Amplitude-shift keying (ASK)

```
#Amplitude-shift keying (ASK)


import matplotlib as plt
import numpy as np
import pylab as pl
from math import pi
import random as gb

v=[0,0,1,1,0,1,1,0,0,1,0]                              # Digital Input
Data Of A Baseband Signal
dim=100                                                # 1D Array
Dimension(1x100)
Vx=[]                                                  #Declare A List
for i in range(1,11):
    f=np.ones(dim)                                     #
https://www.journaldev.com/32792/numpy-ones-in-python
    x=f*v[i]
    Vx=np.concatenate((Vx,x))

pl.subplot(3,1,1)                                      #
https://matplotlib.org/3.1.1/api/_as_gen/matplotlib.pyplot.subplot.htm
l
pl.title(r'$Data:0,0,1,1,0,1,1,0,0,1,0 $')
pl.plot(Vx ,'y')

dim2=len(Vx)                                           # Same Dimension
Of The Length Of The Baseband Signal (Vx)
t=np.linspace(0,5,dim2)
f1=5                                                   # Frecuency
pl.subplot(3,1,2)                                      #
https://matplotlib.org/3.1.1/api/_as_gen/matplotlib.pyplot.subplot.htm
l
w1=2*np.pi*f1*t
y1=np.cos(w1)                                          # Create A
cosine Signal
pl.title(r'$C(t)=A\cos(2 \pi f)$')
#pl.plot(t,y1, '-r', label='cosine')
#pl.legend(loc='upper right')
pl.plot(t,y1,'-r')
#pl.show()
pl.subplot(3,1,3)                                      #
https://matplotlib.org/3.1.1/api/_as_gen/matplotlib.pyplot.subplot.htm
```

```
l
mult=(Vx*y1)                                                    # Create A ASK
Signal By Multiplying Baseband Signal(Vx) And A Carrier Signal(y1)
pl.title(r'$ASK$')
pl.plot(t,mult ,'g')
pl.xlabel('fig:Amplitude Shift Keying(ASK)')
pl.show()                                                       # Showing The
Resulting 3 Signals In One Graph
```

Frequency-shift keying (FSK)

```
# Frequency-shift keying (FSK)

import matplotlib as plt
import numpy as np
import pylab as pl
from math import pi
import random as gb

v=[0,0,1,1,0,1,1,0,0,1,0]                                        # Digital Input
Data Of A Baseband Signal

dim=100                                                          # 1D Array
Dimension(1x100)

Vx=[]                                                            # Declare A
List

for i in range(1,11):
    f=np.ones(dim)                                               #
https://www.journaldev.com/32792/numpy-ones-in-python
    x=f*v[i]
    Vx=np.concatenate((Vx,x))

pl.subplot(4,1,1)                                                #
https://matplotlib.org/3.1.1/api/_as_gen/matplotlib.pyplot.subplot.htm
l
pl.title(r'$Data:0,0,1,1,0,1,1,0,0,1,0 $')
pl.plot(Vx,'y')


dim2=len(Vx)                                                     # Same
Dimension Of The Length Of The Baseband Signal (Vx)
t=np.linspace(0,5,dim2)
```

```
f1=5                                                        # Frecuency
(Hz)
pl.subplot(4,1,2)                                           #
https://matplotlib.org/3.1.1/api/_as_gen/matplotlib.pyplot.subplot.htm
l
w1=2*np.pi*f1*t
y1=np.cos(w1)                                               # Create A
cosine Signal
#pl.xlabel('x-axsis')
#pl.ylabel('y-axsis')
#pylab.plot(x, y2, 'm', label='cosine')
#pylab.legend(loc='upper right')
pl.title(r'$s1(t)=A\cos(2 \pi f1)$')
pl.plot(t,y1, '-r', label='cosine')
pl.legend(loc='best')

pl.subplot(4,1,3)                                           #
https://matplotlib.org/3.1.1/api/_as_gen/matplotlib.pyplot.subplot.htm
l
#mult=(Vx*y1)
#pl.plot(t,mult)
#pl.show()
f2=20                                                       # Frecuency(Hz)
w2=2*np.pi*f2*t
y2=np.cos(w2)                                               # Create A
cosine Signal
pl.plot(t,y2 ,'m',label='cosine')
pl.legend(loc='best')
pl.title(r'$s2(t)=A\cos(2 \pi f2)$')
#pl.plot(x, y2, 'm', label='cosine')

pl.subplot(4,1,4)                                           #
