura 8.3. Es fácil observar cómo ahora los polinomios de interpolación dan como resultado una curva suave en los datos interpolados y en la que además las curvas son también suaves, sin presentar variaciones extrañas, para los puntos contenidos en cada intervalo entre dos datos.

1.3.2. Funciones propias de Python para interpolación por intervalos

Para realizar una interpolación por intervalos mediante cualquiera de los procedimientos descritos, podemos emplear el subpaquete de scipy `Scipy.interpolate`. Este paquete incorpora múltiples métodos de interpolación; solo veremos dos:

La función `interp1d`. Esta función admite como variables de entrada dos arrays con los valores de las coordenadas x e y de los datos que se desea interpolar. Además, admite como variable de entrada una cadena de caracteres que indica el método con el que se quiere realizar la interpolación. Dicha variable puede tomar los valores:

1. `'nearest'`. Interpola el intervalo empleando el valor y_i correspondiente al valor x_i más cercano al punto que se quiere interpolar. El resultado es una interpolación a escalones. del conjunto de datos que se desea interpolar.
2. `'next'`. Interpola el interpalo empleando el valor de y_i correspondiente al punto x_i de los datos que cierra el intervalo.
3. `'previous'` Interpola el interpalo empleando el valor de y_i correspondiente

Figure 8.5 shows the result of interpolating the same data of figure 8.3 using a cubic spline. It is ease to check how the interpolation polynomials which compose the spline cast a smooth result in the points interpolated and how the curves shapes are reasonable smooth also in the intervals between the data. They do not present variations difficult to explain using the data.

1.3.2. Python functions for piecewise interpolation

To carry out a piecewise interpolation using anyone of the method above described, we can use the `scipy` sub-package `scypi.interpolate`. This package includes many intepolation methods; we will only describe two of them:

The `interp1d` function. This function takes two arrays as inpunt variables with the ccordenates x and y values belonging to the data we want to interpolate. In addition, the function may take also a character string, as an input variable which defines the method we want to use to carry out the interpolation. This last variable can take the following values:

1. `'nearest'`. It interpolates the interval using the value y_i corresponding to the value x_i nearest to the point at which we want to compute the interpolation. The result is stepwise interpolation.
2. `'next'`. It interpolates the interval using the value y_i corresponding to the value x_i which closes the interval in which we want to compute the interpolation.
3. `'previous'` It interpolates the interval using the value y_i corresponding to the

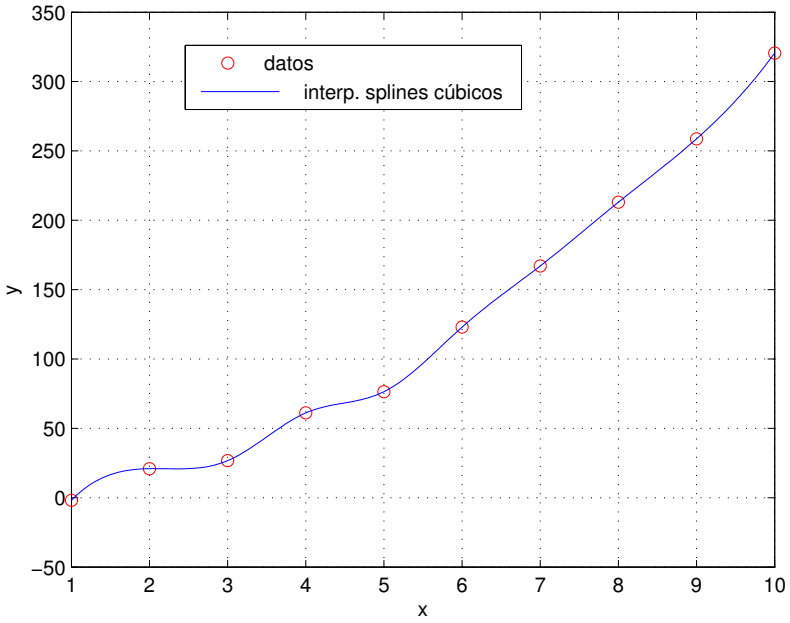


Figura 1.5: Interpolación mediante spline cúbico de los datos de la figura 8.3

Figure 1.5: Cubic Spline interpolation for the data represented in the figure 8.3

al punto x_i de los datos que abre el intervalo.

4. `'linear'` realiza una interpolación lineal entre los extremos del intervalo que se desea intepolar. Esta es la opción por defecto.

`interp1d` Devuelve una funcion que admite como variable de entrada un array con los puntos para para los que se quiere calcular el valor de la interpolación y devuelve como salidad un array con los resultados obtenidos. El siguiente código muestra el modo de usar el comando `interp1d`. Para probarlo se han creado dos array `x` e `y` que contienen el conjunto de datos que se empleará para calcular la interpolación. Además, se ha creado otro vector `xi` que contiene los puntos para los que se quiere calcular el resultado de la interpolación.

value x_i which opens the interval in which we want to compute the interpolation.

4. `'linear'` It calculates a linear interpolation between the points at the ends of the interval we want to interpolate. This is the default option.

`interp1d` returns a function that, in turn, takes as input variable an array with the points we want to calculate the interpolation at, and returns an array we the computed results. The following code shows how to use the function `interp1d`. To try it we have created two arrays `x` and `y` which contain the dataset we want to interpolate. Besidas, we have created another array `xi` with the point we want to compute the interpolation at.

ejemplo_interp1d.py

```
1 # -*- coding: utf-8 -*-
```

```
"""
Created on Fri Aug  2 16:55:04 2024
Ejemplo de uso de la función interp1d de Scipy
@author: abierto
"""
import numpy as np
from scipy.interpolate import interp1d
from matplotlib import pyplot as pl
x = np.array([ 1.,  2.,  3.,  4.,  5.,  6.,  7.,  8.,  9., 10.])
y = np.array([-1.8143451, 20.914356, 26.714303, 61.129501,\
              76.414728, 123.00032, 167.06809, 212.97832,\
              258.67911, 320.53422])
#dibujamos los puntos
pl.plot(x,y,'o')
#creamos unos puntos sobre los que interpolar
xi = np.arange(1.,10.,0.01)
#creamos la función interpoladora empleando inter1d
flin = interp1d(x,y,'linear')#puede omitirse linear el la opcion por defecto
#empleamos la funcion para obtener los valores de los puntos interpolados
yi = flin(xi)
pl.plot(xi,yi)

fnear = interp1d(x,y,'nearest')
yi = fnear(xi)
pl.plot(xi,yi)

fnext = interp1d(x,y,'next')
yi = fnext(xi)
pl.plot(xi,yi)
#and so on and so forth...
```

La segunda función interpoladora de `scipy.interpolation` que vamos a describir se llama `CubicSpline`. Permite interpolar empleando Splines cúbicos. En este caso, además de los datos x e y necesarios para definir el spline, la función admite el parametro `bc_type`, que permite determinar las condiciones de contorno en los extremos del spline. Las opciones posibles son:

`bc_type='not__knot'`. Es la opcion por defecto. Hace que el polinomio empleado en el primer y segundo segmento en los extremos del spline sea el mismo.

`bc_type='natural'`. Spline natural.

`bc_type='periodic'`. Spline periódico.

Hay más opciones; los interesados pueden encontrarlas en la documentación de `scipy`. El uso es similar al descrito para `interp1d`. A partir de los datos se crea una función inter-

The second interpolating function, from `scipy.interpolation` we are going to describe is `CubicSpline`. This function, as can be expected, uses cubic splines to perform the interpolation of a dataset. In this cases, in addition to the x and y data needed to define the spline, the function defines an input paramenter `bc_type`, which allows us to stablish the boundary conditions in the ends of the spline. Some of the available options are:

`bc_type='not__knot'`. This is the default option. The first and second segments at an end of the spline uses the same polynomial.

`bc_type='natural'`. Natural spline.

`bc_type='periodic'`. Periodic spline.

There are more options. Those interested can find them in `scipy` documentation. Its use is similar to the use of `interp1d`. From the data we want to interpolate `CubiSpline` genera-

poladora que permite calcular el valor de la imagen en los puntos que se desee. Por ejemplo, para splines naturales sería,

tes an interpoling function which alows us to compute the spline value in the point we wish. For instance, for natural splines we write,

```
spnat = CubicSpline(x,y,'natural')
yi = spnat(xi)
```

1.4. Ajuste polinómico por el método de mínimos cuadrados

Los métodos de interpolación que hemos descrito hasta ahora, pretenden encontrar un polinomio o una función definida a partir de polinomios que pase por un conjunto de datos. En el caso del ajuste por mínimos cuadrados, lo que se pretende es buscar el polinomio, de un grado dado, que mejor se aproxime a un conjunto de datos.

Supongamos que tenemos un conjunto de m datos,

1.4. Least squared error method for polynomial data fitting

The interpolation methods we have described so far, try to find a polynomial of a function defined using polynomials that passes through a set of data. In the case of Least squared error fitting, the aim is to find the polynomial of a specific degree that better approximates a dataset.

Suppose we have a dataset holding m data,

x	x_1	x_2	$\cdots$	x_m
y	y_1	y_2	$\cdots$	y_m

Queremos construir un polinomio $p(x)$ de grado $n < m-1$, de modo que los valores que toma el polinomio para los datos $p(x_i)$ sean lo más cercanos posibles a los correspondientes valores y_i.

En primer lugar, necesitamos clarificar qué entendemos por *lo más cercano posible*. Una posibilidad, es medir la diferencia, $y_i - p(x_i)$ para cada par de datos del conjunto. Sin embargo, es más frecuente emplear el cuadrado de dicha diferencia, $(y_i - p(x_i))^2$. Esta cantidad tiene, entre otras, la ventaja de que su valor es siempre positivo con independencia de que la diferencia sea positiva o negativa. Además, representa el cuadrado de la distancia entre $p(x_i)$ e y_i. Podemos tomar la suma de dichas distancias al cuadrado, obtenidas por el polinomio para todos los pares de puntos,

We want to build a polynomial $p(x)$ with degree $n < m-1$ so that the values it takes for the data $p(x_i)$ are the nearest possible to the values y_i of the data table.

First, we must clarify what means *the nearest possible*. One option is to measure the difference $y_i - p(x_i)$ for each pair of data in the dataset. However, using the square of such difference $(y_i - p(x_i))^2$ is more common. This quantity has the advantage, among others, of being always positive, no matter whether the difference is positive or negative. Besides, it represents the square distance between $p(x_i)$ and y_i. We can take the sum of these differences for all pairs in the dataset,

$$\sum_{i=1}^{m} (y_i - p(x_i))^2$$

como una medida de la distancia del polinomio a los datos. De este modo, el polinomio *lo más cercano posible* a los datos sería aquel que minimice la suma de diferencias al cuadrado que acabamos de definir. De ahí el nombre del método.

