MÉTODO DE JACOBI

OBJETIVO: resolver $Ax = b$

$$x_i^{(k+1)} = \frac{b_i - \sum_{j=i,\, j\neq i}^{n} a_{ij} x_j^{(k)}}{a_{ii}}$$

vectorización

$$X^{(k+1)} = D_A^{-1}\left(b - (A - D_A)X^{(k)}\right)$$

$$\underbrace{\mathrm{diag}(1./\mathrm{diag}(A))}_{D_A^{-1}} * \underbrace{(\mathrm{diag}(\mathrm{diag}(A))}_{(\,D_A} - A$$

$$\phi = \frac{b}{D_A} = b./\mathrm{diag}(A)$$

Como $B_J(A) = \boxed{D_A^{-1}(D_A - A)}$, $\boxed{\phi = D_A^{-1} \cdot b}$

$$X^{(k+1)} = D_A^{-1}(D_A - A) \cdot X^{(k)} + D_A^{-1} \cdot b$$

$$X^{(k+1)} = B \cdot X^{(k)} + \phi$$

MÉTODO DE GAUSS-SEIDEL

$$x_i^{(k+1)} = \frac{b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k)}}{a_{ii}}$$

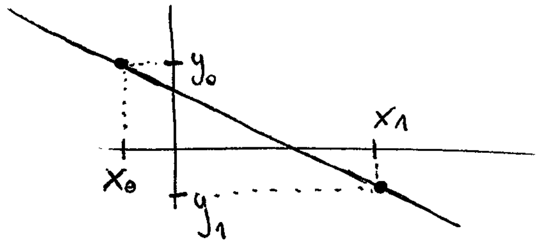
```
while k
    for i=1:n
        for j= ... Σ
```

```
x = ones(n,1)
x1 = x;
while (err > tol and nIter < maxIte
    nIter ++;
    for i = 1:n
        sumat1 = 0;
        sumat2 = 0;
        for j = 1:i-1
            sumat1 = sumat1 + A(i,j)*
        end
                                        x1 (
                                        cual
        for j = i+1:n
            sumat2 = sumat2 + A(i,j) * x(
        end
        x1(i) = (b(i)-sumat1-sumat2)/A(
    end
    err = max(abs(x1-x));
    x = x1;
end
```

CONSTRUCCIÓN DEL POLINOMIO INTERPOLADOR

Método de Lagrange

(x_0, y_0) , (x_1, y_1)



$$P_1(x) = \frac{x-x_0}{x_1-x_0} y_1 + \frac{x-x_1}{x_0-x_1} y_0$$

$\{(x_j, y_j)\}_{j=0}^{n} \subset \mathbb{R}^2$

$$\boxed{\underline{P_n}(x) = \sum_{j=0}^{n} \Big(\underbrace{\prod_{k=0,\,k\neq j}^{n} \frac{x-x_k}{x_j-x_k}}_{\longrightarrow\ \underline{L_j}(x) = \prod_{k=0,\,k\neq j}^{n} \frac{x-x_k}{x_j-x_k}} \Big) y_j}$$

function pLag

INPUT

- $\{(x_j, y_j)\}_{j=0}^{n}$ nodos +
- punto a evaluar "a".

OUTPUT

$P_n(a) \in \mathbb{R}$

function Lj

INPUT

- $\{(x_j)\}_{j=0}^{n}$ nodos (solo coord x
- punto a evaluar "a"

