```
% 1.- LINEAL DISTORION
clear all
close all
T=2;
V1=1.2;V2=-1/3;V3=1/5;
f1=2;f2=3*f1;f3=5*f1;
Fs=1000;
delta_t=1/Fs;
T_obs=1;
t=0:delta_t:T_obs;
x1=V1*cos(2*pi*f1*t);
x2=V2*cos(2*pi*f2*t);
x3=V3*cos(2*pi*f3*t);

% 1.1  Exercise 1: amplitude distortion
figure
plot(t,x1,'r')
hold on
plot(t,x2,'g')
plot(t,x3,'b')
plot(t,x1+x2+x3,'k')
plot(t,x1+0.6*x2+0.4*x3,'k--')
legend('x1','x2','x3','x1+x2+x3','x1+0.6*x2+0.4*x3')
title('Distorsión amplitud')

% Exercise 2: phase distortion.
figure
plot(t,x1,'r')
hold on
plot(t,x2,'g')
plot(t,x3,'b')
plot(t,x1+x2+x3,'k')
x2=V2*cos(2*pi*f2*t-3/4*pi);
x3=V3*cos(2*pi*f3*t-5/4*pi);
plot(t,x1+x2+x3,'k--')
legend('x1','x2','x3','x1+x2+x3','x1(0°)+x2(-3/4*pi)+x3(-5/4*pi)')
title('Distorsión fase')

% Exercise 3: amplitude and phase distortion

TO BE DONE BY THE SUTUDENT

% Exercise 4: amplitude and phase distortion in a real channel
f=0:0.1:10;
K0=1;K1=0.3;t0=1;t1=1.7;
H=K0*exp(-j*2*pi.*f*t0)+K1*exp(-j*2*pi.*f*t1);
figure
subplot(2,1,1)
plot(f,abs(H))
subplot(2,1,2)
plot(f,angle(H))
[abs(H(f1*10)) abs(H(f2*10)) abs(H(f3*10))
angle(H(f1*10)) angle(H(f2*10)) angle(H(f3*10))]
x1=V1*cos(2*pi*f1*t);
x2=V2*cos(2*pi*f2*t);
x3=V3*cos(2*pi*f3*t);
```

```
figure
plot(t,x1,'r')
hold on
plot(t,x2,'g')
plot(t,x3,'b')
plot(t,x1+x2+x3,'k')
x1=abs(H(f1*10+1))*V1*cos(2*pi*f1*t+angle(f1*10));
% Aqui: +1 y multiplicación
x2=abs(H(f2*10+1))*V2*cos(2*pi*f2*t+angle(f2*10));
x3=abs(H(f3*10+1))*V3*cos(2*pi*f3*t+angle(f3*10));
plot(t,x1+x2+x3,'k--')
legend('x1','x2','x3','x1+x2+x3','x*H(f)')
title('Distorsión amplitud de la funcion H')
```

```
% 2.- NON LINEAR DISTORTION
% 2.1 Exercise 5: Distortion of a tone
clear all
close all
k1=10;k2=4.3;k3=-5;
t=0:0.001:0.4;
V1=1.5;f1=20;
x1=V1*cos(2*pi*f1*t);
f=0:(1/(max(t))):(1/(t(2)-t(1)));
figure
subplot (2,1,1)
plot(t,x1)
xlabel('tiempo');ylabel('amplitud')
subplot(2,1,2)
stem(f,abs(fft(x1)))
xlim([0 100])
xlabel('frecuencia');ylabel('amplitud')
y=k1*x1+k2*x1.^2+k3*x1.^3;
subplot (2,1,1)
hold on
plot(t,y,'k')
xlabel('tiempo');ylabel('amplitud')
subplot(2,1,2)
hold on
stem(f,abs(fft(y)),'k')
xlim([0 100])
xlabel('frecuencia');ylabel('amplitud')

%%%%%%%%%%%%%%%%%%%%%%%
% 2.2 Exercise 6: Distortion of Two-tones
V1=1.3;f1=5;
V2=1/2;f2=5*f1;
x1=V1*cos(2*pi*f1*t);
x2=V2*cos(2*pi*f2*t);
f=0:(1/(max(t))):(1/(t(2)-t(1)));
```

TO BE DONE BY THE SUTUDENT

```
% 3.- Noise

clear all
close all
T=0.4;
Fs=1000;
t=0:1/Fs:T;
V1=1.2;V2=1.5;f1=5;f2=5*f1;
x1=V1*cos(2*pi*f1*t);
x2=V2*cos(2*pi*f2*t);
figure
subplot(2,1,1)
plot(t,x1+x2);
xlabel('tiempo');ylabel('amplitud')
subplot(2,1,2)
f=0:(1/(max(t))):(1/(t(2)-t(1)));
stem(f,abs(fft(x1+x2)))
xlabel('frecuencia');ylabel('amplitud')
xlim([0 100])
%%
x=x1+x2;
eta=0.1;
n=tco_wgn(size(x,1),size(x,2),eta,Fs);
y=x+n;
figure
subplot(2,1,1)
plot(t,y);
xlabel('tiempo');ylabel('amplitud')
subplot(2,1,2)
f=0:(1/(max(t))):(1/(t(2)-t(1)));
stem(f,abs(fft(y)))
xlabel('frecuencia');ylabel('amplitud')
xlim([0 100])

% Comprobación ruido
%En tiempo
Px=sum(x.^2)
Py=sum(y.^2)
Pn=Py-Px
%En frecuencia
Px=mean(abs(fft(x)).^2)
Py=mean(abs(fft(y)).^2)
Pn=Py-Px
eta_comp=(2*Pn/Fs)
input('prompt')
```

```
function y = tco_wgn(M,N,eta,Fs)

%TCO_WGN Genera ruido blanco gaussiano.
%   Y = tco_wgn(M,N,ETA,Fs) genera una matriz M-por-N de ruido blanco
gaussiano de
%   densidad espectral de potencia ETA, en la banda de frecuencias
%
%   Adaptada de wgn() del Communication Toolbox de Matlab
%   Rev: X-23-05-07 (compatible con Matlab 6.5)

% --- Initial checks
Pn=10*log10(eta*Fs/2);

eta*Fs/2


y=tco_wgn_interno(M,N,Pn);

function y=tco_wgn_interno(varargin)
% --- Initial checks
error(nargchk(3,7,nargin));

% --- Value set indicators (used for the strings)
pModeSet    = 0;
cplxModeSet = 0;

% --- Set default values
p        = [];
row      = [];
col      = [];
pMode    = 'dbw';
imp      = 1;
cplxMode = 'real';
seed     = [];

% --- Placeholders for the numeric and string index values
numArg = [];
strArg = [];

% --- Identify string and numeric arguments
%     An empty in position 4 (Impedance) or 5 (Seed) are considered
%numeric
for n=1:nargin
   if(isempty(varargin{n}))
      switch n
      case 4
         if(ischar(varargin{n}))
            error('The default impedance should be marked by [].');
         end;
         varargin{n} = imp; % Impedance has a default value
      case 5
         if(ischar(varargin{n}))
            error('The default seed should be marked by [].');
         end;
         varargin{n} = [];  % Seed has no default
```

```
      otherwise
         varargin{n} = '';
      end;
   end;

   % --- Assign the string and numeric vectors
   if(ischar(varargin{n}))
      strArg(size(strArg,2)+1) = n;
   elseif(isnumeric(varargin{n}))
      numArg(size(numArg,2)+1) = n;
   else
      error('Only string and numeric arguments are allowed.');
   end;
end;

% --- Build the numeric argument set
switch(length(numArg))

   case 3
      % --- row is first (element 1), col (element 2), p (element 3)

      if(all(numArg == [1 2 3]))
         row    = varargin{numArg(1)};
         col    = varargin{numArg(2)};
         p      = varargin{numArg(3)};
      else
         error('Illegal syntax.')
