

1. Introducción y algunos comandos

% para comentar líneas

; NO vemos el valor por pantalla

whos nos dice qué son todas nuestras variables y **whos a** nos dice qué es una variable

>> edit invocamos al editor

clear limpia la memoria del ordenador y **clear a** elimina una variable

si el código funciona después de esto significa que es autocontenido y no usa variables previamente creadas

>> helpdesk nos abre la ayuda en MatLab

a = 1 crea un vector/matriz 1x1 → las variables son matrices double de 8 bytes

z = y' es la **traspuesta**

2. Formatos

>> format short muestra 5 cifras significativas si $x = 1/3$ nos daría $x = 0.3333$

>> format short e lo da como exponencial $x = 3.3333e^{-1}$ que es $3.3333 \cdot 10^{-1}$

>> format long muestra 14 cifras significativas $x = 0.33333333333333$

>> format long e lo da como notación científica

3. Creación de matrices y vectores

Matrices

$$A = [\text{una fila} \quad ; \text{ otra fila}] \quad A = [1 \ 2 \ 3; 5 \ 1 \ 4] \quad A = \begin{pmatrix} 1 & 2 & 3 \\ 5 & 1 & 4 \end{pmatrix}$$

size (A) devuelve el tamaño de la matriz, en este caso, 2 3 (nº filas, nº columnas)

length (A) devuelve el nº más grande, o bien las filas o bien las columnas

Vectores con elementos que distan entre sí

- Si los saltos son de uno en uno, **v = a:b**, $y = 1:5 \rightarrow y = 1 \ 2 \ 3 \ 4 \ 5$
- Si los saltos son de h en h, **v = a:h:b**, $y = 1:2:5 \rightarrow y = 1 \ 3 \ 5$
- **linspace (a,b,n)** (desde a, hasta b, con n elementos en medio)

4. Concatenando matrices

Para poder **concatenar** matrices tienen que tener la **misma dimensión**

>>x = [1 2]; **>> A = [x y]** → $\begin{matrix} 1 & 2 & 4 & 5 \\ 4 & 5 \end{matrix}$
>>y = [4 5]; **>> B = [x ; y]** → $\begin{matrix} 1 & 2 \\ 4 & 5 \end{matrix}$

Para guardar en m y n el nº de filas y columnas **[m,n] = size (x)**

5. Indexado/ Acceder a elementos de la matriz o de un vector

A (indicador fila , indicador columna)

$$A = \begin{pmatrix} 2 & 4 & 6 & 8 \\ 3 & 5 & 6 & 9 \\ 1 & 3 & 3 & 4 \end{pmatrix}$$

>> A (1, 1) = 1 //devuelve el elemento de la fila 1, columna 1

>> A (1, [1 3]) = 1 3 //devuelve los elementos 1 y 3 de la primera fila

>> A ([1 2], 2) = 3 //la primera y la segunda fila de la segunda columna
8

>> A (2, [1 2 3 4 5]) = 6 7 8 9 10 //todos los elementos de la segunda fila

>> A (2, [1:5]) = " "

>> A (2, [1:end]) = " "

>> A (2, :) = " " //nos da toda la fila 2

>> A (:, 4) = 4 //nos da toda la columna 4
9

>> A ([1 2], :) = 1 2 3 4 5 //nos daría la primera y la segunda fila al completo
6 7 8 9 10

Indexado lógico

//aquellos valores de A que cumplan la condición se cambian

>> A (A < 10) = 0

0 0 0 0 0

0 0 0 1 0

NOTA: El 1er índice es 1 y el último end

6. Operaciones en MatLab

Operaciones matriciales

suma +, resta -, multiplicación *, división dcha /, división izq \, potencia ^

Operaciones punto a punto

tienen que tener la misma dimensión y operamos elemento a elemento

producto .*, división ./ y .\, potencia.^

$$A/B = A \cdot (B^{-1}) = A \cdot \frac{1}{B} (= A/B)$$

$$A \backslash B = (A^{-1}) \cdot B = \frac{1}{A} \cdot B \left(\frac{1}{A} \cdot B \neq B \cdot \frac{1}{A} \right)$$

OJO con los escalares

ese caso es escalar * matriz mientras que los anteriores son matriz * matriz

$$A = \begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix}; B = \begin{pmatrix} 4 & 5 \\ 6 & 7 \end{pmatrix}$$

$$A \cdot B = \begin{pmatrix} 6 & 7 \\ 26 & 31 \end{pmatrix} \neq A \cdot \cdot B = \begin{pmatrix} 0 & 5 \\ 12 & 21 \end{pmatrix}$$

elto x elto

Funciones definidas ya en MatLab

Operadores de comparación

>> (A < 6)

1 1 1 1 1
0 0 0 0 0

//nos da una matriz lógica de 0s y 1s donde el 1 significa que cumple la condición y 0 no

7. Gráficos

plot (vector eje x, vector eje y, 'estilo línea') con x, y de la misma dimensión

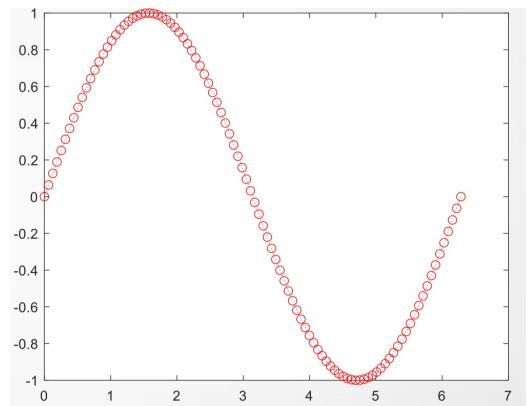
El **estilo de línea** se pone ' : : : ' y es 'tipo de línea : color : marcador'

Acceso a determinados puntos de la gráfica

(x(1), y(1)) primer punto de la gráfica

(x(i), y(i)) punto de la posición i

(x(end), y(end)) último punto de la gráfica



Anotaciones en las gráficas y otros comandos

legend ('nombre') nos sirve para etiquetar el gráfico

title ('título')

xlabel ('eje x')

ylabel ('eje y')

$x = 0:0.1:10;$ marcador
plot(x, cos(x), ':ro')

color

Para superponer varios gráficos usamos **hold on** y **hold off** para desactivarlo

figure crea una nueva ventana y por defecto pinta en la ventana que esté activa

subplot (n m k) divide una ventana gráfica en n partes horizontales, m verticales y k es la subdivisión que se activa

semilog x/y () sirve para estirar lo que está pegado al cero y es una escala logarítmica en el eje de abscisas/ordenadas

truco

primero hago el plot normal y viendo la gráfica veo cuál me viene mejor

8. Uso del fprintf

fprintf(' ',) cadena de caracteres y una lista de variables

pueden ser texto, descriptores (%f) o comandos de impresión (\n)

$x = 65;$

fprintf('Int %d Hex %x Real %f Char %c Not Cient %e\n', x, x, x, x, x)

Int 65 Hex 41 Real 65.000000 Char A Not Cient 6.500000e+001

#El nº de variables que pongamos debe ser igual que el nº de descriptores usados

%.**5f** significa que muestres la variable con 5 decimales y sin dejar espacios

%**12.5f** muéstralala con 5 decimales dejando 12 espacios

el 12 son las columnas dedicadas al formateo y 5 los números detrás de la coma digital

%**+12.5f** le obliga a ponerle signo

%**4d** reservo cuatro espacios y puede ser útil para alinear textos y hacer tablas

#Las matrices se recorren **columna a columna**

9. Funciones en MatLab

//crear un script, nombre función = nombre del fichero script

>> edit __

function [c] output = nombre_funcion(a,b) input

c = a*b;

return

#Si queremos usar nuestra función con un vector debemos usar operadores punto a punto

10. Condiciones y bucles

IF una alternativa

```
if (condición),  
    código si es verdadera;  
else  
    código si es falsa;  
end
```

Bucle **FOR**

```
for k= [ ],  
//índice = lista de números, vector  
//para salir del bucle  
if (), continue; end //saltamos un paso  
if (), break; end //paramos ahí  
código que se repite hasta acabar con k;  
end
```

SWITCH varias alternativas

```
switch (condición)  
    case 1, código;  
otherwise esto;  
end
```

Bucle **WHILE**

```
//contador de iteraciones  
iter = _;  
//inicializamos las variables antes del bucle  
while (condición) && (iter > _)  
    código;  
    iter = iter +1;  
end
```

Representación de números en ordenador. Errores

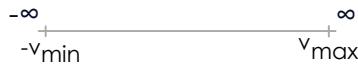
Por def las variables tienen **8 bytes**, son de tipo **double** y la **precisión** es de 10^{-15} (15 dec)

Al añadir **single(x)** pasa a ocupar **4 bytes**, a ser de tipo **single** y la **precisión** es de 10^{-7} (7 dec)

$\hookrightarrow 0.0000001$

1. Números reales y números máquina

Un número se representa en el ordenador mediante un nº finito de dígitos, números maquina



los ordenadores solo trabajan con un conjunto finito de números y esos son los números máquina

Representación decimal $\rightarrow 2.75 = 2 \cdot 10^0 + 7 \cdot 10^{-1} + 5 \cdot 10^{-2} = 2.75 \cdot 10^0 = 0.275 \cdot 10^{-1}$

Representacion binaria $\rightarrow 2.75 = (10.11)_2 = 1 \cdot 2^1 + 0 \cdot 2^0 + 1 \cdot 2^{-1} + 1 \cdot 2^{-2} = (1.011)_2 \cdot 2^1$

2. Error absoluto y error relativo

Error absoluto $\text{abs}(vex-vap)$

$\text{error_abs} = |x - \hat{x}| = |\text{valor exacto} - \text{valor aproximado}| \approx 10^{-(\text{nº de decimales correctos})}$

ejemplo $x = 5922753.123, \hat{x} = 5922753.122 \rightarrow \frac{-0.003}{0.002} \rightarrow 1 \cdot 10^{-3}$

$E_{\text{abs}} = |x - \hat{x}| = 10^{-3} \rightarrow \text{El error está en 3er decimal } (x \approx \hat{x} \pm 10^{-3}),$

$= 0.001 \rightarrow \text{Si } E_{\text{abs}} \approx 10^{-3} \text{ la aproximación garantiza entre 2 y 3 decimales correctos}$