OUTPUT

$L_0(a), \ldots, L_n(a) \in \mathbb{R}^{n+1}$

function graficaLag

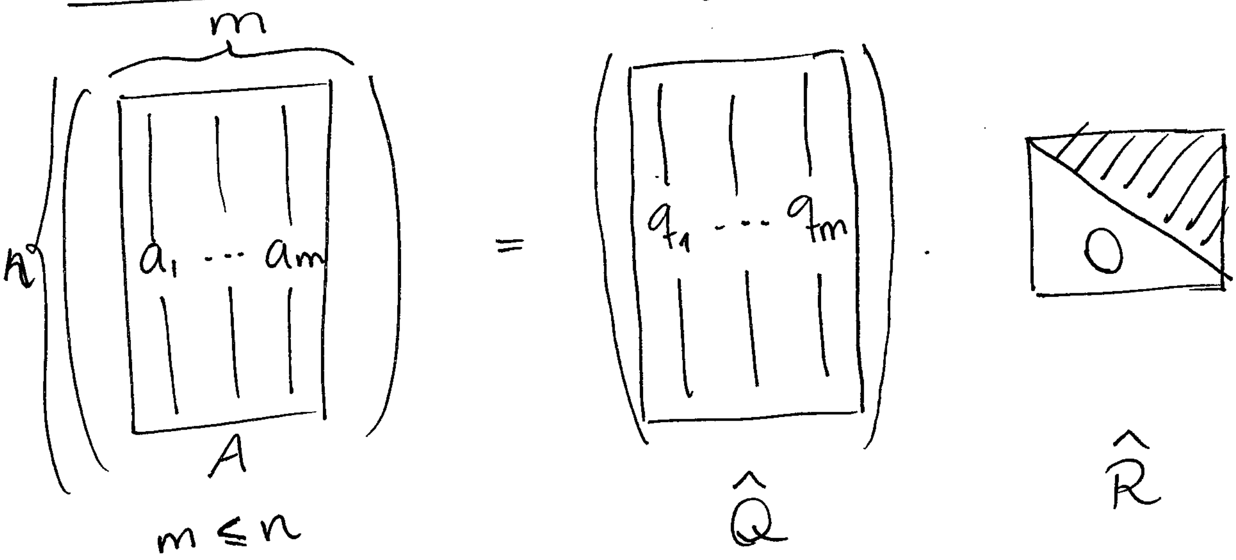
INPUT

- f
- $\{(x_j)\}_{j=0}^{n}$ nodos (solo coord x) $\longrightarrow$ la coord $y_j = f(x_j)$

OUTPUT

la gráfica

FACTORIZACIÓN QR y APLICACIONES



$m \leq n$

$\text{Rang}(A) = m$

- $\mathcal{L}(\{a_j\}_{j=1}^{K}) = \mathcal{L}(\{q_j\}_{j=1}^{K})$

  $\langle q_i, q_j \rangle = \delta_{ij}$

- R triangular superior y invertible

---

1) ALGORITMO QR

$$\begin{pmatrix} 1 & 2 & 3 \\ 2 & 5 & 6 \\ 3 & 6 & 8 \end{pmatrix}$$

$A \Rightarrow [Q^{(0)}, R^{(0)}] = \text{factQR}(A)$

$A^{(1)} = R^{(0)} * Q^{(0)}$

$[Q^{(1)}, R^{(1)}] = \text{factQR}(A^{(1)})$

$\vdots$

$A^{(K+1)} = R^{(K)} * Q^{(K)}$

$[Q^{(K+1)}, R^{(K+1)}] = \text{factQR}(A^{(K+1)})$

INPUT: A , OUTPUT: Q, R

$r_{11} = \|a_1\|_2$, $q_1 = a_1 / r_{11}$

for $k = 2:n$

  $v_k = a_k$

  for $j = 1:k-1$

    $r_{jk} = \langle a_k, q_j \rangle$

    $v_k = v_k - r_{jk} q_j$

  end

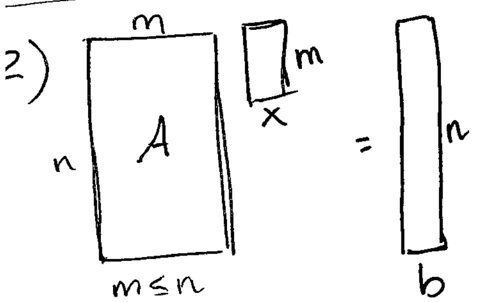
  $r_{kk} = \|v_k\|_2$

  $q_k = v_k / r_{kk}$

end

---

2)



