MonteCarlo

June 6, 2019

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import fsolve
import random
```

In [2]:
```
#La trayectoria del muon en nuestra simulación viene descrita por los parámetros (x,y,

L = 25.4

#El estudio de los centelleadores con una distancia h_min nos dará la fracción máxima

h_min = 24.3
```

In [ ]:
```
#Representamos los centelleadores

plt.figure()

plt.plot([0,L], [0,0], color='g')


plt.plot([0,L], [h_min,h_min], color='g')

#-------------------------------------------------------------------------------------

i=0

#Número de muones que generamos

N = 0

#Inicializamos el número de muones que pasan ambos detectores

N_detec = 0

while i<5*10**4:

#Generador de posiciones x en el centelleador inferior según una distribución uniforme
```

```
    r1 = random.random()
    x = r1*L

#Generador de posiciones y en el centelleador inferior según una distribución uniforme

    r2 = random.random()
    y = r2*L

#Generador de ángulo phi de la trayectoria del muon según una distribución uniforme (es

    r3 = random.random()
    phi = r3*2*np.pi

#Generador de números aleatorios según la distribución de probabilidad no uniforme P(t

    r4= random.random()
    func = lambda theta : r4 - 4*(np.cos(theta)**3)*np.sin(theta)
    theta_solution = fsolve(func,r4)

    if (theta_solution >= 0) & (theta_solution < np.pi/2):

        N = N + 1

        #print('Posición x en el cent. inferior {:.2f}'.format(x))
        #print('Posición y en el cent. inferior {:.2f}'.format(y))
        #print('El angulo phi del muon es {:.2f} radian'.format(phi))
        #print('El angulo theta del muon es {} radian'.format(theta_solution))

        x_prima = x + h_min*(np.cos(phi)*np.tan(theta_solution))
        y_prima = y + h_min*(np.sin(phi)*np.tan(theta_solution))

        #print('x prima es {}'.format(x_prima))

        if (x_prima >= 0) & (x_prima <= L):

            if (y_prima >= 0) & (y_prima <= L):

                N_detec = N_detec + 1

                plt.plot([x,x_prima], [0, h_min], color='b')

        else:

            plt.plot([x,x_prima], [0, h_min], color='r')

            #print('Particula detectada')
    #else:
```

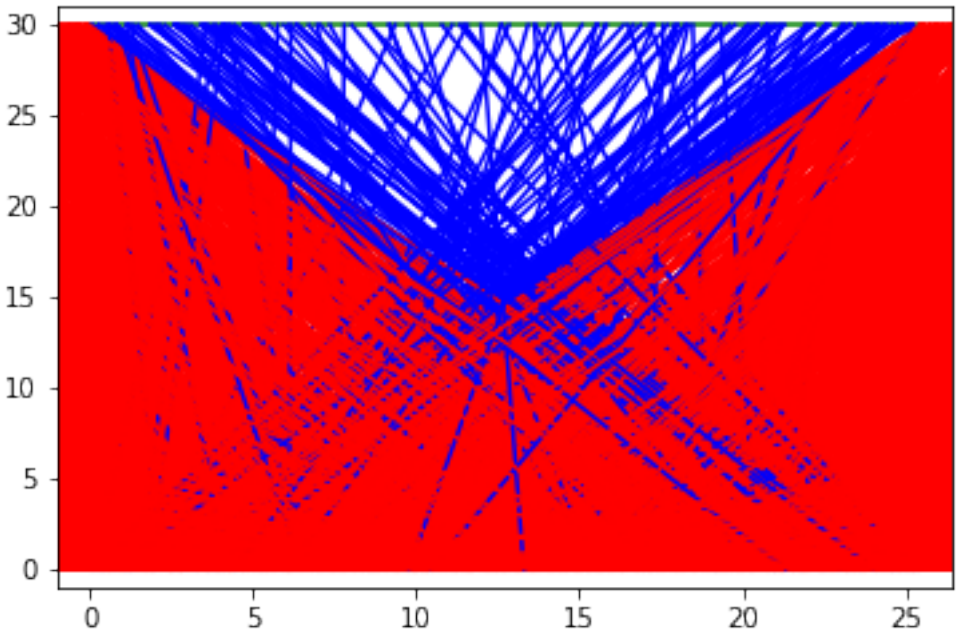
```
            #print('dou')