Error relativo $\text{abs}((vex-vap)/vex)$

$\text{error_rel} = \frac{|x - \hat{x}|}{|x|} \approx 10^{-(\text{nº de cifras significativas})}$

$$\log_2 8 \rightarrow 2^x = 8$$

ejemplo

$$E_{\text{rel}} = \frac{|x - \hat{x}|}{|x|} = \frac{0.001}{5922753.273} = \frac{1.688403946 \times 10^{-10}}{1.6884 \cdot 10^{-9}} < 10^{-9} \rightarrow \text{Garantiza 9 cifras decimales significativas coinciden en 9 cifras significativas}$$

Nº de cifras decimales significativas $\approx n\text{Cif} = \text{floor}(-\log_{10}(\text{error_rel}))$

$$\begin{aligned} \text{¿} 10^{-x} = \text{error_rel} \text{?} \\ \downarrow \\ x = -\log_{10} \text{error_rel} \end{aligned}$$

3. Representación científica o en coma flotante

Un número máquina o en representación científica x se escribe como $x = (-1)^s m \cdot b^e$

b = base de representación (2 en binario, 10 en decimal)

s = signo

m = mantisa en base b

nº cifras significativas

e = exponente en base b

rango de valores de los nº representados

Los números NO son equidistantes

s	mantisa	exponente
-----	---------	-----------

$s < 0 \rightarrow +$ $1,25 \times 10^0$ exponente
 $1 \rightarrow -$ mantisa base

v_{aprox} sumando los 100 primeros términos

$$\text{valor aproximado} \quad \sum_{n=0}^{\infty} \frac{(-1)^n}{2n+1} = \frac{1}{1} - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \dots = \frac{\pi}{4}$$

```
>> n=[0:99];
>> % Secuencia numerador: (-1).^n
>> % Secuencia denominador: 2.^n+1
>> % Secuencia a sumar: ((-1).^n) ./ (2.^n+1)
>> v_aprox=sum((-1).^n ./ (2.^n+1)) % valor aproximado sumando 100 términos
```

Calcular el error relativo y el nº de cifras decimales significativas que proporciona dicha aproximación:

Valor exacto
 Valor aproximado
 $\rightarrow 0.0025$

```
>> error_abs=abs(pi/4-v_aprox)
>> error_rel=abs((pi/4)-v_aprox)/(pi/4)
0.0032      error absoluto      valor exacto
              (~3*10^-3>10^-2, entre 2 y 3 cifras decimales significativas, 2 garantizadas)

>> n_cif_dec_sig=-log10(error_rel)
2.4972      truncamiento

>> n_cif_dec_sig=floor(-log10(error_rel)) → 2 cifras decimales significativas garantizadas
```

Representación normalizada $(-1)^s (\sum_{j=0}^p d_j b^{-j}) b^e$

El primer dígito de la mantisa es $\neq 0$

ejemplo

$$0.15000 = 1.5 \times 10^{-1}$$

representación normalizada

En la representación BINARIA no almacenamos el primer bit de la mantisa



Distancia entre 1 y el siguiente nº máquina eps (1)

eps (1) = b^{-p} siendo p el nº de cifras de la mantisa

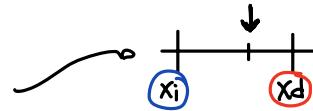
$$\begin{aligned} 1 &= (1. \quad 0 \quad 0 \quad \dots \quad 0)_b \quad b^0 \\ 1+ &= (1. \quad 0 \quad 0 \quad \dots \quad 1)_b \quad b^0 \\ &(b^0 \quad b^{-1} \quad b^{-2} \quad \dots \quad b^{-p}) \end{aligned}$$

$$\xrightarrow{\hspace{2cm}} 1.110100110 \xrightarrow{\hspace{2cm}} 0.000000001 \xrightarrow{\hspace{2cm}} 2^{-9} = \text{eps}$$

Distancia entre nº máquinas consecutivos

$$|x_i - x_d| = b^{-p} * b^{\text{exp}} = \text{eps (1)} * b^{\text{exp}}$$

$$\begin{aligned} x_i &= (1. \quad d_1 \quad d_2 \quad \dots \quad d_p)_b \quad b^{\text{exp}} \\ x_d &= (1. \quad d_1 \quad d_2 \quad \dots \quad d_p + 1)_b \quad b^{\text{exp}} \\ &(b^0 \quad b^{-1} \quad b^{-2} \quad \dots \quad b^{-p}) \end{aligned}$$



Errores por redondeo $* X = x_d$

x se aproxima al nº máquina + próximo

$$\text{Error absoluto} = |x - x| \leq \frac{1}{2} b^{-p} b^e$$

$$\text{Error relativo} = \frac{|x-x|}{|x|} \leq \frac{1}{2} b^{-p}$$

Errores por truncamiento $* X = x_i$

x se aproxima al nº máquina inmediatamente <

$$\text{Error absoluto} = |x - x| \leq b^{-p} b^e$$

$$\text{Error relativo} = \frac{|x-x|}{|x|} \leq b^{-p}$$

4. Estandar IEEE 754

	SIMPLE PRECISIÓN	DOBLE PRECISIÓN
Nº DE BITS	Palabras de 32 bits (4 bytes) 1bit signo + 23 mantisa + 8 exp	Palabras de 64 bits (8 bytes) 1bit signo + 52 mantisa + 11 exp
PRECISIÓN	$\text{eps} = 2^{-23} = 1.19 \times 10^{-7}$ $\text{erel} \leq \frac{1}{2} 2^{-23} \approx 10^{-7}$ 7 cifras decimales significativas	$\text{eps} = 2^{-52} = 2.2 \times 10^{-16}$ $\text{erel} \leq \frac{1}{2} 2^{-52} < 10^{-15}$ 15 cifras decimales significativas
RANGO DE VALORES	Exceso a $2^{n-1}-1$: Simple precision: $2^n - 1 = 127 \rightarrow 00000000000000000000000000000000 = 127 \rightarrow 0-127$ $11111111111111111111111111111111 = 255 \rightarrow 128-255$ donde precision: $2^{n-1} = 1023 \rightarrow 00000000000000000000000000000000 = 1023 \rightarrow 0-1023$ $11111111111111111111111111111111 = 2047 \rightarrow 1024-2047$ $[1-1023, 2047]$ Rango de valores exponente [-126, 127]	Rango de valores exponente [-1022, 1023]
	Rango nº normalizados [-10 ⁻³⁸ , 10 ³⁸]	Rango nº normalizados [-10 ⁻³⁰⁷ , 10 ³⁰⁸]
	Rango nº desnormalizados [-10 ⁻⁴⁵ , 10 ⁻³⁸]	-

5. Operaciones aritméticas con números en coma flotante. Propagación de errores

Suma / Resta

- 1º Igualar exponentes al mayor.
- 2º Sumar mantisas, normalizar y redondear.

NOTA

- Problemas al sumar números de orden de magnitud muy dispar, se podrían perder los dígitos menos significativos del número menor
- Problemas al restar números muy próximos, posible cancelación de dígitos

ejemplo 1 (trabajando con 4 cifras decimales):

$$(1.234 \times 10^3) + (5.678 \times 10^1) = 1.23400 \times 10^3 + 0.05678 \times 10^3 = 1.29078 \times 10^3$$

↑
Igualar exponentes al mayor
↳ 3-1 = 2 ; Desplazamos 2

~1.291 × 10^3 → Erel: 1.7 × 10^-4 < 10^-3
(3 cifras signif. garantizadas)

ejemplo 2 (suma números muy dispares):

$$1.234 \times 10^3 + 1.200 \times 10^{-1} = 1.234 \times 10^3 + 0.0001200 \times 10^3 = 1.2341200 \times 10^3$$

↑
Igualar exponentes al mayor
↳ 3 - (-1) = 4, Desp 4

~1.234 × 10^3
(0 cifras signif. Erel ~ 10^0)

ejemplo 3 (resta de números próximos):

$$1.234 \times 10^3 - 1.231 \times 10^3 = 0.003 \times 10^3 \sim 3.??? \times 10^0$$

↑
Igualar exponentes al mayor

Producto / División

- 1º Sumar / Restar exponentes ¿está en rango?
- 2º Multiplicar / Dividir mantisas, normalizar, redondear

NOTA

- El exponente puede salirse del rango (OVF)
- Problemas al dividir por números cuyo orden de magnitud es muy pequeño

ejemplo 1 (trabajando con 4 cifras dec. significativas):

$$1.234 \cdot 10^3 \times 9.876 \cdot 10^1 = 12.186984 \cdot 10^4 = 1.219 \cdot 10^5$$

Interpolación de funciones/datos

1. Problema general de interpolación

Interpolar es obtener una función de un tipo señalado que coincide con $f(x)$ en el conjunto de datos numéricos dado. En las gráficas, la función debe pasar por los puntos dado por los datos

INPUT: Conjunto discreto de datos numéricos de una función $f(x)$

OUTPUT: Función de determinada clase que coincide con $f(x)$ en los datos numéricos dados

Elementos del problema

- **Datos numéricos a interpolar**

Tabla de datos (valores de la función en un conjunto de puntos)

$$y_0 = f(x_0), y_1 = f(x_1), \dots, y_n = f(x_n).$$

xi	0	1/4	1/2	3/4
yi	1	-1	2	0

Valores de la función y su derivada en dos puntos (interpolación de Hermite)

$$\left. \begin{array}{l} f(x_0), f(x_1) \\ f'(x_0), f'(x_1) \end{array} \right\} \text{conocidos.} \quad \begin{array}{l} \text{lo que vale } f \text{ en } X_0 \text{ y } X_1 \\ \text{lo que vale } f' \text{ en } X_0 \text{ y } X_1 \end{array}$$

Valor de la función en los puntos 0 y 1 y su valor medio en [0,1]

$$f(0) = y_0, f(1) = y_1, \int_0^1 f(x) dx = m, \text{ con } y_0, y_1 \text{ y } m \text{ conocidos.}$$

Valores de la función y sus derivadas en un punto a (interpolación de Taylor)

$f(a), f'(a), f''(a)$ conocidos. lo que vale f en a , f' en a y f'' en a

- **Tipo de función interpolante**

Polinomios de grado n (n° elem -1) $\varphi(x) = c_0 + c_1 x + c_2 x^2 + \dots + c_n x^n$

Funciones trigonométricas $\varphi(x) = a_0 + \sum_{k=1}^n a_k \cos(kx) + \sum_{k=1}^n b_k \sin(kx)$.

Funciones polinómicas a trozos ...

