Problemas resueltos de C

Problemas de Matemáticas: cálculo y álgebra

1. Escriba una función que, pasándole un número entero de hasta 10 cifras decimales, retorne un número entero con las cifras decimales en orden inverso.

   Solución:

```
unsigned int invierte(unsigned int x) {
    int y, u;
    y=0;
    while( x > 0 ) {
        u= x % 10;
        y=10*y+u;
        x= x / 10;
    }
    return y;
}
```

2. El siguiente programa debe calcular la suma de dos matrices definidas en la función `main()`. Definir la función `suma()` para que funcione correctamente.

```
#include <stdio.h>

int main(int argc, char** argv) {
   double A[3][3]={{1.,2.,3.},{4.,5.,6.},{7.,8.,9.}};
   double B[3][3]={{1.,0.,0.},{0.,1.,0.},{0.,0.,1.}};
   double C[3][3];

   suma(A,*B,C[0]);

   printf("La suma de las matrices A y B es\n%g\t%g\t%g\n%g\t%g\t%g\n%g\t%g\t%g\n",
   C[0][0],C[0][1],C[0][2],C[1][0],C[1][1],C[1][2],C[2][0],C[2][1],C[2][2]);

   return 0;
}
```

   Solución:

```
void suma(double A[][3], double *B, double *C) {
    int i, j;
    for (i=0; i<3; i++) {
        for (j=0; j<3; j++)
            *(C+3*i+j)=A[i][j]+*(B+3*i+j);
    }
```

```
        return;
    }
```

3. En un determinado programa estamos trabajando con vectores en el espacio tridimensional definidos a través de estructuras del siguiente modo:

```
struct Vector {
    double x, y, z;
};

typedef struct Vector vector;
```

Defínase una función que calcule (y devuelva) el producto vectorial de dos vectores cualesquiera, introducidos como argumentos de la función.

Solución:

```
Vector producto_vectorial(Vector a, Vector b) {
    Vector c;
    c.x=  a.y*b.z - a.z*b.y;
    c.y= -a.x*b.z + a.z*b.x;
    c.z=  a.x*b.y - a.y*b.x;

    return c;
}
```

4. Supongamos que estamos trabajando con vectores en el espacio tridimensional definidos a través de estructuras del siguiente modo:

```
struct Vector {
    double x, y, z;
};

typedef struct Vector vector;
```

Definir una función, que llamaremos `cambioDeBase()`, que transforme (sin devolver nada) las componentes del vector **r** al aplicar el cambio de base dado por la matriz **C** de números reales. Las nuevas componentes estarán dadas por el producto $\mathbf{C} \cdot \mathbf{r}$. El vector **r** está declarado como:

```
vector r;
```

y la función será llamada desde el programa principal del siguiente modo:

```
cambioDeBase(&r, C);
```

Solución:

```
void cambioDeBase(vector *r, double C[][3]) {
    int j;
    vector aux;
    aux.x=C[0][0]*(r->x)+C[0][1]*(r->y)+C[0][2]*(r->z);
    aux.y=C[1][0]*(r->x)+C[1][1]*(r->y)+C[1][2]*(r->z);
    aux.z=C[2][0]*(r->x)+C[2][1]*(r->y)+C[2][2]*(r->z);

    *r=aux;
}
```

5. Supongamos que tenemos un archivo de datos `datos.dat` con N pares de puntos que representan el comportamiento de una función $y(x)$ en un intervalo dado. La primera columna del archivo corresponde a los valores x_i (que están ordenados de menor a mayor) mientras que la segunda muestra los valores de la función $y(x_i)$. Escribir un programa que lea esos puntos y obtenga los valores x_i en los que la función muestra los mínimos y máximos relativos dentro del intervalo. El valor de N debe ser introducido por línea de comandos como argumento de la función `main()` cuando se ejecuta el programa. El programa debe mostrar en pantalla el resultado del siguiente modo (es un ejemplo):

   `La función tiene tres mínimos relativos en los puntos x = 0.1, 1.2, 4.5`

   `La función tiene dos máximos relativos en los puntos x = 0.9, 2.9`

   Del mismo modo el programa también deberá indicar si la función no tiene mínimos ni máximos.

   Solución: Ver código `funcion_maximos_minimos.c`

6. El objetivo de este ejercicio es definir una función que escriba en un archivo de texto los valores de cualquier función matemática $f(x)$, que será pasada por referencia cuando llamemos a la función, en los puntos x_i pertenecientes al intervalo $[a, b]$ y separados por una distancia Δx. La función debe escribir en el archivo las parejas de puntos $(x_i, f(x_i))$ para luego poder ser representados con Gnuplot.

   Los argumentos de la función serán los extremos del intervalo así como la distancia entre los puntos, que serán pasados por valor, mientras que la función matemática que se quiere representar será pasada por referencia. Finalmente, se pasará también como argumento el nombre del archivo en el que se quieren escribir los datos. Una posible llamada a la función desde el programa principal podría ser

   ```
   tabla_valores (1.0, 2.0, 0.1, &parabola, "datos.dat");
   ```

   donde previamente se debería haber definido la función parábola, por ejemplo:

   ```
   double parabola(double x) {
           return (2*x*x-1);
   }
   ```

   Solución: Ver código `funcion_exportar_puntos.c`

7. Señale cuáles son las 5 líneas de código erróneas en la función siguiente, pensada para calcular el módulo de un vector, y en qué consisten dichos errores:

```
1 double f(x[4]) {
2     int i;
3     double y;
4     for(i=1; i<=4; i++)
5         y+=y+x[i]*x[i];
6     return sqr(y);
7 }
```

   Solución: Línea 1: falta declarar el tipo del vector `x[]`. Línea 3 (por ejemplo): falta inicializar el valor de `y`. Línea 4: el recorrido de `i` debe comenzar en `0`, y debe llegar hasta `3` (`i<4`). Línea 5: o bien se incrementa `y` (con `y+=`) o bien se calcula la nueva y a partir de la previa (`y=y+`), pero no ambas. Línea 6: la raíz cuadrada se calcula en C con la función `double sqrt(double)` de la `<math.h>`.