        i=i+1


    plt.xlim(-1,L+1)
    plt.ylim(-1, h_min+1)

    print('El numero de muones generados ha sido {}'.format(N))
    print('El numero de muones detectados en coincidencias ha sido {}'.format(N_detec))
    print('El valor de F_max={}'.format(N_detec/N))
```

In [33]:
```
    #El estudio de los centelleadores con una distancia h_max nos dará la fracción mínima

    h_max = 30

    #Representamos los centelleadores

    plt.figure()

    plt.plot([0,L], [0,0], color='g')


    plt.plot([0,L], [h_max,h_max], color='g')

    #-------------------------------------------------------------------------------------

    i=0

    #Número de muones que generamos

    N = 0

    #Inicializamos el número de muones que pasan ambos detectores

    N_detec = 0

    while i<5*10**3:

    #Generador de posiciones x en el centelleador inferior según una distribución uniform

        r1 = random.random()
        x = r1*L

    #Generador de posiciones y en el centelleador inferior según una distribución uniform

        r2 = random.random()
        y = r2*L
```

```
#Generador de ángulo phi de la trayectoria del muon según una distribución uniforme (

    r3 = random.random()
    phi = r3*2*np.pi

#Generador de números aleatorios según la distribución de probabilidad no uniforme P(

    r4= random.random()
    func = lambda theta : r4 - 4*(np.cos(theta)**3)*np.sin(theta)
    theta_solution = fsolve(func,r4)

    if (theta_solution >= 0) & (theta_solution < np.pi/2):

        N = N + 1

        #print('Posición x en el cent. inferior {:.2f}'.format(x))
        #print('Posición y en el cent. inferior {:.2f}'.format(y))
        #print('El angulo phi del muon es {:.2f} radian'.format(phi))
        #print('El angulo theta del muon es {} radian'.format(theta_solution))

        x_prima = x + h_max*(np.cos(phi)*np.tan(theta_solution))
        y_prima = y + h_max*(np.sin(phi)*np.tan(theta_solution))

        #print('x prima es {}'.format(x_prima))

        if (x_prima >= 0) & (x_prima <= L):

            if (y_prima >= 0) & (y_prima <= L):

                N_detec = N_detec + 1

                plt.plot([x,x_prima], [0, h_max], color='b')

        else:

            plt.plot([x,x_prima], [0, h_max], color='r')

            #print('Particula detectada')
    #else:

        #print('dou')

    i=i+1


plt.xlim(-1,L+1)
plt.ylim(-1, h_max+1)
```

```
        print('El numero de muones generados ha sido {}'.format(N))
        print('El numero de muones detectados en coincidencias ha sido {}'.format(N_detec))
        print('El valor de F_max={}'.format(N_detec/N))
```

