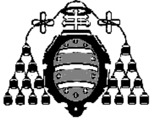


UNIVERSIDAD DE OVIEDO

Departamento de Ingeniería Eléctrica,
Electrónica, de Computadores y Sistemas

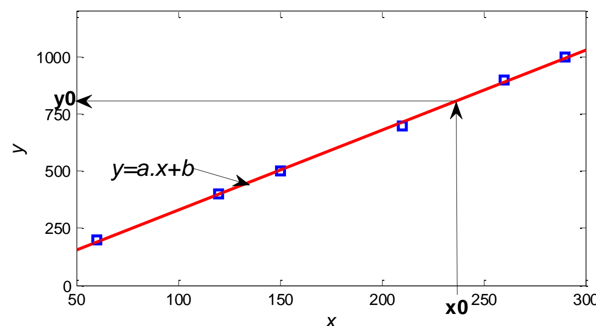
INGENIERÍA DE SISTEMAS Y AUTOMÁTICA

Guía de Prácticas

ASIGNATURA: Informática Industrial y Comunicaciones
CENTRO: Escuela Politécnica de Ingeniería de Gijón
ESTUDIOS: Grado en Ingeniería Electrónica y Automática
CURSO: 3º CUATRIMESTRE: 1
CARÁCTER: Obligatoria CRÉDITOS ECTS: 6
PROFESORADO: Ignacio Alvarez, José Mª Enguita, Angel Navarro, Mariam Saeed

PRACTICA 04: Arrays

1. Ejercicio a realizar: pedir por teclado dos tablas de valores *data_x* y *data_y*, que pueden tener como máximo 20 datos, y calcular a partir de ellos la recta de ajuste por mínimos cuadrados. Una vez obtenida dicha recta, ir pidiendo nuevos valores para *x* y calculando el valor de *y* que le correspondería a cada *x* según el ajuste.



2. Utilizar como referencia para el algoritmo: Ajuste por minimos cuadrados.pdf
3. Detectar funciones que pueden ser necesarias. Crear en archivos aparte (FnTablas.c y FnTablas.h). **Realizar una a una y probar con main( ) sencillos.**

float Sum(const float v[],int n);	$s = \sum_{i=0}^{n-1} v[i]$
float Sum2(const float v[],int n);	$s2 = \sum_{i=0}^{n-1} v[i] * v[i]$
float ProdEsc(const float v1[],const float v2[],int n);	$pe = \sum_{i=0}^{n-1} v_1[i] * v_2\,[i]$

Recordar:

- Que se hacen las funciones de forma que sean independientes del tamaño de la tabla.
- Que sólo existe noción del tamaño de la tabla en su declaración como variable local: a partir de ahí, en el programa sólo existe la dirección de comienzo. Las funciones sólo reciben la dirección de comienzo, por lo que se debe añadir el tamaño como segundo argumento.
- El uso de **const**, ya que al pasar la dirección la función está accediendo a los datos originales y no a una copia.

4. Función RegresionLineal( ):

¿? RegresionLineal(const float x[],const float y[],int n);	$a = \frac{n.ProdEsc(x,y,n) - Sum(x,n).Sum(y,n)}{n.Sum2(x,n) - Sum(x,n) * Sum(x,n)}$ $b = \frac{Sum(y,n) - a.Sum(x,n)}{n}$

Tener en cuenta que cada invocación a función vuelve a calcular sus resultados. Por ejemplo, Sum(x,n) aparece 4 veces en la expresión anterior. Mejor guardar en nuevas variables aquellos valores cuyo cálculo se necesite en varias ocasiones.

