Instrumentación Nuclear
SIMULACIÓN III

Simulación del decaimiento nuclear usando la distribución de Poisson

El tema es: simulación del decaimiento nuclear usando la *distribución de probabilidad de Poisson*.

El problema específico: Simule la evolución de la actividad en el tiempo.

Ojo: su atención va a estar puesta en la actividad $A(t)$, para lo cual, por supuesto, deberá simular $N(t)$.

¿Por qué Poisson? El enunciado "estándar" de la distribución de Poisson afirma que es la *probabilidad de que de N intentos, haya x éxitos cuando p es la probabilidad de éxito en un sólo intento*. En la simulación presente la palabra éxito será asociada a que tenga lugar un decaimiento (o no, pues cero decaimientos también es válido). Por lo tanto la varible x denota el número de decaimientos.

¿En dónde está la variable 'tiempo'? En cada instante t la muestra *tiene que decidir* cuántos núcleos decaerán durante el siguiente intervalo de tiempo Δt. Una vez hayan decaido, la muestra tiene un número diferente de núcleos, del cual, en el siguiente intervalo de tiempo la muestra debe decidir de cuántos deshacerse, es decir, cuántos decaerán. Y este proceso se va repitiendo en cada unidad de tiempo subsiguiente hasta que hayan decaido todos los núcleos iniciales.

Entonces la estrategia para el algoritmo es la siguiente:

1. Para cada instante
$$t_i = i \cdot \Delta t \qquad i = 0, 1, 2, ...$$

   a) Una subrutina calcula la distribución de Poisson con $N(t_i)$ intentos.
$$N(t_i) = \text{ número de núcleos en la muestra}$$

   b) La misma, u otra subrutina calcula la función acumulativa $F(x)$ con
$$0 \leq x \leq N(t_i)$$

   c) Un número aleatorio decide el número de núcleos que decaen en el lapso de tiempo $(t_i, t_i + \Delta t)$. Llamemos X a ese número. Observe que la actividad en ese intervalo de tiempo será
$$A(t_i) = \frac{\mathsf{X}}{\Delta t}$$

2. El número de núcleos que no han decaido al comienzo del siguiente intervalo de tiempo
$$t_{i+1} = (i+1) \cdot \Delta t$$
   será
$$N(t_{i+1}) = N(t_i) - \mathsf{X}$$
   Con este nuevo número, el programa va al punto 1*a*) y continúa.

3. El programa termina, obviamente, cuando $N(t_i) = 0$.

Observación práctica: Todo el meollo del asunto está en escribir una subrutina que calcule tanto la densidad de probabilidad de Poisson como su función acumulativa con $N(t_i)$ como uno de sus "datos de entrada".

La tarea:

T1. Escriba el programa que pone en práctica la anterior estrategia.

T2. Elija Δt acorde con el tiempo de vida del núcleo elegido.

T3. ¿Cuál es el valor de p y de μ que va a usar? Explique y muestre el cálculo.

T4. Para $N(0) = 100$: Para el tiempo inicial y para un tiempo intermedio grafique:

1. La probabilidad de que un número x de núcleos decaigan.
2. La función acumulativa.

T5. Grafique $A(t_i)$ y compárela con la expresión $A(0)e^{-t/\tau}$ para $N(t = 0) = 10^6, 10^3, 100,\ 10$.

T6. En la misma gráfica compare los resultados de la simulación usando Poisson con el resultado que obtiene usando la distribución **exponencial**.

T7. En las mismas gráficas resultantes del punto T5 compare los resultados de la simulación usando Poisson con el resultado usando la distribución **binomial**.

Observación práctica: En estos casos es mucho más fácil examinar la calidad del ajuste cuando se observa la comparación en escala **semi-logarítmica**.

Ayuda

Si usa `python`:

1. `scipy.stats.poisson.pmf(x,mu)` calcula la distribución de poisson para cualquier valor de μ, aún para valores muy grandes! Ojo, es 'pmf', *probability mass function*, no pdf, (*probability density function*) porque se trata de una distribución discreta.
2. `scipy.stats.poisson.rvs(mu,size=N)` le da `N` variables aleatorias sacadas de la distribución de Poisson con $\mu =$ `mu`. `N`$= 1$ puede ser usado, por supuesto. Mnemotécnica: rvs = **r**andom **v**ariate**s**.
3. `scipy.stats.poisson.cdf(x,mu)` *cumulative density function*.
4. `scipy.stats.poisson.ppf(q,mu)` "Inversa" de la cdf. Es decir, si le da un valor aleatorio q=r, este método le devuelve la inversa, o sea a qué variable aleatoria x corresponde.
5. `numpy.random.uniform(low=0.0,high=1.0,size=N)` devuelve `N` variables aleatorias según la distribución uniforme en el rango $[0, 1)$, es decir, incluye el cero pero no el uno.

