V2: AM MODULATION AND DEMODULATION

By Laurence G. Hassebrook

September 5, 2012: updated September 6, 2013

Table of Contents
V2: AM MODULATION AND DEMODULATION ........................................................................................................ 1
0. Overview ............................................................................................................................................................ 1
1. SYSTEM ARCHITECTURE ......................................................................................................................... 2
2. MESSAGE SIGNAL GENERATION ............................................................................................................... 3
  2.1 Group Name and Number of Bits ............................................................................................................... 4
  2.2 Bit Sequence Generation ......................................................................................................................... 4
3. AM MODULATION ........................................................................................................................................ 4
4. CHANNEL MODEL ........................................................................................................................................... 8
5. AM DEMODULATION ................................................................................................................................. 11
6. PERFORMANCE VERIFICATION ............................................................................................................... 13
7. REFERENCES .................................................................................................................................................. 14
A. APPENDIX: INSTRUCTOR PROGRAMS ...................................................................................................... 14
A.1 BGEN .......................................................................................................................................................... 14
A.2 CHANNEL .................................................................................................................................................. 14
A.3 BITCHECK .................................................................................................................................................. 16

0. Overview

In this visualization we introduce and document a MATLAB based communication simulation system. The AM modulation scheme is implemented with the simulation and the student is asked to maximize the number of message bits without receiving an error. Baseline parameter values are given and the modulation and demodulation technique is given mathematically. The student is required to implement the mathematics of the modulator and demodulator to MATLAB code.
The student is required to implement the mathematics of the modulator and demodulator to MATLAB code. In this visualization the student only needs to submit their simulation .m files to the instructor. No written report is necessary.

The students are also required to formulate a group name, send the instructor their modified createBsize.m, modulator.m and demodulator.m files along with their group member names.

1. **SYSTEM ARCHITECTURE**

The system architecture includes programs that the student modifies and programs that the instructor uses to evaluate their system with. The student has access to the instructor programs so that they can pre-evaluate their systems performance. Given the student programs, the students are required to comply with the File and Data formatting specifications. The student programs have the I/O functionality already built in and specify sections where the students can insert their code.

Referring to Figs. 1-1 and 1-2, the procedure for running the system is as follows:

1. (student) Edit in the group name and number of bits (Nbit) into name_createBsize.m and change name to AM_createBsize.m.
2. (student) Run AM_createBsize.m. This stores Nbit and groupname.
3. (instructor) Edit in group name into BgenAM.m and run program. This stores two files, one with the random bit sequence of Nbit bits and the other file contains the active group being processed.
4. (student) Run AM_modulator.m. The input is the Message signal is the random bit sequence. The bit sequence is scaled to be bipolar. Using a kronecker product operation, the bipolar message sequence is upsampled to length N. This in turn is modulated using AM modulation at a carrier frequency, kc, specified by the student. The modulator outputs two files, the first with the modulated signal exactly N real elements long and the other a special trinary signal where 1 indicates time location of a “1” bit and -1 indicates the location of a “0” bit. This trinary signal is NOT passed to the demodulator but ONLY ACCESSED by the bitcheckxx.m program.
5. (instructor) Run Channelxx.m. The channel filters the spectrum of the modulated signal and adds Gaussian white noise to the signal based on the minimum and maximum values in the signal. The instructor determines the channel transfer function and the amount of noise added. The filtered noisy output is stored in AM_r.mat as the real vector “r” that is N elements in length. Channel will also test for errors in the input data format.
6. (student) Run AM_demodulator.m. The student may use any parameters used in the modulator, including the number of bits, and will process the input “r” signal vector from the Channel. However, there is NO KNOWLEDGE of the random bit sequence allowed. The output of the demodulator, real vector Bs, is stored in AM_Bs.mat.
7. (instructor) Run bitcheckxx.m. Bitcheck uses the Bcheck signal generated in the Modulator to test for 1 or 0 bits. It prints out any errors in format but most importantly how many false alarms and misses occur in the detection process. If there are errors in
detection then a figure will be generated revealing the local signal characteristics that led to the error.

