PRÁCTICA 7: Interpolación

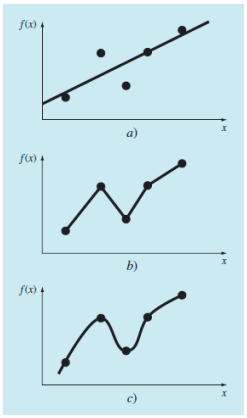
7.1 Resumen de Teoría

7.1.1 Introducción. Ajuste de Curvas

Uno de los problemas más habituales que se plantea en Cálculo Numérico es el de sustituir una función por otra cuyas características se consideren más convenientes para su posterior utilización. Este es el problema llamado de **ajuste de curvas**. La función objeto de estudio puede ser conocida explícitamente, o bien puede venir dada mediante una tabla de valores. Pretendemos que la nueva función "aproxime" a aquella que va a sustituir, existiendo dos criterios muy diferentes para ello:

- **Interpolación**: Cuando queremos que la nueva función coincida con la dada en un conjunto finito de datos.
- **Regresión por Mínimos Cuadrados**: Cuando buscamos que la diferencia sea, en media, lo más pequeña posible.

El ingeniero debe elegir entre estos dos métodos generales para el ajuste de curvas según la cantidad de error asociado con los datos. Si se sabe que los datos son muy precisos, el procedimiento básico será la interpolación, buscando una curva o una serie de curvas que pasen por cada uno de los puntos en forma directa. Pero si los datos exhiben un grado significativo de error o "ruido", la estrategia será la regresión por mínimos cuadrados, obteniendo una sola curva que represente la tendencia general de los datos.



En la figura vemos tres intentos para ajustar una curva con cinco puntos dados. *a*) Regresión por mínimos cuadrados, *b*) interpolación lineal y *c*) interpolación curvilínea.

Dedicaremos esta práctica al estudio del problema de interpolación eligiendo como funciones aproximantes a los polinomios. Los polinomios son las funciones más usadas ya que son fáciles de evaluar a través de la regla de Horner y su almacenamiento en el ordenador queda reducido a un vector. Además, su integral y su derivada son fácilmente calculables a partir de dicho vector.

7.1.2 Interpolación

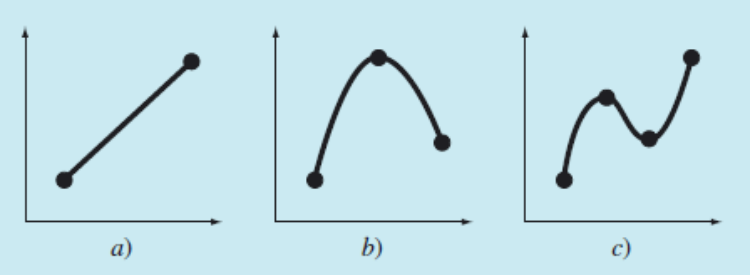
El problema clásico de interpolación es el siguiente: dada una tabla con *n* + 1 puntos, hallar un polinomio del menor grado posible que pase por todos ellos. Buscaremos, en general, la siguiente aproximación polinomial:

$$p_n(x) = a_0 + a_1x + a_2x^2 + \ldots\ldots + a_ix^i + \ldots\ldots + a_nx^n$$

Se puede probar el siguiente resultado:

Teorema. *Dados n + 1 puntos, con abscisas distintas entre sí, existe uno y sólo un polinomio de grado a lo más n que pasa por estos puntos.*

Por ejemplo, hay sólo una línea recta que une dos puntos. De manera similar, únicamente una parábola une un conjunto de tres puntos. La interpolación polinomial consiste en determinar el polinomio único de *n*-ésimo grado que se ajuste a *n* + 1 puntos.



Aunque hay uno y sólo un polinomio de *n*-ésimo grado que se ajusta a *n* + 1 puntos, existe una gran variedad de formas matemáticas en las cuales puede expresarse este polinomio. En esta práctica describiremos dos alternativas que son muy adecuadas para implementarse en un ordenador: los polinomios de Newton y de Lagrange.

7.1.3 Polinomio de Interpolación de Newton

Comenzamos presentando los polinomios llamados de Newton. Mediante un proceso recursivo, cada $p_k(x)$ se obtiene simplemente añadiendo un término a $p_{k-1}(x)$, y al final del proceso se calcula $p_n(x)$ formado por una suma de términos:

$$p_k(x) = c_0 + c_1(x - x_0) + c_2(x - x_0)(x - x_1) + \cdots + c_k(x - x_0)\ldots(x - x_{k-1})$$

Es decir:

$$p_k(x) = \sum_{i=0}^{k} c_i \prod_{j=0}^{i-1}(x - x_j)$$

donde se considera que $\prod_{j=0}^{m}(x - x_j) = 1$, siempre que m<0.

Así, los tres primeros polinomios de interpolación de Newton son:

$p_0(x) = c_0$
$p_1(x) = c_0 + c_1(x - x_0)$
$p_2(x) = c_0 + c_1(x - x_0) + c_2(x - x_0)(x - x_1)$

Los puntos asociados con los datos se utilizan para evaluar los coeficientes: $c_0, c_1, \dots, c_n$ mediante las llamadas diferencias divididas finitas.
Es decir, dada esta tabla de valores conocidos:

Punto	0	1	.....	*i*	.....	*n*
x	x_0	x_1	.....	x_i	.....	x_n
$f(x)$	$f(x_0)$	$f(x_1)$	.....	$f(x_i)$	.....	$f(x_n)$

los valores de los coeficientes se calculan como:

$c_0 = f[x_0]$
$c_1 = f[x_0, x_1]$
$c_2 = f[x_0, x_1, x_2]$
…
$c_n = f[x_0, \dots, x_{n-1}, x_n]$

La primera diferencia dividida de *f*(*x*), respecto a los valores de x_0 y x_1, se denota:

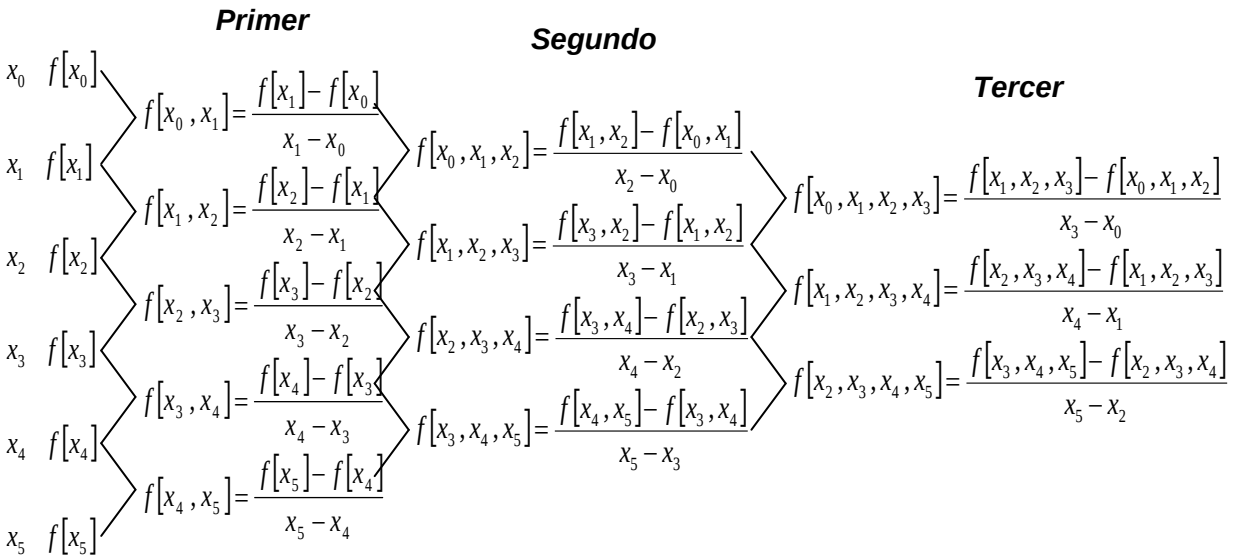
$$f[x_0, x_1] = \frac{f(x_1) - f(x_0)}{x_1 - x_0}$$

Y en general:

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

A $f[x_i]$ se le llama diferencia dividida de orden cero.

La siguiente tabla representa el esquema a seguir, indicando el orden:



Estas diferencias sirven para evaluar los coeficientes, los cuales se sustituirán en la ecuación para obtener el polinomio de interpolación que se conoce como *polinomio de interpolación de Newton en diferencias divididas*. Debe observarse que no se requiere que los datos utilizados estén igualmente espaciados o que los valores de la abscisa estén en orden ascendente.

7.1.4 Polinomio de Interpolación de Lagrange

El *polinomio de interpolación de Lagrange* es simplemente una reformulación del polinomio de Newton que evita el cálculo de las diferencias divididas, y se representa como:

$$P_n(x) = L_0(x) f(x_0) + L_1(x) f(x_1) + ... + L_i(x) f(x_i) + ... + L_n(x) f(x_n)$$

en donde:

$$L_0(x) = \frac{(x - x_1)(x - x_2)...(x - x_i)...(x - x_n)}{(x_0 - x_1)(x_0 - x_2)...(x_0 - x_i)...(x_0 - x_n)}$$

$$L_1(x) = \frac{(x - x_0)(x - x_2)...(x - x_i)...(x - x_n)}{(x_1 - x_0)(x_1 - x_2)...(x_1 - x_i)...(x_1 - x_n)}$$

$$\vdots$$

$$L_i(x) = \frac{(x - x_0)(x - x_1)...(x - x_{i+1})...(x - x_n)}{(x_i - x_0)(x_i - x_1)...(x_i - x_{i+1})...(x_i - x_n)}$$

En general el polinomio se puede escribir:

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

en donde:

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

La aproximación polinomial de Lagrange, es la combinación lineal de los $f(x_i)$ y de los coeficientes $L_i(x)$. Por ejemplo, para grado 1, es decir aproximando por una recta, es:

$$P(x) = \frac{(x - x_1)}{(x_0 - x_1)} f(x_0) + \frac{(x - x_0)}{(x_1 - x_0)} f(x_1)$$

$$P(x) = L_0(x) f(x_0) + L_1(x) f(x_1)$$

y para grado 2 es:

$$P_2(x) = L_0(x) f(x_0) + L_1(x) f(x_1) + L_2(x) f(x_2)$$

en donde:

$$L_0(x) = \frac{(x - x_1)(x - x_2)}{(x_0 - x_1)(x_0 - x_2)}$$

$$L_1(x) = \frac{(x - x_0)(x - x_2)}{(x_1 - x_0)(x_1 - x_2)}$$

$$L_2(x) = \frac{(x - x_0)(x - x_1)}{(x_2 - x_0)(x_2 - x_1)}$$

La forma de Lagrange resulta especialmente útil cuando trabajamos con un número fijo de puntos. Pero si se quieren introducir nuevos puntos de interpolación (por ejemplo, para mejorar la exactitud), al aplicar la forma de Lagrange hay que rehacer todos los cálculos. En este caso es más adecuado utilizar otras estrategias, como por ejemplo, la forma de Newton del polinomio de interpolación.

7.1.5 Error del polinomio interpolador

En muchos casos, los datos $y_i = f(x_i)$ son los valores de una función $f(x)$ que se desea aproximar. Evidentemente, al hacer $f(x) \cong p_n(x)$, estaremos cometiendo un error, salvo que *x* sea uno de los nodos. En realidad, se tendrá:

$$f(x) = p_n(x) + E_n(x)$$

siendo $E_n(x)$ el error cometido, en principio, desconocido. El siguiente teorema nos proporciona una expresión del error que depende la función a interpolar y de los nodos utilizados en el proceso de interpolación.

