

# Métodos Numéricos

Departamento de Matemática Aplicada II  
Escuela Técnica Superior de Ingeniería  
Universidad de Sevilla

## Lección 3: Interpolación polinómica

### Índice

<b>1. El polinomio de interpolación</b>	<b>2</b>
1.1. Polinomios en MATLAB® . . . . .	2
1.2. Existencia y unicidad del polinomio de interpolación . . . . .	3
1.3. La orden <code>polyfit</code> . . . . .	4
1.4. Forma de Lagrange del polinomio de interpolación . . . . .	7
<b>2. Interpolación polinómica a trozos</b>	<b>8</b>
2.1. Interpolación lineal a trozos . . . . .	8
2.2. Splines cúbicos: la orden <code>spline</code> . . . . .	9
<b>3. Apéndice A: Polinomios a trozos en MATLAB®</b>	<b>14</b>
<b>4. Apéndice B: La biblioteca de funciones <code>fdlibm</code></b>	<b>18</b>
<b>5. Soluciones de los proyectos</b>	<b>21</b>

Son muchas y muy variadas las situaciones donde aparecen series de datos o resultados de mediciones experimentales de las que solo se conoce una cantidad finita de valores y para las que, sin embargo, se necesita encontrar cierta «ley general» que sirva para su tratamiento computacional. Precisamente, este es el cometido básico de la interpolación: dada una tabla de datos, se trata de encontrar una función fácil de manejar desde un punto de vista numérico tal que, como mínimo, su gráfica pase por los puntos proporcionados por la tabla.

La elección del tipo de funciones interpoladoras depende básicamente del contexto en que se esté trabajando. En esta lección solo trataremos la interpolación polinómica clásica y ciertos tipos de interpolación polinómica a trozos, también denominada interpolación por *splines*.

La interpolación polinómica clásica tiene gran importancia teórica en análisis numérico, pues permite fundamentar una amplia gama de métodos para la integración y derivación numéricas, para la resolución numérica de ecuaciones diferenciales, etc. Sin embargo, este tipo de interpolación tiene una gran desventaja: cuando se tienen muchos datos, el correspondiente polinomio interpolador tiene necesariamente grado alto y suele presentar numerosas oscilaciones. Para soslayar este inconveniente, la estrategia habitual es usar la interpolación por *splines*, es decir, interpolar con polinomios de grado bajo entre datos consecutivos de modo que la función global a trozos formada por dichos polinomios tenga ciertas propiedades de regularidad.

## 1. El polinomio de interpolación

### 1.1. Polinomios en MATLAB®

En MATLAB®, los polinomios se manejan como vectores (fila o columna). Si queremos introducir un polinomio en MATLAB®, creamos un vector cuyas entradas coincidan con los coeficientes del polinomio, dispuestos en orden descendente de potencias.

#### Ejemplo 1.1 ► Polinomio en MATLAB®

El polinomio  $p(x) = x^5 - 5x^2 + 2$  se representa en MATLAB® en la forma

Command Window

```
>> p=[1 0 0 -5 0 2];
```

Para evaluar polinomios, MATLAB® incorpora la orden `polyval` (*POLYnomial VALuation*), que utiliza el denominado algoritmo de Horner, o de multiplicación anidada. En concreto, dado el polinomio

$$p(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0,$$

el algoritmo de Horner se sustenta en la siguiente forma de escribir el polinomio anterior:

$$p(x) = a_0 + x(a_1 + x(a_2 + x(a_3 + \cdots + x(a_{n-1} + a_n x) \cdots)))^1.$$

<sup>1</sup>El algoritmo de Horner es el procedimiento menos costoso en número de operaciones para evaluar un polinomio: si el polinomio tiene de grado  $n$ , el algoritmo consta tan solo de  $n$  multiplicaciones y  $n$  sumas.

**Ejemplo 1.2 ► Valor de un polinomio en MATLAB®**

Para calcular el valor del polinomio  $p(x) = x^4 - 6x^3 + x^2 + 3x - 2$  en el punto  $x = 2$ , hay que ejecutar en MATLAB® las instrucciones

Command Window

```
>> p=[1 -6 1 3 -2];
>> polyval(p,2)
```

```
ans =
-24
```

La orden `polyval` ha calculado el valor del polinomio en  $x = 2$  en la forma

$$-2 + x(3 + x(1 + x(-6 + x))) = -2 + 2(3 + 2(1 + 2(-6 + 2))) = -24.$$

**1.2. Existencia y unicidad del polinomio de interpolación**

Dada una tabla con  $n \geq 2$  parejas de datos,

$$\begin{array}{|c|c|c|c|c|} \hline x_1 & x_2 & \cdots & x_{n-1} & x_n \\ \hline y_1 & y_2 & \cdots & y_{n-1} & y_n \\ \hline \end{array}, \quad (1)$$

donde  $x_1 < x_2 < \cdots < x_n$ , el problema de la interpolación polinómica consiste en encontrar un polinomio  $p$  de grado menor o igual que  $n - 1$ , denominado **polinomio de interpolación**, de modo que verifique las condiciones

$$p(x_j) = y_j, \quad j = 1, 2, \dots, n,$$

llamadas **condiciones de interpolación**.

Los números  $x_j$  se denominan **nodos de la interpolación** y los  $y_j$ , **valores de la interpolación**. Los puntos  $(x_j, y_j)$  son conocidos como **puntos de la interpolación**. Como veremos, este polinomio siempre existe y es único.

Puesto que  $p$  debe ser de grado menor o igual que  $n - 1$ , podemos considerar que

$$p(x) = a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \cdots + a_1x + a_0,$$

y plantear si los coeficientes  $a_j$  pueden determinarse de modo unívoco a partir de la tabla de interpolación (1). Imponiendo las  $n$  condiciones de interpolación llegamos al sistema de ecuaciones lineales

$$\begin{bmatrix} x_1^{n-1} & x_1^{n-2} & \cdots & x_1 & 1 \\ x_2^{n-1} & x_2^{n-2} & \cdots & x_2 & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ x_{n-1}^{n-1} & x_{n-1}^{n-2} & \cdots & x_{n-1} & 1 \\ x_n^{n-1} & x_n^{n-2} & \cdots & x_n & 1 \end{bmatrix} \begin{bmatrix} a_{n-1} \\ a_{n-2} \\ \vdots \\ a_1 \\ a_0 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_{n-1} \\ y_n \end{bmatrix}.$$

Observemos que la matriz  $V$  de este sistema es cuadrada de orden  $n$ .  $V$  es, además, de un tipo muy especial; en concreto, es una matriz de Vandermonde por filas asociada a los valores  $\{x_1, x_2, \dots, x_n\}$ . Los resultados sobre este tipo de matrices nos permiten afirmar que  $V$  tiene determinante no nulo si, y solamente si, todos los puntos asociados  $x_j$  son distintos. De hecho,

$$\det(V) = \prod_{1 \leq i < j \leq n} (x_j - x_i).$$

Puesto que en nuestro caso los nodos son distintos entre sí, tenemos garantizada la existencia y unicidad del polinomio de interpolación.

En ciertos contextos, la tabla de interpolación (1) procede de evaluaciones de cierta función. Es decir, existe una función  $f : [a, b] \rightarrow \mathbb{R}$ , con  $a \leq x_1 < x_2 < \dots < x_n \leq b$ , de modo que

$$y_j = f(x_j), \quad j = 1, 2, \dots, n.$$

Por tanto, tiene sentido plantear hasta qué punto el polinomio de interpolación  $p$  asociado reconstruye o representa la función  $f$ . En concreto, si  $f$  es de clase  $C^n[a, b]$ , para cada  $x \in [a, b]$  existe un punto  $\xi = \xi(x) \in (a, b)$  tal que

$$f(x) - p(x) = \frac{f^{(n)}(\xi)}{n!} (x - x_1)(x - x_2) \cdots (x - x_n). \quad (2)$$

Respecto a esta fórmula del error puntual, conviene hacer dos precisiones:

1. En el exterior del intervalo  $[x_1, x_n]$ , la función  $x \mapsto |(x - x_1)(x - x_2) \cdots (x - x_n)|$  crece rápidamente (de hecho, más cuanto más alejado esté  $x$  de dicho intervalo) y, lo habitual, es que el error puntual se dispare.
2. El error depende, obviamente, del número de nodos y de la regularidad de  $f$ , pero también de la elección y distribución de dichos nodos.