En muchos casos, los datos a los que se pretende ajustar un polinomio por mínimos cuadrados son datos experimentales. En función del entorno experimental y del método con que se han adquirido los datos, puede resultar que algunos resulten más fiables que otros. En este caso, sería deseable hacer que el polinomio se aproxime más a los datos más fiables. Una forma de hacerlo es añadir unos *pesos*, ω_i, a las diferencias al cuadrado en función de la confianza que nos merece cada dato,

as a measure of the distance from the polynomial to the data. In this way, the *nearest possible to the data* would be the polynomial that minimised the sum of square differences we have just defined. Indeed, this is the origin of the method's name.

The data we often want to fit a polynomial using the least square method are experimental data. Depending on the environment and the method used for data acquisition, some data may be more reliable than others. In this case, it is interesting that the polynomial passes closest to the more reliable data. One way to achieve it is to add some *weights*, ω_i to the square differences according to the confidence that each data deserves,

$$\sum_{i=1}^{m} \omega_i \left(y_i - p(x_i)\right)^2$$

Los datos fiables se multiplican por valores de ω grandes y los poco fiables por valores pequeños.

Para ver cómo obtener los coeficientes de un polinomio de mínimos cuadrados, empezaremos con el caso más sencillo; un polinomio de grado 0. En este caso, el polinomio es una constante, definida por su término independiente $p(x) = a_0$. El objetivo a minimizar sería entonces,

We will multiply more reliable data by large values of ω and the less reliable by small values.

To see how to obtain the coefficients of a least square polynomial, we will start with the simplest case; a 0-degree polynomial. In this case the polynomial is a simple constant value, defined by its constant term $p(x) = a_0$. Then the fucntion to be minimised would be,

$$g(a_0) = \sum_{i=1}^{m} \omega_i \left(y_i - a_0\right)^2$$

El valor mínimo de esta función debe cumplir que su derivada primera $g'(a_0) = 0$ y que su derivada segunda $g''(a_0) \geq 0$,

The minimum of this function must meet that its first derivative $g'(a_0) = 0$ and its second derivative $g''(a_0) \geq 0$,

$$g'(a_0) = -2\sum_{i=1}^{m} \omega_i \left(y_i - a_0\right) = 0 \Rightarrow a_0 = \frac{\sum_{i=1}^{m} \omega_i \cdot y_i}{\sum_{i=1}^{m} \omega_i}$$

$$g''(a_0) = 2\sum_{i=1}^{m} \omega_i \Rightarrow g''(a_0) \geq 0$$

El resultado obtenido para el valor de a_0 es una media, ponderada con los pesos w_i de los datos. Si hacemos $w_i = 1\ \forall w_i$ obtendríamos exactamente la media de los datos. Este resul-

We obtain that the value for a_0 is a mean of the data, weighted by the values ω_i assigned to the data. If we take $w_i = 1\ \forall w_i$, we obtain the mean value of the data. This re-

tado resulta bastante razonable. Aproximar un conjunto de valores por un polinomio de grado cero, es tanto como suponer que la variable y permanece constante para cualquier valor de x. Las diferencias observadas deberían deberse entonces a errores aleatorios experimentales, y la mejor estima del valor de y será precisamente el valor medio de los valores disponibles. La figura 8.6 muestra el resultado de calcular el polinomio de mínimos cuadrados de grado cero para un conjunto de datos.

sult is quite reasonable. Approximating a set of values with a 0-degree polynomial is equivalent to considering that variable y remains constant for every value of x. In this case, we may attribute the observed differences among the values of the table to experimental random errors. Consequently, the best estimation of y would be the mean value of the available data.

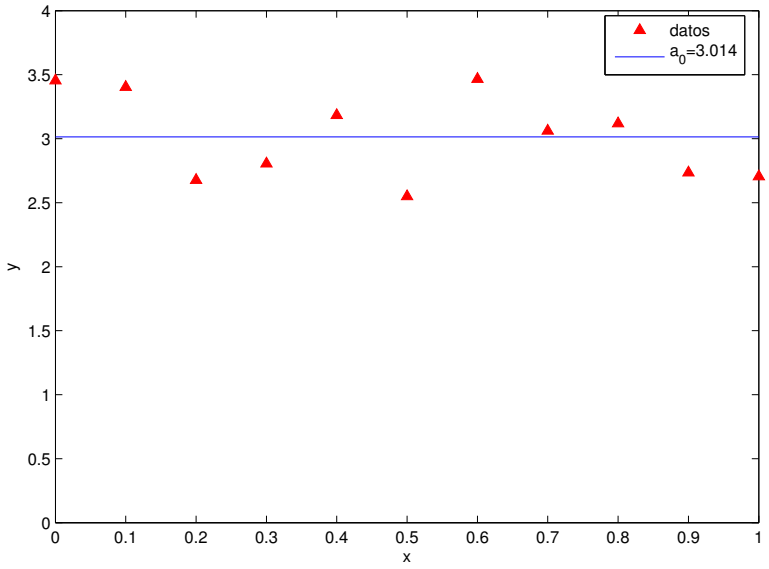


Figura 1.6: Polinomio de mínimos cuadrados de grado 0
Figure 1.6: 0-degree least squared error polynomial

El siguiente paso en dificultad sería tratar de aproximar un conjunto de datos por un polinomio de grado 1, es decir, por una linea recta, $p(x) = a_0 + a_1 x$. En este caso, la suma de diferencias al cuadrado toma la forma,

The nest step would be to approximate the data using a 1-degree polynomial, i.e., a straight line, $p(x) = a_0 + a_1 x$. Inn this case, the sum of squared differences takes the form,

$$g(a_0, a_1) = \sum_{i=1}^{m} \omega_i \left(y_i - a_0 - a_1 x_i\right)^2$$

En este caso, tenemos dos coeficientes sobre los que calcular el mínimo. Éste se obtiene cuando las derivadas parciales de $g(a_0, a_1)$ respecto a ambos coeficientes son iguales a cero.

We have now two coefficient for computing the minimum. We get the minimum making the function $g(a_0, a_1)$ partial derivatives equal to zero.

$$\frac{\partial g}{\partial a_0} = -2 \sum_{i=1}^{m} \omega_i (y_i - a_0 - a_1 x_i) = 0$$

$$\frac{\partial g}{\partial a_1} = -2 \sum_{i=1}^{m} \omega_i x_i (y_i - a_0 - a_1 x_i) = 0$$

Si reordenamos las ecuaciones anteriores,

And after rearranging the previous equations,

$$\left(\sum_{i=1}^{m}\omega_i\right)a_0+\left(\sum_{i=1}^{m}\omega_i x_i\right)a_1=\sum_{i=1}^{m}\omega_i y_i$$
$$\left(\sum_{i=1}^{m}\omega_i x_i\right)a_0+\left(\sum_{i=1}^{m}\omega_i x_i^2\right)a_1=\sum_{i=1}^{m}\omega_i x_i y_i$$

Obtenemos un sistema de dos ecuaciones lineales cuyas incógnitas son precisamente los coeficientes de la recta de mínimos cuadrados.

Podemos ahora generalizar el resultado para un polinomio de grado n, $p(x)=a_0+a_1x+a_2x^2+\cdots+a_nx^n$. La función g toma la forma,

We get a linear system with two equations and two unknowns, which are the coefficients of the least squares line.

We can now generalised this result for a n-degree polynomial $p(x)=a_0+a_1x+a_2x^2+\cdots+a_nx^n$. Function g would be now,

$$g(a_0,a_1,\cdots,a_n)=\sum_{i=1}^{m}\omega_i\left(a_0+a_1x_i+\cdots+a_nx_i^n-y_i\right)^2$$

De nuevo, para obtener los coeficientes del polinomio igualamos las derivadas parciales a cero,

Again, we can get the polynomial coefficents making the partial derivatives equal to zero,

$$\frac{\partial g(a_0,a_1,\cdots,a_n)}{\partial a_j}=0\Rightarrow\sum_{i=1}^{m}\omega_i x_i^j\left(a_0+a_1x_i+\cdots+a_nx_i^n-y_i\right)=0,\ j=0,1\cdots,n$$

Si reordenamos las expresiones anteriores, llegamos a un sistema de $n+1$ ecuaciones lineales, cuyas incógnitas son los coeficientes del polinomio de mínimos cuadrados,

If we rearrange the previous equation, we arrive to a linear system of $n+1$ equations whose unknowns are the coefficients of the least squares polynomial,

$$\begin{pmatrix}s_0 & s_1 & \cdots s_n\\ s_1 & s_2 & \cdots s_{n+1}\\ \vdots & \vdots & \ddots\\ s_n & s_{n+1} & \cdots s_{2n}\end{pmatrix}\cdot\begin{pmatrix}a_0\\ a_1\\ \vdots\\ a_n\end{pmatrix}=\begin{pmatrix}c_0\\ c_1\\ \vdots\\ c_n\end{pmatrix}$$

Donde hemos definido s_j y c_j como,

Where we have defined s_j and c_j as,

$$s_j=\sum_{i=1}^{m}\omega_i x_i^j$$
$$c_j=\sum_{i=1}^{m}\omega_i x_i^j y_i$$

El siguiente código permite obtener el polinomio de mínimos cuadrados que aproxima un conjunto de n datos,

h following code calculates the coefficients of the least squares polynomial that approximate a set of n data.

minimos_cuadrados.py

```python
# -*- coding: utf-8 -*-
"""
Created on Sat Aug  3 20:15:18 2024
Function to obtain the least squared error polynomial from a dataset
@author: abierto
"""
import numpy as np
def lse(x,y,n,w='none'):
    """
    This function compute the least squared error polynomial of
    degree n from a set of data x,y. Optionally it can use a
    set of weights to perform the computation
    Parameters
    ----------
    x : TYPE numpy array
        DESCRIPTION. x data to be fitted
    y : TYPE numpy array
        DESCRIPTION. y data to be fitted
    n : TYPE integer
        DESCRIPTION. Polynomial degree
    w : TYPE numpy array with the same length than x and y
        DESCRIPTION.

    Returns
    -------
    a: TYPE numpy array
       DESCRITION.  polynomial coefficients in increasing powers
           a[0]+a[1]x+a[2]x**2+...+a[n]x**n

    """
    m = x.shape[0]
    #first we check that we have enought points
    if m < n:
        raise Warning('the degree is greater than the number of data')
    #if there is no weight array generate an arry of ones
    if type(w)==str:
        w = np.ones(m)
    #we build the s elements of the system coefficient matrix
    n = n+1 # for degree n I need n+1 coeficientes
    s = np.zeros(2*n)
    for j in range(2*n):
        for i in range(m):
            s[j] = s[j] + w[i]*x[i]**j
    c = np.zeros(n)
    for j in range(n):
        for i in range(m):
            c[j] = c[j] + w[i]*x[i]**j*y[i]
    A = np.zeros([n,n])
    for i in range(n):
        for j in range(n):
            A[i,j] = s[i+j]
```

```
52      a =np.linalg.solve(A,c)
53      return(a)
54
```

Una última observación importante es que si intentamos calcular el polinomio de mínimos cuadrados de grado $m-1$ que aproxima un conjunto de m datos, lo que obtendremos será el polinomio de interpolación. En general, cuanto mayor sea el grado del polinomio más posibilidades hay de que la matriz de sistema empleado para obtener los coeficientes del polinomio esté mal condicionada.