8. Complete los puntos suspensivos en las líneas indicadas en la siguiente función para que el resultado del cálculo sea el número de celdas activas (con valor #definido ON) en las dos matrices pasadas, ambas de tamaño NxN (valor también #definido).

```
int coincidenciasON(int A[N][N], int B[N][N]) {
    int i,j, c=0;
    for(i=...) {
        for(j=...) {
            if( A[i][j] ... )
                c++;
        }
    }
    return c;
}
```

Solución:

```
int coincidenciasON(int A[N][N], int B[N][N]) {
    int i,j, c=0;
    for(i=0; i<N; i++) {
        for(j=0; j<N; j++) {
            if( A[i][j]==ON && B[i][j]==ON )
                c++;
        }
    }
    return c;
}
```

9. Indique qué guarda la siguiente función en el archivo.

```
void comoCSV(char* fn, char** cols, int n, int m, double** vals) {
    FILE *f=fopen(fn, "wt");
    int i, j;
    fprintf(f, "%s", cols[0]);
    for(j=0; j<m; j++) {
        fprintf(f, ",%s", cols[j]);
    }
    fprintf(f, "\n");
    for(i=0; i<n; i++) {
        fprintf(f, "%g", vals[i][0]);
        for(j=1; j<m; j++) {
            fprintf(f, ",␣%g", vals[i][j]);
        }
        fprintf(f, "\n");
    }
    fclose(f);
}
```

Solución: Guarda, en la primera línea, las cadenas que contiene `cols`, separadas por comas (los nombres de las columnas). En las restantes líneas del archivo, guarda los valores de la matriz `vals` fila a fila, separados por comas.

10. Construya una función en C que, dado un vector de $n+1$ coeficientes $a[\,]$ y un valor de la variable x, evalúe la función polinomial

$$P(x) = a[0] + a[1] * x + a[2] * x * x + \cdots + a[n] * \underbrace{x * \cdots * x}_{n \text{ factores}}$$

Solución:

```
double evalPoly(int n, double a[], double x) {
    int m;
    double y=a[0], xm=1;
    for(m=1; m<=n; m++) {
        xm=xm*x;
        y+=a[m]*xm;
    }
    return y;
}
```

11. Construya una función en C que, dados los $2n+1$ coeficientes a_0, $a_1, \ldots a_n$ y $b_1, \ldots, b_n$, evalúe el polinomio trigonométrico:

$$F(x) = a_0 + \sum_{i=1}^{n} (a_i \cos ix + b_i \sin ix)$$

Solución:

```
double evalTrig(int n, double a[], double b[], double x) {
    int m;
    double y=a[0];
    for(m=1; m<=n; m++) {
        y+=a[m]*cos(m*x)+b[m]*sin(m*x);
    }
    return y;
}
```

12. Escriba una función que, dada una matriz **A** y un vector(fila) **v**, calcule un vector $\mathbf{w} = \mathbf{v} \cdot \mathbf{A}$. Además de **v** y **A**, qué otros parámetros requeriría la función, en el caso general? [**Nota:** considerar que la matriz **A** nunca tendrá más de `NMAX` filas o columnas; considerar también que es responsabilidad del programador que use la función que los argumentos de ésta sean los adecuados para realizar el cálculo]

Solución: Para calcular el producto, es necesario conocer el número de filas y columnas de **A**. Además, el número de filas de **A** debe coincidir con la dimensión del vector **v**. El vector retornado, tendrá como dimensión el número de columnas de **A**.

```
void pvecMat(double v[], int nf, int nc, double A[][NMAX], double w[])
    {
    double s;
    double i,j;
    for(j=0; j<nc; j++) {
        s=0.0;
        for(i=0; i<nf; i++) {
            s=s+v[i]*A[i][j];
        }
    w[j]=s;
    }
}
```

13. Escribir una función que, dados el día y mes de un año, y la hora, minutos y segundos transcurridos de ese día, calcule el número de días y fracción transcurridos desde el 1 de enero (fecha juliana), retornando un único valor de tipo `double`. [**Sugerencia:** usar un array global con el número de días de cada mes] [**Nota:** no es necesario tener en cuenta que el año sea bisiesto]

Solución:

```
static int dias_mes[]={31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
double fechaJ(int mes, int dia, int hora, int mint, int segd) {
    int n;
    double jdate=0;
    for(n=0; n<mes-1; n++) {
        jdate+=dias_mes[n];
    }
    jdate+=dia-1;
    jdate+=((double)hora+(double)mint/60+(double)segd/3600)/24;
    return jdate;
}
```

14. La sucesión de Fibonacci: 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, ... es la sucesión infinita de números naturales en la que sus dos primeros términos son unos y a partir de ahí cada elemento es la suma de los dos anteriores: $F_{n>2} = F_{n-1} + F_{n-2}$. Se sabe que $\varphi = \lim_{n\to\infty} \frac{F_{n+1}}{F_n}$, donde φ es el número áureo $\varphi = \frac{1+\sqrt{5}}{2} \simeq 1.6180339887\ldots$, un número irracional con propiedades matemáticas muy interesantes y que representa una proporción que aparece frecuentemente en la naturaleza.

    Construir un programa en C que calcule los términos de esta sucesión hasta obtener el valor de φ con una precisión de n cifras decimales. El programa debe pedir al usuario la precisión deseada. [**Ayuda:** Para saber que hemos llegado a la precisión deseada podemos imponer la siguiente condición en cada iteración: cuando el módulo de la diferencia entre dos valores F_{n+1}/F_n consecutivos de sea menor que 10^{-n}, eso significará que hemos llegado a la precisión deseada]

    Solución: Ver código `fibonacci.c`

15. Explicar detalladamente qué realiza el siguiente programa:

```
#include <stdio.h>
#include <math.h>

#include <stdio.h>
double *funcion(double *v) {
    int i, j;
    double num;
    num=v[0];
    i=0;
    for(j=1; j<6; j++) {
        if (*(v+j) < num) {
            num=*(v+j);
            i=j;
        }
    }
    return (v+i);
}

int main(int argc, char** argv) {
    double *p;
    double v[6]={-22.3, 5.4, 11.6, -2.1, -4.7, 10};
    p=funcion(v);
    printf ("el␣resultado␣del␣programa␣es␣%g\n", *p);
    return 0;
}
```

Solución: El programa calcula el menor valor de una lista de reales definida en el vector `v`. Para ello llama a `funcion`, que devuelve un puntero a ese valor.

16. Escribir una función que, dada una lista de números complejos en la forma de un array de estructuras

```
struct complexf {
        float re, im;
};
```

devuelva el módulo del número complejo resultante de sumar todos ellos.

Solución:

```
float mod_suma(int N, struct complexf x[]) {
    int i;
    float m;
    struct complexf s;
        s.re=0.0;
        s.im=0.0;
    for(i=0; i < N; i++) {
        s.re+=x[i].re;
        s.im+=x[i].im;
    }
    m=sqrt(s.re*s.re + s.im*s.im);

    return p;
}
```

17. Escribir una función que, dada una lista de números complejos en la forma de un array de estructuras

```
struct complexf {
        float re, im;
};
```

devuelva el número complejo resultante de multiplicar todos ellos.

Solución:

```
struct complexf complex_prod(int N, struct complexf x[]) {
    int i;
    float aux;
    struct complexf p;
        p.re=1.0;
        p.im=0.0;
    for(i=0; i < N; i++) {
        aux=p.re;
        p.re=p.re*x[i].re-p.im*x[i].im;
        p.im=aux*x[i].im+p.im*x[i].re;
    }
    return p;
}
```

Problemas de Física

1. Dadas las coordenadas X, Y y Z y las cargas Q de un conjunto de cargas puntuales, calcular la energía potencial electrostática del conjunto debida a la interacción entre cargas. Recuérdese que la energía potencial electrostática de un par de cargas, q_1 y q_2, a una distancia d una de otra, vale

$$U = K_e \frac{q_1 q_2}{d}$$

donde $K_e = 9 \times 10^9$ J mC^{-2}, y que la energía total es la suma de todos los pares de cargas.

Se propone, declarar:

```
double epotencial(int ncargas, double q[], double x[], double y[], double z[])
    ;
```

Solución:

```
#define Ke 9e+9
double epotencial(int ncargas, double q[], double x[], double y[],
    double z[]) {
     int n, m;
     double d, Ep=0;
     for(n=0; n<ncargas; n++) {
          for(m=0; m<n; m++) {
               d=sqrt((x[m]-x[n])*(x[m]-x[n])
                       +(y[m]-y[n])*(y[m]-y[n])
                       +(z[m]-z[n])*(z[m]-z[n]));
               Ep+=Ke*q[n]*q[m]/d;
          }
     }
     return Ep;
}
```

2. Construir un programa que escriba en un archivo los valores de la posición de una partícula que se mueve con un movimiento uniformemente acelerado en el eje X. Estos valores deberán ser calculados en n instantes del tiempo distribuidos uniformemente en un cierto intervalo $[0, T]$. Los valores de la posición inicial, la velocidad inicial, la aceleración, el número n de tiempos considerados y el intervalo de tiempo T, deberán ser introducidos por línea de comandos al ejecutar la función.

Solución: Ver código `mua.c`

3. Construir un programa que calcule el campo eléctrico creado en un punto del plano XY por un conjunto de cargas. El programa debe pedir al usuario que introduzca por línea de comandos el número de cargas que se van a considerar (hasta un número máximo). Después deberá pedir que introduzca para cada carga los datos de la misma, esto es, su posición definida por las coordenadas (x, y) y el valor de su carga q. Los datos de cada carga serán almacenados en una estructura, y las cargas en un vector de estructuras. Finalmente, deberá pedir el punto del espacio donde se quiere obtener el campo y calculará dicho campo.

Recordamos que el campo generado por una carga q en un punto del espacio tiene la forma

$$\mathbf{E} = k \frac{q}{r^3} \mathbf{r}$$

siendo $\mathbf{r}$ el vector que va de la carga al punto y r su módulo, y $k = 9 \times 10^9$ N m^2/C^2.

