Problemas de Estadística y Probabilidad con C

1. Supongamos que en nuestra biblioteca de funciones tenemos definida la siguiente función denominada `muestrea_t()`, que cada vez que es llamada nos devuelve un valor aleatorio para la variable t:

```
double muestrea_t() {
    double t;
        /* la funcion obtiene un numero aleatorio t
        que esta distribuido segun una cierta funcion de probabilidad */
    return t;
}
```

   Supongamos ahora que fijamos un cierto valor T y que rellenamos una lista lineal de tamaño n con los valores aleatorios obtenidos al llamar a la función anterior, pero quedándonos sólo con aquellos valores menores que T. Es decir, si $t > T$ descartamos ese valor y generamos uno nuevo. De ese modo tenemos un conjunto de valores $\{t_i\}_{i=1}^{n}$, con $t_i < T$. Nos preguntamos ahora por la probabilidad de que la suma de los valores de esa lista sea menor que T, definida como $P(n,T)$:

$$P(n,T) \equiv \text{Prob}\left\{\sum_{i=1}^{n} t_i < T\right\}$$

   Para obtener $P(n,T)$ rellenaremos de la forma indicada anteriormente, un número `Ncadenas` de tamaño n y calcularemos qué fracción de esas cadenas tiene una suma total menor que T. Ahora podemos realizar el mismo cálculo cambiando el tamaño de n, que irá aumentando como una potencia de 2 de la forma $n(j) = 2^j$, con $j = 1, \ldots, N$

   El objetivo de este ejercicio es realizar un programa en C que reciba como argumentos el valor T y el valor máximo de n, determinado mediante N, y que escriba en un archivo de datos el valor de $P(n,T)$ para cada tamaño n de cadena, en forma de dos columnas de datos, la primera para n y la segunda para $P(n,T)$. El valor de T, N, `Ncadenas` y el nombre del archivo de datos deberán ser introducidos por línea de comandos como argumentos de la función `main`.

   Solución: Ver código `cadena_aleatoria_a.c`

2. Supongamos que en nuestra biblioteca de funciones tenemos definida la siguiente función denominada `muestrea_t()`, que cada vez que es llamada nos devuelve un valor aleatorio para la variable t:

```
double muestrea_t() {
    double t;
        /* la funcion obtiene un numero aleatorio t
        que esta distribuido segun una cierta funcion de probabilidad */
    return t;
}
```

   Fijemos ahora un cierto valor T. Supongamos que llamamos sucesivamente a la función anterior generando así una lista lineal de valores aleatorios de t, pero quedándonos sólo con aquellos valores menores que T. Es decir, si $t > T$ descartamos ese valor y generamos uno nuevo. Pararemos cuando la

suma de todos los valores de t que hemos generado supere el valor de T. La lista así obtenida tendrá una longitud n. Si ahora quisiéramos calcular el valor medio de n para ese valor de T, denominado $\bar{n}(T)$, tendríamos que repetir este proceso un cierto número de veces que llamaremos `Ncadenas`, y después calcular la media aritmética de los valores de n obtenidos. Ahora podemos repetir el mismo proceso cambiando el valor de T, que irá aumentando como una potencia de 2 de la forma $T(j) = 2^j$, con $j = 1, \ldots, N$.

El objetivo de este ejercicio es realizar un programa en C que reciba como argumento el valor máximo de T, determinado mediante N, y que escriba en un archivo de datos el valor de $\bar{n}$ para cada valor de T, en forma de dos columnas de datos, la primera para T y la segunda para $\bar{n}(T)$. El valor de N, `Ncadenas` y el nombre del archivo de datos deberán ser introducidos por línea de comandos como argumentos de la función `main`.

Solución: Ver código `cadena_aleatoria_b.c`

3. Considere una máquina constituida por dos subsistemas S_1 y S_2 que actúan en serie. Si cualquiera de dichos subsistemas se estropea durante un día determinado, la máquina se mantendrá parada hasta el día siguiente. Supongamos que durante un día en el que la máquina funciona, el subsistema S_1 tiene una probabilidad 1/2 de estropearse, mientras que el subsistema S_2 tiene probabilidad 1/4 de hacerlo. Escriba un programa que estime la probabilidad de que, durante un período de 100 días, la máquina se halle parada 5 días seguidos o más. Para ello, simule el funcionamiento de la máquina durante 1000 períodos de 100 días de funcionamiento. Para la simulación, suponemos que tenemos definida la función

   double rand_uni(){...}

   que en cada llamada retorna un número aleatorio diferente, distribuido uniformemente en el intervalo $[0, 1]$, utilizando una semilla determinada (que no hace falta explicitar, igual que no hace falta implementar la función **double** rand_uni()).