![Diagram of Simulation Flow Chart with associated MATLAB programs and data variables and vectors.](image1)

**Figure 1-1:** Simulation Flow Chart with associated MATLAB programs and data variables and vectors.

The flowchart in Fig. 1-1 shows the relationship of the MATLAB functions with respect to a standard communications system flowchart. Notice that “name_modulator.m” both encodes by upsampling the bit sequence and modulates. The function Bitcheck.m both decodes by thresholding the $B_s$ output vector from the demodulator as well as tests for detection errors based on the $B_{check}$ vector.

![Diagram of Communications Simulation System Architecture.](image2)

**Figure 1-2:** Communications Simulation System Architecture.

The simulation architecture is shown in Fig. 1-2. The left group corresponds to the student controlled MATLAB programs, the center group represents the data storage and the right group shows the instructor controlled MATLAB programs.

### 2. MESSAGE SIGNAL GENERATION

The student renames and uses the “name_Bsize.m” file to establish their groups’ name and number bits to be transmitted through the communications simulator. Once that is run, it stores these two parameters name_Bsize.mat. The next step is to run Bgen.m, given in Appendix: Bgen. But before running Bgen.m, the groupname variable needs to be manually changed to match which student group name that should be run. This way the instructor can choose which groups’ program set is to be run. If the students are using Bgen in development, then they just need to edit in their name one time.
2.1 Group Name and Number of Bits

Below is the source code for the *name_createBsize.m* program. For this example, “AM” has been entered as the group name and ???? need to be replaced by a number. For example, replace ???? with 1024 which means that 1024 bits will be transmitted through the communication system.

```matlab
% generate groupname_Bsize.mat
clear all;
% INSERT GROUP NAME AND NUMBER OF BITS
groupname='AM' % name of group
Nbit=???? % number of bits to transmit
% END OF INSERT
% name of output file that stores Nbit and filename
filename=sprintf('%s_Bsize.mat',groupname)
save(filename); % stores groupname, Nbit to ee51112_Bsize.mat
```

2.2 Bit Sequence Generation

From Appendix: Bgen, the only line of code that would need to be changed is

```matlab
groupname='AM' % instructor enters this name to select student project
```

In this case ‘AM’ is entered but would be whatever the group name is. Bgen.m generates a pseudo-random set of bits, Nbit long, to be transmitted through the communication system.

3. AM MODULATION

The modulator inputs the bit sequence and knows the full number of samples N to be used in the system. It first determines the number of samples, Nsample, per bit. Then using the kronecker product to upsample the bit sequence to the message sequence plus any padding necessary to reach length N. The mathematical representation for this operation is:

\[
{s}_m = (2{b}_m - 1) \otimes u_N,
\]

(3.1)

where \( s_m \) is the signal vector, \( b_m \) is the binary message sequence, \( \otimes \) is the Kronecker product and \( u_{N_s} \) is a unit vector \( N_s = N_{\text{sample}} \) long.

```matlab
% generate bit matrix based on groupname_Bsize.mat
clear all;
load 'AM_B.mat';
load 'AM_Bsize.mat';
% generate a real vector s, N=131072*8 or let N be less for debug process
N=131072*8 % N is set by instructor and cannot be changed
% CREATE THE MESSAGE SIGNAL
```
Nsample=floor(N/Nbit)
% form pulse shape
pulseshape=ones(1,Nsample);
% modulate sequence to either +1 and -1 values
b1(1:Nbit)=2*B(1,1:Nbit)-1;
stemp=kron(b1,pulseshape); % form continuous time approximation of message
sm=-ones(1,N);
if N > (Nsample*Nbit)
    sm(1:(Nsample*Nbit))=stemp(1:(Nsample*Nbit));
else
    sm=stemp;
end;
size(sm) % verify shape
% plot message signal or a section of the message signal
figure(1);
if Nbit<41
    plot(sm);
    axis([1,N,-1.1,1.1]);
    xlabel('Message Signal');
else
    plot(sm(1:(Nsample*40)));
    axis([1,(Nsample*40),-1.1,1.1]);
    xlabel('Sample section of Message Signal');
end;
print -djep Modulator_figure1

% FT of message waveform
Sm=abs(fftshift(fft(sm)));
figure(2);
k=0:(N-1);
k=k-N/2;
plot(k,Sm);
xlabel('DFT spectrum of Message Signal');
print -djep Modulator_figure2