Solución: Ver código `campo_electrico.c`

4. El juego de la vida de Conway consiste en un tablero en cuyas casillas se hallan colocadas "células" que se comportan según las siguientes reglas:

   (a) una célula que tiene menos de dos vecinas en sus 8 celdas cercanas, desaparece (muerte por subpoblación)

   (b) una célula con más de tres vecinas en sus 8 celdas cercanas, desaparece (muerte por sobrepoblación)

   (c) una célula que tiene dos o más vecinas en sus 8 celdas cercanas, permanece como está

   (d) en una casilla vacía cuyas 8 celdas cercanas contienen en total tres células (exactamente), aparece una nueva célula (reproducción)

   Se pide escribir en C una función

   ```
   #define nF 1024
   #define nC 1024
   int conway(char A[nF][], char B[nF][]);
   ```

   que aplique estas reglas a una matriz `A[][]` (que contiene en sus celdas `0`, si no hay "célula", o `1` si hay "célula") y guarde el resultado de su evolución (un único paso) en la matriz `B[][]` [ambas matrices tienen los mismos números de filas (`nF`) y de columnas (`nC`)]. Debe tenerse en cuenta que las celdas en los bordes de la matriz (primera y última filas o columnas) no tienen vecinos. La función devolverá el número de celdas modificadas (número de células que han "muerto" o que han "nacido" en el paso).

   Recomendación: Escriba una función

   `int vecinos(char A[nF][], int i,int j);`

   que devuelva el número de "células vivas" en las celdas vecinas a la (i,j) de la matriz `A`.

   Solución:

```c
#define nF 1024
#define nC 1024

int vecinos(char A[nF][], int i, int j) {
    int n=0, ii, jj;
    for(ii=i-1; ii<=i+1; ii++)
        for(jj=j-1; jj<=j+1; jj++)
            if( 0<ii && ii<nF && 0<jj && jj<nC ) n+=A[ii][jj];
    return n-A[i][j];
}

int conway(char A[nF][], char B[nF]) {
    int nc=0, v;
    int i, j;
    for(i=0; i<nF; i++) {
        for(j=0; j<nC; j++) {
            v=vecinos(A, i,j);
            switch(v) {
                case 0:
                case 1:
                    B[i][j]=0;
                    break;
```

```
                    case 2:
                        B[i][j]=A[i][j];
                        break;
                    case 3:
                        B[i][j]=1;
                        break;
                    default:
                        B[i][j]=0;
                }
                if( B[i][j] != A[i][j] ) nc++;
            }
        }
        return nc;
    }
```

5. En su artículo de 1910, Millikan estimó la carga del electrón a partir de medidas indirectas de la carga contenida en una gota de aceite. En la siguiente tabla se encuentran numerados los valores medios de esta carga; cada número k representa lo que Millikan interpretó como "número de electrones" y las cargas q_k están divididas por 10^{-19} Coulombios.

k	4	5	6	7	8	9	10	11
q_k (10^{-19} C)	6.558	8.206	9.880	11.50	13.14	14.81	16.40	18.04
k	12	13	14	15	16	17	18	
q_k (10^{-19} C)	19.68	21.32	22.96	24.60	26.24	27.88	29.52	

Escriba un programa en C que estime la carga del electrón (e) y el error de esta estimación (Δe), relacionados por $q_k = ke + \Delta e$, a partir de estas $N = 15$ medidas, usando las siguientes ecuaciones de regresión lineal:

$$e = \frac{\sum x_k \sum x_k q_k - \sum x_k^2 \sum q_k}{\left(\sum x_k\right)^2 - N \sum x_k^2}$$

$$\Delta e = \frac{\sum x_k \sum q_k - N \sum x_k q_k}{\left(\sum x_k\right)^2 - N \sum x_k^2}$$

Solución: Ver código `Millikan.c`

6. Una partícula subatómica se mueve en línea recta y se encuentra una barrera A. En esa barrera, la partícula tiene cierta probabilidad P_a de ser absorbida, otra probabilidad de continuar su movimiento P_f y una probabilidad P_b de rebotar hacia atrás. Considere ahora N barreras de este tipo, A_1, A_2 ... A_N. Escriba una función que simule por Monte Carlo el comportamiento de una partícula lanzada inicialmente hacia A_1 y que puede ser absorbida, rebotar o traspasar esta barrera y llegar a A_2, donde puede ser absorbida, rebotar de nuevo hacia A_1 o continuar hacia A_3, etc. hasta que o bien la partícula sea absorbida en alguna barrera, o bien rebote hacia su origen inicial, o bien atraviese todas las barreras: la función devolverá, respectivamente, 0, -1 y $+1$ para indicar el resultado final de la simulación.

Nota: Para tomar la decisión en cada barrera, considere una variable aleatoria uniformemente distribuida entre 0 y 1 (calculada con cierta función `randomf()`) y compárela con las probabilidades acumuladas como

```
double eta=randomf();
if( eta < Pa )     { /* se absorbe */ }
else
if( eta < Pa+Pb ) {  /* rebota = invierte su direccion */ }
else               { /* atraviesa la barrera */ }
```

Solución:

```
int mcparticula(int N, float Pa, float Pb, float Pf) {
    int n, d=1;
    while( d!=0 && 0<=n && n<N ) {
        double eta=randomf();
        if( eta < Pa ) {
            d=0; /* es absorbida */
        } else
        if( eta < Pa+Pb ) {
            d=-d; /* rebota = invierte su direccion */
        }
        n+=d; /* siguiente barrera en la direccion del movimiento */
    }
    return d;
}
```

7. El modelo de Ising consiste en un conjunto de N partículas cuánticas ("espines") cuyo número cuántico s puede tomar sólo los valores $+\frac{1}{2}$ o $-\frac{1}{2}$. Los espines interactúan entre sí, de forma que la energía del sistema aumenta cuando un par de espines en interacción son paralelos y disminuye cuando son antiparalelos; usualmente se dice que cada par de espines s_i y s_j suma a la energía del sistema un valor $s_i s_j$. Los pares de espines en interacción pueden ser todos los pares que se pueden formar (en total, $N(N-1)/2$ pares), o sólo los pares consecutivos en la línea de espines (N pares).

