**PROCESO DE TRANSMISIÓN Y RECEPCIÓN DE UNA COMUNICACIÓN DIGITAL**

***Profesor:*** *Roberto Carlos Hincapie Reyes*

***Programa:*** *Ingeniería en Telecomunicaciones*

***Materia:*** *Teoría de Comunicaciones*

***Elaborado por:*** *Juan Camilo Marín Valencia* ***ID:*** *000330042*

*Jerónimo Echeverri**Vásquez* ***ID:*** *000364629*

*Anderson Olarte* ***ID:***

*Tomas Saldarriaga Posos* ***ID:*** *000288325*

*Luis Eduardo Cahuana Lopez* ***ID:*** *000324966*

**UNIVERSIDAD PONTIFICIA BOLIVARIANA**

**INGENIERÍA DE TELECOMUNICACIONES**

**MEDELLÍN - 2020**

**Comunicación Digital**

Muchas de las señales que se desea comunicar están originalmente en formato analógico, un ejemplo claro es la voz. Como alternativa se puede pensar en digitalizar dichas la señales y transmitir los bits resultantes utilizando técnicas de codificación, corrección de errores luego modularlas. El concepto “comunicaciones digitales” suele implicar la idea de señales de voltaje que cambian entre dos niveles que representan el uno y el cero y que, aunque distorsionadas al ser transmitidas por el canal, pueden ser reconstruidas en el destino gracias a su carácter discreto.

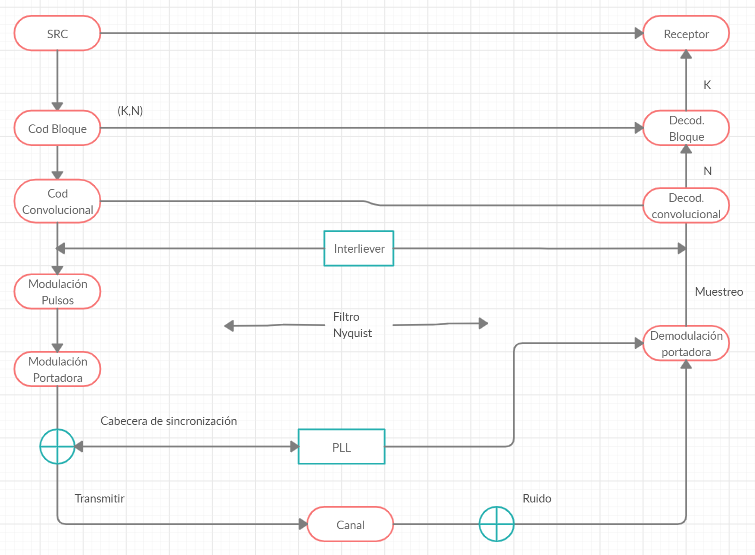


Ilustración 1 Proceso de la comunicación digital

## **Preparación de entorno**

Para este proceso se importará algunas librerías que permitirán una buena manipulación de la información de las señales y poderlas representar tanto numérica como gráficamente.

### **Librería clases:**

La librería creada por el profesor Roberto, será de gran ayuda a la creación de señales y obtener información importante sobre ella, como por ejemplo su espectro.

* **Soundfile:**

Permite leer y escribir archivos de sonido

* **Sounddevice:**

Contiene funciones convenientes para reproducir y grabar matrices Numpy que contienen señales de audio.

### **Numpy:**

Librería que permite trabajar con arreglos numéricos más fácilmente.

### **Scipy:**

Librería científica de Python que proporciona herramientas matemáticas muy útiles en el procesamiento de señales.

### **Mathplotlib:**

Librería que permite dar una representación gráfica a un conjunto de datos numéricos, para este caso, las señales.

***Importación de librerías:***

1. from clases import senal
2. import numpy as np
3. from scipy import fft
4. from scipy.fftpack import fftshift
5. import matplotlib.pyplot as plt

## **Clases**

En las clases se utilizará el código generado por el profesor Roberto Carlos Hincapie, el cual fue modificado con una clase llamada interliever.

Esta clase contiene dos métodos, el primero mezcla las posiciones de cada uno de los bits de la cadena a partir de un arreglo mapeador definido, este se utiliza en el transmisor para codificar. El segundo recupera las posiciones originales de los bits de las cadenas (decodificar) que luego será utilizado en el receptor para recuperar la señal.

**class** **interliever:**

**def** \_\_init\_\_**(**self**):**

self**.**lenPos **=** 100

self**.**mapeador **=** **[**70**,** 16**,** 11**,** 2**,** 57**,** 22**,** 7**,** 87**,** 74**,** 32**,** 96**,** 51**,** 75**,** 49**,** 86**,** 89**,** 23**,** 14**,** 38**,** 9**,** 3**,** 95**,** 82**,** 54**,** 85**,** 45**,** 94**,** 36**,** 66**,** 1**,** 40**,** 4**,** 33**,** 27**,** 81**,** 68**,** 44**,** 64**,** 20**,** 79**,** 8**,** 67**,** 28**,** 55**,** 78**,** 18**,** 5**,** 83**,** 92**,** 0**,** 39**,** 63**,** 15**,** 97**,** 41**,** 12**,** 47**,** 59**,** 76**,** 60**,** 43**,** 80**,** 21**,** 99**,** 91**,** 10**,** 90**,** 50**,** 98**,** 62**,** 88**,** 77**,** 56**,** 46**,** 52**,** 31**,** 53**,** 34**,** 58**,** 25**,** 93**,** 73**,** 37**,** 29**,** 35**,** 48**,** 24**,** 17**,** 13**,** 71**,** 61**,** 69**,** 84**,** 65**,** 30**,** 6**,** 19**,** 42**,** 72**,** 26**]**

**def** mezclar**(**self**,** mensaje **=** **[]):**

desorden **=** np**.**array**([])**

**if(len(**mensaje**)%**100 **!=** 0**):** #Se valida que la longitud del mensaje sea múltiplo de 100

falta **=** np**.**zeros**(**100**-(len(**mensaje**)%**100**))**

mensaje **=** np**.**concatenate**((**mensaje**,** falta**))**

**for** pedazo **in** **range(**0**,len(**mensaje**),**self**.**lenPos**):** #Se mezclan las posiciones del msg

seccion **=** np**.**array**(**mensaje**[**pedazo**:**pedazo**+**self**.**lenPos**])**

concatenador **=** **[]**

**for** pos **in** **range(len(**self**.**mapeador**)):**

concatenador**.**append**(**seccion**[**self**.**mapeador**[**pos**]])**

desorden **=** np**.**concatenate**((**desorden**,**concatenador**))**

**return** desorden

**def** reordenar**(**self**,** desorden **=** **[]):**

recepcion **=** **[]**

seccion **=** **[]**

**if(len(**desorden**)%**100 **!=** 0**):** #Se valida que la longitud del mensaje sea multiplo de 100

falta **=** np**.**zeros**(**100**-(len(**desorden**)%**100**))**

desorden **=** np**.**concatenate**((**desorden**,** falta**))**

**for** pedazo **in** **range(**0**,len(**desorden**),**self**.**lenPos**):** #Se recuperan las posiciones originales del msg

seccion **=**np**.**array**(**desorden**[**pedazo**:**pedazo**+**self**.**lenPos**])**

concatenador **=** np**.**zeros**(**self**.**lenPos**)**

**for** pos **in** **range(len(**self**.**mapeador**)):**

concatenador**[**self**.**mapeador**[**pos**]]** **=** **(**seccion**[**pos**])**

recepcion **=** np**.**concatenate**((**recepcion**,**concatenador**))**

**return** recepción

## **Transmisor**

## Se creó una cadena que contiene el mensaje “hola mundo, mi primer señal”, para luego codificarla en bits según el código ASCII.

cad**=**'Hola mundo, mi primer señal'

cod**=**codificador**()**

bits**=**cod**.**cad2Bin**(**cad**)**

## Se declaró el arreglo para aplicar la codificación convolucional sobre los bits del mensaje, en este caso se hará este proceso en cascada, es decir, se hará dos veces.

cc **=** convCode**(**G**=** np**.**array**([[**1**,**0**,**0**],[**1**,**1**,**0**],[**1**,**1**,**1**],[**1**,**0**,**1**]]))**

bits **=** cc**.**codificar**(**bits**)**

bits **=** cc**.**codificar**(**bits**)**

## Luego se ejecuta la función del *interleaver* sobre el arreglo de bits resultantes en el paso anterior para mezclar sus posiciones, esto sirve para evitar errores en ráfaga. Lo siguiente es separar la señal una parte real y la parte imaginaria.

interliever **=** interliever**()**

bits **=** interliever**.**mezclar**(**bits**)**

bitsR**,** bitsI**=**cod**.**separarRI**(**bits**)**

## Se le agrega una secuencia de sincronización al inicio de la parte real, esto con la finalidad de que cuando el receptor reciba la señal éste le indique en que punto comienza la información del mensaje para luego tomar las muestras y proceder a la decodificación, posteriormente se codifica toda la información a polar.

## La señal será modulada en cuadratura por lo que se requiere llevar la información de la portadora para que el receptor pueda demodularla, para ello se agrega una cierta cantidad de unos a la parte real, de modo tal, que al ser multiplicado por la portadora tendrá un nivel de directa con la información de ésta. Para no alterar las longitudes de los arreglos también se le agrega una cierta cantidad de ceros al inicio de la parte imaginaria para que tenga la misma longitud de la parte real. En seguida, se modulan ambas partes por pulsos, y como característica principal la señal debe cumplir el filtro de Nyquist. Por último, se modula en cuadratura la parte real por una función coseno y la imaginaria por seno y generamos la señal a transmitir.

## Como se puede deducir la parte real posee la información necesaria para la demodulación del mensaje.

fc**=**2000

fs**=**44100

Lport**=int(**500**\***fs**/**fc**)**

xR**=**np**.**concatenate**((**cod**.**seq1**,**bitsR**))**

xR**=**codificador**.**bits2Polar**(**xR**)**

xI**=**codificador**.**bits2Polar**(**bitsI**)**

xI**=**np**.**concatenate**((**np**.**zeros\_like**(**cod**.**seq1**),**xI**))**

plt**.**close**(**'all'**)**

K**=**100

B**=**fs**/**K

plt**.**close**(**'all'**)**

xnR**=**senal**(**x**=**xR**,** fs**=**B**,** nombre**=**'Secuencia a transmitir parte Real'**)**

xnR**.**modularPulsos**(**K**=**K**,** tipo**=**'RC'**,** graficar**=True,** alfa**=**0.3**,** num**=**10**)**

xnR**.**x**=**xnR**.**x**/**np**.max(**np**.abs(**xnR**.**x**))**

xnR**.**x**=**np**.**concatenate**((**np**.**ones**(**Lport**),** xnR**.**x**))**

xnR**.**dibujar**()**

xnI**=**senal**(**x**=**xI**,** fs**=**B**,** nombre**=**'Secuencia a transmitir parte Imaginaria'**)**

xnI**.**modularPulsos**(**K**=**K**,** tipo**=**'RC'**,** graficar**=True,** alfa**=**0.3**,** num**=**10**)**

xnI**.**x**=**xnI**.**x**/**np**.max(**np**.abs(**xnI**.**x**))**

xnI**.**x**=**np**.**concatenate**((**np**.**zeros**(**Lport**),** xnI**.**x**))**

xnI**.**dibujar**()**

y**=**senal**.**modularAMCuadratura**(**fc**=**fc**,** xnR**=**xnR**,** xnI**=**xnI**)**

y**.**x**=**y**.**x**/**np**.max(**np**.abs(**y**.**x**))**

y**.**dibujar**()**

Guardamos el sonido generado con la cadena, llamado salida.wav.

sf**.**write**(**'salida.wav'**,** y**.**x**,** **int(**y**.**fs**))**

***Parte real:***

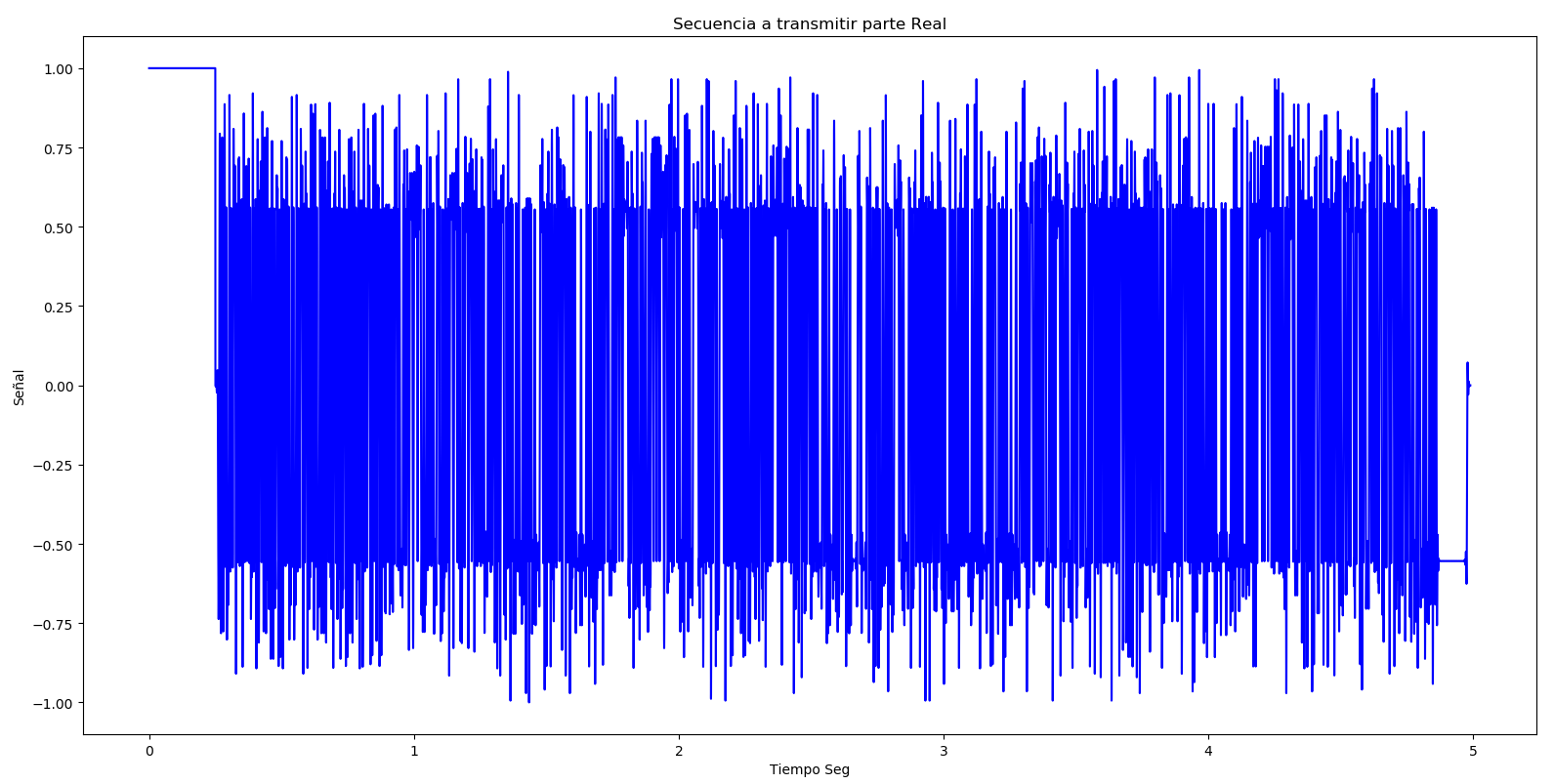


Ilustración 2 Parte real

***Parte imaginaria:***

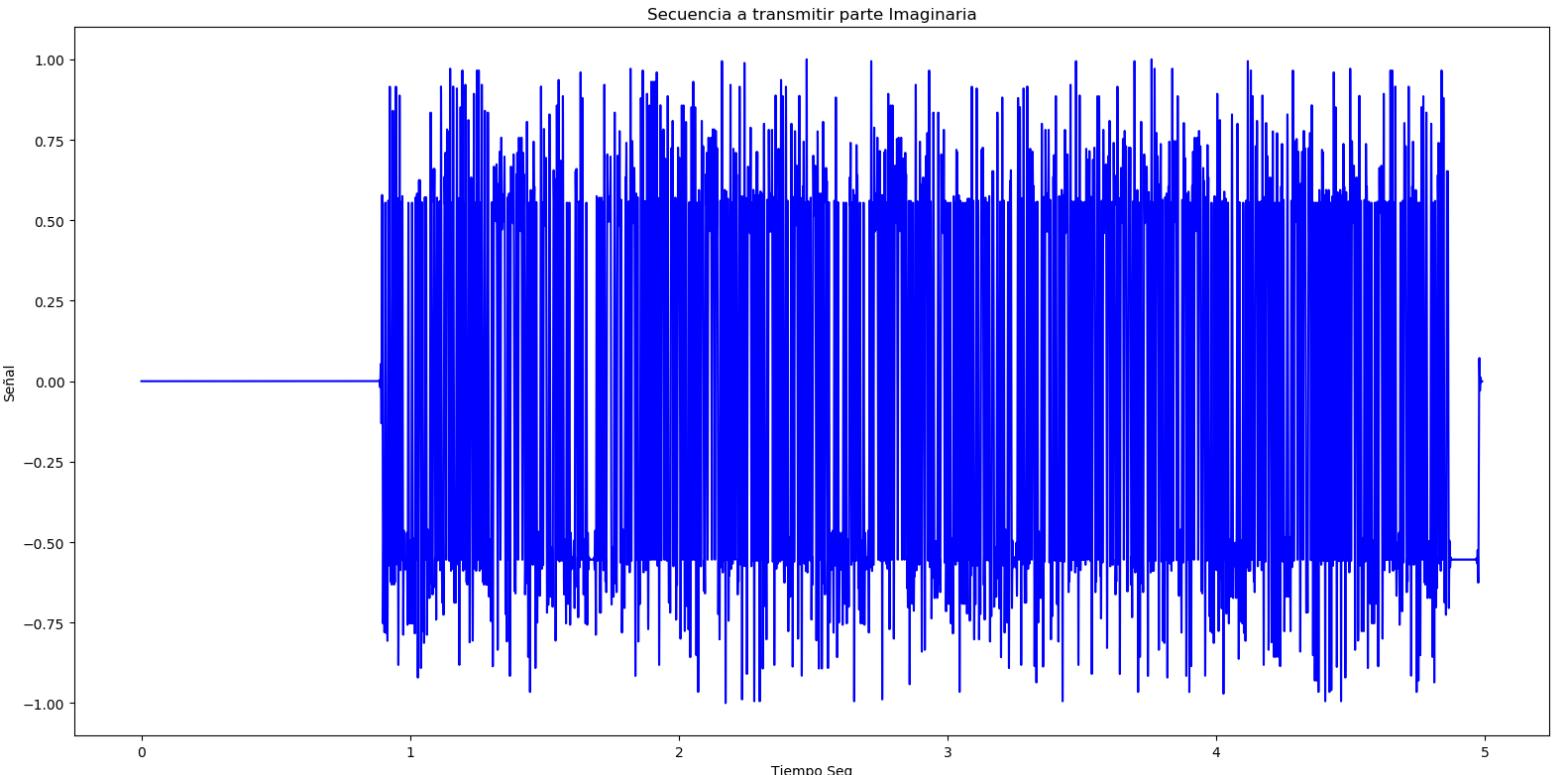


Ilustración 3 Parte imaginaria

***Gráfica de la señal modulada en cuadratura:***

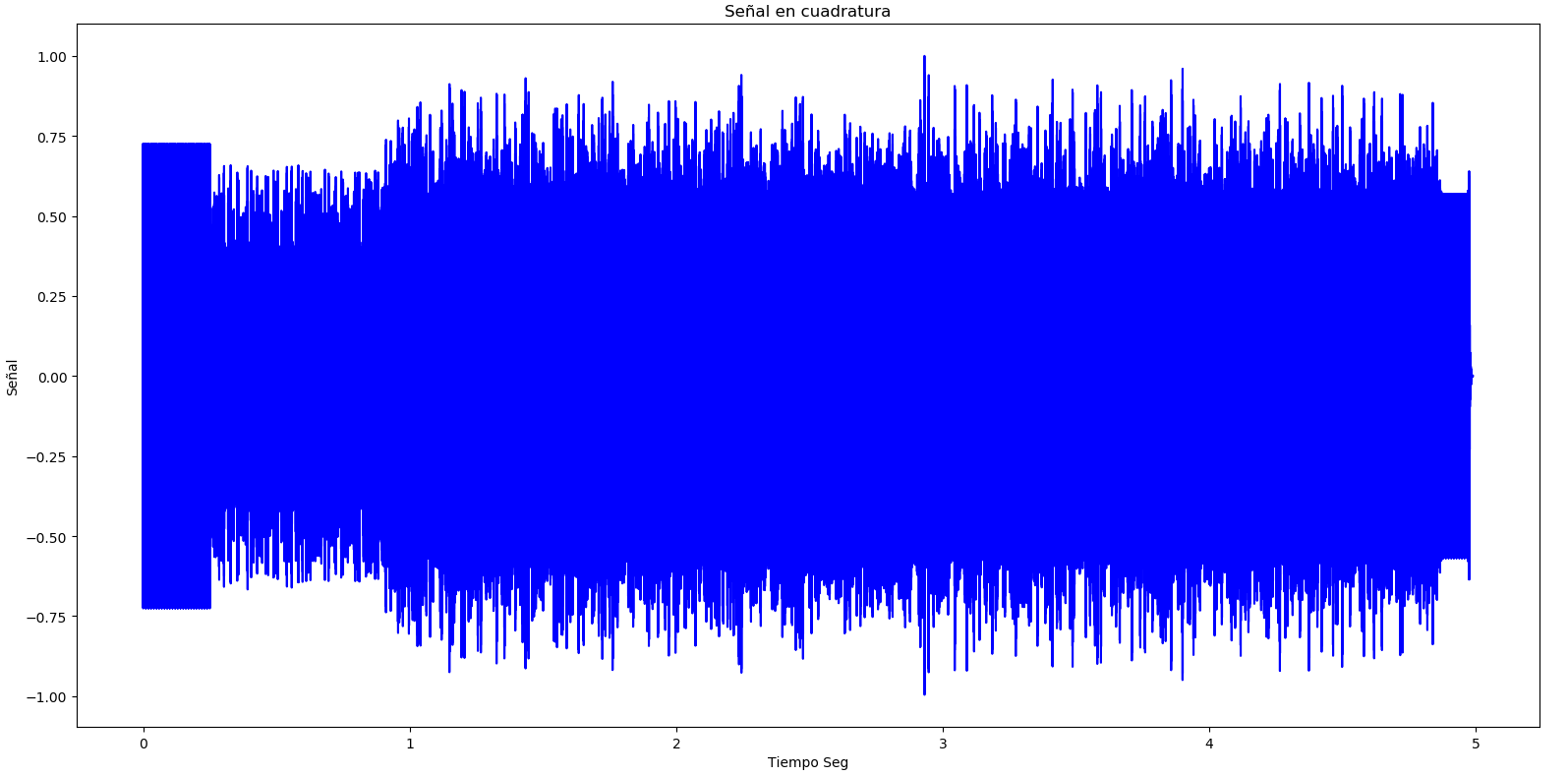


Ilustración 4 Gráfica de Señal Transmitida

***Espectro de la señal transmitida:***

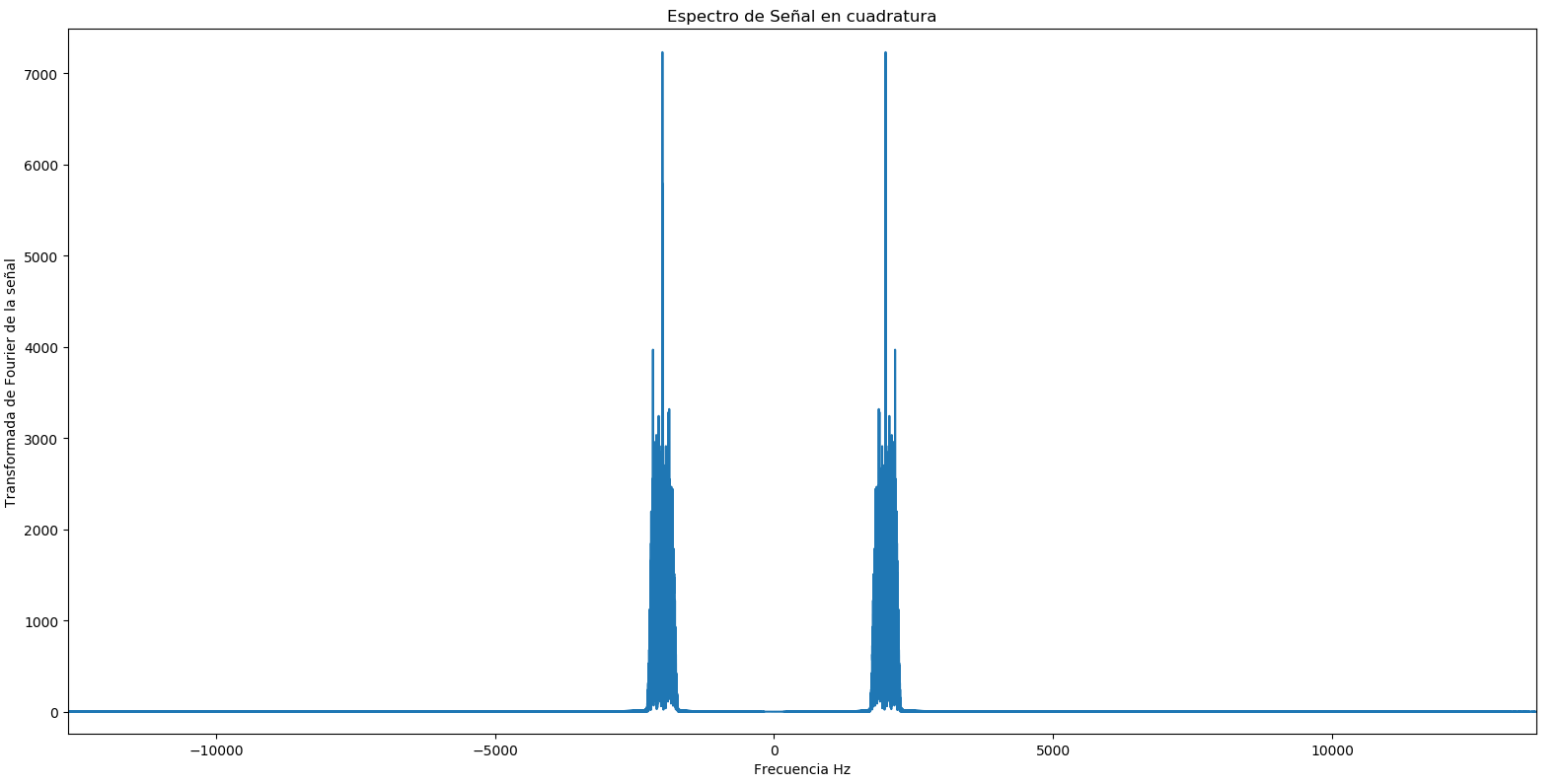


Ilustración 5 Espectro de la señal transmitida

1. **Receptor**

Se importa las librerías necesarias:

* **Correlate:**

Realiza la correlación cruzada entre dos matrices N-dimensionales.

**import** numpy **as** np

**import** matplotlib**.**pyplot **as** plt

**from** clases **import** senal**,** codificador**,** blockCode**,** convCode**,** interliever

**import** soundfile **as** sf

**import** sounddevice **as** sd

**from** scipy**.**signal **import** correlate

plt**.**close**(**'all'**)**

Para el proceso de la recepción de la señal se realizó una simulación grabando el audio generado en el transmisor en un ambiente con música de fondo que simule a un ruido en el canal.

Se carga el audio guardado con el nombre “prueba.wav” guardándolo como una instancia de clase señal.

y**=**senal**.**cargar\_sonido**(**archivo**=**'prueba.wav'**,** nombre**=**'señal recibida'**)**

y**.**x**=**np**.**concatenate**((**np**.**zeros**(**2000**),** y**.**x**))**

#y.awgn(5)

y**.**dibujar**()**

***Grafica de la señal recibida en el receptor:***

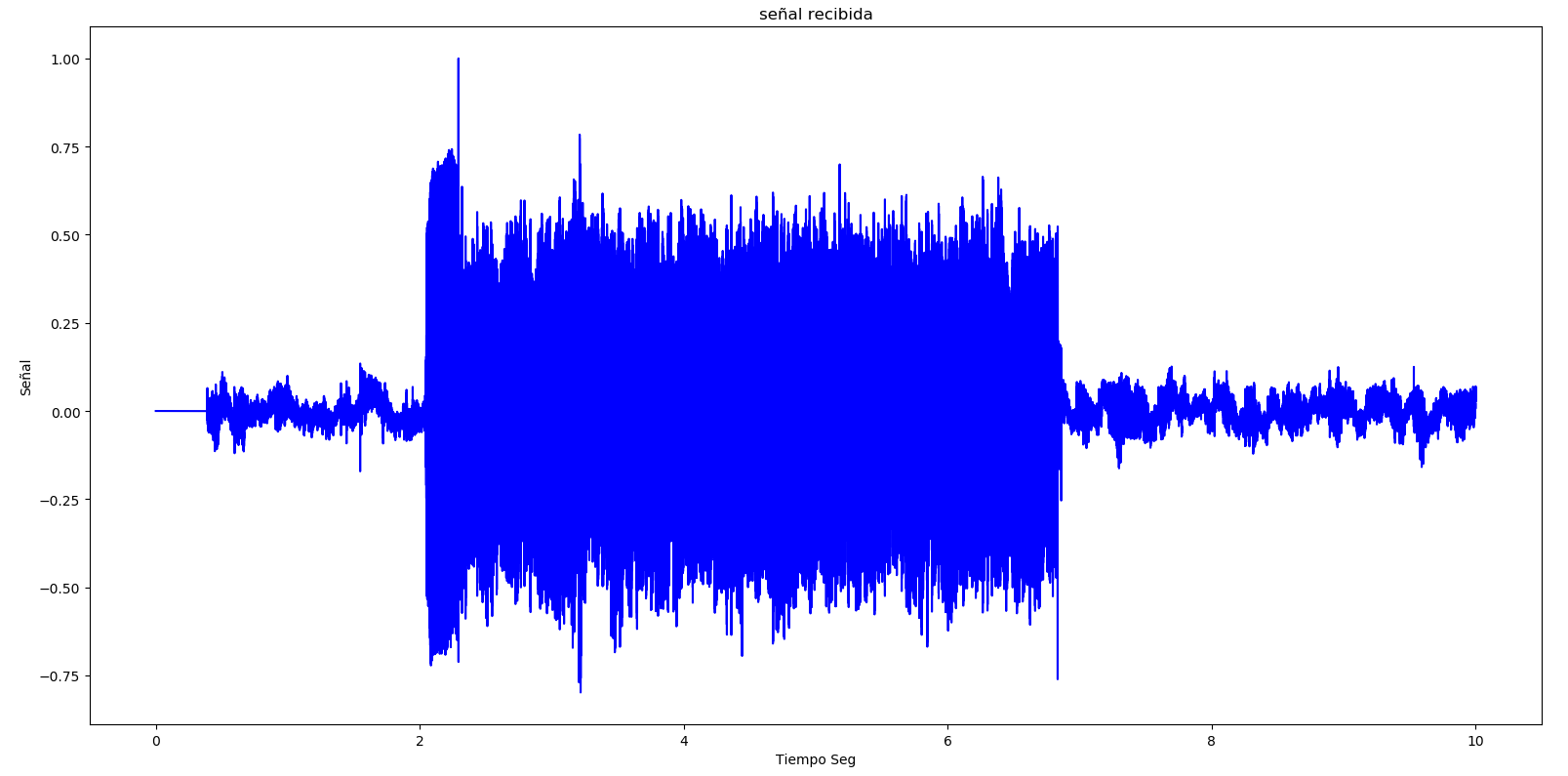


Ilustración 6 Gráfica de la señal recibida

***Detección de la portadora:***

Se procede a identificar la portadora en la gráfica de la señal y la extraemos.

tmin**=**2.13

Yport**=**y**.**recortar**(**tmin**=**tmin**,** tmax**=**2.28**)**

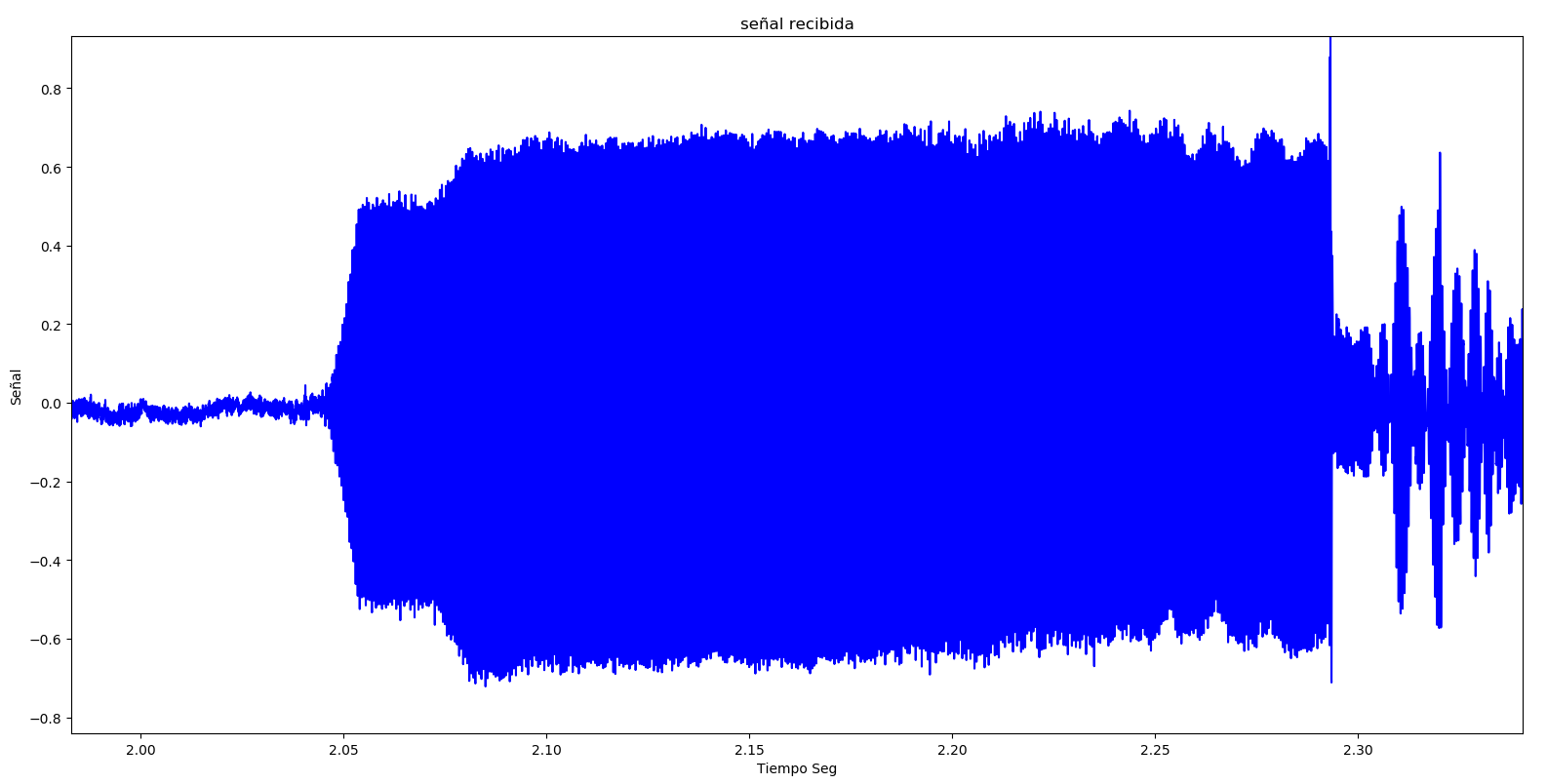


Ilustración 7 Portadora detectada de la señal recibida

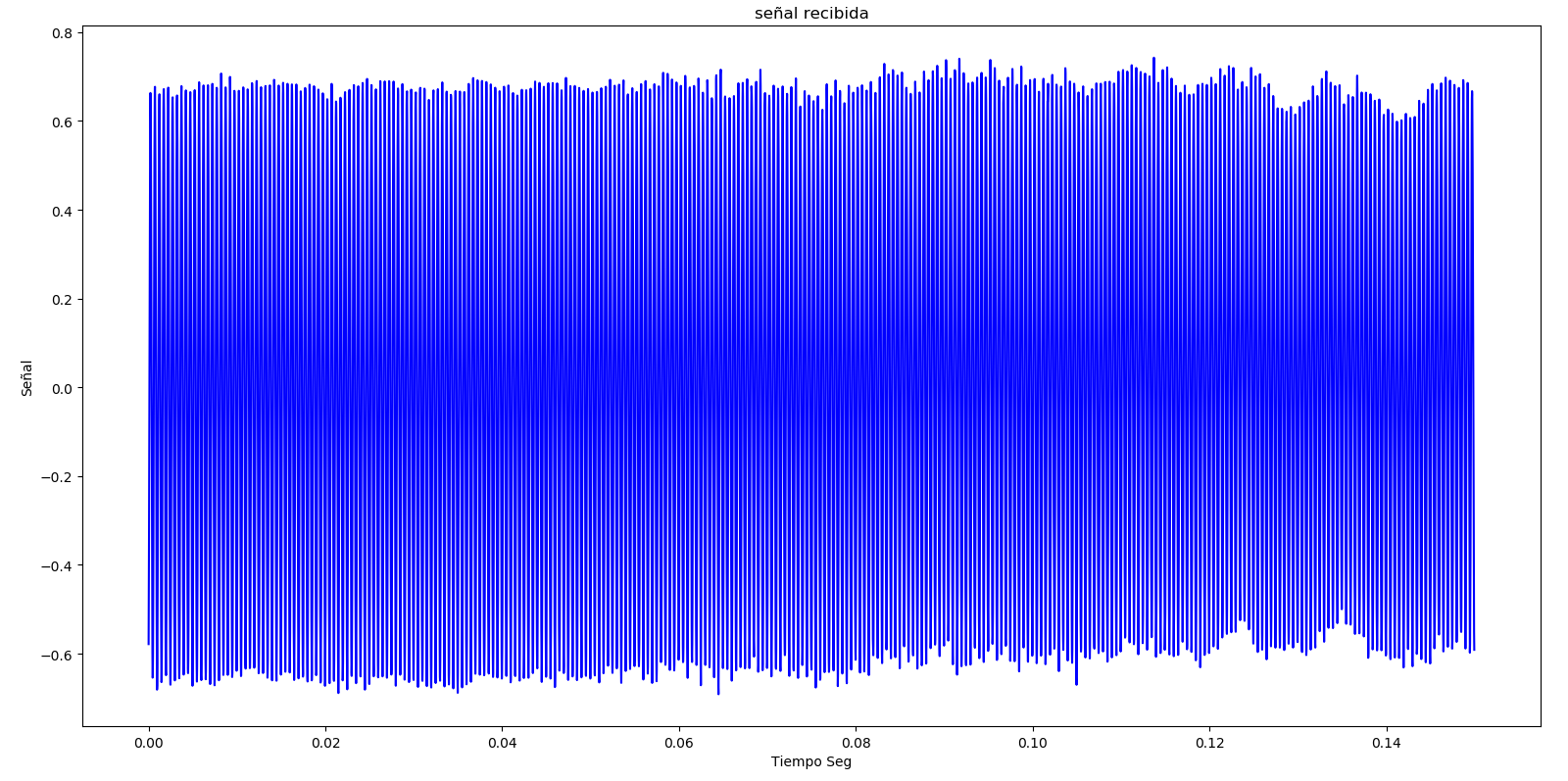


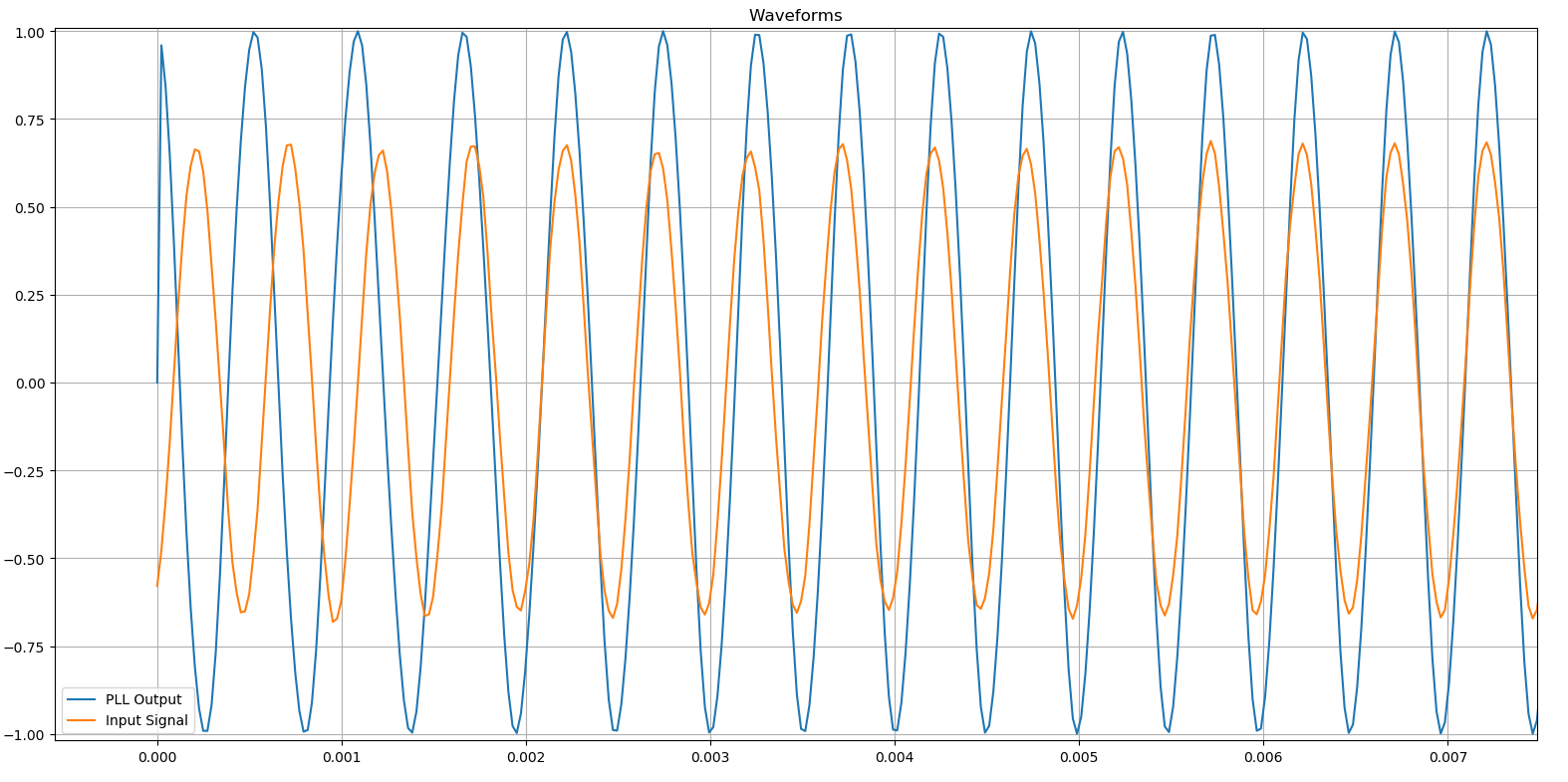
Ilustración 8 Portadora extraída

***PLL:***

Luego de extraer la portadora se le aplica a ésta el proceso del PLL. Ya que, en esta instancia no se sabe cuál fue la frecuencia de la portadora en el transmisor, el PLL tiene configurado una portadora por defecto, pero no necesariamente con los mismos valores de frecuencia que la portadora del transmisor y como casi siempre hay un desfase entre la portadora del PLL con la portadora extraída en este proceso se hallará ese desfase para luego agregarle a la portadora del receptor y tener ambos portadores sincronizados para así poder demodular la señal.

port**=**senal**.**PLL**(**y**=**Yport**.**x**,** fs**=**Yport**.**fs**,** fc**=**2000 **,** dibujar**=True)**

ph**=**senal**(**x**=**port**.**phase**,** fs**=**port**.**fs**,** nombre**=**'Fase'**)**



***Fase en función al tiempo:***

En este punto gracias al PLL se puede crear una función de la fase con respecto al tiempo y agregarle a la portadora del receptor. Para dicha función se requiere hallar dos puntos de la gráfica del mediamóvil y luego con el código crear una función fase para cada instante de tiempo de la señal esta función tiene una gran similitud con la de una recta con una pendiente e intercepto.

ph2**=**ph**.**mediaMovil**(**L**=**100**,** dibujar**=True)**

ta**=**0.01

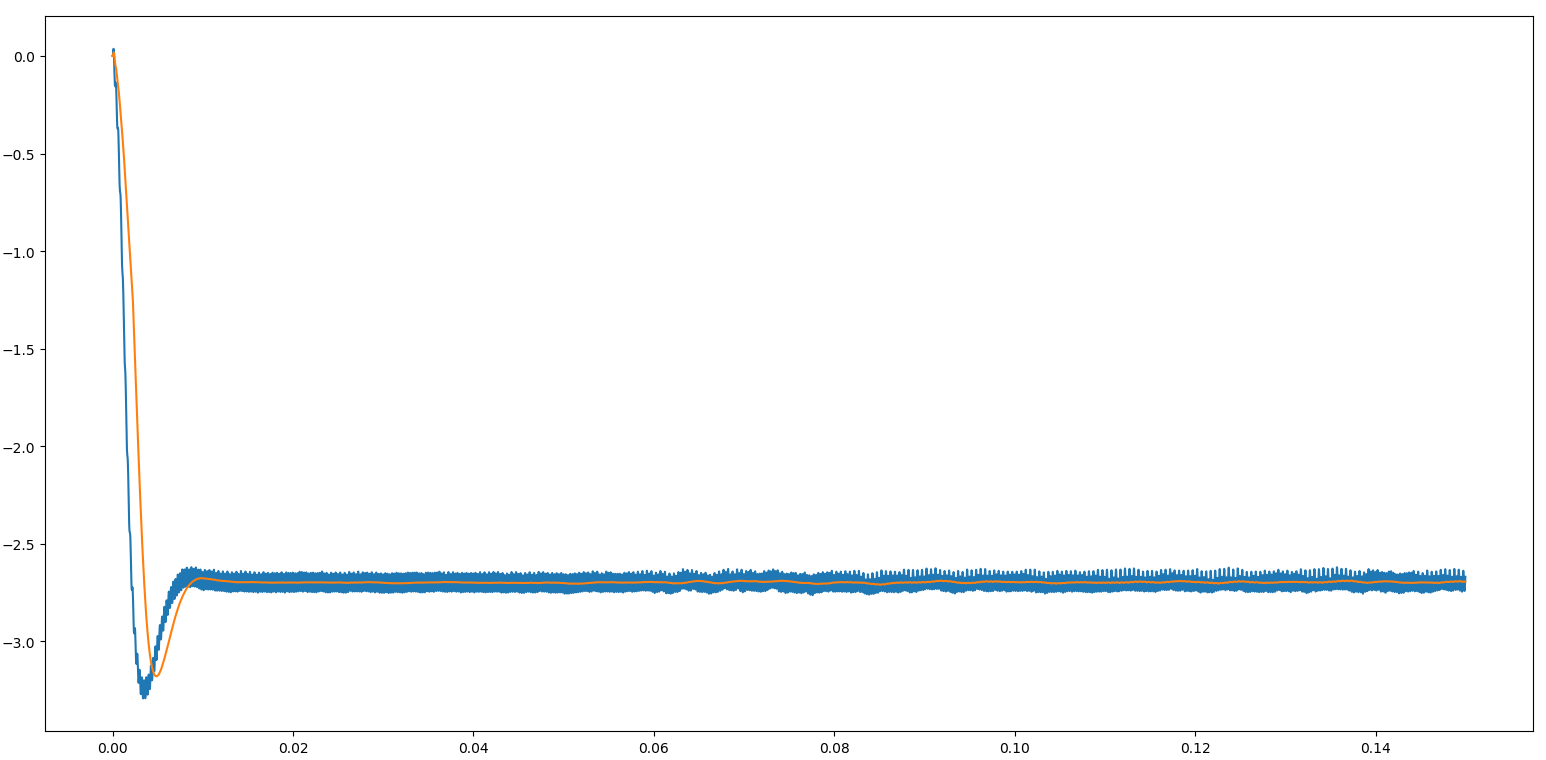
tb**=**0.149

pha**=**ph2**.**getX**(**ta**)**

phb**=**ph2**.**getX**(**tb**)**

m**=(**phb**-**pha**)/(**tb**-**ta**)**

b**=**pha**-**m**\*(**ta**+**tmin**)**



Se extrae el arreglo de tiempo t de la señal recibida y se crea las portadoras agregándole la función fase con pendiente m e intercepto b hallados en el paso anterior, además de un parámetro t de tiempo con lo que se tendría dos portadoras una seno y otra coseno, ambas en fase con las portadora del receptor, ya que, se le está adicionando un valor de diferencia de fase con el objetivo de sincronizarlas.

t**=**y**.**getT**()**

portadora **=** np**.**cos**(**2**\***np**.**pi**\***2000**\***t**+(**m**\***t**+**b**))**

portadoraI**=-**np**.**sin**(**2**\***np**.**pi**\***2000**\***t**+(**m**\***t**+**b**))**

Como la señal está modulada en cuadratura se le multiplica por las portadoras creadas para demodularlas, es decir, ponerlas en banda base, pero como aún queda parte de la señal en altas frecuencias se le aplica un filtro pasa bajo y con esto se tendría la señal demodulada en banda base.

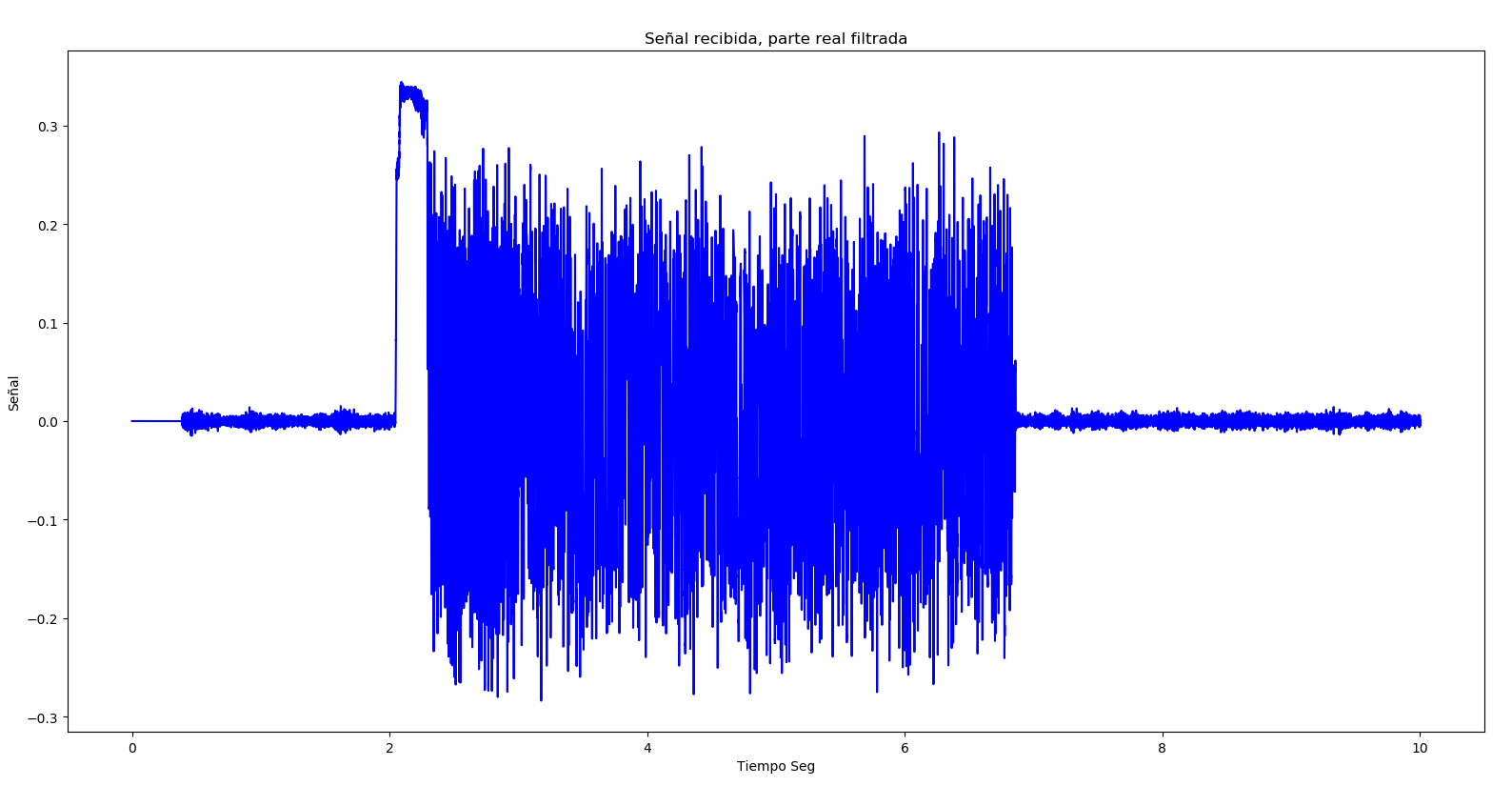
#Recuperación de las dos partes de la señal en caso de que haya cuadratura

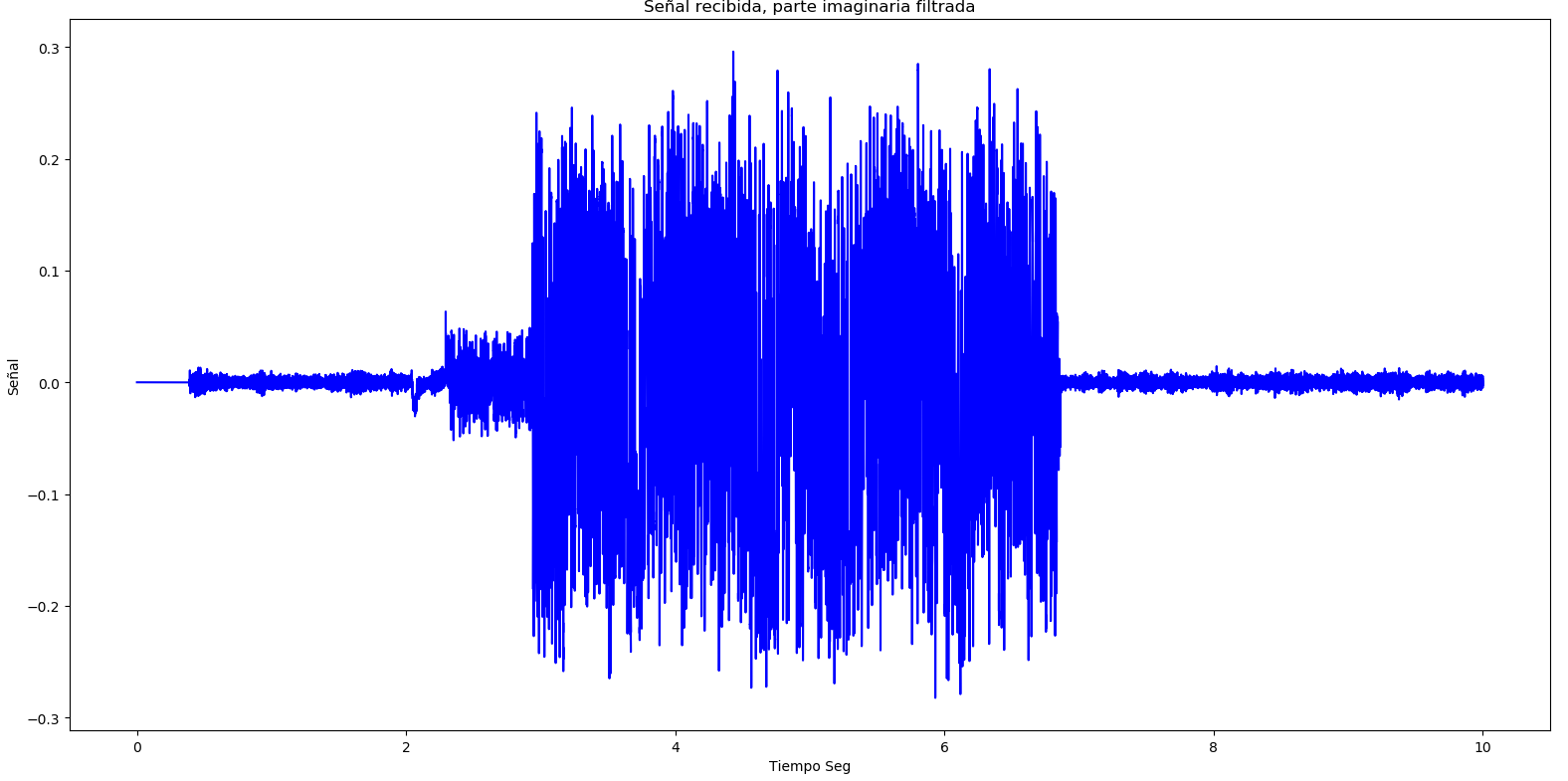
yR**=**senal**(**x**=**y**.**x**\***portadora**,** fs**=**y**.**fs**,** nombre**=**'Señal recibida, parte real'**)**

yI**=**senal**(**x**=**y**.**x**\***portadoraI**,** fs**=**y**.**fs**,** nombre**=**'Señal recibida, parte imaginaria'**)**

yR**=**yR**.**filtrar\_pasabajo**(**fc**=**1000**)**

yI**=**yI**.**filtrar\_pasabajo**(**fc**=**1000**)**





**Búsqueda de la información del mensaje:**

Teniendo la información de la señal, el siguiente paso es detectar la secuencia de sincronización que indica en que parte de toda la señal empieza la información del mensaje e inmediatamente proceder a muestrear a partir de dicha secuencia.

Para este paso tanto el transmisor como el receptor deben conocer la secuencia pseudoaleatoria de sincronización.

Se importa la secuencia de la biblioteca clases y posteriormente se realiza la correlación entre la señal y la secuencia. Es necesario hacer este paso con la parte real de la señal demodulada, ya que, es en esta parte de la señal donde se agregó dicha secuencia.

Se hace la modulación por pulsos a la secuencia de sincronización para tomar las muestras por símbolo de cada parte de la señal recibida.

#Busqueda del inicio de las muestras

cod**=**codificador**()**

fc**=**2000

fs**=**44100

x**=**cod**.**seq1

x**=**codificador**.**bits2Polar**(**x**)**

K**=**100

B**=**44100**/**K

xn**=**senal**(**x**=**x**,** fs**=**B**,** nombre**=**'Secuencia a transmitir'**)**

xn**.**modularPulsos**(**K**=**K**,** tipo**=**'RC'**,** graficar**=True,** alfa**=**0.3**,** num**=**10**)** #Tiene que ser igual al transmisor

xn**.**x**=**xn**.**x**/**np**.max(**np**.abs(**xn**.**x**))**

xn**.**dibujar**()**

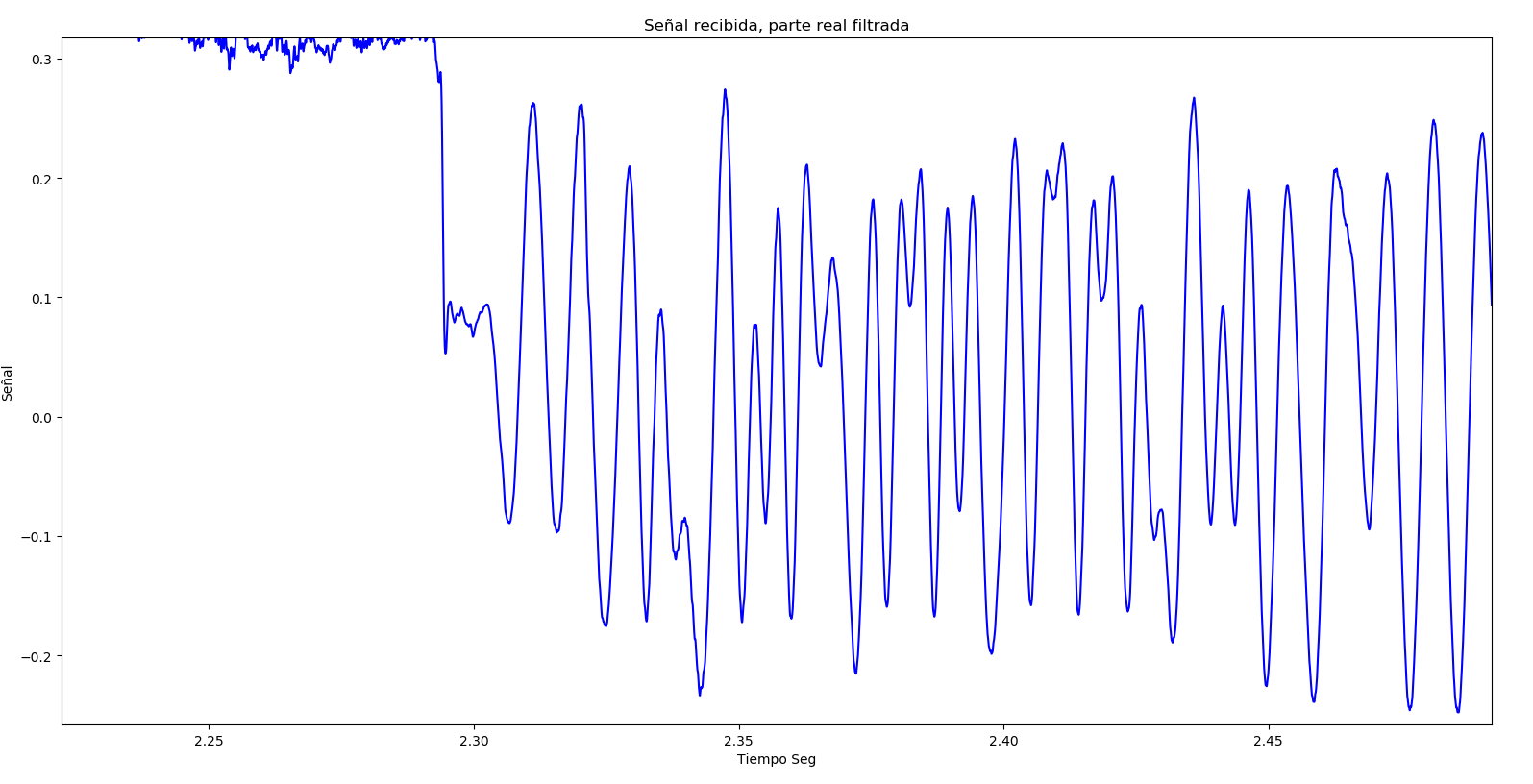
co**=**correlate**(**yR**.**x**,** xn**.**x**,** mode**=**'valid'**)**

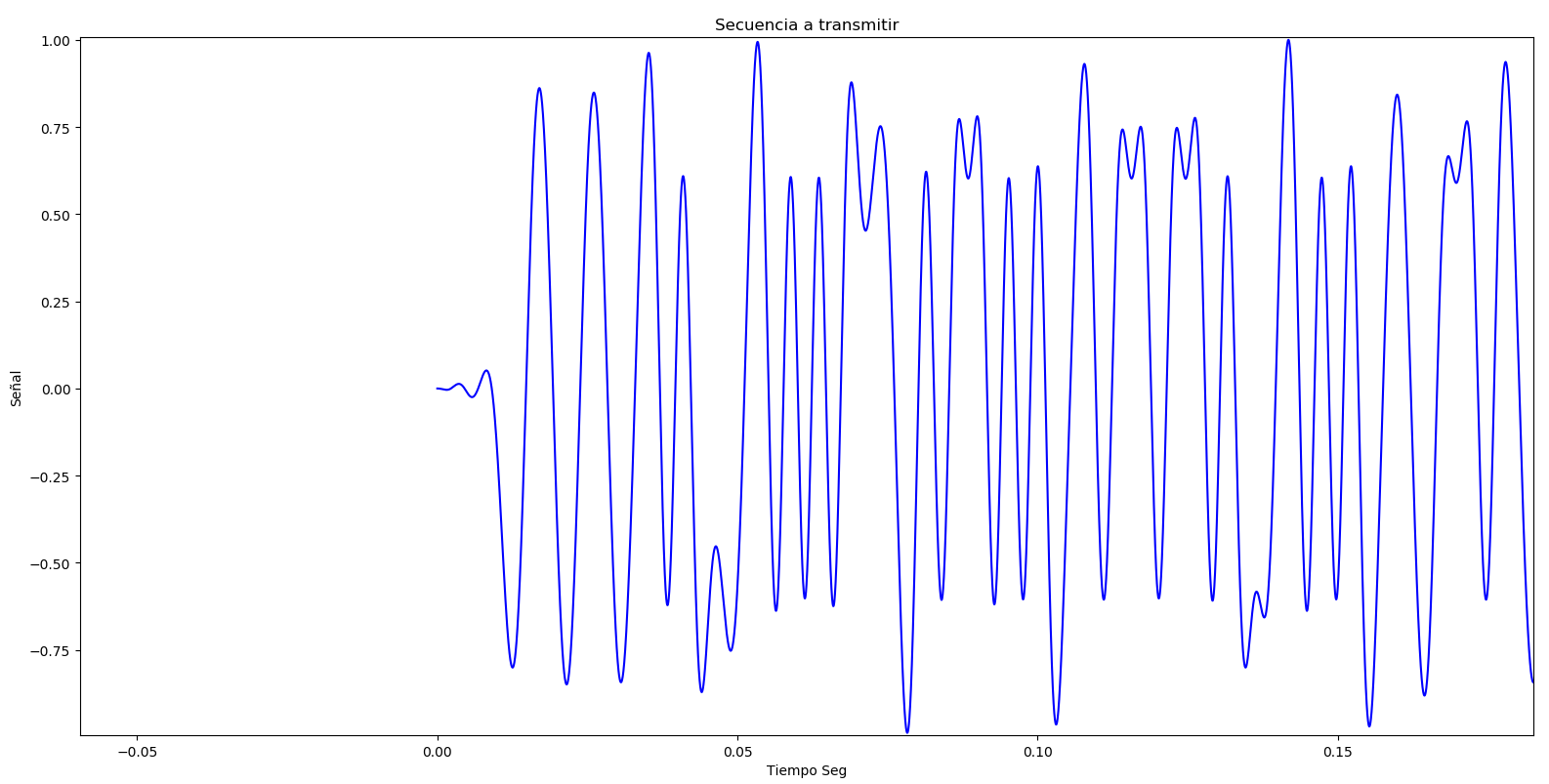
ind**=**np**.**argmax**(**co**)**

plt**.**figure**()**

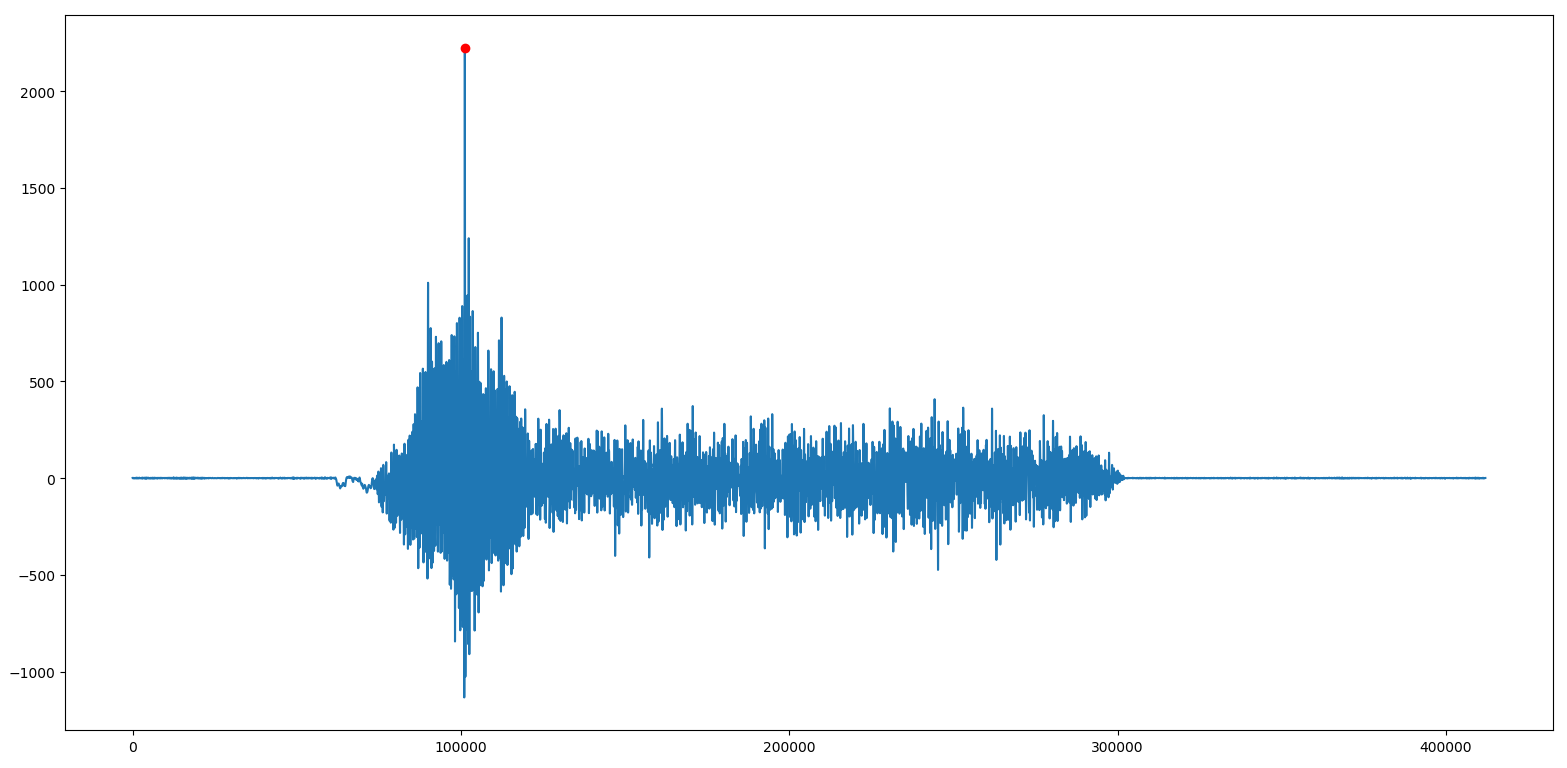
plt**.**plot**(**co**)**

plt**.**plot**(**ind**,**co**[**ind**],**'or'**)**





* ***Correlación entre la secuencia de sincronización y la parte de real de la señal:***



***Índice donde comienza la secuencia de transmisión:***



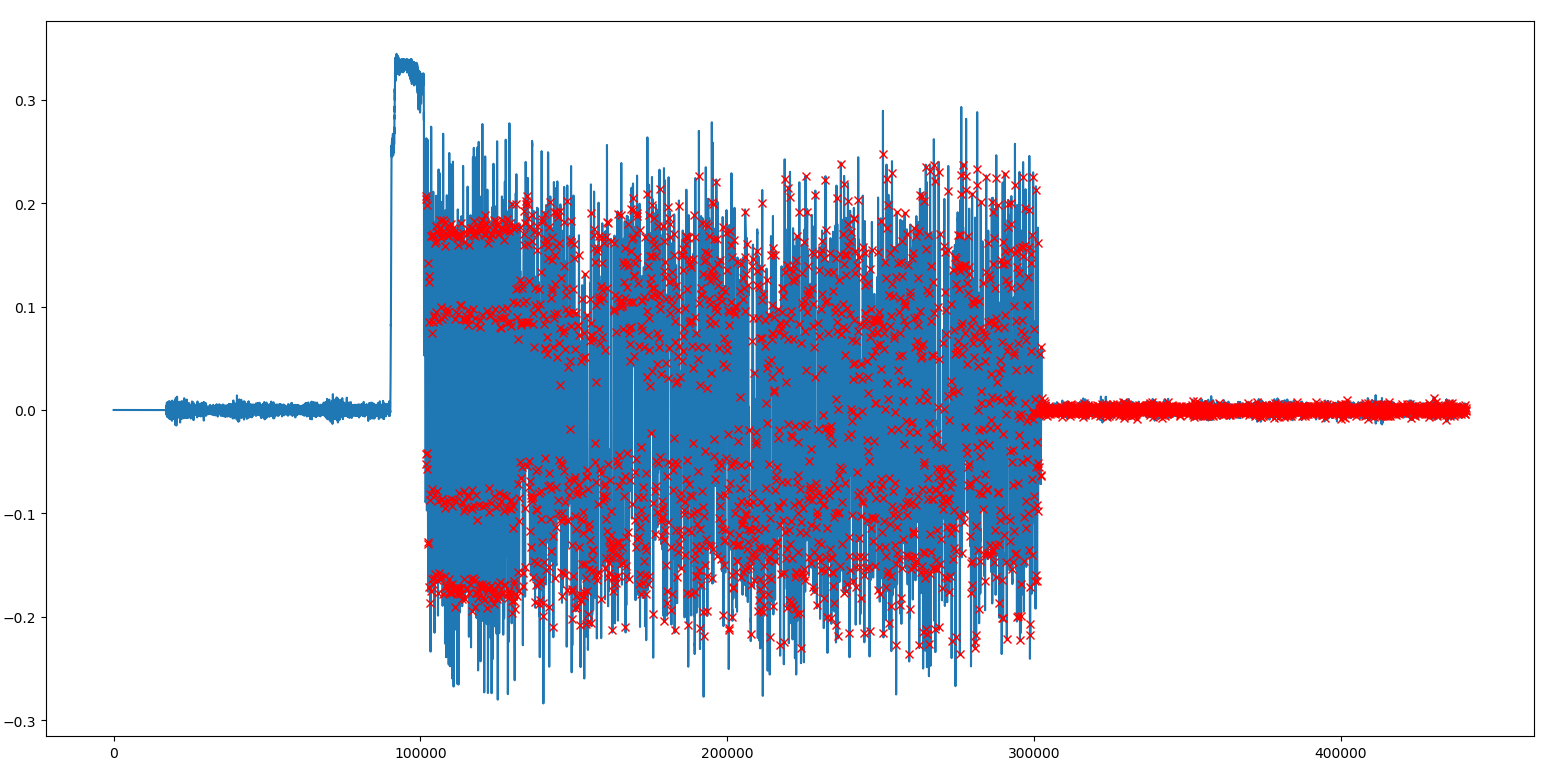
Muestreo de la señal para obtener solo muestras del mensaje en este caso se toma 1 muestras de cada 100 iniciando la primera muestra en la posición 500 del arreglo de cada una de las partes de la señal demodulada.

samples**=**np**.**arange**(**ind**+**xn**.**lo**,** **len(**y**.**x**),** K**)**

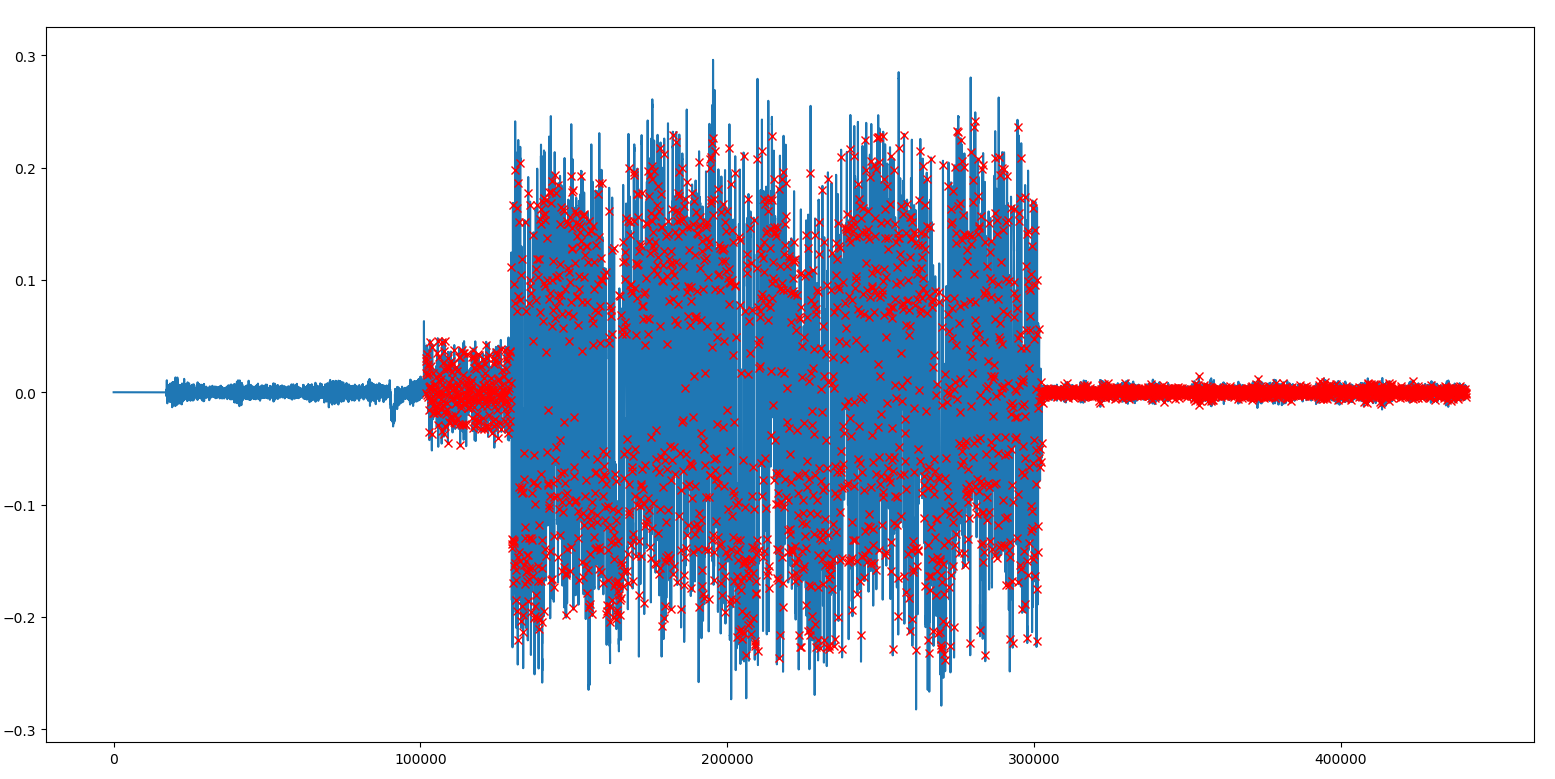
ynR**=**np**.**array**([**yR**.**x**[int(**i**)]** **for** i **in** samples**])**

ynI**=**np**.**array**([**yI**.**x**[int(**i**)]** **for** i **in** samples**])**

* ***Muestras para la parte de real de la señal recibida:***



* ***Muestras para la parte de imaginaria de la señal recibida:***



***Proceso de discriminación:***

En este proceso se analiza cada valor de los elementos de los arreglos si es mayor a 0 entonces se convierte a 1 sino -1.

ynR**[**ynR**>**0**]=**1

ynR**[**ynR**<**0**]=-**1

ynR**=(**ynR**+**1**)/**2

ynR**=**ynR**[**280**:]**

ynI**[**ynI**>**0**]=**1

ynI**[**ynI**<**0**]=-**1

ynI**=(**ynI**+**1**)/**2

ynI**=**ynI**[**280**:]**

***Decodificación de los datos:***

La función agregar añade cada elemento de los arreglos de tal forma intercalada, es decir, el primer elemento de la parte real y luego el primer elemento de la parte imaginaria luego el segundo de la parte real y en seguida el segundo de la parte imaginaria, etc.

El método del interliever.reordenar organiza los elementos del arreglo según sus posiciones originales en el transmisor.

Se realiza la decodificación convolucional en cascada con las mismos arreglos del transmisor.

Finalmente se codifica cada 8 datos del arreglo para generar un byte y convertirlos a la codificación en ASCII para que devuelva la cadena o mensaje contenida en la señal.

yn**=**cod**.**agregarRI**(**ynR**,** ynI**)**

interliever **=** interliever**()**

yn**=**interliever**.**reordenar**(**yn**)**

cc **=** convCode**(**G**=** np**.**array**([[**1**,**0**,**0**],[**1**,**1**,**0**],[**1**,**1**,**1**],[**1**,**0**,**1**]]))**

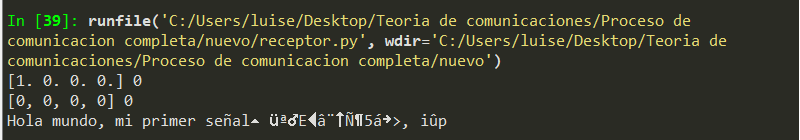
yn **=** cc**.**decodificar**(**yn**)**

yn **=** cc**.**decodificar**(**yn**)**

cad**=**cod**.**bits2Cad**(**yn**)**

**print(**cad**)**

La cadena recuperada es:



Al realizar este ejercicio se llegaron a varias conclusiones:

* Se detecto que al utilizar códigos de bloque se notaba el deterioro de la señal que llega.
* Se evidencio que al utilizar codificación en cascada se ve una gran mejoría en la recuperación de la señal, pero al utilizar por 3ra vez la codificación en cascada, nuevamente se deteriora.
* Al utilizar un ruido ***awgn*** aleatorio se vio que mientras más corta era la cadena más fácilmente se recuperaba de forma correcta.