La existencia de estas rutinas ya escritas y comprobadas hace la tarea muchísimo más fácil. Hay por lo menos dos maneras de resolver los puntos 1*a*), 1*b*) y 1*c*:

El camino largo:

1. `numpy.random.uniform(low=0.0,high=1.0,size=1)` da un número aleatorio, r.
2. `scipy.stats.poisson.ppf(q=r,mu)` proporciona una variable aleatoria perteneciente a una distribución de Poisson. Este proceso está resumido en la Fig. 1

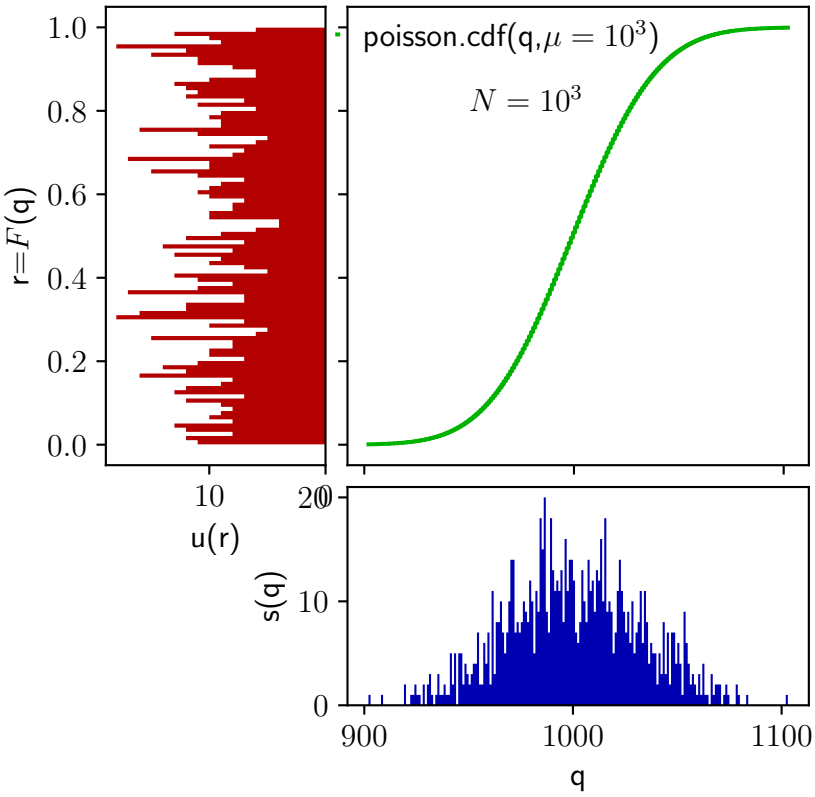

Figura 1: `r` viene de una distribución uniforme. `poissson.ppf(r,mu)` produce una variable aleatoria `q`.

El camino corto: Usar `poisson.rvs(mu,size=1)` cada vez que necesite una variable aleatoria perteneciente a una distribución de Poisson.

Si usa este camino, debe demostrar(se a usted mismo) que de verdad el uso de `N` veces `poisson.rvs(mu,size=1)` produce una muestra de tamaño `N` descrita por `poisson.pmf(x,mu)`. La Fig. 2 sugiere cómo hacerlo.

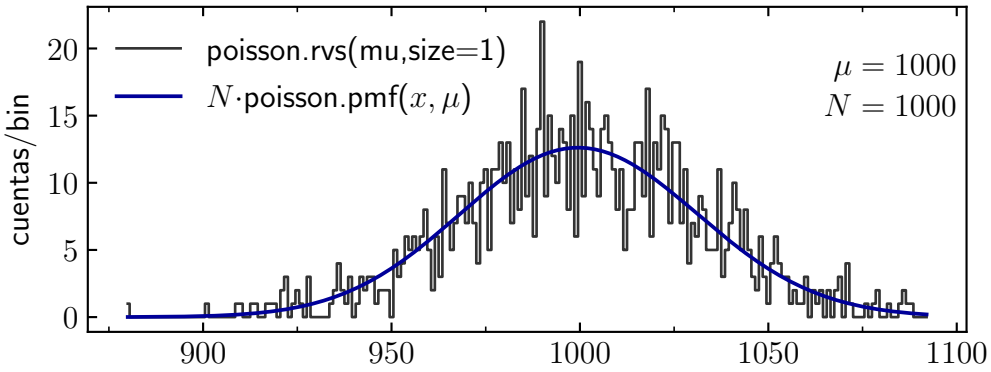

Figura 2: El histograma es el resultado de "lanzar 1000 veces el dado" sobre una distribución de Poisson, es decir, correr 1000 veces `poisson.rvs(mu,size=1)`.