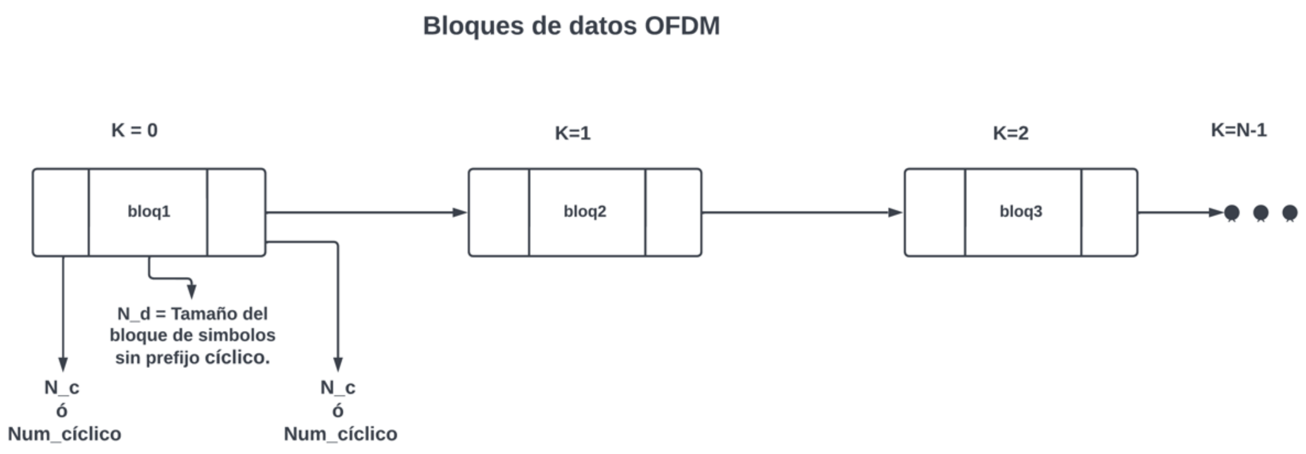
Multiplexación por división de frecuencias ortogonales -OFDM-



Librerias

```
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats as scs
import random
from scipy.stats import chi2
from math import sqrt
import math
import sympy as sp
```

Parámetros comunes para ciclo estacionario y para detector de energia

$\sigma_s = 1$

$P_{FA} = 0.05$

$NUM\_STATISTICS = 10$ Número de veces que se corre las pruebas mientras mayor sea mejor es el resutlado.

$SNR = 100$

$SNR\_LOW = -30$

$SNR\_UP = 0$

$SNR\_STEP = 2$

$Intervalo[-30, -28, -26.....0]$

$K = 50$ Numero de bloques

$Num\_ciclico = 8$ Numero de prefijo cíclico. (N_d)

$N\_d = 32$ Tamaño de un bloque sin prefijo cíclico.

```
# PARAMETERS
sigma_s = 1           # desviación de la señal primaria
P_FA = 0.05           # probabilidad de falso positivo deseada
#num_samples = 10    # número de muestras
NUM_STATISTICS = 200    # cantidad de veces que se repite el test

##############
#SNR = 10
#SNR_LOW = -30
#SNR_UP = 4
#SNR_STEP = 2

SNR_LOW = -20
SNR_UP = 6
SNR_STEP = 2

#SNR_NOISE = 1
########################
K = 50 #Numero de bloques
Num_ciclico = 8 # Numero de prefijo ciclico
N_d = 32        # Tamaño del bloque sin el prefijo ciclico
```

```

N = (K + 1) * (Num_ciclico + N_d) # Tamaño de tods los bloques

SNR_list = np.arange(SNR_LOW, SNR_UP, SNR_STEP)
P_D_CALC_CE = np.zeros(len(SNR_list))
P_FA_THEO_CE = np.zeros(len(SNR_list))
P_FA_CALC_CE = np.zeros(len(SNR_list))


P_M_CALC_DE = np.zeros(len(SNR_list))
P_D_CALC_DE = np.zeros(len(SNR_list))
```

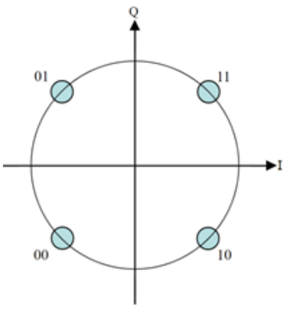
Fase y cuadratura para el simbolo QPSK

Componentes en fase y cuadratura de cada símbolo.

```
def fase(x):
  if (x == [0,0]).all():
    return np.pi*(5/4)
  if (x == [0,1]).all():
    return np.pi*(7/4)
  if (x == [1,0]).all():
    return np.pi*(3/4)
  if (x == [1,1]).all():
    return np.pi*(1/4)
```

Obteniendo valor en QPSK

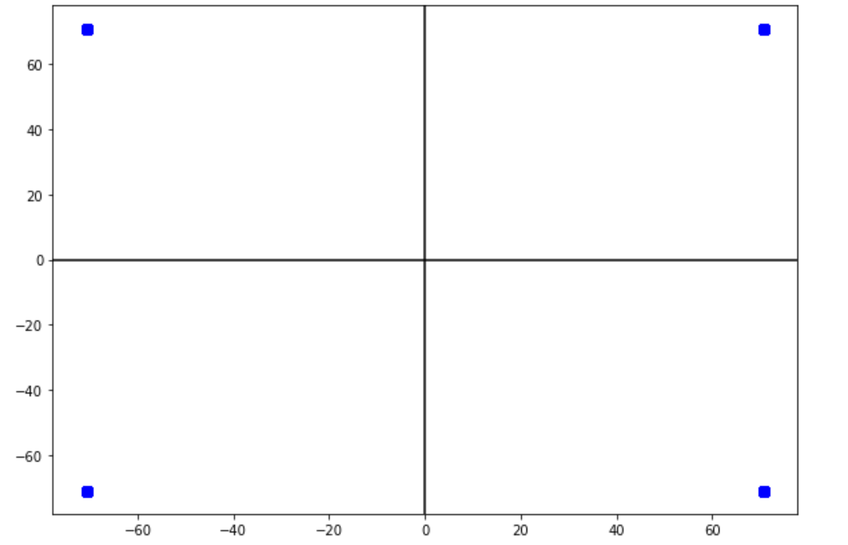
La modulación por desplazamiento de fase o PSK es una forma de modulación angular que consiste en hacer variar la fase de la portadora entre un número determinado de valores discretos. QPSK Es una modulación digital representada en el diagrama de constelación por cuatro puntos equidistantes del origen de coordenadas. Con cuatro fases, QPSK puede codificar dos bits por cada símbolo. La asignación de bits a cada símbolo suele hacerse mediante el código Gray, que consiste en que, entre dos símbolos adyacentes, los símbolos solo se diferencian en 1 bit, con lo que se logra minimizar la tasa de bits erróneos.




```
def qpsk(mensajes_salida):
  temp = 0
  xnr = []
  amplitud = 100 ## Normal amplitud = 1
  for i in range(2,len(mensajes_salida)+2,2):
    x = mensajes_salida[temp:i] # toma de a 2
    res = complex(amplitud *np.cos(fase(x)),amplitud*(np.sin(fase(x)))) # Componentes en fase y cuadratura de cada símbolc
    xnr.append(res)
    temp = i
  return xnr
```

```
###### Observando la funcion
muestras = 1000
simbolos_qpsk = (0,1)
real = []
imaginaria = []
mensaje = np.random.choice(simbolos_qpsk, muestras, p = (0.5,0.5))
vector_qpsk = qpsk(mensaje)
for c in vector_qpsk:
    real.append(c.real)
    imaginaria.append(c.imag)
print(vector_qpsk)
plt.figure(figsize=(10,7))
plt.plot(real,imaginaria,'o',markersize=8,color = "b")
plt.plot
plt.axvline(color="black")
```

```
plt.axhline(color="black")
plt.show()

[(70.71067811865476+70.71067811865474j), (70.71067811865474-70.710678118654
```



Vector OFDM

La multiplexación por división de frecuencias ortogonales(OFDM) es una técnica de transmisión que consiste en la multiplexación de un conjunto de ondas portadoras de diferentes frecuencias, donde cada una transporta información, la cual es modulada en QAM o en PSK(QPSK). OFDM se ha convertido en un esquema popular para la comunicación digital de banda ancha, que se utiliza en aplicaciones como la televisión digital, radiodifusión digital, acceso a internet mediante línea de abonado digital (DSL), redes inalámbricas, comunicaciones mediante redes eléctricas y la telefonía móvil 4G.

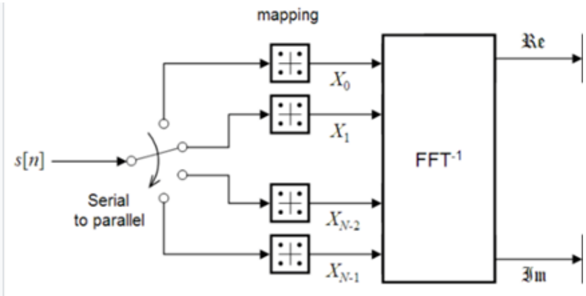


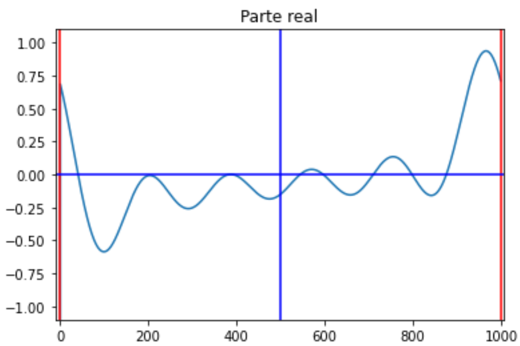
Diagrama de bloques de transmisor ideal de OFDM

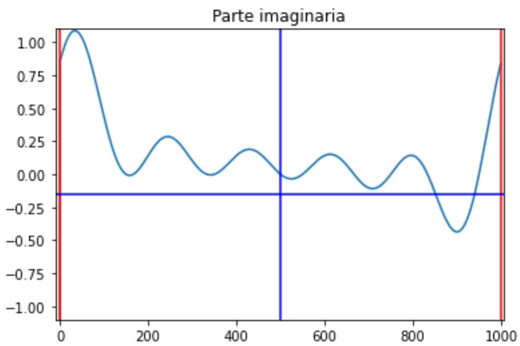
El en siguiente bloque se ingresa como parametro la cantidad de simbolos OFDM(N_ofdm), si se desea observar las componente(observar = True) y la cantidad de componentes(componentes = N°) a observar. Está función retorna un vector al cual se lo codico en QPSK y luego se le realizo la transformada transformada rápida inversa de Fourier $(ifftn)$. Las componentes pueden observarse desplazadas ya que los símbolos están desplazados.

```
from sympy import ifft
import numpy as np
import scipy.fft
def OFDM(N_ofdm,observar,componentes):
  cantidad_de_simbolos = 2         #[0 1]
  cant_mensajes = N_ofdm * cantidad_de_simbolos
  pi = (0.5 , 0.5)
  C = (0,1)        #codificacion QPSK
  mensaje = []
  if observar==True:
    mensaje = np.zeros(cant_mensajes)
    i = 0
    while componentes > 0:
      #print("paso")
      mensaje[i] = 1
      mensaje[i+1] = 1
      #print(mensaje)
      i += 2
      componentes -=1
  else:
    mensaje = np.random.choice(C, cant_mensajes, p = pi)
  vector_qpsk = qpsk(mensaje)
  #Transformada de Fourier inversa
```

```
    vector_OFDM = np.fft.ifftn(vector_qpsk)
    vector_OFDM = np.array(vector_OFDM).reshape(len(vector_OFDM),1)
    return vector_OFDM
```

```
###### Observando la funcion

N_aux = 1000
observar = True
componentes = 6
V = OFDM(N_aux,observar,componentes)
#print(V.real)
#xtick_labels = np.arange(-10, 10, 2)
fig,ax = plt.subplots()

ylim=round(np.max(V.real),2)

plt.plot((V.real)-0.15)
plt.axvline(color="r")
plt.axvline((N_aux/2),color="b")
plt.axvline(N_aux,color="r")
plt.axhline(color="b")
plt.xlim(-10,(N_aux+5))
plt.ylim(-ylim-0.01,ylim+0.01)
plt.title("Parte real")
plt.show()

ylim=round(np.max(V.imag),2)
plt.plot(V.imag)
plt.axvline(color="r")
plt.axvline((N_aux/2),color="b")
plt.axvline(N_aux,color="r")
plt.axhline(-0.15,color="b")
plt.xlim(-10,(N_aux+5))
plt.ylim(-ylim-0.01,ylim+0.01)
plt.title("Parte imaginaria")
plt.show()
```





Prefijo ciclico

Un prefijo cíclico (CP) es una copia del final de un símbolo de multiplexación por división de frecuencia ortogonal (OFDM) insertado al principio. Cada CP sirve como intervalo de guarda entre el símbolo OFDM.

La técnica de agreagr un CP permite disminuir la interferencia entre símbolos (ISI) y la interferencia entre portadoras (ICI) causadas por la propagación del retardo en el sistema OFDM.

La energía de símbolo que puede capturar un receptor OFDM depende de la longitud del CP:

Si la longitud del CP es mayor que la dispersión del retardo por trayectos múltiples de un símbolo OFDM, el receptor OFDM puede capturar toda la energía del símbolo.

Si la longitud del CP es más corta que la dispersión del retardo por trayectos múltiples de un símbolo OFDM, el receptor OFDM solo puede capturar algo de energía del símbolo.

La siguiente función recibe como parametro una señal (x), la cual contiene los distintos bloques de simbolos(tamaño = N_d + Num_ciclico) y a cada bloque de simbolo le extrae los Num_ciclico simbolos finales y los coloca al principio del bloque.


```
def prefijo_ciclico(x):
  for k in range(K):
    #print(k * (Num_ciclico + N_d)," : ",(k * (Num_ciclico + N_d) + Num_ciclico),"---",(k * (Num_ciclico + N_d) + N_d)," "
    x[k * (Num_ciclico + N_d): k * (Num_ciclico + N_d) + Num_ciclico] = x[k * (Num_ciclico + N_d) + N_d: (k + 1) * (Num_ci
  return x
```

```
#Observando la funcion
#Utiliza los parametros globales
print("Antes")
v_pc = np.arange(0,40,1)
print(v_pc)
v = prefijo_ciclico(v_pc)
#print("-"*20)
print("Despues")
print(v_pc)
```

```
Antes
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39]
Despues
[32 33 34 35 36 37 38 39  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39]
```

Sumando ruido gaussiano a la señal

El ruido gaussiano blanco aditivo (AWGN) es un modelo de ruido básico utilizado en la teoría de la información para imitar el efecto de muchos procesos aleatorios que ocurren en la naturaleza.

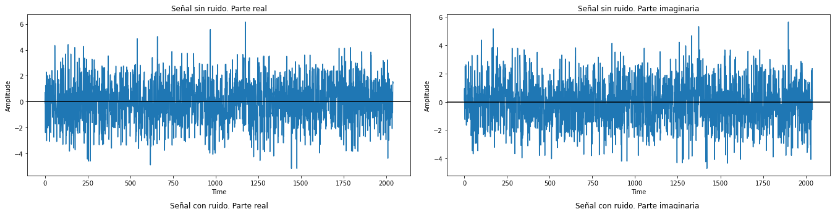
Este ruido proviene de muchas fuentes de ruido natural, como las vibraciones térmicas de los átomos en los conductores , ruido de disparo, fuentes celestes como el sol, entre otras. El teorema del límite central de la teoría de la probabilidad indica que la suma de muchos procesos aleatorios tenderá a tener una distribución llamada Gaussiana o Normal.

La función recibe la cantidad de simbolos del ruido N_ST el desvío de la señal del ruido sigma_w y la media (media = 0) y la señal a la cual se le añade el ruido (x).

```
def señal_transmitida(N_ST,sigma_w,media,x):
  w = sigma_w * np.random.randn(N_ST, 2).view(np.complex128) #+ media = 0
  #w = sigma_w * np.random.randn(N_H1, 2).view(np.complex128)
  #x = vec_pcf
  #w = np.around(np.random.normal(media, sigma_w,size=(N*2)),decimals = 2).view(np.complex128)
  señal_con_ruido = x + w
  return señal_con_ruido
```

```
#1.
#2.
#N = (K + 1) * (N_c + N_d)
observar = False
componentes = 0#
Vec_OFDM = OFDM(N,observar,componentes)
vec_pcf = prefijo_ciclico(Vec_OFDM)
sigma_w1 = 31
vec_wsg = señal_transmitida(N,sigma_w1,0,vec_pcf)

fig, axs = plt.subplots(nrows=2, ncols=2, figsize=(19, 9))

axs[0, 0].set_title("Señal sin ruido. Parte real")
axs[0, 0].plot(vec_pcf.real, color='C0')
axs[0, 0].set_xlabel("Time")
axs[0, 0].set_ylabel("Amplitude")
axs[0, 0].axhline(color="black")

```

```
axs[0, 1].set_title("Señal sin ruido. Parte imaginaria")
axs[0, 1].plot(vec_pcf.imag, color='C0')
axs[0, 1].set_xlabel("Time")
axs[0, 1].set_ylabel("Amplitude")
axs[0, 1].axhline(color="black")

axs[1, 0].set_title("Señal con ruido. Parte real")
axs[1, 0].plot(vec_wsg.real, color='C0')
axs[1, 0].set_xlabel("Time")
axs[1, 0].set_ylabel("Amplitude")
axs[1, 0].axhline(color="black")

axs[1, 1].set_title("Señal con ruido. Parte imaginaria ")
axs[1, 1].plot(vec_wsg.imag, color='C0')
axs[1, 1].set_xlabel("Time")
axs[1, 1].set_ylabel("Amplitude")
axs[1, 1].axhline(color="black")
fig.tight_layout()
plt.show()

plt.hist(vec_wsg.real)
plt.title("Señal gaussiana")
plt.show()

#e = np.fft.fftn(vec_wsg)

#fig2, axs2 = plt.subplots(nrows=1, ncols=2, figsize=(19, 9))
#axs2[0].set_title("Señal completa")
#axs2[0].hist(vec_pcf.real, color='C0')
#axs2[0].set_xlabel("Time")
#axs2[0].set_ylabel("Amplitude")
#axs2[0].axhline(color="black")
#w1 = sigma_w1 * np.random.randn(N, 2).view(np.complex128)
#e = np.fft.fftn(w1)
#axs2[1].set_title("Señal ruido")
#axs2[1].hist(e.real, color='C0')
#axs2[1].set_xlabel("Time")
#axs2[1].set_ylabel("Amplitude")
#axs2[1].axhline(color="black")
#plt.show()


```



Estadistico propuesto para CE y DE (Ciclo estacionario y Detector de energía)

Los símbolos de datos transmitidos x son independientes e idénticamente distribuidos (i.i.d.) con media cero y varianza unitaria. La función de autocorrelación (ACF) de x viene dada por:

$$rx[n,\tau] = E[x[n]x^*[n+\tau]]$$

Podemos definir $ry[n,\tau]$ y $rw[n,\tau]$ de manera similar. Debido a la inserción del CP(Prefijo cíclico), la señal $OFDM$ no es estacionaria. Por lo tanto, el $ACFrx[n,\tau]$ es variable en el tiempo pero periódico ($rx[n,\tau] = rx[n+k(Nd+Nc),\tau]$), y la señal OFDM es cicloestacionaria de segundo orden. Particularmente en un período, $rx[n,Nd], n = 0,1,\ldots,Nd+Nc-1$ se puede describir como

$$rx[n,Nd] = \begin{cases} 1, n = 0,1,\ldots,Nc-1 \\ 0, n = Nc, Nc+1,\ldots,Nc+Nd-1 \end{cases}$$

Dado que w es blanco y de media cero, $rw[n,\tau] = 0$ para cualquier $\tau \neq 0$ y $rw[n,0] = \sigma_w^2$.

Un usuario secundario debe detectar el espectro para decidir si las señales $OFDM$ del usuario principal(PU) está presente o no. Si el usuario principal no está activo, el usuario secundario(SU) puede aprovechar la oportunidad de hacer uso del espectro disponible. En presencia de un usuario principal, la señal $OFDM$ recibida que está corrompida por un ruido gaussiano blanco aditivo(AWGN), el canal en el usuario secundario se puede modelar simplemente como $y_{[n]} = x_{[n]} + w_{[n]}$, donde $w_{[n]}$ representa el ruido $AWGN$. Por lo tanto, deseamos decidir entre las siguientes dos hipótesis:

$$H_0 : y = w$$
$$H_1 : y = x + w$$

donde $w \sim CN(0;\sigma_w^2 I)$, y w es independiente de x.

El detector de energía no aprovecha la estructura especial del OFDM, al ser una señal con un CP(Prefijo ciclico). A continuación derivamos un detector basado en el cicloestacionario ACF(Función de autocorrelación) de la señal OFDM con un CP. Diseñamos un nuevo estadístico de prueba como:

$$T = \sum_{n=0}^{N_c-1} \hat{R}[n]$$

para decidir H_1:

$$|T_{(y)}| > \gamma$$

donde la ACF $r_y[n;N_d]$se define como

$$\hat{R}[n] = \frac{1}{K}\sum_{K=0}^{K-1} \hat{r}[n+k(N_c+N_d), N_d], n = 0,1,\ldots,N_c+N_{(d-1)}$$

con

$$\hat{r}[n,Nd] = y[n]y^*[n+N_d]$$

Tenga en cuenta que:

$$y[n], n = 0,1,\ldots,K(N_c+N_d)+N_{d-1}$$

son usado para generar

$$\hat{R}_{[n]}, n = 0,1,\ldots,N_c+N_{d-1}.$$

Según el teorema del límite centra y suponiendo una $IFFT$ lo suficientemente grande, se puede suponer que $T_{(y)}$ es una variable aleatoria Gaussiana compleja.

Obtencón del estadisco de prueba H_0

En este caso la señal solo contiene ruido $AWGN$.

```
def señal_recibida(y):
  return np.fft.fftn(y)

```

```
import math
def generar_estatistico_H0(NUM_STATISTICS, sigma_w, N_H0):
    T_y = np.zeros(NUM_STATISTICS, dtype=np.complex128)
    T_DE =  np.zeros(NUM_STATISTICS)
    for ind in range(NUM_STATISTICS):
        #result = []
        w = sigma_w * np.random.randn(N_H0, 2).view(np.complex128)
```

```
        y = w
        #y = señal_recibida(y)

        # Calculate test statistic DE
        #T_DE[ind] = np.sum(np.square(np.abs(y.real)))
        T_DE[ind] = np.sum((np.square(np.abs(y))))
        # Calculate test statistic CE
        val = np.complex128(0)
        for n in range(Num_ciclico):
            for k in range(K):
                val += y[n + k * (Num_ciclico + N_d)] * np.conjugate(y[n + k * (Num_ciclico + N_d) + N_d])
        T_y[ind] = 1 / K * val
    return T_y,T_DE

```

Obtención del estadisco de prueba $H1$

Señal del usuario principal con ruido $AWGN$.

```
def generar_estatistico_H1(NUM_STATISTICS, sigma_w, N_H1):
  T_y = np.zeros(NUM_STATISTICS, dtype=np.complex128)
  T_DE =  np.zeros(NUM_STATISTICS)

  for ind in range(NUM_STATISTICS):
        result = []
        x = OFDM(N_H1,False,0)
        #x  = sigma_s * np.random.randn(N_H1, 1)
        x = prefijo_ciclico(x)
        y = señal_transmitida(N_H1,sigma_w,0,x)

        T_DE[ind] = np.sum((np.square(np.abs(y))))
        #T.append(np.sum(np.square(np.abs(y))))

        val = np.complex128(0)
        for n in range(Num_ciclico):
            for k in range(K):
                val += y[n + k * (Num_ciclico + N_d)] * np.conjugate(y[n + k * (Num_ciclico + N_d) + N_d])
                #if n==0:
                  #print("val :",val)
        T_y[ind] = 1 / K * val

  return T_y,T_DE
```

Obtención de σ_w apartir del SNR y σ_s

SNR

El SNR se puede denotar tambie como $\gamma = \frac{\sigma_s^2}{\sigma_w^2}$ donde σ_s^2 es la varianza de la señal de un usuario principal y σ_w^2 es la varianza del ruido.

Cuando expresamos el ruido en decibeles obtenemos la figura del ruido de la seguiente manera:

$$SNR = 10 * log_{10} F$$

donde F es el factor de ruido en este caso es $\gamma = \frac{\sigma_s^2}{\sigma_w^2}$ entonces:

$$SNR = 10 * log_{10} \frac{\sigma_s^2}{\sigma_w^2}$$

despejando se obtiene

$$\frac{SNR}{10} = log_{10} \frac{\sigma_s^2}{\sigma_w^2}$$
$$10^{\frac{SNR}{10}} = \frac{\sigma_s^2}{\sigma_w^2}$$
$$\sigma_w^2 = \frac{\sigma_s^2}{10^{\frac{SNR}{10}}}$$
$$\sigma_w = \sqrt{\frac{\sigma_s^2}{10^{\frac{SNR}{10}}}}$$

La relación señal/ruido o S/R (signal-to-noise-ratio,(SNR)) se define como la proporción existente entre la potencia de la señal que se transmite y la potencia del ruido que la corrompe.

La siguiente funcion retorna un σ_w apartir de un SNR y un σ_s.

```
def ret_sigma_w(SNR,sigma_s):
  #SNR_NOISE = 1
  sigma_w = np.sqrt(sigma_s ** 2 / 10 ** (SNR / 10))
  #sigma_w = np.sqrt(sigma_w * 10 ** (SNR_NOISE / 10))
  return sigma_w
```

```
Senr = -28
a = ret_sigma_w(Senr,1)
print("a : ",a)
print(10*np.log10(1/((25**2))))


#b = 8/(50*(a** 4))
#print("b : ",b)
#c = chi2.isf(q=P_FA, df=2)
#Num_ciclico / K * (sigma_w ** 4)
#print("Desvio :",(Num_ciclico / (K*a**4)))
#p = (chi2.isf(q=P_FA, df=2) * (((Num_ciclico / (2*K)) * ((2*a) ** 4))))
#print("p :",p)
#print(np.sqrt(round(p,4)))
#print("fin",(chi2.isf(q=P_FA, df=2) * (2*(a ** 2))))
```

```
a :  25.118864315095795
-27.95880017344075
```

FUNCIÓN PRINCIPAL

En esta función se obtienen los estadisticos H_0 y H_1 para distintos SNR

A modo de ejemplo para un cierto SNR se resetean las variables

NUM_FALSE_ALARM = 0

NUM_DETECTION = 0

calculamos el σ_w luego calculamos nuestro gamma de acuerdo a la distribucion X^2 la cual la calculamos con una probabilidad de $P_{FA} = 0.05$ para el dectector ciclo estacionario utizamos un numero dos grado de libertad ($df = 2$ H_0 y H_1) y un desvío $\frac{N_c}{K*\sigma_w^4}$ luego, la función generar_estatistico_H0 nos devuelve un vector en el cuál tenemos los valores del analisis de cada bloque con el estadistico $T_{(y)}H_0$ ahora lo evaluamos con nuestro gamma_CE para tomar la decision sin $T_{(y)}H0 > \gamma$ incrementamos NUM_FALSE_ALARM y si la función generar_estatistico_H1 nos devuelve $T_{(y)} > \gamma$ incremenatamos NUM_DETECTION.

Finalmente

$$P\_FA\_CALC\_CE = \frac{NUM\_FALSE\_ALARM}{NUM\_STATISTICS}$$
$$P\_D\_CALC\_CE = \frac{NUM\_DETECTION}{NUM\_STATISTICS}$$

Observacion de γ

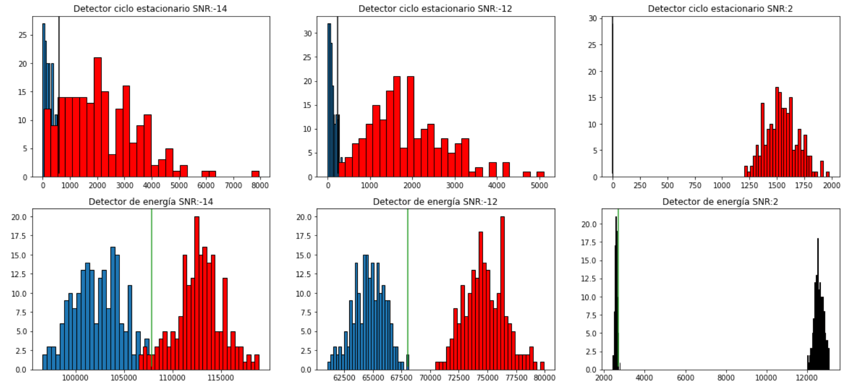
```

def imprimir(T_aux_H0_CE,T_aux_H1_CE,T_aux_H0_DE,T_aux_H1_DE,SNRS ,gammas_CE,gammas_DE):

  fig, axs = plt.subplots(nrows=2, ncols=3, figsize=(20, 9))

  axs[0, 0].set_title("Detector ciclo estacionario SNR:"+str(SNRS[0]))
  axs[0, 0].hist(T_aux_H0_CE[0].real,bins = 30,edgecolor = "black", color='C0')
  axs[0, 0].hist(T_aux_H1_CE[0].real, bins = 30,edgecolor = "black",color='r')
  axs[0, 0].axvline(gammas_CE[0],color="black",ymin=0.02,ymax=1,linestyle="-")

  axs[0, 1].set_title("Detector ciclo estacionario SNR:"+str(SNRS[1]))
  axs[0, 1].hist(T_aux_H0_CE[1].real,bins = 30,edgecolor = "black", color='C0')
  axs[0, 1].hist(T_aux_H1_CE[1].real, bins = 30,edgecolor = "black",color='r')
  axs[0, 1].axvline(gammas_CE[1],color="black",ymin=0.02,ymax=1,linestyle="-")

  axs[0, 2].set_title("Detector ciclo estacionario SNR:"+str(SNRS[2]))
  axs[0, 2].hist(T_aux_H0_CE[2].real,bins = 30,edgecolor = "black", color='C0')
  axs[0, 2].hist(T_aux_H1_CE[2].real, bins = 30,edgecolor = "black",color='r')
  axs[0, 2].axvline(gammas_CE[2],color="black",ymin=0.02,ymax=1,linestyle="-")

  axs[1, 0].set_title("Detector de energía SNR:"+str(SNRS[0]))
  axs[1, 0].hist(T_aux_H0_DE[0].real,bins = 30,edgecolor = "black", color='C0')
  axs[1, 0].hist(T_aux_H1_DE[0].real, bins = 30,edgecolor = "black",color='r')
  axs[1, 0].axvline(gammas_DE[0],color="C2",ymin=0.02,ymax=1,linestyle="-")

  axs[1, 1].set_title("Detector de energía SNR:"+str(SNRS[1]))
  axs[1, 1].hist(T_aux_H0_DE[1].real,bins = 30,edgecolor = "black", color='C0')
  axs[1, 1].hist(T_aux_H1_DE[1].real, bins = 30,edgecolor = "black", color='r')
  axs[1, 1].axvline(gammas_DE[1],color="C2",ymin=0.02,ymax=1,linestyle="-")

  axs[1, 2].set_title("Detector de energía SNR:"+str(SNRS[2]))
  axs[1, 2].hist(T_aux_H0_DE[2].real,bins = 30,edgecolor = "black",color='C0')
```

```
  axs[1, 2].hist(T_aux_H1_DE[2].real, bins = 30,edgecolor = "black",color='r')
  axs[1, 2].axvline(gammas_DE[2],color="C2",ymin=0.02,ymax=1,linestyle="-")
  plt.show()

```

```

#Parametros para evaluar SNR en el detector de energía
T_aux_H0_CE = []
T_aux_H1_CE = []

T_aux_H0_DE = []
T_aux_H1_DE = []

SNRS = []
gammas_DE = []
gammas_CE = []

for ind, SNR in enumerate(SNR_list):
  NUM_FALSE_ALARM = 0
  NUM_DETECTION = 0
  NUM_FALSE_ALARM_DE = 0
  NUM_DETECTION_DE = 0
  NUM_MISS_DETECTION_DE = 0

  sigma_w = ret_sigma_w(SNR,sigma_s)

  gamma_CE = chi2.isf(q=P_FA, df=2)* Num_ciclico / K * (sigma_w ** 4) #* Num_ciclico / (K * (sigma_w ** 4))
  gamma_DE = scs.chi2.isf(q=P_FA, df=N) *  2*(sigma_w ** 2)
  #gamma_DE = scs.norm.isf(P_FA) *( 2*np.sqrt(2*N*(sigma_w)**4)) + N*(sigma_w)**2

  T_y_0_CE,T_y_0_DE = generar_estatistico_H0(NUM_STATISTICS, sigma_w, N)
  T_y_1_CE,T_y_1_DE = generar_estatistico_H1(NUM_STATISTICS, sigma_w, N)
  #media_H0 = 2*N*sigma_w**2
  #media_H1 = 2*N*(sigma_s**2+sigma_w**2)
  #gamma_DE = ((-np.square(media_H1) + np.square(media_H0))/( (2*(media_H0)) - (2*media_H1)))

  if ind  == 4:


    #media_H0 = 2*N*sigma_w**2
    #media_H1 = np.mean(T_y_1_DE)

    #valor = ((-np.square(media_H1) + np.square(media_H0))/( (2*(media_H0)) - (2*media_H1)))

    print("sigma_w",sigma_w," media h1:",np.mean(T_y_1_DE)," N:",N," mediaH0:",np.mean(T_y_0_DE)," sigma calcu",N*sigma_w*
    #print("este deberia ser similar a-->", np.mean(T_y_0_DE)," -->", gamma_DE)
    #print("MEDIA CALCULADA -->", (N*sigma_w**2), "SIGMA ",sigma_w)

    #print("2m-->", np.mean(T_y_1_DE))
    #print("1s-->", np.std(T_y_0_DE))
    #print("2s-->", np.std(T_y_1_DE))
    #print("N ",N)

    T_aux_H0_CE.append(np.square(np.abs(T_y_0_CE)))
    T_aux_H1_CE.append(np.square(np.abs(T_y_1_CE)))

    T_aux_H0_DE.append(T_y_0_DE)
    T_aux_H1_DE.append(T_y_1_DE)

    SNRS.append(SNR)
    gammas_CE.append(gamma_CE)
    gammas_DE.append(gamma_DE)
    #plt.hist(T_y_0_DE,bins = 30)
    #plt.plot()
    #print("sigma_w :",sigma_w)
  if SNR  == -14:
    print("sigma_w",sigma_w," media h1:",np.mean(T_y_1_DE)," N:",N," mediaH0:",np.mean(T_y_0_DE)," sigma calcu",N*sigma_w*
    T_aux_H0_CE.append(np.square(np.abs(T_y_0_CE)))
    T_aux_H1_CE.append(np.square(np.abs(T_y_1_CE)))

    T_aux_H0_DE.append(T_y_0_DE)
    T_aux_H1_DE.append(T_y_1_DE)

    SNRS.append(SNR)
    gammas_CE.append(gamma_CE)
    gammas_DE.append(gamma_DE)

  if SNR  == 2:
    print("sigma_w",sigma_w," media h1:",np.mean(T_y_1_DE)," N:",N," mediaH0:",np.mean(T_y_0_DE)," sigma calcu",N*sigma_w*
    T_aux_H0_CE.append(np.square(np.abs(T_y_0_CE)))
    T_aux_H1_CE.append(np.square(np.abs(T_y_1_CE)))

```

```
    T_aux_H0_DE.append(T_y_0_DE)
    T_aux_H1_DE.append(T_y_1_DE)

    SNRS.append(SNR)
    gammas_CE.append(gamma_CE)
    gammas_DE.append(gamma_DE)

######### Ciclo estacionario #########
  for T in T_y_0_CE:
    if np.square(np.abs(T)) >= gamma_CE:
      NUM_FALSE_ALARM += 1

  for T in T_y_1_CE:
    if np.square(np.abs(T)) >= gamma_CE:
      NUM_DETECTION += 1

  P_FA_CALC_CE[ind] = NUM_FALSE_ALARM / NUM_STATISTICS
  P_D_CALC_CE[ind] = NUM_DETECTION / NUM_STATISTICS
  P_FA_THEO_CE[ind] = P_FA

######### Detector de energia #########
  for T in T_y_0_DE:
    if T >= (gamma_DE):
      NUM_FALSE_ALARM_DE += 1

  for T in T_y_1_DE:
    if T >= (gamma_DE):
      NUM_DETECTION_DE += 1

  P_M_CALC_DE[ind] = NUM_MISS_DETECTION_DE / NUM_STATISTICS
  P_D_CALC_DE[ind] = NUM_DETECTION_DE / NUM_STATISTICS

#gammas_DE[0] = valor
#print("valor",valor)
print("NUM_STATISTICS :",NUM_STATISTICS)
print("N :",N)
imprimir(T_aux_H0_CE,T_aux_H1_CE,T_aux_H0_DE,T_aux_H1_DE,SNRS ,gammas_CE,gammas_DE)
```

```
sigma_w 5.011872336272722  media h1: 112677.61285532663  N: 2040  mediaH0:
sigma_w 3.9810717055349722  media h1: 74784.12052273113  N: 2040  mediaH0:
sigma_w 0.7943282347242815  media h1: 12568.468931487338  N: 2040  mediaH0:
NUM_STATISTICS : 200
N : 2040
```

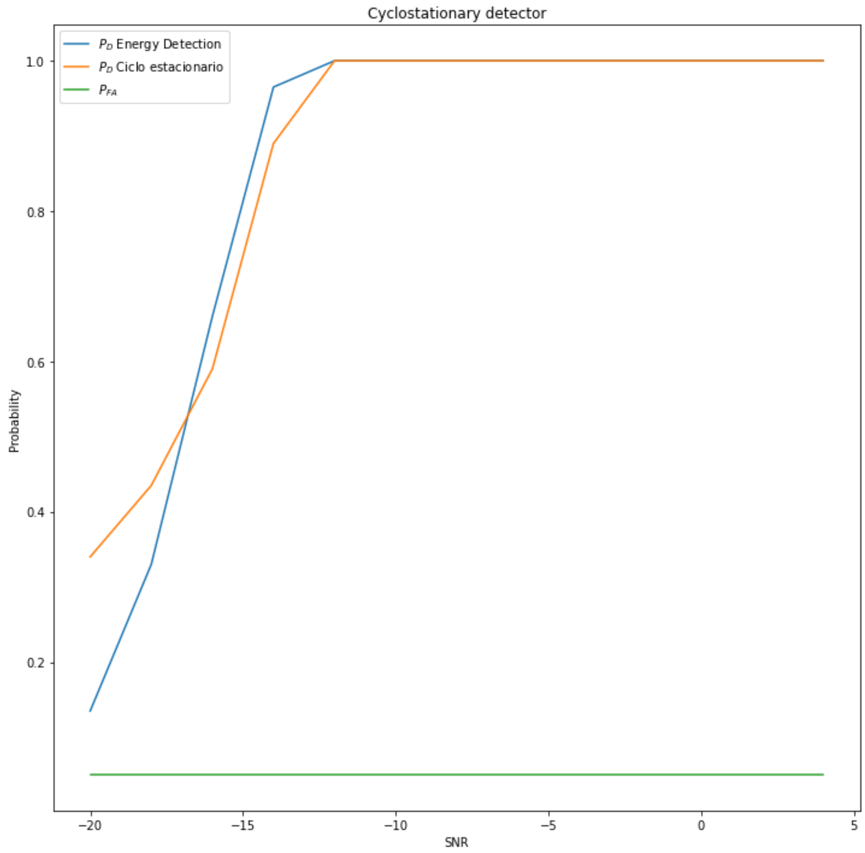


Comparando metodos

Finalmente comparamos los resultados obtenidos anteriormente para los distinto SNR de los dos métodos.

En el eje vertical se coloco la probabilidades de deteción tanto del ciclo estacionario como del detector de enrgía y en el eje vertical se colocaron los distintos SNR

```
plt.figure(figsize=(12,12))
plt.plot(SNR_list, P_D_CALC_DE, label="$P_D$ Energy Detection")
plt.plot(SNR_list, P_D_CALC_CE, label="$P_D$ Ciclo estacionario")
plt.plot(SNR_list, P_FA_THEO_CE, label="$P_{FA}$")
plt.xlabel("SNR")
plt.ylabel("Probability")
plt.title("Cyclostationary detector")
plt.legend()
plt.show()
```



En la gráfica anterior es posible observar claramente que los resultados obtenidos para una baja relación señal-ruido en el modelo de caracteristicas ciclo estacionaria son mejores que las respuestas del detector de energía. Sin embargo implementar un sistema de sensado por ciclos estacionarios requiere que se cumplan ciertas condiciones repecto a la utilización del canal, específicamente las señales transmitidas por el ancho de banda que está siendo sensado deben tener propiedades cicloestacionaria.

Para la elección alguno de los métodos propuestos debemos tener en cuenta:

Método de detección	Parámetro de decisión	Ventajas	Desventajas	Mejoras sugeridas
Detector de energía	Comparación de la energía de la señal recibida con un umbral	Simple de implementar, no requiere información previa sobre la señal primaria	No puede funcionar en un entorno SNR bajo, tiene una alta tasa de falsas alarmas	Detector de energía mejorado, detector de energía adaptativa de ruido, detector de energía de umbral adaptativo
Detector de características cicloestacionarias	Comparación de los valores distintos de cero obtenidos por CSD para las propiedades cicloestacionarias de la señal primaria	Robusto al ruido, puede diferenciar entre diferentes tipos de transmisiones primarias	Fallará si la señal primaria no tiene propiedad cicloestacionaria, tiene una alta complejidad computacional, alto costo, requiere algún conocimiento previo de la señal primaria	Reducción de la complejidad: dividiendo la entrada en subseries, submuestreando la señal mediante tunelización
Detector de filtro emparejado	Correlacionar la señal recibida con la señal primaria ya conocida	Requiere menos tiempo de detección, óptimo si se conoce la señal principal	Alta complejidad ya que requiere un receptor separado para cada usuario principal, requiere un conocimiento previo sobre la señal principal	Detección coherente usando menos detalles con respecto a la señal primaria

Bibliografía utilizada:

https://revistas.udistrital.edu.co/index.php/reving/article/view/2699

http://blog.espol.edu.ec/estg1003/category/temas-por-unidad/temas-2da-evaluacion/funciones-densidad-y-acumulada/

https://www.researchgate.net/publication/269672304_Energy_Detection_Technique_for_Spectrum_Sensing_in_Cognitive_Radio_A_Survey

https://github.com/vineeths96/Spectrum-Sensing-for-Cognitive-Radio

https://www.scirp.org/journal/paperinformation.aspx?paperid=101790#ref6

https://en.wikipedia.org/wiki/Signal-to-noise_ratio

https://en.wikipedia.org/wiki/Q-function

https://es.wikipedia.org/wiki/Radio_cognitiva

https://www.idc-online.com/technical_references/pdfs/data_communications/ENERGY%20DETECTION.pdf

Energía y Potencia:

https://www.siue.edu/~yadwang/ECE351_Lec3.pdf (pag 29)

https://www.rgpv.ac.in/campus/ECE/EC%20402%20SignalSystems%20Notes.pdf (pag 4)

✓ 0 s completado a las 2:04