$A = Q * R$, $(\overline{Q^t}) * Q = I$ $\boxed{Q^* = Q' \text{ matlab}}$

Entonces:

$$\underbrace{Q^* A}_{\shortparallel} x = Q^* b$$

$$Rx = Q^* b \longrightarrow x = R^{-1} * Q^* * b$$

---

3) polinomio interpolador

$$p(x) = a_1 + a_2 x + a_3 x^2 + \cdots + a_n x^{n-1}$$

$$n \left\{ \underbrace{\begin{pmatrix} 1 & s_1 & \cdots & s_1^{m-1} & \cdots & s_1^n \\ \vdots & \vdots & & \vdots & & s_2^n \\ & & & & & \vdots \\ 1 & s_n & \cdots & s_n^{m-1} & \cdots & s_n^n \end{pmatrix}}_{m} \right. \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_m \\ a_n \end{pmatrix} = \begin{pmatrix} f(s_1) \\ f(s_2) \\ \vdots \\ f(s_n) \end{pmatrix}$$

IN: f, $\{s_j\}_{j=1}^n$. $m < n$

OUT: $\{a_k\}_{k=1}^m$

CONSTRUCCIÓN POLINOMIO INTERPOLADOR

- MÉTODO DE LAGRANGE

$$\{(x_j, y_j)\}_{j=0}^{n} \subset \mathbb{R}^2$$

$$P_n(x) = \sum_{j=0}^{n} \left(\prod_{k=0, k\neq j}^{n} \frac{x - x_k}{x_j - x_k} \right) y_j \qquad L_j(x) = \prod_{k=0, k\neq j}^{n} \frac{x-x_k}{x_j-x_k}$$

$$P_n(x) = \sum_{j=0}^{n} L_j(x) y_j$$

L_j

INPUT
- $\rightarrow \{(x_j)\}_{j=0}^{n}$ NODOS (coord. x)
- $\rightarrow$ punto a evaluar: "a"

OUTPUT $\rightarrow L_0(a), L_1(a), \dots, L_n(a)$

PsC

1. Comprobar que los nodos que nos pasan es $n\times 1$ ó $1\times n$ (un vector, no matriz)
2. n = length(Xvector)
3. Inicializar vector ljVector:
   ljVector = ones(n,1)
4. Bucle j = 1:n
   4.1. Bucle k = 1:n
   if (j~=k)
5. ljVector(j) = ljVector(j) * ((a - Xvector(k))/(Xvector(j) - Xvector(k)));

polLagrange

INPUT
- $\rightarrow \{(x_j, y_j)\}_{j=0}^{n}$ nodos e imágenes
- $\rightarrow$ punto a evaluar: "a"

OUTPUT: $\rightarrow P_n(a) \in \mathbb{R}$

PsC

1. Comprobar que Xvector e Yvector son realmente vectores.
2. auxV = Lj(Xvector, a)
   $\rightarrow$ vector columna
3. dot(auxV, Yvector)

(*) dot() hace el prod. escalar y dentro ya hace las comprobaciones de las dimensiones. Podríamos haber hecho nosotros el producto escalar vectorizado auxV * Yvector, pero antes tendríamos que hacer las comprobaciones del vector Y, ya que dependiendo de si es fila o columna tendríamos que hacer Yvector * auxV ó auxV' * Yvector respectivamente.

graficaPolLagrange

INPUT:
- → función f (handle function → tiene que servir para vectores)
- → $\{(x_j)\}_{j=0}^{n}$ nodos (solo coord. x) → la coord $y = f(x_j)$

OUTPUT → gráfica

PsC

1. Comprobar que Xvector es un vector
2. n = length(Xvector)
3. Yvector = f(Xvector)
4. auxV = Xvector(1) : 0.1 : Xvector(n) ⊛ → comprobar que Xvector(n) > > Xvector(1)
   m = length(auxV)
5. Bucle i=1:m
   Pvector(i) = polLagrange(Xvector, Yvector, auxV(i));
6. Mostramos gráficas → gráfica real de f → polinomio interpolador generado → nodos (puntos)
   6.1. fplot(f, [Xvector(1), Xvector(n)], 'r') función real
   hold on; → misma gráfica
   6.2. plot(auxV, Pvector, 'b') polinomio generado
   6.3. plot(Xvector, Yvector, '*m') nodos

