Felipe Cornejo I. 20.427.782-6 | Redes de Computadores | Control 5 | 2021-2

**P1. (3.0 pts.) Modulación Digital**

a. (2.0) Utilice la Figura 1 como base para implementar un modulador FSK con las portadoras en f0=75 Hz y f1=125 Hz y una tasa de datos de 10 bits por segundo.
   i. (1.0) Realice una implementación en Python del modulador propuesto. Genere 10 bits aleatorios, module y muestra la salida en función del tiempo. Indique qué parámetros del modulador puede verificar con este gráfico y justifique si se cumplen o no.
   ii. (0.5) Genere 100 bits aleatorios y grafique la señal en el tiempo y en frecuencia (Transformada de Fourier). ¿Indique qué parámetros del modulador puede verificar con estos gráficos y justifique si se cumplen o no?
   iii. (0.5) Varíe la tasa de datos del modulador y utilice una prueba similar al punto ii) hasta encontrar experimentalmente la máxima tasa de datos que se puede lograr con las frecuencias portadoras dadas.

b. (1.0) Una señal QAM de la forma

$$s(t) = d_1(t)\, cos(2\pi f_c t) + d_2(t)\, sen(2\pi f_c t)$$

se genera usando el modulador de la Figura 2.a. Demuestre que el demodulador de la Figura 2.b puede generar las señales $d_1(t)$ y $d_2(t)$ que combinadas recuperan la señal de entrada $d(t)$

Se escogió realizar el problema 1.

a)
i. Utilizando Python 3.10+, se realizó la implementación de la modulación de una señal de 10 bits aleatorios. Por partes, se definieron las siguientes funcionalidades para ser ocupadas. Se importan las siguientes librerías:

```
import matplotlib.pyplot as plt
import numpy as np
import random as rn
from scipy.fftpack import fftshift
from numpy import cos, pi
```

```
#Entradas: nada
#Salidas: un arreglo de largo 10 y otro 100, los cuales contienen enteros 0
y 1 almacenados de forma aleatoria
#Genera 2 arreglos representando señales de bits aleatorios.
def gen_signals():
    array_10 = []
    array_100 = []

    i = 0
    while i<10:
        array_10.append(rn.randint(0, 1))
        i+=1

    i = 0
    while i<100:
        array_100.append(rn.randint(0, 1))
        i+=1
    return array_10, array_100

#Entradas: la tasa de bits.
#Salidas: np.array el cual funcionará como arreglo de tiempo.
#Genera 2 np.array que representan el tiempo de una función
def gen_time(tbits):
    t_10 = np.linspace(0, tbits,1000)
    t_100 = np.linspace(0, tbits,1000)
    return t_10, t_100

#Entrada: 2 np.array que representen el tiempo
#Salida: 2 Señales portadoras, la primera a 75 Hz y la segunda a 125 Hz
#Genera 2 señales portadoras.
def gen_porta(t_10,t_100):
    #Frecuencias
    freq0 = 75
    freq1 = 125
```

```
    #Definición de las señales portadoras
    porta_75 = cos(2*pi*freq0*t_10)
    porta_125 = cos(2*pi*freq1*t_100)
    return porta_75, porta_125
```

Las funcionalidades presentadas hasta el momento solo son generadoras de los datos iniciales para poder realizar el ejercicio. Junto con el uso de estas se define también la Tasa de bits.

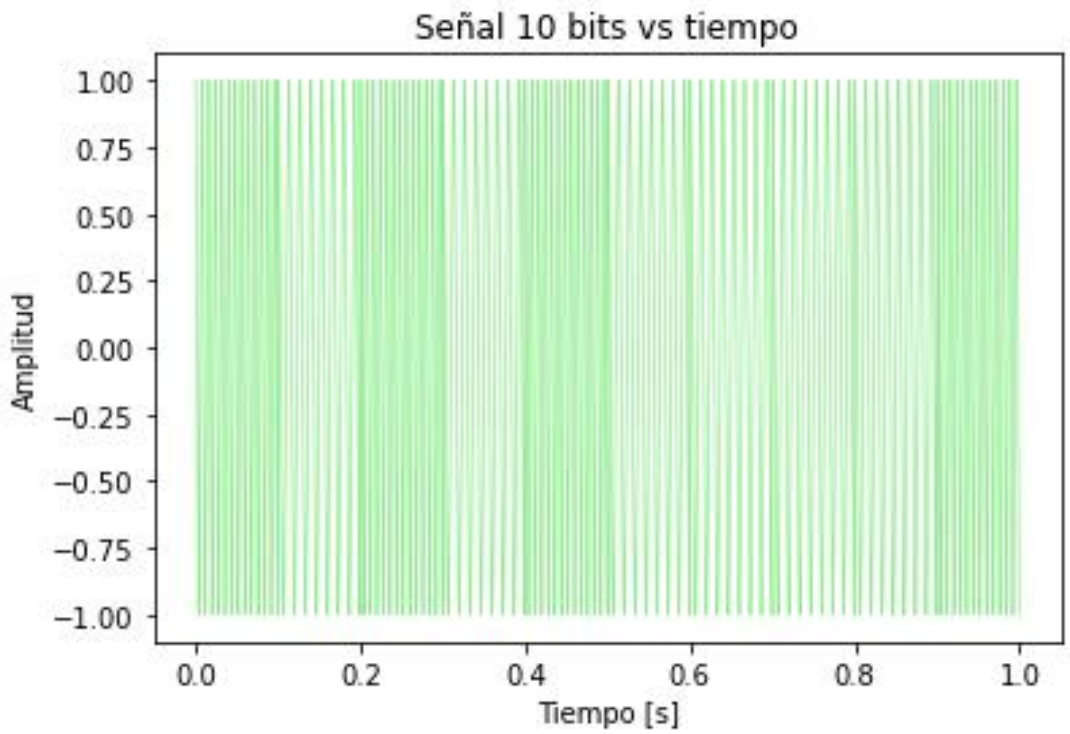
```
#Definir Tasa de bits.
tbits = 1/10

#Generar las señales correspondientes a 10 y 100 bits.
array_10, array_100 = gen_signals()
#Generar los vectores de tiempo debido a una tasa de 10 bits por segundo
t_10, t_100 = gen_time(tbits)
#Generar las señales portadoras debido a los vectores de tiempo.
porta_75, porta_125 = gen_porta(t_10,t_100)
```

Por siguiente se realiza la funcionalidad del Modulador, de tal modo que al encontrar un 0 en la señal responda para la primera señal portadora, y si encuentra un 1, responde añadiendo la siguiente señal. Cabe destacar que la primera señal se encuentra a 75 Hz y la segunda a 125 Hz.

```
#Entrada: La señal a modular, la señal portadora 1 y la señal portadora 2
#Salida: La señal modulada en FSK
#Modula una lista de 0 y 1 que represnenta una señal de datos binarios.
def modulador_FSK(signal,porta_75,porta_125):
    modulada = []
    for e in signal:
        if e == 0:
            modulada.extend(porta_75)
        else:
            modulada.extend(porta_125)
    return modulada
```

Con esta señal modulada, se vuelve a realizar un vector de tiempo y para que se ajuste a la muestra obtenida. Luego de esto se grafica respecto a ese tiempo.

```
#Modulacion FSK
modulada_10 = modulador_FSK(array_10, porta_75, porta_125)
modulada_100 = modulador_FSK(array_100, porta_75, porta_125)

tiempo_10 = np.linspace(0,len(array_10)*tbits,1000*len(array_10))

plotter("figura1", "Señal 10 bits vs tiempo", "Tiempo [s]", "Amplitud",
tiempo_10, modulada_10)
```




Se pueden apreciar las variaciones de frecuencias que se obtiene al modular, junto con eso también se aprecia que la amplitud no varía entre -1 y 1 debido al uso de la modulación de coseno asumiendo que la amplitud es 1. (A = 1).

(Esta funcionalidad de obtener el gráfico se utilizó la misma que en los laboratorios)

ii. Para la segunda parte, y revisar la transformada de Fourier de la señal modulada, utilizando una señal de 100 bits. Se utilizó, como se vió en el item anterior, las mismas funcionalidades para generar los elementos iniciales.

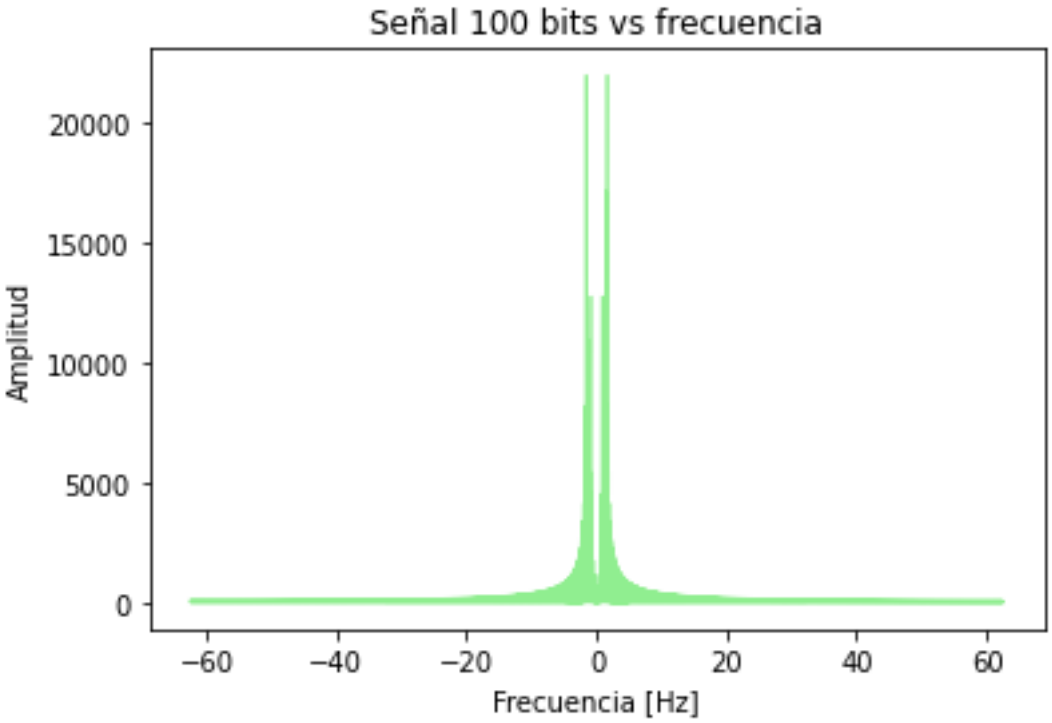
Por lo que solo falta diseñar la función de la transformada de Fourier. Utilizando las librerías correspondientes ya importadas en el item 1.

```
#Entrada:Arreglo de amplitudes de una señal, la frecuencia de la señal.
#Salida: Arreglo de amplitudes de la transformada de fourier, arreglo de
frecuencias de la señal
#Funcion que calcula la transformada de fourier entregando el eje x e y (se
hizo esta copia ya que la primera no dejaba ingresar por parametro datos de
tipo np.ndarray)
def fourier(data,freq):
    largo = len(data)
    tiempo = largo/freq
    fourier = np.fft.fft(data)
    aux = np.arange(-len(fourier)/2,len(fourier)/2)
    #shift
    ffreq=fftshift(aux/tiempo)
    return fourier,ffreq
```

Luego la llamada a la función en el bloque principal y su grafico es de la siguiente manera.

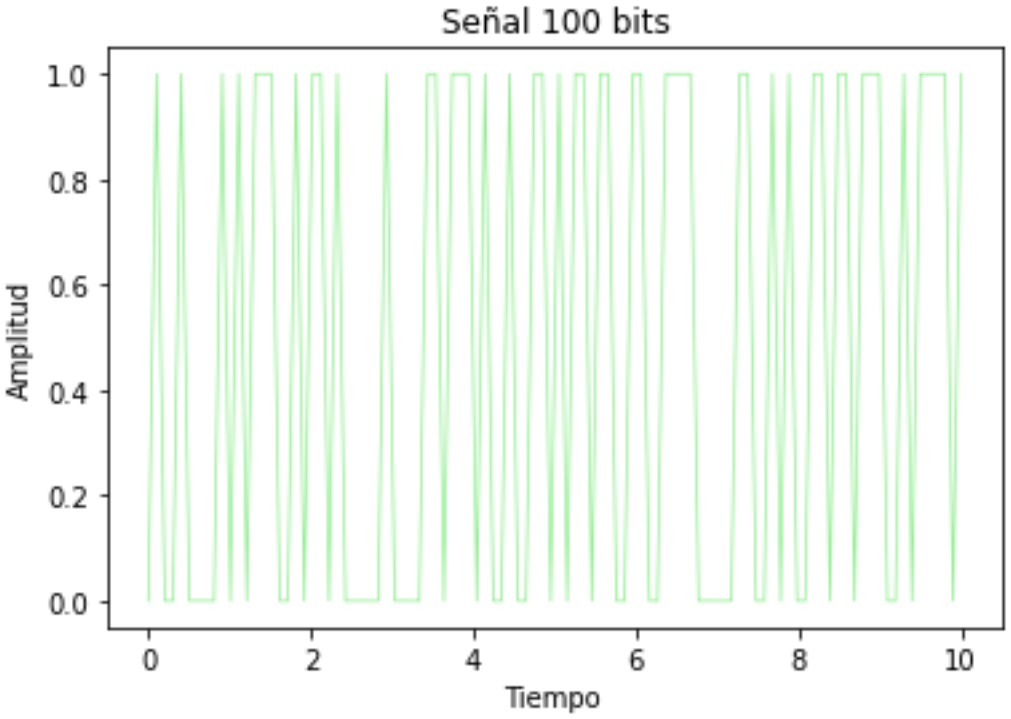
```
fourier_100,ffreq_100 = fourier(modulada_100,125)

plotter("figura2", "Señal 100 bits vs frecuencia", "Frecuencia [Hz]",
"Amplitud", ffreq_100, abs(fourier_100))
```

Finalmente, el gráfico queda de esta manera.




Se puede apreciar que el ancho de banda para esta señal es muy pequeño. Pero su amplitud se eleva a más de 20.000.

Además este es el gráfico de la señal en el tiempo.

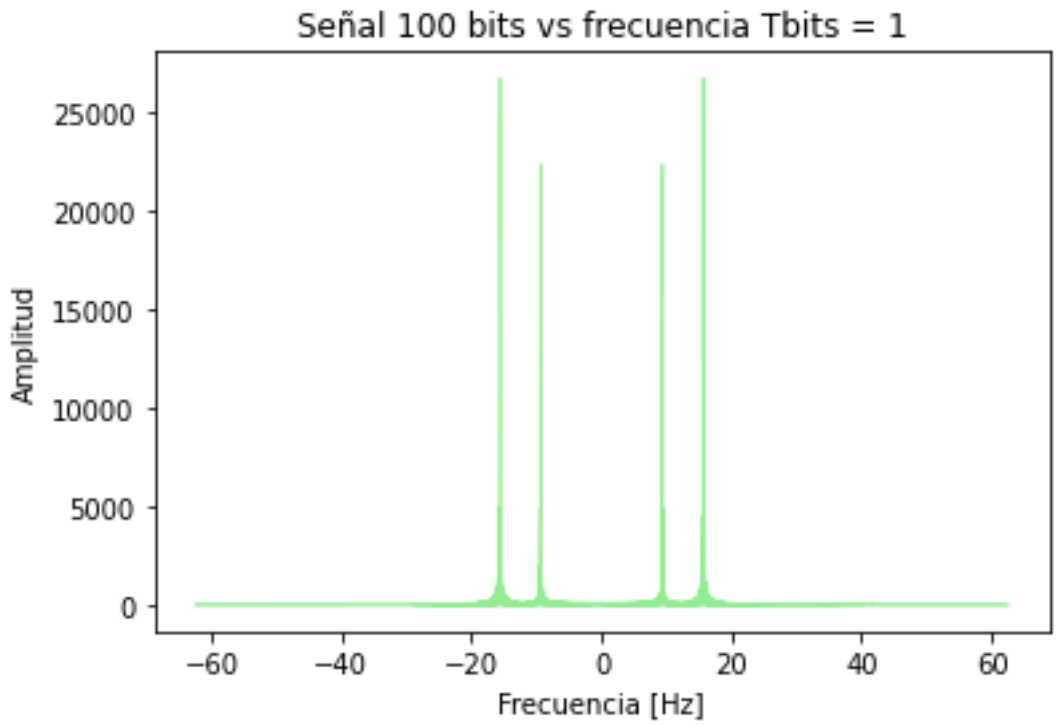


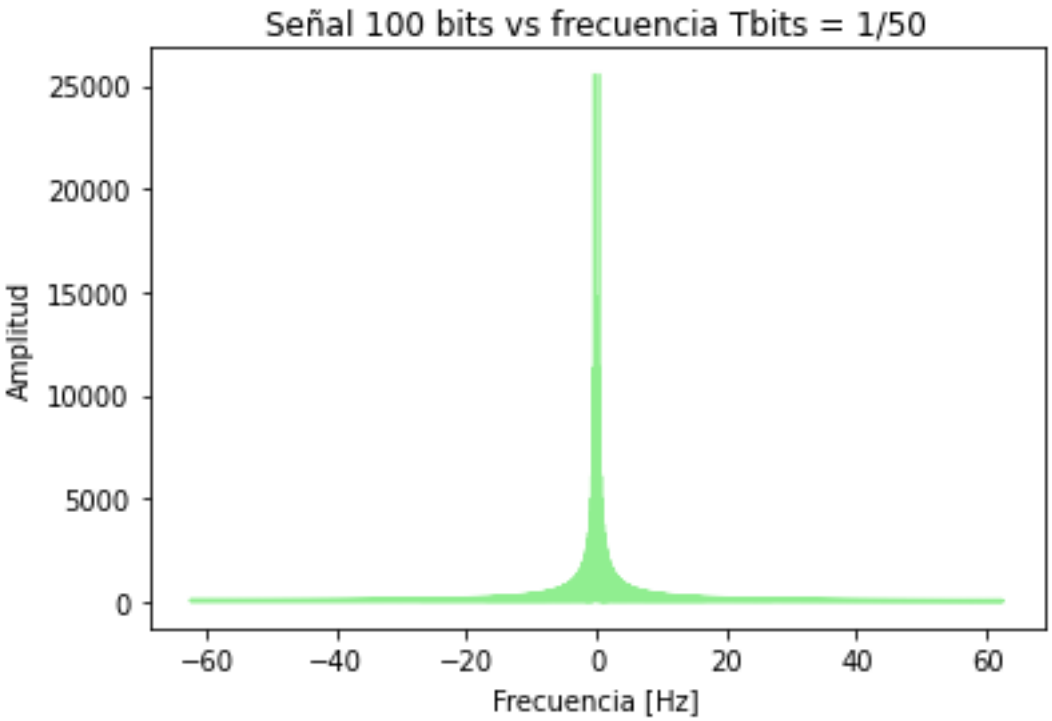
Iii. Para esta etapa se redefinieron 5 nuevas tasas de bits para observar lo que ocurre cuando aumentan los bits por segundo.

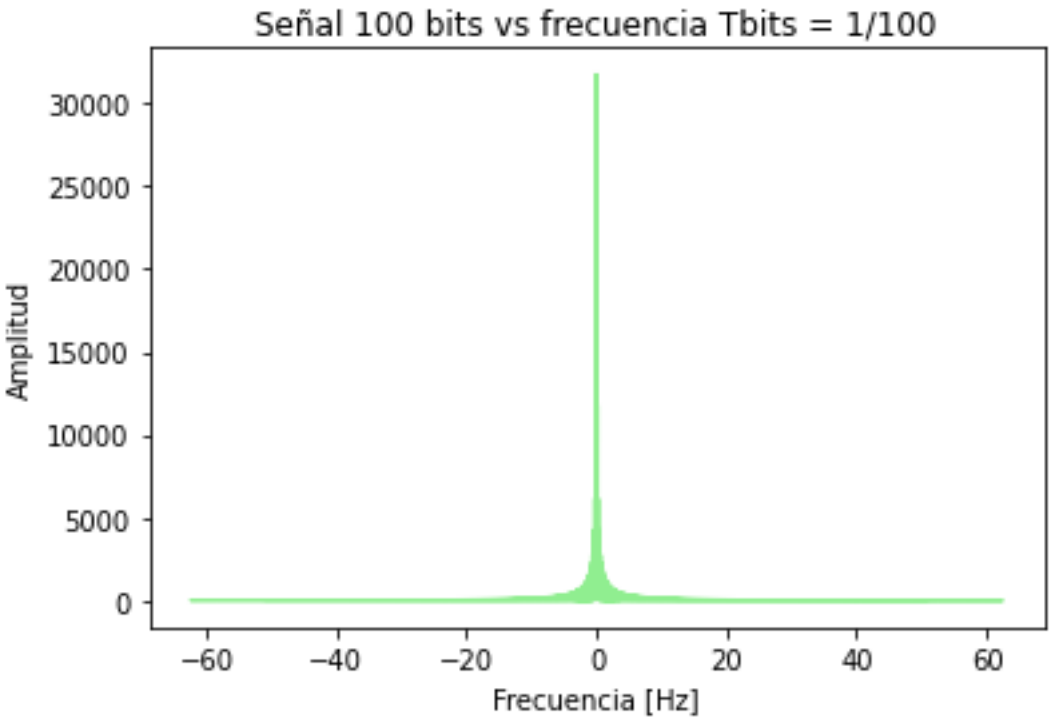
```
#Definir Tasa de bits.
tbits_1 = 1
tbits_100 = 1/50
tbits_1000 = 1/100
tbits_10000 = 1/200
tbits_20000 = 1/500
```

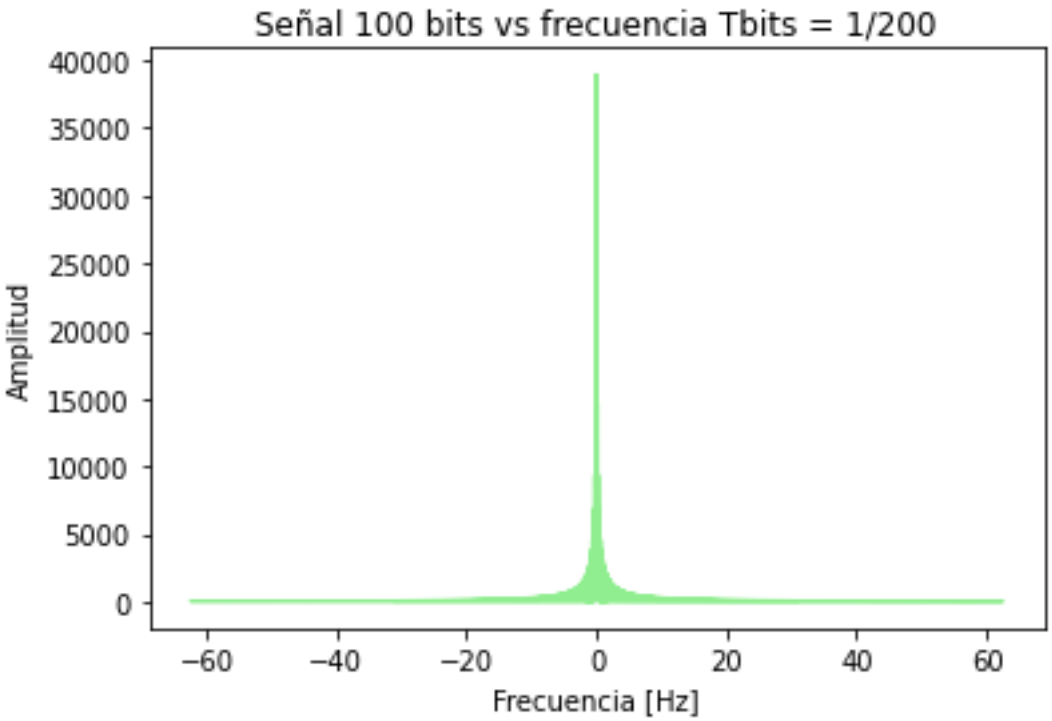
Luego de esto se realizaron los mismos procedimientos hechos en el item i y ii.

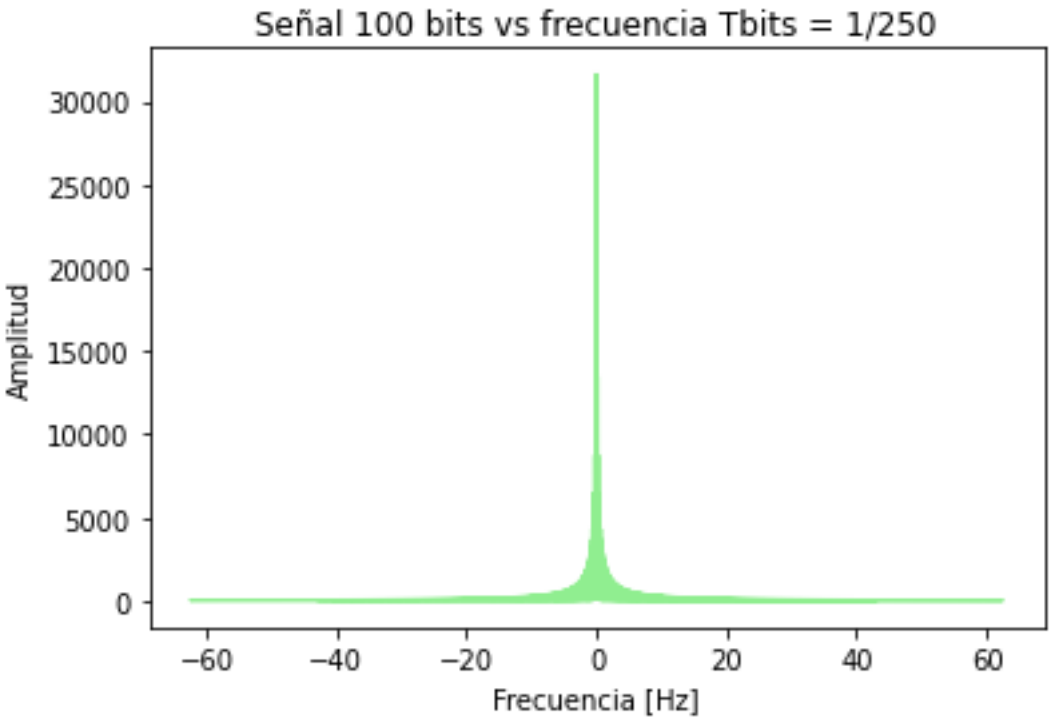
Y se obtuvieron los siguientes resultados:

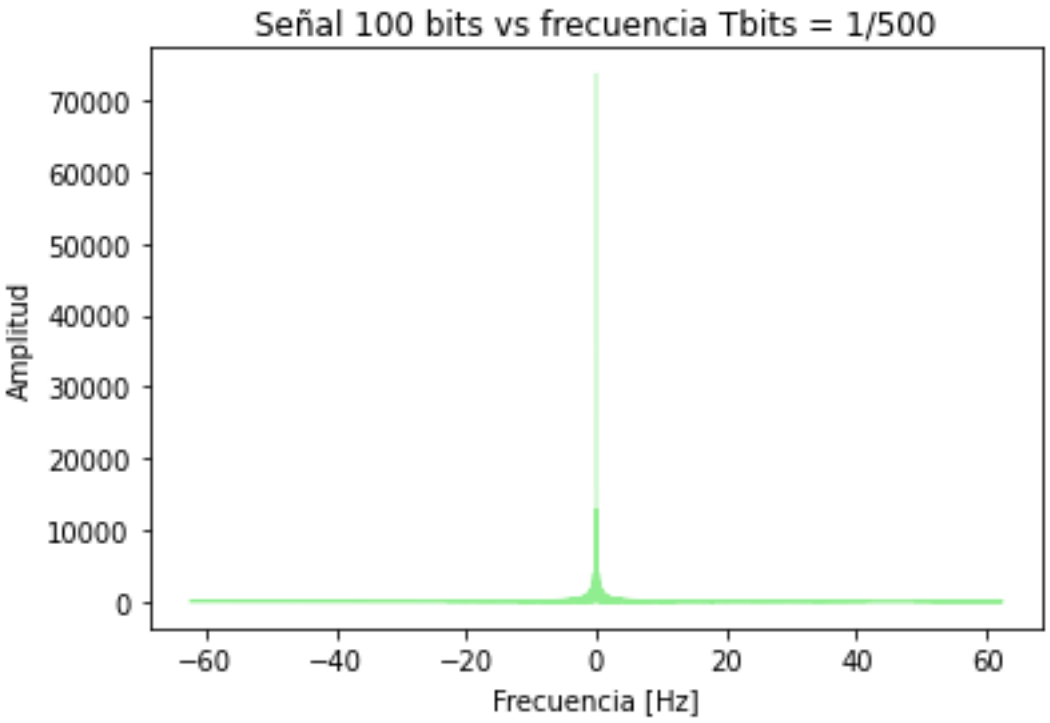



Señal 100 bits vs frecuencia Tbits = 1/50
25000
20000
15000
10000
5000
0
Amplitud
−60
−40
−20
0
20
40
60
Frecuencia [Hz]


Señal 100 bits vs frecuencia Tbits = 1/100
30000
25000
20000
15000
10000
5000
0
Amplitud
−60
−40
−20
0
20
40
60
Frecuencia [Hz]


Señal 100 bits vs frecuencia Tbits = 1/200
Amplitud
Frecuencia [Hz]
40000
35000
30000
25000
20000
15000
10000
5000
0
−60
−40
−20
0
20
40
60


Señal 100 bits vs frecuencia Tbits = 1/250
Amplitud
Frecuencia [Hz]
30000
25000
20000
15000
10000
5000
0
−60
−40
−20
0
20
40
60




Se puede ver como el ancho de banda se va estrechando cada vez que Tbits (la cantidad de bits en 1 segundo) va disminuyendo (Es decir, va aumentando la cantidad de bits). Pero no solo esto, si no que la amplitud se va haciendo cada vez mucho más grande que la anterior.

Se puede apreciar que una tasa cercana a la frecuencia de la portadora (120 bits /s  vs 125 Hz) se obtienen frecuencias muy concentradas en el origen. Se nota más cuando la tasa es el doble que la frecuencia de la portadora, es decir 250, o en este caso se puede ver que 200, 250 y 500 son gráficos que la concentración de las frecuencias esta muy concentrada en el 0.

1b.