   Escríbase una función

   float ising_U(**int** N, **float** s[], **int** local);

   que, al pasarle el número de espines, el array con los valores de los espines y un valor que indique si la interacción es global (0: todos los pares posibles) o local (1: sólo los vecinos próximos) calcule la energía del modelo de Ising.

   Solución:

```
float ising_U(int N, float s[], int local) {
    int i, j;
    float egy=0.0;
    if( local ) {
        for(i=1; i < N; i++)
            egy+=s[i]*s[i-1];
        egy+=s[0]*s[N-1];
    } else {
        for(i=0; i < N; i++)
            for(j=0; j < i; j++)
                egy+=s[i]*s[j];
    }
    return egy;
}
```

8. El objetivo de este ejercicio es diseñar un programa que calcule el centro de masas de un sistema discreto de partículas. Supondremos que los datos de las partículas se encuentran en un archivo con cuatro columnas: las tres primeras columnas corresponden a las coordenadas (x_i, y_i, z_i) de cada partícula i, mientras que la cuarta columna corresponde a la masa m_i. Aunque no conocemos exactamente el número de partículas que componen el sistema, sabemos que es menor que 1000. El nombre del archivo que contiene los datos deberá ser introducido por línea de comandos (como argumento de la

función `main()`) cuando se ejecute el programa. Para leer y almacenar los datos de las partículas, se deberá definir una estructura denominada "`Punto`" que representa una partícula con su posición y masa. Después, en la función `main()` de deberá declarar un array de "`Puntos`", es decir, un array de estas estructuras, para almacenar en él la información de todo el sistema de partículas. El programa debe escribir en pantalla la posición del centro de masas del sistema.

La posición del centro de masas de un conjunto de partículas con posición $\mathbf{r}_i$ y masa m_i es:

$$\mathbf{r}_{CM} = \frac{\sum_{i=1}^{n} m_i \mathbf{r}_i}{\sum_{i=1}^{n} m_i}$$

Solución: Ver código `centro_de_masas.c`

Problemas de Cálculo Numérico

1. Para calcular manualmente la raíz de un número hacemos una primera aproximación a ella, luego dividimos el número por esta estimación y calculamos la media entre la estimación y ese cociente; el resultado nos da una nueva estimación y volvemos a repetir el proceso. Escríbase un programa en C que realice estas iteraciones y calcule el error respecto al valor exacto obtenido con `sqrt()` en función del número de veces que se haya iterado. Considere números entre 1 y 10, y que la primera estimación es siempre 1; deténgase cuando el error sea inferior a 10^{-6}.

   Solución: Ver código `raiz_cuadrada.c`

2. Una manera de eliminar el ruido de una señal adquirida en función del tiempo es sustituir cada valor por la media de los valores en una ventana que comprende los n anteriores y los n posteriores a él. Escriba una función que, recibido un primer array con los datos adquiridos, un segundo array en el que se espera la respuesta (del mismo tamaño que el primero) y el número n de vecinos antes y después, calcule en el segundo array los datos promediados. [**Nota**: téngase en cuenta que los primeros datos, no tienen n anteriores, por lo que el promedio contendrá menos valores; lo mismo para los últimos de la serie]

   Solución:

```
#define LBUFFER 1024
void suaviza(float x[], float y[], int n) {
     int k, m;
     float s, w;

     for(k=0; k < LBUFFER; k++) {
          s=0;
          w=0;
          for(m=-n; m<=n; m++) {
               if( 0<=k+m && k+m<LBUFFER ) {
                    s=s+x[k+m];
                    w++;
               }
          }
          y[k]=s;
     }

     return;
}
```

3. El objetivo de este ejercicio es realizar un programa que ajuste por mínimos cuadrados un conjunto de puntos experimentales a una función conocida que depende un parámetro de ajuste.

   Supongamos que tenemos un archivo de datos con n pares de puntos (x_i, y_i) obtenidos de un experimento (x_i son enteros). Sabemos que esos puntos siguen una distribución de Poisson

   $$f(x;\lambda) = \frac{1}{x!}e^{-\lambda}\lambda^x$$

   que depende del parámetro λ y queremos obtener el valor del parámetro que mejor ajusta los puntos experimentales. Para ello se presenta a continuación un programa que lee los puntos del archivo y los almacena. Después pide al usuario que introduzca un intervalo de valores de λ: $[\lambda_1, \lambda_2]$, así como la precisión con la que debe moverse en ese intervalo $\Delta\lambda$. Comenzando desde λ_1 hasta λ_2, con incrementos

de $\Delta\lambda$, para cada valor del parámetro el programa debe calcular el sumatorio de los cuadrados de los residuos

$$R(\lambda) = \sum_{i=1}^{n} [y_i - f(x_i; \lambda)]^2$$

esto es, la suma de los cuadrados de las diferencias entre los valores obtenidos experimentalmente y los valores de la función en los puntos experimentales utilizando ese valor de λ. El valor de λ que mejor ajuste los puntos experimentales será aquel que proporcione el menor valor de R, en esto consiste el método de mínimos cuadrados. Complete el programa para que devuelva el valor de λ que mejor ajusta.