Figure 3-1: Sample section of Message sequence after upsampling.
The actual modulator code is placed between the commented sections below. The carrier frequency, kc, is in cycles per N samples. Its value should be as high as possible without suffering too much attenuation from the Channel response. If it is attenuated by the channel then the SNR will decrease because the channel noise is proportional to the max-min value before the signal is filtered by the channel. The cutoff for the channel is fc=N/4. So a good first guess of a kc value might be kc=N/8 which is half the channel cutoff. Given the carrier frequency, the AM modulation would be

$$s(t) = \frac{(1 + s_m(t)) \cos\left(\frac{2\pi k_c t}{N}\right)}{2}$$  \hspace{1cm} (3.2)

% INSERT MODULATION EQUATION:
% INSERT MODULATION EQUATION:
% INSERT MODULATION EQUATION: Inputs sm vector, kc, t and N
% create AM modulation signal s
\t\text{t=0:}(N-1);\text{\hspace{1cm}}
\text{kc=???}; \text{\hspace{1cm}}
\text{s=???}; \text{\hspace{1cm}}% use equation 3.2 to write the vector-MATLAB equivalent
% END OF MODULATION INSERT
% END OF MODULATION INSERT
% END OF MODULATION INSERT
% plot AM signal
\text{figure(3)};
\text{if Nbit<41}
\hspace{1cm} \text{plot(s);\hspace{1cm}}
\hspace{1cm} \text{axis([1,N,-1.1,1.1]);\hspace{1cm}}
\hspace{1cm} \text{xlabel('AM Signal');\hspace{1cm}}
\text{else}
\hspace{1cm} \text{Ntemp=Nsamp*40;\hspace{1cm}}
\hspace{1cm} \text{plot(s(1:Ntemp));\hspace{1cm}}
\hspace{1cm} \text{axis([1,Ntemp,-1.1,1.1]);\hspace{1cm}}
\hspace{1cm} \text{xlabel('Sample section of AM Signal');\hspace{1cm}}
\text{end;}
\text{print -djpeg Modulator_figure3}
Figure 3-3: Sample section of AM modulated signal.

% FT of modulated waveform
S=abs(fftshift(fft(s)));
figure(4);
k=0:(N-1);
k=k-N/2;
plot(k,S);
xlabel('Spectrum of AM Signal');
print -djepg Modulator_figure4

Figure 3-4: Spectrum of AM modulated Signal.

% create the bit check matrix to only be used by the Bcheckxx.m file
% YOU CANNOT PASS THIS INFORMATION TO YOUR DEMODULATOR!!
samplepulse=zeros(1,Nsample);
samplepulse(Nsample/2)=1;
Bcheck=zeros(1,N);
% modulate first sequence to either +1 and -1 values
blcheck(1:Nbit)=2*B(1,1:Nbit)-1;
bcchecktemp=kron(blcheck,samplepulse);
Bcheck=zeros(1,N);
if N > (Nsample*Nbit)
    Bcheck(1:(Nsample*Nbit))=bchecktemp(1:(Nsample*Nbit));
else
    Bcheck=bchecktemp;
end;
figure(5);
if Nbit<41
    n=1:N;
    plot(n,sm,n,Bcheck);
    axis([1,N,-1.1,1.1]);
    xlabel('Bit Check Signal');
else
    Ntemp=Nsample*40;
    n=1:Ntemp;
    plot(n,sm(1:Ntemp),n,(0.9*Bcheck(1:Ntemp)));
    axis([1,Ntemp,-1.1,1.1]);
    xlabel('Sample Section of Bit Check Signal');
end;
print -djpeg Modulator_figure5
save 'AM_signal' s;
save 'AM_Bcheck' Bcheck;

![Image of graph showing sample section of bit check signal]

Figure 3-5: Sample section of the bit check signal.

There are two signals shown in Fig. 3-5, the bipolar binary signal in blue and the bitcheck signal in green. The bitcheck has been slightly attenuated from its true +/- 1 value to disconnect from the binary signal curve but it really has 3 values +1, 0 and -1. A “0” indicates no bit value in that time location, a “1” indicates there should be a high bit value and -1 indicates a low bit value for that time location. The student is not allowed to pass this information to the demodulator so there is no prior knowledge of the random bit sequence being used in the demodulator. However, bitcheckxx.m uses this information to test the demodulator output.

### 4. CHANNEL MODEL
See Appendix Channel code. The channel filters and adds noise to the input signal. The mathematical representation of the channel output is

\[ r(t) = s(t) \ast h(t) + \tilde{\omega}(t) \]  

(4.1)

where \( s(t) \) is the input signal, \( h(t) \) is the channel impulse response and \( \tilde{\omega}(t) \) is a white Gaussian noise process.

![Log Magnitude Spectrum of input signal.](image1)

Fig. 4-1 is obtained by taking the log of the input spectrum such that

\[ \log(|S(f)| + 0.1) \]

![Channel filter response.](image2)

The channel uses a butterworth frequency response as shown in Fig. 4-2.
Figure 4-3: Input response after filtering.

Figure 4-4: Sample sequence of original modulated input signal.
Fig. 4-6 shows the final channel response after filtering and additive noise corruption.

5. AM DEMODULATION

The demodulation simulates rectification by a single diode followed by low pass filtering, leaving the baseband signal. The mathematics for these is indicated near the question marked lines in the source code below.

```matlab
% generate bit matrix based on groupname_Bsize.mat
clear all;
load 'AM_Bsize'; % get number of bits sizes
load 'AM_r';
[M,N]=size(r)
```
figure(1)
Nsample=floor(N/Nbit)
if Nbit<41
    plot(r);
    axis([1,N,-1.1,1.1]);
    xlabel('Received AM Signal');
else
    Ntemp=Nsample*40;
    plot(r(1:Ntemp));
    axis([1,Ntemp,-1.1,1.1]);
    xlabel('Sample section of Received AM Signal with Noise');
end;
print -djpeg Demod_figure1

Figure 5-1: Sample section of input signal from channel model.

Ignoring the $v_{be}$ drop of a diode, the single diode rectification can be modeled by finding all the input signal values below 0. In matlab, this can be done with a “find” function such as “J=find(r<0)” followed by r1=r; and r1(J)=0; thereby clipping all values below zero.

$$r_1(t) = \begin{cases} r(t) & r(t) \geq 0 \\ 0 & r(t) < 0 \end{cases}$$  \hspace{1cm} (5.1)

% INSERT DEMODULATION CODE:
% INSERT DEMODULATION CODE:
% INSERT DEMODULATION CODE: input cutoff fc and r
% rectify signal with single diode (half wave rectification)
% r1=????;
% form reconstruction filter
kc=N/8; % This should correspond to the channel parameter kc
fc=????; % cutoff frequency of reconstruction filter
% filter with some recommended parameters
Norder=8;fmax=N/2;K=1; % filter gain
[f H]=lp_butterworth_oN_dft(3*fc,K,fmax,N,Norder);
% filter signal through channel via frequency domain
rn=????;
% END OF DEMODULATION INSERT: output real vector rn that is N long
% END OF DEMODULATION INSERT:
The reconstruction filter cutoff frequency $f_c$ is determined by trial and error. Once it is

determined then the non-normalized output is simply

$$r_c(t) = \text{real} \left( \mathcal{F}^{-1} \left\{ \mathcal{F} \{ r(t) \} H(f) \right\} \right)$$

(5.2)

The last step in the demodulation is to scale the signal between 0 and 1 for the bitcheck process. Since the signal was rectified, the minimum values are close to zero so the final non-normalized signal is simply scaled by dividing by its maximum. The result is shown in Fig. 5-2.

**Figure 5-2: Section of final output of demodulation system.**

**6. PERFORMANCE VERIFICATION**
The detection error is determined by bitcheckxx.m See Appendix A.3 Bitcheck for source code. The bitcheck program checks for error in formatting and provides the false alarm and miss count. If a miss or false alarm occurs then the program will create a figure of the local region where the error occurred.

7. REFERENCES


A. APPENDIX: INSTRUCTOR PROGRAMS

There are 3 instructor controlled programs, Bgenxx.m, Channelxx.m and bitcheckxx.m. The “xx” will be different for different years or visualizations or projects. These programs are in sections A.1, A.2 and A.3 respectively.