### 1.3. La orden polyfit

La orden **polyfit** (*POLYNomial FITting*) determina el polinomio de interpolación mediante la resolución de sistemas de ecuaciones lineales. De hecho, esta orden permite abordar, de manera completamente general, los denominados ajustes polinomiales. En concreto, volviendo a los datos de la tabla (1) y dado cierto grado  $M \geq 1$ , esta orden obtiene un polinomio  $p_M$  de grado menor o igual que  $M$ ,

$$p_M(x) = a_M x^M + \cdots + a_1 x + a_0,$$

de modo que los coeficientes  $a_j$  son la solución (clásica o en el sentido de los mínimos cuadrados) del sistema de ecuaciones lineales

$$\begin{bmatrix} x_1^M & \cdots & x_1 & 1 \\ x_2^M & \cdots & x_2 & 1 \\ \vdots & & \vdots & \vdots \\ x_n^M & \cdots & x_n & 1 \end{bmatrix} \begin{bmatrix} a_M \\ a_{M-1} \\ \vdots \\ a_0 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix},$$

cuya matriz tiene orden  $n \times (M + 1)$ . Puesto que los nodos  $x_j$  son distintos entre sí, la matriz del sistema anterior tiene rango máximo. En particular, si  $M \leq n - 1$ , el sistema tiene

solución única (en el sentido de los mínimos cuadrados si  $M < n - 1$  y clásica si  $M = n - 1$ ). Por otro lado, no es razonable tomar  $M > n - 1$ , pues en este caso el sistema de ecuaciones lineales es claramente compatible, pero indeterminado y, por tanto, estamos considerando un problema donde la solución no es única ni siquiera en el sentido de los mínimos cuadrados.

Si  $M = n - 1$ , el sistema proporciona el polinomio de interpolación asociado a la tabla, luego dicho polinomio pasa por todos los puntos. En cambio, si  $M < n - 1$ , el ajuste polinomial conduce, en general, a un polinomio que no pasa por todos los datos a interpolar.

### Ejemplo 1.3 ► Polinomio de interpolación

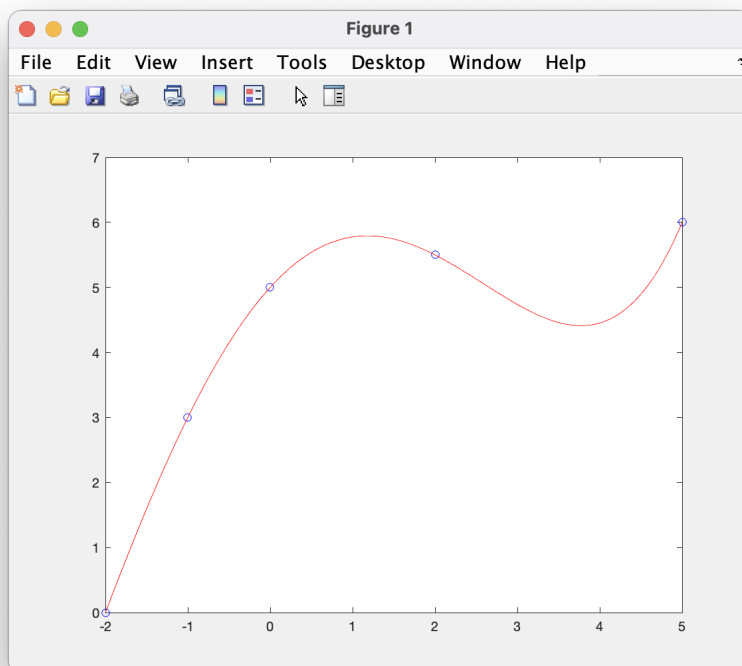
Consideremos el polinomio de interpolación asociado a la tabla de datos

-2	-1	0	2	5
0	3	5	5.5	6

Las siguientes órdenes de MATLAB® generan su representación gráfica, calculándolo previamente en una partición fina del intervalo  $[-2, 5]$ . Observemos que el polinomio de interpolación pasa por todos los puntos de la tabla.

Command Window

```
>> x=[-2 -1 0 2 5];  
>> y=[0 3 5 5.5 6];  
>> s=-2:0.01:5;  
>> c=polyfit(x,y,4);  
>> t=polyval(c,s);  
>> plot(x,y,'ob',s,t,'r')
```



Cuando el número de nodos de la tabla (1) es alto, pueden presentarse dos problemas:

1. El polinomio de interpolación oscila cerca de los extremos del intervalo de interpolación. Es el conocido fenómeno de Runge, que veremos en el proyecto 1.1. El surgimiento de estas oscilaciones es consecuencia del elevado grado del polinomio y no tiene ninguna relación con el tratamiento numérico del problema de interpolación.
2. Las correspondientes matrices de Vandermonde que surgen al resolver los sistemas de ecuaciones lineales asociados suelen tener números de condición muy elevados. En cualquier caso, cuando aparecen problemas de condicionamiento, MATLAB® da un mensaje de aviso e indica posibles estrategias para resolver o, al menos, paliar este problema. En el proyecto 1.2 veremos un ejemplo de este fenómeno de mal condicionamiento y de los consiguientes problemas que ocasiona, así como de la táctica que permite atenuar los efectos desastrosos de la mala condición.

Estos dos fenómenos nos previenen contra el uso, en análisis numérico, de polinomios de interpolación de grado relativamente alto y conduce, de modo natural, a la interpolación polinómica a trozos, que estudiaremos en la siguiente sección.

### Proyecto 1.1 (solución en página 21)

Considera la función

$$f(x) = \frac{1}{1+x^2}, \quad x \in [-5, 5].$$

El polinomio de interpolación  $p_{n-1}(x)$  que interpola  $f$  en los nodos

$$x_i = -5 + \frac{10}{n-1}(i-1), \quad i = 1, 2, \dots, n,$$

no converge a  $f$  cuando  $n$  tiende a infinito. De hecho, se verifica

$$\lim_{n \rightarrow \infty} \left( \max_{x \in [-5, 5]} |f(x) - p_n(x)| \right) = +\infty.$$

Dibuja conjuntamente las gráficas de la función  $f$  y del polinomio de interpolación  $p_n$ , para valores de  $n$  crecientes, y observa este fenómeno.

### Proyecto 1.2 (solución en página 23)

Calcula y representa gráficamente el polinomio de interpolación asociado a la tabla de datos

51	52	53	54	55	56	57	58	59	60	61	62	63	64	65
22.9	23.2	23	24.4	26.7	25	25.5	26.3	26.1	26.2	27	26.3	28	26.4	27.8

¿Qué observas?

Indaga en la ayuda de MATLAB® y, siguiendo las indicaciones que el programa te ha dado al calcular el polinomio de interpolación, plantea alguna estrategia que pueda paliar los malos resultados obtenidos.

## 1.4. Forma de Lagrange del polinomio de interpolación

Como hemos visto, la orden `polyfit` permite representar el polinomio de interpolación mediante monomios. Se trata de la denominada forma «canónica». Pero existen otras formas alternativas de representar el polinomio de interpolación.

Posiblemente la manera de representar el polinomio de interpolación con más implicaciones teóricas en análisis numérico sea la denominada forma de Lagrange. En concreto, a partir de los nodos de la tabla de interpolación (1), consideremos los polinomios

$$L_j(x) := \prod_{i \neq j} \frac{x - x_i}{x_j - x_i}, \quad j = 1, 2, \dots, n.$$

Es inmediato comprobar que cada  $L_j$ , para  $j = 1, 2, \dots, n$ , tiene grado exactamente  $n - 1$  y verifica

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

Esta propiedad permite resolver fácilmente el problema de interpolación. En concreto, el polinomio de interpolación asociado a la tabla (1) viene dado por

$$p(x) = y_1 L_1(x) + y_2 L_2(x) + \dots + y_n L_n(x).$$

Los polinomios  $L_j$  se denominan polinomios de Lagrange asociados a  $\{x_1, x_2, \dots, x_n\}$  y la expresión anterior, **forma de Lagrange del polinomio de interpolación** asociado a la tabla de datos.