Teorema. *Si* $f \in C^{n+1}[a,b]$ *y todos los* $x_i \in [a,b]$, *entonces existe un* $c \in [a,b]$ *tal que:*

$$E_n(x) = \frac{f^{(n+1)}(c)}{(n+1)!} \prod_{i=0}^{n} (x - x_i)$$

Este teorema no nos permite encontrar el error de manera exacta, ya que no nos dice cómo calcular el valor de *c*, sin embargo, nos va a permitir acotar el error.

Por un lado, para la derivada $f^{(n+1)}(c)$, por el Teorema de Weierstrass, se tiene que cumplir:

$$|f^{(n+1)}(c)| \leq M^{n+1}$$

Por otro lado está la expresión: $\prod_{i=0}^{n} |x - x_i|$, que en el caso particular de nodos equiespaciados, con:

$$h = \frac{b-a}{n} \; ; \; x_i = a + (i-1)h; \;\; \text{para } i = 0, 1, \ldots, \text{n}$$

nos lleva a que:

$$E_1(x) \leq \frac{h^2}{8} M^2; \;\; E_2(x) \leq \frac{h^3}{9\sqrt{3}} M^3; \;\; E_3(x) \leq \frac{h^4}{24} M^4; \;\; \ldots$$

Pero en general, M^{n+1} va a crecer más rápido de lo que decrece h^n y aparece el fenómeno de oscilación polinomial.

Si nuestro objetivo es reducir el error cometido en el proceso de interpolación, se abre ahora un abanico de posibilidades:

1. **Cambiar los nodos**. Buscar nodos no equiespaciados que hagan más pequeña la cantidad $\prod_{i=0}^{n} |x - x_i|$. Nodos de Chebyshev.

2. **No usar polinomios de grado muy alto.** Por ejemplo mediante la Interpolación polinomial a trozos.

3. **Usar otro tipo de funciones.** Como funciones trigonométricas: análisis de Fourier.

7.2 Cálculo directo del polinomio de interpolación

Como es sabido, dados los valores de una función $f(x)$, llamados $f(x_i)$ en $n + 1$ puntos distintos $(x_0, x_1, \ldots, x_{n-1}, x_n)$ del intervalo $[a, b]$, existe un único polinomio $p_n(x)$ de grado menor o igual a n tal que $p_n(x_i) = f(x_i)$ para cada $i = 0, 1, 2, \ldots, n$. A dicho polinomio se le llama el polinomio de interpolación de f en los $n + 1$ puntos dados.

Puede escribirse como:

$$p_n(x) = a_0 + a_1 x + a_2 x^2 + \ldots\ldots + a_i x^i + \ldots\ldots + a_n x^n$$

y hallar los coeficientes $a_0, a_1, a_2, \ldots a_n$ del polinomio de interpolación de manera que se cumplan las $n + 1$ ecuaciones lineales:

$$p_n(x_i) = f(x_i)\,,\ i = 0, 1, 2, \ldots, n$$

siendo el sistema correspondiente compatible determinado, pues el determinante de la matriz de coeficientes es un determinante de Vandermonde, que no se anula por ser los puntos distintos, aunque sabemos que la matriz está mal condicionada, de ahí el interés en otros métodos para obtener el polinomio de interpolación.

Pero veamos algún ejemplo con este método.

Se desea obtener el polinomio de interpolación que pasa por los puntos:

(0, 1); (1, 5); (2, 31); (3, 121); (4, 341)

```
(%i1)    kill(all)$
```

Utilizaremos la notación:

```
(%i2)    x:[0,1,2,3,4];
         y:[1,5,31,121,341];
(x)      [0,1,2,3,4]
  (y)      [1,5,31,121,341]
```

Y llamaremos z a la variable independiente del polinomio interpolador $p(z)$, para de esta forma no confundir con el vector x de abscisas.

Como se trata de 5 puntos por ellos pasa un único polinomio de grado, a lo más, cuatro de la forma:

$$p_4(z) = a_0 + a_1 z + a_2 z^2 + a_3 z^3 + a_4 z^4$$

Para hallarlo hemos de resolver el sistema lineal:

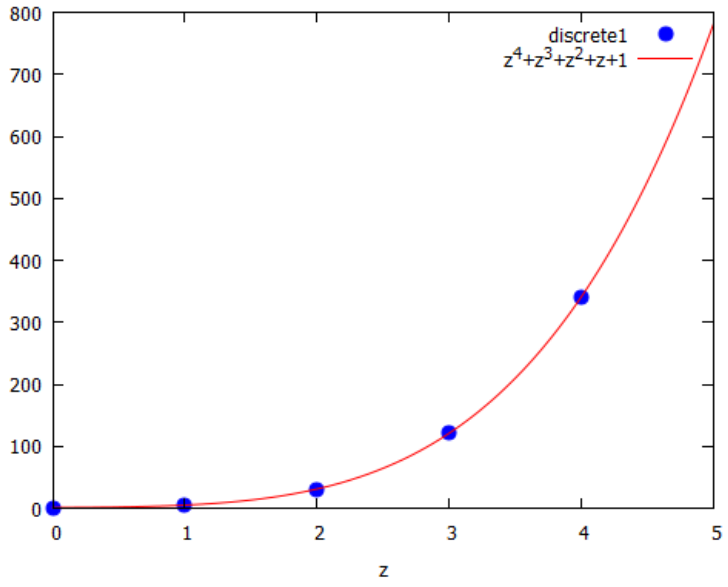
```
(%i4)    p(z):=a0+a1*z+a2*z^2+a3*z^3+a4*z^4;
         sol:linsolve([p(0)=1,p(1)=5,p(2)=31,p(3)=121,p(4)=341],
             [a0,a1,a2,a3,a4]);
(%o3)    p(z):=a0+a1 z+a2 z^2+a3 z^3+a4 z^4
 (sol)     [a0=1,a1=1,a2=1,a3=1,a4=1]
```

Hemos hallado los coeficientes, pero debido a la forma de devolver los resultados **solve**, para capturar estos valores e introducirlos en *p*(*z*) aún debemos hacer:

```
(%i6)    p4(z):=ev(p(z),sol)$
         p4(z);
(%o6)    z^4+z^3+z^2+z+1
```

Ahora vamos a dibujar juntos el polinomio interpolador y los puntos dados.

```
(%i7)    plot2d([[discrete,x,y], p4(z)], [z,0,5],
             [style,points,lines])$
```



Para dibujar los puntos usamos la opción **discrete** incluida en **plot2d**. El **style points** me indica que dibuje los puntos sin unirlos entre sí, y **lines** que dibuje la línea. Como era de esperar el polinomio de grado 4 pasa por los 5 puntos.

Pero como comentamos, al hallar el polinomio interpolador con éste método, la matriz de coeficientes del sistema es una Matriz de Vandermonde (una matriz que presenta una progresión geométrica en cada fila). El número de condición de estas matrices es alto y además crece con la dimensión.

Usaremos el comando **coefmatrix** que nos ahorra tiempo al darnos de forma directa la matriz de coeficientes de un sistema. En este caso tenemos:

```
(%i8)    A:coefmatrix([p(0)=1,p(1)=5,p(2)=31,p(3)=121,p(4)=341],
             [a0,a1,a2,a3,a4]);
```

(A)
$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 2 & 4 & 8 & 16 \\ 1 & 3 & 9 & 27 & 81 \\ 1 & 4 & 16 & 64 & 256 \end{bmatrix}$$

```
(%i9)    mat_cond(A,1);
0 errors, 0 warnings
  (%o9)    3540
```

Y de hecho, basta un incremento de una unidad en el valor de la *y* del último punto (pasar de 341 a 342 debido, por ejemplo, a un error en una medida) para provocar esta variación en la solución.

```
(%i11)   p(z):=a0+a1*z+a2*z^2+a3*z^3+a4*z^4;
         sol:linsolve([p(0)=1,p(1)=5,p(2)=31,p(3)=121,p(4)=342],
             [a0,a1,a2,a3,a4]);
```

(%o10) $p(z):=a0+a1\,z+a2\,z^2+a3\,z^3+a4\,z^4$

(sol) $[a0=1,a1=\frac{3}{4},a2=\frac{35}{24},a3=\frac{3}{4},a4=\frac{25}{24}]$

7.3 Fórmula de Newton en diferencias divididas

Debido al fenómeno anterior interesa buscar el polinomio de interpolación, pero sin necesidad de resolver un sistema. Existe una forma sencilla de calcular el polinomio de interpolación a través de las llamadas diferencias divididas. La fórmula de Newton, en diferencias divididas, del polinomio de interpolación está dada por:

$$p_n(x) = c_0 + c_1(x - x_0) + c_2(x - x_0)(x - x_1) + \cdots + c_n(x - x_0)\ldots(x - x_{n-1})$$

$$c_0 = f[x_0];\ c_1 = f[x_0, x_1];\ c_2 = f[x_0, x_1, x_2],\ \ldots\ c_n = f[x_0, \ldots, x_{n-1}, x_n]$$

La tabla que vimos representa el esquema a seguir para construir las diferencias divididas.

Lo vamos a aplicar al problema anterior. Veamos un bloque que permite de forma sencilla calcularlas. Llamaremos *h* a la matriz de diferencias divididas. Su elemento *h*[*i*, *j*] representa la diferencia de orden *j*, que comienza por x_i. Por ejemplo:

$$h[0,1] = f[x_0, x_1];\ h[1,1] = f[x_1, x_2];\ h[1,2] = f[x_1, x_2, x_3]$$

Debido a cuestiones de programación (por almacenar en una matriz), no podemos utilizar las diferencias de orden cero *h*[*i*, 0], como gusta hablar en teoría. Estas serán simplemente los valores de $f(x_i)$. Por ello nuestro contador comienza en 1. Además se crea la matriz *h* con el número de ceros necesarios para que alcance la misma longitud que una columna de la matriz de entrada.

```
(%i1)    kill(all)$
```

```
(%i2)   x:[0,1,2,3,4]$
        y:[1,5,31,121,341]$
(%i3)   /* Función difdiv(x,y) */
        /* Calcula las diferencias divididas
           con la fórmula de Newton */
        /* ARGUMENTOS DE ENTRADA: */
        /* x ..... Vector de valores de xi */
        /* y ..... Vector de Valores de f(xi) */
        /* ARGUMENTOS DE SALIDA: */
        /* h(i,j) ... diferencias divididas */
        difdiv(x,y):=block(
            n:length(x),
            h:zeromatrix(n,n-1),
            for i:1 thru n-1 do(
                h[i,1]:(y[i+1]-y[i])/(x[i+1]-x[i]),
                print('h[i,1]=h[i,1])
                ),
            for j:2 thru n-1 do(
                for i:1 thru n-j do(
                h[i,j]:(h[i+1,j-1]-h[i,j-1])/(x[i+j]-x[i]),
                print('h[i,j]=h[i,j])
                )
            )
            )$
```

Para obtener la notación con la que estamos más habituados llamaremos *m* a una matriz de 2 columnas y *n* filas, que contiene en cada una de las *n* filas los valores $(x_i, f(x_i))$. Añadimos a la matriz de trabajo *m* la lista obtenida con las diferencias divididas como una nueva columna.

```
(%i6)   m:transpose(matrix(x,y))$
        addcol(m,h);
```

$$\begin{bmatrix} 0 & 1 & 4 & 11 & 7 & 1 \\ 1 & 5 & 26 & 32 & 11 & 0 \\ 2 & 31 & 90 & 65 & 0 & 0 \\ 3 & 121 & 220 & 0 & 0 & 0 \\ 4 & 341 & 0 & 0 & 0 & 0 \end{bmatrix} \quad \text{(\%o6)}$$