A.1 BGEN

```matlab
% generate bit matrix based on groupname_Bsize.mat
clear all;
groupname='AM' % instructor enters this name to select student project
filename=sprintf('%s_Bsize.mat',groupname);
load (filename) % retrieve data
filename
Nbit
B=rand(1,Nbit); % generate uniformly distributed random sequence
B=binarize(B); % threshold sequence into 0s and 1s
size(B)
filename=sprintf('%s_B.mat',groupname);
save(filename); % save the random bit sequence B
% save the active groupname
save 'activegroup' groupname;
```

A.2 CHANNEL

```matlab
% channel function
clear all;
noiseCoef=0.02;
% input active group
load 'activegroup' groupname;
% input groupname_signal.mat
filename=sprintf('%s_signal.mat',groupname);
load(filename);
% make sure s is real
signal=real(s);
```
[M,N]=size(signal)
if N~= 1048576
    'Incorrect vector length, should be 1048576'
end;
%
% form filter
fc=N/4;
Norder=6;
fmax=N/2;
n=1:N;
K=1; % filter gain
% low pass filter
[f HLP]=lp_butterworth_on_dft(fc,K,fmax,N,Norder);
% filter signal through channel
S=fft(signal);
H=HLP;
R=S.*H;
sn=real(ifft(R));
k=n;k=k-N/2;
figure(1);
plot(k,log(abs(fftshift(S))+.1));
xlabel('Log Magnitude Spectrum of Input Signal');
print -djpeg Channel_figure1
figure(2);
plot(k,abs(fftshift(H)));
axis([k(1),k(N),-.1, 1.1]);
xlabel('Spectrum of Channel');
print -djpeg Channel_figure2
figure(3);
plot(n,log(abs(R)+.1));
xlabel('Log Spectrum of Output Signal, No Noise');
print -djpeg Channel_figure3
% find noise deviation based on input signal extrema
sigma=noiseCoef*(max(signal)-min(signal))
% add noise
w=sigma*randn(1,N);
r=sn+w;
% store result in groupname_r.mat
filename=sprintf('%s_r.mat',groupname);
save(filename,'r');
% PLOT spectrum and sample sections of the signal
figure(4);
Nsamplesection=20;
Nsamples=floor(N/Nsamplesection);
if N<Nsamples
    plot(n,signal);
    axis([1,N,-1.1,1.1]);
xlabel('AM Signal');
else
    plot(signal(1:Nsamples));
    axis([1,Nsamples,-1.1,1.1]);
xlabel('Sample section of AM Signal');
end;
print -djpeg Channel_figure4
figure(5);
if N<Nsamples
    plot(n,sn);
axis([1,N,-1.1,1.1]);
xlabel('Output AM Signal, No Noise');
else
    plot(sn(1:Nsamples));
    axis([1,Nsamples,-1.1,1.1]);
    xlabel('Sample section of Output AM Signal, No Noise');
end;
print -djpeg Channel_figure5
figure(6);
if N<Nsamples
    plot(n,r);
    axis([1,N,-1.1,1.1]);
    xlabel('Output AM Signal');
else
    plot(r(1:Nsamples));
    axis([1,Nsamples,-1.1,1.1]);
    xlabel('Sample section of Output AM Signal with Noise');
end;
print -djpeg Channel_figure6

A.3 BITCHECK

% bit check
clear all;
% input active group
load 'activegroup' groupname;

groupname

% input original size
filename=sprintf('%s_Bsize.mat',groupname);
load (filename) % retrieve matrix size
filename

Nbitb4=Nbit
% input original bit matrix
filename=sprintf('%s_B.mat',groupname);
load (filename);
% load bitcheck
filename=sprintf('%s_Bcheck.mat',groupname);
load (filename);
% load received signal
filename=sprintf('%s_Bs.mat',groupname);
load (filename);
% check for consistancy
[Nseqnow,N]=size(Bs);

