Física Numérica

Tarea #7

D. A. Vázquez Gutiérrez

Escuela Superior de Física y Matemáticas, Instituto Politécnico Nacional,Unidad Profesional .ªdolfo López Mateos", Zacatenco, Edificio 9, Col. San Pedro Zacatenco, C.P. 07730 del. Gustavo A. Madero, Ciudad de México, México

email: dvazquezg1600@alumno.ipn.mx

20 de junio de 2024

1. Conjunto de números aleatorios.

Existen diversos métodos para estudiar si un conjunto de números tiene o no una distribución uniforme. Investigue una de ellas, explíquela y aplique dicha prueba a los números generados con la función `random` de Python.

Prueba Kolmogorov-Smirnov.

Es una técnica estadística no paramétrica que se utiliza para determinar si una muestra de datos sigue una distribución específica. Puede aplicarse a una amplia variedad de distribuciones teóricas, como la distribución normal, uniforme, exponencial, etc; Esto debido a que no es inusual el considerar cantidades aleatorias sobre un rango de infinitos valores , tal como las fracciones aleatorias. Queremos esencialmente que nuestros valores aleatorios se comporten como si todos los reales entre $[0, 1)$ fueran igualmente probables.

Primero que nada, hay que recordar lo que es la *Función de Distribución acumulada* $F(X)$, donde:

$$F(x) = Pr(X \leq x) \tag{1}$$

Esto no es mas que la probabilidad de que $X \leq x$. Notamos que entonces $F(x)$ siempre se incrementa de 0 a 1 en función de que x aumente de $-\infty$ a ∞.

Ademas , tenemos que ver lo que es la *Distribución Empírica* . Si tenemos una muestra de datos X_i con $i\epsilon[1, n] \cap N_0$ ordenada en forma ascendente ,entonces, la distribución empírica (FDE), denotada como $F_n(x)$, se define como :

$$F_n(x) = \frac{1}{n} \sum_{i=1}^{n} I(X_i \leq x) \tag{2}$$

Con n el tamaño de la muestra e $I(X_i \leq x)$ una función indicadora que es igual a 1 si $X_i \leq x$ y 0 en caso contrario.

Podemos entonces superponer en un grafico la función de distribución $F(x)$ que consideremos que ajusta con los datos.

La prueba de Kolmogorov-Smirnov (prueba KS), puede ser usada en $F(x)$ donde no haya saltos, es decir , que sean continuas; esta basada en las diferencias entre $F(x)$ y $F_n(x)$, grandes desviaciones entre estas dos son extremadamente improbables, y la prueba KS es usada para indicarnos que tan improbable es.

Para esto usaremos las siguientes dos estadísticas :

$$K_n^+ = \sqrt{n} \sub{-\infty<x<\infty} (F_n(x) - F(x))$$
$$K_n^- = \sqrt{n} \sub{-\infty<x<\infty} (F(x) - F_n(x)) \quad (3)$$

Entonces K_n^+ mide la mayor desviación cuando F_n es mayor que F , mientras que K_n^- mide la máxima desviación cuando F_n es menor que F; igualmente el factor $\sqrt{n}$ magnifica las estadísticas anteriores , ya que es posible demostrar que la desviación estándar de F_n es proporcional justo a $1/\sqrt{n}$.

Claramente, observando los valores de K_n^+ y K_n^- podemos determinar si la hipótesis de que nuestros datos tienen una distribución correspondiente a F es correcta. Entonces podemos utilizar el siguiente criterio , tomando la estadistica de Kolmogorov-Smirnov como :

$$D_n = \sup[K_n^+, K_n^-] \quad (4)$$

Ademas , existe un numero α , llamado *nivel de significancia*, sus valores comunes son 0.05, 0,01 y 0.1.Por ejemplo, si α = 0,05 significa que hay u 5 % de probabilidad de rechazar incorrectamente la hipótesis de que F es la distribución de los datos.

