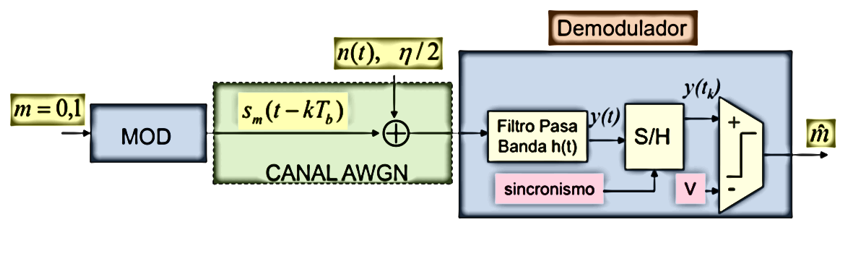
1. **FSK Sunde**

En el proceso de la modulación FSK Sunde la finalidad consiste en extraer los datos binarios transmitidos por medio de la variación de frecuencia. Para dicha demodulación tenemos que tener en consideración los parámetros de la modulación, como son: la frecuencia, la amplitud, entre otros. En la siguiente expresión matemática se mostrará los parámetros elementales y la expresión matemática de la modulación de frecuencia de una señal binaria.

Expresión general de la modulación:

Donde:

Para la modulación de la señal se tiene que tener en cuenta la estructura de la transmisión y recepción de los datos. Lo cual se puede observar en la siguiente figura:

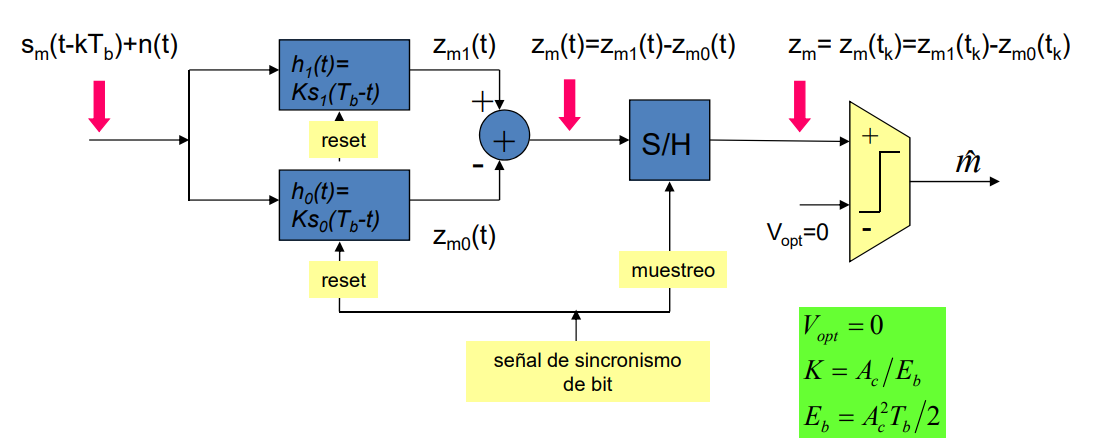


Como se observa la señal transmitida por el modulador pasa por el canal AWGN, el cual adiciona ruido gaussiano a la señal antes de llegar al demodulador. Una vez que la señal haya llegado al demodulador, pasará por un filtro pasa banda, seguido por un muestreo Sample and Hold (S/H) y un comparador. En la modulación existen errores por la imprecisión de los elementos del mismo y el ruido gaussiano introducido. Por ello, en el filtro pasa banda es posible eliminar los errores, lo cual se logra mediante su optimización de probabilidad de error el cual se define por la expresión:

Con la finalidad de minimizar la probabilidad de error mediante herramientas matemáticas, una de ellas la desigualdad de Schwarz. Se obtiene la siguiente expresión:

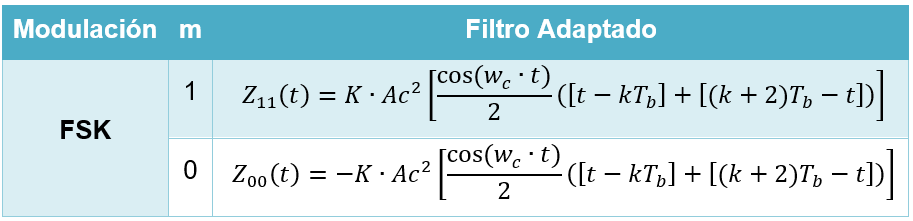
1. **Filtro adaptado:**

Donde nace la primera versión del filtro óptimo llamado adaptado porque representa las posibles señales moduladas como s1, s0 donde su implementación en diagrama de bloques se compone:

****

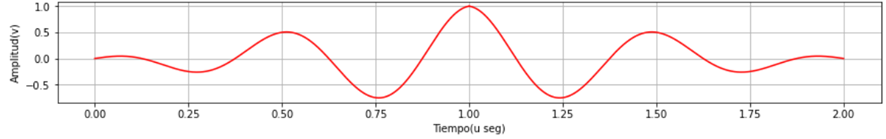
Donde se representa como la señal de entrada y con los filtros. En el dominio del tiempo dichos filtros representan una convolución matemática, la cual se puede representar de la siguiente manera:

Tras resolver dicha convolución, y se pueden simplificar en las expresiones matemáticas:

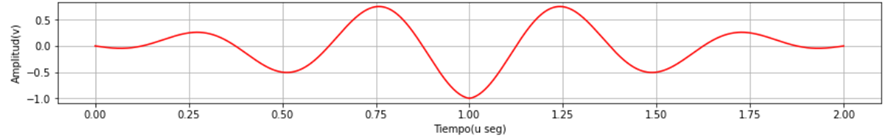


Teniendo como respectivas salidas del filtro:

Para m=1



Para m=0

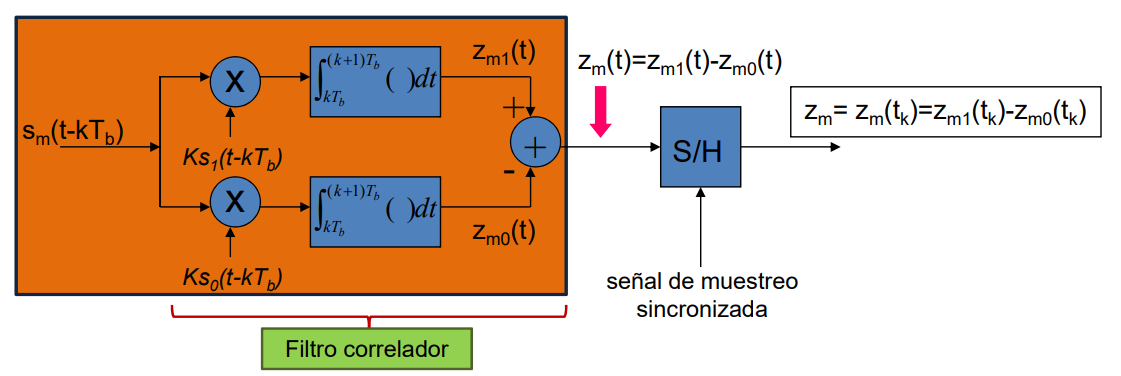


1. **Filtro correlador:**

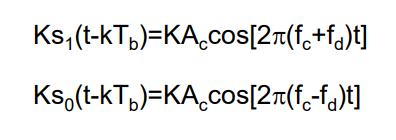
Por otro lado, existe otro tipo de filtro que nace a partir del resultado del filtro adaptado, específicamente a partir de la convolución. Se expresa de la siguiente manera:

Mediante las expresiones matemáticas anteriores y teniendo en cuenta se puede obtener la siguiente expresión matemática:

Con ello se puede implementar mediante un diagrama de bloques de la siguiente manera:

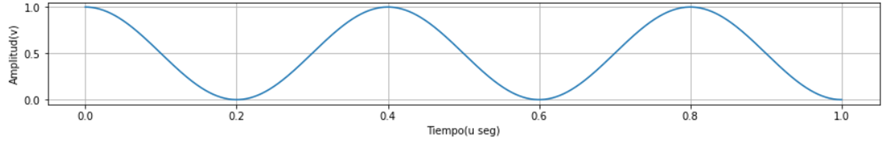


Teniendo en cuenta las siguientes transformaciones:

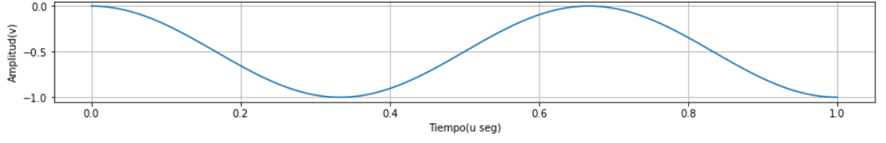


Al entrar la señal y pasar por los multiplicadores cuya grafica de onda tienen las siguientes formas:

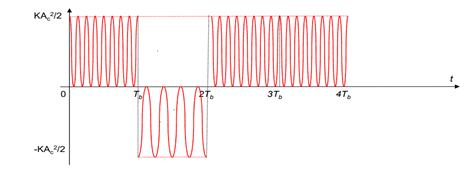
Para :



Para :

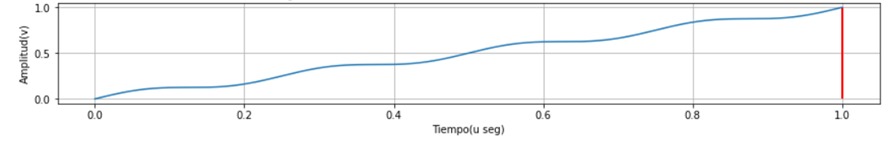


La salida del multiplicador tendrá la siguiente forma:

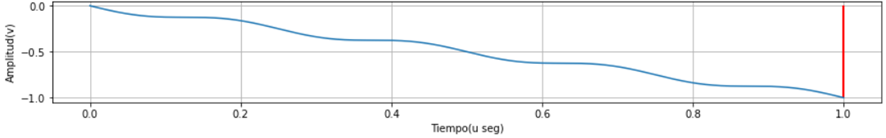


Después del multiplicador, las señales pasan al bloque de integración que transformará dicha onda en una pendiente con la misma forma debido al filtro y al valor de la señal.