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define n 10  //numero de puntos experimentales
#define E 2.718281828

long int factorial(int x) {
    int i;
    long int f;
    f=1;
    for(i=2; i<=x; i++)
        f*=i;
    return f;
}

double poisson(int x, double lambda) {
    return pow(E,-lambda)*pow(lambda,x)/factorial(x);
}

int main(int argc, char** argv) {
    int i;
    int x[n];
    double l, lambda_1, lambda_2, inc_lambda;
    double y[n];
    FILE *fin;
    fin=fopen("datos.dat", "r");

    for (i=0; i<n; i++) {
        fscanf(fin,"%d %lf", &x[i], &y[i]);
    }

    printf("Introduzca lambda_1 lambda_2 inc_lambda:\n");
    scanf("%lf %lf %lf",&lambda_1, &lambda_2, &inc_lambda);

/*
Complete el programa
*/

    return 0;
}
```

Solución: Ver código `ajuste_no_lineal.c`

4. El objetivo de este ejercicio es escribir un programa que proporcione una estimación del número π por el método numérico de Monte Carlo. Para ello supondremos que disponemos de la función `rnd()`, declarada del siguiente modo

   ```
   double rnd();
   ```

   que devuelve cada vez que es llamada un número real (double) aleatorio uniformemente distribuido entre -1 y 1. El método consiste en generar parejas de números aleatorios que corresponderán a las coordenadas de puntos (x, y). De este modo, los puntos generados caerán en el cuadrado de lado 2 centrado en el origen de coordenadas (0, 0). La probabilidad P de que un punto caiga en el círculo de radio 1 centrado en el origen vendrá dada por el área total del círculo dividida por el área del cuadrado:

   $$P = \frac{\text{área}_{\text{círculo}}}{\text{área}_{\text{cuadrado}}} = \frac{\pi R^2}{(2R)^2} = \frac{\pi}{4}$$

   El programa debe aproximar esta probabilidad para N puntos (valor que debe ser introducido por línea de comandos al ejecutar el programa). Si definimos como n el número de esos N puntos que han caído dentro del círculo, tendremos que

   $$P \simeq \frac{n}{N}$$

   Igualando este valor con el resultado anterior tenemos que nuestra estimación de π es

   $$\pi \simeq \frac{4n}{N}$$

   Éste es el valor que debe retornar el programa para un N dado.

   Solución: Ver código `pi_Monte_Carlo.c`

5. Cómo se deberían declarar e inicializar las variables `msk`, `n`, `b`, y `x` para que el siguiente código compile sin errores y funcione correctamente.

   ```
   for(n=0; n<32; m++) {
       b= msk & (1<<n);
       if(b) {
           x=x+(double)random()/RANDOM_MAX;
       }
   }
   printf("Resultado:␣%f\n", x);
   ```

   Solución: La variable `msk` deberá ser un entero `int` o `long` (para que, al menos, tenga 32 bits). Las variables `n` y `b` pueden ser cualesquiera enteros (en particular, para `b` se podría usar un `char`). `x` debería ser un `double`, pero también podría ser un `float`.

6. Indique cuál debería ser el resultado de compilar y ejecutar este programa en C. ¿Qué debería contener el archivo "`datos.dat`"?

   ```
   #include <stdio.h>
   int main(int argc, char argv) {
       int n=0;
       double x,y, sx=0,sy=0;
       FILE *fIn=fopen("datos.dat", "r");
       while( !feof(fIn) ) {
   ```

```
        fscanf(fIn, "%f%f", &x,&y);
        sx+=x;
        sy+=y;
        n++;
    }
    fclose(fIn);
    printf("xm=%g, ym=%g\n", sx/n, sy/n);
    return 0;
}
```

Solución: El archivo datos.dat debería contener una lista de pares de números (X,Y), separados por espacios (posiblemente un par por línea). El programa lee par a par, acumula la suma de valores de las X y las Y, cuenta el número de pares leídos y, finalmente, imprime el valor medio de las X y el valor medio de la Y.

7. La integral definida de una función en un intervalo $\int_a^b f(x)\,dx$, puede aproximarse numéricamente mediante la integral —o suma— de Riemann $S(n) = \sum_{i=1}^{n} f(x_i) \cdot (x_i - x_{i-1})$, donde los puntos x_i corresponden a una partición del intervalo de integración $[a, b]$: $x_0 = a < x_1 < x_2 < \cdots < x_{n-1} < x_n = b$. Se trata por tanto de una aproximación al área encerrada debajo de la curva de la función $f(x)$.

   Si la función está acotada en el intervalo y es continua, o bien tiene un conjunto numerable de discontinuidades, se cumple que

$$\int_a^b f(x)\,dx = \lim_{n\to\infty} S(n)$$

   Escribir un programa que aproxime la integral de una función mediante la integral del Riemann. Elegir la expresión matemática $f(x)$ que se desee integrar y definirla en una función externa que será llamada desde la función `main()`. Deberemos introducir por línea de comandos el intervalo de integración $[a, b]$. El programa deberá calcular la integral de Riemann para diferentes valores de n y exportar los valores de $S(n)$ obtenidos a un archivo de modo que luego puedan ser representados con Gnuplot y así poder comprobar que estos tienden al límite dado por la integral.

   Por simplicidad, supondremos que la partición del intervalo es una potencia de 2, de modo que n variará entre 2^{n_1} y 2^{n_2}, aumentando en un factor de 2 en cada iteración. Los valores n_1 y n_2 también deberán ser introducidos por línea de comandos.

   Solución: Ver código `integral_Riemann.c`

8. Obtenga una aproximación a la integral de Riemann con una cierta precisión, indicada por línea de comandos. Para ello deberá comparar el resultado obtenido para cada n con el obtenido para el n anterior: cuando el valor absoluto de la diferencia sea menor que la precisión requerida, se entenderá que esa es la suma de Riemann que aproxima la integral definida.

   Solución: Ver código `integral_Riemann_precision.c`

9. Explicar detalladamente qué realiza el siguiente programa:

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define NMAXPTS 10000
int main(int argc, char** argv) {
```

```
    int i, j, n, npts, CONTROL;
    float num[NMAXPTS], x;
    FILE *fin, *fout;
    fin=fopen("datos.dat", "r");
    n=fscanf(fin, "%f", &x);
    if(n==1) {
        num[0]=x;
        npts=1;
    }
    do {
        n=fscanf(fin, "%f", &x);
        if(n==1) {
            CONTROL=0;
            for(i=0; i<npts; i++) {
                if(x < num[i]) {
                    for(j=npts; j>i; j--) {
                        num[j]=num[j-1];
                    }
                num[i]=x;
                CONTROL=1;
                break;
                }
            }
            if(!CONTROL) num[npts]=x;
            npts++;
        }
    } while( n==1 && npts<=NMAXPTS );

    fclose(fin);
    fout=fopen("salida.dat", "w");
    for(i=0; i<npts; i++) {
        fprintf(fout, "%g\n", num[i]);
    }
    fclose(fout);

    return 0;
}
```