Habiendo elegido el nivel de significancia, en función de este y del numero de datos podemos obtener un valor critico D_α , el cual es:

$$D_\alpha = \sqrt{-\frac{1}{2n} \mathrm{In}\Big(\frac{\alpha}{2}\Big)} \quad (5)$$

Sabremos entonces que nuestra hipótesis es valida si :

$$D_n \leq D_\alpha \quad (6)$$

Ahora , una forma fácil de pasar este método computacionalmente es, después de haber obtenido nuestras medidas , hay que ordenarlas de menor a mayor , de tal forma que sean:

$$X_1 \leq X_2 \leq X_3 \leq ... \leq X_n$$

De esta forma podemos usar una ecuación análoga a ecuación 3 , siendo esta :

$$K_n^+ = \sqrt{n} \sub{0<j<n} \Big(\frac{j}{n} - F(X_j)\Big)$$
$$K_n^- = \sqrt{n} \sub{0<j<n} \Big(F(X_j) - \frac{j}{n}\Big) \quad (7)$$

Ahora, un ultimo punto importante es la cantidad de datos n de la muestra. Queremos que n sea comparativamente grande, de tal forma que podamos rechazar la hipótesis si es el caso , entonces necesitamos n lo suficientemente grande para que la distribucion empírica pueda ser lo suficientemente diferente de F si es el caso, podríamos detectar un comportamiento de no aleatoriedad global

Por otro lado , los valores grandes de n tienden a promediar la no aleatoriedad local, y esto es indeseable computacionalmente. Una solución a esto es la *Segmentación de datos*, dividiendo nuestra muestra en otras mas pequeñas, digamos que la dividimos en r segmentos, entonces, conseguiríamos una muestra de r valores de $D_{n/r}$, a los cuales también se les puede hacer la prueba KS para medir la calidad de los datos.

Con esto en mente ,el autor creo el código 1 que se encuentra en el apéndice A.1 con la cual conseguimos la grafica de la figura **1**.

En la simulación hecha se utilizo un valor de α de 0.1. Conseguimos por una parte la prueba KS global , correspondiente con la grafica de la izquierda, y una prueba KS con el ajuste para detectar irregularidades locales, la grafica de la derecha, teniendo una muestra relativamente grande de 200 muestras experimentales, entonces , conseguimos los datos del cuadro 1.

Con esto vemos que globalmente , la función `Random` es aceptable como distribucion aleatoria uniforme , sin embargo , existe la posibilidad de que localmente , esta no sea tan confiable , dada las grandes fluctuaciones que tiene la prueba KS con segmentación y por lo tanto , con ajuste local.

Un punto interesante al evaluar por segunda vez el test KS en el ajuste local , fue que , por examinacion visual , decidimos tomar como función F de esta segunda KS , a la propia distribucion uniforme, dándonos buenos resultados , siendo esta la

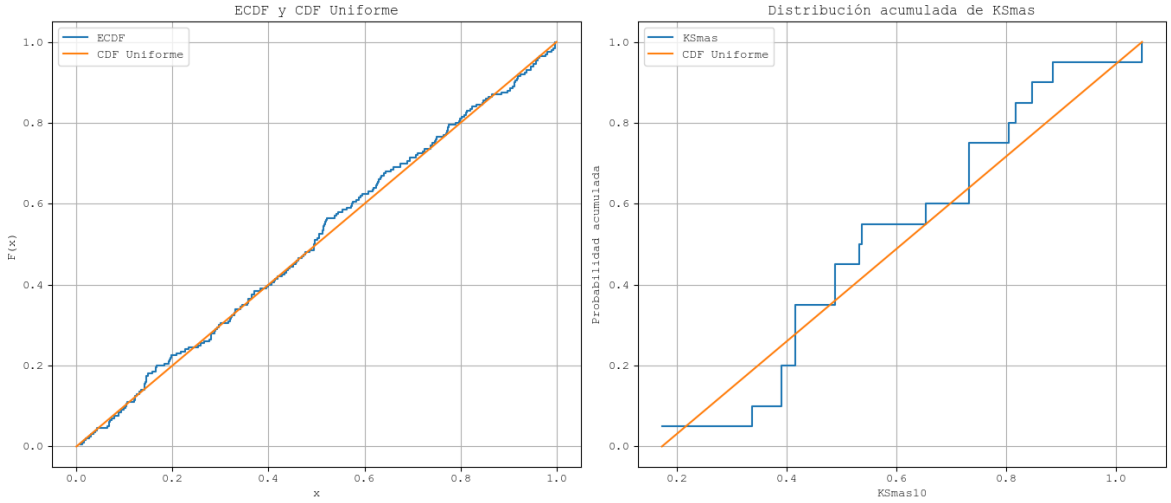

Figura 1: **Distribuciones acumuladas junto con distribuciones empíricas**. Prueba KS. Ajuste global (izquierda), asi como un ajuste local (derecha), con un n de 200 y 10 respectivamente