### Ejemplo 1.4 ► Forma de Lagrange

Vamos a determinar la forma de Lagrange del polinomio de interpolación asociado a la tabla de datos

-1	2	3
3	1	2

La forma de Lagrange viene dada por

$$p(x) = y_1 L_1(x) + y_2 L_2(x) + y_3 L_3(x) = 3L_1(x) + L_2(x) + 2L_3(x),$$

donde

$$L_1(x) = \frac{(x - x_2)(x - x_3)}{(x_1 - x_2)(x_1 - x_3)}, \quad L_2(x) = \frac{(x - x_1)(x - x_3)}{(x_2 - x_1)(x_2 - x_3)}, \quad L_3(x) = \frac{(x - x_1)(x - x_2)}{(x_3 - x_1)(x_3 - x_2)},$$

son los polinomios de Lagrange, esto es,

$$L_1(x) = \frac{(x - 2)(x - 3)}{12}, \quad L_2(x) = \frac{(x + 1)(x - 3)}{-3}, \quad L_3(x) = \frac{(x + 1)(x - 2)}{4}.$$

Así, el polinomio de interpolación en forma de Lagrange asociado a la tabla anterior es

$$p(x) = \frac{(x - 2)(x - 3)}{4} - \frac{(x + 1)(x - 3)}{3} + \frac{(x + 1)(x - 2)}{2}.$$

## 2. Interpolación polinómica a trozos

Matemáticamente, la interpolación polinómica a trozos es simplemente un refinamiento de la interpolación polinómica. Consiste, en general, en usar polinomios diferentes y de grado bajo en cada uno de los distintos subintervalos determinados por los nodos de interpolación. Asimismo, a la correspondiente función global definida por partes se le suele exigir algún tipo de regularidad.

### 2.1. Interpolación lineal a trozos

Consideremos, nuevamente, la tabla de interpolación (1). En esta ocasión, buscamos en cada intervalo  $[x_{j-1}, x_j]$ ,  $j = 2, 3, \dots, n$ , un polinomio  $p_j$  de grado menor o igual que uno que interpole los datos en los extremos del intervalo. Deducimos fácilmente que es el segmento rectilíneo

$$p_j(x) = \frac{x_j - x}{x_j - x_{j-1}}y_{j-1} + \frac{x - x_{j-1}}{x_j - x_{j-1}}y_j, \quad x \in [x_{j-1}, x_j].$$

Con estos  $n - 1$  polinomios se define, por partes, el interpolante  $p$ . A saber,

$$p(x) := p_j(x), \quad x \in [x_{j-1}, x_j], \quad j = 2, 3, \dots, n.$$

Observemos que  $p$  está bien definido en todos los nodos y determina, de hecho, una función continua en el intervalo de interpolación  $[x_1, x_n]$ . Al interpolante  $p$  se le conoce con el nombre de **interpolante lineal a trozos** (asociado a la tabla de datos dada) y su gráfica no es sino la poligonal que pasa por los puntos de interpolación.

Además, si los valores  $y_j$  de la tabla (1) proceden de evaluaciones de cierta función  $f : [a, b] \rightarrow \mathbb{R}$  de clase  $C^2[a, b]$ , se tiene que

$$\max_{x \in [a, b]} |f(x) - p(x)| = \mathcal{O}(h^2)$$

siendo  $h$  el diámetro de la partición generada por los nodos, es decir,

$$h := \max\{x_j - x_{j-1} : j = 2, 3, \dots, n\}.$$

Esta estimación del error (denominada de tipo cuadrático) nos indica que con un mínimo de regularidad de  $f$ , el interpolante lineal a trozos converge a  $f$  rápidamente a medida que el diámetro  $h$  asociado a la familia de nodos tiende a cero.

La orden `interp1`<sup>2</sup> de MATLAB® proporciona el interpolante lineal a trozos.

#### Ejercicio 2.1

Considerando particiones uniformes del intervalo  $[0, 3]$ , muestra una tabla con los errores máximos cometidos al interpolar la función  $f(x) = \cos(x)$  por los correspondientes interpolantes lineales a trozos. ¿Dónde puede detectarse la convergencia cuadrática en la tabla obtenida?

<sup>2</sup>El número 1 indica unidimensionalidad.

## 2.2. Splines cúbicos: la orden spline

El proceso descrito para la interpolación lineal a trozos puede extenderse a un grado arbitrario. En concreto, dada la tabla de interpolación (1), se denomina *spline* de orden  $k \geq 1$  asociado a dicha tabla, a toda función

$$S : [x_1, x_n] \rightarrow \mathbb{R}$$

de clase, al menos,  $C^{k-1}[x_1, x_n]$  tal que interpola los puntos de la tabla (1) y coincide en cada subintervalo  $[x_{j-1}, x_j]$ ,  $j = 2, 3, \dots, n$ , con un polinomio de grado menor o igual que  $k$ .

En la práctica, además de los *splines* de orden 1 (esto es, los interpolantes lineales a trozos), se manejan preferentemente los de orden 2 y 3 y, en algunos contextos, los de orden 4. En cualquier caso, en esta sección nos centraremos exclusivamente en los de orden tres, también denominados *splines* cúbicos<sup>3</sup>.

Un **spline** asociado a la tabla de interpolación (1), es, por tanto, una función

$$S : [x_1, x_n] \rightarrow \mathbb{R}$$

de clase, al menos,  $C^2[x_1, x_n]$  tal que interpola los puntos de la tabla (1), esto es

$$S(x_j) = y_j, \quad \text{para todo } j = 1, 2, \dots, n.$$

y coincide en cada subintervalo  $[x_{j-1}, x_j]$ ,  $j = 2, 3, \dots, n$ , con un polinomio de grado menor o igual que tres.

Los  $n - 1$  polinomios de grado menor o igual que tres de los que consta  $S$  se denominan cúbicas del *spline*. Denotemos dichas cúbicas por  $C_{j-1}(x)$ ,  $x \in [x_{j-1}, x_j]$ , con  $j = 2, 3, \dots, n$ .

Si abordamos el problema de determinar el *spline* planteando el correspondiente sistema de ecuaciones lineales, vemos que debemos hallar  $4(n - 1)$  coeficientes (4 coeficientes por cada cúbica), imponiendo las siguientes condiciones:

i)  $n - 2$  ecuaciones para asegurar que  $S$  es continua en los  $n - 2$  nodos interiores,

$$C_{j-1}(x_j) = C_j(x_j), \quad j = 2, 3, \dots, n - 1.$$

ii)  $n - 2$  ecuaciones para asegurar que  $S'$  es continua en los  $n - 2$  nodos interiores,

$$C'_{j-1}(x_j) = C'_j(x_j), \quad j = 2, 3, \dots, n - 1.$$

iii)  $n - 2$  ecuaciones para asegurar que  $S''$  es continua en los  $n - 2$  nodos interiores,

$$C''_{j-1}(x_j) = C''_j(x_j), \quad j = 2, 3, \dots, n - 1.$$

iv)  $n$  ecuaciones para imponer las condiciones de interpolación

$$C_j(x_j) = y_j, \quad j = 1, 2, \dots, n - 1, \quad C_{n-1}(x_n) = y_n.$$

---

<sup>3</sup>O, simplemente, *splines*.

Es decir, tenemos un sistema de  $4n - 6$  ecuaciones lineales con  $4n - 4$  incógnitas. Puede comprobarse que dichas ecuaciones son independientes, luego el sistema es compatible indeterminado y deducimos, por tanto, que nuestro problema tiene solución; es más, tiene infinitas soluciones. Para evitar esta indefinición, se añaden dos ecuaciones lineales de modo que el sistema resultante sea compatible determinado. Hay una enorme variedad de formas de realizar este proceso y así nos encontramos con *splines* naturales, sujetos, periódicos, de torsión, *not-a-knot*, etc. A un nivel introductorio, es suficiente conocer los tipos siguientes:

1. **Spline sujeto**: Se impone

$$S'(x_1) = \alpha, \quad S'(x_n) = \beta,$$

esto es,

$$C'_1(x_1) = \alpha, \quad C'_{n-1}(x_n) = \beta,$$

con  $\alpha$  y  $\beta$  números reales arbitrarios.

2. **Spline Not-a-Knot**: Se impone que  $S$  sea de clase  $C^3$  en los nodos  $x_2$  y  $x_{n-1}$ <sup>4</sup>,

$$S'''(x_2^-) = S'''(x_2^+), \quad S'''(x_{n-1}^-) = S'''(x_{n-1}^+),$$

esto es,

$$C'''_1(x_2) = C'''_2(x_2), \quad C'''_{n-2}(x_{n-1}) = C'''_{n-1}(x_{n-1}),$$

lo que equivale a exigir que sean idénticas, por un lado, las cúbicas primera y segunda y, por otro, las cúbicas penúltima y última<sup>5</sup>. Esto se debe a que el desarrollo de Taylor de una cúbica en torno a un punto está completamente determinado por el valor de dicha cúbica y de sus tres primeras derivadas en dicho punto.

3. **Spline Natural**: Se impone

$$S''(x_1) = 0, \quad S''(x_n) = 0,$$

esto es,

$$C''_1(x_1) = 0, \quad C''_{n-1}(x_n) = 0.$$

Como vimos en la sección anterior, en el caso en que los valores de interpolación proceden de evaluaciones de cierta función  $f$ , se tiene que la evolución del error de interpolación para interpolantes lineales a trozos era esencialmente de magnitud  $\mathcal{O}(h^2)$ , siendo  $h$  el diámetro de la partición generada por los nodos. Un resultado similar es cierto, pero ahora de orden  $\mathcal{O}(h^4)$ , tanto para los *splines not-a-knot* como para aquellos *splines* sujetos verificando

$$S'(x_1) = f'(x_1), \quad S'(x_n) = f'(x_n).$$

La orden *spline* de MATLAB® proporciona tanto el *spline not-a-knot* como el de tipo sujeto<sup>6</sup>.

<sup>4</sup>Luego necesariamente  $n \geq 4$ .

<sup>5</sup>Si  $n = 4$ , el *spline not-a-knot* es un polinomio de grado menor o igual que tres.

<sup>6</sup>El *spline* natural se puede obtener en MATLAB® con la orden `csape` (*Cubic Spline with Additional Prescribed Ends*), que pertenece al *Curve Fitting Toolbox*. En concreto, se obtiene mediante la opción `variational`.

### Problema 2.1

Considera la función de Bessel  $J_2(x)^a$ .

1. Dibuja dicha función en el intervalo  $[2, 8]$ . Mediante la orden de MATLAB® `polyfit`, dibuja también el polinomio de interpolación que interpola la función de Bessel en los nodos 2, 3, 4, 5, 6, 7, 8.

Estima el error máximo que se comete al aproximar la función de Bessel por este polinomio de interpolación. Calcula este error máximo sobre los puntos de una partición fina de  $[2, 8]$ .

2. Mediante la orden de MATLAB® `spline`, dibuja el spline sujeto  $S$  que interpola la función de Bessel en los nodos 2, 3, 4, 5, 6, 7, 8 y tiene valores de la derivada en los nodos extremos

$$S'(2) = 0.0009, \quad S'(8) = 0.5588.$$

¿Crees que este spline ofrece una buena aproximación de la función de Bessel?

3. Aproxima el valor de la derivada de la función de Bessel en los nodos extremos mediante un cociente incremental con paso  $h = 10^{-5}$ <sup>b</sup>. Repite el apartado anterior sustituyendo los valores de la derivada en los nodos extremos por las aproximaciones obtenidas. Estima, sobre los puntos de una partición fina de  $[2, 8]$ , el error máximo que se comete al aproximar la función de Bessel por este interpolante.
4. Vuelve a utilizar la orden de MATLAB® `spline`, pero en esta ocasión para dibujar el spline not-a-knot que interpola la función de Bessel en los nodos 2, 3, 4, 5, 6, 7, 8. Estima, sobre los puntos de una partición fina de  $[2, 8]$ , el error máximo que se comete al aproximar la función de Bessel por este interpolante.

<sup>a</sup>MATLAB® implementa esta función mediante la orden `besselj`.

<sup>b</sup>Las fórmulas para aproximar la derivada se estudian en la lección 5. No obstante, puedes utilizar en este apartado la denominada *fórmula de la diferencia progresiva*, esto es,

$$f'(x) \approx \frac{f(x+h) - f(x)}{h}.$$

### Ejemplo 2.1 ► La orden `pchip`

Consideremos la siguiente tabla de interpolación (denominada ejemplo de Akima):

0	2	3	5	6	8	9	11	12	14	15
10	10	10	10	10	10	10.5	15	50	60	85

Observemos que los valores de interpolación forman una sucesión creciente, por lo que parece razonable que se exija este comportamiento al correspondiente interpolante. Sin embargo, si aplicamos la orden `spline` de MATLAB® a la tabla anterior y dibujamos el *spline not-a-knot*, obtenemos:

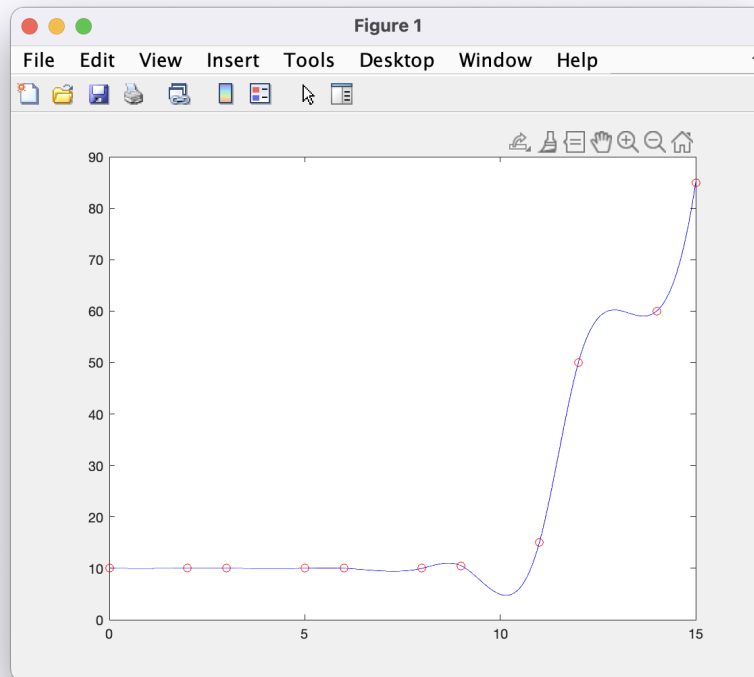
## Editor ► Ejemplo de Akima (con spline)

ejem21a.m

```

nodos=[0 2 3 5 6 8 9 11 12 14 15];
valores=[10 10 10 10 10 10 10.5 15 50 60 85];
x=0:0.01:15;
s=spline(nodos,valores,x);
plot(x,s,'b',nodos,valores,'ro'),axis([0 15 0 90]),shg

```



Como se puede ver, la monotonía inicial del interpolante se rompe en varias zonas del subintervalo  $[5, 15]$ .

El interpolante que proporciona la orden **pchip** (*Piecewise Cubic Hermite Interpolation Polynomial*) de MATLAB®, aunque da lugar a curvas menos suaves que los *splines* cúbicos usuales, no padece este defecto de ruptura de la monotonía. De hecho, si repetimos la interpolación anterior con esta orden, obtenemos ahora:

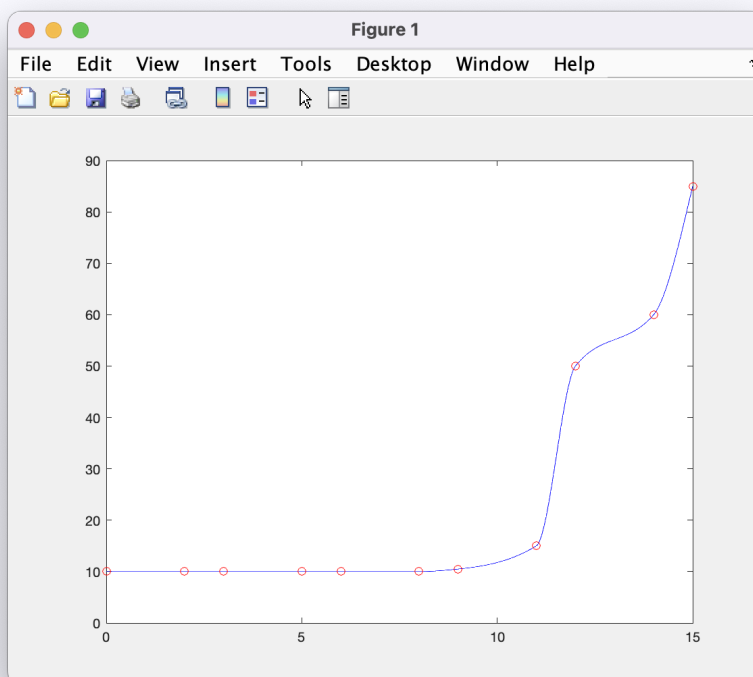
## Editor ► Ejemplo de Akima (con pchip)

ejem21b.m

```

nodos=[0 2 3 5 6 8 9 11 12 14 15];
valores=[10 10 10 10 10 10 10.5 15 50 60 85];
x=0:0.01:15;
s=pchip(nodos,valores,x);
plot(x,s,'b',nodos,valores,'ro'),axis([0 15 0 90]),shg

```



Estrictamente hablando, la orden `pchip` no es un *spline* cúbico pues, si bien es un polinomio a trozos definido mediante una serie de cúbicas, solo puede asegurarse que es de clase  $C^1$  en el intervalo de interpolación. A este respecto, conviene observar la evolución de la convexidad de la curva alrededor de  $x = 11$ .