Solución: El programa lee de un archivo de datos `datos.dat` una lista de números reales y los imprime de forma ordenada en otro archivo `salida.dat`.

10. En el archivo de texto plano `datos.dat` tenemos N pares de puntos reales (x_i, y_i), $i = 1, \ldots N$, que representan los valores de la función $y(x)$ en el conjunto discreto de valores de x en los que se ha dividido el intervalo $[x_1, x_N]$. Escribir un programa que obtenga la derivada de la función en cada punto y exporte las parejas de datos a un archivo `derivada.dat`.

    Para obtener la derivada de un conjunto discreto de puntos consideraremos la aproximación basada en diferencias finitas centradas en el punto:

$$y'(x_i) = \frac{1}{2}\left(\frac{y_i - y_{i-1}}{x_i - x_{i-1}} + \frac{y_{i+1} - y_i}{x_{i+1} - x_i}\right)$$

    que, como puede observarse, calcula el promedio de las pendientes entre el punto y sus dos vecinos más próximos. En los puntos extremos sólo consideraremos la pendiente con su vecino más próximo:

$$y'(x_1) = \frac{y_2 - y_1}{x_2 - x_1} \qquad y'(x_N) = \frac{y_N - y_{N-1}}{x_N - x_{N-1}}$$

Solución: Ver código `derivada_numerica.c`

11. Escribir un programa que lea desde un archivo de texto `datos.dat` una serie de valores numéricos x_i (un valor, no necesariamente entero, por cada línea del archivo) y escriba luego por la salida estándar los siguientes resultados:

    - número de valores leídos (N)
    - media de los valores leídos, $\bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i$
    - desviación típica de los valores leídos, $\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N} x_i^2 - \bar{x}^2}$

    Téngase en cuenta que el número de valores puede ser arbitrariamente grande (no se conoce a priori), por lo que no se deben guardar los x_i en un array.

    Nota: Para leer los valores desde el archivo use la función fscanf como

    ```
    int fscanf(FILE*, char*, double*)
    ```

    que recibe como argumentos el archivo desde el que leer, una cadena de formato, en este caso "%lf", y la posición de memoria de una variable de tipo double donde guardará el valor leído, y retorna 0 si no ha podido leer el valor (por haber llegado al final del archivo) o 1 si ha podido leer el valor double.

    Solución: Ver código `estadistica_datos.c`

12. El método de Newton-Raphson es un método numérico para resolver ecuaciones algebraicas no-lineales. Este método está basado en el hecho de que las soluciones de la ecuación $h(x) = g(x)$ son las raíces de la función $h(x) - g(x)$. Para obtener numéricamente estas raíces se utiliza el algoritmo iterativo de Newton-Raphson:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

    donde $f'(x_n)$ es la derivada de la función $f(x_n)$ evaluada en el punto x_n. Partiendo de una semilla inicial x_0 la sucesión convergerá (no siempre, dependerá de las características de la ecuación y de la semilla de partida) a una de las raíces de la función $f(x)$. En general el algoritmo concluye cuando el error relativo entre dos aproximaciones sucesivas es menor que una cierta tolerancia fijada, que llamaremos E, que se considera como el error relativo de la aproximación:

$$\frac{|x_{n+1} - x_n|}{|x_{n+1}|} < E$$

    El objetivo de este ejercicio es escribir un programa que calcule las soluciones reales de la ecuación $e^x = x^3$. La función $f(x)$ cuyas raíces queremos obtener, y su derivada $f'(x)$ deberán ser definidas en dos funciones. Los valores de la semilla x_0 y del error E serán introducidos por línea de comandos (como argumentos de la función `main()`) cuando se ejecute el programa. El programa deberá imprimir en pantalla el valor aproximado de la raíz obtenida con esa tolerancia.

    Solución: Ver código `Newton_Raphson.c`

Problemas de Geometría

1. Defina una estructura `Circulo` que guarde las coordenadas del centro de un disco y su radio. Después, escriba una función que reciba dos círculos y devuelva 1 si ambos se superponen o 0 si no lo hacen.

   Solución:

```
struct Circulo {
    float x,y;
    float r;
};

int circulo_super(struct Circulo c1, struct Circulo c2) {
    return (c1.x-c2.x)*(c1.x-c2.x)+(c1.y-c2.y)*(c1.y-c2.y) < (c1.r+c2.
        r)*(c1.r+c2.r);
}
```

2. Defina una estructura `Treslados` que guarde las longitudes de tres segmentos. Para que con estos tres segmentos se pueda construir un triángulo, sus longitudes deben verificar la desigualdad triangular: cada lado no puede ser mayor que la suma de los otros dos. Escriba una función que devuelva 1 si la estructura es válida para construir un triángulo o 0 si no lo es.