https://matplotlib.org/3.1.1/api/_as_gen/matplotlib.pyplot.subplot.htm
l

Di=[]                                                       # Declare A
List

for i in range(0,dim2):

    if Vx[i]==0:
        x=np.array([y1[i]])
        Di=np.concatenate((Di,x))
    else:
        y=np.array([y2[i]])
        Di=np.concatenate((Di,y))
```

```
pl.title(r'$S(t)=A\cos(2 \pi f1) + A\cos(2 \pi f2) $')
pl.xlabel('fig: FSK (Frequincy Shifting Key)')
pl.plot(t,Di ,'b')

pl.show()                                                    # Showing The
Resulting 3 Signals In One Graph
```

Phase-shift keying (PSK)

```
# Phase-shift keying (PSK)

import matplotlib as plt
import numpy as np
import pylab as pl
from math import pi
import random as gb

                                                                  # Digital
Input Data Of A Baseband Signal
v=[0,0,1,1,0,1,1,0,0,1,0,0,1,1,1,0,1,0,1,1,0,0,0,1,1,0,1]

dim=100                                                           # 1D Array
Dimension(1x100)

Vx=[]                                                             # Declare A
List
Di=[]                                                             # Declare A
List

for i in range(1,27):
    f=np.ones(dim)                                                #
https://www.journaldev.com/32792/numpy-ones-in-python
    x=f*v[i]
    Vx=np.concatenate((Vx,x))

pl.subplot(4,1,1)                                                 #
https://matplotlib.org/3.1.1/api/_as_gen/matplotlib.pyplot.subplot.htm
l
pl.title(r'$Data:0,0,1,1,0,1,1,0,0,1,0,0,1,1,1,0,1,0,1,1,0,0,0,1,1,0,1
$')
pl.plot(Vx,'y')

dim2=len(Vx)                                                      # Same
```

```
Dimension Of The Length Of The Baseband Signal (Vx)
t=np.linspace(0,5,dim2)
f1=4                                                           # Frecuency
(Hz)
pl.subplot(4,1,2)                                              #
https://matplotlib.org/3.1.1/api/_as_gen/matplotlib.pyplot.subplot.htm
l
w1=2*np.pi*f1*t
y1=np.cos(w1)                                                  # Create A
cosine Signal
#pl.xlabel('x-axsis')
pl.ylabel('Amplitude(A)')
#pylab.plot(x, y2, 'm', label='cosine')
#pylab.legend(loc='upper right')
#pl.title(r'$s1(t)=A\cos(2 \pi f1)$')
pl.title(r'$s1(t)=A\cos(2 \pi f1)$')
pl.plot(t,y1, '-r', label='cosine')
pl.legend(loc='best')                                          # Plot The
Signal (y1) in The Graph


f2=4                                                           # Frecuency
(Hz)
pl.subplot(4,1,3)                                              #
https://matplotlib.org/3.1.1/api/_as_gen/matplotlib.pyplot.subplot.htm
l
#mult=(Vx*y1)
#pl.plot(t,mult)
#pl.show()
w2=2*np.pi*f2*t
y2=np.sin(w2)                                                  # Create A
sine Signal
                                                               # Plot The
Signal (y2) in The Graph
pl.plot(t,y2 ,'m',label='sine')
pl.legend(loc='best')
pl.title(r'$s2(t)=A\sin(2 \pi f2)$')



pl.subplot(4,1,4)                                              #
https://matplotlib.org/3.1.1/api/_as_gen/matplotlib.pyplot.subplot.htm
l
res=((y2*Vx)-(y1*Vx) + (y1))
pl.plot(t,res,'g')                                             # Plot The
Resultant Signal (res) in The Graph
```

```
pl.title(r'$S(t)=A\cos(2 \pi f1) + A\sin(2 \pi f2) $')
pl.xlabel('fig: PSK (Phase Shifting Key)')

pl.show()                                                # Showing
The Resulting 3 Signals In One Graph
```

Frequency-division multiplexing (FDM)

```
# Frequency-division multiplexing (FDM)

import numpy as np
import pylab as pl
from scipy import signal
import matplotlib.pyplot as plt
from math import pi

t = np.linspace(0, 1,500)

Am1=1                                                   #Amplitude
Of The First Signal (m1)
fm1=3                                                   #Frequency
Of The First Signal(Hz)
m1=Am1*np.sin(2*np.pi*fm1*t)                            #Create The