	Prueba KS A.global	**Prueba KS A.local**
K_n^+	0.66688	1.3279
K_n^-	0.5530	1.1058
D_n	0.66688	1.32795
D_α	1.2238	1.2238
Validez	valido	no valido

Cuadro 1: Valores de la prueba KS con ajuste global (izquierda), asi como un ajuste local (derecha), con un n de 200 y 10 respectivamente.

mas obvia , pero el autor aun no esta convencido de esta decisión.

2. Mesones π

Los mesones π son partículas inestables de masa m_π =139.6 $\frac{MeV}{c^2}$. Su tiempo de vida promedio en el *sistema en reposo* es $\tau = 3.6 \times 10^{-8} s$

- Si su energía cinética es $200 MeV$, elabore un programa que simule cuántos piones sobreviviran después de viajar 20 m. Inicie con una muestra de 10^6 piones. Suponga que los piones son monoenergéticos.
- Modifique su programa de manera tal que los piones no sean monoenergéticos, si no que su energía tenga una distribución Gaussiana alrededor de 200 MeV con $\sigma = 50 MeV$.

Para resolver este problema , hay que recordar algunos conceptos importantes de relatividad especial, como lo es la *energia total de una partícula*, que es :

$$\begin{aligned} E = \gamma mc^2 &= \frac{mc^2}{\sqrt{1 - v^2/c^2}} \\ &= \frac{E_0}{\sqrt{1 - u^2/c^2}} = K + E_0 \end{aligned} \tag{8}$$

Por otro lado también sirve el recordar el

concepto de *Dilatación temporal*, representado a través de:

$$T' = \gamma T_0 \tag{9}$$

Donde T' es el tiempo medido desde el sistema de referencia en movimiento, mientras que T_0 es medido desde un sistema en reposo. Igualmente, el recordar que los muones tienen una probabilidad de decaimiento de tipo exponencial con la forma:

$$P(t) = e^{-t/\tau_{Tierra}} \tag{10}$$

Con esto puesto en la mesa , vemos que podemos obtener el valor de γ desde la ecuación 8, ya que nos dan la energía cinética K así como su energía en reposo m_π. Con γ podemos obtener la velocidad v , que es la velocidad medida desde la tierra, en nuestro sistema de referencia.

Un tema interesante es el hecho de que se menciona que la vida promedio, o tiempo de vida esta medido en el *sistema en reposo*, esto quiere decir que esta medido desde el punto de vista del muon , como si este estuviera estático y todo a su alrededor se moviera, por lo que podemos utilizar la ecuación 9 y la γ recién encontrada para conseguir el τ_{Tierra}.

Hasta este punto todo es analítico y no hay problema alguno cuando la cantidad de energía es estable (monoenergéticos), el aspecto nuevo es el hecho de utilizar variables aleatorias para calcular las partículas que sobrevivirán.

Definimos entonces una distribucion normal o gaussiana aparir de la media y la desviación estándar ya conocida, creando una cantidad de puntos aleatorios equivalentes a N_0 medidas (muestra) , usando estas energías aleatorias , hacemos el mismo proceso que antes hicimos con una sola energía . Al final vemos que una variable aleatoria sea menor que la probabilidad propuesta por la ecuación 10 , si esto es asi , entonces lo contamos como un muon que ha sobrevivido.

Esto queda plasmado en el codigo 2, que se encuentra en el apéndice A.2, donde encontramos los siguientes resultados:

Con un error porcentual entre ambos valores de 1.801 % , por lo que se puede decir que los resultados son muy parecidos , y el metodo de variables aleatorias es adecuado.