Error de interpolación

$$e(x) = |f(x) - \varphi(x)| = |\text{función interpolada} - \text{función interpolante}|$$

Soluciones del problema

El problema tiene **solución** y es **única** si y solo si el sistema lineal resultante es un SCD, el determinante de la matriz de coeficientes del sistema es NO nulo ($|H| \neq 0$)

$$\det(H) = \begin{vmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & 2 & 4 \end{vmatrix} \neq 0.$$

matriz de coeficientes

lo q acompaña a las variables/
incógnitas ej $3X = 6$ coeficiente = 3

$$Hc = b$$

matriz términos
independientes

sería 6 en el ej anterior

incógnitas

sería x en el ej anterior

donde se guardan las
soluciones del sistema

$$c = H \setminus b;$$

Gráfica de la función interpolante

1º definimos un conjunto de puntos auxiliares (bastante resolución) en el intervalo dado, [x,y]

$$x_{aux} = [x:0.01:y]$$

2º creamos el vector de valores obtenidos en dichos puntos auxiliares

p_{aux} = al polinomio y donde haya una x ponemos x_{aux}

3º pintamos la gráfica del polinomio en verde y los puntos en los que se interpola en rojo

`plot (x_{aux}, p_{aux}, 'g', xi, yi, 'r')`

2. Ejemplos ilustrativos

Elementos a identificar < Función interpolante
Datos numéricos a coincidir

EJEMPLO 1

Interpolación por un polinomio de grado 3

$$\varphi(x) = c_1 + c_2 x + c_3 x^2 + c_4 x^3$$

los datos de la tabla.

xi	0	1/4	1/2	3/4
yi	1	-1	2	0

Tipo función interpolante
Base: $\{1, x, x^2, x^3\}$
Dimensión 4

Datos numéricos
(4 condiciones interpol)

Plantamiento problema interpolación:

$$\varphi(x) = c_1 + c_2 x + c_3 x^2 + c_4 x^3$$

Imponemos verifique condiciones de interpolación:

$$\begin{aligned}\varphi(0) &= c_1 + c_2 \cdot 0 + c_3 \cdot 0^2 + c_4 \cdot 0^3 = c_1 = 1 \\ \varphi(0.25) &= c_1 + c_2 \cdot 0.25 + c_3 \cdot 0.25^2 + c_4 \cdot 0.25^3 = -1 \\ \varphi(0.5) &= c_1 + c_2 \cdot 0.5 + c_3 \cdot 0.5^2 + c_4 \cdot 0.5^3 = 2 \\ \varphi(0.75) &= c_1 + c_2 \cdot 0.75 + c_3 \cdot 0.75^2 + c_4 \cdot 0.75^3 = 0\end{aligned}$$

Se resuelve el sistema lineal resultante y se calculan los valores de c_1, c_2, c_3 y c_4

$$(1 \quad -31.33 \quad 120 \quad -106.66)$$

Función interpolante:

$$\varphi(x) = 1 - 31.33x + 120x^2 - 106.66x^3$$

sust valores de c en $f(x)$

en MATLAB

% ejemplo 1

% datos numéricos, metemos la tabla de datos como vectores columna ('')

$$\begin{array}{l} x_i = [0 \ 1/4 \ 1/2 \ 3/4]'; \\ y_i = [1 \ -1 \ 2 \ 0]'; \\ \% sistema lineal (Hc = b) \\ \% H matriz de coef \\ H = [x_i.^0 \ x_i.^1 \ x_i.^2 \ x_i.^3]; \\ \% [1 \ x \ x^2 \ x^3] \\ \% b matriz de términos independientes \\ b = y_i; \\ \% ahora resolvemos el sistema y guardamos las soluciones en el vector c \\ c = H \ b; \\ \% pintamos el polinomio, lo visualizamos en el intervalo [0,1] \\ x_{aux} = 0:0.01:1; \\ \% = xi NO porque son pocos puntos \\ \% evaluamos el polinomio para cada valor de x_{aux} \\ p_{aux} = c(1) + c(2)*x_{aux} + c(3)*x_{aux}.^2 + c(4)*x_{aux}.^3; \\ \% lo dibujamos \\ plot(x_{aux}, p_{aux}, 'g', xi, yi, 'r'); \\ \% la verde es el polinomio evaluado en ese intervalo \\ \% la roja con puntos son los valores de la tabla y tienen que coincidir \\ \text{escribimos el polinomio y donde haya x ponemos x_{aux}} \end{array}$$

$$(H^{-1})Hc = (H^{-1})b \rightarrow Ic = H^{-1}b ; c = H^{-1}b$$

EJEMPLO 7 INTERPOLACIÓN DE TAYLOR

Los datos ya no son valores de la función en determinados puntos si no el valor de la función en un punto, el de su primera derivada en ese mismo punto y el de su segunda derivada en ese punto

Elementos del problema:

1. Datos numéricos:

$$f(a) = f_a, \quad f'(a) = f'_a, \quad f''(a) = f''_a \quad (a, f_a, f'_a, f''_a \text{ conocidos})$$

2. Tipo función interpolante: $\varphi(x) = c_1 + c_2 e^x + c_3 e^{-x}$

Generada a partir de las funciones {1, exp(x), exp(-x)} (base)

Plantamiento problema interpolación:

$$\varphi(x) = c_1 + c_2 e^x + c_3 e^{-x}$$

Imponemos verifique condiciones de interpolación:

$$\begin{aligned} \varphi(a) &= c_1 + c_2 e^a + c_3 e^{-a} = f_a \\ \varphi'(a) &= c_2 e^a - c_3 e^{-a} = f'_a \\ \varphi''(a) &= c_2 e^a + c_3 e^{-a} = f''_a \end{aligned} \longrightarrow \underbrace{\begin{pmatrix} 1 & e^a & e^{-a} \\ 0 & e^a & -e^{-a} \\ 0 & e^a & e^{-a} \end{pmatrix}}_H \underbrace{\begin{pmatrix} c_1 \\ c_2 \\ c_3 \end{pmatrix}}_c = \underbrace{\begin{pmatrix} f_a \\ f'_a \\ f''_a \end{pmatrix}}_b$$

derivamos $\varphi(x)$ e imponemos la condición
derivamos $\varphi'(x)$ e imponemos la condición

Resolvemos el sistema y calculamos la solución:

$$c_1 = f_a - f'_a, \quad c_2 = \frac{1}{2}(f_a + f'_a)e^{-a}, \quad c_3 = \frac{1}{2}(f''_a - f'_a)e^a$$

EJEMPLO 8 INTERPOLACIÓN DE TAYLOR - sistema NO lineal

El sistema no es lineal porque los parámetros (c_1, c_2 y c_3) se relacionan entre sí, el objetivo es linealizarlo para que ningún parámetro dependa de los otros

Elementos del problema:

1. Datos numéricos:

$$f(a) = f_a, \quad f'(a) = f'_a, \quad f''(a) = f''_a \quad (a, f_a, f'_a, f''_a \text{ conocidos})$$

2. Tipo función interpolante: $\varphi(x) = \frac{c_1 + c_2(x-a)}{1 + c_3(x-a)}$

Plantamiento problema interpolación:

Imponemos las condiciones

$$\begin{aligned} \varphi(a) &= c_1 &= f(a) & (1) \\ \varphi'(a) &= c_2 - c_1 c_3 &= f'(a) & (2) \\ \varphi''(a) &= -2c_3(c_2 - c_1 c_3) &= f''(a) & (3) \end{aligned}$$

sistema NO lineal

parámetros relacionados entre sí

Linealizamos el sistema

1. Caso $f'(a) \neq 0$. Resolviendo el sistema se obtiene:

$$c_1 = f(a), \quad c_3 = \frac{-f''(a)}{2f'(a)}, \quad c_2 = f'(a) - \frac{f(a)f''(a)}{2f'(a)}$$

Por tanto: existe solución y es única (dada por los coeficientes anteriores)

2. Caso $f'(a) = 0$. Distinguimos dos subcasos:

2.1. Caso $f''(a) = 0$, se obtiene $c_1 = f(a), \quad c_2 = c_1 c_3 = f(a)c_3$

Por tanto, en este caso existen infinitas soluciones.

2.2. Caso $f''(a) \neq 0$, no existe solución al ser incompatibles las ecuaciones (2) y (3)

3. Interpolación polinomial clásica

3.1 Interpolación polinomial clásica

DATOS NUMÉRICOS n+1 datos

Se conocen los valores de una función $f(x)$ en un conjunto de puntos $x_0, x_1 \dots x_n$ distintos

- Valores de la función en un conjunto de puntos dados

$$y_0 = f(x_0), y_1 = f(x_1), \dots, y_n = f(x_n).$$

- Tabla de datos

xi	x0	...	xn
yi	y0	...	yn

- Valores del polinomio en los puntos

$$y_0 = p(x_0), y_1 = p(x_1), \dots, y_n = p(x_n).$$

- Puntos por los que tiene que pasar la gráfica de la función pedida

$$(x_0, y_0), (x_1, y_1), \dots, (x_n, y_n)$$

TIPO DE FUNCIÓN INTERPOLANTE $p(x)$ polinomio de grado n (nº datos -1)

Hay que encontrar el polinomio de grado n o menor que coincida con $f(x)$ en los datos dados y cuya gráfica pase por esos puntos

RESOLUCIÓN

1º Construimos el polinomio de grado n nº datos -1

2º Imponemos que $p(x)$ verifique los datos de interpolación (n+1)

3º Resolvemos el sistema lineal de ecuaciones que tendrá solución y será única si $\det(H) \neq 0$

4º Sustituimos en $p(x)$ los valores de c obtenidos al resolver el sistema y obtenemos el polinomio interpolador

Ejemplo 1. Construir el polinomio $p(x)$ de menor grado que interpola los datos $f(0)=0, f(1)=2, f(2)=0$. 3 datos

↳ polinomio de grado 3-1 = 2

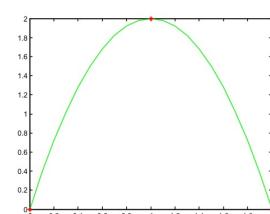
• **Tipo función interpolante:** En principio 3 parámetros para interpolar 3 datos:
Polinomio grado 2:

$$p_2(x) = c_0 + c_1 x + c_2 x^2$$

• **Planteamiento y resolución del problema:**

Calcular c_0, c_1 y c_2 tales que $p(x) = c_0 + c_1 x + c_2 x^2$ verifique las condiciones de interpolación:

$$\left. \begin{array}{l} p(0) = c_0 + c_1 0 + c_2 0 = f(0) = 0 \\ p(1) = c_0 + c_1 + c_2 = f(1) = 2 \\ p(2) = c_0 + c_1 2 + c_2 4 = f(2) = 0 \end{array} \right\}$$



con MATLAB

% 1. DATOS IN:

% Vector columna con los nodos sobre los que tenemos los datos numéricos

xi=[0:2]';

% Vector columna con los valores de la función sobre los nodos anteriores:

yi=[0 2 0]';

% 2. MATRIZ COEFICIENTES (H) Y TÉRMINO INDEPENDIENTE (b) DEL SISTEMA:

H=[xi.^0 xi.^1 xi.^2];

b=yi;

% 3. RESOLVEMOS EL SISTEMA, obteniendo los coeficientes del polinomio en el vector c:

c=H\b;

c0=c(1);

c1=c(2);

c2=c(3);

Coeficientes de $p(x)$ en base $\{1, x, x^2, x^3\}$

% 4. GRÁFICA DE LA FUNCIÓN INTERPOLANTE: Queremos representar gráficamente el polinomio $p(x)$ y los puntos donde interpola:

% Primero definimos un conjunto de puntos auxiliares en el intervalo [0,2] sobre los que haremos la gráfica:

xx=0:0.1:2;

% Creamos el vector de valores del polinomio obtenido en dichos puntos:

yy=c0*xx.^0+c1*xx.^1+c2*xx.^2; (o $\text{polyval}(c(\text{end}:1:1), xx)$);

% Pintamos la gráfica del polinomio interpolante en verde, señalando los puntos en los que se interpola con * rojos:

plot(xx, yy, 'g', xi, yi, 'r');

% 5. ESTIMAMOS $f(1.1)$ vía su polinomio de interpolación $f(1.1) \sim p(1.1)$. Evaluamos $p(1.1)$:

Veremos técnicas más eficientes

a=1.1; faprox=c0*c1*a + c2*a^2; (o $\text{polyval}(c(\text{end}:1:1), a)$);

3.2 Fórmula de Newton valores de la función en unos puntos $f(x) = y$

- Diferencias divididas

Diferencias divididas de grado 0 de f en x_0 :

$$f[x_0] = f(x_0).$$

Diferencias divididas de grado 1 de f en x_0 y x_1 :

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

Diferencias divididas de grado n de f en x_0, x_1, \dots, x_n :

$$f[x_0, x_1, \dots, x_n] = \frac{f[x_1, \dots, x_n] - f[x_0, \dots, x_{n-1}]}{x_n - x_0}, \quad (x_0 \neq x_n)$$

- Fórmula de Newton

$$p_n(x) = \underbrace{f[x_0]}_{p_0(x)} + \underbrace{\underbrace{f[x_0, x_1](x-x_0)}_{p_1(x)} + f[x_0, x_1, x_2](x-x_0)(x-x_1) + \dots + f[x_0, x_1, x_2, \dots, x_n](x-x_0)(x-x_1)\dots(x-x_{n-1})}_{p_2(x)}$$

Una vez obtenidas las diferencias divididas (la diagonal de la tabla) sustituimos en la fórmula y obtenemos el polinomio interpolador

Ejemplo 3: Construir el polinomio $p(x)$ de grado 3 que interpole los datos:

$$f(0)=0, f(1)=2, f(2)=0, f(3)=1$$

1º Tabla de diferencias divididas $f(x) = y$

x_k	$f[x_k]$	$f[\cdot, \cdot]$	$f[\cdot, \cdot, \cdot]$	$f[\cdot, \cdot, \cdot, \cdot]$
0	0			
1	2	$f[0, 1] = \frac{f(1) - f(0)}{1 - 0} = 2$		
2	0	$f[1, 2] = \frac{f(2) - f(1)}{2 - 1} = -2$	$f[0, 1, 2] = \frac{f[1, 2] - f[0, 1]}{2 - 0} = -2$	
3	1	$f[2, 3] = \frac{f(3) - f(2)}{3 - 2} = 1$	$f[1, 2, 3] = \frac{f[2, 3] - f[1, 2]}{3 - 1} = \frac{3}{2}$	$f[0, 1, 2, 3] = \frac{f[1, 2, 3] - f[0, 1, 2]}{3 - 0} = \frac{3 + 2}{3} = \frac{7}{6}$

Las x que me dan → el valor de $f(x)$ en la x ant → grado 1 saltos de 1 → grado 2 saltos de 2 → grado 3 saltos de 3

2º Obtención del polinomio de interpolación mediante la fórmula de Newton

$$\begin{aligned}
 p_3(x) &= \underbrace{f[0]}_{p_0(x)} + \underbrace{f[0, 1](x-0)}_{p_1(x)} + \underbrace{f[0, 1, 2](x-0)(x-1)}_{p_2(x)} + f[0, 1, 2, 3](x-0)(x-1)(x-2) = \\
 &= \underbrace{2(x-0)}_{p_1(x)} - \underbrace{2(x-0)(x-1)}_{p_2(x)} + \frac{7}{6}(x-0)(x-1)(x-2)
 \end{aligned}$$

si desarrollamos esto obtendríamos el polinomio del estilo $a + bx + cx^2 \dots$

3.3 Fórmula de Hermite

valores de la función en unos puntos $f(x) = y$ y los de sus derivadas $f'(x) = y'$

- Diferencias divididas con NODOS REPETIDOS

Recordamos diferencias divididas de grado 1 de f en x_0 y x_1 :

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

Diferencias divididas de grado 1 de f en x_0 y x_0 : Al tener el valor de $f(x)$ y el de su derivada, tenemos $f[x_0, x_0]$ y eso da $f'(x_0)$ que es dato

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

Diferencias divididas de grado n de f en x_0, x_0, \dots, x_0 :

$$f[x_0, x_0, \dots, x_0] = \frac{f^n(x_0)}{n!}$$

- Tabla de diferencias divididas con NODOS REPETIDOS

x_i	$f[x_i]$	$f[\cdot, \cdot]$	$f[\cdot, \cdot, \cdot]$...
x_0	$f(x_0)$			
x_0	$f(x_0)$	$f[x_0, x_0] = f'(x_0)$...
x_1	$f(x_1)$	$f[x_0, x_1] = \frac{f(x_1) - f(x_0)}{x_1 - x_0}$	$f[x_0, x_0, x_1] = \frac{f[x_0, x_1] - f[x_0, x_0]}{x_1 - x_0}$	
x_1	$f(x_1)$	$f[x_1, x_1] = f'(x_1)$	$f[x_0, x_1, x_1] = \frac{f[x_1, x_1] - f[x_0, x_1]}{x_1 - x_0}$..

como $f(x)$ y $f'(x)$ son datos, ponemos la x dos veces

- Fórmula de Newton generalizada del polinomio de interpolación de Hermite

$$\begin{aligned} p_{n+1}(x) &= \underbrace{f[x_0]}_{P_0(x)} + \underbrace{f[x_0, x_0](x-x_0)}_{P_1(x)} + \underbrace{f[x_0, x_0, x_1](x-x_0)(x-x_1)}_{P_2(x)} + f[x_0, x_0, x_1, x_1](x-x_0)^2(x-x_1) \\ &\quad + \dots + f[x_0, x_0, x_1, x_1, \dots, x_n, x_n](x-x_0)^2(x-x_1)^2 \dots (x-x_n) \end{aligned}$$

Ejemplo 3: Construir el polinomio $p(x)$ de menor grado que interpola los datos de la siguiente tabla

x_i	1	2
$f(x_i)$	2	6
$f'(x_i)$	3	7
$f''(x_i)$	No dado	8

2º Fórmula generalizada de Newton del polinomio interpolador:

$$\begin{aligned} p(x) &= f[1] + f[1, 1](x-1) + f[1, 1, 2](x-1)^2 + f[1, 1, 2, 2](x-1)^2(x-2) + f[1, 1, 2, 2, 2](x-1)^2(x-2)^2 \\ &= 2 + 3(x-1) + (x-1)^2 + 2(x-1)^2(x-2) - (x-1)^2(x-2)^2 \end{aligned}$$

1º Tabla de diferencias divididas: Como me dan $f(1)$ y $f'(1)$ ponemos el 1 dos veces

Como me dan $f(2)$, $f'(2)$ y $f''(2)$ ponemos el 2 tres veces

1	2	ponemos lo que vale $f(x)$, no $f'(x)$ ni $f''(x)$
1	2	$f[1, 1] = f'(1) = 3$ Completamos igual que con newton pero cuando tengamos $f[1, 1]$ ponemos su derivada
2	6	$f[1, 2] = 4$ $f[1, 1, 2] = 1$ valores iguales (1, 1/ 2, 2)...
2	6	$f[2, 2] = f'(2) = 7$ $f[1, 2, 2] = 3$ $f[1, 1, 2, 2] = 2$
x_i	$f(x_i)$	$f[2, 2] = f'(2) = 7$ $f[2, 2, 2] = \frac{f''(2)}{2!} = 4$ $f[1, 2, 2, 2] = 1$ $f[1, 1, 2, 2, 2] = -1$

Ajuste de datos y funciones. Mejor aproximación mínimos cuadrados

1. Interpolación vs Ajuste de datos

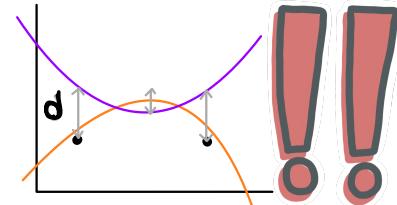
Interpolación n^o datos = n^o parámetros a calcular
sistema determinado
solución única

Ajuste n^o datos > n^o parámetros
sistema sobre determinado
infinitas soluciones

Entre las infinitas soluciones, debemos calcular la que "mejor se ajuste a los datos dados, es decir, aquella cuya distancia a esos valores sea mínima $p(x_i)$ lo + próximo a y_i para los puntos dados

$$|p(x_i) - y_i| \rightarrow \left(\sum_{i=1}^n |p(x_i) - y_i|^2 \right)^{1/2} \rightarrow \boxed{\sum_{i=1}^n (p(x_i) - y_i)^2} \text{ sea mínimo}$$

Buscamos el polinomio cuya distancia a los datos dados es menor



Consideraciones de estos problemas

- El sistema lineal esta SOBREDETERMINADO (+ec que incógnitas) tiene infinitas soluciones
- En general, NO HAY SOLUCIÓN EXACTA $Hc - b = 0$
- Buscamos el residuo mínimo, la solución de los mínimos cuadrados

