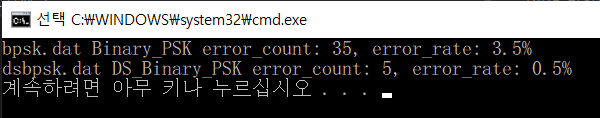
통신이론2 과제 #3

2016706018 박건희

1. 각 통신에 대한 최적 수신기를 C code로 구현하고, bit error rate를 구하시오.



2. BPSK와 DS/BPSK의 BER 관계를 이론적으로 설명하시오.

BPSK와 DS/BPSK일 때 BER 공식은 각각 위와 같다.

DS/BPSK에서 송, 수신 과정에서 PNcode를 곱하게된다. PNCODE가 랜덤할수록, N이 높을수록 성능이 좋아진다. 이때 Tb = NTC 이기 때문에 Bit period가 둘다 일정할 때, chip period가 낮을수록 성능이 좋아진다는 의미와 같아진다. DS/BPSK가 jamming noise가 송수신 과정에서 들어갈 때 PNcode가 jamming noise가 추가되기 전과 추가 후에 곱해지게된다. 이때 변조된 신호는 1과 -1이 chip period마다 랜덤하게 있는 pncode가 제곱되면서 원신호로 복조가 되고 noise는 pncode가 한 번만 곱해지면서 영향력이 매우 줄어들게된다. DS/BPSK가 즉 BPSK보다 jamming noise에 더 강하여 코딩 결과와 같이 BER이 더 낮게 나온 것을 확인하였다

**C code**

//2016706018 박건희

//header file

#include <stdio.h>

#include <math.h> //사인, 코사인함수 사용을 위해 호출

#define PI 3.1415926535897 //파이 정의

void Binary\_PSK(char file\_name[20], int phase, int sample\_num, int bit\_num);

// BPSK에 의한 bit판정

// 파일이름, 위상차이(90도 180도 차이), 심볼당 샘플갯수, 비트수

void DS\_Binary\_PSK(char file\_name[20], int phase, int sample\_num, int chip\_period, int bit\_num);

// DS/BPSK에 의한 bit판정

// 파일이름, 위상차이(180도 차이), 심볼당 샘플갯수, chip\_period, 비트수

//main function

int main() {

Binary\_PSK("bpsk.dat", 180, 60, 1000); //bpsk.dat, symbol간 phase차이 180, 심볼당 샘플수 60, 비트수 1000

DS\_Binary\_PSK("dsbpsk.dat", 180, 60, 4, 1000); //dsbpsk.dat, symbol간 phase차이 180, 심볼당 샘플수 60, chip\_period=4, 비트수 1000

}

// BPSK function

void Binary\_PSK(char file\_name[20], int phase, int sample\_num, int bit\_num) {

FILE\* fin, \* fbit; //파일구조체의 포인터 선언

fopen\_s(&fin, file\_name, "rt"); //fin에 file\_name 인자에 들어가는 파일 txt 형태로 읽기

fopen\_s(&fbit, "bit.dat", "rt"); //fbit에 bit.dat 파일 txt 형태로 읽기

float input[60] = { 0.0, }; // 파형 배열, sample 갯수 60개로 선언

float sum = 0; // ML Rule에 의거한 적분값

int detect; // 판정된 bit

int bit; // bit.dat에 저장된 실제 bit

int i, k; // 반복문용

float error\_count = 0; // 에러 bit 갯수

for (i = 0; i < bit\_num; i++) { //bit갯수 만큼 반복

for (k = 0; k < sample\_num; k++) { //symbol당 sample수만큼 반복

fscanf\_s(fin, "%f", &input[k]); //symbol의 sample을 input배열에 순서대로 저장

sum += (input[k] \* cos((2 \* PI \* k) / 8)); //sample을 코사인(위상 0도)과 correlation

} //for문 종료시 sum에 각 phase간의 sample 적분값 저장

//적분값의 크기비교 후 판정

if (sum > 0) //부호로 결정된다.

detect = 1;

else

detect = 0;

fscanf\_s(fbit, "%d", &bit); //실제 bit 읽어들임

if (detect != bit) //만약 판정된 bit와 실제 bit가 다르면

error\_count++; //에러 갯수 추가

sum = 0; //적분값 0 초기화

}

//모든 bit의 판정 종료

printf("%s Binary\_PSK error\_count: %d, error\_rate: %.1f%%\n", file\_name, (int)error\_count, error\_count / (float)bit\_num \* 100); //파일 이름, 에러비트갯수, 비트에러율 출력

}

// DS\_BPSK function

void DS\_Binary\_PSK(char file\_name[20], int phase, int sample\_num, int chip\_period, int bit\_num) {