### Proyecto 2.1 (solución en página 26)

Repite el ejercicio 2.1, pero sustituye los interpolantes lineales a trozos por splines not-a-knot y comprueba su convergencia de orden cuatro.

### Proyecto 2.2 (solución en página 27)

En este proyecto pondremos a prueba tus recursos.

Aunque no se ha estudiado en la lección, es posible conocer las ecuaciones de las cúbicas de las que consta el spline, sea sujeto o not-a-knot. Indaga en la ayuda de MATLAB® (consulta también el apéndice A) y, usando la estructura asociada a la orden `spline`, determina la ecuación de la segunda cúbica del spline not-a-knot asociado a la tabla de datos

-2	-1	0	2	4
0	2	5	6	8

### 3. Apéndice A: Polinomios a trozos en MATLAB®

Para el diseño y evaluación de funciones polinomiales a trozos, MATLAB® incorpora varias órdenes. Las dos básicas son:

1. **mkpp** (*MaKe Piecewise Polynomial*): Permite construir una función polinómica a trozos, dada cierta colección de  $n$  nodos (puntos de separación entre los polinomios) y una matriz  $M$  de orden  $(n-1) \times (m+1)$ , donde cada fila consta de los coeficientes de cada uno de los polinomios que definen la función. Cada uno de estos polinomios es de grado menor o igual que  $m$ .

Es más, si la colección de nodos es  $x_1 < x_2 < \dots < x_n$  y, para  $j = 1, 2, \dots, n-1$ , la fila  $j$ -ésima de la matriz  $M$  viene dada por

$$[a_{jm}, \dots, a_{j1}, a_{j0}],$$

el correspondiente polinomio  $j$ -ésimo es

$$p_j(x) = a_{jm}(x - x_j)^m + \dots + a_{j1}(x - x_j) + a_{j0}.$$

Observemos que  $p_j$  está definido en  $[x_j, x_{j+1})$ , para  $j = 1, 2, \dots, n-1$ .

2. **ppval** (*Piecewise Polynomial VALuation*): Permite evaluar polinomios a trozos.

#### Ejemplo 3.1 ► Polinomio a trozos

Consideremos el polinomio a trozos

$$f(x) = \begin{cases} x^2 + 8x + 7, & x \in [-9, 0), \\ x^2 - 4x + 4, & x \in [0, 4], \\ -2x + 12, & x \in [4, 6]. \end{cases}$$

Antes de definir  $f$  en MATLAB® tenemos que expresar cada uno de los polinomios que aparecen en la función en potencias de  $x$  menos el extremo izquierdo del intervalo donde el polinomio está definido. Por ejemplo, el primer polinomio de  $f$ ,

$$p(x) = x^2 + 8x + 7,$$

debe ser escrito en potencias de  $x + 9$ . Un cálculo elemental usando el polinomio de Taylor de segundo grado nos permite asegurar que

$$\begin{aligned} p(x) &= \frac{p''(-9)}{2!}(x+9)^2 + p'(-9)(x+9) + p(-9) \\ &= (x+9)^2 - 10(x+9) + 16. \end{aligned}$$

Así, se tiene que

$$f(x) = \begin{cases} (x+9)^2 - 10(x+9) + 16, & x \in [-9, 0), \\ x^2 - 4x + 4, & x \in [0, 4], \\ -2(x-4) + 4, & x \in [4, 6]. \end{cases}$$

A continuación, introducimos  $f$  en MATLAB® mediante la orden `mkpp`:

```
Command Window
>> pp=mkpp([-9 0 4 6],[1 -10 16;1 -4 4;0 -2 4])

pp =

struct with fields:

    form: 'pp'
  breaks: [-9 0 4 6]
   coefs: [3x3 double]
  pieces: 3
   order: 3
    dim: 1
```

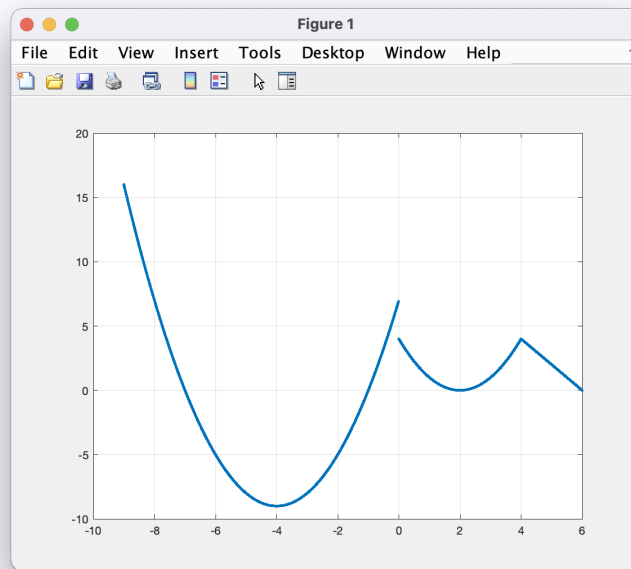
La información recogida en el dato de tipo estructura de MATLAB® `pp` tiene seis apartados:

1. la primera línea, `form: 'pp'`, nos dice que la función construida es una función polinómica a trozos (*Piecewise Polynomial*);
2. la segunda línea, `breaks: [-9 0 4 6]`, explicita los puntos de separación de las partes que definen el polinomio a trozos;
3. la tercera línea, `coefs: [4x4 double]`, nos informa de que los coeficientes de los polinomios de los que consta el polinomio a trozos se encuentran almacenados en una matriz llamada `coefs` de orden  $4 \times 4$  cuyas entradas son números escritos en doble precisión;
4. la cuarta línea, `pieces: 4`, indica que el número total de tramos del polinomio a trozos es cuatro;
5. la quinta línea, `order: 4`, informa del número de coeficientes de cada polinomio;
6. finalmente, la sexta línea, `dim: 1`, hace referencia a que el polinomio a trozos es una función de una variable.

Ahora ya estamos en condiciones de evaluar la función  $f$  con la orden `ppval`. En concreto, las siguientes instrucciones

```
Command Window
>> xs=-9:0.01:6;
>> ys=ppval(pp,xs);
>> plot(xs,ys, '.'),grid
```

generan la gráfica de la función (hemos utilizado el marcador `'.'` al ejecutar `plot` para resaltar el hecho de que la función no es continua en cero):



De los dos valores posibles, 7 y 4, que MATLAB® podría asignar al punto cero, el programa elige el que proporciona el segundo polinomio, esto es, 4. En efecto,

```
Command Window
>> ppval(pp,0)

ans =

     4
```

Por otro lado, para recuperar la información introducida al describir el polinomio a trozos basta acceder al correspondiente campo de *pp*. Por ejemplo,

```
Command Window
>> M=pp.coefs

M =

     1    -10    16
     1     -4     4
     0     -2     4

>> M=pp.breaks

M =

    -9     0     4     6
```

**Ejemplo 3.2 ► Polinomio a trozos**

Veamos ahora un polinomio a trozos que no presenta discontinuidades. En concreto, sea