First Sinusoidal Signal (m1)
pl.figure(1)                                            #Mark The
Signal(m3) as a Figure 1
pl.plot(t,m1,'r')                                       #Plot The
First (m1) Signal
                                                        #Lable The
First Signal (m1)
pl.xlabel('Absolute Frequincy')
pl.ylabel('DFT Values')
pl.title('Signal 1 with frequency 3Hz')
pl.show()                                               # Showing
The First Signal(m1)


Am2=2                                                   #Amplitude
Of The Second Signal (m2)
fm2=4                                                   #Frequency
Of The Second Signal(Hz)
m2=Am2*np.sin(2*np.pi*fm2*t )                           #Create The
Second Sinusoidal Signal (m2)
pl.figure(2)                                            #Mark The
```

```
Signal(m2) as a Figure 2
pl.plot(t,m2,'g')                                                        #Plot The
Second Signal(m2)
                                                                         #Lable The
Second Signal (m2)
pl.xlabel('Absolute Frequincy')
pl.ylabel('DFT Values')
pl.title('Signal 2 with frequency 4Hz')
pl.show()                                                                #Showing The
Second Signal(m2)

Am3=0.18                                                                 #Amplitude
Of The Third Signal (m3)
fm3=5                                                                    #Frequency
Of The Third Signal(Hz)
m3=Am3*np.cos(2*np.pi*fm3*t)                                             #Create The
Third cosine Signal (m3)
pl.figure(3)                                                             #Mark The
Signal(m3) as a Figure 3
                                                                         #Lable The
Third Signal (m3)
pl.plot(t,m3,'b')                                                        #Plot The
Third Signal(m3)
                                                                         #Lable The
Third Signal (m3)
pl.xlabel('Absolute Frequincy')
pl.ylabel('DFT Values')
pl.title('Signal 3 with frequency 5Hz')
pl.show()                                                                # Showing
The Third Signal(m3)

Ac1=2                                                                    #Amplitude
Of The First Carrier Signal (c1)
fc1=100                                                                  #Frequency
Of The First Carrier Signal(Hz)
c1=Ac1*np.sin(2*np.pi*fc1*t)                                             #Create The
First Carrier Sine Signal (c1)

Ac2=2                                                                    #Amplitude
Of The Second Carrier Signal (c2)
fc2=150                                                                  #Frequency
Of The Second Carrier Signal(Hz)
c2=Ac2*np.sin(2*np.pi*fc2*t)                                             #Create The
Second Carrier Sine Signal (c2)

Ac3=2                                                                    #Amplitude
```

```
Of The Third Carrier Signal (c3)
fc3=60                                                         #Frequency
Of The Third Carrier Signal(Hz)
c3=Ac3*np.sin(2*np.pi*fc3*t)                                   #Create The
Third Carrier Sine Signal (c3)

x1=m1*c1 + m2*c2 + m3*c3                                       #Create The
Composite Signal(x1) With
                                                               # Three
Carrier Signals(m1*c1,m2*c2,m3*c3)


pl.figure(4)                                                   #Mark The
Signal(x1) as a Figure 4
pl.plot(t,x1 ,'c')                                             #Plot The
Signal(x1)
                                                               #Lable The
Signal (x1)
pl.xlabel('Time')
pl.ylabel('Amplitude')
pl.title('Composite Signal of signals -1 ,2 ,3 ')
pl.show()                                                      #Showing The
Signal (x1)


x=m1 + m2 + m3                                                 #Create The
Composite Signal(x) With
                                                               # Three
Baseband Signals(m1,m2,m3)
xn = x + np.random.randn(len(t)) * 0.08
pl.figure(5)                                                   #Mark The
Signal(xn) as a Figure 5
pl.plot(t,xn,'m')                                              #Plot The
Signal(xn)
                                                               #Lable The
Signal (xn)
pl.xlabel('Absolute Frequincy')
pl.ylabel('DFT Values')
pl.title('Specturm of Composite Signal(signal-1 ,2 ,3)')
pl.show()                                                      #Showing The
Signal (xn)


pl.figure(6)                                                   #Mark The
Signal (m1*c1) as a Figure 6
pl.plot(t,m1*c1 ,'y')                                          #Plot The
```

```
Signal(m1*c1)
                                                          #Lable The
Signal (m1*c1)
pl.xlabel('Absolute Frequincy')
pl.ylabel('DFT Values')
pl.title('Specturm of Signal 1 with 3Hz')
pl.show()                                                 #Showing The
Signal (m1*c1)


pl.figure(7)                                              #Mark The
Signal (m2*c1) as a Figure 7
pl.plot(t,m2*c1 ,'k')                                     #Plot The
Signal(m2*c1)
                                                          #Lable The
Signal (m2*c1)
pl.xlabel('Absolute Frequincy')
pl.ylabel('DFT Values')
pl.title('Specturm of Signal 2 with 4Hz')
pl.show()                                                 #Showing The
Signal (m2*c1)



pl.figure(8)                                              #Mark The
Signal (m2*c1) as a Figure 8
pl.plot(t,m3*c1 ,'-r')
                                                          #Lable The
Signal (m3*c1)
pl.xlabel('Absolute Frequincy')
pl.ylabel('DFT Values')
pl.title('Specturm of Signal 3 with 5Hz')
pl.show()                                                 #Showing The
Signal (m3*c1)
#t = np.linspace(-1, 1, 201)
#x = (np.sin(2*np.pi*0.75*t*(1-t) + 2.1) + 0.1*np.sin(2*np.pi*1.25*t +
1) + 0.18*np.cos(2*np.pi*3.85*t))
#xn = x + np.random.randn(len(t)) * 0.08

#======================================================================
==================================
#                             creating a filter for signal-1
#======================================================================
==================================
b, a = signal.butter(3, 0.05) #Infinite impulse response (IIR)
zi = signal.lfilter_zi(b, a)
```

```
z, _ = signal.lfilter(b, a, xn, zi=zi*xn[0])
z2, _ = signal.lfilter(b, a, z, zi=zi*z[0])
y = signal.filtfilt(b, a, xn)
plt.figure(9)                                                       #Mark The
First Filtered Signal (m1) as a Figure 9
plt.plot(t, xn, 'b', alpha=0.75)                                    #Plot The
Signal(xn)
plt.plot(t, z, '-r', t, z2, 'r', t, y, 'k')                         #Plot The
Signal(z)
plt.legend(('noisy signal', 'lfilter, once', 'lfilter,
twice','filtfilt'), loc='best')
plt.grid(True)                                                      #Showing The
Grides In The Graph
                                                                    #Lable The
Signal
pl.xlabel('Absolute Frequincy')
pl.ylabel('DFT Values')
pl.title('Filtered Signal-1 with 3Hz')
plt.show()                                                          #Showing The
First Filtered Signal (m1)

#======================================================================
==================================
#                                   creating a filter for signal-2
#======================================================================
==================================
b, a = signal.butter(3, 0.07)
zi = signal.lfilter_zi(b, a)
z, _ = signal.lfilter(b, a, xn, zi=zi*xn[1])
z2, _ = signal.lfilter(b, a, z, zi=zi*z[1])
y = signal.filtfilt(b, a, xn)
plt.figure(10)                                                      #Mark The
Scecond Filtered Signal (m2) as a Figure 10
plt.plot(t, xn, 'b', alpha=0.75)                                    #Plot The
Signal(xn)
plt.plot(t, z, '-y', t, z2, 'r', t, y, 'k')                         #Plot The
Signal(z)
plt.legend(('noisy signal', 'lfilter, once', 'lfilter,
twice','filtfilt'), loc='best')
plt.grid(True)                                                      #Showing The
Grides In The Graph
                                                                    #Lable The
Second Filtered Signal
pl.xlabel('Absolute Frequincy')
pl.ylabel('DFT Values')
pl.title('Filtered Signal-2 with 4Hz')
```

```
plt.show()                                                   #Showing The
Second Filtered Signal (m2)


#======================================================================
==================================
#                                 creating a filter for signal-3
#======================================================================
==================================
b, a = signal.butter(3, 0.09)
zi = signal.lfilter_zi(b, a)
z, _ = signal.lfilter(b, a, xn, zi=zi*xn[2])
z2, _ = signal.lfilter(b, a, z, zi=zi*z[2])
y = signal.filtfilt(b, a, xn)
plt.figure(11)                                              #Mark The
Third Filtered Signal (m3) as a Figure 11
plt.plot(t, xn, 'b', alpha=0.75)                            #Plot The
Signal(xn)
plt.plot(t, z, '-m', t, z2, 'r', t, y, 'k')                 #Plot The
Signal(z)
plt.legend(('noisy signal', 'lfilter, once', 'lfilter,
twice','filtfilt'), loc='best')
plt.grid(True)                                              #Showing The
Grides In The Graph
                                                            #Lable The
Third Filtered Signal
pl.xlabel('Absolute Frequincy')
pl.ylabel('DFT Values')
pl.title('Filtered Signal-3 with 5Hz')
plt.show()                                                  #Showing The
Third Filtered Signal (m3)
```

Delta Modulation

```
# Delta Modulation

import numpy as np
import matplotlib.pyplot as plt
import pylab as pl

#fsz = (7,5)                                  # figure size
Fs = 80                                      # sampling rate
fm = 10                                      # frequency of sinusoid
tlen = 1.0                                   # length in seconds
```

```
tt = np.arange(np.round(tlen*Fs))/float(Fs) # generate time axis
xt = np.sin(2*np.pi*fm*tt)                  # generate a sine wave:

pl.subplot(2,1,1)
pl.title(r'$Delta Modulation Technique$')
pl.xlabel('Figure:Modulated Signal')
pl.plot(xt,'-k',label='cosine')
pl.legend(loc='best')

l=len(xt)
delta=1                                     # define step size and
plot the delta modulated signal
xn=[0,0]
d=[]

for i in range(1,l):
    if xt[i]>xn[i]:
        d.append(1)
        xn.append(xn[i]+delta)
    else:
        d.append(0)
        xn.append(xn[i]-delta)
x = np.arange(l+1)

plt.step(x,xn ,where='mid', label='mid')
plt.plot(x, xn, 'C3o', alpha=1)
#stairs(xn)

                                            # recover the original
signal (apply demodulation)
for i in range(1,len(d)):
    if d[i]>xn[i]:
        d.append(0)
        xn.append(xn[i]-delta)
    else:
        d.append(1)
        xn.append(xn[i]+delta)

pl.subplot(2,1,2)
pl.xlabel('Figure:Recovered Original Signal')
pl.plot(xt,'r',label='original signal')
pl.legend(loc='best')
plt.grid()
plt.show()
```

PCM

```
import numpy as np
import matplotlib.pyplot as plt

# Sampling, quantization and coding

fsz = (7,5)                                                            #
figure size
fsz2 = (fsz[0],fsz[1]/2.0)                                             # half
high figure
# initial parameters
Fs = 8000                                                              #
sampling rate
fm = 1000                                                              #
frequency of sinusoidal