FILE\* fin, \* fbit; //파일구조체의 포인터 선언

fopen\_s(&fin, file\_name, "rt"); //fin에 file\_name 인자에 들어가는 파일 txt 형태로 읽기

fopen\_s(&fbit, "bit.dat", "rt"); //fbit에 bit.dat 파일 txt 형개로 선언태로 읽기

float input[60] = { 0.0, }; // 파형 배열, sample 갯수 60

float PNCODE[15] = { 1, -1, -1, -1, 1, 1, 1, 1, -1, 1, -1, 1, 1, -1, -1 }; //PNCODE

float sum = 0; // ML Rule에 의거한 적분값

int detect; // 판정된 bit

int bit; // bit.dat에 저장된 실제 bit

int i, k; // 반복문용

float error\_count = 0; // 에러 bit 갯수

for (i = 0; i < bit\_num; i++) { //bit갯수 만큼 반복

for (k = 0; k < sample\_num; k++) { //symbol당 sample수만큼 반복

fscanf\_s(fin, "%f", &input[k]); //symbol의 sample을 input배열에 순서대로 저장

sum += (input[k] \* cos((2 \* PI \* k) / 8) \* PNCODE[k/chip\_period]); //sample을 코사인(위상 0도)과 correlation

} //for문 종료시 sum1, sum2에 각 phase간의 sample 적분값 저장

//적분값의 크기비교 후 판정

if (sum > 0)

detect = 1;

else

detect = 0;

fscanf\_s(fbit, "%d", &bit); //실제 bit 읽어들임

if (detect != bit) //만약 판정된 bit와 실제 bit가 다르면

error\_count++; //에러 갯수 추가

sum = 0; //적분값 0 초기화

}

//모든 bit의 판정 종료

printf("%s DS\_Binary\_PSK error\_count: %d, error\_rate: %.1f%%\n", file\_name, (int)error\_count, error\_count / (float)bit\_num \* 100); //파일 이름, 에러비트갯수, 비트에러율 출력

}

// QPSK function

void Quadrature\_PSK(char file\_name[20], int phase, int sample\_num, int bit\_num) {

FILE\* fin, \* fbit; //파일구조체의 포인터 선언

fopen\_s(&fin, file\_name, "rt"); //fin에 file\_name 인자에 들어가는 파일 txt 형태로 읽기

fopen\_s(&fbit, "bit.dat", "rt"); //fbit에 bit.dat 파일 txt 형태로 읽기

float input[40] = { 0.0, }; // 파형 배열, sample 갯수 40개인 점을 고려해서 40개로 선언

float sum1 = 0, sum2 = 0; // ML Rule에 의거한 적분값

int detect; // 판정된 bit

int bit; // bit.dat에 저장된 실제 bit

int i, k; // 반복문용

float error\_count = 0; // 에러 bit 갯수

int sym\_detect;

for (i = 0; i < bit\_num / 2; i++) { //symbol당 2bit를 가지므로 bit갯수 절반 만큼 반복

for (k = 0; k < sample\_num; k++) { //bit당 sample수만큼 반복

fscanf\_s(fin, "%f", &input[k]); //bit의 sample을 input배열에 순서대로 저장

sum1 += (input[k] \* cos((2 \* PI \* k) / 10 + (phase / 180 \* PI))); //sample을 코사인축의 phase 45도에서 적분

sum2 += (input[k] \* sin((2 \* PI \* k) / 10 + (phase / 180 \* PI))); //sample을 사인축의 phase 45도에서 적분

} //for문 종료시 sum1,sum2에 bit 하나의 cos45도,sin45도에 대한 sample 적분값 저장

fscanf\_s(fbit, "%d", &bit); //실제 bit 읽어들임

if (sum1 > 0) //첫번째비트판정 코사인45도 위치에서 적분값의 부호에따라 비트 결정

detect = 1;

else detect = 0;

if (detect != bit) //만약 판정된 bit와 실제 bit가 다르면

error\_count++; //에러 갯수 추가

fscanf\_s(fbit, "%d", &bit); //실제 bit 읽어들임

if (sum2 > 0) //두번째비트판정 사인45도 위치에서 적분값의 부호에따라 비트 결정

detect = 1;

else detect = 0;

if (detect != bit) //만약 판정된 bit와 실제 bit가 다르면

error\_count++; //에러 갯수 추가

sum1 = 0; sum2 = 0;

}

//모든 bit의 판정 종료

printf("%s Quadrature\_PSK error\_count: %d, error\_rate: %.1f%%\n", file\_name, (int)error\_count, error\_count / (float)bit\_num \* 100); //파일 이름, 에러비트갯수, 비트에러율 출력

}