   Solución:

```
struct Treslados {
    float l1,l2,l3;
};

int es_triangulo(struct Treslados a) {
    return (a.l1 < a.l2+a.l3) && (a.l2 < a.l1+a.l3) && (a.l3 < a.l1+a.l2);
}
```

3. Defina una estructura `Rectangulo` que almacene las coordenadas de un rectángulo genérico, es decir, las coordenadas X,Y de dos de sus vértices opuestos: el inferior izquierdo y el superior derecho (asuma que siempre se guardan en el mismo orden). Después, escriba una función que reciba dos estructuras Rectangulo y un puntero a una tercera y que retorne 0 en caso de que no se intersequen los dos primeros, o 1 en caso de que sí lo hagan. En este último caso, la función rellenará además la tercera estructura con las coordenadas correspondientes a la intersección (que será también un rectángulo).

   Solución:

```
struct Rectangulo {
    float x1,y1;
    float x2,y2;
};

#define min(a,b) (((a)<(b))?(a):(b))
#define max(a,b) (((a)>(b))?(a):(b))

int rectangulo_interseca(struct Rectangulo r1, struct Rectangulo r2,
                         struct Rectangulo *r3) {
    if( r1.x2 < r2.x1 || r1.x1 > r2.x2
     || r1.y2 < r2.y1 || r1.y1 > r2.y2 ) {
        return 0;
```

```
    } else {
        r3->x1 = max(r2.x1, r1.x1);
        r3->y1 = max(r2.y1, r1.y1);
        r3->x2 = min(r1.x2, r2.x2);
        r3->y2 = min(r1.y2, r2.y2);
    }

    return 1;
}
```

4. Defina una estructura `Punto` que almacene las coordenadas de un punto del plano y otra estructura `Triangulo` que almacene los puntos que definen los vértices de un triángulo del plano. Después, escriba una función que reciba un `Triangulo` y un `Punto` y determine si el punto se halla dentro del área del `Triangulo` (y retorne, entonces, 1) o fuera de él (y retorne 0).

   [**Nota**: un punto P se halla en el interior de un triángulo si los productos vectoriales de los vectores que unen P con cada par de vértices consecutivos tienen el mismo signo]

   Solución:

```
struct Punto {
    float x,y;
};

struct Triangulo {
    struct Punto a, b, c;
}

float area(struct Punto o, struct Punto a, struct Punto b) {
    return (a.x-o.x)*(b.y-o.y)-(a.y-o.y)*(b.x-o.x);
}

int dentro_triangulo(struct Triangulo t, struct Punto p) {
    float a1, a2, a3;

    a1=area(p, t.a, t.b);
    a2=area(p, t.b, t.c);
    a3=area(p, t.c, t.a);

    if( (a1>0 && a2>0 && a3>0)
     || (a1<0 && a2<0 && a3<0) )
        return 1;

    return 0;
}
```

5. Se quiere buscar el mejor recubrimiento de un rectángulo de lados W y H por círculos de radio R. Para ello, se van colocando los centros de éstos en puntos del interior del rectángulo, tomados aleatoriamente (calcúlense usando la función `uniforme()` que devuelve un número pseudoaleatorio entre 0 y 1). Cada vez que se escoge un punto, se comprueba que el círculo no colisionará con los anteriores; si lo hace, se escoge otro centro y se vuelve a probar. Si después de un número K de intentos fallidos no se ha podido añadir un nuevo círculo, se considera que se ha completado el recubrimiento. Escríbase una función que haga esto:

```
int recubre(float W, float H, float R, float Xc[], float Yc[])
```

W, H: dimensiones del rectángulo y dos arrays de números flotantes

R: radio de los círculos del recubrimiento

Xc, Yc: arrays en los que se guardarán las coordenadas de los centros de los círculos del recubrimiento

La función devolverá el número de círculos que ha logrado colocar.

Solución:

```
#define K 100
int recubre(float W, float H, float R, float Xc[], float Yc[]) {
    int n, nc, ni;
    float x, y;

    nc=0;
    ni=0;
    while( ni < K ) {
        x=W*uniforme();
        y=H*uniforme();
        for(n=0; n < nc; nc++) {
            if( hypot(x-Xc[n], y-Yc[n]) < 2*R ) {
                ni++;
                break;
            }
        }
        if( ni == 0 ) {
            Xc[nc]=x;
            Yc[nc]=y;
            nc++;
            ni=0;
        }
    }

    return nc;
}
```

6. Declare una estructura que represente un poliedro: que guarde el número de vértices de éste (menor o igual que 20) y sus coordenadas XYZ. Defina una función que calcule la distancia entre dos cualesquiera de esos vértices, con la salvaguarda de que ambos vértices pertenezcan al poliedro (en caso contrario deberá devolver el valor NAN, *not a number* o indeterminado).

Solución:

```
#define NVERTICES_MAX 20
struct Poliedro {
    int nVertices;
    float x[NVERTICES_MAX],
          y[NVERTICES_MAX],
          z[NVERTICES_MAX];
};
float dVertices(struct Poliedro* p, int na, int nb) {
    float dx, dy, dz, d=NAN;
    if( 0<=na && na<p->nVertices && 0<=nb && nb<p->nVertices ) {
```

```
            dx=p->x[na]-p->x[nb];
            dy=p->y[na]-p->y[nb];
            dz=p->z[na]-p->z[nb];
            d=sqrt(dx*dx+dy*dy+dz*dz);
        }
        return d;
}
```