testLagrange

1. f = function handle
2. a = primer punto intervalo
3. b = fin intervalo
4. n = nº nodos
5. → NODOS EQUIESPACIADOS
   s = a + (0:n) * (b-a)/n ;
   → NODOS DE CHEBYSHEV
   t = cos(0.5 * pi * (2*(0:n) +1)/(n+1)); en [-1, 1] → [a,b]
   s = 0'5 * (b-a) * (cos(0'5*pi* (2*(0:n) +1)/(n+1))) + 0'5*(b-a)
6. graficaPolLagrande (f, s)

• MÉTODO DE NEWTON

difDivididas

INPUT:
- f: function handler
- Xvector: nodos (coord. x)

OUTPUT:
- matriz de diferencias divididas
- diagonal de la matriz (diferencias divididas)

PsC
1. Comprobar que Xvector es un vector
2. n = length(Xvector)
   matriz = zeros(n)
3. Primera columna:
   - matriz(1:n,1) = f(Xvector)
   - for i=1:n
     matriz(i,1) = f(X(i))
4. Resto matriz
   4.1 Bucle j=2:n
   4.2. Bucle K=j:n
   matriz(K,j) = matriz(K,j-1) - matriz(K-1,j-1)./ Xvector(K) - Xvector(K-
   - VECTORIZADO
     matriz(j:n,j) = matriz(j:n,j-1) - matriz((j:n)-1, j-1)./ Xvector(j:n) -
     - Xvector((j:n)-
5. return diag(matriz), matriz

PolNewton

INPUT:
- f: function handler
- Xvector: nodos (coord. x)
- a: punto a evaluar

OUTPUT
- p: polinomio
- difDiv

PsC
1. Comprobar que Xvector es un vector
2. difDiv = difDivididas(f, Xvecto
3. n = length(Xvector)
4. newt=1 → auxiliar producto
5. p = difDiv(1) * newt
   5.1 Bucle j=2:n
   newt = newt .* (a- Xvector
   p = p + difDiv(j) * new

grafica + test Newton

1. a = inicio intervalo
2. b = fin intervalo
3. n = nº nodos
4. → NODOS EQUIESPACIADOS

   s=a + (0:n) *(b-a) /n

   → NODOS CHEBYSHEV

   t= cos(0.5 *pi * (2*(0:n) +1) / (n+1) en [-1,1]

   s = 0.5 * (b-a) * t + 0.5* (b-a) en [a,b]

5. x = a: (b-1)/(10000) :b
6. f = @(x) ______
7. p = polNewton(f, s, x) → puntos a evaluar

   function → nodos

8. plot (x,p) → polinomio

   hold on

   plot (x, f(x), 'r') → función real

   plot (s, f(s), '*r') → nodos

Integración

integración_simple_ab

INPUT:
- f: function handler
- a: inicio intervalo
- b: fin intervalo
- W: vector de pesos → [-1,1]
- X: vector de nodos
  - en [-1,1]

OUTPUT:
- resultado de la integral

PsC

1. Comprobar que x y w son realmente vectores
2. Comprobar que x y w son vectores del mismo tamaño
3. Ventorizando el producto de f(-).w:
   - si rowX > rowW   X columna W fila
     - res = w * f(0.5 * (b-a) * x + 0.5(b+a))
   - elseif rowX < rowW   X fila W columna
     - res = f(0.5 * (b-a) * x + 0.5(b+a)) * w
   - else X y W mismo tipo de vector (fila o columna)
     - si rowX > colX   ambos columna
       - res = w' * f(0.5 * (b-a) * x + 0.5(b+a))
     - else   ambos fila
       - res = f(0.5 * (b-a) * x + 0.5(b+a)) * w'
4. res = res * 0.5 * (b-a)

(*) El apartado 3 se puede simplificar utilizando dot(-)

integracion_compuesta_ab

INPUT:
- f: function handler
- a: vector de intervalos. En cada uno se va a realizar una integracion simple y sumar resulta
- w: vector de pesos } ∈ [-1
- x: vector de nodos