¿? → Problemática de devolver 2 valores desde una función (a,b). Dos opciones:

Código	Uso

struct recta
{
    float pend,y0; // pend=a , y0=b
};
struct recta RegresionLineal(const float x[],const float y[],int n);
	USAR ESTA OPCION SOLO SI EL RESULTADO TIENE SENTIDO COMO ESTRUCTURA (lo que ocurre en este caso)

void RegresionLineal(const float x[],const float y[],int n, float* ptA,float* ptB);	USAR ESTA OPCION SI LOS RESULTADOS NO TIENEN SENTIDO COMO ESTRUCTURA (como en el ejemplo del punto 7)

Invocación de la función desde main(). Si se han utilizado estructuras ...

```
int main() {
  float data_x[...],data_y[...];
  int n_datos;
  struct recta mi_recta;
  ... Pedir datos ...
  mi_recta=RegresionLineal(data_x,data_y,n_datos);
  ...
}
```

Si no se han utilizado estructuras…

```
int main() {
  float data_x[...],data_y[...];
  int n_datos;
  float a_recta,b_recta;
  ... Pedir datos ...
  RegresionLineal(data_x,data_y,n_datos,&a_recta,&b_recta);
  ...
}
```

5. Función ValorRecta( ). Si se han utilizado estructuras…

float ValorRecta(struct recta r,float x);	*y=r.pend*x+r.y0*

Si no se han utilizado estructuras…

float ValorRecta(float a,float b,float x);	*y=a*x+b*

AMPLIACIONES PROPUESTAS:

6. Realizar una función para pedir los datos de un vector, dado su tamaño, y reescribir la parte inicial del programa para que la utilice. La salida de la función sería una tabla, pero como no es posible, se le pasa la dirección inicial para que la rellene (tabla no const):

```
void PideTabla(float t[],int n);
```

7. Mejorar la función RegresionLineal( ) para que devuelva un código de error (int) si no puede hacer "bien" su trabajo, lo que ocurrirá en los casos:

- n < 2 (una recta requiere 2 puntos o más)
- $n.\left(\sum x_i^2\right)-\left(\sum x_i\right)^2 == 0$ (todos los puntos son el mismo)

Para devolución de error junto al resultado, se necesita nuevamente devolver 2 valores:

- No usar estructuras, no tiene sentido agrupar el resultado y el código de error bajo un mismo tipo de datos para el resto del programa, por tanto se usa la 2ª opción del punto 4.
- Dos opciones nuevamente:
  - Devolver con return la recta y por puntero el código de error:

struct recta RegresionLineal (*resto de params*, int *pt_err);	*struct recta r;* *int err;* *r=RegresionLineal(x,y,20,&err);* *if (err==NO_ERROR)* *lo que sea;*

  - Devolver con return el código de error y por puntero la recta:

int RegresionLineal (*resto de params* , struct recta* pt_r);	*struct recta r;* *int err;* *if (RegresionLineal(x,y,20,&r)==NO_ERROR)* *lo que sea;*

- Para el valor de error devuelto, también dos opciones:
  - Fallo = 0, Ok ≠ 0 . Facilita el tratamiento como true/false en if, pero no se puede distinguir la causa de fallo ya que sólo hay un valor (cero):

```
if ( RegresionLineal(.....) )
        acciones si Ok
    else
        acciones si error
```

  - Ok = 0, Fallo ≠ 0 . Facilita la gestión de códigos de error:

```
#define ERROR_N_MENOR_QUE_2    1
#define ERROR_DENOMINADOR_0    2
#define ERROR_NINGUNO           0

int error;
error=RegresionLineal(.....);
if (error==ERROR_NINGUNO)
   acciones si Ok
else
   acciones en función del valor de error
```

  - Para el 2º caso, mejor utilizar switch / case en lugar de if. Su formato es:

switch (error)
{
case ERROR_NINGUNO:
acciones si Ok;
break;
case ERROR_N_MENOR_QUE_2:
acciones para error n < 2;
break;
case ERROR_DENOMINADOR_0:
acciones para error por denominador 0;
break;
default:
acciones para otros valores de error;
break;
}
	¡¡¡ ATENCIÓN !!!
	La expresión del switch (en este caso el valor de error) se evalúa sólo una vez, y se 'salta' directamente a su caso.
	Los valores de los casos deben ser constantes, no pueden ser valores de variables ni resultados de expresiones