```
El numero de muones generados ha sido 4641
El numero de muones detectados en coincidencias ha sido 181
El valor de F_max=0.039000215470803706
```



In [ ]:

In [ ]:

In [ ]:

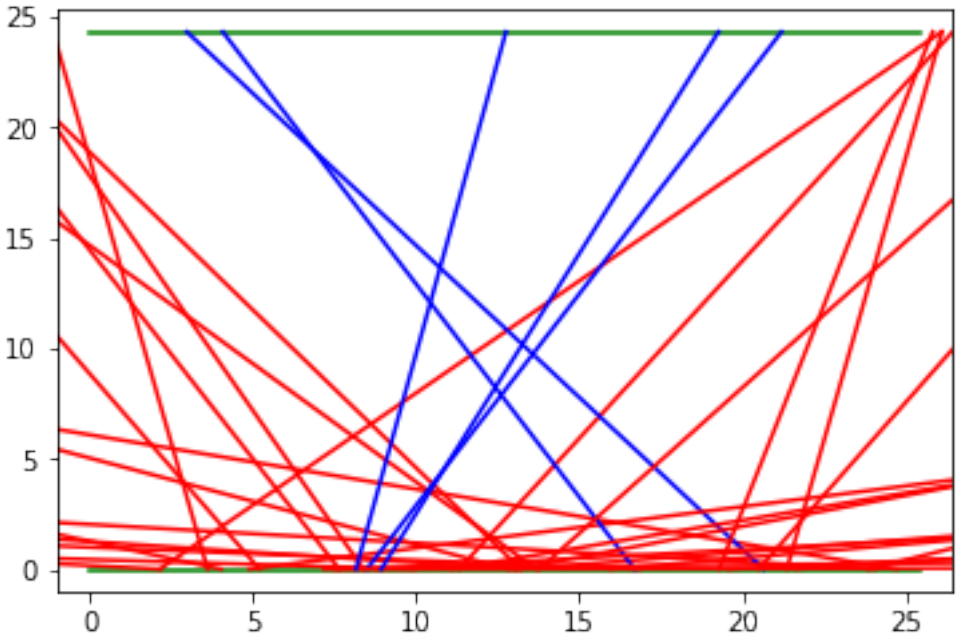
In [ ]:

In [34]:
```
#Representamos los centelleadores

        plt.figure()

        plt.plot([0,L], [0,0], color='g')


        plt.plot([0,L], [h_min,h_min], color='g')
```

```
#-------------------------------------------------------------------------------

i=0

#Número de muones que generamos

N = 0

#Inicializamos el número de muones que pasan ambos detectores

N_detec = 0

while i<5*10**1:

#Generador de posiciones x en el centelleador inferior según una distribución uniform

    r1 = random.random()
    x = r1*L

#Generador de posiciones y en el centelleador inferior según una distribución uniform

    r2 = random.random()
    y = r2*L

#Generador de ángulo phi de la trayectoria del muon según una distribución uniforme (

    r3 = random.random()
    phi = r3*2*np.pi

#Generador de números aleatorios según la distribución de probabilidad no uniforme P(

    r4= random.random()
    func = lambda theta : r4 - 4*(np.cos(theta)**3)*np.sin(theta)
    theta_solution = fsolve(func,r4)

    if (theta_solution >= 0) & (theta_solution < np.pi/2):

        N = N + 1

        #print('Posición x en el cent. inferior {:.2f}'.format(x))
        #print('Posición y en el cent. inferior {:.2f}'.format(y))
        #print('El angulo phi del muon es {:.2f} radian'.format(phi))
        #print('El angulo theta del muon es {} radian'.format(theta_solution))

        x_prima = x + h_min*(np.cos(phi)/np.tan(theta_solution))
        y_prima = y + h_min*(np.sin(phi)/np.tan(theta_solution))
```

```
                #print('x prima es {}'.format(x_prima))

                if (x_prima >= 0) & (x_prima <= L):

                    if (y_prima >= 0) & (y_prima <= L):

                        N_detec = N_detec + 1

                        plt.plot([x,x_prima], [0, h_min], color='b')

                else:

                    plt.plot([x,x_prima], [0, h_min], color='r')

                    #print('Particula detectada')
            #else:

                #print('dou')

            i=i+1


        plt.xlim(-1,L+1)
        plt.ylim(-1, h_min+1)

        print('El numero de muones generados ha sido {}'.format(N))
        print('El numero de muones detectados en coincidencias ha sido {}'.format(N_detec))
        print('El valor de F_max={}'.format(N_detec/N))
```

```
El numero de muones generados ha sido 48
El numero de muones detectados en coincidencias ha sido 5
El valor de F_max=0.10416666666666667
```



In [ ]: