



# Interpolación numérica y sus aplicaciones en la estadística

Luis Carlos Ocaña Hoeber

Universidad de Sevilla  
Facultad de Matemáticas  
Departamento de Análisis Matemático

11 de junio de 2024

# Índice

- 1 Introducción
- 2 Interpolación de Lagrange
- 3 Interpolación mediante funciones spline
- 4 Normal tabulada
- 5 Regresión spline

# Introducción

## ① La interpolación numérica

- Rol fundamental en estadística y ciencia de datos
- Estimación y predicción entre observaciones discretas
- Creación de muestras sintéticas

## ② Objetivos

- Explorar y analizar técnicas de interpolación numérica
- Comparar métodos de interpolación en términos de precisión y desempeño

## ③ Implementación práctica

- Ejemplo de implementación de la teoría en Python
- Reproducibilidad y verificación de resultados
- Disponibilidad del código en el repositorio luiocahoe en GitHub

# Interpolación de Lagrange

## Definición

Dada la función  $f : [a, b] \rightarrow \mathbb{R}$  y  $\{x_0, x_1, \dots, x_n\} \subset [a, b]$  con  $x_i \neq x_j$  si  $i \neq j$  existe un único polinomio  $P_n \in \mathcal{P}_n$  que verifica:

$$P(x_i) = f(x_i)$$

para  $i = 0, 1, \dots, n$ . Además, este polinomio viene dado por:

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

donde, para cada  $i \in \{0, 1, \dots, n\}$ ,

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

# Interpolación de Lagrange

## Definición

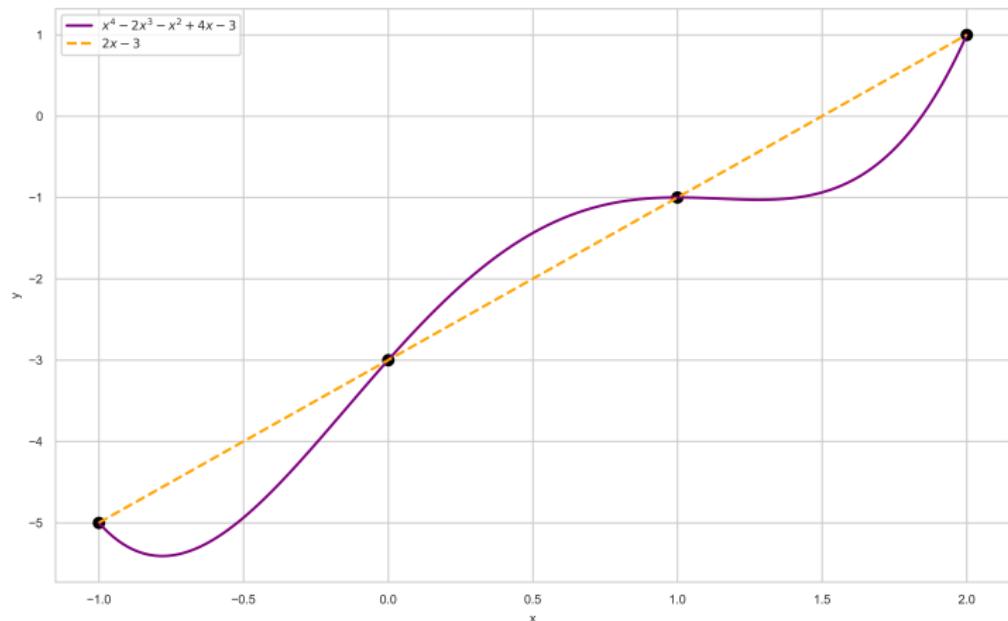


Figura: Polinomio de interpolación  $P_3(x) = 2x - 3$

# Interpolación de Lagrange

## Error de interpolación

Sea  $f \in \mathcal{C}^{n+1}([a, b])$ ,  $\{x_0, x_1, \dots, x_n\} \subset [a, b]$  con  $x_i \neq x_j$  si  $i \neq j$  y  $P_n \in \mathcal{P}_n$  el polinomio de interpolación de  $f$  en los puntos  $\{x_0, x_1, \dots, x_n\}$ . Para cada  $x \in [a, b]$  existe  $\xi_n \in I_x$ , siendo  $I_x$  el mínimo intervalo cerrado que contiene a los puntos  $\{x_0, x_1, \dots, x_n, x\}$ , tal que

$$E_n(x) = f(x) - P_n(x) = \frac{f^{n+1}(\xi_x)}{(n+1)!} \Pi_n(x) \text{ donde } \Pi_n(x) = \prod_{i=0}^n (x - x_i).$$

Podemos establecer un límite:

$$|f(x) - P_n(x)| \leq \frac{|\Pi_n(x)|}{(n+1)!} \|f^{n+1}\|_{L^\infty(a,b)}, \quad x \in [a, b]$$

donde

$$\|g\|_{L^\infty(a,b)} = \max_{a \leq x \leq b} |g(x)|$$

representa la *norma máxima* de una función continua  $g : [a, b] \rightarrow \mathbb{R}$ .

# Interpolación de Lagrange

## Aproximación de datos y método de Neville

Consideremos  $f$  como una función definida en el conjunto  $x_0, x_1, \dots, x_n$  y asumamos que  $m_1, m_2, \dots, m_k$  enteros distintos, con  $0 \leq m_i \leq n$  para cada  $i$ . El polinomio de Lagrange que coincide con  $f(x)$  en los puntos  $x_{m_1}, x_{m_2}, \dots, x_{m_k}$  se representa como  $P_{m_1, m_2, \dots, m_k}(x)$ .

Sea  $f$  definida en  $\{x_0, x_1, \dots, x_k\}$  y sean  $x_j$  y  $x_i$  dos números distintos en este conjunto. Entonces

$$P(x) = \frac{(x - x_j)P_{0,1,\dots,j-1,j+1,\dots,k}(x) - (x - x_i)P_{0,1,\dots,i-1,i+1,\dots,k}(x)}{(x_i - x_j)}$$

es el  $k$ -ésimo polinomio de Lagrange que interpola  $f$  en los puntos  $\{x_0, x_1, \dots, x_k\}$ .

# Interpolación de Lagrange

## Aproximación de datos y método de Neville

La fórmula  $P(x)$  del  $k$ -ésimo polinomio de Lagrange implica que los polinomios de interpolación pueden generarse de manera recursiva. Por ejemplo:

$$P_{0,1} = \frac{1}{x_1 - x_0} [(x - x_0)P_1 - (x - x_1)P_0],$$

$$P_{1,2} = \frac{1}{x_2 - x_1} [(x - x_1)P_2 - (x - x_2)P_1],$$

$$P_{0,1,2} = \frac{1}{x_2 - x_0} [(x - x_0)P_{1,2} - (x - x_2)P_{0,1}].$$

Por lo tanto, el algoritmo de Neville es particularmente indicado en el caso que se desee evaluar el polinomio interpolador en un único punto, o en un número muy reducido de puntos.

# Interpolación de Lagrange

## Fórmula de interpolación de Newton

Sea  $f : [a, b] \rightarrow \mathbb{R}$  y  $\{x_0, x_1, \dots, x_n\} \subset [a, b]$  con  $x_i \neq x_j$  si  $i \neq j$ . El polinomio de interpolación de  $f$  en los puntos  $\{x_0, x_1, \dots, x_n\}$  viene dado por

$$\begin{aligned} P_n(x) &= f(x_0) + \sum_{i=1}^n \Pi_{i-1}(x) f[x_0, x_1, \dots, x_i] \\ &= f(x_0) + (x - x_0)f[x_0, x_1] + (x - x_0)(x - x_1)f[x_0, x_1, x_2] \\ &\quad + \cdots + (x - x_0)(x - x_1) \cdots (x - x_{n-1})f[x_0, x_1, \dots, x_n]. \end{aligned}$$

Además, si  $x \notin \{x_0, x_1, \dots, x_n\}$ , entonces

$$E_n(x) = f(x) - P_n(x) = \Pi_n(x) f[x_0, x_1, \dots, x_n, x].$$

Siendo la diferencia dividida:

$$\left\{ \begin{array}{l} f[x_i] = f(x_i) \\ f[x_1, x_2, \dots, x_m] = \frac{f[x_i, x_{i+1}, \dots, x_{i+m-1}] - f[x_{i+1}, x_{i+2}, \dots, x_{i+m}]}{x_i - x_{i+m}}. \end{array} \right.$$

# Interpolación de Lagrange

## Polinomio de Chebychev

A la sucesión de polinomios  $\{T_n\}_{n \geq 0}$  definida por recurrencia como sigue:

$$\begin{cases} T_0 = 1, & T_1(x) = x, \\ T_{n+1}(x) = 2x T_n(x) - T_{n-1}(x), & \forall n \geq 1 \end{cases}$$

se le denomina *sucesión de polinomios de Chebychev*.

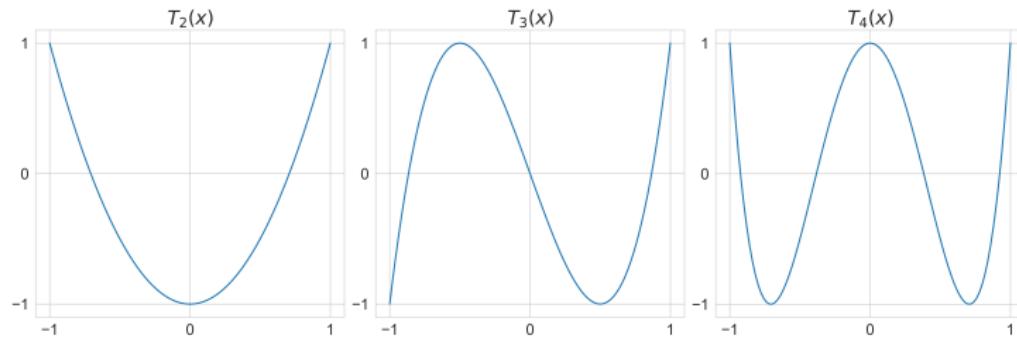


Figura: Polinomios de Chebychev

# Interpolación de Lagrange

## Polinomio de Chebychev

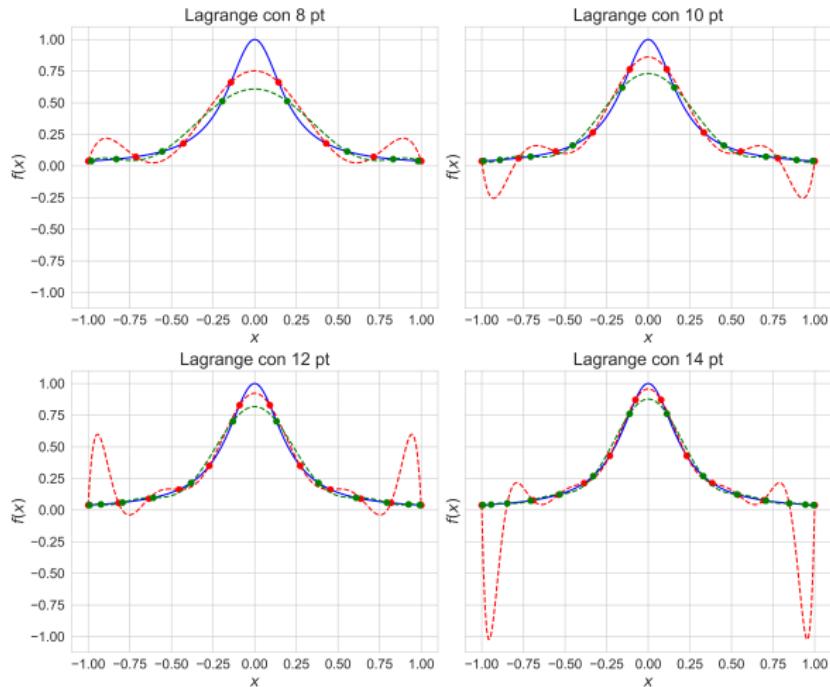


Figura: Evitando el fenómeno de Runge

# Interpolación mediante funciones spline

## Definición

Consideremos  $\Delta = \{a = x_0 < x_1 < \cdots < x_n = b\}$  como una *partición* del intervalo  $[a, b]$ . Una *función spline* de orden  $k \in \mathbb{N}$  asociada a  $\Delta$ , denotada por  $S_\Delta : [a, b] \rightarrow \mathbb{R}$ , cumple las siguientes condiciones:

- $S_\Delta \in \mathcal{C}^{k-1}([a, b])$ .
- En cada subintervalo  $[x_i, x_{i+1}]$ , para  $i = 0, 1, \dots, n - 1$ , coincide con un polinomio de grado  $\leq k$ .

# Interpolación mediante funciones spline

## Definición

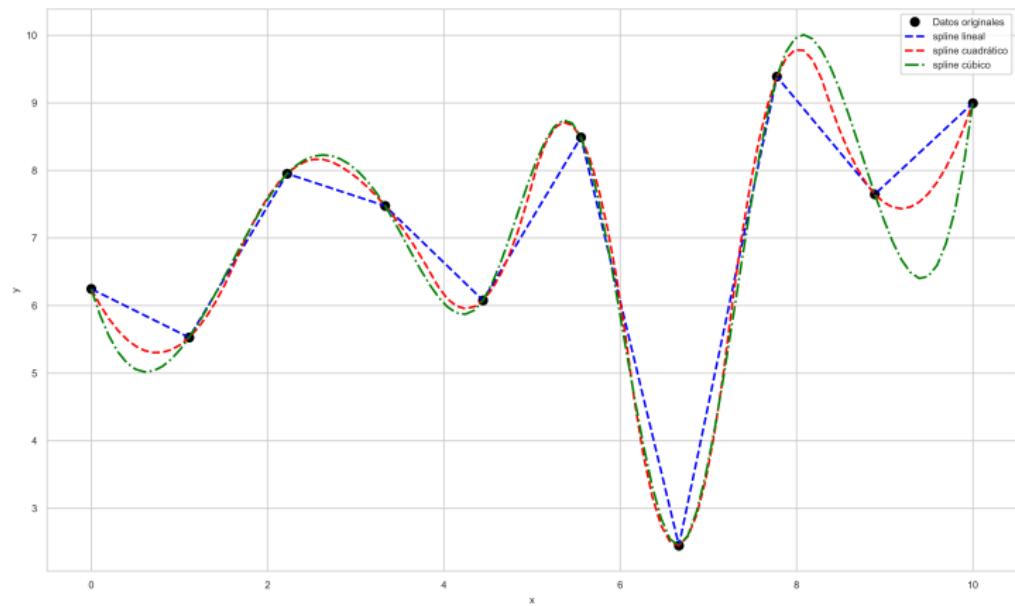


Figura: Comparación de splines

# La normal tabulada

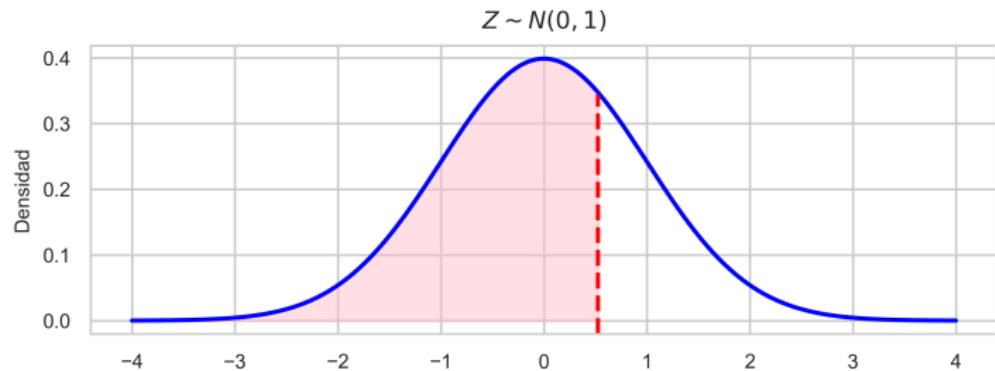
## Definición de la normal estándar

Se dice que una variable aleatoria sigue una distribución normal estándar si su función de densidad viene dada por

$$f(z) = \frac{1}{\sqrt{2\pi}} e^{\frac{-z^2}{2}}.$$

A esta variable aleatoria se le denota usualmente por  $Z \sim N(0, 1)$ . Su función de distribución viene dada por

$$F(z_1) = \int_{-\infty}^{z_1} f(t) dt = \int_{-\infty}^{z_1} \frac{1}{\sqrt{2\pi}} e^{\frac{-t^2}{2}} dt.$$