OUTPUT:
- resultado de la integral

PsC

1. Comprobar que x y w son vectores
2. Comprobar que x y w son vectores del mismo tamaño.
3. tam = length(a)
4. res = 0
   - for k = 1:tam-1
     - res += integ-simple_ab (
       - (f, a(k), a(k+1), w, x)

integracion_variada

INPUT:
- f: function handler
- a: vector de intervalos

UTPUT:
- medio
- trapecio
- simpson
- gauss2
- gauss3

SC

Medio
- f
- a
- 2
- 0

Trapecio
- f
- a
- [1,1]
- [1,-1]

impson
- f
- a
- $\left[\frac{1}{3}, \frac{4}{3}, \frac{1}{3}\right]$
- [-1,0,1]

auss2
- f
- a
- [1,1]
- $[1/\sqrt{3}, -1/\sqrt{3}]$

auss3
- f
- a
- [5/9, 8/9, 5/9]
- $[-\sqrt{3/5}, 0, \sqrt{3/5}]$

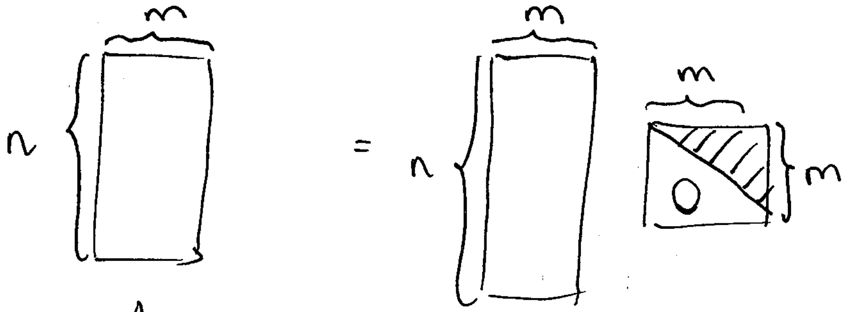
test_integracion

1. f = function handler
2. a = inicio intervalo
3. b = fin intervalo
4. TOP = nº de intervalos
5. for N=2 : TOP
   V= a: (b-a)/N-1 : b
   → nodos equiespaciados
   [m,t,s, g2,g3] = integr_variada(f,V)
6. subplot(2,2,1); plot(2:TOP, m(2:TOP))
   subplot(2,2,2); plot(2:TOP, t(2:TOP))
   subplot(2,2,3) ⋮
   subplot(2,2,4)
7. I= g3(TOP) → mejor aproximacion
8. Mostramos errores:
   m(2:TOP) = abs(m(2:TOP) - I)
   ⋮

figure
subplot(2,2,1) ; plot(2:TOP , log10(m(2:TOP)))
⋮
→ mejor para representación de errores

QR

factorizacionQR — usa GramSmidt



A → $m \leq n$ = → Rang(A) = m   Q   R

- $\mathcal{L}\left(\{a_j\}_{j=1}^{k}\right) = \mathcal{L}\left(\{q_j\}_{j=1}^{k}\right)$
  $\langle q_i, q_j \rangle = \delta_{ij}$
- R triangular superior e invertible

INPUT
→ A

OUTPUT
→ Q,R

PsC

1. n = nº de filas de A
   m = nº de columnas de A
2. Comprobar que $m \leq n$ y que Rang(A) = m → comprobación opcional
3. Inicializamos Q = zeros(n,m)
   R = zeros(m,m)
4. R_{11} = norm (A(:,1)) → norma 1ª col A
5. $Q(:,1) = \dfrac{A(:,1)}{R_{11}}$
6. for k=2:m

```
    v = A(:,k)
    for j = 1:k-1
        R(j,k) = dot(Q(:,j), A(:,k))
        R(j,k) = Q(:,j)' * A(:,k)
        v = v - R(j,k) .* Q(:,j)
    R(k,k) = norm(v)
    Q(:,k) = vector ./ R(k,k)
```

dot

a mano (valores complejos)

opcional

opcional

autovalores QR

Iterando la factorizacion QR podem
obtener los autovalores de A.