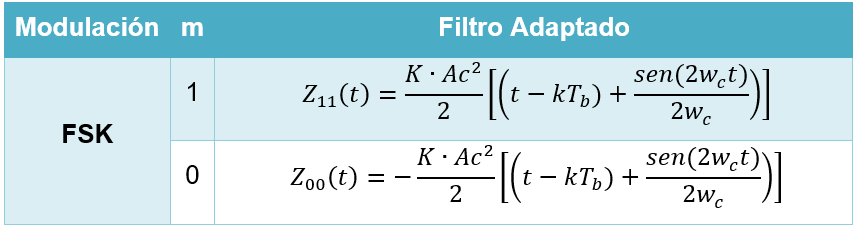
Para :



Para :



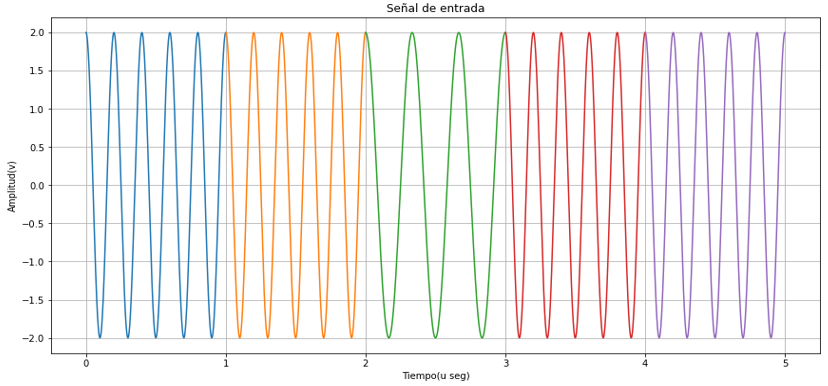
Estas ondas se pueden representar matemáticamente por las siguientes expresiones:



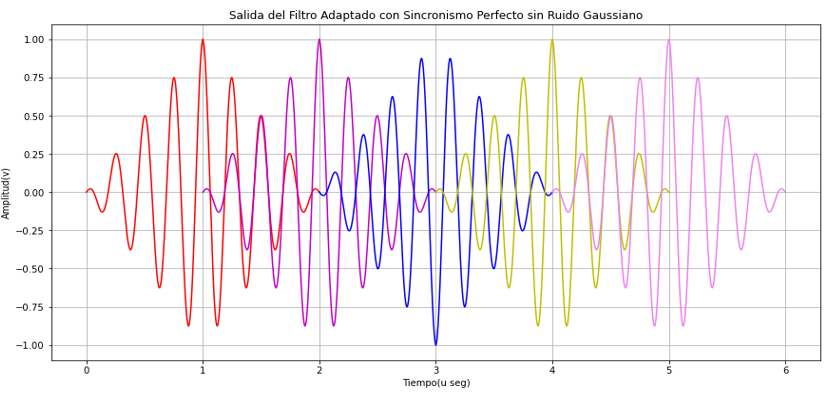
**Desarrollo:**

**Parte 1:**

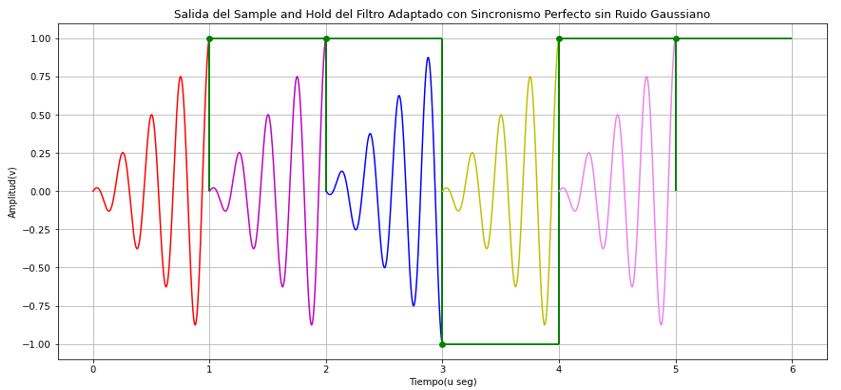
Señal de entrada de ejemplo: “11011”

****

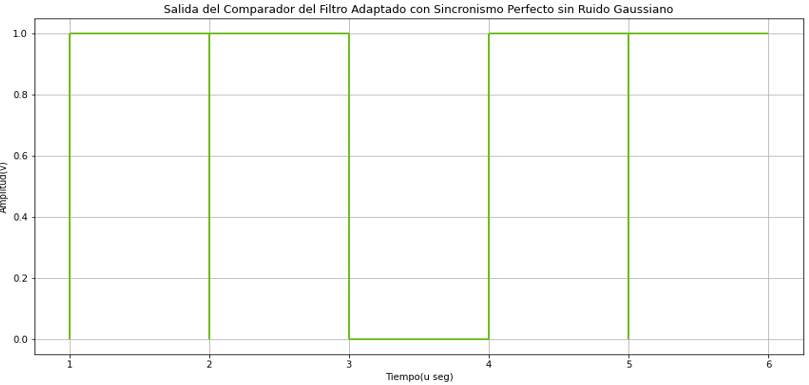
**Filtro adaptivo:**



En esta figura se puede observar la salida del filtro adaptado con sus respectivas interferencias de símbolos (IES)

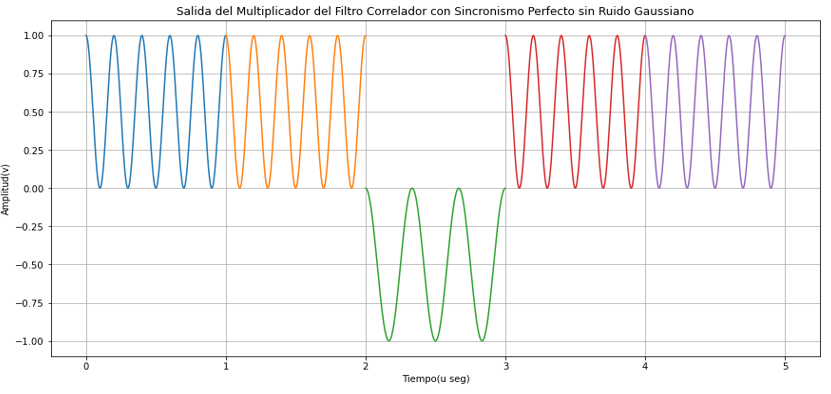


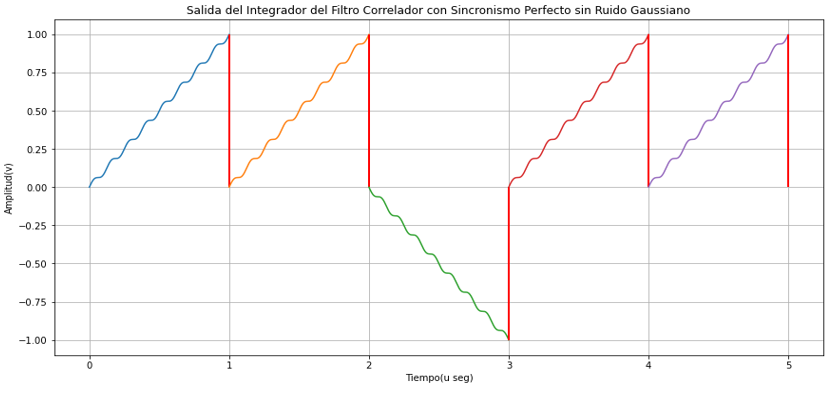
Aquí la señal sale del Sample and Hold y se logra observar la eliminación de las interferencias de símbolos y los valores de la amplitud del muestreo en cada Tb = 1 us.

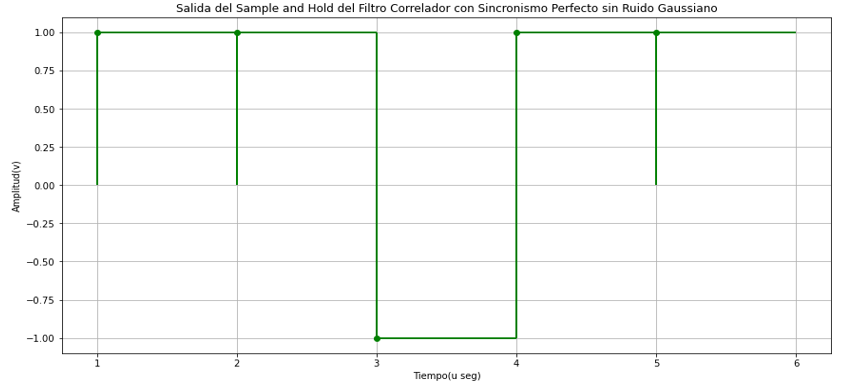


En esta última figura se observa que en la salida del comparador se tiene la señal final binaria tomando en cuenta Vopt = 0.