Ahora para construir el polinomio, en este caso, hemos optado por llamar *z* a la variable del polinomio, para no coincidir con las abscisas de los puntos, llamadas antes *x*.

```
(%i8)   p(z):=y[1]+sum(h[1,j]*product(z-x[i],i,1,j),j,1,n-1);
        expand(p(z));
```

$$p(z) := y_1 + \sum_{j=1}^{n-1} h_{1,j} \prod_{i=1}^{j} z - x_i \quad \text{(\%o7)}$$

$$z^4 + z^3 + z^2 + z + 1 \quad \text{(\%o8)}$$

7.4. Fórmula de Lagrange

Veamos ahora una fórmula para calcular el polinomio de interpolación: la forma de Lagrange. Se utiliza sobre todo cuando el grado del polinomio se conoce *a priori*. En general el polinomio se puede escribir:

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

en donde:

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

Los polinomios $L_i(x)$ se llaman *polinomios de Lagrange* de grado *n* en los puntos x_0, x_1, ... , x_n. Tienen una propiedad muy interesante. Si evaluamos en el mismo índice $L_i(x_i) = 1$, ya que numerador y denominador coinciden. Si evaluamos en x_k con $k \neq i$, entonces $L_i(x_k) = 0$, ya que uno de los factores del numerador, en concreto cuando $j = k$, se anula.

En el siguiente bloque calculamos estos polinomios $L_i(x)$.

Debemos señalar que no podemos utilizar la notación tal y como aparece en la teoría. En teoría tenemos *n*+1 puntos y el contador va de 0 a n. Con nuestra notación los puntos son *n* (la longitud del vector *x*) y por eso nuestro contador va de 1 a *n*.

```
(%i1)    kill(all)$
(%i1)    /* Función lag(x,y) */
         /* Calcula el polinomio p(z) de Lagrange */
         /* ARGUMENTOS DE ENTRADA: */
         /* x ..... Vector de valores de xi */
         /* y ..... Vector de valores de yi */
         /* ARGUMENTOS DE SALIDA: */
         /* p(z) ... polinomio de Lagrange */
         lag(x,y):= block(
             n:length(x),
             l:makelist(1,i,1,n),
                for j:1 thru n do(
                   for i:1 thru n do(
                   if i # j then
                   l[j]:l[j]*((z-x[i])/(x[j]-x[i]))
                    )),
             define(p(z),sum(l[i]*y[i],i,1,n)),
             print("El polinomio int. de Lagrange es:
         p(z)=",expand(p(z)),"")
             )$
```

De nuevo para el ejemplo del párrafo anterior se tendrá:

```
(%i3)    x:[0,1,2,3,4]$
         y:[1,5,31,121,341]$
```

```
(%i4)    lag(x,y);
El polinomio int. de Lagrange es: p(z)=   z^4+z^3+z^2+z+1
 (%o4)
```

Para construir el polinomio de nuevo hemos optado por llamar *z* a la variable del polinomio, para no coincidir con las abscisas de los puntos, llamadas *x*.

Observemos que para definir el polinomio a partir de las expresiones de los *l*[*i*], hemos usado el comando **define**. Si utilizamos para definirlo el operador **:=** nos devuelve el valor esperado. Pero si luego intentamos valorar el polinomio tendremos una sorpresa.

```
(%i7)    p(z):=sum(l[i]*y[i],i,1,n)$
         expand(p(z));
(%o7)    z^4+z^3+z^2+z+1
(%i8)    p(1);
(%o8)    341 (z-3) (z-2) (z-1) z / 24 + 121 (4-z) (z-2) (z-1) z / 6 - ...
```

Para finalizar este apartado indicar que en *Maxima* disponemos del paquete **interpol** que calcula el polinomio interpolador de Lagrange. Vamos a usarlo para comprobar nuestros resultados. En primer lugar cargamos el módulo**:**

```
(%i9)    load(interpol)$
```

y ya podemos usar la orden **lagrange** para calcular el polinomio que interpola una lista de pares de la forma: (nodo, valor).

```
(%i11)   lagrange([[0,1],[1,5],[2,31],[3,121],[4,341]])$
         expand(%);
(%o11)   x^4+x^3+x^2+x+1
```

Obsérvese que la forma de introducir los datos es distinta de la usada hasta ahora. Profundizaremos en este paquete en la siguiente práctica.

El polinomio de interpolación tiene el problema de que, si hay muchos puntos es de grado muy alto y oscila mucho. Es el conocido como fenómeno de oscilación polinomial. Veámoslo en la siguiente sección con más detalle.

7.5 Error del polinomio de interpolación

Supongamos ahora que los datos $y_i = f(x_i)$ son los valores de una función $f(x)$ que se desea aproximar en un intervalo $[a, b]$.

Ya vimos en el resumen de teoría que en que en el caso particular de nodos equiespaciados, el error $E_n(x)$ cometido al hacer $f(x) \cong p_n(x)$ es:

$$E_1(x) \leq \frac{h^2}{8}M^2;\ \ E_2(x) \leq \frac{h^3}{9\sqrt{3}}M^3;\ \ E_3(x) \leq \frac{h^4}{24}M^4;\ \ \ldots$$

donde M^{n+1} es una cota de la derivada:

$$|f^{(n+1)}(c)| \leq M^{n+1}$$

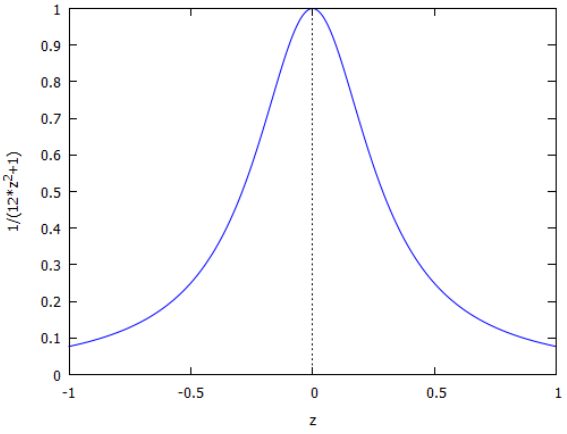
Si aumentamos el número de puntos lo que ocurre, en general, es que M^{n+1} va a crecer más rápido de lo que decrece $h = \frac{b-a}{n}$ y aparece el fenómeno de oscilación polinomial. Este efecto hace que la interpolación de Lagrange sea una técnica poco usada para problemas reales de ajuste de curvas.

Veamos un ejemplo. Consideremos la llamada función de Runge:

$$f(x) = \frac{1}{1+12x^2}$$

En este caso no partimos de una tabla de valores sino que conocemos la expresión de la función. Para realizar la interpolación de dicha función y poder aproximarla mediante polinomios, vamos a valorarla en una serie de puntos (nodos). En primer lugar vamos a aproximarla con 11 nodos equiespaciados en [-1, 1]. Usaremos nuestra función **lag.** El resultado es el siguiente.

```
(%i1)    kill(all)$
(%i1)    f(z):=1/(1+12*z^2)$
(%i2)    plot2d(f(z),[z,-1,1])$
```



Para construir los nodos usamos el comando **makelist**. Como vemos las abscisas *x* se construyen entre los límites del intervalo *a* y *b*. Para conseguir *n* nodos debemos tomar como paso:

$$h = \frac{b-a}{n-1}$$

```
(%i4)    x:makelist(i,i,-1,1,2/(11-1));
         y:makelist(1/(1+12*i^2),i,-1,1,2/(11-1));
```

$$(x)\quad [-1, -\frac{4}{5}, -\frac{3}{5}, -\frac{2}{5}, -\frac{1}{5}, 0, \frac{1}{5}, \frac{2}{5}, \frac{3}{5}, \frac{4}{5}, 1]$$

$$(y)\quad [\frac{1}{13}, \frac{25}{217}, \frac{25}{133}, \frac{25}{73}, \frac{25}{37}, 1, \frac{25}{37}, \frac{25}{73}, \frac{25}{133}, \frac{25}{217}, \frac{1}{13}]$$

Para construir las abscisas hemos utilizado la propia definición de la función e introducimos el valor de *x* como *i*. Otra opción sería:

```
(%i5)    y:makelist(f(x[i]),i,1,11);
```

$$(y)\quad \left[\frac{1}{13},\frac{25}{217},\frac{25}{133},\frac{25}{73},\frac{25}{37},1,\frac{25}{37},\frac{25}{73},\frac{25}{133},\frac{25}{217},\frac{1}{13}\right]$$

Y como ya comentamos en la primera práctica otra opción sería usar el comando **map**.

```
(%i6)    map(f,x);
```

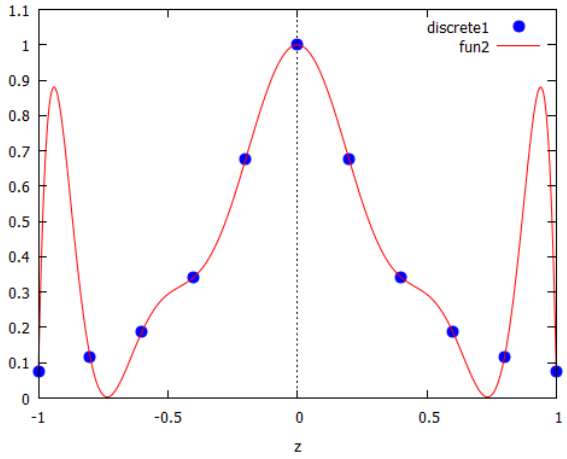
$$(\%o6)\quad \left[\frac{1}{13},\frac{25}{217},\frac{25}{133},\frac{25}{73},\frac{25}{37},1,\frac{25}{37},\frac{25}{73},\frac{25}{133},\frac{25}{217},\frac{1}{13}\right]$$

Volvemos a cargar el bloque **lag(x,y)** y lo ejecutamos, viendo el efecto antes comentado.

```
(%i6)    lag(x,y)$
```

$$El\ polinomio\ int.\ de\ Lagrange\ es:\ p(z)=\ -\frac{97200000000\,z^{10}}{1013396293}+\ \ldots$$

```
(%i7)    plot2d([[discrete,x,y], p(z)], [z,-1,1],
             [style,points,lines])$
```



Vemos el efecto de oscilación que se produce en los nodos de los extremos.

Vamos a intentar calcular una cota del error cometido con el polinomio interpolador de grado 10 (dado que tenemos 11 nodos), usando la fórmula:

$$E_{10}(x)=\frac{f^{(11)}(c)}{(11)!}\prod_{i=0}^{10}(x-x_i)$$

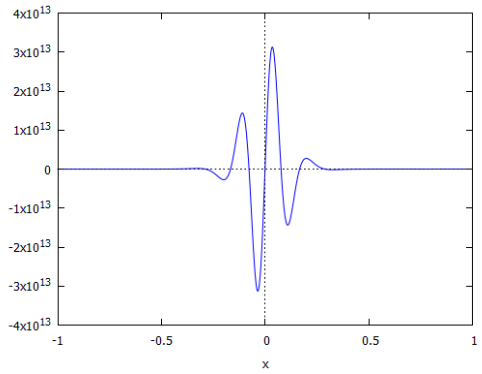
Para estimar el máximo de la derivada $f^{(11)}(x)$ lo haremos a través de su gráfica. Entendemos que es suficiente con este estudio aproximado.

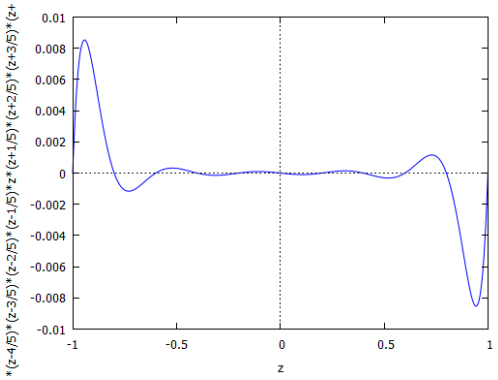
```
(%i1)    kill(all)$
```

```
(%i1)    f(z):=1/(1+12*z^2)$

(%i2)    define(der11(z),diff(f(z),z,11))$

(%i3)    plot2d(der11(z),[z,-1,1])$
```





Obtenemos valores máximos del orden de 3.1210^{13}.

En cuanto al término $\prod_{i=0}^{10}(x - x_i)$, si también lo representamos obtenemos valores máximos del orden de 0.0085:

```
(%i5)    x:makelist(i,i,-1,1,2/(11-1))$
         define(pro(z),product(z-x[i],i,1,11))$

(%i6)    plot2d(pro(z),[z,-1,1])$
```

Por tanto, podemos acotar el error por:

```
(%i7)    (3.12*10^13)*(0.0085)/11!;
(%o7)    6643.819143819144
```

Este valor sólo es una cota superior del error: no es obligatorio alcanzarlo. De hecho vimos en la figura que la máxima diferencia en [-1,1] era del orden de 0.8. Pero nos puede dar pistas de que algo no va bien en nuestra aproximación.

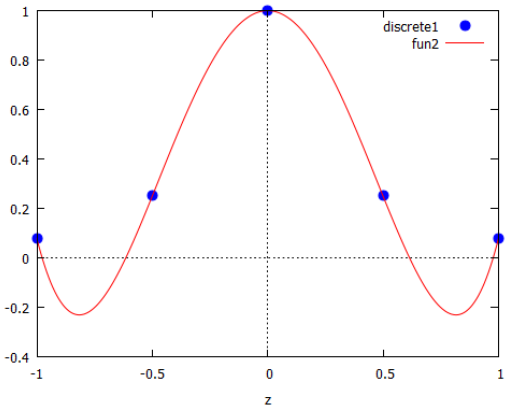
Este efecto se produce al incrementar demasiado el número de puntos de interpolación y por tanto el orden del polinomio. De hecho, si resolvemos el mismo problema pero con menos nodos, por ejemplo con sólo 5 puntos en vez de 11, el resultado es mucho mejor, como vemos a continuación.

```
(%i1)    kill(all)$
(%i2)    x:makelist(i,i,-1,1,2/(5-1));
         y:makelist(1/(1+12*i^2),i,-1,1,2/(5-1));
(x)      [-1,-1/2,0,1/2,1]
(y)      [1/13,1/4,1,1/4,1/13]
```

```
(%i4)    lag(x,y)$
```

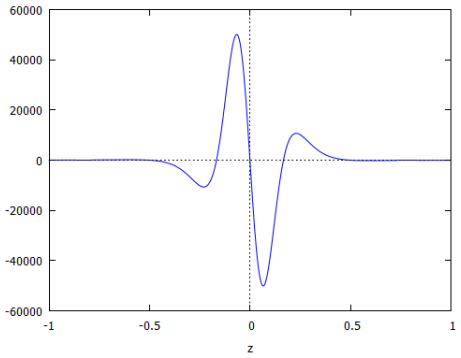
El polinomio int. de Lagrange es: p(z)= $\frac{36\,z^4}{13}-\frac{48\,z^2}{13}+1$

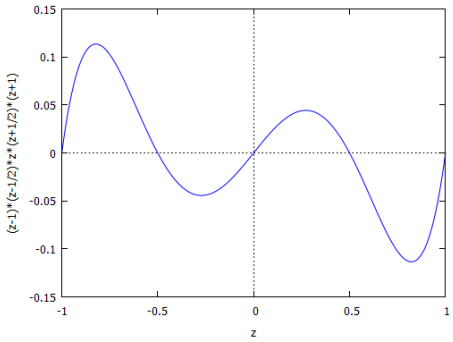
```
(%i5)    plot2d([[discrete,x,y], p(z)], [z,-1,1],
             [style,points,lines])$
```



Observamos ahora que el efecto de oscilación que se produce en los nodos de los extremos es mucho menos acusado que antes con 11 nodos. Vamos a obtener de nuevo, y de forma aproximada una cota del error cometido en la interpolación.

```
(%i6)    f(z):=1/(1+12*z^2)$
(%i7)    define(der5(z),diff(f(z),z,5))$
(%i8)    plot2d(der5(z),[z,-1,1])$
```





Obtenemos valores máximos para la derivada quinta del orden de 5.10^4.

También representamos el término $\prod_{i=0}^{4}(x-x_i)$.

```
(%i9)    define(pro(z),product(z-x[i],i,1,5))$
(%i10)   plot2d(pro(z),[z,-1,1])$
```

Siendo ahora la cota del error mucho menor que antes.

```
(%i11)   (5*10^4)*(0.11)/5!;
(%o11)   45.83333333333334
```

7.6 Nodos de Chebyshev

Aunque parece razonable construir el polinomio de interpolación a partir de nodos equiespaciados, no siempre es la mejor solución para obtener buenas aproximaciones. Como vimos antes con la función de Runge:

$$f(x) = \frac{1}{1 + 12x^2}$$

al aproximarla en [−1, 1] se observa que, en los extremos del intervalo, la aproximación empeora, al aumentar el número de nodos. Una forma de corregir esto es utilizar los nodos de Chebyshev, que son los que minimizan en el intervalo [−1, 1] la función:

$$\prod_{i=0}^{n} (x - x_i)$$

y por tanto disminuyen el error cometido.

Se demuestra que en el caso de querer utilizar *n* puntos en [−1, 1], en lugar de tomarlos equiespaciados, se deben tomar los nodos de Chebyshev, que se pueden hallar como:

$$x_i = cos\frac{(2i-1)\pi}{2n};\ \ i = 1, \dots, n$$

Si lo aplicamos al problema anterior, los 11 nodos serían ahora:

```
(%i1)    kill(all)$
(%i1)    f(z):=1/(1+12*z^2)$
(%i3)    x:makelist(cos((2*i-1)*%pi/(2*11)),i,1,11),numer$
         y:makelist(f(x[i]),i,1,11),numer$
```

Los valores de las abscisas *x* que se obtienen ahora, sin más que aplicar la fórmula dada, son:

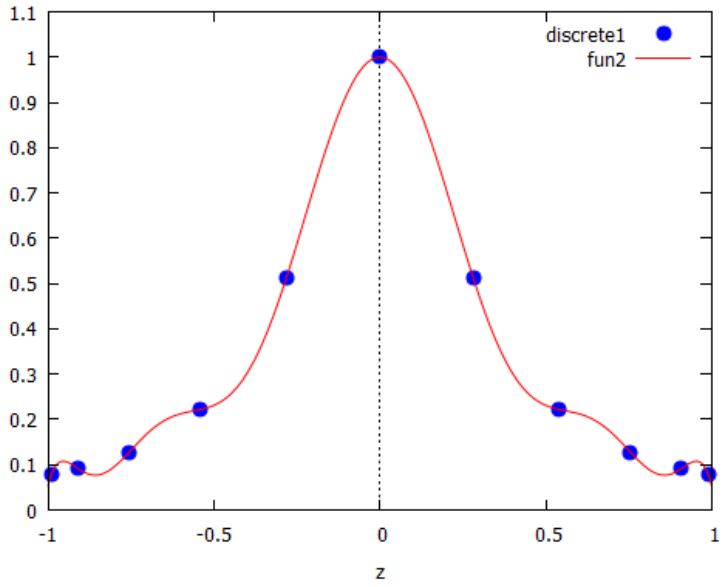
```
(%i3)    transpose(x);
```

$$\begin{bmatrix} 0.9898214418809327 \\ 0.9096319953545184 \\ 0.7557495743542583 \\ 0.5406408174555977 \\ 0.2817325568414298 \\ -1.607689385800854\ 10^{-16} \\ -0.2817325568414297 \\ -0.5406408174555977 \\ -0.7557495743542582 \\ -0.9096319953545184 \\ -0.9898214418809327 \end{bmatrix} \quad (\%o3)$$

Volvemos a cargar el bloque **lag(x,y)** y lo ejecutamos.

```
(%i6)    lag(x,y)$
El polinomio int. de Lagrange es: p(z)=   -25.82048355297292 z^10 -
...
```

```
(%i7)    plot2d([[discrete,x,y], p(z)], [z,-1,1],
              [style,points,lines])$
```



Obtenemos, como vemos, un resultado mucho mejor, evitando el fenómeno de oscilación polinomial.

Las fórmulas que acabamos de ver sólo son válidas para el caso particular del intervalo $[-1, 1]$.

Para poder obtener los nodos de Chebyshev, en el caso general de un intervalo $[a, b]$ basta con aplicar el cambio de variable:

$$x = \frac{b+a}{2} + \frac{b-a}{2}\hat{x}$$

En esta fórmula $\hat{x}$ es el valor obtenido con la fórmula inicial.

Se demuestra que con esos nodos el valor máximo en el intervalo $[a, b]$ de la función:

$$\prod_{i=0}^{n}(x - x_i)$$

es:

$$max_{x\in[a,b]} \prod_{i=0}^{n}(x - x_i) = \left(\frac{b-a}{2}\right)^{n+1} \cdot \frac{1}{2^{n+1}}$$

7.7 Ejercicios Propuestos

Ejercicio 1. (a) Obtener mediante la definición o cálculo directo el polinomio interpolador que pasa por los puntos:

(3.2, 22); (2.7, 17.8); (1, 14.2); (4.8, 38.3); (5.6, 51.7)

Comprobar que pasa por dichos puntos y construir la gráfica de los puntos y el polinomio.
(b) Usar ahora diferencias divididas.
(c) Usar la forma de Lagrange.

Ejercicio 2. Dada la tabla de valores:

x: [0, 1, 2, 3, 4, 5]
y: [1.1, 1.5, 2.4, 2, 3, 1]

hallar el correspondiente polinomio de interpolación y represéntese su gráfica, junto con los nodos de interpolación. Usar tanto diferencias divididas de Newton como la forma de Lagrange.

Ejercicio 3. Vamos a aproximar la función $f(x) = x\ \text{sen}(x)$, en el intervalo [0, 3], y para ello vamos a usar el siguiente método. Construimos un vector de 5 elementos x_i, igualmente espaciados en dicho intervalo, y calculamos el polinomio interpolador $p(x)$ que pasa por los puntos $(x_i, f(x_i))$.

Acotar el error absoluto máximo cometido al aproximar $f(x)$ por $p(x)$ y representar, para $0 \leq x \leq 3$, tanto el polinomio hallado como la función $f(x)$, observando el fenómeno de extrapolación que se produce si dichas gráficas se hacen en el intervalo [0, 4].

Ejercicio 4. Aproximar, en [−1, 1], la función:

$$f(x) = \frac{1}{1 + 25x^2}$$

utilizando polinomios interpoladores con nodos equiespaciados de grados 8 y 10. Mostrar claramente qué ocurre. A continuación construir las mismas aproximaciones a partir de los nodos de Chebyshev, y represéntense gráficamente los resultados obtenidos.

Ejercicio 5. Aproximar, en [0, 1], la función:

$$f(x) = \frac{1}{1 + 15x^2}$$

utilizando polinomios interpoladores con nodos equiespaciados de grados 5 y 10. Mostrar claramente qué ocurre. A continuación construir las mismas aproximaciones a partir de los nodos de Chebyshev, y represéntense gráficamente los resultados obtenidos.