1.4.1. Mínimos cuadrados en Python.

Hay varias formas de llevar a cabo un ajuste polinómico por mínimos cuadrados empleando Python. Solo vamos a describir uno de ellos. El subpaquete de `numpy`, `numpy.polynomial` incluye una clase llamada `Polynomial` (con mayúscula) permite, entre otras cosas, crear polinomios como objetos de programación. Una de las formas de crear un polinomio, es como resultado del ajuste por mínimos cuadrados a un conjunto de datos. Para ello, `Polynomial`, cuenta con la función `fit`. Este comando admite como entradas un vector de coordenadas x y otro de coordenadas y, de los datos que se quieren aproximar, y una tercera variable con el grado del polinomio. Opcionalmente se puede añadir un vector de pesos, si se quiere ponderar la importancia que damos a cada dato.

La funcion `Polynomial.fit` devuelve como resultado un 'objeto' polinomio. Este objeto contiene toda la información sobre el polinomio creado y puede usarse directamente como una función para calcular el valor del polinomio en un punto. El siguiente script de Python crea un par de vectores y muestra como manejar el comando,

A last important remark: If we try to compute the least squared error polynomial of degree $m-1$ to approximate a set of m data, we will get the interpolating polynomial. In general, if we increase the degree of the least squared polynomial, we increase the probability that the system matrix we use to obtain the polynomial coefficients will be poorly conditioned.

1.4.1. Least squares in Python.

There are several ways to compute the Least squares polynomial for a set of data using Python. We will describe only one of them. The `numpy` sub-package `numpy.polynomial` includes a class called `Polynomial` (Mean the first capital letter!) allows us, among many other things, to create polynomial as programming objects. One way to create a polynomial is to compute the least squares polynomial from a dataset. The class `Polynomial` has a special function `fit` to accomplish it. This command takes as inputs two arrays of data, one with the coordinates x and the other with the coordinates y of the value we want to interpolate, plus a third variable for the polynomial degree. Optionally, we can add an array of weights to balance the importance we give to each data.

Function `Polynomial.fit` returns a polynomial 'object'. This object contains all the information on the polynomial it represents and it can be use straightforwardly as a function to compute the value of the polynomial in any point. The following Python's script generates a pair of vectors and shows how to deal with the `fit` command.

ejemplo_ajuste.py

```
1  # -*- coding: utf-8 -*-
2  """
3  Created on Sun Aug  4 16:00:12 2024
4  Example of least squares fit using the numpy.polynomial class Polynomial
5  @author: abierto
```

```
"""
import numpy as np
from numpy.polynomial import Polynomial as py
from matplotlib import pyplot as pl
#we import the function included above to compare the result
#with the numpy.polynomial function
from minimos_cuadrados import lse

#define a data set to play with
x = np.array([ 1.,  2.,  3.,  4.,  5.,  6.,  7.,  8.,  9., 10.])
y = np.array([-1.8143451, 20.914356, 26.714303, 61.129501,\
              350.414728, 123.00032, 167.06809, 212.97832,\
              258.67911, 320.53422])

p3 = py.fit(x,y,3) #we are not including weight
#we get the polynomial coefficients just to see them
a = p3.convert().coef

#a set of point to evaluate the polynomial
xi = np.arange(1.,10.,0.01)

yi = p3(xi)

#point y[4] (see graphics bellow) apears to be an outlayer we can
#use weights to atenuate its influence
w = np.ones(x.shape[0])
w[4] = 0.5 #we reduce the weight of the outlayer
p3w = py.fit(x,y,3,w=w)
yiw = p3w(xi)

#####################################################################
#we repeate the calculation using our own funtion lse
#####################################################################
an = lse(x,y,3)
p = py(an) #we create a polynomial object using the coeficients
yin = p(xi)
#repeat the computing using the weight
aw = lse(x,y,3,w)
pw = py(aw)
yinw = pw(xi)

#drawing the results to compare
ax1 = pl.subplot(2,1,1)
pl.plot(x,y,'o')
pl.plot(xi,yi)
pl.plot(xi,yiw)
ax1.legend(['Data','Polynomial.fit',\
            'Polynomial.fit (weights)'])
ax2 = pl.subplot(2,1,2)
pl.plot(x,y,'o')
pl.plot(xi,yin)
pl.plot(xi,yinw)
```

```
58 ax2.legend(['Data','lse','lse (weights)'])
```

Vamos a revisar algunos puntos importantes de este código. en la línea 8, importamos el objeto `Polynomial`. Lo hacemos siguiendo el mismo método con que habríamos importado un submódulo cualquiera de `numpy`. Le asignamos el alias `py`. En la línea 12 hemos importado la función lse. Se trata de la función que hemos creado nosotros para realizar ajustes por mínimos cuadrados y cuyo código hemos incluido mas arriba. En las líneas 15 y 16 creamos un conjunto de datos.

En la línea 20 creamos un polinomio de minimos cuadrados de grado 3, `p3`, que ajusta los datos anteriores. `p3`, contiene toda la información del polynomio creado. En la línea 22, empleamos el comando `convert` para obtener un array con los coeficientes del polinomio, ordenados de menor a mayor grado. $p(x) = a_0 + a_1x + a_2x^2 + \cdots + a_nx^n$. En la línea 27 empleamos directamente `p3` como una función, pasándole los valores del array `xi`, para evaluar el polinomio en dichos puntos.

En las líneas 29-34 repetimos el cálculo, pero ahora empleando pesos. Damos a todos los pesos un valor 1 excepto para el quinto dato, al que damos un peso menor.

Las líneas 39 a 45 repiten los mismos cálculos, pero ahora empleando la función `lse`, creada por nosotros.

Por último, dibujamos los resultados para compararlos, tal y como se muestran en la figura 8.7

La figura muestra claramente el efecto de los pesos en la regresión. Al atenuar la influencia del quinto punto, que claramente parece fuera de rango, la curva se acerca más al resto de los puntos.

Si miramos con atención ambos gráficos, es fácil darse cuenta que la función de Python acerca más el polinomio a los puntos que la que hemos escrito nosotros. La razón está en que hemos usado un modo distinto de introducir los pesos. La función `fit`, aplica el el peso directamente a la diferencia entre el dato y el valor del polinomio $\omega_i|y_i - p(x_i)|$ mientras que nosotros aplicamos los pesos al cuadrado de dicha diferencia $\omega_i(y_i - p(x_i))^2$. Si compa-

We are going to review some important points of the previous code. In line 8, we import the object `Polynomial`. We do it in the same way that we import whatever `numpy` submodule. We have assigned to `Polynomial` the alias `py`. In line 12 we import the function `lse`. This is the function we previously created to perform least squares fits and whose code we included above. In lines 15 and 16 we create a dataset.

In line 20, we create a 3-degree least squares polynomial, `p3`, which fits the previous data. `p3` holds all the information on the polynomial we have created. In line 22, we use the command `convert` to obtain an array with the polynomial coefficients, ordered in ascending degree. $p(x) = a_0 + a_1x + a_2x^2 + \cdots + a_nx^n$. In line 27, we use `p3` straightforwardly, as a function, to obtain the values the polynomial takes at the points held by the array `x_i`.

We repeat the same computation in lines 29-34 but now use weights. We assign a value of 1 to every weight except for the fifth data point, to which we assign a smaller weight.

In lines 39 to 45 we repeat the the same computation but this time using our own function, `lse`

Lastly, we draw the results to compare, as can be seen in figure 8.7.

The figure clearly shows the effect the weights have on the regression results. When we attenuate the influence of the fifth point, which appears to be quite an outlier, the curve approaches more to the remaining points.

If we look at both graphics attentively, we will see that the Python function approximates the polynomial more to the points than the function we have written. The difference comes from how we introduce the weights, which are not the same as those used by Python. Function `fit`, applies the weights directly to the difference between the data an the value taken by the polynomial $\omega_i|y_i - p(x_i)|$, whike our function applies the weights to the squa-

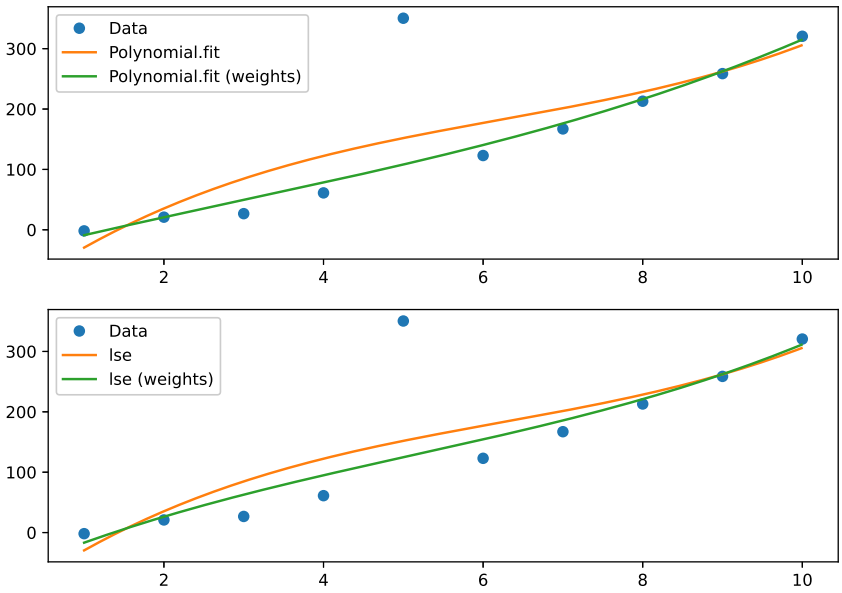


Figura 1.7: Ejemplo de cálculo de ajuste por mínimos cuadrados empleando la función fit de Polynomial y nuestra función lse.
Figure 1.7: An example of least squares polynomial fit, using the function fit from Polynomial and our function lse.

ramos lo coeficientes de los polinomios obtenidos,

red difference $\omega_i(y_i - p(x_i))^2$. If we compare the coefficients of the polynomials computed by both methods,

```
In [47]: p3w.convert().coef
Out[47]: array([-40.71774313,  33.22583298,  -1.61930914,   0.1854822 ])

In [48]: aw
Out[48]: array([-68.26797522,  56.50545799,  -5.31193931,   0.34603134])
```

vemos que no coinciden. Para obtener el mismo resultado, deberíamos introducir el cuadrado del valor de los pesos cuando usamos nuestra funcion,

We can see they are different. To get the same result, we must introduce the squared values of our weight when we use our function,

```
In [49]: aw2 = lse(x,y,3,w**2)

In [50]: aw2
Out[50]: array([-40.71774313,  33.22583298,  -1.61930914,   0.1854822 ])
```

Esto nos enseña una lección muy importante: no se deben emplear nunca paquetes de cálculo numérico sin saber qué están calculando exactamente.