# La normal tabulada

## Integración numérica

Consideremos  $f \in \mathcal{C}^4([a, b])$ ,  $h = \frac{b-a}{2m}$  con  $m \in \mathbb{N}$  y

$$x_i = a + ih, \quad i = 0, 1, \dots, 2m.$$

La regla de Simpson compuesta con  $m$  subintervalos se escribe por

$$\int_a^b f(x) dx \simeq \frac{h}{3} \left( f(a) + 4 \sum_{i=1}^m f(x_{2i-1}) + 2 \sum_{i=1}^{m-1} f(x_{2i}) + f(b) \right).$$

# La normal tabulada

## La función error

Sea  $F(x)$  la función de distribución de la normal estándar y sea  $\text{erf}(x)$  la función de error dada por

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

Se tiene la siguiente relación entre  $F(x)$  y  $\text{erf}$ :

$$F(x) = \frac{1}{2} \left( 1 + \text{erf}\left(\frac{x}{\sqrt{2}}\right) \right).$$

# La normal tabulada

## Desarrollo de McLaurin

Pero sí podemos aproximarla usando el desarrollo de Taylor en 0 (desarrollo McLaurin). Para ello, consideramos  $g(x) = e^{-x^2}$ , donde el desarrollo está dado por:

$$g(x) = g(0) + g'(0) + \frac{g''(0)}{2} + \frac{g'''(0)}{3!} + \dots$$

Teniendo en cuenta que:

$$\begin{aligned}g(0) &= 1; & g'(x) &= -2xe^{-x^2} \Rightarrow g'(0) = 0; \\g''(x) &= -2e^{-x^2} + 4x^2e^{-x^2} \Rightarrow g''(0) = -2; & \dots\end{aligned}$$

obtenemos:

$$\begin{aligned}g(x) &= 1 - \frac{2}{2}x^2 + \dots \Rightarrow e^{-x^2} = 1 - x^2 + \dots \\&\Rightarrow \text{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt = \frac{2}{\sqrt{\pi}} \int_0^x (1 - t^2 + \dots) dt \\&= \frac{2}{\sqrt{\pi}} \left[ t - \frac{t^3}{3} + \dots \right]_{t=0}^{t=x} = \frac{2}{\sqrt{\pi}} \left( x - \frac{x^3}{3} + \dots \right).\end{aligned}$$

# La normal tabulada

## Implementación en Python

Para hacer cálculos iterativos se usan fórmulas alternativas a la anterior.  
Podemos usar la fórmula alternativa:

$$\text{erf}(x) = \frac{2}{\sqrt{\pi}} \sum_{n=0}^{\infty} \frac{x}{(2n+1)} \prod_{k=1}^n \frac{-x^2}{k}.$$

```
1 import numpy as np
2 def erf(x, m=50):
3     suma_eraf = 0.0
4     for n in range(m):
5         primero = x / (2*n + 1)
6         producto = 1.0
7         for k in range(1, n+1):
8             producto *= -(x**2) / k
9             suma_eraf += primero * producto
10    erf_valor = (2 / np.sqrt(np.pi)) * suma_eraf
11    return erf_valor
12
```

# La normal tabulada

## Implementación en Python

Por último, anidamos la función creada usando la relación que hemos mencionado anteriormente:

```
1 def norm_acum(x):  
2     return 0.5 * (1 + erf(x / np.sqrt(2)))  
3
```

```
norm_acum(2)
```

```
## 0.9772498680518207
```

```
import scipy  
scipy.stats.norm.cdf(2)
```

```
## 0.9772498680518208
```

# Regresión spline cúbica

¿Qué es la regresión?