En interpolación esa operación ($Hc - b$) nos daba 0 porque la función interpolante pasaba por esos puntos mientras que ahora el resultado será la distancia entre la función y los datos

2. Planteamiento del problema

INPUT: Conjunto discreto de datos numéricos de una función $f(x)$ o tabla de datos

OUTPU: Construir una función $\phi(x)$ de un determinado tipo que AJUSTE LO MEJOR POSIBLE los datos numéricos datos = error mínimo

3. Residuos. Errores

- Error o residuo en el dato i $e_i = |\phi(x_i) - y_i|$

- Vector de residuos $R = (e_1 \ e_2 \ \dots \ e_n) = |Hc - b|$

- Error cuadrático $E = \sum_{i=1}^n e_i^2 = \sum_{i=1}^n (c_1 b_1(x_i) + \dots + c_m b_m(x_i) - y_i)^2 = \|Hc - b\|_2^2$

$$\text{Error} = e = \sqrt{E} = \|R\|$$

En MATLAB

$$R = abs(H * c - b);$$

$$\text{Error} = norm(H * c - b)$$

EJEMPLO

%Datos a ajustar:
 $x_i = [-1 \ 0 \ 1 \ 2]^T$; $f_i = [-2 \ -1 \ 0 \ 3]^T$;

%Ajuste por $r(x)$:
 $H1 = [x_i.^0 \ x_i.^1]$; $c1 = H1 \ f_i$

%Ajuste por $p(x)$:
 $H2 = [x_i.^0 \ x_i.^1 \ x_i.^2]$; $c2 = H2 \ f_i$

%Ajuste por $q(x)$:
 $H3 = [x_i.^0 \ x_i.^1 \ x_i.^2]$; $c3 = H3 \ f_i$

% Residuos y errores

$$R1 = abs(f_i - H1 * c1); \quad \text{cuando } X=0 \rightarrow f_i = -1, \phi = -0,8 = -1 + 0,8 = 10,21$$

$$R2 = abs(f_i - H2 * c2);$$

$$R3 = abs(f_i - H3 * c3);$$

$$e1 = norm(R1)$$

$$e2 = norm(R2)$$

$$e3 = norm(R3)$$

% Gráficas

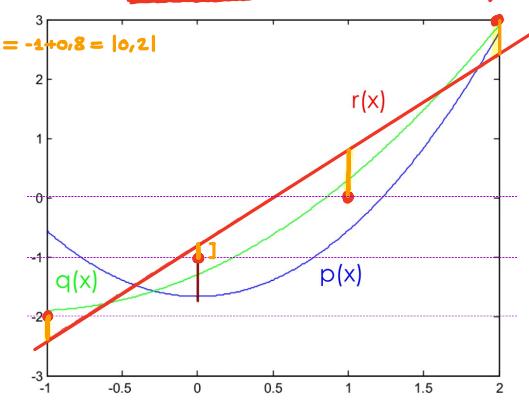
$$xx = -1:0.01:2;$$

$$p1 = c1(1) + c1(2) * xx;$$

$$p2 = c2(1) + c2(2) * xx.^2;$$

$$p3 = c3(1) + c3(2) * xx + c3(3) * xx.^2$$

$$\text{plot}(xx, p1, 'r', xx, p2, 'b', xx, p3, 'g', xi, fi, 'ro')$$



$$\begin{aligned} H &= (X_i^0 \ X_i^1) = \begin{pmatrix} 1 & 1 \\ 1 & 0 \\ 1 & 1 \\ 1 & 2 \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 2 \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \end{pmatrix} \rightarrow \\ H &= (X_i^0 \ X_i^1 \ X_i^2) = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \\ 1 & 2 & 4 \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \\ c_3 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 2 \\ 1 & 1 & 1 & 4 \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \\ c_3 \end{pmatrix} \rightarrow \\ H &= (X_i^0 \ X_i^1 \ X_i^2) \begin{pmatrix} c_1 \\ c_2 \\ c_3 \end{pmatrix} = \begin{pmatrix} 0 \\ 8 \\ 8 \end{pmatrix}; \quad \begin{cases} 4c_1 + 2c_2 + 0c_3 = 0 \\ 2c_1 + 6c_2 + 8c_3 = 8 \end{cases} \rightarrow \begin{cases} c_1 = -0,8 \\ c_2 = 1,6 \\ c_3 = 1,6 \end{cases} \end{aligned}$$

4. Resolución del problema

EJEMPLO

Dada la tabla, calcular la recta que mejor ajusta los datos $r(x) = a + bx$

Como Problema de Minimización

1º Calculamos a y b tales que

$$\text{Min} \sum_{i=1}^n (r(x_i) - y_i)^2 = \text{Min} \underbrace{((a-b+2)^2 + (a+1)^2 + (a+b-0)^2 + (a+2b-3)^2)}_{F(a,b)}$$

2º Operamos eso y lo simplificamos $F(a,b) = 4a^2 + 4ab + 6b^2 - 16b + 14$

3º Hacemos las **derivadas parciales** respecto de a y b

$$\frac{\partial F(a,b)}{\partial a} = 8a + 4b \quad \frac{\partial F(a,b)}{\partial b} = 4a + 12b - 16$$

recta q mejor se ajusta

$$r(x) = -0.8 + 1.6x$$

4º Las = 0 y resolvemos el sistema obteniendo a y b $\begin{cases} 2(4a+2b) = 0 \\ 2(2a+6b-8) = 0 \end{cases} \rightarrow a = -0.8, b = 1.6$ sust en r

Como Sistema Lineal Sobre determinado

1º Escribimos matricialmente el sistema lineal sobre determinado $Hc = b$ resultante

$$r(-1) = a - b = -2$$

$$r(0) = a = -1$$

$$r(1) = a + b = 0$$

$$r(2) = a + 2b = 3$$

$$\begin{bmatrix} 1 & -1 \\ 1 & 0 \\ 1 & 1 \\ 1 & 2 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} -2 \\ -1 \\ 0 \\ 3 \end{bmatrix}$$

2º Resolvemos las **ecuaciones normales** que vendrán dadas por $H'Hc = H'b$

$$\begin{array}{ccc} H & & b \\ \begin{bmatrix} 1 & 1 & 1 & 1 \\ -1 & 0 & 1 & 2 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} & \begin{bmatrix} -2 \\ -1 \\ 0 \\ 3 \end{bmatrix} \\ \hline H' & & \\ \begin{bmatrix} 1 & 1 & 1 & 1 \\ -1 & 0 & 1 & 2 \end{bmatrix} & \begin{bmatrix} 1 & 1 & 1 & 1 \\ -1 & 0 & 1 & 2 \end{bmatrix} & \begin{bmatrix} 4 & 2 \\ 2 & 6 \end{bmatrix} \end{array} \leftrightarrow \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} 0 \\ 4 \end{bmatrix} \quad a = -0.8, b = 1.6$$

En MatLab

% datos de la tabla TRASPUESTOS

$$xi = [-1 0 1 2]$$

$$yi = [-2 -1 0 3]$$

% matriz de coef (H) y termino indep del sistema (b)

$$H = [xi.^0 xi.^1]; \% r(x) = a + bx$$

$$b = yi;$$

% resolvemos el sistema

$$c = H \setminus b;$$

% guardamos el valor de a y b, 1º y 2º valor de c

$$a = c(1); \% \text{ el valor de } a \text{ será el } 1^\circ \text{ valor del vector } c \\ b = c(2); \% \text{ el valor de } b \text{ será el } 2^\circ \text{ valor del vector } c$$

% construimos la recta que mejor se ajusta

fprintf ("r(x) = %.1f + %.1fx", a, b); %.1fx nos da solo un decimal

$$\rightarrow r(x) = -0.8 + 1.6x$$

Problemas de ajuste CON RESTRICCIONES

1º Función aproximante

$$p(x) = c_0 + c_1x + c_2x^2 + c_3x^3$$

2º Imponemos las restricciones

$$p(0) = c_0 = 3$$

$$p''(0) = 2c_2 = 1 \rightarrow c_2 = 0.5$$

$$p(x) = 3 + c_1x + 0.5x^2 + c_3x^3 = yi$$

3º Escribimos el sistema $Hc = b$, agrupamos conocido en un lado, desconocido en otro

$$\begin{array}{l} c_1x_i + c_3x_i^3 = y_i - 3 - 0.5x_i^2 \\ \hline H \end{array} \quad b$$

5. Clases de funciones aproximantes

DATOS NUMÉRICOS: Tabla (xi,yi)

ALGUNOS TIPOS DE FUNCIONES APROXIMANTES DE INTERÉS:

1. $\varphi(x) = ae^{bx}$
2. $\varphi(x) = axe^{bx}$
3. $\varphi(x) = \frac{ax^2}{b+x^2}$
4. $\varphi(x) = \frac{a}{1+b\cos(x)}$

Estas funciones no derivan es sistemas lineales porque las incógnitas, a y b, se relacionan entre sí. Antes de resolver un problema de ajuste de este tipo, debemos linealizar el sistema, es decir, que las incógnitas no estén relacionadas entre sí.