      end;

   case 4
      % --- row is first (element 1), col (element 2), p (element 3), imp
%(element 4)
      %

      if(all(numArg(1:3) == [1 2 3]))
         row    = varargin{numArg(1)};
         col    = varargin{numArg(2)};
         p      = varargin{numArg(3)};
         imp    = varargin{numArg(4)};
      else
         error('Illegal syntax.')
      end;

   case 5
      % --- row is first (element 1), col (element 2), p (element 3), imp
%(element 4), seed (element 5)

      if(all(numArg(1:3) == [1 2 3]))
         row    = varargin{numArg(1)};
         col    = varargin{numArg(2)};
         p      = varargin{numArg(3)};
         imp    = varargin{numArg(4)};
         seed   = varargin{numArg(5)};
      else
         error('Illegal syntax.');
```

```
      end;
   otherwise
      error('Illegal syntax.');
end;

% --- Build the string argument set
for n=1:length(strArg)
   switch lower(varargin{strArg(n)})
   case {'dbw' 'dbm' 'linear'}
      if(~pModeSet)
         pModeSet = 1;
         pMode = lower(varargin{strArg(n)});
      else
         error('The Power mode must only be set once.');
      end;
   case {'db'}
      error('Incorrect power mode passed in.  Please use ''dBW'',
''dBm'', or ''linear.''');
   case {'real' 'complex'}
      if(~cplxModeSet)
         cplxModeSet = 1;
         cplxMode = lower(varargin{strArg(n)});
      else
         error('The complexity mode must only be set once.');
      end;
   otherwise
      error('Unknown option passed in.');
   end;
end;

% --- Arguments and defaults have all been set, either to their defaults
%or by the values passed in
%     so, perform range and type checks

% --- p
if(isempty(p))
   error('The power value must be a real scalar.');
end;

if(any([~isreal(p) (length(p)>1) (length(p)==0)]))
   error('The power value must be a real scalar.');
end;

if(strcmp(pMode,'linear'))
   if(p<0)
      error('In linear mode, the required noise power must be >= 0.');
   end;
end;

% --- Dimensions
if(any([isempty(row) isempty(col) ~isscalar(row) ~isscalar(col)]))
   error('The required dimensions must be real, integer scalars > 1.');
end;

if(any([(row<=0) (col<=0) ~isreal(row) ~isreal(col) ((row-floor(row))~=0)
((col-floor(col))~=0)]))
```

```
   error('The required dimensions must be real, integer scalars > 1.');
end;

% --- Impedance
if(any([~isreal(imp) (length(imp)>1) (length(imp)==0) any(imp<=0)]))
   error('The Impedance value must be a real scalar > 0.');
end;

% --- Seed
if(~isempty(seed))
   if(any([~isreal(seed) (length(seed)>1) (length(seed)==0) any((seed-
floor(seed))~=0)]))
      error('The State must be a real, integer scalar.');
   end;
end;

% --- All parameters are valid, so no extra checking is required
switch lower(pMode)
   case 'linear'
      noisePower = p;
   case 'dbw'
      noisePower = 10^(p/10);
   case 'dbm'
      noisePower = 10^((p-30)/10);
end;

% --- Generate the noise
if(~isempty(seed))
   randn('state',seed);
end;

if(strcmp(cplxMode,'complex'))
   y = (sqrt(imp*noisePower/2))*(randn(row,col)+j*randn(row,col));
else
   %y = (sqrt(imp*noisePower))*randn(row,col);
   y = randn(row,col);
   N=length(y);
   pot=sum(y.^2);
   A=sqrt(noisePower/pot);
   y=y*A;
end;

noisePower
sum(y.^2)
input('prompt')
```