**Filtro correlador:**

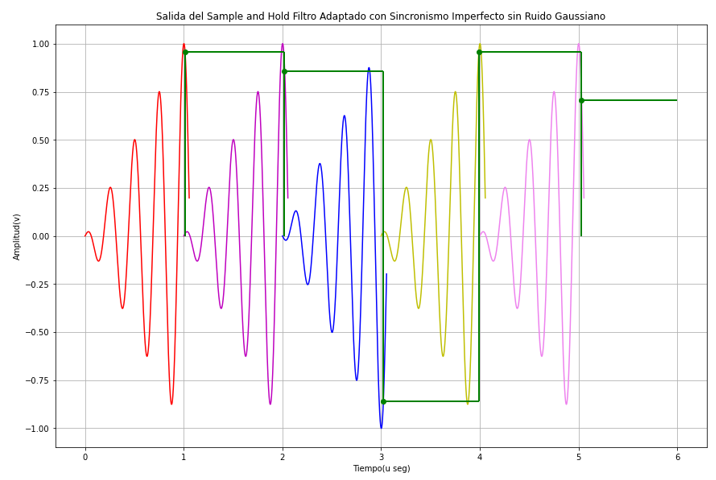
****

****

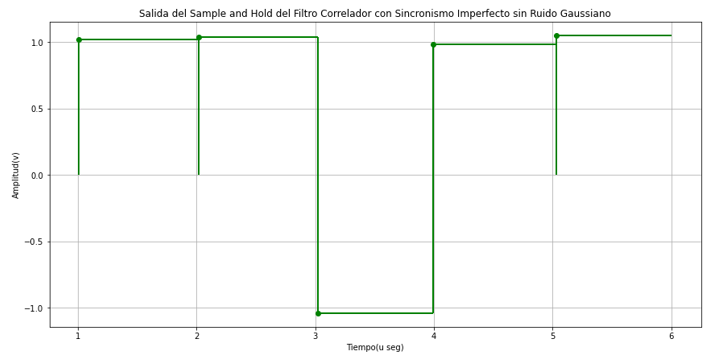


**Parte 2:**

**Filtro adaptado:**

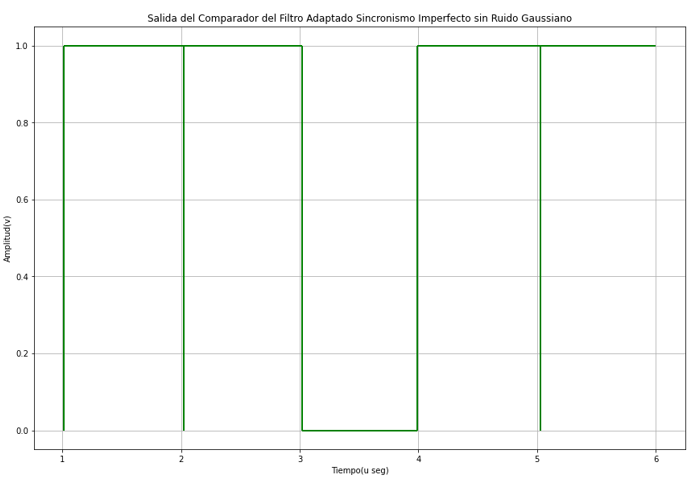


**Filtro Correlador:**

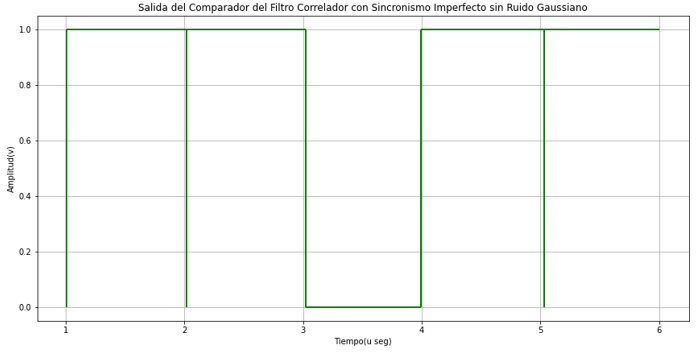


**Parte 3:**

**Filtro adaptado:**

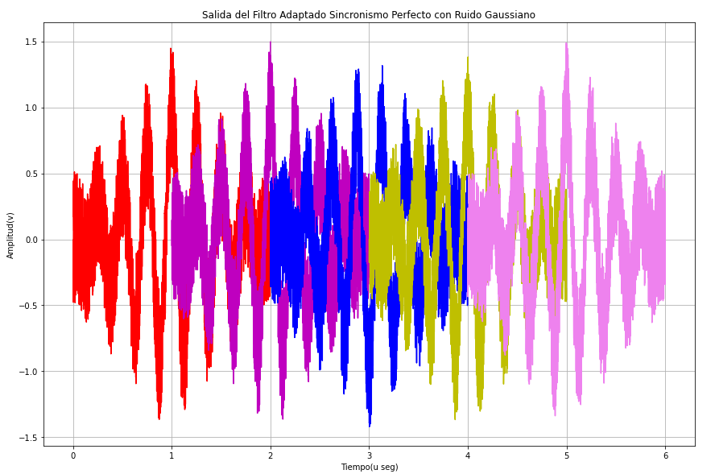


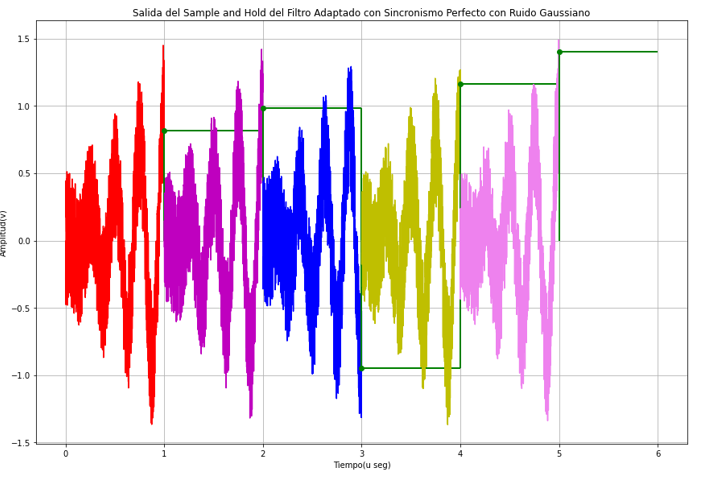
**Filtro correlador:**

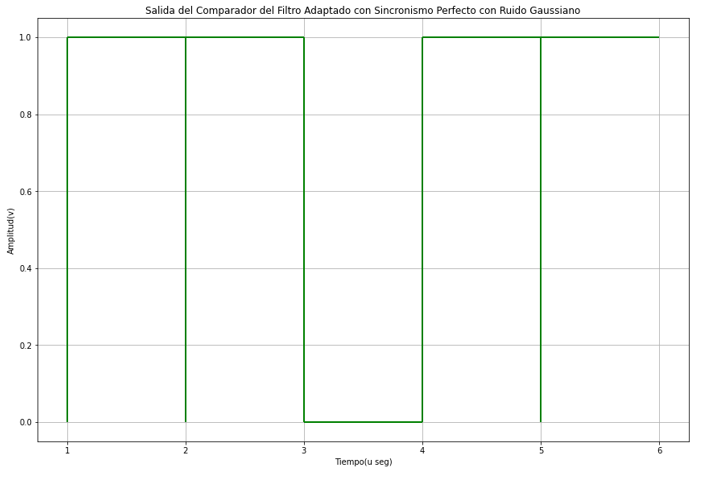


**Parte 4:**

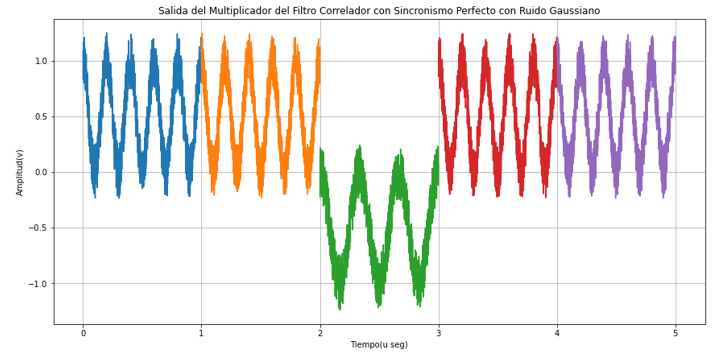