if Nseqnow~=1
    'ERROR:bitcheck matrices inconsistent'
else
    'OK: bitcheck matrices consistant'
end;
% plot
k=1:N;
\[ f = 1; \]
\[ \text{figure}(f); \]
\[ bsl(1:N) = Bs(f,1:N); \]
\[ bcheckl(1:N) = Bcheck(f,1:N); \]
\[ k1 = 1:128; \]
\[ bs1temp(1:128) = bs1(1:128); \]
\[ bcheck1temp(1:128) = bcheck1(1:128); \]
\[ \text{plot}(k1,bs1temp,k1,bcheck1temp); \]
\[ \]
% loop through bits
\[ \text{Btest}=\text{zeros}(1,\text{Nbit});\]
\[ \text{miss}=0;\]
\[ \text{false}=0;\]
\[ \text{Nerror}=0;\]
\[ \text{nbreceived}=0;\]
\[ \text{m}=1;\]
\[ \text{nb}=1;\]
\[ \text{for } n=1:N \]
\[ \quad \text{if } \text{Bcheck}(m,n) > 0.5 \text{ "1" should be present in check signal} \]
\[ \quad \quad \text{if } \text{Bs}(m,n)>0.5\]
\[ \quad \quad \quad \text{Btest}(m,nb)=1; \]
\[ \quad \quad \text{else} \]
\[ \quad \quad \quad '\text{ERROR, missing 1'} \]
\[ \quad \quad \quad '\text{Bcheck'} \]
\[ \quad \quad \quad \text{Bcheck}(m,n) \]
\[ \quad \quad \quad '\text{Bs'} \]
\[ \quad \quad \quad \text{Bs}(m,n) \]
\[ \quad \quad \quad m \]
\[ \quad \quad \quad n \]
\[ \quad \quad \quad \text{nb} \]
\[ \quad \quad \quad \text{if } \text{Nerror}<10\]
\[ \quad \quad \quad \quad \text{figure}(1+1+Nerror); \]
\[ \quad \quad \quad \quad \text{istart}=n-(2*N/Nbit); \]
\[ \quad \quad \quad \quad \text{istop}=n+(2*N/Nbit); \]
\[ \quad \quad \quad \quad \text{if } \text{istart}<1 \]
\[ \quad \quad \quad \quad \quad \text{istart}=1 \]
\[ \quad \quad \quad \end{if} \]
\[ \quad \quad \quad \text{end}; \]
\[ \quad \quad \text{if } \text{istop}>N \]
\[ \quad \quad \quad \quad \text{istop}=N \]
\[ \quad \quad \quad \end{if}; \]
\[ \quad \quad \text{clear } x; \]
\[ \quad \quad \quad x=1:(1+\text{istop}-\text{istart}); \]
\[ \quad \quad \quad \text{btemp}=x; \]
\[ \quad \quad \quad \text{bchecktemp}=x; \]
\[ \quad \quad \quad \text{btemp}(1:(1+\text{istop}-\text{istart}))=\text{Bs}(m,\text{istart}:\text{istop}); \]
\[ \quad \quad \quad \text{bchecktemp}(1:(1+\text{istop}-\text{istart}))=\text{Bcheck}(m,\text{istart}:\text{istop}); \]
\[ \quad \quad \quad \text{plot}(x,btemp,x,bchecktemp); \]
\[ \quad \quad \quad \%\text{clear } x,btemp,bchecktemp; \]
\[ \quad \quad \quad \text{end}; \]
\[ \quad \quad \quad \text{miss} = \text{miss}+1; \]
\[ \quad \quad \quad \text{Nerror} = \text{Nerror}+1; \]
\[ \quad \quad \text{end}; \]
\[ \quad \text{nb} = \text{nb}+1; \]
\[ \quad \text{nbreceived} = \text{nbreceived}+1; \]
\[ \text{end}; \]
\[ \text{if } \text{Bcheck}(m,n) < -0.5 \text{ "-1" should be present in check signal} \]
\[ \text{if } \text{Bs}(m,n) < 0.5 \text{ "0" is present demodulated/binarized signal} \]
\[ \quad \text{Btest}(m,nb)=0; \]
else
    'ERROR, missing 0'
    'Bcheck'
    Bcheck(m,n)
    'Bs'
    Bs(m,n)
    m
    n
    nb
    if Nerror<10
        figure(1+1+Nerror);
        istart=n-(2*N/Nbit);
        istop=n+(2*N/Nbit);
        if istart<1
            istart=1
        end;
        if istop>N
            istop=N
        end;
        clear x;
        x=1:(1+istop-istart);
        btemp=x;
        bchecktemp=btemp;
        btemp(1:(1+istop-istart))=Bs(m,istart:istop);
        bchecktemp(1:(1+istop-istart))=Bcheck(m,istart:istop);
        istart
        istop
        size(x)
        size(btemp)
        size(bchecktemp)
        plot(x,btemp,x,bchecktemp);
    end;
    false=false+1;
    Nerror=Nerror+1;
end;
end;
end;
nbsent=Nbit
nbreceived
miss
false
Nerror
if nbsent~=nbreceived
    'Error between sent and recieved'
    'Number of ones and zeros sent'
    Nones=sum(sum(B))
    Nzeros=nbsent-Nones
end;