EJERCICIO PROPUESTO EN R: INTEGRACIÓN Y MÉTODO MONTECARLO

En el siguiente ejercicio se va a utilizar el método de Montecarlo. Empecemos por describir en qué consiste estos métodos:

- Mediante los métodos de Montecarlo se pueden dar soluciones a problemas matemáticos y físicos de cierta envergadura y complejidad mediante la generación de números aleatorios. Este método es muy potente para ordenadores y sistemas informáticos, ya que pueden generar infinidad de valores aleatorios en poco tiempo, por lo que es un método muy potente.[1]

Casino de Montecarlo (Mónaco). Fue precisamente este concepto ligado al método del azar (como las que se encuentran en las ruletas de los casinos) lo que llevó a denominar así al método. Fuente: Wikimedia Commons.

En nuestro caso, vamos a utilizar el método de Montecarlo para encontrar el valor aproximado de una integral definida (*el área bajo una curva*), pero también puede utilizarse este método para encontrar el número **π.**

Como se ha mencionado previamente, lo que se busca es generar puntos aleatorios para posteriormente calcular la relación entre puntos que se encuentran dentro del área que define la curva y la cantidad total de puntos.

Vamos a seguir los pasos que se nos indican:

- En primer lugar, es necesario programar d (la expresión de la densidad, que es un dato del problema) como una función. Según los datos, *d(x)=exp(x)sin(x).* Por lo que se ve, el argumento que vamos a utilizar es x, y en el cuerpo de operaciones pondremos la expresión que nos viene dada:

```
d=function(x){exp(x)*sin(x)}
```

Figura 1. La función de densidad se programa como una función. Entre paréntesis se incluyen los argumentos, y entre llaves el cuerpo de operaciones. Es importante no olvidarse abrir y cerrar las llaves.

- Asimismo, se insta a introducir una variable llamada *num_puntos* donde escribamos el número de puntos a considerar para poner en práctica el método. Comenzamos escribiendo 1000, aunque puede incrementarse

[1] Si quieres una explicación breve e ilustrativa, puedes ver el vídeo de YouTube ¿En qué consiste el método de Montecarlo? de *Date un Voltio*.

posteriormente. Se supone que cuanto mayor sea el número de puntos, más afinado será el resultado, aunque se puede comprobar que esto no es siempre así.

- A continuación, se pide que se definan los extremos del dominio de cálculo, por lo que hacemos lo propio, según las indicaciones del enunciado.

```
num_puntos=1000
AA=0
BB=7.4
A=0
B=3
```

Figura 2. Escribimos el número de puntos y lo almacenamos en la variable num_puntos. A continuación, acotamos los extremos del dominio de cálculo.

- En tercer lugar se nos pide *generar los vectores ss, ff que contengan num_puntos aleatorios en el intervalo [A,B] y [AA,BB] respectivamente*. Es decir, tenemos que crear vectores con los 1000 puntos -en nuestro caso- aleatorios en los intervalos que se han definido previamente. Hay dos formas de obtener valores aleatorios en R. El primero, *sample()* no nos interesa, ya que con esta función solamente se obtienen números enteros. Para obtener números racionales aleatorios, que se ajusta más a lo que buscamos, utilizamos la función *runif()*.
  La función *runif()* tiene tres atributos. Con el primero, indicamos el número de valores aleatorios que queremos que nos genere. Nosotros seleccionamos *num_puntos*, es decir, 1000. Los otros dos atributos representan los intervalos en los que van a estar comprendidos estos valores. En el caso del vector *ss*, esos valores aleatorios se encontrarán entre el A y B, como extremos del intervalo. En el caso del vector *ff*, los valores aleatorios se van a extraer entre AA y BB. Procedemos a escribir esto en nuestro script de R:
  ¡Atención!: Es importante que el orden de estos atributos mencionados se mantenga.

```
ss=runif(num_puntos,A,B)
ff=runif(num_puntos,AA,BB)
```

Figura 3: Código que hemos de escribir en el script para obtener los num_puntos aleatorios en los intervalos señalados en los enunciados.

Puede comprobarse que al llamar el vector *ss* en la consola se obtienen los 1000 puntos generados aleatoriamente entre A y B (es decir, entre 0 y 3):

```
[1301] 1.300 1.301 1.302 1.303 1.304 1.305 1.306 1.307 1.308 1.309 1.310 1.311 1.312 1.313 1.314 1.315
[1327] 1.326 1.327 1.328 1.329 1.330 1.331 1.332 1.333 1.334 1.335 1.336 1.337 1.338 1.339 1.340 1.341
[1353] 1.352 1.353 1.354 1.355 1.356 1.357 1.358 1.359 1.360 1.361 1.362 1.363 1.364 1.365 1.366 1.367
[1379] 1.378 1.379 1.380 1.381 1.382 1.383 1.384 1.385 1.386 1.387 1.388 1.389 1.390 1.391 1.392 1.393
[1405] 1.404 1.405 1.406 1.407 1.408 1.409 1.410 1.411 1.412 1.413 1.414 1.415 1.416 1.417 1.418 1.419
[1431] 1.430 1.431 1.432 1.433 1.434 1.435 1.436 1.437 1.438 1.439 1.440 1.441 1.442 1.443 1.444 1.445
[1457] 1.456 1.457 1.458 1.459 1.460 1.461 1.462 1.463 1.464 1.465 1.466 1.467 1.468 1.469 1.470 1.471
[1483] 1.482 1.483 1.484 1.485 1.486 1.487 1.488 1.489 1.490 1.491 1.492 1.493 1.494 1.495 1.496 1.497
[1509] 1.508 1.509 1.510 1.511 1.512 1.513 1.514 1.515 1.516 1.517 1.518 1.519 1.520 1.521 1.522 1.523
[1535] 1.534 1.535 1.536 1.537 1.538 1.539 1.540 1.541 1.542 1.543 1.544 1.545 1.546 1.547 1.548 1.549
[1561] 1.560 1.561 1.562 1.563 1.564 1.565 1.566 1.567 1.568 1.569 1.570 1.571 1.572 1.573 1.574 1.575
[1587] 1.586 1.587 1.588 1.589 1.590 1.591 1.592 1.593 1.594 1.595 1.596 1.597 1.598 1.599 1.600 1.601
[1613] 1.612 1.613 1.614 1.615 1.616 1.617 1.618 1.619 1.620 1.621 1.622 1.623 1.624 1.625 1.626 1.627
[1639] 1.638 1.639 1.640 1.641 1.642 1.643 1.644 1.645 1.646 1.647 1.648 1.649 1.650 1.651 1.652 1.653
[1665] 1.664 1.665 1.666 1.667 1.668 1.669 1.670 1.671 1.672 1.673 1.674 1.675 1.676 1.677 1.678 1.679
[1691] 1.690 1.691 1.692 1.693 1.694 1.695 1.696 1.697 1.698 1.699 1.700 1.701 1.702 1.703 1.704 1.705
[1717] 1.716 1.717 1.718 1.719 1.720 1.721 1.722 1.723 1.724 1.725 1.726 1.727 1.728 1.729 1.730 1.731
[1743] 1.742 1.743 1.744 1.745 1.746 1.747 1.748 1.749 1.750 1.751 1.752 1.753 1.754 1.755 1.756 1.757
[1769] 1.768 1.769 1.770 1.771 1.772 1.773 1.774 1.775 1.776 1.777 1.778 1.779 1.780 1.781 1.782 1.783
[1795] 1.794 1.795 1.796 1.797 1.798 1.799 1.800 1.801 1.802 1.803 1.804 1.805 1.806 1.807 1.808 1.809
[1821] 1.820 1.821 1.822 1.823 1.824 1.825 1.826 1.827 1.828 1.829 1.830 1.831 1.832 1.833 1.834 1.835
[1847] 1.846 1.847 1.848 1.849 1.850 1.851 1.852 1.853 1.854 1.855 1.856 1.857 1.858 1.859 1.860 1.861
[1873] 1.872 1.873 1.874 1.875 1.876 1.877 1.878 1.879 1.880 1.881 1.882 1.883 1.884 1.885 1.886 1.887
[1899] 1.898 1.899 1.900 1.901 1.902 1.903 1.904 1.905 1.906 1.907 1.908 1.909 1.910 1.911 1.912 1.913
[1925] 1.924 1.925 1.926 1.927 1.928 1.929 1.930 1.931 1.932 1.933 1.934 1.935 1.936 1.937 1.938 1.939
[1951] 1.950 1.951 1.952 1.953 1.954 1.955 1.956 1.957 1.958 1.959 1.960 1.961 1.962 1.963 1.964 1.965
[1977] 1.976 1.977 1.978 1.979 1.980 1.981 1.982 1.983 1.984 1.985 1.986 1.987 1.988 1.989 1.990 1.991
[2003] 2.002 2.003 2.004 2.005 2.006 2.007 2.008 2.009 2.010 2.011 2.012 2.013 2.014 2.015 2.016 2.017
[2029] 2.028 2.029 2.030 2.031 2.032 2.033 2.034 2.035 2.036 2.037 2.038 2.039 2.040 2.041 2.042 2.043
[2055] 2.054 2.055 2.056 2.057 2.058 2.059 2.060 2.061 2.062 2.063 2.064 2.065 2.066 2.067 2.068 2.069
```

¡La lista es interminable!

- Como se había mencionado inicialmente, lo que vamos a buscar es la relación entre el número de puntos por debajo de la curva y el número de puntos totales, para calcular el valor de la integral definida. Para ello, vamos a abrir un bucle en el que iremos discriminando entre los puntos que se encuentran debajo la curva y los que se encuentran fuera de ella, mediante una condición. Pero, antes de nada, inicialicemos los vectores a 0.

```
puntos_dentro=0
puntos_fuera=0
color=0
```

Figura 5: Inicializamos los vectores puntos_dentro, puntos_fuera y color a 0.