EJEMPLO EXPONENCIALES para linealizarlo tomamos logaritmos

1. $\varphi(x) = ae^{bx}$

$$\left\{ \begin{array}{l} \varphi(x_i) = ae^{bx_i} = y_i \\ i=1, \dots, n \end{array} \right. \rightarrow \log a + bx_i = \log y_i$$

NO lineal (variables a y b)
Aplicamos logaritmos

Lineal (en variables A y b)

$$a \cdot e^{bx} = y \rightarrow \underbrace{\log a + \log e^{bx}}_{\log a + (*)} = y$$

- Se resuelve el sistema lineal $A + bx_i = \tilde{y}_i$

Observación: Los elementos de este sistema son:

- Matriz de coeficientes: $H = [x_i \ ^0 \ x_i]$
- Término independiente: $\tilde{y}_i = \log(y_i)$
- $c = H \tilde{y}_i$ ($A = c(1)$ y $b = c(2)$)

$\log a = A \xrightarrow{\text{def}}$

Una vez calculados A y b, se deshace la transformación logarítmica y se obtiene $a = e^A$

También se puede plantear como problema de minimización:

$$\text{Min} \sum_{i=1}^n (A + bx_i - \tilde{y}_i)^2$$

Residuo i = $|\varphi(x_i) - y_i| = |ae^{bx_i} - y_i|$



$$\underbrace{\log a + (*)}_{\log a + bx} = \log y$$

$$\underbrace{\log a}_{\log a} + \underbrace{bx}_{bx} = \underbrace{\log y}_{\log y}$$

EJEMPLO RACIONALES para linealizarlo pasamos el denominador al otro lado y agrupamos

3. $\varphi(x) = \frac{ax^2}{b+x^2} \rightsquigarrow \frac{ax^2}{b+x^2} = y \rightarrow ax^2 = y(b+x^2) (=yb+yx^2)$

$$\frac{ax^2 - by}{b+x^2} = y \rightarrow \text{desconocido}$$

$$\left\{ \begin{array}{l} \varphi(x_i) = \frac{ax_i^2}{b+x_i^2} = y_i \\ i=1, \dots, n \end{array} \right. \rightarrow \dots \rightarrow ax_i^2 - by_i = x_i^2 y_i$$

COLOCAMOS

$$ax_i^2 = y_i (b + x_i^2)$$

desconocido

$$ax_i^2 = y_i b + y_i x_i^2$$

conocido

Se resuelve el sistema lineal anterior con elementos $Hc = b$

- Matriz de coeficientes: $H = [x_i \ ^2 \ -y_i]$
- Término independiente: $\tilde{y}_i = x_i^2 y_i = (x_i \ ^2)^2 * y_i$
- $c = H \tilde{y}_i$ ($a = c(1)$ y $b = c(2)$)

agrupamos lo conocido en un lado y lo desconocido en otro

Residuo i = $|\varphi(x_i) - y_i| = \left| \frac{ax_i^2}{b+x_i^2} - y_i \right|$

6. Ajuste de datos con pesos

DATOS

- Tabla de datos (x_i, y_i)
- Tipo de función aproximante: $u(x)$
- Pesos: $w_i \geq 0$ asignados a los distintos puntos

Peso importancia
error en dato i -ésimo

PROBLEMA DE AJUSTE CON PESOS:

$$\text{Min} \left\{ \sum_{i=1}^n w_i [u(x_i) - y_i]^2 \right\}$$

Error/desviación/residuo
en dato i -ésimo

- Resolución mínimos cuadrados del sistema lineal 'con pesos': Multiplicamos las ecuaciones por la raíz cuadrada de los correspondientes pesos:

$$W^{1/2} H c = W^{1/2} b \quad \text{con} \quad W = \begin{pmatrix} w_1 & 0 & 0 \\ 0 & w_2 & 0 \\ 0 & 0 & \dots \\ 0 & 0 & w_n \end{pmatrix} \quad (\text{matriz diagonal})$$

- Ecuaciones normales: Multiplicamos (1) por $(W^{1/2} H)^T = H^T W^{-1/2}$

$$H^T W^{1/2} W^{1/2} H c = H^T W^{1/2} W^{1/2} b \quad \Leftrightarrow \quad H^T W H c = H^T W b$$



- Residuos: $\text{abs}(u(x_i) - y_i) = \text{abs}(Hc - b)$

EJEMPLO Ajustar los datos de la Tabla por un polinomio de grado 1 atendiendo a los pesos w_i .

- Plantear sistema a resolver.
- Plantear ecuaciones normales.
- Resolver con Matlab.

x_i	-1	0	1
y_i	2	3	7
w_i	1	1	1/2

1. Función aproximante: $p(x) = a + bx \rightarrow 2$ parámetros para ajustar los datos

2. Ecuaciones lineales incluyendo pesos ($\sqrt{W} H c = \sqrt{W} b$):

$$\begin{cases} \sqrt{w_i}(a + bx_i) = \sqrt{w_i}y_i & \leftrightarrow \\ i=1,2,3 \end{cases} \quad \begin{bmatrix} \sqrt{1} & 0 & 0 \\ 0 & \sqrt{1} & 0 \\ 0 & 0 & \sqrt{0.5} \end{bmatrix} \begin{bmatrix} 1 & -1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} \sqrt{1} & 0 & 0 \\ 0 & \sqrt{1} & 0 \\ 0 & 0 & \sqrt{0.5} \end{bmatrix} \begin{bmatrix} 2 \\ 3 \\ 7 \end{bmatrix}$$

3. Ecuaciones normales (incluyendo pesos): $(\sqrt{W} H)^T \sqrt{W} H c = (\sqrt{W} H)^T \sqrt{W} b \Leftrightarrow H^T W H c = H^T W b$

$$\underbrace{\begin{bmatrix} 1 & 1 & 1 \\ -1 & 0 & 1 \end{bmatrix}}_{H^T} \underbrace{\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0.5 \end{bmatrix}}_W \underbrace{\begin{bmatrix} 1 & -1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix}}_H \underbrace{\begin{bmatrix} a \\ b \end{bmatrix}}_c = \underbrace{\begin{bmatrix} 1 & 1 & 1 \\ -1 & 0 & 1 \end{bmatrix}}_{H^T} \underbrace{\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0.5 \end{bmatrix}}_W \underbrace{\begin{bmatrix} 2 \\ 3 \\ 7 \end{bmatrix}}_b$$

con MatLab

% datos

```
xi=[-1:1]';  
yi=[2 3 7]';  
wi=[1 1 0.5];
```

% sistema lineal con pesos a resolver $\sqrt{W} * H * c = \sqrt{W} * yi$

```
H=[xi.^0 xi];  
W=diag(wi); % W matriz diagonal con los pesos en la diagonal  
c=(sqrt(W)*H)\(sqrt(W)*yi);  
a=c(1)  
b=c(2)
```

% vector de residuos

```
res=abs(H*c-yi)
```

Resolución de ecuaciones no lineales

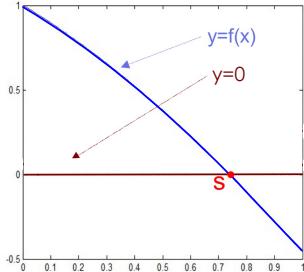
1. Introducción

OBJETIVO Calcular la solución/es de una ecuación no lineal $f(x) = 0$

Calculamos s tal que $f(s) = 0$ con una determinada precisión

- s es raíz de la ecuación $f(x) = 0$
- s es el cero de la función $f(x)$

Gráficamente, s es la abscisa punto de corte de las curvas $\begin{cases} y = f(x) \\ y = 0 \end{cases}$



METODOS ITERATIVOS x_0 inicial, $x_{k+1} = g(x_k)$

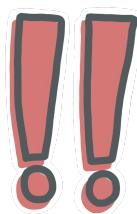
- Programamos x_0 , criterio de parada...
- Estudiamos la convergencia, la velocidad de convergencia y la estimación del error

2. Preliminares. Localizar raíces

Tma de Bolzano que NO se cumpla no implica que NO haya solución/es en el intervalo

Si $f(x)$ es

- continua en $[a, b]$
- $f(a)*f(b) < 0$ tienen signos contrarios



entonces $\exists s \in [a, b]$ donde $f(s) = 0$

$f(x)$ tiene al menos una raíz en el intervalo $[a, b]$

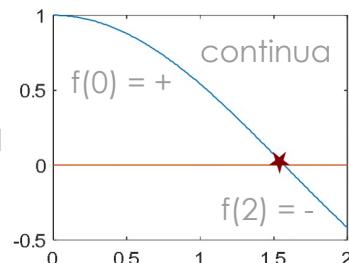
Basado en el Tma de Rolle

Si $f(x)$ es monótona en $[a, b] \rightarrow \exists s \in [a, b]$ es única

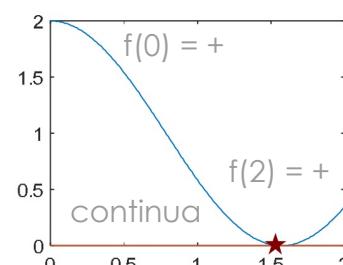
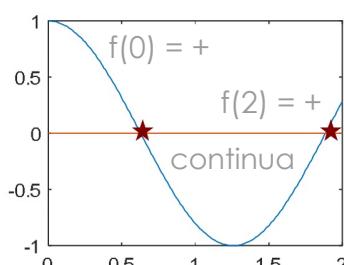
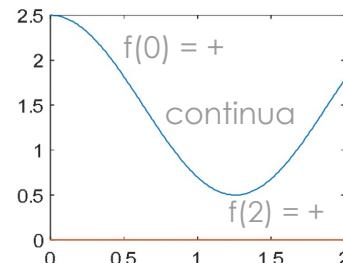
$f'(x)$ no cambia de signo en $[a, b]$, solo crece o decrece en el intervalo

cumple ambos teoremas luego la solución existe en el intervalo y es única

SI \exists solución en el intervalo



NO \exists solución en el intervalo



SI \exists solución en el intervalo

SI \exists solución en el intervalo

en estos dos NO se cumple bolzano pero hay solución/es en el intervalo

3. Generalidades. Métodos Iterativos

PROBLEMA Calcular el valor aprox de s que verifica $f(s) = 0$

Relación entre $f(x)$ y $g(x)$

$f(s)=0$ (s cero de f) $\longleftrightarrow s=g(s)$ (s punto fijo de g)

ALGORITMO ITERATIVO

$$\begin{cases} x_0 \text{ punto inicial} \\ x_{n+1} = g(x_n) \end{cases}$$

3.1 ESTIMACIÓN DEL ERROR en iteración n -ésima, ORDEN, y CONVERGENCIA

ERROR en iteración n $e_n = x_n - s $	ESTIMACIÓN $e_n \approx x_{n+1} - x_n $
--	--

CONVERGENCIA $e_n \rightarrow 0$? Para cualquier x_0 en $[a, b]$

En caso de convergencia, si $s = \lim (x_n)$ con $x_{n+1} = g(x_n) \rightarrow s = g(s)$

VELOCIDAD DE CONVERGENCIA/ORDEN

El método será de orden 1 (al menos) / método con convergencia lineal cuando el exponente del error calculado, vaya aumentando más o menos de 1 en 1

```
Iter 1 Sol 0.606530659712633 Error 3.93e-01  
Iter 2 Sol 0.738403149974731 Error 1.32e-01  
Iter 3 Sol 0.691286050428152 Error 4.71e-02  
Iter 4 Sol 0.707765096310001 Error 1.65e-02  
Iter 5 Sol 0.701957408706619 Error 5.81e-03  
Iter 6 Sol 0.703998745804550 Error 2.04e-03
```

El método será de orden 2 (al menos) / método con convergencia cuadrática cuando el exponente del error calculado, vaya aumentando más o menos de 2 en 2

```
Iter 1 Sol 1.000000000000000 Error 5.000000e-01  
Iter 2 Sol 0.733043605245445 Error 8.247401e-02  
Iter 3 Sol 0.703807786324133 Error 2.631407e-03  
Iter 4 Sol 0.703467468331798 Error 2.738812e-06  
Iter 5 Sol 0.703467422498392 Error 2.968181e-12
```

VALORACIÓN DE LA SOLUCIÓN OBTENIDA distancia entre iteraciones sucesivas

Se puede calcular como $f(x_n) \sim 0$?