N_{o_π} **Monoenergéticos**	N_{o_π} **No Monoenergéticos**
433847	426152

Cuadro 2: Valores de la prueba KS con ajuste global (izquierda), asi como un ajuste local (derecha), con un n de 200 y 10 respectivamente.

3. Partículas en la caja

Considere una caja dividida en dos partes iguales por una pared. Al inicio, $t = 0$, hay N partículas en el lado izquierdo de la caja. Se abre un pequeño agujero en la pared y puede pasar una partícula a través del agujero por unidad de tiempo. Después de cierto tiempo, el sistema alcanza su estado de equilibrio con el mismo número de partículas en ambos lados. En lugar de determinar condiciones iniciales complicadas para el sistema de N partículas, lo modelaremos por un modelo estadístico simple. Para poder simular este sistema, que debe consistir en $N \gg 1$ partículas, suponemos que todas las partículas en el lado izquierdo tienen la misma probabilidad de pasar al lado derecho. Introducimos la etiqueta n_i para denotar el número de partículas en cada tiempo en el lado izquierdo y $n_d = N - n_i$ para las del lado derecho. La probabilidad de un movimiento a la derecha durante el paso de tiempo Δt es $\frac{n_i}{N}$. Elabore un algoritmo para simular este problema que debe considerar:

- La elección del número de partículas N.
- Hacer un ciclo en el tiempo, donde el tiempo máximo debe ser mayor que el número de partículas N.
- Para cada paso de tiempo Δt, hay una probabilidad $\frac{n_i}{N}$ para moverse a la derecha. Compare esta probabilidad con un número aleatorio x.
- Si $x \leq \frac{n_i}{N}$, decrezca el número de partículas en la mitad izquierda por uno, es decir, $n_i =$

$n_i - 1$. También debe mover una partícula a la mitad derecha $n_d = n_d + 1$.

- Incremente el tiempo en una unidad.

Realice 10 simulaciones, con suficiente número de partículas y suficiente tiempo para que el sistema alcance el equilibrio. Compare la gráfica de cada una de ellas $n_i$vs t y su promedio contra la solución analítica

$$n_i = \frac{N}{2}\left(1 + e^{-\frac{2t}{N}}\right). \tag{11}$$

Entonces, creamos un programa con las especificaciones en los `items` previos, que es el código 3, que se encuentra en el apéndice A.3.

En cada periodo de tiempo es posible que pase una partícula del lado izquierdo al lado derecho , en nuestro caso , elegimos 3000 partículas , en un tiempo de 5000 segundos.

El periodo de tiempo Δt elegido , y por naturalidad es el segundo. Por facilidad , creamos una lista de valores de n_i numero de partículas en la izquierda , que tenia por segundo , siendo tecnicamente i el correspondiente al tiempo transcurrido en segundos.

La parte importante , y aleatoria , transcurre cuando llamamos la variable `random` y al contrastarla con la probabilidad $\frac{n_i}{N}$, decidimos si una partícula pasa a la derecha o no .

Cuando n_i es muy cercano a N , entonces la probabilidad de debe ser muy alta , como `random` , basta con tomar que esta sea menor a $\frac{n_i}{N}$. Siendo un proceso aleatorio , conviene repetirlo , en este caso , 10 veces y remediar las simulaciones. De esta forma , obtenemos la grafica de la Figura **2** donde también añadimos los valores de n_i según la ecuación 11

Observando la figura **2** claramente algo esta mal a menos que el demonio de Maxwell sea un ente real , esto contradice la segunda ley de la termodinámica. Por lo que es claro que el problema radica en como estamos planteando el problema y en las hipótesis que estamos tomando por ciertas .