1.4.2. Análisis de la bondad de un ajuste por mínimos cuadrados.

Supongamos que tenemos un conjunto de datos obtenidos como resultado de un experimento. En muchos casos la finalidad de un ajuste por mínimos cuadrados, es encontrar una ley que nos permita relacionar los datos de la variable independiente con la variable dependiente. Por ejemplo si aplicamos distintas fuerzas a un muelle y medimos la elongación sufrida por el muelle, esperamos obtener, en primera aproximación, una relación lineal: $\Delta x \propto F$. (Ley de Hooke).

Sin embargo, los resultados de un experimento rara vez se ajustan a una ley debido a errores aleatorios que no es posible corregir.

Cuando realizamos un ajuste por mínimos cuadrados de m datos, podemos emplear cualquier polinomio desde grado 0 hasta grado $m-1$. Desde el punto de vista del error cometido con respecto a los datos disponibles el mejor polinomio sería precisamente el de grado $m-1$ que da error cero para todos los datos, por tratarse del polinomio de interpolación. Sin embargo, si los datos son experimentales estamos incluyendo los errores experimentales en el ajuste.

Por ello, para datos experimentales y suponiendo que los datos solo contienen errores aleatorios, el mejor ajuste lo dará el polinomio de menor grado para el cual las diferencias entre los datos y el polinomio $y_i - p(x_i)$ se distribuyan aleatoriamente. Estas diferencias reciben habitualmente el nombre de residuos.

La figura 8.8 muestra un ejemplo de ajuste por mínimos cuadrados empleando cada vez un polinomio de mayor grado.

En la figura 8.8(a) se observa claramente que el ajusto no es bueno, la recta de mínimos cuadrados no es capaz de adaptarse a la forma que presentan los datos. Los residuos

This result teaches us a fundamental lesson: never use a numerical computing software package without understanding its operations.

1.4.2. Analyzing goodness of fit using least squares method.

Suppose we have a dataset we have obtained as a result of an experiment. In many cases, least squares data fitting aims to find a law that allows us to establish a mathematical relationship between the independent and dependent variables. For instance, if we apply different stresses to a spring and measure the elongation acquired by the spring, we expect to get a linear relationship between stress and elongation as a first approximation: $\Delta x \propto F$. (Hooke's law).

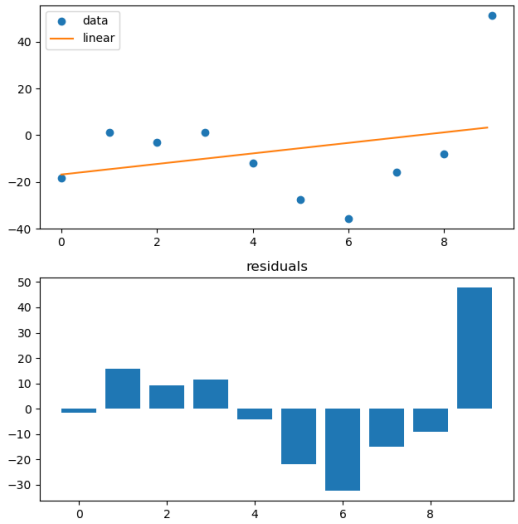
However, the results of an experiment hardly ever fit precisely into any law due to random errors, which we cannot amend.

When we carry out a least squares fitting, we may use any polynomial of degrees 0 to $m-1$. From the point of view of the error we make, considering only the available data, the best polynomial would be the $m-1$-degree polynomial because this is the interpolation polynomial, and, so, it makes a zero error for every data. Nevertheless, if we are trying to fit experimental data, we are including the experimental errors in the fitting.

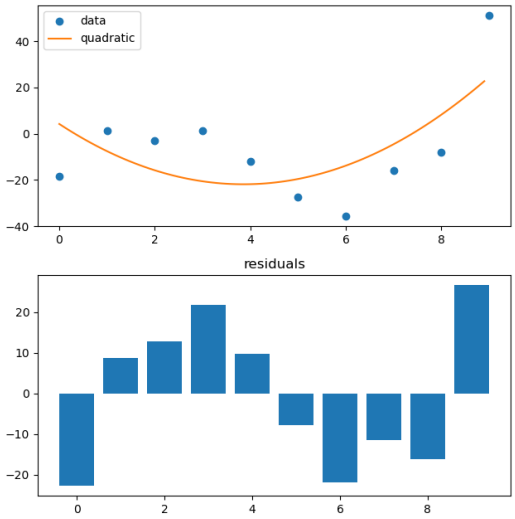
As a consequence, for experimental data and assuming that the data only are affected by random errors, we get the best fitting using the polynomial of lest degree for which the differences between data and polynomial value $y_i - p(x_i)$ are randomly distributed. Such differences are usually called residuals.

Figure 8.8 shows a least square fitting example, using polynomials of increasing degrees.

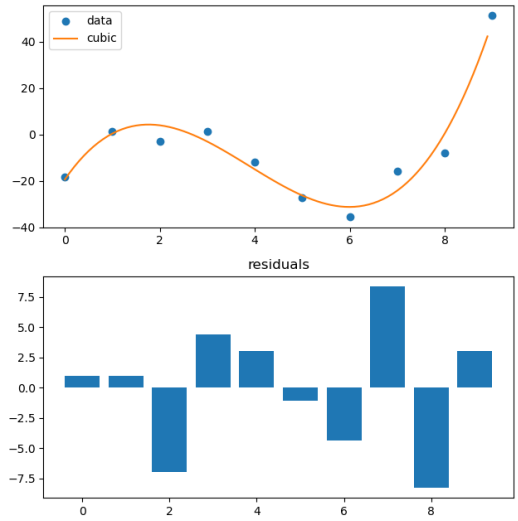
The residual also shows the data shape; they are not randomly distributed. In figure 8.8(b), we can see how a parabola better approximates the data but cannot follow the data up and down. The residuals clearly show



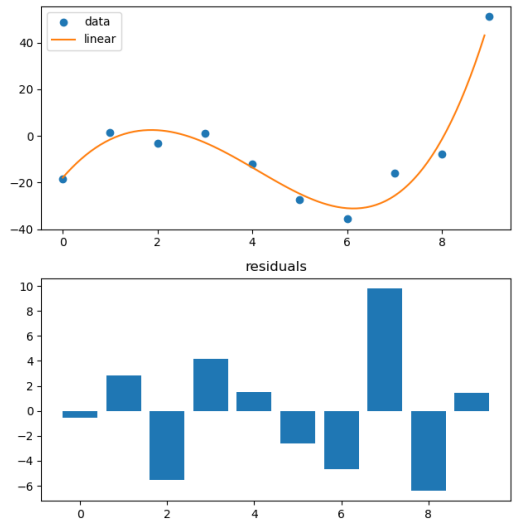
(a) Recta de mínimos cuadrados y residuos obtenidos / Least squares line and residuals.



(b) Parabola de mínimos cuadrados y residuos obtenidos / Least squares parabola and residuals.



(c) polinomio de tercer grado de mínimos cuadrados y residuos obtenidos / 3-degree polynomial and residuals



(d) polinomio de cuarto grado de mínimos cuadrados y residuos obtenidos / 4-degree polynomial and residuals

Figura 1.8: Comparación entre los residuos obtenidos para los ajustes de mínimos cuadrados de un conjunto de datos empleando polinomios de grados 1 a 4.

Figure 1.8: Comparison among the residuals resulting from a dataset least squares fitting, using polynomials of degrees 1 to 4.

muestran claramente esta tendencia: no están distribuidos de forma aleatoria. En la figura 8.8(b), la parábola aproxima mejor el conjunto de datos. Sin embargo, los residuos presentan un máximo y un mínimo suaves, no son realmente aleatorio. En la figura 8.8(c), los residuos están distribuidos de forma aleatoria. Si comparamos estos resultados con los de la figura 8.8(d) vemos que en este último caso los residuos son más pequeños, pero conservan esencialmente la misma distribución aleatoria que en la figura anterior. La aproximación de los datos empleando un polinomio de cuarto grado no añade información sobre la forma de la función que siguen los datos, y ha empezado a incluir en el ajuste los errores de los datos.

a smooth maximum and minimum, but they are not actually random. In figure 8.8(c), the polynomial fits better the data, and the residuals are randomly distributed. If we compare them with the residuals presented in figure 8.8(d), we see these last are smaller but present essentially the same random distribution. We may conclude that the data approximation using a fourth-degree polynomial does not add any useful information on the function shape the data follow, and, worse than this, we are starting to include the data errors in the fitting.

$\Xi_i(t)$

1.5. Curvas de Bézier

Las curvas de Bézier permiten unir puntos mediante curvas de forma arbitraria. Una curva de Bézier viene definida por un conjunto de $n+1$ puntos, $\{\vec{p}_0, \vec{p}_2, \cdots, \vec{p}_n\}$, que reciben el nombre de puntos de control. El orden en que se presentan los puntos de control es importante para la definición de la curva de Bézier correspondiente. Ésta pasa siempre por los puntos $\vec{p_0}$ y $\vec{p_n}$. El resto de los puntos de control permiten dar forma a la curva, que tiene siempre la propiedad de ser tangente en el punto $\vec{p}_0$ a la recta que une $\vec{p}_0$ con $\vec{p}_1$ y tangente en el punto $\vec{p}_n$ a la recta que une $\vec{p}_{n-1}$ con $\vec{p}_n$.

Para definir la curva, se asocia a cada punto de control $\vec{p}_i$ una función conocida con el nombre de función de fusión. En el caso de las curvas de Bézier, las funciones de fusión empleadas son los polinomios de Bernstein,

1.5. Bézier Curves

Bezier's curves can connect points using curves with arbitrary shape. We define a Bezier's curve using a set of of $n+1$ points, $\{\vec{p}_0, \vec{p}_2, \cdots, \vec{p}_n\}$ which are known as control points. The order of the control points play a key role in the definition of the corresponding Bézier's curve. The curve pass through the end points $\vec{p}_0$ and $\vec{p}_n$. The remaining control points allow us to define the curve shape, which has the interesting properties of being always tangent in point $\vec{p}_0$ to the line that connects $\vec{p}_0$ with $\vec{p}_1$ and tangent in point $\vec{p}_n$ to the line that connect $\vec{p}_{n-1}$ with $\vec{p}_n$.

