Laboratorio de Probabilidad I
Segundo de Matemáticas
Curso 2002-2003

Práctica 2

El objetivo de esta práctica es el de construir una serie de herramientas que nos sean útiles a la hora de estudiar variables aleatorias.

1. Empezaremos diseñando una función `momentos` que lea un vector de valores $(x_1, \ldots, x_n)$ y un vector de probabilidades $(p_1, \ldots, p_n)$ (donde $p_j = \mathbf{P}(X = x_j)$), y nos devuelva la esperanza y la varianza de X. Quizás convenga incluir en el código alguna comprobación de que los dos vectores tienen la misma dimensión, e incluso de que $(p_1, \ldots, p_n)$ es una verdadera función de masa.

Comprobamos el funcionamiento de esta función para casos sencillos (por ejemplo, una variable $Ber(p)$, o una variable que tome tres o cuatro valores, con sus respectivas probabilidades), cuyos datos podemos introducir por el teclado.

2. Para verla en funcionamiento con distribuciones de probabilidad como la binomial, la Poisson, etc., construimos las funciones adecuadas. Por ejemplo,

- `binomial` toma como datos de entrada n y p y devuelve un vector con los valores $(0, 1, \ldots, n)$ y las probabilidades $(p_0, p_1, \ldots, p_n)$ definidas por

$$p_j = \mathbf{P}(X = j) = \binom{n}{j} p^j (1-p)^{n-j} .$$

  Habrá que tener cuidado con la manera en que Matlab numera los elementos de un vector. También observaremos que Matlab tiene problemas (y nos avisará de ello) para calcular $\binom{n}{k}$ cuando n es grande. ¿Hay alguna manera de evitar que aparezcan estos mensajes de aviso?

- `poisson` debe tomar un dato de entrada λ y devolver los valores $0, 1, 2, 3, \ldots$ Cuidado, es obvio que tenemos que cortar esta serie de valores. Quizás convenga establecer *a priori* un umbral: los números j para los que $\mathbf{P}(X = j)$ sea menor que ese umbral quedan descartados. Lo meditamos y, una vez decidido dónde cortamos la serie de valores, calculamos las probabilidades:

$$\mathbf{P}(X = j) = \frac{\lambda^j}{j!} e^{-\lambda} .$$

Lo mismo podríamos hacer con la geométrica, la binomial negativa, etc. De todas ellas calculamos la esperanza y la varianza (y comparamos con los valores que hemos calculado en clase).

3. Ya nos hemos aburrido de hacer este tipo de cuentas: ahora queremos generar números acordes a una cierta distribución de probabilidad. No ponemos en el caso más general posible: la función `generador` debería tomar un vector de valores $(x_1, \ldots, x_n)$ y uno de probabilidades $(p_1, \ldots, p_n)$, junto con un número N (la cantidad de números que queremos generar). Y la salida debería ser una lista de N números extraídos según la distribución de probabilidad.

No queremos usar las herramientas que tiene Matlab, construiremos la función utilizando únicamente la instrucción `rand`. Evitaremos, en la medida de lo posible, los bucles (salvo que no se nos ocurra otra cosa). Las instrucciones `find`(*condición*) y `cumsum` nos pueden ser útiles.

Quizás, en lugar de que la salida sea la lista de números, nos puede interesar dibujar el histograma. Investigaremos las posibilidades de la instrucción `hist`; quizás convenga pensar en construir una herramienta que dibuje histogramas, para tener más control sobre el aspecto de las gráficas que obtenemos.

4. Con todo lo hecho hasta aquí, no nos será difícil modificar el código de las funciones ya elaboradas para diseñar, por ejemplo, un generador de números extraídos de una $Bin(n, p)$, o de una $Poiss(\lambda)$, o quizás...

5. ¿Y por qué no utilizar estos generadores de números aleatorios para comprobar, experimentalmente, la desigualdad de Chebyshev? Recordemos que esta desigualdad afirmaba que si X es una variable aleatoria (con esperanza $\mathbf{E}(X)$ y desviación típica $\sigma(X)$) y ρ un cierto número positivo, entonces

$$\mathbf{P}(|X - \mathbf{E}(X)| \geq \rho\sigma(X)) \leq \frac{1}{\rho^2}\,.$$

¿Se te ocurre cómo hacerlo?