- A continuación, abrimos el bucle, donde *i* naturalmente se desplazará desde 1 hasta *num_puntos*, que en nuestro caso habíamos definido inicialmente como 1000, aunque podría ser cualquier otra cifra, como se ha mencionado antes. Para poder discernir entre los puntos dentro de la gráfica y fuera de ella, abrimos una condición. A cada punto se les da un color u otro, en función de si se encuentra dentro o fuera. Para ello, el vector color contendrá el color de cada punto *i*. Esto será utilizado más adelante en la representación gráfica.

```
for(i in 1:num_puntos){
      if(ff[i]<=d(ss[i])){
            puntos_dentro=puntos_dentro+1
            color[i]='blue'
      }else{
            puntos_fuera=puntos_fuera+1
            color[i]='orange'
      }
}
```

Figura 6. Abrimos un bucle y ponemos la condición: si los puntos generados se encuentran por debajo de la función de densidad, entonces, se contabiliza como puntos_dentro y se les da el color azul. En caso contrario, se suma dentro de los puntos_fuera y se les colorea de naranja.

De esta manera, aquellos puntos que queden por debajo de la curva quedarán colorados de azul, mientras que aquellos que hayan quedado por encima se colorearán en naranja. Esto cobrará importancia a continuación, al dibujarlo.

Para dibujarlo, utilizamos el código que se proporciona en el enunciado:

```
plot(ss,ff,xlim=c(A,B),ylim=c(AA,BB),col=color,ylab='Densidad',xlab='x')
par(new='TRUE')
ss=seq(A,B,0.001)
plot(ss,d(ss[1:length(ss)]),type='l',col='dark green',xlim=c(A,B),ylim=c(AA,BB),xlab='',ylab='')
```

Figura 7.

Recordemos que la función *plot ()* se compone de varios argumentos, que son opcionales excepto el argumento x. En el caso de la primera gráfica, *ss* es nuestro argumento x, mientras que nuestro argumento y es *ff*. Con xlim e ylim limitados los ejes. El color elegido hace referencia al vector color, que dependerá de si el punto se encuentra por encima o por debajo de la gráfica. Además, ponemos etiquetas a los ejes para una mejor lectura del gráfico.

Recordemos que para unir varias gráficas que queremos solapar en una sola, utilizamos *par (new='TRUE')*. El segundo plot nos dibuja la curva.

Al ejecutar esto, obtenemos la siguiente gráfica:

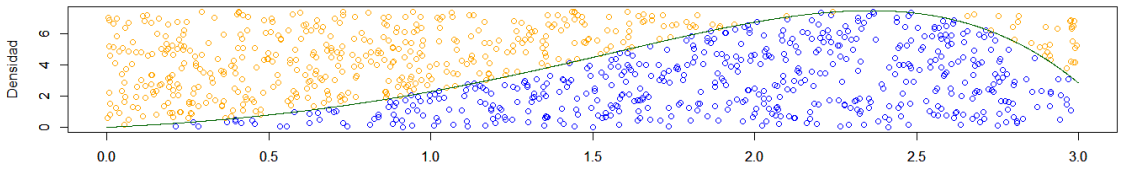

Figura 8. Obsérvese la gráfica: se han generado 1000 puntos aleatorios que nos permiten calcular, mediante una relación entre los puntos dentro de la curva y los puntos totales, el valor de la integral definida.

- Para obtener el valor de la integral -o mejor dicho, una estimación-, hemos de hacer esa relación que se ha mencionado. Para ello, escribimos en el script *Area=puntos_dentro/num_puntos*B*BB*.

Como se ha recalcado, el valor obtenido es una aproximación. Para calcular una integral, quizás se pueda pensar que este método es largo y tedioso, teniendo otras maneras más sencillas de obtener su valor, como una calculadora. Eso es cierto, pero con esto se pretende enseñar de la potencialidad de este método de Montecarlo, que para problema de una mayor envergadura resulta muy interesante.

- Procedemos a calcular el error. Para ello, se nos indica cuál es el valor exacto de la integral, que lo almacenamos en una variable *Vexact=11.97907159*.
- Para calcular el error, escribimos *ErrorM=(Area-Vexact)/Vexact*. De esta forma, como el denominador es constante (es el valor que se ha dado previamente), cuanto menor sea el denominador, es decir, cuánto menor sea la diferencia entre el área calculada bajo la curva mediante el método de Montecarlo y el valor exacto, menor será el error. Vemos que el error que obtenemos es bastante pequeño: -0.06411779. Se invita a que sea el usuario el que juegue con el número de puntos generados aleatoriamente para comprobar cuánto varía el error con más o menos puntos.