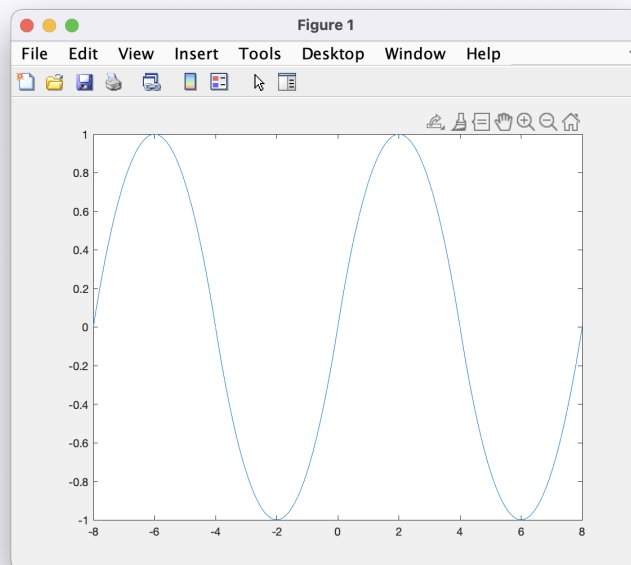
$$g(x) = \begin{cases} -\frac{1}{4}(x+8)^2 + (x+8), & x \in [-8, -4], \\ \frac{1}{4}(x+4)^2 - (x+4), & x \in [-4, 0], \\ -\frac{1}{4}x^2 + x, & x \in [0, 4], \\ \frac{1}{4}(x-4)^2 - (x-4), & x \in [4, 8]. \end{cases}$$

El siguiente código, que incorpora la orden `mkpp`,

Command Window

```
>> cc=[-1/4 1 0];  
>> pp=mkpp([-8 -4 0 4 8],[cc;-cc;cc;-cc]);  
>> xs=-8:0.1:8;  
>> plot(xs,ppval(pp,xs))
```

permite obtener la gráfica de la función:



## 4. Apéndice B: La biblioteca de funciones `fdlibm`

Tras ver en la sección 1.1 y en el apéndice A cómo trata y evalúa MATLAB® los polinomios y los polinomios a trozos, respectivamente, surge de modo natural la pregunta de cómo se evalúan en MATLAB® otras funciones elementales, como, por ejemplo, las funciones exponencial, seno, coseno, logaritmo neperiano, arcotangente, coseno hiperbólico, etc. Es decir, qué algoritmos lleva implementados, por ejemplo, el archivo de función `cos.m`, que es el que utiliza MATLAB® para evaluar el coseno de cierto número. Sin entrar en consideraciones muy técnicas, la respuesta es fácil: MATLAB® utiliza funciones polinomiales a trozos o, en algunos casos, cocientes de funciones polinomiales a trozos. En concreto, MATLAB® utiliza la biblioteca de funciones `fdlibm` (Freely Distributable LIBrary of Mathematics).

La biblioteca `fdlibm` está constituida por un conjunto de funciones matemáticas escritas en lenguaje C y muy bien adaptadas al formato IEEE de doble precisión. En la página web

[netlib.org/fdlibm](http://netlib.org/fdlibm)

se pueden consultar todas las funciones que lleva incorporadas, así como sus códigos.

En general, la implementación de algoritmos robustos y de alta calidad para evaluar funciones elementales o más genéricas (como las funciones de Bessel, por ejemplo, u otras) es una tarea bastante delicada.

### Ejemplo 4.1 ► Evaluación de la función seno

Un caso paradigmático donde se pueden apreciar las sutilidades de estos códigos es la evaluación de la función seno.

Puesto que para todo número real  $x$  se tiene la representación en serie de potencias

$$\text{sen}(x) = \sum_{n=1}^{\infty} (-1)^{n+1} \frac{x^{2n-1}}{(2n-1)!},$$

parece razonable que la estrategia para estimar el seno de un número sea tomar una suma parcial adecuada y evaluar el correspondiente polinomio. Un posible código para implementar esta estrategia podría ser:

#### Editor ► `misen.m`

```
function [s,num_sumandos]=misen(x)
s=0;
t=x;
n=1;
num_sumandos=0;
while s+t ~= s
    s=s+t;
    t=-x.^2/((n+1)*(n+2)).*t;
    n=n+2;
    num_sumandos=num_sumandos+1;
end
```

Conviene observar que el bucle de la orden `while` termina cuando  $s + t$  es igual que  $t$  a nivel de la precisión de MATLAB®.

Usando la función anterior, ejecutamos el siguiente archivo de instrucciones.

#### Editor ▶ `misenejer.m`

```
v=[1,11,21,31,41,51];
for ind=v
    x=(pi/2)*ind;
    [valor,num]=misen(x);
    er=abs(valor-sin(x));
    fprintf('x=%2d*pi/2 Error=%.5e Número sumandos=%3d \n',ind,er,num)
end
```

La primera columna es el valor del punto, la segunda, el error absoluto cometido y la tercera, el número de sumandos utilizado en la estimación.

#### Command Window

```
>> misenejer
x= 1*pi/2 Error=2.22045e-16 Número sumandos= 11
x=11*pi/2 Error=2.12873e-10 Número sumandos= 37
x=21*pi/2 Error=1.33236e-04 Número sumandos= 60
x=31*pi/2 Error=5.82102e+03 Número sumandos= 78
x=41*pi/2 Error=3.35993e+08 Número sumandos= 94
x=51*pi/2 Error=1.65181e+17 Número sumandos=106
```

Como podemos observar, estimar la función seno mediante su serie de potencias para valores de  $x$  grandes conduce a un alto coste computacional (número de sumandos) y una paulatina y creciente pérdida de precisión en el resultado. Puede comprobarse que para valores pequeños de  $x$  (esencialmente, cuando  $x \in [0, \frac{\pi}{2}]$ ) ninguno de estos fenómenos numéricos ocurre y, de hecho, pueden obtenerse estimaciones muy precisas y con pocos sumandos.

La función seno, tal como está implementada en `fdlibm` (y, por tanto, en `MATLAB®`) tiene en cuenta los comentarios anteriores. De hecho, usando distintas propiedades de dicha función (periodicidad, carácter impar, etc.), lo primero que se hace es reducir el cálculo pedido en un punto arbitrario al cálculo en el intervalo  $[0, \frac{\pi}{4}]$ . Finalmente, la función seno se estima en este intervalo mediante un polinomio de grado trece que sólo contiene potencias impares. En concreto

$$\sin(x) \approx x + c_1x^3 + c_2x^5 + c_3x^7 + c_4x^9 + c_5x^{11} + c_6x^{13},$$

donde  $c_1, c_2, c_3, c_4, c_5, c_6$  son ciertos coeficientes, cuyos valores detallaremos enseguida, que no coinciden con los del desarrollo de Taylor, pero están muy cerca de ellos.

Por tanto, tal como se comentó al principio, al evaluar la función seno en `MATLAB®` lo que estamos evaluando en última instancia es un polinomio. Por ejemplo, el siguiente archivo de instrucciones muestra, usando `format hex`, que el valor que proporciona `MATLAB®` para  $\sin(\frac{1}{2})$  se obtiene evaluando el polinomio de grado 13 anterior.

## Editor ► calsen.m

```
c1= -1.666666666666666324348e-01;  
c2=  8.33333333332248946124e-03;  
c3= -1.98412698298579493134e-04;  
c4=  2.75573137070700676789e-06;  
c5= -2.50507602534068634195e-08;  
c6=  1.58969099521155010221e-10;  
p=[c6 0 c5 0 c4 0 c3 0 c2 0 c1 0 1 0];  
v1=polyval(p,1/2);  
v2=sin(1/2);  
format hex  
disp([v1;v2])  
format("default")
```

## Command Window

```
>> calsen  
    3fdeae8744b05f0  
    3fdeae8744b05f0
```

## 5. Soluciones de los proyectos

### Proyecto 1.1

El fenómeno de Runge puede surgir<sup>7</sup> cuando se utilizan nodos de interpolación equiespaciados y polinomios interpoladores de grado alto. Se caracteriza por la aparición de oscilaciones cerca de los extremos del intervalo de interpolación. Fue descubierto por Carl Runge al explorar el comportamiento del error cuando ciertas funciones se aproximaban mediante polinomios de interpolación. El descubrimiento fue importante al poner de manifiesto que grados más altos no siempre mejoran la aproximación, lo que justifica el abandono de los polinomios como interpolantes.

Sea la función

$$f(x) = \frac{1}{1+x^2}, \quad x \in [-5, 5].$$

Vamos a calcular polinomios de interpolación que interpolen la función anterior en un número creciente de puntos. En concreto, para  $n = 6, 11, 21, 31$ , sea  $p_{n-1}$  el polinomio de interpolación asociado a  $n$  nodos equiespaciados del intervalo  $[-5, 5]$ .

El siguiente código de MATLAB® dibuja conjuntamente la función  $f$  y el polinomio de interpolación  $p_{n-1}$ , para  $n = 6, 11, 21, 31$ .