   Nota: Sea X una variable aleatoria que puede tomar el valor 0 con probabilidad p y el valor 1 con probabilidad $1-p$. Para obtener una realización de dicha variable, sea u un número aleatorio con distribución uniforme en el intervalo $[0, 1]$. Si $u \leq p$ la variable X tomará el valor 0. Si $u > p$ la variable tomará el valor 1.

   Solución: Ver código `maquina_funcionamiento.c`

4. Supongamos una moneda equilibrada (probabilidad de cara $p = 1/2$). Para simular una tirada de la moneda, supongamos que u es un número aleatorio distribuido uniformemente en el intervalo $[0, 1]$. Podemos entonces considerar que si $u \leq 1/2$ el resultado de la tirada es cara, siendo cruz el suceso complementario. Asumiendo este procedimiento, escriba un programa que simule N_i tiradas de la moneda y que calcule la probabilidad

$$p_i = \frac{\text{Número de caras}}{N_i}$$

   en función del número de tiradas N_i. Esta probabilidad será un estimador de la probabilidad de obtener cara en una tirada. Escriba el programa de tal manera que realice el procedimiento anterior para diferentes valores de N_i, empezando con $N_1 = 10$, con incrementos $N_{i+1} - N_i = 10$, hasta que $|p_i - \frac{1}{2}| < 0.001$. El programa debe de escribir todos los pares N_i, p_i en líneas separadas dentro de un fichero `caras.dat`. La primera columna del fichero contendrá los valores de N_i y la segunda los de p_i.

Nota: Para la simulación, supondremos que tenemos definida la función **double** u() que en cada llamada retorna un número aleatorio diferente con distribución uniforme en el intervalo $[0, 1]$ (no hace falta implementar la función).

Solución:

```
#include<stdlib.h>
#include<time.h>
#include<stdio.h>

int main(int argc, char **argv) {
    double u;
    double tol=0.001;
    srand(time(NULL));
    FILE *fout;
    fout=fopen("datos.dat", "w");
    int N=0;
    int Nc;
    double p=0;
    while(abs(p-0.5)>tol){
        N+=10;
        Nc=0;
        for (int i=0; i<=N; i++) {
            u=(double)rand()/(RAND_MAX+1.00);
            if(u<=0.5){Nc++;}
        }
        p=(Nc+0.0)/(N+0.0);
        fprintf(fout, "%i\t%0.4f\n", N,p);
    }
    fclose(fout);
    return 0;
}
```

5. 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`

6. Sabemos que una variable aleatoria η normalmente distribuida (según la distribución $N(0;1)$, de media 0 y desviación típica 1) se puede simular como

$$\eta = \frac{12}{N}\sum_{i=1}^{N}\left(\mu_i - \frac{1}{2}\right)$$

con N un número grande (p.ej. 15 o 30) y μ_i valores calculados de una variable aleatoria uniformemente distribuida en el intervalo [0, 1). De la misma forma se puede simular una variable aleatoria χ distribuida según la distribución chi-cuadrado con n grados de libertad como

$$\chi = \sum_{j=1}^{n}\eta_j^2$$

con η_j valores calculados de una variable aleatoria distribuida gaussianamente.

Asumiendo que ya disponemos de una función **double** uniforme() que genera un número aleatorio distribuido uniformemente en el intervalo [0, 1), escriba una función **double** chi2(**int** n) que genere un número aleatorio distribuido según la distribución chi-cuadrado con n grados de libertad.

Solución:

```
#define Ngauss 20

double chi2(int n) {
    int i, nn;
    double g, x;
    x=0;
    for(nn=0; nn<n; nn++) {
        /* gaussiano */
        g=0;
        for(i=0; i<Ngauss; i++) {
            g+=uniforme() -0.5;
        }
        g=(12.0/Ngauss)*g;
        x+=g*g;
    }
    return x;
}
```

7. 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:

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

int main(int argc, char** argv) {
    char *fn="datos.dat";
    FILE *f;
    int n;
```

```
    double xi, sxi, sxi2, xm, xs;
    n=0;
    sxi=0.0;
    sxi2=0.0;
    f=fopen(fn, "r");
    while( fscanf(f, "%lf", &xi) ) {
        n++;
        sxi+=xi;
        sxi2+=xi*xi;
    }
    fclose(f);
    xm=sxi/(double)n;
    xs=sqrt( sxi2/(double)n - xm*xm );
    printf("N=%d\n", n);
    printf("xm=%g\n", xm);
    printf("xs=%g\n", xs);
    return 0;
}
```