Un modelo de regresión se representa como

$$Y = f(X_1, X_2, \dots, X_n) + \varepsilon.$$

En particular, analizaremos modelos de regresión de la forma

$$Y = f(X) + \varepsilon,$$

donde  $\varepsilon = (\varepsilon_1, \dots, \varepsilon_n)^T$  es un vector de errores aleatorios no correlados que tienen media cero y varianza común  $\sigma^2$  y  $f$  es una función de regresión desconocida.

# Regresión spline

## Regresión spline cúbica

Lo más usual es usar un polinomio cúbico por partes:

$$y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \beta_3 x_i^3 + \varepsilon_i,$$

donde los coeficientes  $\beta_0$ ,  $\beta_1$ ,  $\beta_2$  y  $\beta_3$  difieren en diferentes partes del rango de  $X$ . Los puntos donde los coeficientes cambian se llaman nodos (knots). De forma matricial:

$$Y = X\beta^T + \varepsilon$$

donde:

$$Y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}, \quad X = \begin{bmatrix} 1 & x_1 & x_1^2 & x_1^3 \\ 1 & x_2 & x_2^2 & x_2^3 \\ \vdots & \vdots & \vdots & \vdots \\ 1 & x_n & x_n^2 & x_n^3 \end{bmatrix}, \quad \beta = \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \beta_3 \end{bmatrix}, \quad \text{y} \quad \varepsilon = \begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \vdots \\ \varepsilon_n \end{bmatrix}.$$

# Regresión spline

## Regresión spline

Otra forma de representar un spline cúbico con nodos en  $\xi_k$  para  $k = 1, \dots, K$  de forma más concisa es:

$$y_i = \beta_0 + \beta_1 b_1(x_i) + \beta_2 b_2(x_i) + \cdots + \beta_{K+3} b_{K+3}(x_i) + \varepsilon_i.$$

Iniciamos la base con  $X, X^2$  y  $X^3$ , y luego agregar *funciones de base de potencia truncada*, denotadas por  $b_i$ . Es decir:

$$b_1(X) = X^1,$$

$$b_2(X) = X^2,$$

$$b_3(X) = X^3,$$

$$b_{(3+k)}(X) = (X - \xi_k)_+^3, \quad k = 1, \dots, K$$

donde:

$$h(X, \xi_k) = (X - \xi_k)_+^3 = \begin{cases} (X - \xi_k)^3 & \text{si } X > \xi_k, \\ 0 & \text{en caso contrario.} \end{cases}$$

# Regresión spline

## Medidas de evaluación

Suma de cuadrados residual

$$SS_{Res} = \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

Suma de cuadrados de regresión

$$SS_R = \sum_{i=1}^n (\hat{y}_i - \bar{y})^2$$

Suma total de cuadrados

$$SS_T = \sum_{i=1}^n (y_i - \bar{y})^2$$

Coeficiente de determinación

$$R^2 = \frac{SS_R}{SS_T} = 1 - \frac{SS_{Res}}{SS_T}.$$

Criterio de Información de Akaike

$$AIC = 2k - \log L$$

# Regresión spline

## Implementación en Python

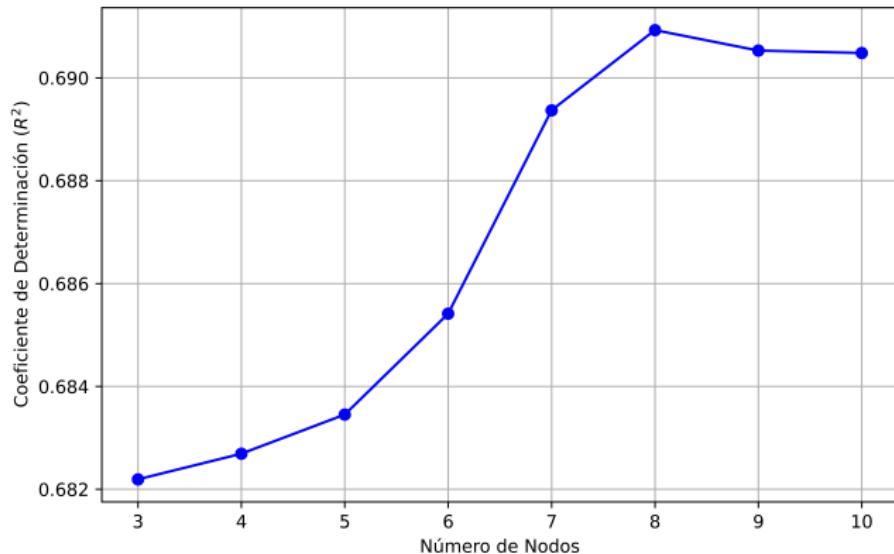


Figura: Rendimiento del modelo spline en función del número de nodos

# Regresión spline

## Implementación en Python

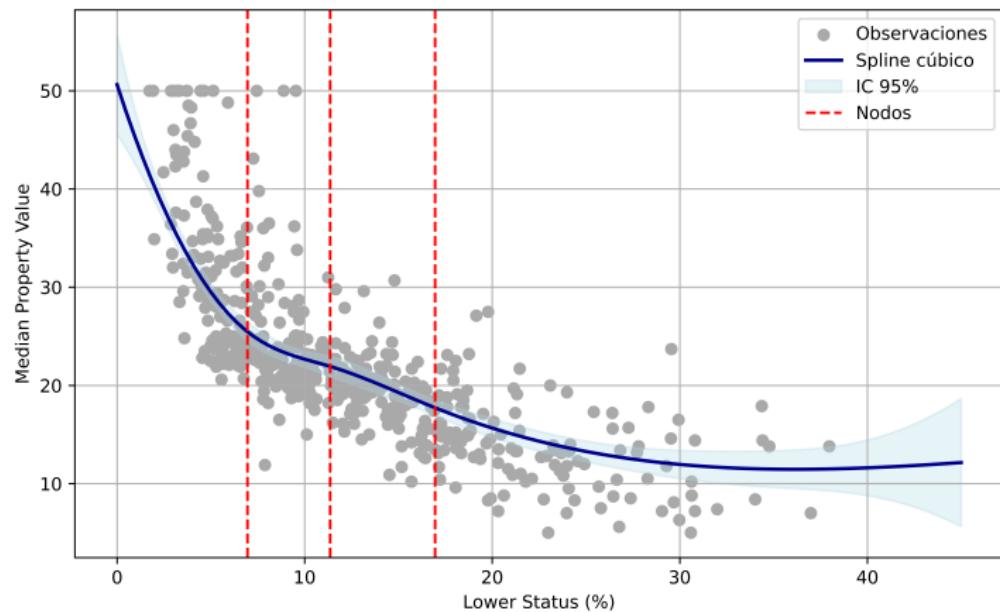


Figura: Spline cúbico con intervalo de confianza

# Regresión spline

## Implementación en Python

OLS Regression Results			
Dep. Variable:	medv	R-squared:	0.683
Model:	OLS	Adj. R-squared:	0.680
Method:	Least Squares	F-statistic:	179.5
Date:	Fri, 17 May 2024	Prob (F-statistic):	3.68e-121
Time:	12:42:49	Log-Likelihood:	-1549.3
No. Observations:	506	AIC:	3113.
Df Residuals:	499	BIC:	3142.
Df Model:	6		
Covariance Type:	nonrobust		