Editor ► Fenómeno de Runge

proyecto11.m

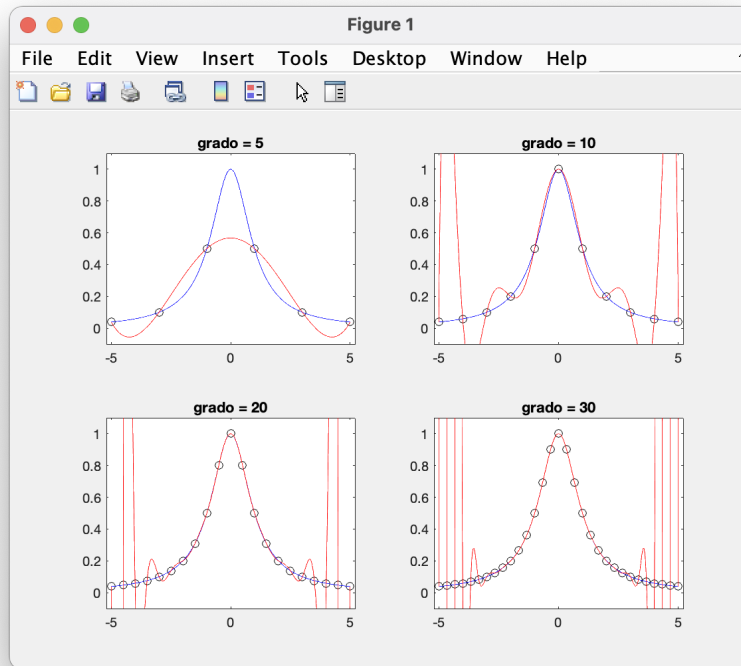
```
format short g
f=@(x)1./(1+x.^2);
i=1;
for n=[6 11 21 31]
    x=linspace(-5,5,n);
    y=f(x);
    p=polyfit(x,y,n-1);
    s=-5:0.01:5;
    t=polyval(p,s);
    subplot(2,2,i)
    plot(x,y,'ok',s,f(s),'b',s,t,'r'),shg
    axis([-5.2 5.2 -0.1 1.1])
    title(['grado = ',num2str(n-1)])
    i=i+1;
end
```

Una vez ejecutado el código en MATLAB®, observa las cuatro imágenes. Nota cómo, a medida que el grado del polinomio de interpolación crece, el ajuste mejora en la zona central del intervalo, pero presenta oscilaciones de amplitud creciente cerca de los extremos del intervalo.

<sup>7</sup>El fenómeno de Runge es imprevisible. Anticipar su aparición exigiría conocer el comportamiento del error (2), pero esto no es, en general, posible.

Command Window

&gt;&gt; proyecto11



**Proyecto 1.2**

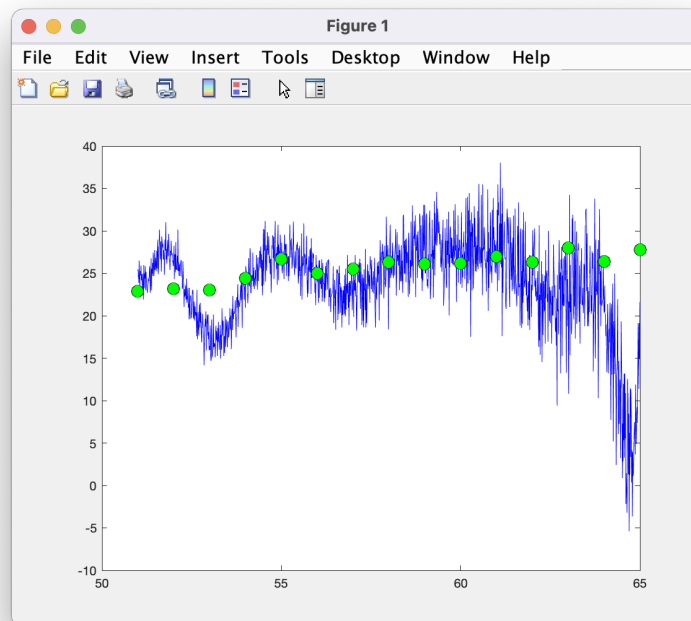
El siguiente código calcula y representa gráficamente el polinomio de interpolación.

```
Command Window
>> x=51:65;
>> y=[22.9 23.2 23 24.4 26.7 25 25.5 26.3 26.1 26.2 27 26.3 28 26.4 27.8];
>> [c,S]=polyfit(x,y,14);
Warning: Polynomial is badly conditioned. Add points with distinct
X values, reduce the degree of the polynomial, or try centering and
scaling as described in HELP POLYFIT.
> In polyfit (line 84)
>> S

S =

struct with fields:

    R: [15x15 double]
   df: 0
 normr: 16.1111
>> s=51:0.01:65;
>> t=polyval(c,s);
>> plot(x,y,'ok',s,t,'b','MarkerEdgeColor','k','MarkerFaceColor','g',...
'MarkerSize',10),shg
```



Hemos usado la orden `polyfit` con dos argumentos de salida. El primero, `c`, nos da los coeficientes del polinomio de interpolación y el segundo, `S`, es una estructura asociada al proceso de ajuste. En concreto, la matriz `R` que aparece en la estructura `S` corresponde a la matriz  $R$  de la factorización  $QR$  de la matriz  $V$  de Vandermonde asociada a los nodos  $\{51, 52, \dots, 65\}$ . El valor `df` es el denominado grado de libertad (*degree freedom*) del ajuste polinómico que, con nuestra notación, es exactamente  $M - n$  (en este caso es cero, pues  $M = n$ ). Finalmente, `normr` es la norma residual (euclídea) del ajuste en los nodos `y`, teóricamente, debería ser cero.

Ahora bien,

```
Command Window
>> cond(S.R)

ans =

    8.4128e+39
```

Como podemos ver, el número de condición de  $R$  (y por tanto el de la matriz  $V$ ) es desmesurado. Esto explica los malos resultados obtenidos en el cálculo del polinomio de interpolación, reflejados en el gráfico anterior.

Sin embargo, si utilizamos `polyfit` de la siguiente manera

```
Command Window
>> [p,S,mu]=polyfit(x,y,14);
>> cond(S.R)

ans =

    2.5356e+06

>> S.normr

ans =

    1.6280e-11
```

obtenemos que, ahora, la matriz  $R$  tiene un número de condición muchísimo más bajo y la norma residual (si bien no nula) es realmente aceptable. Es más, MATLAB® ni siquiera muestra ya ningún tipo de aviso (*Warning*). La explicación está en que cuando ejecutamos `polyfit` con el tercer argumento de salida, `mu`, la interpolación polinómica se realiza tras un proceso de centrado y escalado de los nodos. Este proceso no sólo permite un cálculo más eficiente de los coeficientes del polinomio de interpolación, sino que mejora notablemente las propiedades numéricas de dicho polinomio.

En concreto, para la tabla (1), MATLAB® determina el polinomio de interpolación expresado de la siguiente forma:

$$p(x) = a_n \left( \frac{x - \mu_1}{\mu_2} \right)^n + a_{n-1} \left( \frac{x - \mu_1}{\mu_2} \right)^{n-1} + \dots + a_1 \left( \frac{x - \mu_1}{\mu_2} \right) + a_0,$$

donde

$$\mu_1 = \frac{1}{n+1} \sum_{j=0}^n x_j, \quad \mu_2 = \sqrt{\frac{1}{n} \sum_{j=0}^n (x_j - \mu_1)^2}$$

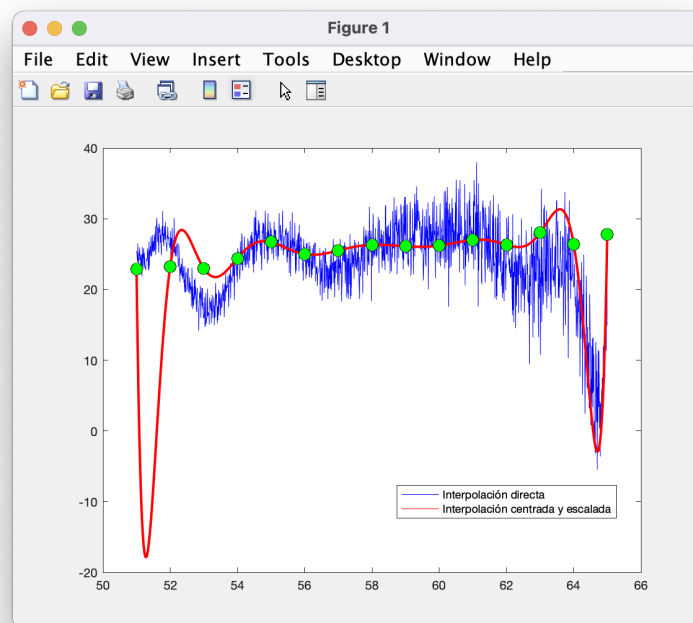
son, respectivamente, la media aritmética de los nodos y la desviación típica estadística. Estos dos valores son las dos componentes del vector  $\mu$ .