**Filtro adaptado:**

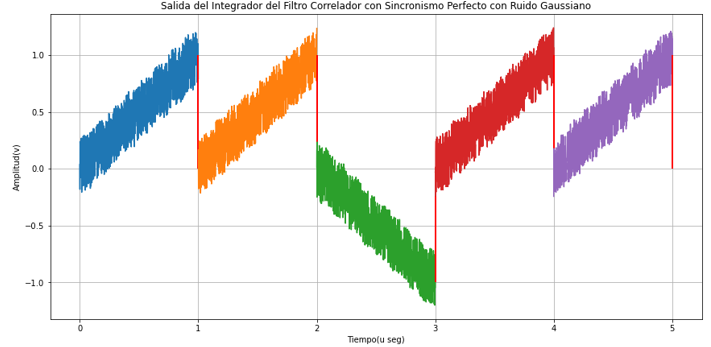
****

****

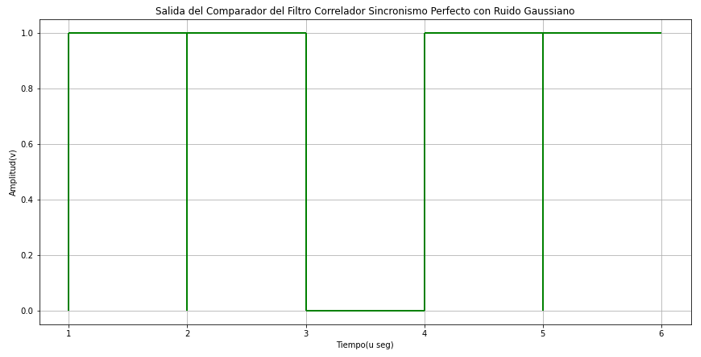
****

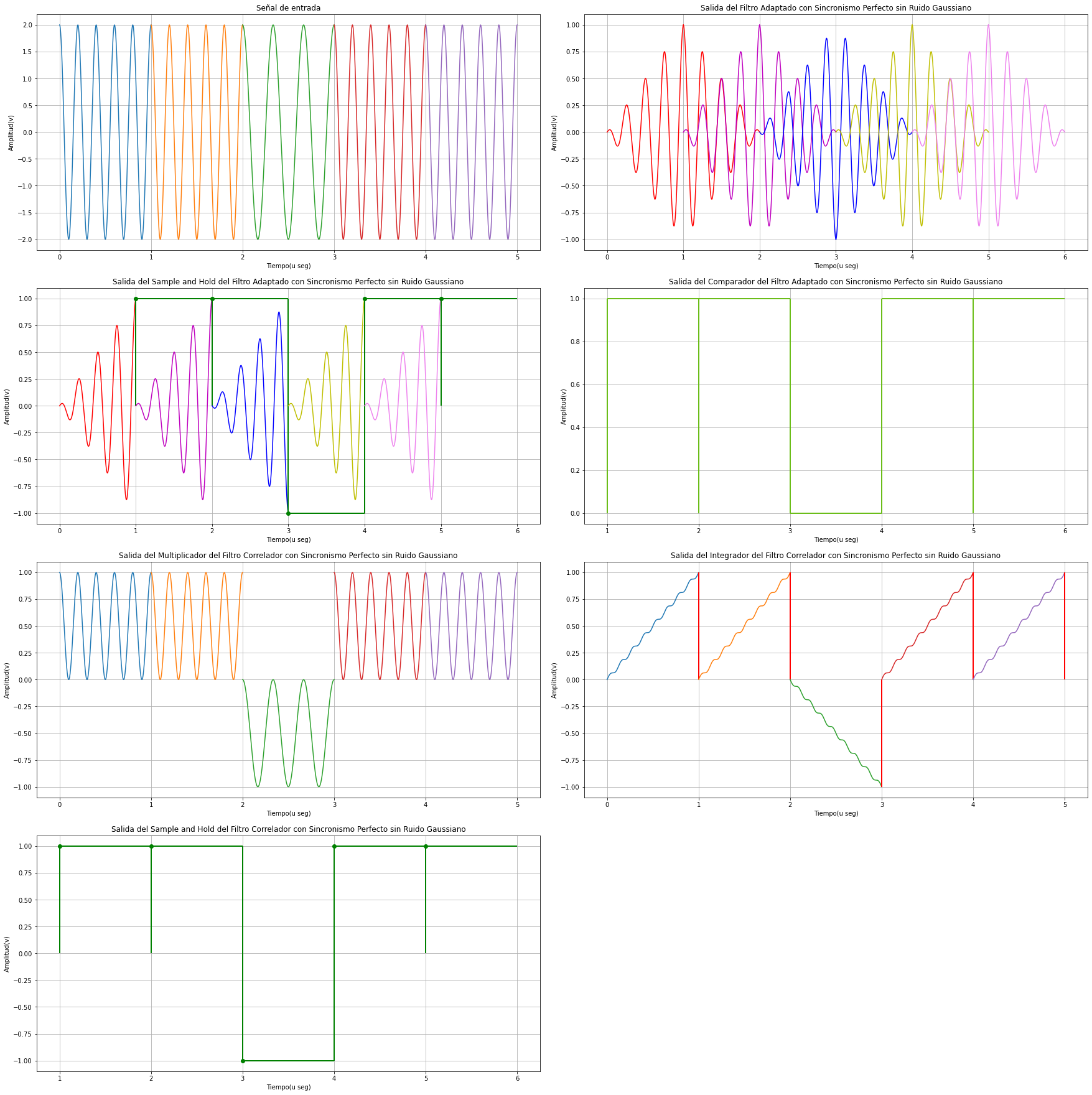
**Filtro correlador:**

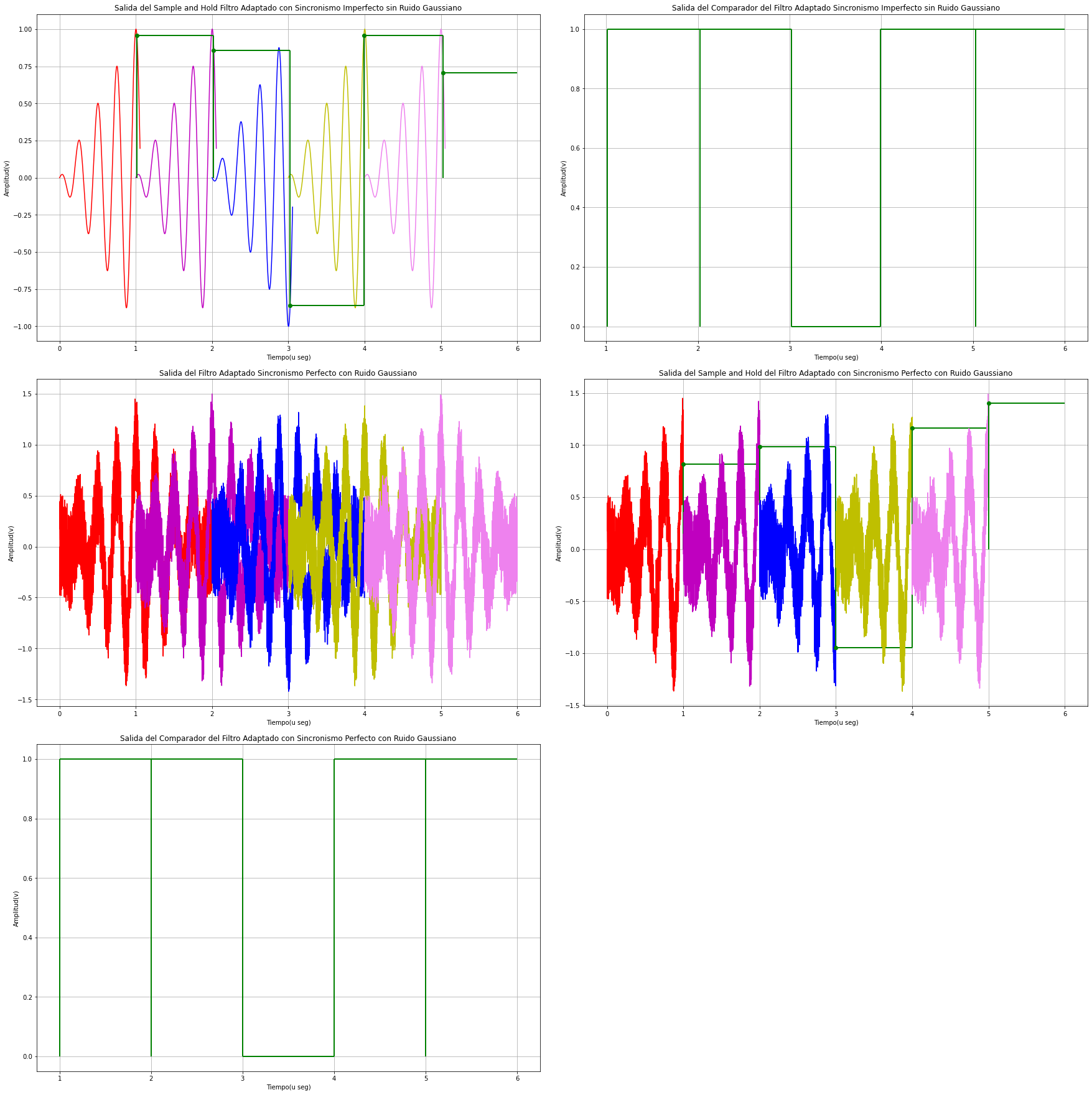
****

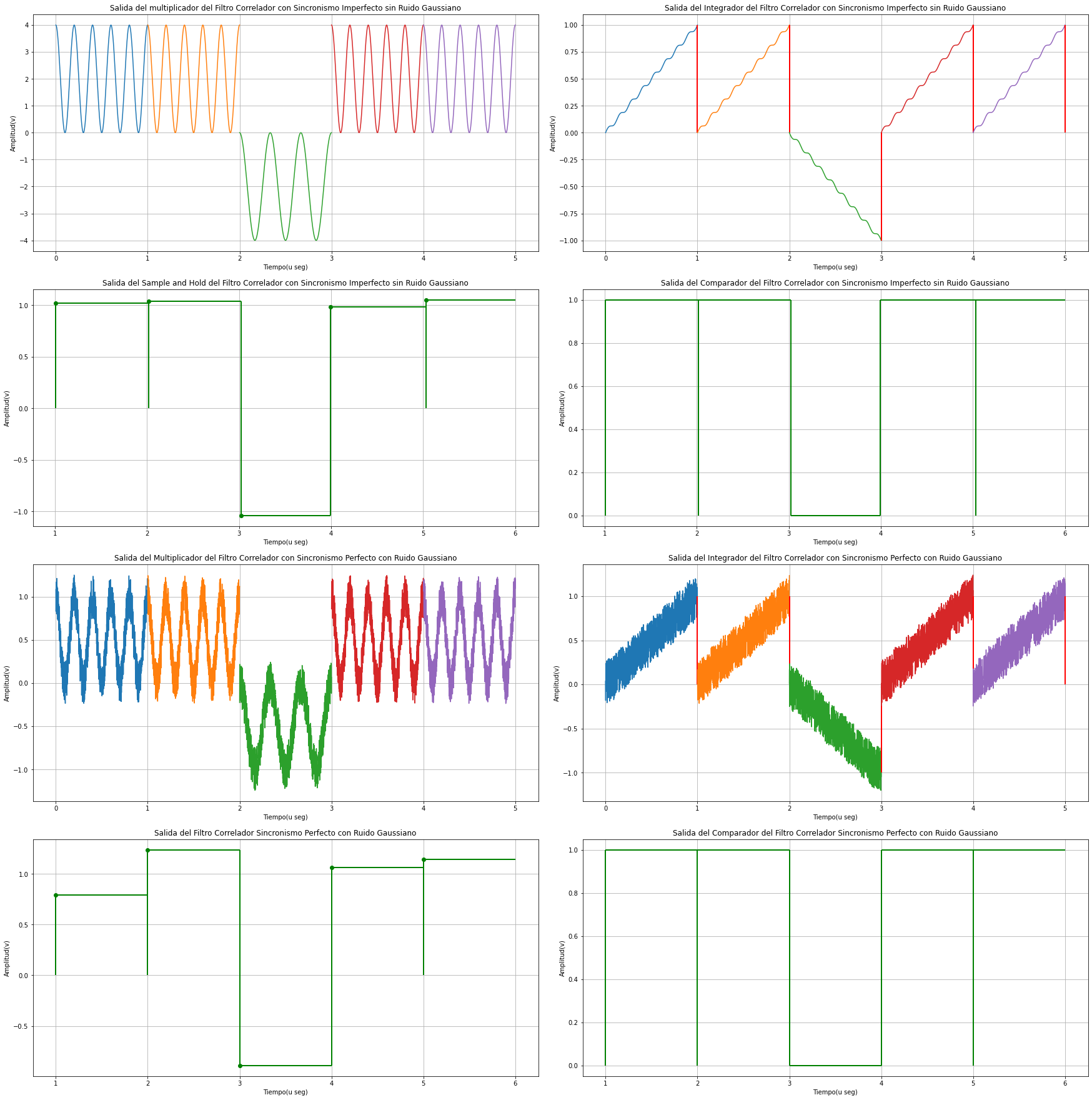
****

****

****







#------------Se importan las librerias para trabajar------------

import matplotlib.pyplot as plt

import math

import numpy as np

import random

#entrada de variables

secuencia=input("Ingrese una cadena de bits: ")