INPUT
→ A
→ maxIter
→ (tol)

OUTPUT
→ autovalores
→ nIter

PsC

1. Comprobar que A es una matriz cuadr
2. Comprobar que A es una matriz hermít
   max(max(abs(A-A'))) debe ser cero.
3. nIter=0

```
mientras nIter < maxIter
    Si A diagonal:
        break
    [Q,R] = factQR
    A = R*Q
autovalores = diagonal(A)
```

→ B = abs(A - diag(diag(A)))
Si max(max(B)) < tol
break

Resolver sistema incompatible

INPUT
- → A
- → b

OUTPUT
- → vector de soluciones X

PsC

1. $[n,m] = size(A)$
2. Si $m > n$ → ERROR (A tiene más cols que filas)

   A debe tener más filas que columnas

3. Si $length(b)\ != n$ → ERROR

   b tiene que tener el mismo tamaño que el nº de filas que A

   Si b es vector fila:
   $b = b'$

$[Q,R] = factQR(A)$

1. Mi solución

   $X = pinv(R) * Q' * b;$

   R es invertible por def.

2. Solución profesor

```
y = Q' * b
x = y./diag(R)
for i = m-1:-1:1
    for j = i+1:m
        x(i) = x(i) - R(i,j) * x(j) / R(i,i)
```

Polinomio_incompatible

$$p(x) = a_1 + a_2 x + a_3 x^2 + \cdots + a_n x^{n-1}$$

$$n\left\{\underbrace{\begin{pmatrix} 1 & S_1 & \cdots & S_1^{m-1} \\ \vdots & \vdots & & \vdots \\ 1 & S_n & & S_n^{m-1} \end{pmatrix}}_{m} \begin{matrix} \cdots S_1^n \\ \vdots \\ \cdots S_n^n \end{matrix}\right. \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{pmatrix} = \begin{pmatrix} f(S_1) \\ f(S_2) \\ \vdots \\ f(S_n) \end{pmatrix}$$

INPUT
- → $b = f(s)$
- → nodos $\equiv$ S
- → m (menor que n)

OUTPUT
- → coeficientes $a_1, \dots, a_n$

OJO: b es un vector de mediciones: puede ser f(nodos) = f(s) o unas mediciones más inexactas y se busca el polinomio más ajustable a esos puntos.

PsC

1. Comprobar que b es un vector, y si es fila trasponerlo.
2. $n = length(b)$
   ```
   V(:,1) = ones(n,1)
   for K = 2:m
       V(:,K) = S.^(K-1) = nodos^(K-1)
   ```
3. $a = solve\_incompatible(V, b)$
4. $x = min(nodos) : \frac{max(nodos) - min(nodos)}{1000} : max(nodos)$
5.
   ```
   for i = 1:length(x)
       p(i) = 0
       for K = 1:m
           p(i) = p(i) + a(K) * x(i)^(K-1)
   ```
6. figure; plot(s,b,'*r'); hold on; plot(x,p)

posible test pol_incomp

- f = function handler ← grado máximo m (m+1 nodos)
- S = -5:0.1:5
- b = f(s) + randn(size(s))
- pol-incomp(b, S, m+1)

- no imprimir en pantalla el resultado → ; al final
- limpiar → clc
- format long g (mayor precisión)
- format long e (exponencial)
- format short (menor precisión)
- eliminar variable → clear nombre_variable
- parte entera → floor (algo)
- techo → ceil (algo)
- redondeo estándar → round (algo)
- número más grande → realmax = $1'7977 \cdot 10^{308}$
- número más pequeño → realmin = $2'2251 \cdot 10^{-308}$
- borrar todo el workspace → clear

- vector fila → vf = [1 2 3 4]
- vector columna (traspuesta fila) → vc = vf'
- componente por componente → z = v.^2

  el punto

  z = v.*w (multipl. coord. por coord)

- vector columna → vc = [1;2;3;4]

- v = 1:5 → v = [1 2 3 4 5]
- v = 1:2:10 → v = [1 3 5 7 9]

  desde de hasta

  v(1) v(2) v(3) v(4) v(5)