A partir de los datos anteriores,  $x$  e  $y$ , el siguiente archivo de MATLAB® genera un gráfico donde puede verse con claridad lo diferente que puede ser, desde un punto de vista numérico, abordar un ajuste directo o abordarlo tras realizar un proceso de centrado y escalado.

Editor ► Mala condición

malacond.m

```
x=51:65;
y=[22.9 23.2 23 24.4 26.7 25 25.5 26.3 26.1 26.2 27 26.3 28 26.4 27.8];
c1=polyfit(x,y,14);
[c2,S,mu]=polyfit(x,y,14);
s=51:0.01:65;
t1=polyval(c1,s);
t2=polyval(c2,s,S,mu);
plot(s,t1,'b',s,t2,'r'),shg
hold on
plot(s,t2,'r','LineWidth',2),axis([50 66 -20 40]),shg
plot(x,y,'ok','MarkerEdgeColor','k','MarkerFaceColor','g','MarkerSize',10)
legend('Interpolación directa','Interpolación centrada y escalada',...
'Location','best')
hold off
```



### Proyecto 2.1

Vamos a interpolar la función  $f(x) = \cos(x)$  en el intervalo  $[0, 3]$  mediante *splines not-a-knot* asociados a un número creciente de puntos de interpolación. En concreto, para  $n = 10, 100, 1000, 10000$ , tomaremos  $n + 1$  nodos de interpolación equiespaciados en el intervalo. El objetivo es observar la evolución del error, evaluado como la máxima separación que se produce entre la función interpolada y el *spline not-a-knot* sobre los puntos de una partición fina del intervalo  $[0, 3]$ .

#### Editor ► *Spline not-a-knot* y error

proyecto21.m

```
s=0:0.001:3;
n=10;
for i=1:4
    x=linspace(0,3,n+1);
    h=3/n;
    y=cos(x);
    t=spline(x,y,s);
    err=max(abs(cos(s)-t));
    fprintf('n = 10^%1d  h = %.4e  error = %.4e \n',i,h,err)
    n=10*n;
end
```

Ejecutamos en MATLAB® el código anterior:

#### Command Window

```
>> proyecto21
n = 10^1  h = 3.0000e-01  error = 2.1774e-04
n = 10^2  h = 3.0000e-02  error = 2.2878e-08
n = 10^3  h = 3.0000e-03  error = 2.2767e-12
n = 10^4  h = 3.0000e-04  error = 1.1102e-16
```

Fíjate en dos fenómenos:

1. El primero es la convergencia. Observa cómo a medida que el número  $n$  de cúbicas crece, el error disminuye. Esto significa que podemos aproximar una función por medio de un *spline not-a-knot* con todo el nivel de precisión que queramos, sin otro límite que el que impone la precisión de la máquina.
2. El segundo es el orden de convergencia, esto es, la «rapidez» con la que el error se aproxima a cero. Nota cómo al dividir  $h$  (la distancia constante entre nodos consecutivos) por 10, el error se divide (aproximadamente) por  $10^4$ . En consecuencia, el error es del orden de  $h^4$ , cuando  $h$  tiende a cero, o, dicho con otras palabras, la convergencia de la aproximación por *splines not-a-knot* es de orden 4.

**Proyecto 2.2**

En primer lugar, introducimos en MATLAB® los nodos y valores de interpolación, separados en sendos vectores:

```
Command Window
```

```
>> x=[-2 -1 0 2 4];  
>> y=[0 2 5 6 8];
```

Si ejecutamos en MATLAB® la orden `spline`, pero no incluimos el tercer argumento de entrada (este argumento indica el punto o los puntos donde deseamos evaluar el *spline*), MATLAB® crea una «estructura» que recoge, entre otras cosas, los coeficientes de las cúbicas que componen el *spline*.

Vamos a crear la estructura asociada al *spline not-a-knot* (el procedimiento para la creación de la estructura asociada a un *spline* sujeto sería análogo):

```
Command Window
```

```
>> S=spline(x,y);
```

Si escribimos en MATLAB® el nombre de la estructura, `S`, el programa devuelve cierta información sobre su contenido:

```
Command Window
```

```
>> S =  
  
struct with fields:  
  
    form: 'pp'  
  breaks: [-2 -1 0 2 4]  
   coefs: [4x4 double]  
  pieces: 4  
   order: 4  
    dim: 1
```

La primera línea, `form: 'pp'`, nos dice que la función construida es una función polinómica a trozos (*Piecewise Polynomial*); la segunda línea, `breaks: [-2 -1 0 2 4]`, explicita los puntos de separación de las partes que definen la función a trozos (de hecho, se trata de los nodos de interpolación); la tercera línea, `coefs: [4x4 double]`, nos informa de que los coeficientes de las cúbicas de las que consta el *spline not-a-knot* se encuentran almacenados en una matriz llamada `coefs` de orden  $4 \times 4$  y cuyas entradas son números escritos en doble precisión; la cuarta línea, `pieces: 4`, indica que el número total de cúbicas del *spline not-a-knot* es cuatro; la quinta línea, `order: 4`, informa del número de coeficientes de cada cúbica; finalmente, la sexta línea, `dim: 1`, hace referencia a que el *spline not-a-knot* es una función de una variable.

La información más importante contenida en la estructura `S` es la matriz `coefs`. Esta matriz contiene, como ya hemos indicado, los coeficientes de las cúbicas que componen el

*spline not-a-knot*. Para acceder a esta información debemos ejecutar en MATLAB® la orden `S.coefs`:

```
Command Window
>> format
>> c=S.coefs

c =

   -0.6250    2.3750    0.2500         0
   -0.6250    0.5000    3.1250    2.0000
    0.2500   -1.3750    2.2500    5.0000
    0.2500    0.1250   -0.2500    6.0000
```

El *spline not-a-knot* consta de cuatro cúbicas. La matriz anterior contiene, fila a fila, los coeficientes de cada cúbica, de manera que la primera fila de la matriz contiene los coeficientes de la primera cúbica, es decir, la que está definida entre los nodos primero,  $x_1 = -2$ , y segundo,  $x_2 = -1$ , y así sucesivamente con el resto de las cúbicas.

Hemos de fijarnos en dos hechos importantes sobre la forma en la que MATLAB® muestra los coeficientes de las cúbicas:

1. Los coeficientes aparecen siempre escritos de mayor a menor potencia, léidos de izquierda a derecha.
2. Si el *spline* (sea sujeto o *not-a-knot*) está asociado, en general, a los nodos de interpolación  $x_i$ , para  $i = 1, 2, \dots, n$ , la fila  $i$ -ésima de la matriz corresponde a los coeficientes de la cúbica  $i$ -ésima escrita en potencias de  $x - x_i$ .

Por ejemplo, la segunda cúbica del *spline not-a-knot*,  $S_2$ , cuya ecuación se pide determinar, está definida entre los nodos  $x_2 = -1$  y  $x_3 = 0$ . Dicha ecuación se expresa, por lo tanto, en potencias de  $x - x_2 = x + 1$ . Accedemos a sus coeficientes extrayendo en MATLAB® la segunda fila de la matriz anterior:

```
Command Window
>> c(2,:)

ans =

   -0.6250    0.5000    3.1250    2.0000
```

Por tanto, su ecuación viene dada por

$$\begin{aligned} S_2(x) &= c(2,1)(x - x_2)^3 + c(2,2)(x - x_2)^2 + c(2,3)(x - x_2) + c(2,4) \\ &\approx -0.6250(x + 1)^3 + 0.5000(x + 1)^2 + 3.1250(x + 1) + 2.0000. \end{aligned}$$

Observa que, al tratarse de un *spline not-a-knot*, las cúbicas primera y segunda (igualmente la penúltima y la última) son idénticas. Sin embargo, sus coeficientes son distintos. Esto se debe, evidentemente, a que se expresan en potencias que tienen bases diferentes.