PASOS para resolver los problemas

1º LOCALIZAMOS LAS RAÍCES

- Gráfica
- Teorema de Bolzano

ALGORITMO ITERATIVO

2º ITERAMOS BUSCANDO LA SOLUCIÓN s

- N veces (for)
- Hasta un error/precisión pedida (while)

$\begin{cases} x_0 \text{ punto inicial} \\ x_{n+1} = g(x_n) \end{cases}$ función iteradora

es lo que cambiará en función del método iterativo que usemos

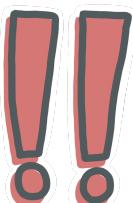
clásico, Newton-R o bisección

Bucle FOR

```
N = ;  
x0 = ;  
  
for k=1:N  
    x1 = g(x0);  
    e = abs(x1-x0);  
    fprintf();  
    x0 = x1;  
end
```

Bucle WHILE

```
N = ; x0 = ; e = ; k = 1;  
  
while (e >) & (k<=N)  
    x1 = g(x0);  
    e = abs(x1-x0);  
    fprintf();  
    x0 = x1;  
    k = k+1;  
end
```



EJEMPLO

a) Realizar una gráfica de la función $f(x) = x^2 - \exp(-x)$ en el intervalo $[-2, 2]$. Dar un intervalo aproximado con una longitud máxima de 1 donde se encuentre la raíz a partir de la gráfica. Demostrar analíticamente que en dicho intervalo existe al menos una raíz.

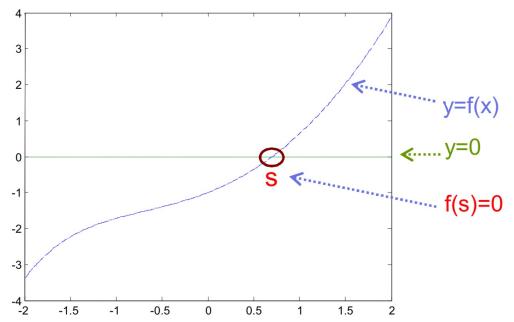
1º LOCALIZAMOS RAÍCES estudiamos si hay solución

% pintamos la función en $[-2, 2]$

$x = -2:0.01:2;$

$fx = x.^2 - \exp(-x);$

$plot(x, fx, 'b', x, fx*0, 'g');$



% en la gráfica observamos que la solución se encuentra en $[0, 1]$

% lo demostramos con bolzano

% fx es continua en $[0, 1]$

% son $f(0)$ y $f(1)$ de signos contrarios? $f(0)*f(1) < 0$?

$f0 = 0.^2 - \exp(-0);$

$f1 = 1.^2 - \exp(-1);$

$signo = f0*f1;$

% signo = -0.6321 -> SI cumple Tma Bolzano, existe al menos una raíz

MÉTODO ITERATIVO 1 - Método Clásico

b) Implementar y ejecutar el siguiente método para encontrar la raíz de $f(x)$

$$x_{n+1} = g(x_n) = \sqrt{e^{-x_n}} \quad \text{con } x_0 = 1 \quad \text{función iteradora}$$

El método deberá iterar hasta que el error $e_n = |x_n - s| \approx |x_n - x_{n-1}| < 1e-10$

En cada iteración imprimir: número de iteración, la raíz obtenida y el error (con el formato 'Iter %d Sol %.15f Error %0.2e\n')

2º ITERAMOS para encontrar la solución, el valor de s

% inicializamos todo lo necesario

% control

error = 100.0; damos este valor para asegurarnos de que entra en el bucle, luego se actualiza

% contador de iteraciones

k = 1;

% valor inicial

x0 = 1.0;

% método iterativo

while (error > 1e-10) como NO sabemos el número de iteraciones exactas, utilizamos un while

$x1 = sqrt(exp(-x0)); \quad \% x1 = g(x0)$

error = abs(x1-x0); % estimación del error se actualiza el error con su valor real

% imprimimos el nº de iteración, k, la raíz obtenida, x1 y el error

$\text{fprintf} ('Iter \%d Sol \%15f Error \%0.2e \n', k, x1, error);$

$k = k + 1; \quad \% aumentamos el contador$

$x0 = x1; \quad \% cambiamos el elem al siguiente$

end

$s = x0; \quad \% la solución es la última antes de superar el error pedido$

que se ha calculado

MÉTODO ITERATIVO 2 - **Método de Newton-R**

$$\left\{ \begin{array}{l} x_0 \text{ punto inicial} \\ x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} = g(x_n) \text{ función iteradora} \end{array} \right.$$

c) Implementar y ejecutar el siguiente método para encontrar la raíz de $f(x)$

$$x_{n+1} = x_n - ((x_n^2 - \exp(-x_n)) / (2*x_n + \exp(-x_n))) \text{ con } x_0 = 1 \text{ función iteradora}$$

El método deberá iterar hasta que el error $e_n = |x_n - s| < 1e-10$

tomando como s el valor final obtenido en el apartado anterior

En cada iteración imprimir: número de iteración, la raíz obtenida y el error (con el formato 'Iter %d Sol %.15f Error %e\n')

2º ITERAMOS para encontrar la solución, el valor de s

% inicializamos todo lo necesario

```
error = 100.0; % control
k = 1; % contador iteraciones
x0 = 1.0; % valor inicial
```

% método iterativo

```
while (error > 1e-10)
```

```
x1 = x0 - ((x0^2 - exp(-x0)) / (2*x0 + exp(-x0))); % x1 = g(x0) = x0 - f(x0) / f'(x0)
error = abs(x0 - x1); % estimación del error
```

```
fprintf('Iter %d Sol %.15f Error %.02e\n', k, x1, error);
```

```
k = k + 1; % avanzamos el contador de iteraciones
```

```
x0 = x1; % avanzamos al siguiente valor
```

```
end
```

ANÁLISIS DEL PROBLEMA

El método 1 ha necesitado 23 iteraciones para calcular la solución con la precisión pedida mientras que el método 2 solo ha necesitado 4

El método 1 cada dos iteraciones ganaba aproximadamente 1 cifra decimal de precisión, orden 1, mientras que el método 2 en cada iteración ganaba 2, orden 2

NOTA si NO nos dan la función iteradora $g(x)$, la sacamos nosotros a partir de $f(x)$

Si iteramos con el **método clásico** despejamos la x de mayor grado y lo del otro lado del = es $g(x)$

EJEMPLO $f(x) = x^2 - \exp(-x) = 0 ; x^2 = \exp(-x) ; x = \sqrt{\exp(-x)}$, $g(x)$

Si es con el **método de Newton Raphson**

$$g(x_k) = x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$$

Si es con el **método de la bisección**

$$g(x_k) = x_{k+1} = \frac{a_{k+1} + b_{k+1}}{2}$$

4. Método de la bisección divide el int/2 y selecciona el subintervalo que contiene la raíz

Suponemos que se cumple el T^{ma} de Bolzano y por tanto \exists una raíz s en [a, b]

ETAPA 1

$$[a_1, b_1] = [a, b] \rightarrow x_1 = \frac{a_1 + b_1}{2}, \quad f(x_1) \begin{cases} = 0 \rightarrow s = x_1 \text{ Fin} \\ \neq 0 \text{ siguiente etapa} \end{cases}$$

ETAPA 2

$$[a_2, b_2] = \begin{cases} [a_1, x_1] & \text{si } f(a_1)f(x_1) < 0 \\ [x_1, b_1] & \text{si } f(b_1)f(x_1) < 0 \end{cases} \rightarrow x_2 = \frac{a_2 + b_2}{2}, \quad f(x_2) \begin{cases} = 0 \rightarrow s = x_2 \text{ Fin} \\ \neq 0 \text{ siguiente etapa} \end{cases}$$

ETAPA k+1

$$[a_{k+1}, b_{k+1}] = \begin{cases} [a_k, x_k] & \text{si } f(a_k)f(x_k) < 0 \\ [x_k, b_k] & \text{si } f(b_k)f(x_k) < 0 \end{cases} \rightarrow x_{k+1} = \frac{a_{k+1} + b_{k+1}}{2}, \quad f(x_{k+1}) \begin{cases} = 0 \rightarrow s = x_{k+1} \text{ Fin} \\ \neq 0 \text{ siguiente etapa} \end{cases}$$

g(xk)

4.1 ALGORITMO

```

aprox de la raíz   función que evalúa f y f'
function s = biseccion(fun, a, b, N)   n° max de iteraciones
% evaluamos f(a) y f(b)   intervalo
fa = fun(a);
fb = fun(b);

if fa*fb > 0
    fprintf ('Método no aplicable en el intervalo [a,b] \n');
end

for i = 1:N
    c = (a+b)/2;
    fc = fun(c); % evaluamos f en punto medio del intervalo

    if fa*fc < 0
        b = c; %fb=fc; (no es necesario)
    else
        a = c;
        fa = fc;
    end
end

s = a; % podria ser = b ó = (a+b)/2
fprintf ('La raiz aproximada es %f \n', s);
end

```

function [f,fp] = fun(x)
f = ; % pondriamos f(x), la expresión
if nargout == 1, return; end
fp = ; % pondriamos f'(x), la expresión
end

Son tres scripts, uno llamado **biseccion**, otro **fun** con sus códigos correspondientes y el del ejercicio que resuelvas en cuestión que llamará a la función **biseccion** (ver ejemplo)

4.2 PROPIEDADES. OBSERVACIONES

Coste computacional: una evaluación de f(x) en cada iteración.