  (*) NO EXISTE v(0)

- v(2:5) → v(2) v(3) v(4) v(5)
- Sumar todas las componentes de un vector → sum(v)
- " " " " " " " en v.absoluto → abs(v)
- Norma de un vector → norm(v)
- Dimensiones vector → size(v)
- Tamaño → length(v)

- Para saber como funciona una función → help nombre_función
- Mayor información → doc nombre_función

- exponencial → exp (algo)

- Editor → New > Script > undock
- Comentarios → %

(*) LAS FUNCIONES DECLARADAS EN UN FICHERO PARA PODER EJECUTARSE EN OTROS TIENEN QUE TENER EL MISMO NOMBRE QUE EL ARCHIVO.

- para devolver cosas en funciones

```
function [ret1, ret2] = nombre(par1, ...)
    [código
end
```

el igual que no se olvide

En el return:

[r, E] = nombre (par1, ...)

r = nombre (par1, ...) (almacena solo la 1ª)

[~, E] = nombre (par1, ...) (almacena solo E)

alt gr+4 para no almacenar variables

```
x = -5:0.01:5;
y = exp(-x.^2);
plot(x,y)
figure(2)
```
→ otra salida (solo para el siguiente plot) → quizá sea para siempr

```
x = -5:0.5:5;
y = exp(-x.^2);
```
→ modificamos el tamaño de x, entonces también el de y.

```
plot(x,y)
```
→ ploteamos en figure 2

```
plot(x,y,'*')
```
→ ploteamos en (figure 1) con asteriscos. ?

```
grid on
```
→ cuadrícula

```
f = @(x) x.^2 + 7*x + 12
```
(FUNCTION HANDLE)

```
f(4)
```
→ evaluación (da el resultado)

```
fplot(f, [a,b])
```
$[a,b]$ → intervalo de representación

f → function handle

HOJA 1 - EJ.6 $\qquad f(x) = e^x \cdot \log(1 + e^{-x}) \qquad x \in [0,40]$

```
f = @(x) exp(x).*log(1 + exp(-x))
ylim([0 2])
```
→ $y \in [0,2]$ en la gráfica

SUBPLOTS

```
figure(1);
subplot(2,1,1); fplot(f,[-1:1]); grid on
subplot(2,1,2); fplot(df, [-1:1]); grid on
```
(2 → nº plots verticalmente; 1 → nº plots horizontalmente; 1 → id del plot)

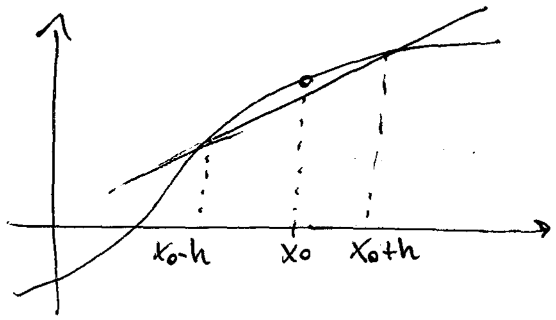
```
hold on;
⋮
hold off;
```
→ gráficas en la misma imagen

EJERCICIO CLASE $\to$ function nanales
$\to$ matlab scripts y functions
$\to$ errores de cancelación

Dada una $f \in C^1((x_0-H, x_0+H))$.
Calcular la mejor aproximación numérica de $f'(x) = \lim_{h\to 0} \frac{f(x_0+h)-f(x_0-h)}{2h}$

$$f'(x_0) = \frac{f(x_0+h_2) - f(x_0-h_2)}{2h_2}$$

Si la h es demasiado grande:



la recta secante varía mucho de la derivada

Si h es demasiado pequeño se restan dos números iguales $\to$ ERRORES DE CANCELACIÓN

function DERIV. $\to$ input: f, x_0, h
$\to$ output: $f'(x_0)$

OBSERVACIÓN

$f \in C^3((x_0-H, x_0+H))$

$$f(x+h) = f(x) + f'(x)\cdot h + \frac{f''(x)}{2}\cdot h^2 + \frac{f'''(x)}{6}\cdot h^3 + o(h^3)$$