Cuadro: Cuadro resumen regresión spline

# Regresión spline

## Implementación en Python

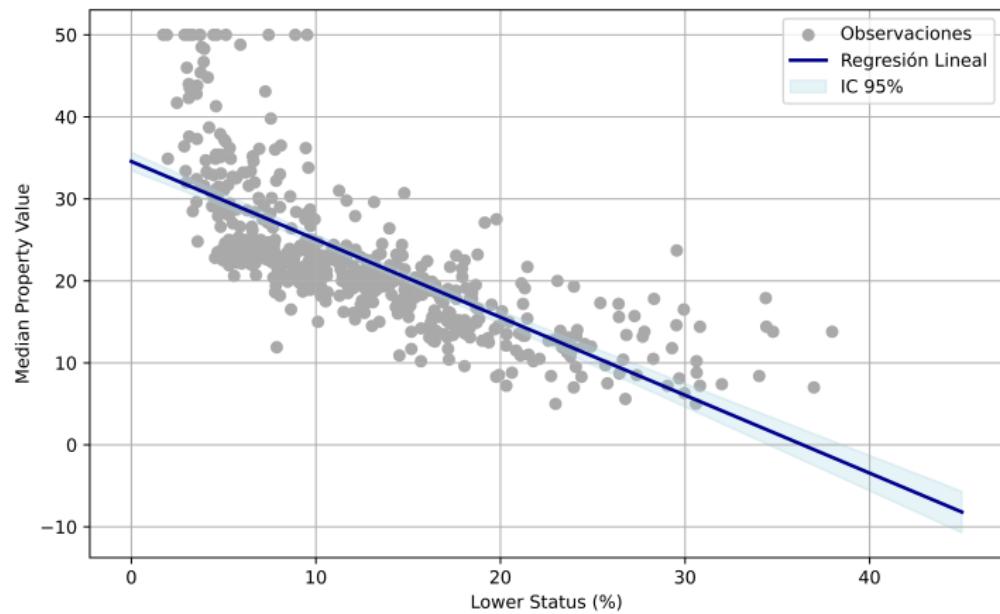


Figura: Regresión lineal con intervalo de confianza

# Regresión spline

## Implementación en Python

```
OLS Regression Results
=====
Dep. Variable: medv    R-squared:  0.544
Model: OLS      Adj. R-squared:  0.543
Method: Least Squares F-statistic: 601.6
Date: Fri, 17 May 2024 Prob (F-statistic): 5.08e-88
Time: 12:44:52 Log-Likelihood: -1641.5
No. Observations: 506 AIC: 3287.
Df Residuals: 504 BIC: 3295.
Df Model: 1
Covariance Type: nonrobust
=====
```

Cuadro: Cuadro resumen regresión lineal

# Bibliografía más relevante I

- [1] R. L. Burden, D. J. Faires y A. M. Burden, *Numerical analysis*, 10.<sup>a</sup> ed. Boston, MA: Cengage Learning, 2016, 896 págs., OCLC: ocn898154569, ISBN: 978-1-305-25366-7.
- [2] J. M. Rey Cabezas y J. A. Infante del Río, *Métodos numéricos: Teoría, problemas y prácticas con MATLAB*, 4.<sup>a</sup> ed. Pirámide, 2015, ISBN: 978-84-368-4580-8.
- [3] E. d. Rosario, “Interpolación polinómica de Lagrange con Python – Métodos numéricos,” (4 de sep. de 2017), dirección: <http://blog.espol.edu.ec/analisisnumerico/interpolacion-de-lagrange/> (visitado 11-03-2024).
- [4] A. Doubova y F. Guillén González, *Un curso de cálculo numérico: Interpolación, Aproximación, Integración y Resolución de Ecuaciones Diferenciales*. Universidad de Sevilla, 2007, ISBN: 978-84-472-0941-5.

# Bibliografía más relevante II

- [5] Wikipedia, *Función error*, en Wikipedia, la enciclopedia libre, Page Version ID: 159001549, 24 de mar. de 2024. dirección: [https://es.wikipedia.org/w/index.php?title=Funci%C3%A3n\\_error&oldid=159001549](https://es.wikipedia.org/w/index.php?title=Funci%C3%A3n_error&oldid=159001549) (visitado 26-04-2024).
- [6] G. James, D. Witten, T. Hastie, R. Tibshirani y J. E. Taylor, *An introduction to statistical learning: with applications in Python* (Springer texts in statistics). Cham, Switzerland: Springer, 2023, 607 págs., ISBN: 978-3-031-38747-0 978-3-031-38746-3.
- [7] S. Jackson, “Chapter 9 Splines — Machine Learning,” (), dirección: <https://bookdown.org/ssjackson300/Machine-Learning-Lecture-Notes/splines.html> (visitado 03-05-2024).



## Interpolación numérica y sus aplicaciones en la estadística

Muchas gracias por su atención