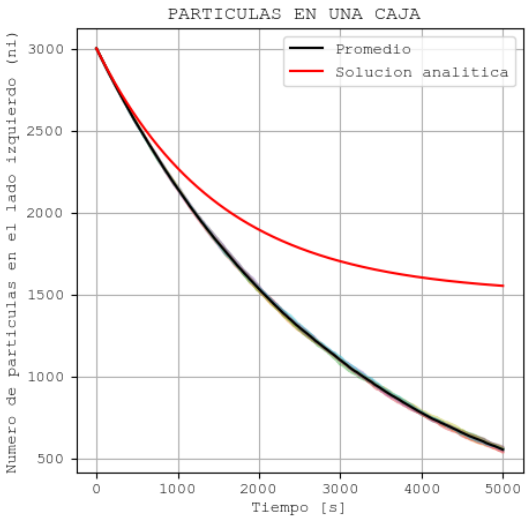

Figura 2: **Cambio en el tiempo de numero de partículas en el lado derecho de una caja** . En esta simulación no hay un buen ajuste de la dinámica de las partículas por lo que en numero de estas decae hasta que todas cruzan a la derecha.

Por lo que el autor pudo deducir, el defecto queda en que suponemos que unicamente hay particulas que salen de izquierda a derecha , y no viceversa, ya que es claro que llega un punto donde las particulas del lado derecho igualan o superan a las del lado izquierdo, por lo que no deberia haber ningun impedimento para que salgan de la derecha y entren a la izquierda.

Haciendo una pequeña modificacion al codigo , note que esto no era suficiente, tambien importa el orden de como se esta decidiendo que variable tendria primero la probabilidad de arrojar una particula al otro lado.

Si tomamos que la derecha solo puede arrojar particulas en el caso de que el lado izquierdo no pueda , entonces, la probabilidad se esta haciendo dependiente, y esto no debe ser asi, ambos lados, en funcion de su cantidad de particulas , deberian de tener la probabilidad de arrojar primero la particula al otro lado.

Entonces definimos una tercer variable aleatoria con la que definicmos quien arroja primero, consiguiendo el código 4, que se encuentra en el apéndice A.4, donde implementamos todas las observaciones anteriores, que se ven verificadas en la Figura **3**, la cual tiene un buen ajuste con la solución analítica, al menos a primera vista.

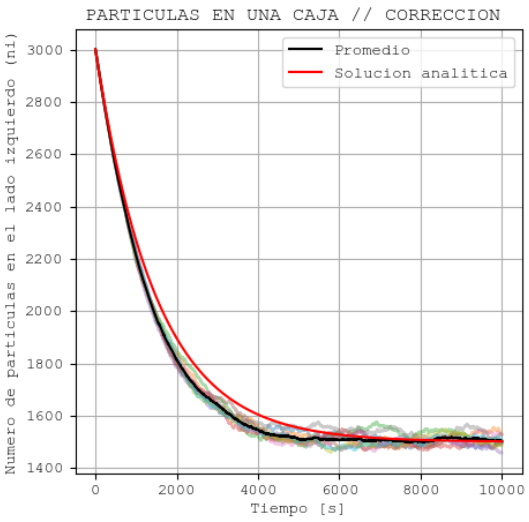

Figura 3: **Cambio en el tiempo de numero de partículas en el lado derecho de una caja modificada** . En esta simulación un ajuste adecuado de la dinámica de las partículas por lo que en numero de estas se ajusta mejor con la solución analítica que verifica que esta va de acuerdo con la física real

Apéndice

A. Código en Python

A.1. Prueba Kolmogorov-Smirnov

```
import numpy as np
from scipy.stats import kstest
import random
import matplotlib.pyplot as plt
from scipy.stats import uniform

# Generar una muestra de datos utilizando la funcion random()
n = 200# Tamano de la muestra
N = 1000
r=20
w=n//r
alpha=0.1

data = [random.random() for _ in range(n)]

dataS = sorted(data)


#Distribucion Uniforme
def F(x):
    h = uniform.cdf(x, loc=0, scale=1)
    return h

#Definicion de funciones  de Kolmogorov - Smirnov

def kMENOS(datos, distribucion):
    num = len(datos)
    q = np.zeros(num)
    for j in range(num):
        q[j] = distribucion(datos[j]) - (j / num)
    return  np.sqrt(num)*np.max(q)

def kMAS(datos, distribucion):
    num = len(datos)
    q = np.zeros(num)
    for j in range(num):
        q[j] = (j + 1) / num - distribucion(datos[j])
    return np.sqrt(num)*np.max(q)


def Local(datos,Cmenos,Cmas,distribucion,r):
    num=len(datos)
    KSmas=np.zeros(r)
    KSmenos=np.zeros(r)
    e=num//r #Dividimos el rango de nuestra muestra en pequenas partes

    dat=np.zeros(e)
    for i in range(r):
        for j in range(e):
            dat[j]=datos[j+i]
        datS=sorted(dat)
        KSmas[i]=Cmas(datS,distribucion)
        KSmenos[i]=Cmenos(datS,distribucion)

    Kmas=Cmas(sorted(KSmas),F)
```

```
    Kmenos=Cmenos(sorted(KSmenos),F)
    return KSmas,KSmenos,Kmas,Kmenos


def D(n,a):# valor critico
    return np.sqrt(-(1/(2))*np.log(a/2))


# -----Aplicar la prueba de Kolmogorov-Smirnov GLOBAL ARTESANAL ---
kmas=kMAS(dataS,F)
kmenos=kMENOS(dataS,F)

print('----Estadistico de prueba KS Artesanal SOLO GLOBAL----')

print(f'K+{n}:',kmas)
print(f'K-{n}:',kmenos)

Dn1=max(kmas,kmenos)

print(f'D{n}:',Dn1)
print(f'D{alpha}:',D(n,alpha))

if Dn1<=D(n,alpha):
    print('Se acepta la hipotesis de que la funcion random crea una dsitribucion uniforme ')
else:
    print('Se descarta la hipotesis de que la funcion random crea una dsitribucion uniforme ')


# -----Aplicar la prueba de Kolmogorov-Smirnov Parche LOCAL ARTESANAL ---
KSmas,KSmenos,Kmas,Kmenos=Local(data,kMENOS,kMAS,F,r)

print('----Estadistico de prueba KS Artesanal parche LOCAL----')

print(f'K+{w}:',Kmas)
print(f'K-{w}:',Kmenos)

Dn2=max(Kmas,Kmenos)

print(f'D{w}:',Dn2)
print(f'D{alpha}:',D(w,alpha))

if Dn2<=D(w,alpha):
    print('Se acepta la hipotesis de que la funcion random crea una dsitribucion uniforme ')
else:
    print('Se descarta la hipotesis de que la funcion random crea una dsitribucion uniforme ')

#-----GRAFICACION ----------
# Definimos el tamano de la figura
plt.figure(figsize=(12, 6))

# Ajustes de fuente a Courier New
plt.rc('font', family='Courier New')

y_ecdf = np.arange(1, n + 1) / n
x_ecdf = np.sort(data)

# Calcular la funcion de distribucion acumulada para KSmas y KSmenos
y_ksmas = np.arange(1, r + 1) / r

# Crear figura y subplots
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(14, 6))

# Graficar la ECDF de los datos en el primer subplot
ax1.step(x_ecdf, y_ecdf, where='post', label='ECDF')
ax1.plot(np.linspace(0, 1, N), F(np.linspace(0, 1, N)), label='CDF Uniforme')
ax1.set_xlabel('x')
ax1.set_ylabel('F(x)')
ax1.set_title('ECDF y CDF Uniforme')
ax1.grid(True)
ax1.legend()

# Graficar la distribucion acumulada de KSmas en el segundo subplot
ax2.step(np.sort(KSmas), y_ksmas, where='post', label='KSmas')
ax2.plot(np.linspace(min(KSmas), max(KSmas), N), uniform.cdf(np.linspace(min(KSmas), max(KSmas), N),loc=min(KSmas), scale=max(KSmas)-min(KSmas)), label='CDF Uniforme')
ax2.set_xlabel(f'KSmas{w}')
ax2.set_ylabel('Probabilidad acumulada')
ax2.set_title('Distribucion acumulada de KSmas')
ax2.grid(True)
ax2.legend()


plt.tight_layout()
plt.show()
```

Código 1: Programa que verifica si un conjunto de datos es , bajo ciertos parametros , aleatorio usando la prueba de Kolmogorov-Smirnov . NOMBRE : "K-smirnov.py"