secuencia=str(secuencia)

gammadB=input("Ingrese valor de gamma en dB: ")

gammadB=int(gammadB)

fb=input("ingrese la frecuencia de la se�al: ")

fb=int(fb)

M=2 #Dato

n=1 #Logaritmo base 2 de M

Ac=1 # Amplitud de la portadora

theta=0 #Sin fase

# fb=2 #frecuencia de bit

fs=fb #fb=fs debido a que M=2

Ts=1/fs #Se halla Ts como inversa del periodo fs

fc=fb\*2 #Valor fc

wc=2\*math.pi\*fc #Frecuencia radial

#intervalos=5

TB=Ts/n

#---------Se plotean las graficas de onda---------------

#\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_parte 1\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

fig0, ax=plt.subplots(4,2)

fig0.set\_size\_inches(25,25)

fig0.tight\_layout(pad=4)

#\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_parte 2\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

fig1, ax1=plt.subplots(3,2)

fig1.set\_size\_inches(25,25)

fig1.tight\_layout(pad=4)

#\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_parte 3\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

fig2, ax2=plt.subplots(4,2)

fig2.set\_size\_inches(25,25)

fig2.tight\_layout(pad=4)

#Tomar en cuenta  el final del codigo para saber  que ax# es cada grafica

#--------------------------------datos---------------------------------------

TB=1

K1=2

Ac=1

wc=2\*np.pi\*fc

a=[]

b=[]

cors=["r","m","b","y","violet","brown","k","grey","w","pink","c","g"]

z1=[]

z0=[]

fd=fc/4

K=2

#--------------------------señal de entrada--------------------------

for k in range(0,int(len(secuencia))):

    t=[]

    t=np.linspace(TB\*k,((k+1)\*TB),1000)

    if(int(secuencia[k])==1):

        t=np.linspace(TB\*k,((k+1)\*TB),1000)

        multiplicador1=(K\*Ac\*np.cos(2\*np.pi\*(fc+fd)\*t)) #Entrada de 1

        ax[0,0].plot(t,multiplicador1)

    else:

        t=np.linspace(TB\*k,((k+1)\*TB),1000)

        multiplicador2=(K\*Ac\*np.cos(2\*np.pi\*((fc-fd))\*t)) #Entrada de 0

        ax[0,0].plot(t,multiplicador2)

#\_\_\_\_\_\_\_\_\_\_filtro adaptado  sincronismo perfecto \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

for k in range(0,int(len(secuencia))):

    t=[]

    if(int(secuencia[k])==1): #caso de 1

        #la parte del z0

        t=np.linspace(TB\*k,((k+1)\*TB),1000)

        a=((np.cos(wc\*t))/2)\*(t-k\*TB) #formula

        z=(K1)\*(Ac\*Ac)\*(a)

        z1.append(z)

        ax[0,1].plot(t,z,cors[k]) #ploteo primera mitad de la salida del adaptivo

        #---------------------------graficando el S&H----------------------------------

        ax[1,0].plot(t,z1[0],cors[k]) #salida de adaptivo en grafica de S&H

        ax[1,0].vlines(x=TB\*(k+1), ymin=0, ymax=Ac, linewidth=2, color='g')

        ax[1,0].plot(TB\*(k+1),Ac, marker="o", color="g")

        ax[1,0].hlines(y=Ac, xmin=TB\*(k+1), xmax=TB\*(k+2), linewidth=2, color='g')

        if(k<len(secuencia)-1):

            ax[1,0].vlines(x=TB\*(k+2), ymin=0, ymax=Ac, linewidth=2, color='g')

        #-------------------para la grafica  de la salida------------------------------

        ax[1,1].vlines(x=TB\*(k+1), ymin=0, ymax=1, linewidth=2, color='#6BBD19')

        ax[1,1].hlines(y=1, xmin=TB\*(k+1), xmax=TB\*(k+2), linewidth=2, color='#6BBD19')

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

        if(k<len(secuencia)-1):

            ax[1,1].vlines(x=TB\*(k+2), ymin=0, ymax=1, linewidth=2, color='#6BBD19')

        #----------------segunda mitad de la salida adaptiva------------------------

        t=np.linspace(TB\*(k+1),((k+2)\*TB),1000)

        b=((np.cos(wc\*t))/2)\*((k+2)\*TB-t)

        z=(K1)\*(Ac\*Ac)\*(b)

        z1.append(z)

        ax[0,1].plot(t,z,cors[k])

        #---------------------salida del adaptivo sincronismo imperfecto------------

        #-------------------dato 1--------------------------

        t=np.linspace(TB\*k,(((k+1)\*(TB))+0.055),1000) #desplazamiento diferente a Tb

        a=((np.cos(wc\*t))/2)\*(t-k\*TB)# formula

        z=(K1)\*(Ac\*Ac)\*(a)

        ax1[0,0].plot(t,z,cors[k])

    else: #caso de 0

        #la parte del z1

        t=np.linspace(TB\*k,((k+1)\*TB),1000)

        a=((np.cos(wc\*t))/2)\*(t-k\*TB) #fomula

        z=-(K1)\*(Ac\*Ac)\*(a)

        ax[0,1].plot(t,z,cors[k])

        ax[1,0].plot(t,z,cors[k])

        #------------adaptivo  sincronismo imperfecto--------------

        #--------------------dato 0------------------------------------------

        t=np.linspace(((TB\*k)),((k+1)\*TB)+0.055,1000) #desplazamiento diferente del Tb

        a=((np.cos(wc\*t))/2)\*(t-k\*TB)#ojo formula

        z=-(K1)\*(Ac\*Ac)\*(a)

        ax1[0,0].plot(t,z,cors[k])

        # z=[]

        #\_-----------------------------------------------------------------------------

        #-------------------plotea el S&H--------------------------------

        ax[1,0].vlines(x=TB\*(k+1), ymin=0, ymax=-Ac, linewidth=2, color='g') #trazo linea vertical del S&H

        ax[1,0].plot(TB\*(k+1),-Ac, marker="o", color="g")#trazo de punto  de referencia

        ax[1,0].hlines(y=-Ac, xmin=TB\*(k+1), xmax=TB\*(k+2), linewidth=2, color='g')

        if(k<len(secuencia)-1):

            ax[1,0].vlines(x=TB\*(k+2), ymin=0, ymax=-Ac, linewidth=2, color='g')

        #-------------caso salida de 0---------

        #---------grafica de lineas---------

        ax[1,1].vlines(x=TB\*(k+1), ymin=0, ymax=0, linewidth=2, color='#6BBD19')

        ax[1,1].hlines(y=0, xmin=TB\*(k+1), xmax=TB\*(k+2), linewidth=2, color='#6BBD19')

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

        if(k<len(secuencia)-1):

            ax[1,1].vlines(x=TB\*(k+2), ymin=0, ymax=0, linewidth=2, color='#6BBD19')

        z0.append(z)

        z=[]

        t=np.linspace(TB\*(k+1),((k+2)\*TB),1000)

        b=((np.cos(wc\*t))/2)\*((k+2)\*TB-t) #ojo formula

        z=-(K1)\*(Ac\*Ac)\*(b)

        ax[0,1].plot(t,z,cors[k])

        z0.append(z)

#\_\_\_\_\_\_\_\_filtro correlador sincronismo perfecto\_\_\_\_\_\_\_\_\_

t=[]

z11=[]#para guardar salida de integral 1

z00=[]#para guardar salida de integral 0

for k in range(0,int(len(secuencia))):

    if(int(secuencia[k])==1):#dato 1--------------------------------

        #salida del multiplicador----------------------------

        t=np.linspace(TB\*k,((k+1)\*TB),1000)

        multiplicador1=(K\*Ac\*np.cos(2\*np.pi\*(fc+fd)\*t))/4+Ac/2

        ax[2,0].plot(t,multiplicador1)

        #-----grafica de la salida de la integral------------------------

        z11=((K/2)\*Ac\*\*2)\*((t-k\*TB)+(np.sin(2\*wc\*t))/(2\*wc))

        ax[2,1].plot(t,z11)

        ax[2,1].vlines(x=TB\*(k+1), ymin=0, ymax=Ac, linewidth=2, color='r')

        #-------------salida del S&H--------------caso1

        ax[3,0].vlines(x=TB\*(k+1), ymin=0, ymax=Ac, linewidth=2, color='g')

        ax[3,0].plot(TB\*(k+1),Ac, marker="o", color="g")

        ax[3,0].hlines(y=Ac, xmin=TB\*(k+1), xmax=TB\*(k+2), linewidth=2, color='g')

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

        if(k<len(secuencia)-1):

            ax[3,0].vlines(x=TB\*(k+2), ymin=0, ymax=Ac, linewidth=2, color='g')

    else: #dato 0------------------------------------------------

        t=np.linspace(TB\*k,((k+1)\*TB),1000)

        multiplicador2=-(K\*Ac\*np.cos(2\*np.pi\*((fc-fd))\*t+np.pi))/4-Ac/2

        ax[2,0].plot(t,multiplicador2)

        #------------grafica salida de la integral caso 0-----------

        z00=-((K/2)\*Ac\*\*2)\*((t-k\*TB)+(np.sin(2\*wc\*t))/(2\*wc))

        ax[2,1].plot(t,z00)

        ax[2,1].vlines(x=TB\*(k+1), ymin=0, ymax=-Ac, linewidth=2, color='r')

        #-----------salida del S&H-------------------------------

        ax[3,0].vlines(x=TB\*(k+1), ymin=0, ymax=-Ac, linewidth=2, color='g')

        ax[3,0].plot(TB\*(k+1),-Ac, marker="o", color="g")

        ax[3,0].hlines(y=-Ac, xmin=TB\*(k+1), xmax=TB\*(k+2), linewidth=2, color='g')

        if(k<len(secuencia)-1):

            ax[3,0].vlines(x=TB\*(k+2), ymin=0, ymax=-Ac, linewidth=2, color='g')

#\_\_\_\_\_\_\_\_\_\_\_\_\_SINCRONISMOS IMPERFECTOS\_\_\_\_\_\_\_\_\_\_\_\_\_

tk=[]

epsilon=[]

amplitudes=[]

#\_\_\_\_\_\_\_\_\_\_\_\_\_filtro adaptado  sincronismo imperfecto \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

for k in range(0,len(secuencia)): #obtenemos posiciones en el tiempo a trabajar y los valores de las amplitudes

    epsilon.append(round(random.uniform(-0.04,0.04),2))

    tk.append((k)+TB\*(1+epsilon[k])) #

    ttt=tk[k]

    # a=0

    # b=0

    if(int(secuencia[k])==1):#caso 1--------------------

        if(epsilon[k]<=0): #si es menor o igual que cero se aplica el 'a' porque es la mitad de la trama

            a=((math.cos(wc\*ttt))/2)\*(ttt-k\*TB)

            a=(K1)\*(Ac\*Ac)\*(a)

            amplitudes.append(a)

        else:

            b=((math.cos(wc\*ttt))/2)\*((k+2)\*TB-ttt)

            b=(K1)\*(Ac\*Ac)\*(b)

            amplitudes.append(b)

    else: #caso 0-------------------

        if(epsilon[k]<=0): #si es menor o igual que cero se aplica el 'a' porque es la mitad de la trama

            a=((math.cos(wc\*ttt))/2)\*(ttt-k\*TB)

            a=-(K1)\*(Ac\*Ac)\*(a)

            amplitudes.append(a)

        else:

            b=((math.cos(wc\*ttt))/2)\*((k+2)\*TB-ttt)

            b=-(K1)\*(Ac\*Ac)\*(b)

            amplitudes.append(b)

#----------------se plotea S/H-----------------------------------

for k in range(0,len(secuencia)):

    if(int(secuencia[k])==1):

        #sample and hold

        ax1[0,0].vlines(x=tk[k], ymin=0, ymax=amplitudes[k], linewidth=2, color='g')

        ax1[0,0].plot(tk[k],amplitudes[k], marker="o", color="g")

        if(len(secuencia)-1>k):

            #sample and hold

            ax1[0,0].hlines(y=amplitudes[k], xmin=tk[k], xmax=tk[k+1], linewidth=2, color='g')

            ax1[0,0].vlines(x=tk[k+1], ymin=0, ymax=amplitudes[k], linewidth=2, color='g')

        elif(k==len(secuencia)-1):

            ax1[0,0].hlines(y=amplitudes[k], xmin=tk[k], xmax=(k+2)\*TB, linewidth=2, color='g')

    else:

        #sample and hold

        ax1[0,0].vlines(x=tk[k], ymin=0, ymax=amplitudes[k], linewidth=2, color='g')

        ax1[0,0].plot(tk[k],amplitudes[k], marker="o", color="g")

        if(len(secuencia)-1>k):

            #sample and hold

            ax1[0,0].hlines(y=amplitudes[k], xmin=tk[k], xmax=tk[k+1], linewidth=2, color='g')

            ax1[0,0].vlines(x=tk[k+1], ymin=0, ymax=amplitudes[k],linewidth=2, color='g')

        elif(k==len(secuencia)-1):

            ax1[0,0].hlines(y=amplitudes[k], xmin=tk[k], xmax=(k+2)\*TB, linewidth=2, color='g')

#Dato de comparador

Vop=0

#--------------------------------COMPARADOR--------------------------------------

for k in range(0,len(secuencia)):

    if(amplitudes[k]>=Vop): #si el Vop es superado se considera como 1

        #------------comparador dibujo de 1-------------------------

        ax1[0,1].vlines(x=tk[k], ymin=0, ymax=1, linewidth=2, color='g')

        if(len(secuencia)-1>k):

            #comparador

            ax1[0,1].hlines(y=1, xmin=tk[k], xmax=tk[k+1], linewidth=2, color='g')

            ax1[0,1].vlines(x=tk[k+1], ymin=0, ymax=1, linewidth=2, color='g')

        elif(k==len(secuencia)-1):

            ax1[0,1].hlines(y=1, xmin=tk[k], xmax=(k+2)\*TB, linewidth=2, color='g')

    else:

        #---------------comparador dibujo de 0----------------------

        ax1[0,1].vlines(x=tk[k], ymin=0, ymax=0, linewidth=2, color='g')

        if(len(secuencia)-1>k):

            #comparador

            ax1[0,1].hlines(y=0, xmin=tk[k], xmax=tk[k+1], linewidth=2, color='g')

            ax1[0,1].vlines(x=tk[k+1], ymin=0, ymax=0,linewidth=2, color='g')

        elif(k==len(secuencia)-1):

            ax1[0,1].hlines(y=0, xmin=tk[k], xmax=(k+2)\*TB, linewidth=2, color='g')

#\_\_\_\_\_\_\_\_\_\_\_filtro correlador sincronismo imperfecto\_\_\_\_\_\_\_\_\_\_\_

tk2=[]#para desincronizacion

amplitudes\_correlador=[]

t=[]

z11=[] #pendiente caso 1

z00=[] #pendiente caso 0

for k in range(0,int(len(secuencia))):

    t=np.linspace(TB\*k,((k+1)\*TB),1000)

    tk2.append((k)+TB\*(1+epsilon[k])) #dato sacado anteriormente

    if(int(secuencia[k])==1):#Dato 1

        # ----------grafica  salida de multiplificador-----------------

        multiplicador1=K\*Ac\*np.cos(2\*np.pi\*(fc+fd)\*t)+K\*Ac\*Ac

        ax2[0,0].plot(t,multiplicador1)

        #--------------- grafica salida de integral-------------

        z11=((K/2)\*Ac\*\*2)\*((t-k\*TB)+(np.sin(2\*wc\*t))/(2\*wc))

        ax2[0,1].plot(t,z11)

        ax2[0,1].vlines(x=TB\*(k+1), ymin=0, ymax=Ac, linewidth=2, color='r')

        t=tk2[k]

        z=((K/2)\*Ac\*\*2)\*((t-k\*TB)+(np.sin(2\*wc\*t))/(2\*wc))

        amplitudes\_correlador.append(z) #Dato de S/H

    else:#Dato 0

        # -------grafica de salida de multiplicador---------------

        multiplicador2=-K\*Ac\*np.cos(2\*np.pi\*(fc-fd)\*t+np.pi)-K\*Ac\*Ac

        ax2[0,0].plot(t,multiplicador2)

        # --------grafica de salida de integral-------------------------

        z00=-((K/2)\*Ac\*\*2)\*((t-k\*TB)+(np.sin(2\*wc\*t))/(2\*wc))

        ax2[0,1].plot(t,z00)

        ax2[0,1].vlines(x=TB\*(k+1), ymin=0, ymax=-Ac, linewidth=2, color='r')

        t=tk2[k]

        z=-((K/2)\*Ac\*\*2)\*((t-k\*TB)+(np.sin(2\*wc\*t))/(2\*wc))

        amplitudes\_correlador.append(z) #dato para S/H

    #-------------------------Se plotea S/H-----------------------

for k in range(0,len(secuencia)):

    if(int(secuencia[k])==1): #Dato 1

        #sample and hold

        ax2[1,0].vlines(x=tk2[k], ymin=0, ymax=amplitudes\_correlador[k], linewidth=2, color='g')

        ax2[1,0].plot(tk2[k],amplitudes\_correlador[k], marker="o", color="g")

        if(len(secuencia)-1>k):

            #sample and hold

            ax2[1,0].hlines(y=amplitudes\_correlador[k], xmin=tk2[k], xmax=tk2[k+1], linewidth=2, color='g')

            ax2[1,0].vlines(x=tk2[k+1], ymin=0, ymax=amplitudes\_correlador[k], linewidth=2, color='g')

        elif(k==len(secuencia)-1):

            ax2[1,0].hlines(y=amplitudes\_correlador[k], xmin=tk2[k], xmax=(k+2)\*TB, linewidth=2, color='g')

    else:#Dato 0

        #sample and hold

        ax2[1,0].vlines(x=tk2[k], ymin=0, ymax=amplitudes\_correlador[k], linewidth=2, color='g')

        ax2[1,0].plot(tk2[k],amplitudes\_correlador[k], marker="o", color="g")

        if(len(secuencia)-1>k):

            #sample and hold

            ax2[1,0].hlines(y=amplitudes\_correlador[k], xmin=tk2[k], xmax=tk2[k+1], linewidth=2, color='g')

            ax2[1,0].vlines(x=tk2[k+1], ymin=0, ymax=amplitudes\_correlador[k],linewidth=2, color='g')

        elif(k==len(secuencia)-1):

            ax2[1,0].hlines(y=amplitudes\_correlador[k], xmin=tk2[k], xmax=(k+2)\*TB, linewidth=2, color='g')

#---------------------comparador con sincronismo imperfecto---------------------

for k in range(0,len(secuencia)):

    if(amplitudes\_correlador[k]>=Vop): #si el Vop es superado se considera como 1

        # Grafica de la salida caso 1

        ax2[1,1].vlines(x=tk2[k], ymin=0, ymax=1, linewidth=2, color='g')

        if(len(secuencia)-1>k):

            ax2[1,1].hlines(y=1, xmin=tk2[k], xmax=tk2[k+1], linewidth=2, color='g')

            ax2[1,1].vlines(x=tk2[k+1], ymin=0, ymax=1, linewidth=2, color='g')

        elif(k==len(secuencia)-1):

            ax2[1,1].hlines(y=1, xmin=tk2[k], xmax=(k+2)\*TB, linewidth=2, color='g')

    else:

        #grafica de la salida caso 0

        ax2[1,1].vlines(x=tk2[k], ymin=0, ymax=0, linewidth=2, color='g')

        if(len(secuencia)-1>k):

            #comparador

            ax2[1,1].hlines(y=0, xmin=tk2[k], xmax=tk2[k+1], linewidth=2, color='g')

            ax2[1,1].vlines(x=tk2[k+1], ymin=0, ymax=0,linewidth=2, color='g')

        elif(k==len(secuencia)-1):

            ax2[1,1].hlines(y=0, xmin=tk2[k], xmax=(k+2)\*TB, linewidth=2, color='g')

#####################CASOS CON RUIDO###################

#\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_filtro adaptado sincronismo perfecto con ruido gausiano \_\_\_\_\_\_\_\_\_\_\_

# gammadB=3

gamma=10\*\*(gammadB/10)

varianza=Ac\*Ac/(2\*gamma)

amplitudes\_con\_ruido=[]

for k in range(0,int(len(secuencia))):

    t=[]

    t=np.linspace(TB\*k,((k+1)\*TB),1000)

    if(int(secuencia[k])==1):

        #la parte del z0

        ruido=np.random.uniform(-varianza,+varianza,1000) #generador de se�al con gaussiano

        a=((np.cos(wc\*t))/2)\*(t-k\*TB)+ruido

        z=(K1)\*(Ac\*Ac)\*(a) #ecuacion general

        amplitudes\_con\_ruido.append(z[-1])

        ax1[1,0].plot(t,z,cors[k])

        #------------------------graficando el S&H------------------------------------

        #salida adaptivo

        ax1[1,1].plot(t,z,cors[k])

        #grafica del S/H

        ax1[1,1].vlines(x=TB\*(k+1), ymin=0, ymax=amplitudes\_con\_ruido[k], linewidth=2, color='g')

        ax1[1,1].plot(TB\*(k+1),amplitudes\_con\_ruido[k], marker="o", color="g")

        ax1[1,1].hlines(y=amplitudes\_con\_ruido[k], xmin=TB\*(k+1), xmax=TB\*(k+2), linewidth=2, color='g')

        if(k<len(secuencia)-1):

            ax1[1,1].vlines(x=TB\*(k+2), ymin=0, ymax=amplitudes\_con\_ruido[k], linewidth=2, color='g')

        #----------------la otra mitad de la se�al  adaptiva con ruido-------

        ruido=np.random.uniform(-varianza,+varianza,1000) #generador de se�al con gausssiano

        t=np.linspace(TB\*(k+1),((k+2)\*TB),1000)

        b=((np.cos(wc\*t))/2)\*((k+2)\*TB-t)+ruido

        z=(K1)\*(Ac\*Ac)\*(b)

        ax1[1,0].plot(t,z,cors[k])

    else:

        #la parte del z1

        ruido=np.random.uniform(-varianza,+varianza,1000) #generador de se�al con gaussiano

        a=((np.cos(wc\*t))/2)\*(t-k\*TB)+ruido

        z=-(K1)\*(Ac\*Ac)\*(a) #ecuacion

        amplitudes\_con\_ruido.append(z[-1])

        ax1[1,0].plot(t,z,cors[k])

        #-----------------------plotea el S&H----------------------------

        ax1[1,1].plot(t,z,cors[k])

        ax1[1,1].vlines(x=TB\*(k+1), ymin=0, ymax=amplitudes\_con\_ruido[k], linewidth=2, color='g')

        ax1[1,1].plot(TB\*(k+1),amplitudes\_con\_ruido[k], marker="o", color="g")

        ax1[1,1].hlines(y=amplitudes\_con\_ruido[k], xmin=TB\*(k+1), xmax=TB\*(k+2), linewidth=2, color='g')

        if(k<len(secuencia)-1):

            ax1[1,1].vlines(x=TB\*(k+2), ymin=0, ymax=amplitudes\_con\_ruido[k], linewidth=2, color='g')

        #-----------generador de la otra mitad del adaptivo

        ruido=np.random.uniform(-varianza,+varianza,1000)

        t=np.linspace(TB\*(k+1),((k+2)\*TB),1000)

        b=((np.cos(wc\*t))/2)\*((k+2)\*TB-t)+ruido

        z=-(K1)\*(Ac\*Ac)\*(b)

        ax1[1,0].plot(t,z,cors[k])

        z0.append(z)

            #\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_comparador salida\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Vop=0

for k in range(0,len(secuencia)): #comparador

    if(amplitudes\_con\_ruido[k]>=Vop): #si el Vop es superado se considera como 1

        #comparador

        ax1[2,0].vlines(x=(k+1)\*TB, ymin=0, ymax=1, linewidth=2, color='g')

        if(len(secuencia)-1>k):

            #comparador

            ax1[2,0].hlines(y=1, xmin=(k+1)\*TB, xmax=(k+2)\*TB, linewidth=2, color='g')

            ax1[2,0].vlines(x=(k+2)\*TB, ymin=0, ymax=1, linewidth=2, color='g')

        elif(k==len(secuencia)-1):

            ax1[2,0].hlines(y=1, xmin=(k+1)\*TB, xmax=(k+2)\*TB, linewidth=2, color='g')

    else:

        #comparador

        ax1[2,0].vlines(x=(k+1)\*TB, ymin=0, ymax=0, linewidth=2, color='g')

        if(len(secuencia)-1>k):

            #comparador

            ax1[2,0].hlines(y=0, xmin=(k+1)\*TB, xmax=(k+2)\*TB, linewidth=2, color='g')

            ax1[2,0].vlines(x=(k+2)\*TB, ymin=0, ymax=0,linewidth=2, color='g')

        elif(k==len(secuencia)-1):

            ax1[2,0].hlines(y=0, xmin=(k+1)\*TB, xmax=(k+2)\*TB, linewidth=2, color='g')

#\_\_\_\_\_\_\_\_\_\_\_filtro correlado  sincronismo perfecto con ruido gausiano \_\_\_\_\_\_\_\_

amplitudes\_ruido\_correlador=[]

z11=[]

z00=[]

t=[]

for k in range(0,int(len(secuencia))):

    t=np.linspace(TB\*k,((k+1)\*TB),1000)

    if(int(secuencia[k])==1): #caso 1

        ruido=np.random.uniform(-varianza,+varianza,1000) #generacion del ruido

        multiplicador1=((K\*Ac\*np.cos(2\*np.pi\*(fc+fd)\*t))/4+Ac/2)+ruido

        ax2[2,0].plot(t,multiplicador1)

        #generador de ruido gaussiano salida integral

        t=np.linspace(0,TB,1000)

        ruido=np.random.uniform(-varianza,+varianza,1000)

        z11=((K/2)\*Ac\*\*2)\*((t-K\*TB)+(np.sin(2\*wc\*t))/(2\*wc))+Ac\*2+ruido

        t=np.linspace(TB\*k,((k+1)\*TB),1000)

        amplitudes\_ruido\_correlador.append(z11[-1])

        ax2[2,1].plot(t,z11)

        ax2[2,1].vlines(x=TB\*(k+1), ymin=0, ymax=Ac, linewidth=2, color='r')

    else:#caso 0

        ruido=np.random.uniform(-varianza,+varianza,1000) #generador de ruido

        multiplicador2=-(K\*Ac\*np.cos(2\*np.pi\*(fc-fd)\*t+np.pi))/4-Ac/2+ruido

        ax2[2,0].plot(t,multiplicador2)

        #generador de ruido gaussiano salida integral

        t=np.linspace(0,TB,1000)

        ruido=np.random.uniform(-varianza,+varianza,1000)

        z00=-((K/2)\*Ac\*\*2)\*((t-K\*TB)+(np.sin(2\*-wc\*t))/(2\*-wc))-Ac\*2+ruido

        amplitudes\_ruido\_correlador.append(z00[-1])

        t=np.linspace(TB\*k,((k+1)\*TB),1000)

        ax2[2,1].plot(t,z00)

        ax2[2,1].vlines(x=TB\*(k+1), ymin=0, ymax=-Ac, linewidth=2, color='r')

    #---------------------------------S/H con ruido-------------------------------------------------------------

for k in range(0,len(secuencia)):

    if(int(secuencia[k])==1): #caso 1

        #Grafica  S/H

        ax2[3,0].vlines(x=(k+1)\*TB, ymin=0, ymax=amplitudes\_ruido\_correlador[k], linewidth=2, color='g')

        ax2[3,0].plot((k+1)\*TB,amplitudes\_ruido\_correlador[k], marker="o", color="g")

        if(len(secuencia)-1>k):

            ax2[3,0].hlines(y=amplitudes\_ruido\_correlador[k], xmin=(k+1)\*TB, xmax=(k+2)\*TB, linewidth=2, color='g')

            ax2[3,0].vlines(x=(k+2)\*TB, ymin=0, ymax=amplitudes\_ruido\_correlador[k], linewidth=2, color='g')

        elif(k==len(secuencia)-1):

            ax2[3,0].hlines(y=amplitudes\_ruido\_correlador[k], xmin=(k+1)\*TB, xmax=(k+2)\*TB, linewidth=2, color='g')

    else: #caso 0

        #grafica S/H

        ax2[3,0].vlines(x=(k+1)\*TB, ymin=0, ymax=amplitudes\_ruido\_correlador[k], linewidth=2, color='g')

        ax2[3,0].plot((k+1)\*TB,amplitudes\_ruido\_correlador[k], marker="o", color="g")

        if(len(secuencia)-1>k):

            ax2[3,0].hlines(y=amplitudes\_ruido\_correlador[k], xmin=(k+1)\*TB, xmax=(k+2)\*TB, linewidth=2, color='g')

            ax2[3,0].vlines(x=(k+2)\*TB, ymin=0, ymax=amplitudes\_ruido\_correlador[k],linewidth=2, color='g')

        elif(k==len(secuencia)-1):

            ax2[3,0].hlines(y=amplitudes\_ruido\_correlador[k], xmin=(k+1)\*TB, xmax=(k+2)\*TB, linewidth=2, color='g')

   #----------------------------------COMPARADOR CON GAUSSIANO----------------------------------

for k in range(0,len(secuencia)):

    if(amplitudes\_ruido\_correlador[k]>=Vop): #si el Vop es superado se considera como 1

        #Grafica comparador 1

        ax2[3,1].vlines(x=(k+1)\*TB, ymin=0, ymax=1, linewidth=2, color='g')

        if(len(secuencia)-1>k):

            ax2[3,1].hlines(y=1, xmin=(k+1)\*TB, xmax=(k+2)\*TB, linewidth=2, color='g')

            ax2[3,1].vlines(x=(k+2)\*TB, ymin=0, ymax=1, linewidth=2, color='g')

        elif(k==len(secuencia)-1):

            ax2[3,1].hlines(y=1, xmin=(k+1)\*TB, xmax=(k+2)\*TB, linewidth=2, color='g')

    else:#caso 0

        #grafica comparador 0

        ax2[3,1].vlines(x=(k+1)\*TB, ymin=0, ymax=0, linewidth=2, color='g')

        if(len(secuencia)-1>k):

            ax2[3,1].hlines(y=0, xmin=(k+1)\*TB, xmax=(k+2)\*TB, linewidth=2, color='g')

            ax2[3,1].vlines(x=(k+2)\*TB, ymin=0, ymax=0,linewidth=2, color='g')

        elif(k==len(secuencia)-1):

            ax2[3,1].hlines(y=0, xmin=(k+1)\*TB, xmax=(k+2)\*TB, linewidth=2, color='g')

# print(amplitudes)

# print(amplitudes\_correlador)

# print(tk2)

# print(amplitudes\_con\_ruido)

# print(tk)

# print(epsilon)

#--------------------------------parte1---------------------------------------

ax[0,0].set\_xlabel('Tiempo(u seg)')

ax[0,0].set\_ylabel('Amplitud(v)')

ax[0,0].set\_title("Se�al de entrada")

ax[0,0].grid()

ax[0,1].set\_xlabel('Tiempo(u seg)')

ax[0,1].set\_ylabel('Amplitud(v)')

ax[0,1].set\_title("Salida del Filtro Adaptado con Sincronismo Perfecto sin Ruido Gaussiano")

ax[0,1].grid()

ax[1,0].set\_xlabel('Tiempo(u seg)')

ax[1,0].set\_ylabel('Amplitud(v)')

ax[1,0].set\_title("Salida del Sample and Hold del Filtro Adaptado con Sincronismo Perfecto sin Ruido Gaussiano")

ax[1,0].grid()

ax[1,1].set\_xlabel('Tiempo(u seg)')

ax[1,1].set\_ylabel('Amplitud(v)')

ax[1,1].set\_title("Salida del Comparador del Filtro Adaptado con Sincronismo Perfecto sin Ruido Gaussiano")

ax[1,1].grid()

ax[2,0].set\_xlabel('Tiempo(u seg)')

ax[2,0].set\_ylabel('Amplitud(v)')

ax[2,0].set\_title("Salida del Multiplicador del Filtro Correlador con Sincronismo Perfecto sin Ruido Gaussiano")

ax[2,0].grid()

ax[2,1].set\_xlabel('Tiempo(u seg)')

ax[2,1].set\_ylabel('Amplitud(v)')

ax[2,1].set\_title('Salida del Integrador del Filtro Correlador con Sincronismo Perfecto sin Ruido Gaussiano')

ax[2,1].grid()

ax[3,0].set\_xlabel('Tiempo(u seg)')

ax[3,0].set\_ylabel('Amplitud(v)')

ax[3,0].set\_title("Salida del Sample and Hold del Filtro Correlador con Sincronismo Perfecto sin Ruido Gaussiano")

ax[3,0].grid()

#-----------------------------------------parte2------------------------------

ax1[0,0].set\_xlabel('Tiempo(u seg)')

ax1[0,0].set\_ylabel('Amplitud(v)')

ax1[0,0].set\_title('Salida del Sample and Hold Filtro Adaptado con Sincronismo Imperfecto sin Ruido Gaussiano')

ax1[0,0].grid()

ax1[0,1].set\_xlabel('Tiempo(u seg)')

ax1[0,1].set\_ylabel('Amplitud(v)')

ax1[0,1].set\_title('Salida del Comparador del Filtro Adaptado Sincronismo Imperfecto sin Ruido Gaussiano')

ax1[0,1].grid()

ax1[1,0].set\_xlabel('Tiempo(u seg)')

ax1[1,0].set\_ylabel('Amplitud(v)')

ax1[1,0].set\_title('Salida del Filtro Adaptado Sincronismo Perfecto con Ruido Gaussiano')

ax1[1,0].grid()

ax1[1,1].set\_xlabel('Tiempo(u seg)')

ax1[1,1].set\_ylabel('Amplitud(v)')

ax1[1,1].set\_title('Salida del Sample and Hold del Filtro Adaptado con Sincronismo Perfecto con Ruido Gaussiano')

ax1[1,1].grid()

ax1[2,0].set\_xlabel('Tiempo(u seg)')

ax1[2,0].set\_ylabel('Amplitud(v)')

ax1[2,0].set\_title('Salida del Comparador del Filtro Adaptado con Sincronismo Perfecto con Ruido Gaussiano')

ax1[2,0].grid()

#---------------------------------parte3---------------------------------------

ax2[0,0].set\_xlabel('Tiempo(u seg)')

ax2[0,0].set\_ylabel('Amplitud(v)')

ax2[0,0].set\_title('Salida del multiplicador del Filtro Correlador con Sincronismo Imperfecto sin Ruido Gaussiano')

ax2[0,0].grid()

ax2[0,1].set\_xlabel('Tiempo(u seg)')

ax2[0,1].set\_ylabel('Amplitud(v)')

ax2[0,1].set\_title('Salida del Integrador del Filtro Correlador con Sincronismo Imperfecto sin Ruido Gaussiano')

ax2[0,1].grid()

ax2[1,0].set\_xlabel('Tiempo(u seg)')

ax2[1,0].set\_ylabel('Amplitud(v)')

ax2[1,0].set\_title('Salida del Sample and Hold del Filtro Correlador con Sincronismo Imperfecto sin Ruido Gaussiano')

ax2[1,0].grid()

ax2[1,1].set\_xlabel('Tiempo(u seg)')

ax2[1,1].set\_ylabel('Amplitud(v)')

ax2[1,1].set\_title('Salida del Comparador del Filtro Correlador con Sincronismo Imperfecto sin Ruido Gaussiano')

ax2[1,1].grid()

ax2[2,0].set\_xlabel('Tiempo(u seg)')

ax2[2,0].set\_ylabel('Amplitud(v)')

ax2[2,0].set\_title('Salida del Multiplicador del Filtro Correlador con Sincronismo Perfecto con Ruido Gaussiano')

ax2[2,0].grid()

ax2[2,1].set\_xlabel('Tiempo(u seg)')

ax2[2,1].set\_ylabel('Amplitud(v)')

ax2[2,1].set\_title('Salida del Integrador del Filtro Correlador con Sincronismo Perfecto con Ruido Gaussiano')

ax2[2,1].grid()

ax2[3,0].set\_xlabel('Tiempo(u seg)')

ax2[3,0].set\_ylabel('Amplitud(v)')

ax2[3,0].set\_title('Salida del Filtro Correlador Sincronismo Perfecto con Ruido Gaussiano')

ax2[3,0].grid()

ax2[3,1].set\_xlabel('Tiempo(u seg)')

ax2[3,1].set\_ylabel('Amplitud(v)')

ax2[3,1].set\_title('Salida del Comparador del Filtro Correlador Sincronismo Perfecto con Ruido Gaussiano')

ax2[3,1].grid()

#------------------------eliminacion de graficas vacias-----------------------

fig0.delaxes(ax[3,1])

fig1.delaxes(ax1[2,1])

**Conclusiones:**

El filtro adaptivo es más inestable ante la desincronización como podemos notar en las gráficas anteriores ya que la variación del punto a analizar para aplicar S/H hay una clara diferencia en la ubicación del punto a analizar si lo comparamos con el filtro correlacionado que a pesar que también varia el punto a analizar para el S/H no es tan exagerado como lo es en el filtro adaptivo

Para una FSK no afecta mucho el ruido gaussiano a la hora de aplicar S/H en comparación con otras modulaciones. Sin embargo, hay que tener cuidado con la el de sincronismo den el caso del filtro adaptivo ya que podría llegar a tener errores de bit.