tlen = 1.0                                                             #
length in seconds
tt = np.arange(np.round(tlen*Fs))/float(Fs)                            #
generate time axis
xt = np.sin(2*np.pi*fm*tt)                                             #
generate sine

print('xt = {}'.format(xt[:12]))                                       # print
the first 12 values of x(t)

plt.figure(1, figsize=fsz)                                             #
figure no 1 and its size
plt.plot(tt[:24], xt[:24])                                             # x-
axis=tt , y-axis=xt
plt.grid()                                                             #
showing girds
plt.show()                                                             # print
base band sinusoidal

# create a labeled graph
plt.figure(2, figsize=fsz)                                             #
figure no 2 and its size
plt.plot(tt[:24], xt[:24], '-b')                                       # x-
axis=tt , y-axis=xt ,color = blue
plt.plot(tt[:24], xt[:24], 'or', label='xt values')                    # label
the graph
plt.ylabel('$x(t)$')                                                   # label
y-axis
plt.xlabel('t [sec]')                                                  # label
```

```
x-axis
strt2 = 'Sinusoidal Waveform $x(t)$'
strt2 = strt2 + ', $f_m={}$ Hz, $F_s={}$ Hz'.format(fm, Fs)
plt.title(strt2)                                                     # title
of the graph
plt.legend()
plt.grid()                                                           #
showing girds
plt.show()                                                           # print
the labeled base band sinusoidal


fm = 500                                                             #
frequency of sinusoid
phi=30
A=1.2
# make stem plot
plt.figure(3, figsize=fsz)
plt.stem(tt[:24], xt[:24],use_line_collection=True)
plt.ylabel('$x(t)$')
plt.xlabel('t [sec]')
strt1 = 'Stem Plot of Sinusoidal Waveform $x(t)$'
strt1 = strt1 + ', $f_m={}$ Hz, $F_s={}$ Hz'.format(fm, Fs)
plt.title(strt1)
plt.grid()
plt.show()

N = 3                                          # upsampling factor
xNt = np.vstack((xt, np.zeros((N-1, xt.size)))) # expand N times
xNt = np.reshape(xNt, -1, order='F')           # reshape into array
print(xNt[:24])                                # check readout order
FsN = N*Fs                                     # new sampling rate
ttN = np.arange(xNt.size)/float(FsN)           # new time axis

# new stem plot
plt.figure(2, figsize=fsz)
plt.stem(ttN[:N*24], xNt[:N*24],use_line_collection=True)
plt.ylim([-1.2, 1.2])
plt.ylabel('$x_N(t)$')
plt.xlabel('t [sec]')
strt2 = 'Expanded by $N={}$ Sine Waveform $x_N(t)$'.format(N)
strt2 = strt2 + ', $f_m={}$ Hz, $F_{{sN}}={}$ Hz'.format(fm, FsN)
plt.title(strt2)
plt.grid()
plt.show()
```

```python
def sinc_ipol(Fs, fL, k):
# create time axis
 ixk = int(np.round(Fs*k/float(2*fL)))
 tth = np.arange(-ixk, ixk+1)/float(Fs)
# sinc pulse
 ht = 2*fL*np.sinc(2*fL*tth)
 return tth, ht


# plot of interpolation waveform
fL = 3000                           # cutoff frequency
k = 10                              # sinc pulse truncation
tth, ht = sinc_ipol(FsN, fL, k)
plt.figure(3, figsize=fsz)
plt.plot(tth, ht, '-m')
plt.ylabel('$h(t)$')
plt.xlabel('t [sec]')
strt3 = "'sinc' Pulse for Interpolation"
strt3 = strt3 + ', $F_s={}$ Hz, $f_L={}$ Hz, $k={}$'.format(FsN, fL,
k)
plt.title(strt3)
plt.grid()
plt.show()

# convolve expanded sine sequence with interpolation waveform to
# obtain upsampled (by factor N) sequence yNt with sampling rate FsN
yNt = np.convolve(xNt, ht, 'same')/float(Fs)
# stem plot of upsampled sequence
plt.figure(4, figsize=fsz)
plt.stem(ttN[:N*24], yNt[:N*24],use_line_collection=True)
plt.ylim([-1.2, 1.2])
plt.ylabel('$y_N(t)$')
plt.xlabel('t [sec]')
strt4 = 'Sine Waveform $y_N(t)$, Upsampled $N={}$'.format(N)
strt4 = strt4 + ', $f_m={}$ Hz, $F_{{sN}}={}$ Hz'.format(fm, FsN)
plt.title(strt4)
plt.grid()
plt.show()
```