To define the curve, we associate to the control point $\vec{p}_i$, a function known as fusion function. In the case of Bezier's curves, we use as fusion function Berstein's polynomials,

$$B_i^n(t) = \binom{n}{i} (1-t)^{n-i}\, t^i, \quad i = 0, 1, \ldots, n$$

El grado de los polinomios de Bernstein empleados depende del número de puntos de control; para un conjunto de $n+1$ puntos, los

The Berstein's polynomials degree we use depends on the number of control points; for a set of $n+1$, we need polynomials of degree n.

polinomios son de grado n. La variable t es un parámetro que varía entre 0 y 1.

La ecuación de la curva de Bézier que utiliza un conjunto de $n+1$ puntos de control, $\{\vec{p}_0, \vec{p}_2, \cdots, \vec{p}_n\}$, se define a partir de los polinomios de Bernstein como,

Variable t is a parameter which varies between 0 and 1.

The Bezier's curve for a set of $n+1$ control points, $\{\vec{p}_0, \vec{p}_2, \cdots, \vec{p}_n\}$, can be defined using Berstein's polynomials, according to the following equation,

$$\vec{p}(t) = \sum_{i=0}^{n} B_i^n(t) \cdot \vec{p}_i = \sum_{i=0}^{n} \binom{n}{i} (1-t)^{n-i} t^i \cdot \vec{p}_i$$

La expresión anterior da como resultado $\vec{p}_0$ si hacemos $t = 0$ y $\vec{p}_n$ si hacemos $t = 1$. para los valores comprendidos entre $t = 0$ y $t = 1$, los puntos $\vec{p}(t)$ trazarán una curva entre $\vec{p}_0$ y $\vec{p}_n$.

Veamos un ejemplo sencillo. Supongamos que queremos unir los puntos $\vec{p}_0 = (0,1)$ y $\vec{p}_n = (3,1)$ mediante curvas de Bézier. Si no añadimos ningún punto más de control, $\{\vec{p}_0 = (0,1), \vec{p}_1 = (3,1)\}$, la curva de Bézier que obtendremos será una recta que unirá los dos puntos,

The above expression yields $\vec{p}_0$ as a result, if we take $t = 0$ and $\vec{p}_n$ for $t = 1$. For values of t between $t = 0$ and $t = 1$, points $\vec{p}(t)$ will describe a curve between $\vec{p}_0$ y $\vec{p}_n$.

Let's see a basic example. Suppose we want to connect point $\vec{p}_0 = (0,1)$ with point $\vec{p}_1$ using a Bezier's curve. If we do not define any other control point, $\{\vec{p}_0 = (0,1), \vec{p}_1 = (3,1)\}$, The Bezier's curve we will obtain will be a line connection both points.

$$\vec{p}(t) = \binom{1}{0} (1-t)^1 t^0 \cdot (0,1) + \binom{1}{1} (1-t)^0 t^1 \cdot (3,1) = (1-t) \cdot (0,1) + t \cdot (3,1)$$

Si separamos las componentes x e y del vector $\vec{p}(t)$,

If we split the components x and y of vector $\vec{p}(t)$,

$$x = 3t$$
$$y = 1$$

Se trata de la ecuación del segmento horizontal que une los puntos $\vec{p}_0 = (0,1)$ y $\vec{p}_n = (3,1)$

Si añadimos un punto más de control, por ejemplo: $\vec{p}_1 = (1,2) \rightarrow \{\vec{p}_0 = (0,1), \vec{p}_1 = (1,2), \vec{p}_2 = (3,1)\}$, la curva resultante será ahora un segmento de un polinomio de segundo grado en la variable t,

It is the equation of the horizontal segment that joints the points $\vec{p}_0 = (0,1)$ y $\vec{p}_n = (3,1)$

If we add a control point more, for instance: $\vec{p}_1 = (1,2) \rightarrow \{\vec{p}_0 = (0,1), \vec{p}_1 = (1,2), \vec{p}_2 = (3,1)\}$, the resulting curve will be now a segment of a second degree polynomial on the variable t,

$$\vec{p}(t) = \binom{2}{0} (1-t)^2 t^0 \cdot (0,1) + \binom{2}{1} (1-t) t \cdot (1,2) + \binom{2}{2} (1-t)^0 t^2 \cdot (3,1) =$$
$$= (1-t)^2 \cdot (0,1) + 2(1-t) t \cdot (1,2) + t^2 \cdot (3,1)$$

Según vamos aumentando el número de los puntos de control, iremos empleando polinomios de mayor grado. El segmento de polinomio empleado en cada caso para unir los

As we increase the number of control points, we also increase de degree of the polynomial used to connect the first and last control points. The shape of this polynomial will change de-

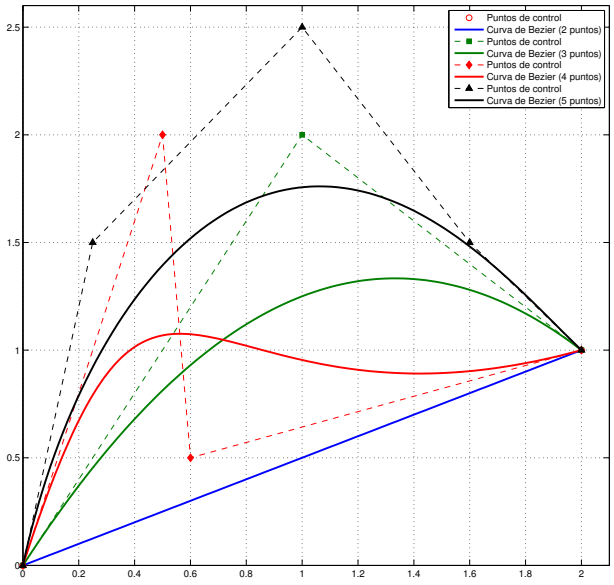


Figura 1.9: Curvas de Bézier trazadas entre los puntos $P_0 = (0, 0)$ y $P_n = (2, 1)$, variando el número y posición de los puntos de control.
Figure 1.9: Bézier curves traced between points $P_0 = (0, 0)$ y $P_n = (2, 1)$, variating the control points number and positions

puntos de control inicial y final variará dependiendo de los puntos de control intermedios empleados.

La figura 8.9 muestra un ejemplo en el que se han construido varias curvas de Bézier sobre los mismos puntos de paso inicial y final. Es fácil observar cómo la forma de la curva depende del número y la posición de los puntos de control. Si unimos éstos por orden mediante segmentos rectos, obtenemos un polígono que recibe el nombre de polígono de control. En la figura 8.9 se han representado los polígonos de control mediante líneas de puntos.

El siguiente código permite calcular y dibujar una curva de Bézier a partir de sus puntos de control, empleando las funciones `bezier` y `bezier_sc`

pending on the intermediate control point we use.

Figure 8.9 shows an example in which we have built several Bézier's curves using the same initial and final controls points. Notice how the shape of the curve depends on the number and position of the control points. If we orderly connect the control points using straight segments we get a polygon know as control polygon. In figure 8.9 the control polygons have been represented using dashed lines.

The following code includes functions `bezier` and `bezier_sc` to compute and draw Bezier's curves define from their control points.

bezier.py

```
1 # -*- coding: utf-8 -*-
2 """
3 Created on Thu Aug  8 19:41:24 2024
```

```
4   This module defines a functions to generate Bezier Polynomials
5   bezier returns the value of the curve as a function of t
6   bezier_rc uses function bezier to obtain the values of the curve
7   y an array of point a draw the result
8   @author: abierto
9   """
10  import numpy as np
11  import matplotlib.pyplot as pl
12
13
14  def bezier(p,t):
15      """
16      Calculates de value of a bezier curve with control points
17      in the array p at value t
18
19      Parameters
20      ----------
21      p : TYPE numpy array
22          DESCRIPTION. Numpy array of dimension 2Xn where n is the
23          number of control points. first row contains x coordinate
24          and second row contain y coordinate, the curve pass through
25          the first a last control points
26      t : TYPE
27          DESCRIPTION. Berstein's polynomial parameter t belongs
28          to [0,1]
29
30      Returns
31      -------
32      ber : TYPE np.array
33            DESCRIPTION. ber[0] x coordinate of the bezier's curve
34            at t. ber[1] y coordinate of the bezier's curve at t
35
36      """
37      n = p.shape[1]
38      num = np.arange(1,n).prod()
39      #we firt calculate all factorial needed
40      ftal = np.insert(np.arange(1.,n).cumprod(),0,1)
41      ber = np.zeros(2)
42      for i in range(n):
43          f = num/ftal[i]/ftal[n-1-i]
44          ber = ber + f*p[:,i]*(1-t)**(n-1-i)*t**i
45      return(ber)
46
47  def bezier_sc(p,t):
48      """
49      This function use funtion bezier to reproduce the bezier curve
50      with control point containing in p.
51      Parameters
52      ----------
53      p : TYPE numpy array
54          DESCRIPTION. Numpy array of dimension 2Xn where n is the
55          number of control points. first row contains x coordinate
```

```
56          and second row contain y coordinate, the curve pass through
57          the first a last control points
58      t : TYPE numpy array
59          DESCRIPTION. It should be an array of ordered values for
60          parameter t. To reproduce de whole Bezier's curve use
61          t = numpy.arange(0,1+step,step) the smaller step the smoother
62          the result
63
64      Returns
65      -------
66      pt : TYPE numpy array
67           DESCRIPTION. a 2xm array with m the number of data in t
68           first row containts the x coordinates at the values of t
69           and second row the y coordinates at the values of t
70
71      """
72      pt = np.array([bezier(p,i) for i in t]).T
73      pl.plot(p[0],p[1],'or')
74      pl.plot(p[0],p[1],'-.')
75      pl.plot(pt[0],pt[1])
76      return(pt)
77
78  def bzeq(p,n,step):
79      """
80      Computes Bezier's curves of higher degree, equivalent to a
81      given Bezier's curve
82
83      Parameters
84      ----------
85      p : TYPE numpy array
86          DESCRIPTION. 2xn array containing the control points of
87          the Bezier's curve to be derivate. (each column a point)
88      n : TYPE integer
89          DESCRIPTION. Degree of the equivalent Bezier's, it should
90          be greater than the number of points in p.
91      step : TYPE double
92          DESCRIPTION. a step to generate an ordered,  array which
93          equispaciate values in [0,1].
94
95      Returns
96      -------
97      p : TYPE numpy array
98          DESCRIPTION. 2x(n-1) array containing the control points
99          of the equivalent Bezier's curve
100
101     """
102     c = p.shape[1]
103     t = np.arange(0,1+step,step)
104     bezier_sc(p, t) #draw the Bz curve
105     if c < n:
106         #adding one more control point and recalculating
107         # the control points
```

```
108         p = np.append(p,[[0],[0]],axis=1)
109         pp =p.copy()
110         for i in range(1,c+1):
111             pp[:,i] = p[:,i-1]*i/(c) + (1 -i/(c))*p[:,i]
112         #the function calls itself till the number of points
113         #equals n
114         p = bzeq(pp,n,step)
115     # and it returns the control points. In the meanwhile it
116     #has drawn all the equivalent curves with degrees between
117     # c and n
118     return(p)
119 
120 def dbez(p):
121     """
122     Compute a Bezier's . Curve derivative, defined by control
123     points in p. It also draw de odograph and the derivatives
124     at some selected points and the Bezier's curve and
125     the derivative, as tangents vector, as some selected points.
126 
127     Parameters
128     ----------
129     p : TYPE numpy array
130         DESCRIPTION. Control points of the Bezier's curve we
131         wanna derivate.
132 
133     Returns
134     -------
135     d : TYPE numpy array
136         DESCRIPTION. Controls points of the derivative.
137 
138     """
139     grado = p.shape[1]-1
140     d = np.zeros([2,grado])
141     #compute the control points of the derivativa
142     for i in range(grado):
143         d[:,i] = (grado-1)*(p[:,i+1]-p[:,i])
144     t = np.arange(0,1+0.01,0.01)
145     #plot de odograph
146     pl.figure(1)
147     v = bezier_sc(d, t)
148     pl.quiver(np.zeros(12),np.zeros(12),\
149               v[0,::9],v[1,::9],\
150               angles='xy', scale_units='xy', scale=1)
151 
152     #plot the curve and its derivative. (the escale of the
153     #vector is arbitrary)
154     pl.figure(2)
155     ptos = bezier_sc(p,t)
156     pl.quiver(ptos[0,::9],ptos[1,::9],\
157               v[0,::9],v[1,::9])
158     pl.axis('equal')
159     return(d)
```

Curvas equivalentes de grado superior. Dada una curva de Bézier, representada por un polinomio de Bernstein de grado n, es posible encontrar polinomios de grado superior que representan la misma curva de Bézier. Lógicamente esto supone un aumento del número de puntos de control.

Supongamos que tenemos una curva de Bézier con cuatro puntos de control,

Upper degree equivalent curves. Given a Bezier's curve, represented by a Berstein's polynomial of degree n, it is possible to find Berstein's polynomials of higher degree, which also describes this same Beezier's curve. Indeed, this means an increase in the number of the control points.

Suposse we have a Bezier's curve defined by four control points,

$$\vec{p}(t) = \vec{p}_0 B_0^3(t) + \vec{p}_1 B_1^3(t) + \vec{p}_2 B_2^3(t) + \vec{p}_3 B_3^3(t)$$

Si multiplicamos este polinomio por $t + (1-t) \equiv 1$, el polinomio no varía y por tanto representa la misma curva de Bézier. Sin embargo, tendríamos ahora un polinomio un grado superior al inicial. Si después de la multiplicación agrupamos términos de la forma $(1-t)^{n-1}t^i$, Podríamos obtener el valor de los nuevos puntos de control,

If we multiply this polynomial by $t + (1-t) \equiv 1$, the polynomial remains the same an so it represent the same Bezier's curve too. However, we have now a polynomial one degree higher than the initial one. If after multiplying we group terms like $(1-t)^{n-1}t^i$, we can compute the value of the new control points,

$$\begin{aligned}
\vec{p}_0^{\,+} &= \vec{p}_0 \\
\vec{p}_1^{\,+} &= \frac{1}{4}\vec{p}_0 + \frac{3}{4}\vec{p}_1 \\
\vec{p}_2^{\,+} &= \frac{2}{4}\vec{p}_2 + \frac{2}{4}\vec{p}_3 \\
\vec{p}_3^{\,+} &= \frac{3}{4}\vec{p}_2 + \frac{1}{4}\vec{p}_3 \\
\vec{p}_4^{\,+} &= \vec{p}_3
\end{aligned}$$

y, en general, los puntos de control de la curva de Bézier de grado $n+1$ equivalente a una dada de grado n puede obtenerse como,

And, in general, we can compute the control points for the Bezier's curve of degree $n+1$ equivalent to a previous defined curve of degree n as,

$$\vec{p}_i^{\,+} = \alpha_i \vec{p}_{i-1} + (1-\alpha_i)\vec{p}_i, \qquad \alpha_i = \frac{i}{n+1}$$

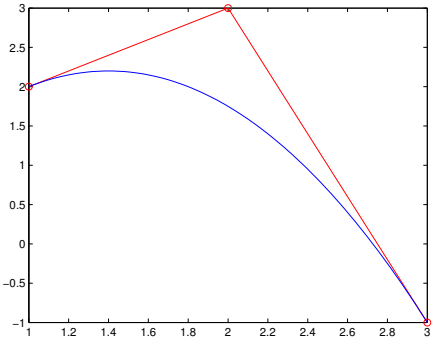
Mediante la ecuación anterior, es posible obtener iterativamente los puntos de control de la curva de Bézier equivalente a una dada de cualquier grado. Una propiedad interesante es que según aumentamos el número de puntos de control empleados, estos y el polígono de control correspondiente, van convergiendo a la curva de Bézier.

Using the above equation, we can iteratively obtain the control points of a Bezier's curve equivalent to any other given of whatever degree. An interesting property is that, as we increase the number of control points, these and the control polygon tend to converge to the Bezier's curve.

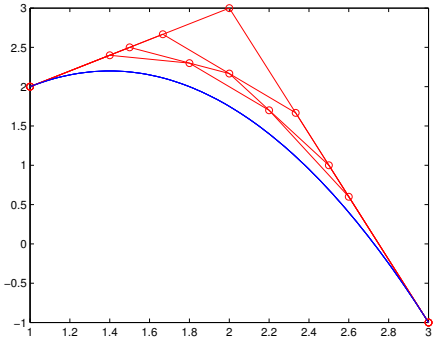
Figure 8.10 shows an example using a Bézier's

La figura 8.10 muestra un ejemplo para una curva de Bézier construida a partir de tres puntos de control. Es fácil ver cómo a pesar de aumentar el número de puntos de control, la curva resultante es siempre la misma. También es fácil ver en la figura 8.10(d) la convergencia de los polígonos de control hacia la curva.

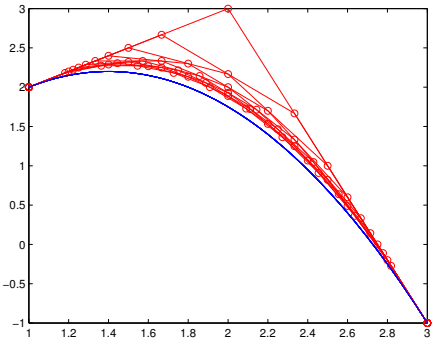
curve built from three control points. As can be seen, we always obtain the same curve, although we have increased the number of control points. It is also easy to see in figure 8.10(d) that the control polygons also converge to the Bezier's curve.



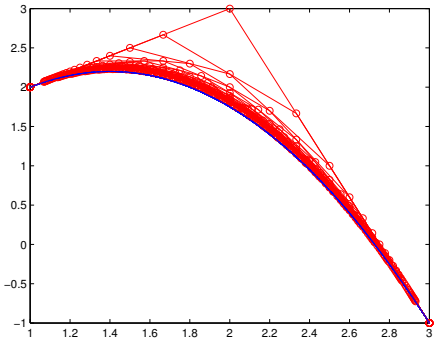
(a) Curva original (3 puntos) / Original curve (3 points)



(b) Curvas equivalentes de 4 a 6 puntos / 4 to 6 points equivalent curves



(c) Curvas equivalentes de 4 a 12 puntos / 4 to 12 points Equivalent curves



(d) Curvas equivalentes de de 4 a 30 puntos / 4 to 30 points equivalent curve

Figura 1.10: Curvas de Bézier equivalentes, construidas a partir de una curva con tres puntos de control.
Figure 1.10: Equivalent curves, built from a three control point Bézier's curve.

La función `bzeq` del módulo `bezier.py`, incluido más arriba permite calcular la curva equivalente de Bézier a una dada, para cualquier número de puntos de control que se desee.

Function `bzeq` in `bezier.py` module above include, allows us to calculate the Bezier's curve equivalent to any other given, using any number of control points.

Derivadas. Las derivadas de una curva de Bézier con respecto al parámetro t son particularmente fáciles de obtener a partir de los puntos de control. Si tenemos una curva de Bézier de grado n, definida mediante puntos de control $\vec{p_i}$. Su derivada primera con respecto a t será una curva de Bézier de grado $n-1$, cuyos puntos de control $\vec{d_i}$ puede obtenerse como:

Derivatives. To compute the derivatives of a Bezier's curve with respect to parameter t is particularly easy, using the control points. Suppose we have a degree n Bezier's curve, defined from control points $\vec{p_i}$. It first derivative with respect to t will be a Bezier's curve of degree $n-1$, whose control point we can compute as:

$$\vec{d_i} = n\,(\vec{p_{i+1}} - \vec{p_i})$$

La nueva curva de Bézier obtenida de este modo, es una hodógrafa; representa el extremo de un vector tangente en cada punto a la curva de Bézier original y guarda una relación directa con la velocidad a la que se recorrería dicha curva.

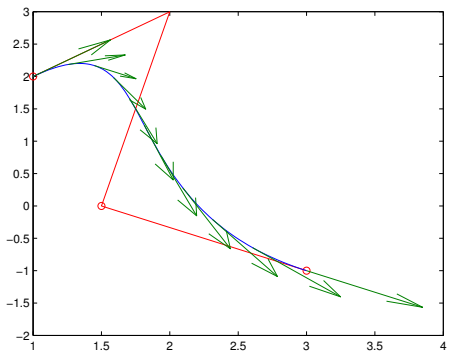
La figura 8.11(a), muestra una curva de Bézier sobre la que se ha trazado el vector derivada para algunos puntos. La figura 8.11(b) muestra la hodógrafa correspondiente y de nuevo los mismos vectores derivada de la figura 8.11(a)

La función `dbez`, incluida en el módulo `bezier.py` permite calcular los puntos de control de la derivada de una curva de Bézier.

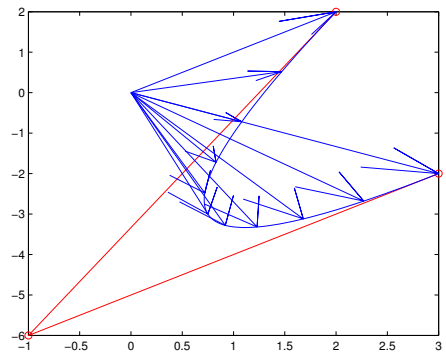
The new Bézier curve we obtain is a hodograph. It represents the position of the tips of vectors tangent in each point to the original Bézier curve and has a direct relationship with the speed at which you can travel across the curve at any point.

Figure 8.11(a) shows a Bezier's curve an the derivative vector at some points of the curve. Figure 8.11(b) show the hodograph corresponding to and the same derivative vectors of de curve presented in figure 8.11(a)

Function `dbez`, belonging to the module `bezier.py` included above, allows us to compute the derivative of any Bezier's curve.



(a) Curva de Bézier (4 puntos) / Bézier's curve (4 points)



(b) Derivada (hodógrafa) / Derivative (hodograph)

Figura 1.11: Curva de Bézier y su derivada con respecto al parámetro del polinomio de Bernstein que la define: $t \in [0, 1]$

Figure 1.11: Bézier's curve and its derivative with respect to the Berstein's polynomial parameter which defines the curve $t \in [0, 1]$

Interpolación con curvas de Bézier Podemos emplear curvas de Bézier para interpolar un conjunto de puntos $\{\vec{p}_0, \cdots \vec{p}_m\}$. Si empleamos un curva para interpolar cada par de puntos, $\vec{p}_i, \vec{p}_{i+1},\ i = 1, \cdots m-1$ tenemos asegurada la continuidad en los puntos interpolados puesto que las curvas tienen que pasar por ellos. Como en el caso de la interpolación mediante splines, podemos imponer continuidad en las derivadas para conseguir una curva de interpolación suave. En el caso de las curvas de Bézier esto es particularmente simple. Si llamamos B a la curva de Bézier de grado n construida entre los puntos $\vec{p}_{i-1}, \vec{p}_i$ con puntos de control, $\vec{p}_{i-1}, b_1, \vec{b}_2, \cdots, \vec{b}_{n-1}, \vec{p}_i$, y C a la curva de Bézier de grado s construida entre los puntos $\vec{p}_i, \vec{p}_{i+1}$ con puntos de control, $\vec{p}_i, \vec{c}_1, \vec{c}_2, \cdots, \vec{c}_{s-1}, \vec{p}_{i+1}$. Para asegurar la continuidad en la primera derivada en el punto $\vec{p}_i$ basta imponer,

Interpolation using Bézier's curves We can use Bézier's curve for interpolating a set of points, $\{\vec{p}_0, \cdots \vec{p}_m\}$. If we use a curve to interpolate every pair of consecutive points, $\vec{p}_i, \vec{p}_{i+1},\ i = 1, \cdots m-1$ we assure the continuity in the interpolated points because the curves have necessarily to pass through them. As in the case of spline interpolation, we can also impose continuity to the derivatives to get a smooth interpolating curve. In the case of Bèzier's curves, this is especially simple. If we call B to the n-degree Bezier's curve built between the points $\vec{p}_{i-1}, \vec{p}_i$ with control points, $\vec{p}_{i-1}, b_1, \vec{b}_2, \cdots, \vec{b}_{n-1}, \vec{p}_i$, and C to the s-degree Bezier's curve built between the points $\vec{p}_i, \vec{p}_{i+1}$ with control points, $\vec{p}_i, \vec{c}_1, \vec{c}_2, \cdots, \vec{c}_{s-1}, \vec{p}_{i+1}$; we can assure the continuity of the first derivative in the point $\vec{p}_i$ just imposing that,

$$n \cdot \left(\vec{p}_i - \vec{b}_{n-1}\right) = s \cdot (\vec{c}_1 - \vec{p}_i)$$

Esta condición impone una relación entre el penúltimo punto de control de la curva B y el segundo punto de control de la curva C. Pero deja completa libertad sobre el resto de los puntos de control elegidos para construir las curvas.

Podemos, por ejemplo, elegir libremente todos los puntos de control de la curva B y obtener a partir de ella el punto $\vec{c}_1$,

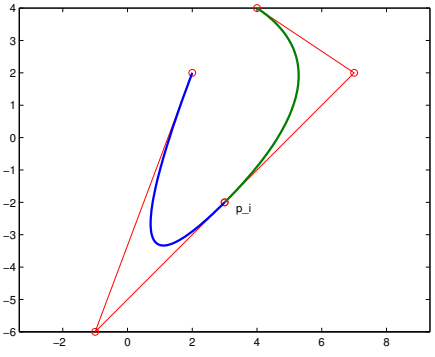
This condition imposes a link between the second-last control point of the curve B and the second control point of the curve C. However, it allows free election of the remaining control points to build the curves as desired.

For instance we can freely choose all control points of curve B and use it to compute the point $\vec{c}_1$,

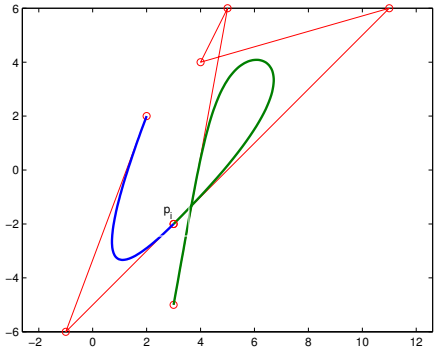
$$\vec{c}_1 = \frac{n+s}{s}\vec{p}_i - \frac{n}{s}\vec{b}_{n-1}$$

La figura 8.12 muestra un ejemplo de interpolación en la que se ha aplicado la condición de continuidad en la derivada que acabamos de describir.

Figure 8.12 shoes an example of interpolation in which we have applied the derivative continuity condition we have described.



(a) Interpolación mediante curvas de Bézier de 3 puntos / Interpolation using three control points Bezier's curves)



(b) Interpolación mediante curvas de Bézier de 3 y 4 puntos / iNterpolation using 3 and 4 control points Beziers's curves

Figura 1.12: Interpolación de tres puntos mediante dos curvas de Bézier
Figure 1.12: interpolating three point using two Bezier's curves

1.6. ejercicios

1. Carga en python los datos del fichero `datos.txt`[1] y realiza las siguientes tareas:

   a) Crea una función que a partir de dos vectores de datos x, y de igual longitud $n+1$, calcula la matriz de Vandermonde necesaria para obtener el polinomio de interpolación asociado a los puntos. Empleando la primera columna de datos contenida en `datos.txt` como datos x y la segunda como datos y, genera el polinomio de interpolación,

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

   Calcula el valor que toma el polinomio de interpolación en 100 puntos equiespaciados entre los valores x_0 y x_n de los datos del fichero. Dibuja en una misma gráfica los resultados obtenidos, empleando una línea continua, y los valores del fichero, mediante puntos separados

[1] disponible en https://github.com/UCM-237/LCC/tree/master/datos

1.6. exercises

1. Load in Python the date held in file `datos.txt`[1] an carry out the following tasks:

   a) Build a function that taking two data vector, x, y of the same length, $n+1$, computes the Vandermonde's matrix needed for obtaining the interpolating polynomial associated to the points. Using the first data column in `data.txt` as x and the second column as y compute the the interpolating polynomial,

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

   Compute the values of the interpolating polynomial in 100 equispaced points between values x_0 and x_n of the data file. In the same graphic, draw the results obtained using a continuous line and the file data as single points, using the symbol

[1] available in https://github.com/UCM-237/LCC/tree/master/datos

empleando el símbolo que prefieras. Comprueba que el polinomio pasa por los puntos contenidos en el fichero.

b) Reproduce en python la función 'Lagrange' de la sección 8.2.2 para calcular el polinomio interpolador de Lagrange. Emplea la función que acabas de crear para recalcular los valores del polinomio de interpolación realizado en el ejercicio anterior y comprueba que los resultados obtenidos son los mismos que empleando la matriz de Vandermonde.

c) A partir de los ejemplos de la sección sección 8.2.3, crea un programa que calcule los coeficientes del polinomio interpolador de diferencias divididas a partir de dos vectores de datos x, y de igual longitud $n+1$ y un segundo programa que calcule el valor del polinomio en un punto cualquiera a partir de los coeficientes obtenidos con el primer programa. Vuelve a calcular, empelando ahora el polinomio de diferencias dividas, los valores del polinomio de interpolación sobre los mismos datos empleados en los ejercicios anteriores y comprueba que da los mismos resultados.

d) Por último, crea una función que calcule el polinomio de interpolación de Newton-Gregory (sección 8.2.4). ¿Es posible usarlos para interpolar los datos del archivo `datos.txt`? Si la respuesta el afirmativa, repite el cálculo del polinomio de interpolación, empleando Newton-Gregory y comprueba si coincide con lo obtenido en los ejercicios anteriores.

e) Usa la función de numpy `interp1d` y emplea de nuevo los datos del fichero `datos.txt`, para obtener el resultado de interpolar los valores en 100 puntos equiespaciados entre los valores x_0 y x_n del fichero

you prefer. Check that the polynomial passes through the data point included in the file.

b) Write the 'Lagrange' function for computing the Interpolating Lagrange polynomial in Python, included in section 8.2.2. Use the function you just created to recompute the values of the interpolating polynomial obtained in the previous exercise and check that you get the same results as using Vandermonde's matrix.

c) Departing from the examples in section 8.2.3, build a program for computing the divided differences polynomial coefficients, usin two data arrays x and y with the same length, $n+1$. Buid also a second program for computing the value of the polynomial in any point, using the coefficients computed with the previous program. Recompute one more time, now using the divided differences polynomial the values of the interpolation polynomial, of the same data use in previous exercises. Check that you get the same results.

d) Lastly, build a function that compute the Newton-Gregory's interpolating polynomials (section 8.2.4). It is possible to interpolate the data held in the file `datos.txt` using the Newton-Gregory's polynomial? If the answer is 'yes', use your function just writen to repeat the computing of the intepolation polynomial carried out in previous section and check that the results meet those obtained in previous sections.

e) Using the Numpy's function `interp1d` and the data contained in file `datos.txt`, compute the result of interpolating in 100 equispaced points between values x_0 and x_n of the file dataset. Do it using the method `'nearest'` and `'linear'`. Repeat the computing but now using

ro. Emplea para ello los métodos `'nearest'` y `'linear'`. Repite el cálculo empleando ahora la función de numpy `'CubicSpline'`. Dibuja los resultados en la misma gráfica empleada en el ejercicio 1*a*)

2. Construye a partir del código de ejemplo de la sección 8.4, una función que calcule el polinomio de grado n que ajusta por mínimos cuadrados un conjunto de pares de datos (x, y). Pruébalo sobre los datos del fichero `datos.txt`, sin emplear pesos. Compara los coeficientes del polinomio obtenido con los que se obtienen empleando la función `Polynomial.fit` de Numpy. ¿Qué conclusión sacas?

3. Añade el código necesario al programa anterior para que, una vez obtenidos los coeficientes del polinomio de mínimos cuadrados, la función calcule y devuelva un vector r con los valores de los residuos, $r = y - p(x)$, donde x e y son los vectores del conjunto de datos para los que se ha obtenido el polinomio de mínimos cuadrados y p los valores obtenidos aplicando el polinomio a los valores x de la colección de datos.

1.7. Test del curso 2020/21

Problema 1. Una cuerda de escalada aumenta su longitud cuando está sometida a una tensión estacionaria. En particular, el aumento de longitud sigue la siguiente ley

$$T = b \tanh(ax), \tag{1.1}$$

donde $T \in \mathbb{R}^+$ es la tensión aplicada en Newtons, $x \in \mathbb{R}^+$ es la elongación de la cuerda en metros y $a, b \in \mathbb{R}^+$ son parámetros constantes que dependen de las características de la cuerda.

En función del comportamiento físico de la cuerda, podemos distinguir tres regímenes:

- *Comportamiento elástico*: Se dice que el comportamiento de la cuerda es elástico si al desaparecer la tensión la cuerda recupera su longitud original. Esto

the Numpy function `'CubicSpline'`. Draw the results in the same graphic where you drew exercise 1*a*) results.

2. create a function that calculates the n-degree polynomial using the least squared method to fit a set of data pairs (x, y), using the code in section 8.4 as a reference. Test your function with the data in the file `datos.txt` without employing weights. Afterward, compare the coefficients produced by your program with those obtained using the Numpy function `Polynomial.fit`. What conclusion can you draw from the comparison?

3. Add the necessary code to the previous program so that, once the coefficients of the least square polynomials has been obtained, the function calculates and return a vector r with the values of the residuals, $r = y - p(x)$, where x and y are the data set for which the function has computed the least square polynomial and p the values obtained applying the polynomial to the x values of the data set.

1.7. Course 2020/21 test

Question 1. A Climbing rope stretches when it suffers a stationary strain. Specifically, the increase in length follows the following law,

$$T = b \tanh(ax), \tag{1.1}$$

Where $T \in \mathbb{R}^+$ is the applied strain in Newtons, $x \in \mathbb{R}^+$ is the rope stretching in meters and $a, b \in \mathbb{R}^+$ are constant parameter which depend on the rope characteristics.

According to the rope physical behaviour, we can observe three regimes.

- *elastic behaviour*: The rope follows an elastic behaviour if it recovers its original length once we withdraw the strain. This is the normal behaviour under small

se cumple para tensiones estacionarias pequeñas, y las elongaciones están acotadas por un valor positivo x_{le}, esto es, $0 \leq x \leq x_{le}$. En régimen elástico, la ecuación (8.7) puede aproximarse por la siguiente expresión

$$T = \kappa x - \gamma x^3, \tag{1.2}$$

donde $\kappa, \gamma \in \mathbb{R}^+$ son también constantes.

- *Comportamiento plástico*: Se dice que el comportamiento de la cuerda es plástico si al desaparecer la tensión la cuerda se deforma y no recupera su longitud original. Esto ocurre para tensiones medias, y la elongación alcanzada en este régimen está también acotada por $x_{le} < x \leq x_{max}$.

- *Rotura*: Para tensiones grandes la cuerda no admite elongaciones mayores que x_{max} y se rompe.

La fábrica de cuerdas de escalada *Pa'bennos Matao S.L.* ha realizado un estudio sobre un nuevo modelo de cuerda. En dicho estudio se fijó un extremo de la cuerda a una mesa abatible suficientemente grande (que hará la función de un plano inclinado), y se ató el otro extremo de la cuerda a una pesa. De manera secuencial se fue incrementando el ángulo formado por la mesa con la horizontal. En particular, se empezó con cero grados y aumentando el ángulo hasta que finalmente la cuerda alcanzó x_{max} y se rompió.

Del estudio se pudo registrar la elongación sufrida por la cuerda por los distintos $i \in \{1, \ldots, 52\}$ ángulos de inclinación. Estos datos están en el archivo: `cuerda.txt`[2], en donde:

- La primera columna corresponde a los ángulos θ_i medidos en radianes.

- La segunda columna corresponde a las elongaciones x_i medidas en metros.

[2]disponible en https://github.com/UCM-237/LCC/tree/master/datos

stationary strains. In this case, the stretching is upper bounded by a positive value x_{le}, that is, $0 \leq x \leq x_{le}$. In the elastic regime equation can be approximated by the following expression,

$$T = \kappa x - \gamma x^3, \tag{1.2}$$

where $\kappa, \gamma \in \mathbb{R}^+$ are also constant.

- *Plastic behaviour:* The rope behaviour is considered plastic if the rope deforms after removing the strain and does not recover its original length. This behaviour takes place for intermediate strains and, in this plastic regime, the rope stretching is also bounded between two limits $x_{le} < x \leq x_{max}$.

- *Breaking off:* The rope does not admit a bigger stretching and breaks off for larger strains.

The climbing ropes company *Slink Yer Hook, ltd.* conducted a study on a new brand of rope. They attached one end of the rope to a tilting table, which was used as an inclined plane, and the other to a weight for the study. The angle between the table and the horizontal was gradually increased. The test started with a zero-degree angle, and the angle was increased until the rope reached the length x_{max} and broke.

During the study, the stretching suffered by the rope for different tilting angles $i \in \{1, \cdots, 52\}$ was registered. These data are available in the file `cuerda.txt`[2], where:

- The first column holds the angles θ_i measured in radians.

- The second column holds the elongations x_i measured in meters.

1. (**1 point**) Estimate the stretching limit value x_{max} beyond which the rope breaks.

2. (**1 point**) compute the strain exerted to the rope for the weight at any table tilting angle θ_i, that is,

$$T_i = mg \sin(\theta_i), \tag{1.3}$$

[2]Available in https://github.com/UCM-237/LCC/tree/master/datos

1. (**1 punto**) Estima el valor de la elongación x_{max} para el cual se produce la rotura de la cuerda.

2. (**1 punto**) Obtén la tensión ejercida por la pesa sobre la cuerda para cada ángulo de inclinación θ_i de la mesa, esto es

$$T_i = mg\sin(\theta_i), \tag{1.3}$$

donde $m = 1000$ Kg y $g = 9.8$ m/s^2. Representa gráficamente los datos: T_i frente a x_i.

3. A partir de los pares de datos T_i y x_i podemos estimar la ecuación (8.2).

   a) (**2 puntos**) Ajusta los datos por mínimos cuadrados a un polinomio de grado tres. Dado que dicha aproximación solo es válida para el régimen de comportamiento elástico, es imprescindible realizar el ajuste del polinomio asignando pesos a cada par de datos. Para dar más valor a las elongaciones pequeñas y menos a las grandes, utiliza la siguiente expresión para definir los pesos

$$\omega_i = e^{-10x_i^2}. \tag{1.4}$$

   b) (**1 punto**) Representa, sobre la gráfica dibujada en el apartado 2a, el polinomio obtenido en el apartado 3a. Según tu criterio, ¿es razonable el ajuste realizado?

4. (**1 punto**) Los coeficientes de las ecuaciones (8.7) y (8.2) están relacionados por las siguientes expresiones

$$\kappa = b \cdot a, \quad \gamma = \frac{b \cdot a^3}{3}.$$

Calcula los valores de a y b a partir de los coeficientes del polinomio obtenido en el apartado 3a. Representa, en la misma gráfica de los apartados anteriores, la función $T(x) = b\tanh(ax)$. Explica, a la vista del gráfico, si los resultados obtenidos son razonables o no.

where $m = 1000$ Kg and $g = 9.8$ m/s^2. Draw a T_i vs x_i graph.

3. From data pairs T_i and x_i we can estimate equation (8.2).

   a) (**2 points**)To fit the data, utilize a three-degree least squares polynomial. Because this method is only suitable for elastic behavior, it's important to assign weights to the data when calculating the polynomial. Use the following expression to define the weights, which will enhance the influence of small elongations and reduce the impact of larger ones.

$$\omega_i = e^{-10x_i^2}. \tag{1.4}$$

   b) (**1 point**) Plot on top the graph drew in question 2a, the polynomial you have computed in question 3b. Is it reasonable the fitting carried out? explain why.

4. (**1 point**) Coefficient in equations (8.7) y (8.2) are related by the following expressions,

$$\kappa = b \cdot a, \quad \gamma = \frac{b \cdot a^3}{3}.$$

Compute the values of parameters a and b departing from the coefficients of the polynomial obtained in question 3a. Plot the function $T(x) = b\tanh(ax)$ in the same graphic used in the previous questions. Explain, according to the graphic, is the results achieved are reasonable or not.

5. (**1 point**) Compute the value of residuals $r_i = T_i - P_3(x_i)$ where $P_3(x)$ is the polynomial obtained in question 3a. Suppose that the limit x_{le} for the rope elastic behaviour is reached when $r_i \approx 500N$, Find an approximated value for x_{le}.
Note: It is not enough to draw the residuals and estimate x_{le} by simple inspection. You must use code to do it.

5. (**1 punto**) Calcula el valor de los residuos $r_i = T_i - P_3(x_i)$, donde $P_3(x)$ es el polinomio de grado tres obtenido en el apartado 3a. Si la cota x_{le} para el comportamiento elástico de la cuerda viene definida cuando $r_i \approx 500N$. Encuentra un valor aproximado para x_{le}.
   Nota: Hay que buscar x_{le} empleando código. No vale dibujar los residuos y estimarlo a vista.

Problema 2. Dados los siguiente valores de la función $f(x)$:

$$f(0) = 1, \quad f\left(\frac{\pi}{4}\right) = \frac{\sqrt{2}}{2}, \quad f\left(\frac{\pi}{2}\right) = 0, \quad f\left(\frac{3\pi}{4}\right) = -\frac{\sqrt{2}}{2}, \quad f(\pi) = -1$$

1. (**1.5 puntos**) Utilizando el comando de Numpy correspondiente, obtener mediante interpolación con splines cúbicos los valores de la función $f(x)$ sobre cien puntos equiespaciados en el intervalo $[0, \pi]$.

2. (**1.5 puntos**) Dibuja mediante un diagrama de barras las differencias entre los $f(x)$ y la función $\cos(x)$ de Python en los mismos puntos del intervalo anterior. Considera que no necesitamos más de dos decimales para la precisión el cálculo de la función coseno; ¿podríamos utilizar la interpolación para $f(x)$?

Problem 2. Given the following values belonging to function $f(x)$:

1. (**1.5 points**) Using an appropriate Numpy function, compute using cubic splines interpolation the values of function $f(c)$ at 100 equispaced points into the interval $[0, \pi]$.

2. (**1.5 points**) Use a bar plot to draw the differences between $f(x)$ and the function $\cos(x)$ in the same points interpolated in the previous question. Suppose we only need up to two decimal precisions to calculate the cosine function. Could we then use the interpolation calculated for $f(x)$?

Índice alfabético

Alphabetic Index