1) Implemente mediante una función un correlador

Implemente mediante una función un correlador de la señal recibida con una forma de onda triangular dada por:

$$\begin{matrix} t & 0 \le t \le 1 \\ 2-t & 1 < t \le 2 \\ 0 & \text{otherwise} \end{matrix}$$

La función tiene dos parámetros: la señal recibida representada por un vector, y el intervalo de tiempo entre elementos consecutivos de dicho vector

Nota: la correlación entre dos señales de tiempo continuo puede calcularse numéricamente aproximando la integral numérica mediante la regla del rectángulo

$\int_t^{t+\Delta t} f(x)dx \simeq f(t)\Delta t$

```
import numpy as np
def correlador(signal,T):
 t = np.arange(0,2+T,T)
 waveform = np.zeros(len(t))
 for i in range(len(t)):
  if t[i] <= 1:
   waveform[i] = t[i]
  else:
   waveform[i] = 2 - t[i]
 min_len = len(signal) if len(signal)<len(waveform) else len(waveform)
 corr = 0
 for i in range(min_len):
  corr +=  signal[i] * waveform[i]
 return corr*T
```

2) A través del procedimiento Gram-Schmidt

A través del procedimiento de Gram-Schmidt de ortonormalización encontrar una base ortonormal para el espacio generado por los vectores \alpha_1=(1,0,1,1)^T , \alpha_2=(2,1,0,1)^T , \alpha_3=(1,0,1,-2)^T y \alpha_4=(2,0,2,-1)^T.

Seleccione una:

- a. Ningua es correcta.
- b. $\phi_1=(-\frac{1}{\sqrt{3}},0,\frac{1}{\sqrt{3}},\frac{1}{\sqrt{3}})$, $\phi_2=(\frac{1}{\sqrt{3}},\frac{1}{\sqrt{3}},0,-\frac{1}{\sqrt{3}})$, $\phi_3=(\frac{1}{\sqrt{6}},0,\frac{1}{\sqrt{6}},\frac{2}{\sqrt{6}})$ y $\phi_4=(0,0,0,0)$ ✖
- c. $\phi_1=(0,-\frac{1}{\sqrt{3}},\frac{1}{\sqrt{3}},\frac{1}{\sqrt{3}})$, $\phi_2=(\frac{1}{\sqrt{3}},\frac{1}{\sqrt{3}},-\frac{1}{\sqrt{3}},0)$, $\phi_3=(\frac{2}{\sqrt{6}},0,\frac{2}{\sqrt{6}},\frac{2}{\sqrt{6}})$ y $\phi_4=(0,0,0,0)$
- d. $\phi_1=(\frac{1}{\sqrt{3}},0,\frac{1}{\sqrt{3}},\frac{1}{\sqrt{3}})$, $\phi_2=(\frac{1}{\sqrt{3}},\frac{1}{\sqrt{3}},-\frac{1}{\sqrt{3}},0)$, $\phi_3=(\frac{1}{\sqrt{6}},0,\frac{1}{\sqrt{6}},-\frac{2}{\sqrt{6}})$ y $\phi_4=(0,0,0,0)$

\phi_3=(\frac{1}{\sqrt{6}},0,\frac{1}{\sqrt{6}},-\frac{2}{\sqrt{6}})

```
import numpy as np

def gram_schmidt(vectors):
   num_vecs = len(vectors)
   ortho_basis = np.zeros_like(vectors, dtype=np.float64)

   for i in range(num_vecs):
      # Ortogonalización
      ortho_basis[i] = vectors[i]
      for j in range(i):
```

```
            ortho_basis[i] -= np.dot(vectors[i], ortho_basis[j]) / np.dot(ortho_basis[j], 
ortho_basis[j]) * ortho_basis[j]

        # Normalización
        ortho_basis[i] /= np.linalg.norm(ortho_basis[i])

    return ortho_basis

# Definir los vectores como float
alpha_1 = np.array([1.0, 0.0, 1.0, 1.0])
alpha_2 = np.array([2.0, 1.0, 0.0, 1.0])
alpha_3 = np.array([1.0, 0.0, 1.0, -2.0])
alpha_4 = np.array([2.0, 0.0, 2.0, -1.0])

# Crear una matriz con los vectores
vectors = np.array([alpha_1, alpha_2, alpha_3, alpha_4])

# Aplicar Gram-Schmidt
ortho_basis = gram_schmidt(vectors)

print("Base ortonormal:")
for i, vec in enumerate(ortho_basis):
    print(f"v_{i+1}:", vec)
```

3) conformador de onda basado en una base ortonormal

Implementar un conformador de onda basado en una base ortonormal a desplazamientos múltiplos de T de la familia de raiz de coseno realzado. Sección 5.5 del libro de Bixio

Proveer una función:

```
def raiz_coseno_realzado(simbolos, Beta, T, tt):
    ................................
    return signal_added,signal
```

con los siguiente argumentos:

- símbolos: lista de símbolos de un alfabeto M-PAM
- Beta: exceso de ancho de banda
- T: periodo de símbolo
- tt: intervalo de tiempo

La función retorna:

- signal_added: la forma de onda resultante de los símbolos transmitidos
- signal: matriz (lista) de mxn donde m es la cantidad de símbolos y n el intervalo de tiempo

Ayuda: Si se grafica las salidas de la función se obtendrían señales similares a las mostradas en la figura 5.7 del libro de Bixio (pag. 172)

```
import numpy as np

def rrcosfilter(t, beta, Ts,iT):
    return 1/np.sqrt(Ts) * np.sinc((t-iT)/Ts) * np.cos(np.pi*beta*(t-iT)/Ts) / (1 - 
(2*beta*(t-iT)/Ts) ** 2)

def raiz_coseno_realzado(simbolos, Beta, T, tt):
  sum_sig = 0
```

```
 signal = []


 for i in range(len(simbolos)):
   signal.append(simbolos[i]*rrcosfilter(tt,Beta,T,i*T))
   sum_sig = sum_sig + signal[i]
   #plt.plot(tt,output[i])
 return sum_sig,signal
```

4)Realizar un conformador de onda con:

Este ejercicio está basado en el ejemplo 3.10, ubicado en la página 106 del libro Principles of Digital Communication: A top-down approach - Bixio Rimoldi. La señalización se conoce como Single-Shot QAM

Realizar un conformador de onda con:

$\phi_1(t) = \sqrt{\frac{2}{T}} cos(2\pi f_c t)$

$\phi_2(t) = \sqrt{\frac{2}{T}} sin(2\pi f_c t)$

con $t \in [0, T]$

Considerando que $2 f_c T$ es un entero. Las componentes del código $c_i = (c_{i,1}, c_{i,2})$ toma valores en un alfabeto discreto de la forma $\{\pm a, \pm 3a \ldots, \pm (m-1)a\}$

El programa debe generar la forma de onda para el código de entrada c_i, esto es:

$w_i(t) = c_{i,1}\phi_1(t) + c_{i,2}\phi_2(t)$

La forma de onda tiene $f_c = 1$ y se genera con un muestreo $\Delta t = 0.1$

```
def waveformer(c, T=1, fc=1, delta_t=0.1):
    def phi1(t):
        return np.sqrt(2/T) * np.cos(2 * np.pi * fc * t)

    def phi2(t):
        return np.sqrt(2/T) * np.sin(2 * np.pi * fc * t)

    c1, c2 = c

    num = int(T / delta_t) + 1
    t = np.linspace(0, T, num)

    w = c1 * phi1(t) + c2 * phi2(t)

    return w
```

5)Los siguientes parametros pertenecen a una sonda espacial con rumbo al planeta Mercurio

Los siguientes parámetros pertenecen a una sonda espacial con rumbo al planeta Mercurio

$P_T = 18,8 \quad watt$
$\lambda = 0,15 \ m$
$G_T = 575,44$
$G_R = 1380000$
$d = 160000000 \ Km$
$T_N = 13,5 \ kelvin$
$R_b = 117,6 \ kbps$ BPSK

Calcule la relación $\frac{\mathcal{E}}{N_o}$ (relación energía por bit / densidad espectral de potencia de ruido)

```
import math
from scipy.stats import norm
import numpy as np
kb=1.381e-23 #Constante de Boltzmann
def BER(pt, lda, gt, gr, d, tn, rb):
   d = d * 1000
   rb = rb * 1000
   epsilon = ((1 / rb) * gt * gr * pt) / ((((4 * math.pi * d) / lda) ** 2) * kb * tn)
   return epsilon
```

6)Los siguientes parametros permiten calcular el bit error

Los siguientes parámetros permiten calcular el bit error rate para una modulación antipodal (por ejemplo BPSK)

P_T =Potencia de señal transmitida $[watts]$
λ =Longitud de onda de la portadora $[m]$
G_T =Ganancia de la antena transmisora
G_R =Ganancia de la antena receptora
d = distancia $[Km]$
T_N = Temperatura de ruido del receptor $[kelvin]$
R_b = bit rate $[kbps]$

Escriba una función en python3 *BER(pt,lda,gt,gr,d,tn,rb)* que devuelva el bit error rate (BER)
Exprese el resultado con con 4 decimales (para su corrección automática)

Ayuda:

incluya la librería numpy para acceder a la función sqrt()

incluya la librería scipy para acceder a la función Q

"from scipy.stats import norm"

```
import math
from scipy.stats import norm
import numpy as np
kb=1.381e-23 #Constante de Boltzmann
def BER(pt, lda, gt, gr, d, tn, rb):
   d = d * 1000
   rb = rb * 1000
```

```
    epsilon = ((1 / rb) * gt * gr * pt) / ((((4 * math.pi * d) / lda) ** 2) * kb * tn)
    valorx = np.sqrt(2*epsilon)
    tasa_error_bits = norm.sf(valorx)
    return round(tasa_error_bits,4)
```

7)Se dispone de un alfabeto de símbolos reales

Se dispone de un alfabeto de símbolos reales con modulación *6-PAM*, $A = \{0, 1, 2, 3, 4, 5\}$. Un decodificador *ML* los selecciona de acuerdo al criterio de mínima distancia. El receptor realiza una detección incorrecta si la observación real $\boldsymbol{y}$ aparece fuera de su región de decodificación: este caso se da si el ruido real $\boldsymbol{Z}$ es mayor que $d/2$, donde $d = c_i - c_{i-1}$, para $i = 1,2,...,5$. Se considera que la potencia de ruido está normalizada.

A) Imprimir en pantalla el valor numérico de $P(5)$, la probabilidad de error en la detección suponiendo que se transmitió el símbolo de amplitud 5.

B) Repetir el ejercicio A) esta vez imprimiendo debajo el valor numérico de $P(3)$.

Sugerencia: para que el sistema pueda validar los resultados, imprimir solo valores numéricos **redondeados** a dos decimales, por ejemplo la salida del código debería tener esta forma:
0.10
0.20

Editor online: https://repl.it/languages/python3

```
from scipy.stats import norm as nm
# qx = nm.sf(x, loc=0, scale=1) calcula la función Q evaluada en x
dsobre2 = 1/2
qx = nm.sf(dsobre2, loc=0, scale=1)
prob5 = qx
print(round(prob5,2))
prob3 = 2*qx
print(round(prob3,2))
```

9) Los siguientes parámetros pertenecen a una sonda espacial con rumbo al planeta Mercurio

Los siguientes parámetros pertenecen a una sonda espacial con rumbo al planeta Mercurio

$P_T = 14,9 \quad watt$
$\lambda = 0,13 \;\; m$
$G_T = 575,44$
$G_R = 1380000$
$d = 1600000 \;\; Km$
$T_N = 13,5 \;\; kelvin$
$R_b = 117,6 \;\; kbps$ BPSK

Calcule la relación $\frac{\xi}{N_o}$ (relación energía por bit / densidad espectral de potencia de ruido)

```
import math
from scipy.stats import norm
import numpy as np
kb=1.381e-23 #Constante de Boltzmann
def BER(pt, lda, gt, gr, d, tn, rb):
    d = d * 1000
    rb = rb * 1000
```

```
    epsilon = ((1 / rb) * gt * gr * pt) / ((((4 * 3.14 * d) / lda) ** 2) * kb * tn)
    return epsilon
print(BER(14.9,0.13,575.44 ,1.38e6 ,1.6e6,13.5 ,117.6))#Cambiar datos no sean vergas
```

11)Calcular la probabilidad de error cuando se transmite una de las hipotesis binarias

Calcular la probabilidad de error cuando se transmite una de las hipótesis binarias {0 1} por un canal AWGN. La varianza del ruido es 0.01

```
import numpy as np
from scipy import special
from scipy.stats import norm


valorx = 1/(2*np.sqrt(0.01))
q = norm.sf(valorx)
print(q)
```

12)sea X una variable aleatoria (VA) que toma valores en el alfabeto

Sea X una variable aleatoria (VA) que toma valores en el alfabeto $\Omega_x = (x_1, x_2)$. Sea la VA Y = X + N, donde N es una variable aleatoria gaussiana de media cero y varianza σ^2, independiente de X. La expresión para la frontera de decisión MAP entre x_1 y x_2 es la siguiente:

$$\frac{x_1+x_2}{2} + \frac{\sigma^2}{x_1-x_2} ln\left(\frac{p_x(x_2)}{p_x(x_1)}\right)$$

Calcular el valor que corresponde para $x_1 =$ -1,6, $x_2 =$ 1,9, considerando varianza unitaria y que la probabilidad del símbolo x_2 es dos veces la del símbolo x_1. Expresar el resultado con tres cifras de precisión.

```
import numpy as np

# Valores dados
x1 = -1.6
x2 = 1.9
sigma_squared = 1
px_x2_over_px_x1 = 2

# Calcular la frontera de decisión MAP
decision_boundary = (x1 + x2) / 2 + sigma_squared / (x1 - x2) *
np.log(px_x2_over_px_x1)

# Imprimir el resultado con tres cifras de precisión
print(f"La frontera de decisión MAP es {decision_boundary:.3f}")
```

13)Cartas de colores

Cartas de colores

En un recinto un par de cajas contienen cartas blancas y rojas con la siguiente distribución:

Cajas con cartas.

	Blancas	Rojas	Total por caja
Caja 1	999	1	1000
Caja 2	1	999	1000
Total	1000	1000	2000

Luego, seleccionamos aleatoriamente una caja y luego también de manera aleatoria una carta dentro de la caja. Esta carta resulta de color blanco. ¿Cuál es la probabilidad de que fuese extraída de la primera caja?

Hint: Aplique teorema de Bayes.

Respuesta: 0,99 ✖

La respuesta correcta es: 0,999

```
# Probabilidades iniciales
P_C1 = 0.5
P_C2 = 0.5
P_B_C1 = 999/1000
P_B_C2 = 1/1000

# Aplicar el teorema de Bayes
P_C1_B = (P_B_C1 * P_C1) / ((P_B_C1 * P_C1) + (P_B_C2 * P_C2))

# Imprimir el resultado
print(f"La probabilidad de que la carta blanca seleccionada provenga de la Caja 1 es {P_C1_B}")
```

14) Sea X una variable aleatoria discreta tal que

Sea X una variable aleatoria discreta tal que:

$P\{X=n\} = \frac{2}{5^n}\, \forall n \in \mathbb{N} \setminus \{0\}.$

Libro: https://fcefyn.aulavirtual.unc.edu.ar/pluginfile.php/858119/questi

Seleccione una:

- a. E{x}=5/8 ✔
- b. Ninguna de las anteriores. ✖
- c. E{x}=5/2
- d. E{x}=3/2

```
def funcion(n):
```

```
    return(2/(5**n))*n

sumatoria = sum(funcion(n) for n in range(101))

print(sumatoria)
```

15)Para una variable aleatoria X

Para una variable aleatoria X Gaussiana de media cero determine la "función de distribución acumulativa" $F_X(0)$

Respuesta: 0,5

```
import scipy.stats as stats

# Parámetros de la distribución
mu = 0
sigma = 1  # Asumimos una desviación estándar de 1 para una distribución normal estándar

# Calcular la CDF en x = 0
F_0 = stats.norm.cdf(0, mu, sigma)

# Imprimir el resultado
print(f"La función de distribución acumulativa en x = 0 es {F_0}")
```

16)Se define a la función de distribución acumulativa de una variable aleatoria X de la siguiente manera:

Se define a la función de distribución acumulativa de una variable aleatoria X de la siguiente manera:

$F_X(x) = P(X <= x)$, con X una variable real

Para el experimento del lanzamiento de un dado, X es la variable que identifica la cara del dado que salió. Determine $F_X(3.3)$

Respuesta: 0,5

```
import numpy as np

# Parámetros de la distribución
caras_dado = 6

# Calcular la CDF en x = 3.3
x = np.floor(3.3)  # Redondear hacia abajo a entero más cercano
F_x = x / caras_dado

# Imprimir el resultado
print(f"La función de distribución acumulativa en x = 3.3 es {F_x}")
```

17)Obtenga el valor de Q(0). Indicar la respuesta con coma como separador decimal.

Obtenga el valor de Q(0). Indicar la respuesta con coma como separador decimal.

Respuesta: 0.5

```
from scipy.stats import norm

# Calcular Q(0)
Q_0 = norm.sf(0)
# Imprimir el resultado
print(f"Q(0) es {Q_0}")
```

18)En un recinto existen 4 cajas con

¿Será de esta caja?

En un recinto existen 4 cajas con diferentes cantidades de componentes. Cada caja contiene componentes buenos y defectuosos según el detalle del siguiente cuadro:

Distribución de Componentes buenos y defectuosos.

	Buenos	Defectuosos	Total H
Caja 1	1900	100	2000
Caja 2	300	200	500
Caja 3	900	100	1000
Caja 4	900	100	1000
Total V	4000	500	4500

Seleccionando de manera aleatoria la caja y luego extrayendo aleatoriamente un componente, resulta ser un componente defectuoso. ¿cuál es la probabilidad de que el mismo haya sido extraído de la caja 2?

Respuesta: 0,615 ✔

```
# Probabilidades a priori de seleccionar cada caja
P_H = 1/4

# Probabilidad de seleccionar un componente defectuoso de cada caja
P_D_H1 = 100/2000
P_D_H2 = 200/500
P_D_H3 = 100/1000
P_D_H4 = 100/1000

# Probabilidad total de seleccionar un componente defectuoso
P_D = (P_D_H1 + P_D_H2 + P_D_H3 + P_D_H4) * P_H

# Probabilidad de que un componente defectuoso haya sido extraído de la caja 2
P_H2_D = P_D_H2 * P_H / P_D
```

# Imprimir el resultado
print(f"La probabilidad de que un componente defectuoso haya sido extraído de la caja 2 es {P_H2_D}")

19) Sea X una variable uniforme en el intervalo

Sea X una variable aleatoria uniforme en el intervalo [1, 5]. Determine el valor de la función de densidad de probabilidad.

Recuerde que P(1<= X <=5) = 1

Respuesta: 0,25 ✔

# Parámetros de la distribución
a = 1
b = 5
# Calcular la función de densidad de probabilidad
pdf = 1 / (b - a)
# Imprimir el resultado
print(f"La función de densidad de probabilidad es {pdf}")

En un experimento de lanzamiento de un dado se define:

A = { salidas mayores a 3}

B = { salidas pares}

Determine la probabilidad condicional P(A|B)

Exprese el resultado con dos dígitos decimales truncados.

Respuesta: 0,67 ✖

La respuesta correcta es: 0,66

# Probabilidad de B
P_B = 3/6

# Probabilidad de la intersección de A y B
P_A_inter_B = 2/6

# Calcular la probabilidad condicional P(A|B)
P_A_given_B = P_A_inter_B / P_B

# Imprimir el resultado con tres dígitos decimales truncados
print(f"P(A|B) es {P_A_given_B:.3f}")

Diseñar un detector de mentira y estimar estadísticamente su probabilidad de error.

El detector debe decidir en base a la actividad neuronal si una persona está diciendo la verdad o está mintiendo. La actividad neuronal se registra mediante un instrumento que mide el periodo de tiempo y de la onda cerebral. Si la persona está diciendo la verdad, la variable aleatoria y tiene una densidad de probabilidad

$$f_{Y|verdad}(y/verdad) = \alpha e^{-\alpha y}, \quad y \geq 0$$

Mientras que si la persona está diciendo una mentira, la variable aleatoria y tiene una densidad de probabilidad

$$f_{Y|mentira}(y/mentira) = \beta e^{-\beta y}, \quad y \geq 0 \quad \text{con } \alpha < \beta$$

Se supone que la probabilidad a priori de que una persona diga la verdad es p_{true}

1. Realizar una función que modele una fuente de mensajes verdaderos y falsos. Los mensajes verdaderos se representan con 1, y los falsos con 0. Los argumentos de la función son: probabilidad de decir la verdad (p_truth) y cantidad de mensajes (hipótesis) a generar. La función retorna el vector out con los mensajes generados
   ```
   def get_hypothesis(p_true,nb)
       return out
   ```
2. Realizar una función que modele el periodo de las ondas cerebrales mediante una variable aleatoria con función de densidad de probabilidad exponencial $f_{y|verdad}$ y $f_{y|mentira}$. Los argumentos de la función son: el vector de hipótesis generado y los parámetros α y β de la función de densidad correspondiente. La función retorna el vector out que contiene el periodo, y de las ondas cerebrales
   ```
   def get_brain_activity(hypothesis, alfa, beta)
       return out
   ```
   - Ayuda: para generar la variable aleatoria con función de densidad exponencial emplear de la librería numpy el método random.exponential(parametro) donde parámetro es $1/\alpha$ o $1/\beta$ según corresponda
3. Realizar la función que modele el detector de verdad o mentira. La función recibe como argumentos el vector con el periodo de las ondas cerebrales registradas, los parámetros α y β y la probabilidad a priori "p_truth"; y retorna el nivel de decisión (theta) y la hipótesis detectada
   ```
   def detect(input, alpha, beta, p_true
       return [theta, out_hypothesis]
   ```
4. Realizar una función que estime la probabilidad de error del detector de mentiras. Para ello se debe simulara el sistema empleando las funciones anteriores y estimando la probabilidad de error de manera estadística (contando errores)

```
import numpy as np

# Hypothesis source : TRUE (1) False (0)
def get_hypothesis(p_true,nb):
  p_lye = 1 - p_true
  out = np.random.choice(2,nb,p=[p_true,p_lye])
  return out

# Output of brain cell activity instrument
def get_brain_activity(hypothesis, alfa, beta):
  out = [np.random.exponential(1/alfa) if hypothesis[i]==1 else np.random.exponential(1/beta)
for i in range(len(hypothesis)) ]
  return out

# Lyier Detector
def detect(input, alpha, beta, p_true):
  p_lye = 1 - p_true
  theta = 1/(beta-alpha) * np.log( (p_lye*beta)/(p_true*alpha))
  out_hypothesis = [1 if input[i]>theta else 0 for i in range(len(input))]
  return [theta, out_hypothesis]

def simulador_pe(alpha, beta, p_truth, nb):
  hypothesis = get_hypothesis(p_truth,nb)
  observation = get_brain_activity(hypothesis, alpha, beta)
  [theta, hyp_detected] = detect(observation, alpha, beta, p_truth)
  error = [1 if hypothesis[i]!= hyp_detected[i] else 0 for i in range(nb)]
  return (np.sum(error)/nb)
```

Se transmiten los símbolos -1 , 1 , -3, 3 a través de un canal AWGN con *N(0,σ)*.

La probabilidad de transmitir un 1 y un -3 es p_1 y la probabilidad de trasmitir un -1 y un 3 es p_2.

La señal muestreada es y_s.

Escriba una función en python3 *decoder(ys,p1,p2,sigma)* que devuelva el símbolo estimado de máxima verosimilitud.

Ayuda: incluya la librería numphy para acceder a la función logaritmo

Por ejemplo:

Prueba	Resultado
`print(decoder(0.1,0.5,0.5,1))`	1

```
#Dejo comentada una resolucion que utiliza una regla de decision MAP
#la cual creo que no quedo funcionando debido a que creo que no esta logrando
#acceder a las librerias que se importan.
#De todas maneras dejo sin comentar una funcion decoder que utiliza
#una regla de decision ML
"""
from scipy.stats import norm
from scipy.optimize import brentq

#Se define la siguiente funcion que devuelve la interseccion
#entre dos campanas de gauss
def interseccion_gauss(mu1,sigma1,altura1,mu2,sigma2,altura2):
  pdf_1 = lambda x: norm.pdf(x,mu1,sigma1)*altura1
  pdf_2 = lambda x: norm.pdf(x,mu2,sigma2)*altura2

  diff_pdf = lambda x: pdf_1(x) - pdf_2(x)

  #Se busca la interseccion entre las dos campanas por el metodo de la bisectriz
  interseccion =
brentq(diff_pdf,min(mu1-3*sigma1,mu2-3*sigma2),max(mu1+3*sigma1,mu2+3*sigma2))

  return interseccion

def decoder(ys,p1,p2,sigma): #Utiliza regla MAP
  # tu código
  #Utilizando la funcion interseccion_gauss se calculan las intersecciones
  #entre las campanas para poder usarlas como nivel de decision
  primer_interseccion = interseccion_gauss(-3, sigma, p1, -1, sigma, p2)
  segunda_interseccion = interseccion_gauss(-1, sigma, p2, 1, sigma, p1)
  tercera_interseccion = interseccion_gauss(1, sigma, p1, 3, sigma, p2)

  if(ys<=primer_interseccion):
    return -3
  elif(ys<=segunda_interseccion):
    return -1
  elif(ys<=tercera_interseccion):
```

```
        return 1
    else:
        return 3


"""

def decoder(ys,p1,p2,sigma): #En caso de querer utilizar regla ML
    if(ys<=-2):
        return -3
    elif(ys<=0):
        return -1
    elif(ys<=2):
        return 1
    else:
        return 3
```

Implementar un conformador de onda basado en una base ortonormal a desplazamientos múltiplos de T de la familia de raiz de coseno realzado. Sección 5.5 del libro de Bixio

Proveer una función:

```
def raiz_coseno_realzado(simbolos, Beta, T, tt):
   ................................
   return signal_added,signal
```

con los siguiente argumentos:

- símbolos: lista de símbolos de un alfabeto M-PAM
- Beta: exceso de ancho de banda
- T: periodo de símbolo
- tt: intervalo de tiempo

La función retorna:

- signal_added: la forma de onda resultante de los símbolos transmitidos
- signal: matriz (lista) de mxn donde m es la cantidad de símbolos y n el intervalo de tiempo

Ayuda: Si se grafica las salidas de la función se obtendrían señales similares a las mostradas en la figura 5.7 del libro de Bixio (pag. 172)

```
import numpy as np
def filtro_coseno(t,beta,Ts,iT):
        return
1/np.sqrt(Ts)*np.sinc((t-iT)/Ts)*np.cos(np.pi*beta*(t-iT)/Ts)/(1-(2*beta*(t-iT)/Ts)**2)
def raiz_coseno_realzado(simbolos, Beta, T, tt):
        sum_sig = 0
        signal = []

for i in range(len(simbolos)):
        signal.append(simbolos[i]*filtro_coseno(tt,Beta,T,i*T))
        sum_sig += signal[i]
return sum_sig,signal
```

Se transmiten los símbolos -1 y 1 a través de un canal AWGN con $N(0,\sigma)$.

La probabilidad de transmitir un 1 es p_1 y la probabilidad de trasmitir un -1 es p_2.

La señal muestreada es y_s.

Escriba una función en python3 *MAP_decoder(ys,p1,p2,sigma)* que devuelva el símbolo estimado.

Ayuda: incluya la librería numphy para acceder a la función logaritmo

Por ejemplo:

Prueba	Resultado
`print(MAP_decoder(0.1,0.5,0.5,1))`	1

```
import numpy as np
def MAP_decoder(ys,p1,p2,sigma):
        eta=p2/p1
        theta=(sigma**2/2)*np.log(eta)
        if ys> theta:
                return 1
        else:
                return -1
```