Error. Error en la iteración n-ésima:

$$e_n = |x_n - s| \leq \frac{1}{2} |b_n - a_n| \leq \frac{1}{2^2} |b_{n-1} - a_{n-1}| \leq \dots \leq \frac{1}{2^n} |b - a|$$

(En cada iteración el tamaño del subintervalo de búsqueda se reduce a la mitad)

Convergencia garantizada para cualquier x0 en intervalo [a,b]:

$$e_n = |x_n - s| \leq \frac{1}{2^n} |b - a| \xrightarrow{n \rightarrow \infty} 0$$

Velocidad convergencia: Método de orden 1 $\frac{e_{n+1}}{e_n} \approx \frac{1}{2}$

En cada it gana 1 dígito binario de precisión, 0.3 cifras dec

Nº iteraciones que garantizan el cálculo de **la solución** con una prec determinada (el error sea menor que una tolerancia tol fijada):

$$e_n = |x_n - s| \leq \frac{1}{2^n} |b - a| < tol \rightarrow 2^n > \frac{|b - a|}{tol} \rightarrow n > \log\left(\frac{|b - a|}{tol}\right) / \log(2)$$

No es aplicable a cálculo raíces múltiples (no cambio signo).

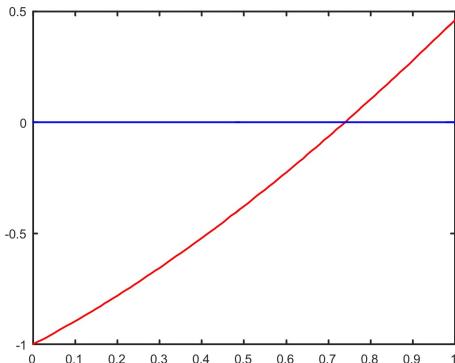
EJEMPLO

Aplicar método de la bisección para el cálculo de la raíz de la ecuación $f(x) = x - \cos(x) = 0$ en $[0,1]$

1º Comprobamos que hay una raíz en dicho intervalo (Bolzano)

- $f(x)$ es continua en $[0,1]$
 - $f(0)f(1) = -0.4597 < 0$
- } -> Existe (al menos) una raíz en el intervalo $[0,1]$

Además, gráficamente



2º Aplicamos el método de la bisección en $[0,1]$

Etapa k	x_k solución aproximada	$f(x_k)$	Cota error
Etapa 1. $[a_1, b_1] = [0, 1]$, $f(0) < 0$ y $f(1) > 0$;	$x_1 = (a_1 + b_1)/2 = 0.5000000000000000$	$f(x_1) = -3.78e-001 < 0$	$Error_1 < 1/2^1$
Etapa 2. $[a_2, b_2] = [1/2, 1]$, $f(1/2) < 0$ y $f(1) > 0$;	$x_2 = (a_2 + b_2)/2 = 0.7500000000000000$	$f(x_2) = 1.83e-002 > 0$	$Error_2 < 1/2^2$
Etapa 3. $[a_3, b_3] = [1/2, 0.75]$,	$x_3 = (a_3 + b_3)/2 = 0.6250000000000000$		$Error_3 < 1/2^3$
...	$x_4 = 0.6875000000000000$		
	$x_5 = 0.7187500000000000$		
	$x_6 = 0.7343750000000000$		

En cada iteración se gana 1 dígito binario de precisión

¿Cuántas iteraciones son necesarias para estimar la solución con un error menor que 10^{-8} ?

ver código en la última hoja

$$e_n = |x_n - s| \leq \frac{1}{2^n} |b - a| \leq \frac{1}{2^n} < 10^{-8} \Rightarrow -n \log_{10}(2) < -8 \Rightarrow n \log_{10}(2) > 8 \Rightarrow n > \frac{8}{\log_{10}(2)} = 26.575$$

Por tanto, $n \geq 27$ iteraciones garantizan que se calcula la solución con la precisión pedida

EN MATLAB

```

Editor - D:\MATLAB\tema4\ejemplo.m
1 function s = biseccion(fun, a, b, N)
2 % evaluamos f(a) y f(b)
3 fa = fun(a);
4 fb = fun(b);
5
6 if fa*fb > 0
7 fprintf ('Metodo no aplicable en el intervalo [a,b] \n');
8 end
9
10 for i = 1:N
11 c = (a+b)/2;
12 fc = fun(c); % evaluamos f en punto medio del intervalo
13 if fa*fc < 0
14 b = c; %fb=fc; (no es necesario)
15 else
16 a = c;
17 fa =fc;
18 end
19 end
20 s = a;
21 fprintf ('La raíz aproximada es %f \n', s);

ejemplo.m
1 % EJEMPLO
2 clear
3 clc
4
5 % LOCALIZAMOS LAS RAICES
6 % Gráficamente
7 x = 0:0.01:1;
8 fx = x - cos(x);
9 plot (x, fx, 'b', x, fx*0, 'g');
10
11 % Bolzano
12 f0 = 0 - cos(0);
13 f1 = 1 - cos(1);
14 signo = f0*f1; % como es negativo, hay solucion en el []
15
16 % 2 OBTENEMOS s - METODO DE LA BISECCION
17 % Llamamos a la función
18 biseccion(@fun, 0, 1, 6);

fun.m
1 function [f,fp] = fun(x)
2 f = x - cos(x);
3 if nargin == 1, return; end
4 fp = 1 + sin (x);
5 end

```

Workspace:

Name	Value
ans	0.7344
f0	-1
f1	0.4597
fx	1x101 double
signo	-0.4597
x	1x101 double

Command Window:

```

La raíz aproximada es 0.734375
>>

```

5. Método de Newton-Raphson $\left\{ \begin{array}{l} x_0 \text{ punto inicial} \\ x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} = g(x_n) \end{array} \right.$

5.1 IDEA GEOMÉTRICA. ALGORITMO

x_0 punto inicial

ETAPA 1 $f(x) \approx p_1(x) = f(x_0) + f'(x_0)(x - x_0)$ (Recta tangente a $y=f(x)$ en $(x_0, f(x_0))$)

$$f(x)=0 \rightarrow \text{Se resuelve } p_1(x)=0 \rightarrow x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$$

...

ETAPA $k+1$ $f(x) \approx p_{k+1}(x) = f(x_k) + f'(x_k)(x - x_k)$ (Recta tangente a $y=f(x)$ en $(x_k, f(x_k))$)

$$f(x)=0 \rightarrow \text{Se resuelve } p_{k+1}(x)=0 \rightarrow x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)} = g(x_k)$$

EJEMPLO Aplicar el método de Newton para calcular la raíz cuadrada de 2, sin calcular raíces cuadradas.

Ecuación a resolver: $f(x) = x^2 - 2 = 0$

Método de Newton-R.:

x_0 inicial

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)} = x_k - \frac{x_k^2 - 2}{2x_k} = \frac{x_k^2}{2} + \frac{1}{x_k}$$

CÓDIGO estructura

Bucle FOR

$N = ;$

$x0 = ;$

```
for k=1:N
    x1 = g(x0); x1 =  $\frac{g(x0)}{x0 - (f(x0)/f'(x0))}$ 
    e = abs(x1-x0);
    fprintf();
    x0 = x1;
end
```

Bucle WHILE

$N = ;$

$x0 = ;$

$e = ;$

$k = 1;$

```
while (e > ) & (k<=N)
    x1 = g(x0); x1 =  $\frac{x0 - (f(x0)/f'(x0))}{g(x0)}$ 
    e = abs(x1-x0);
    fprintf();
    x0 = x1;
    k = k+1;
end
```

5.2 ESTUDIO DEL ERROR. CONVERGENCIA. VELOCIDAD CONVERGENCIA

COTAS ERROR $e_n = |x_n - s| \approx |x_{n+1} - x_n|$

Relación entre etapa n y distancia entre iteraciones sucesivas

$$- \boxed{e_n \leq m e_{n-1}^2} \quad (1)$$

$$- \boxed{\boxed{e_n \leq m e_{n-1}^2 \leq \frac{1}{m} (m e_{n-1})^2 \leq \frac{1}{m} (m e_{n-2})^{2^2} \leq \dots \leq \frac{1}{m} (m e_0)^{2^n}}} \quad (2)$$

CONVERGENCIA del método

Si se inicializa el método con x_0 tal que $me_0 < 1$, esto es $e_0 < 1/m$:

$$e_n \leq \frac{1}{m} (me_0)^{2^n} \xrightarrow{n \rightarrow \infty} 0$$

el método converge.

- En caso de convergencia y si s es raiz simple ($f'(s) \neq 0$), el método es ORDEN 2:

$$e_{n+1} \leq me_n^2 \rightarrow \lim_{n \rightarrow \infty} \frac{e_{n+1}}{e_n^2} = \frac{|f''(s)|}{2|f'(s)|} \approx \text{constante} \neq 0 \quad \text{si } f'(s) \neq 0$$

OBS A "groso modo" en cada iteración se duplican el nº de decimales significativos:

$$-\log_{10}(e_{n+1}) \approx 2(-\log_{10}(e_n)) + \log_{10}(1/\text{constante})$$

Cifras dec. en iter. n+1

Cifras dec. en iter. n

- Nº de iteraciones que garantizan un error menor que una tolerancia tol

$$e_n \leq \frac{1}{m} (me_0)^{2^n} < tol \rightarrow \text{Aplicamos logaritmos y despejamos n:}$$

$$2^n \log(me_0) < \log(tol) + \log(m) \xrightarrow{me_0 < 1 \rightarrow \log(me_0) < 0} 2^n > \frac{\log(tol)}{\log(me_0)} \rightarrow n > \log\left(\frac{\log(tol)}{\log(me_0)}\right) / \log(2)$$

¿Cuántas iteraciones son necesarias para estimar la solución con un error menor que 10^{-8} ?

EN MATLAB

The screenshot shows the MATLAB environment with several windows open:

- Editor - D:\MATLAB\tema4\biseccion.m**: The code for the bisection method. It includes a while loop that continues until the error is less than 10^{-8} . Inside the loop, it calculates the midpoint c , evaluates the function at c , and updates the interval based on the sign of the function values at a , b , and c .
- Command Window**: Displays the output: "La raíz aproximada es 0.739085 y el nº de it necesarias 28".
- Workspace**: Shows variables and their values: ans (0.7391), t0 (-1), f0 (0.4597), f1 (1x101 double), fx (1x101 double), signo (-0.4597), and x (1x101 double).
- ejemplo.m**: A script file containing code to find roots using the bisection method and Bolzano's method, and to plot the function.
- fun.m**: A function file for the function $f(x) = x - \cos(x)$.

Como no sabemos el número de iteraciones, usamos un while y le ponemos la condición del error

A la hora de llamar a la función, en vez de meterle una N como último parámetro, metemos un valor del error para asegurarnos de que entra en el bucle