$$f(x-h) = f(x) - f'(x)\cdot h + \frac{f''(x)}{2}\cdot h^2 - \frac{f'''(x)}{6} h^3 + o(h^3)$$

$$\frac{f(x+h)-f(x)}{h} = f'(x)\cdot\cancel{h} + \frac{f''(x)}{2}\cdot h^{\cancel{2}} + o(h^2) \quad (1)$$

$$\frac{f(x+h)-f(x-h)}{2h} = \cancel{2}\cdot f'(x)\cdot\cancel{h} + \frac{f'''(x)}{\cancel{6}12}\cdot h^{\cancel{3}\,2} + o(h^3) \quad (2)$$

Errores menores con (2)

$$\underbrace{\begin{pmatrix} & & \\ & & \\ & & \end{pmatrix}}_{A_{3\times3}} \underbrace{\begin{pmatrix} \\ \\ \\ \end{pmatrix}}_{x_{3\times1}} = \underbrace{\begin{pmatrix} \\ \\ \\ \end{pmatrix}}_{b_{3\times1}}$$

FACTORIZACIÓN LU con PIVOTAJE

```
L=I ;  U=A
for  k=1:n
     [~,j] = max(abs(U(k:n,k)));
     if j~=1 :
        fila k <-> fila i* de U,L,P
     end
     for i=k+1:n
         L(i,k) = U(i,k) / U(k,k);
         U(i,k:n) = L(i,k) * U(k,k:n);
     end
end
```

- buscar a cada paso $i^* = \underset{i\in\{k,k+1,\dots,n\}}{\operatorname{argmax}} \left|U^{(k-1)}_{(i,k)}\right|$

↳ construir la matriz de permutación:

$$\Pi_k(i) = \begin{cases} i & \text{si } i \neq k, i^* \\ k & \text{si } i = i^* \\ i^* & \text{si } i = k \end{cases}$$

$[\text{valor}, \text{pos}] = \max(\text{abs}(U(\underbrace{k:n}_{\text{filas desde } k \text{ a } n}, \underbrace{k}_{\text{col}})))$

Solve

$$LUx = b \Rightarrow \begin{cases} Ly = b \\ y = Ux \end{cases}$$

$$P^{-1}LUx = b \Rightarrow \begin{cases} P^{-1}Ly = b \\ y = Ux \end{cases}$$

$\cancel{y(i) = y(i) - P^{-1}(j,i) * L(i,j) * y(}$

$PAx = Pb$ porque $PA = LU$

Solve

$A = LU$

$$Ax = b \iff LUx = b \Rightarrow \begin{cases} Ly = b \\ y = Ux \end{cases}$$

Solve P

$PA = LU$

$$Ax = b \iff PA\cancel{x} = Pb \iff LUx = Pb \Rightarrow \begin{cases} Ly = Pb \\ y = Ux \end{cases}$$

- function handle booleana :

$f = @(x) \quad x>=3$

$1<3 \quad 2<3 \quad 3>=3 \quad 4>=3 \quad 5>=3$

Entonces : $f(1:5) = [0 \; 0 \; 1 \; 1 \; 1]$

MATRICES

$$A = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{pmatrix}$$

fila 3 — elementos 3 al 5

$a = A(3 ; 2:4)$

$a = 10 \quad 11 \quad 12$

$a = A(2:3 ; 2:4)$

$$a = \begin{pmatrix} 6 & 7 & 8 \\ 10 & 11 & 12 \end{pmatrix}$$

$b = \text{sum}(A) \rightarrow$ suma de columnas

$b = [15 \quad 18 \quad 21 \quad 24]$

$c = \text{sum}(\text{sum}(A)) \rightarrow$ suma de todos los elementos de una matriz

triu(A) → parte triangular alta de A (diagonal-arriba, resto ceros)
tril(A) → parte triangular baja de A (diagonal-bajo, resto ceros)

EVITAR PROPAGACIÓN DE ERRORES

```
d(f, x0, h)
        temp = x0+h;
        h = temp - x0;
            ⋮
return
```