A.2. Piones

```python
import numpy as np
import matplotlib.pyplot as plt
import random
# Constantes
m_pi = 139.6  # MeV/c^2
tau = 3.6e-8  # s
c = 3e8  # m/s
E_k = 200  # MeV
d = 20  # m
N0 = 1e6  # Numero inicial de piones

# ----Piones Monoenergeticos-----------

E_tot = E_k + m_pi  # Energia total en MeV
gamma = E_tot / m_pi
v = c * np.sqrt(1 - 1/gamma**2)
tau_lab = gamma * tau

# Distancia a recorrer y tiempo
t = d / v

# Probabilidad de supervivencia
P_surv = np.exp(-t / tau_lab)
N_surv1 = N0 * P_surv

print(f"Probabilidad de supervivencia: {
    P_surv:.6f}")
print(f"Numero de piones monoenergeticos
    que sobreviven: {int(N_surv1)}")


# ----Piones NO Monoenergeticos-----------

# Parametros de la distribucion gaussiana
E_mean = 200  # MeV
E_sigma = 50  # MeV

# Generar las energias cineticas
np.random.seed(425)  # Para
    reproducibilidad
E_kinetic = np.random.normal(E_mean,
    E_sigma, int(N0))

# Filtrar energias no fisicas (negativas)
E_kinetic = E_kinetic[E_kinetic > 0]

# Calcular el numero de piones que
    sobreviven
N_surv = 0
for E_k in E_kinetic:
    E_tot = E_k + m_pi
    gamma = E_tot / m_pi
    v = c * np.sqrt(1 - 1/gamma**2)
    tau_lab = gamma * tau
    t = d / v
    P_surv = np.exp(-t / tau_lab)
    if np.random.rand() < P_surv:
        N_surv += 1

print(f"Numero de piones no
    monoenergeticos  que sobreviven: {
    N_surv}")

print('el error relativo es entonces:',(
    N_surv1-N_surv)/N_surv1)
```

Código 2: Programa que encuentra el tiempo tau de laboratorio de un muon para determinar su tiempo de vida y asi determinar cuantos muones sobreviven . NOMBRE : "muones.py"

A.3. Caja

```python

import numpy as np
import matplotlib.pyplot as plt

N = 3000 # Numero de PARTICULAS global
t = 5000 # TIEMPO , mayor que el numero de
     particulas
nS = 10 #numero de simulaciones

#------Definicion de Funciones--------
def SIM(N,t):

    ni = N # Numero de particulas inicial
    del lado izquierdo igual a N
    ni_t = [ni] #En esta lista colocamos
    el numero de particulas  segun vaya
    transcurriendo el tiempo t

    for i in range(t):
        if ((ni/N)>np.random.rand()): #
    Conseguimos un numero aleatorio entre
    0 y 1 y lo comparamos con la
    probabilidad ni/N
            ni -= 1 # En el caso de ser
    cierto , una particula va a la derecha
     , perdiendo una del lado izquierdo.
        ni_t.append(ni) #Anadimos el nuevo
     numero de particulas en el lado
    izquierdo
    return ni_t

simulaciones  = [] #Guardamos simulaciones
for i in range(nS):
    simulaciones.append(SIM(N, t)) #
    Hacemos un nS veces la simulacion


time = np.arange(t) #arreglo en el tiempo
    para la solucion analitica
```

```
sol_an = (N/2) * (1 + np.exp(-2 * time / N )) #calculo analItico
ProM = np.mean(simulaciones, axis=0) # Promedio de todas las simulaciones // Para que se haga esto sobre las columnas , se coloca axis=0

#-----GRAFICACION ---------------
# Definimos el tamano de la figura
plt.figure(figsize=(5, 5))

# Ajustes de fuente a Courier New
plt.rc('font', family='Courier New')


plt.figure(1)
for i in simulaciones:
    plt.plot(i, alpha=0.4) #Alfa determina lo translucido de la grafica

plt.plot(ProM,color="black",linewidth=1.5, label='Promedio')
plt.plot(time, sol_an, 'r', label='Solucion analitica')

plt.xlabel("Tiempo [s]")
plt.ylabel("Numero de particulas en el lado izquierdo (ni)")
plt.title("PARTICULAS EN UNA CAJA ")
plt.legend()
plt.grid(True)
plt.show()
```

Código 3: Programa que genera simulaciones de una mitad izquierda de caja que despide particulas a el lado derecho . NOMBRE : ”3.c4ja”

A.4. Caja MODIFICADA

```
import numpy as np
import matplotlib.pyplot as plt

N = 3000 # Numero de PARTICULAS global
t = 10000 # TIEMPO , mayor que el numero de particulas
nS = 10 #numero de simulaciones

#------Definicion de Funciones--------
def SIM(N,t):

    ni = N # Numero de particulas inicial del lado izquierdo igual a N
    ni_t = [ni] #En esta lista colocamos el numero de particulas del lado izquierdo segun vaya transcurriendo el tiempo t
    nd=0 #Numero de particulas inicial del lado derecho igual a N
    nd_t=[nd]  #En esta lista colocamos el numero de particulas del lado derecho segun vaya transcurriendo el tiempo t

    for i in range(t):
        s=np.random.rand() #Variable aleatoria si una particula sale de la izquierda
        u=np.random.rand() #Variable aleatoria si una particula sale de la iderecha
        k=np.random.rand() #Variable aleatoria que determina que lado de la caja tendra prioridad  para expulsar una particula
        if (ni/N)>k:  #Prioridad en la izquierda
            if ni>0:  #Nos aseguramos que la caja tenga particulas
                if (ni/N)>s: #Conseguimos un numero aleatorio entre 0 y 1 y lo comparamos con la probabilidad ni/N
                    ni -= 1 # En el caso de ser cierto , una particula va a la derecha , perdiendo una del lado izquierdo.
                    nd += 1
                elif nd>0 :
                    if (nd/N)>u:
                        ni += 1 # En el caso de ser cierto , una particula va a la derecha , perdiendo una del lado izquierdo.
                        nd -= 1
        else:#Prioridad en la iderecha
            if nd>0: #Nos aseguramos que la caja tenga particulas
                if (nd/N)>s: #Conseguimos un numero aleatorio entre 0 y 1 y lo comparamos con la probabilidad ni/N
                    nd -= 1 # En el caso de ser cierto , una particula va a la derecha , perdiendo una del lado izquierdo.
                    ni += 1
                elif ni>0 :
                    if (ni/N)>u:
                        nd += 1 # En el caso de ser cierto , una particula va a la derecha , perdiendo una del lado izquierdo.
                        ni-= 1
        ni_t.append(ni) #Anadimos el nuevo numero de particulas en el lado
```

```
    izquierdo
    print(nd)
    return ni_t

simulaciones  = [] #Guardamos simulaciones
for i in range(nS):
    simulaciones.append(SIM(N, t)) #
    Hacemos un nS veces la simulacion


time = np.arange(t) #arreglo en el tiempo
    para la solucion analitica
sol_an = (N/2) * (1 + np.exp(-2 * time / N
    )) #calculo analItico
ProM = np.mean(simulaciones, axis=0) #
    Promedio de todas las simulaciones //
    Para que se haga esto sobre las
    columnas , se coloca axis=0

#-----GRAFICACION ---------------
# Definimos el tamano de la figura
plt.figure(figsize=(5, 5))

# Ajustes de fuente a Courier New
plt.rc('font', family='Courier New')


plt.figure(1)
for i in simulaciones:
    plt.plot(i, alpha=0.4) #Alfa determina
     lo translucido de la grafica

plt.plot(ProM,color="black",linewidth=1.5,
     label='Promedio')
plt.plot(time, sol_an, 'r', label='
    Solucion analitica')

plt.xlabel("Tiempo [s]")
plt.ylabel("Numero de particulas en el
    lado izquierdo (ni)")
plt.title("PARTICULAS EN UNA CAJA //
    CORRECCION ")
plt.legend()
plt.grid(True)
plt.show()
```

Código 4: Programa que genera simulaciones de una mitad izquierda de caja que despide particulas a el lado derecho con un buen modelo para la busqueda equilibrio entre las particulas de la caja . NOMBRE : ”3.c4ja2.py”