M113070003 張昱辰－ECC Final project.

BCH code (15,7,2)

一、 模擬環境

- SNR：0 : 0.5 : 12
- 模擬次數：最少 100,000 次；根據 100 倍法則，總錯誤少於 100 個 bits 時不中斷。但節省模擬時間及 BER=0 的可能性，仍設最大次數設為 3,000,000。
- 接收訊號 $r[n]=y[n]+noise$ ， $noise$ ~N(0,1) $=\sqrt{\frac{N_0}{2*coderate}}$ ，

  $N_0=10^{\frac{SNR\_in\_dB}{10}}$

二、 步驟

- **construct GF(16)**

根據 $g(x)=1+x^4+x^6+x^7+x^8$ ，從課堂講義上可以找到有一個 $p(x)=1+x+x^4$ ，用十進位 exclusive or 以及右移的方式來建表。建完表後會得到 16*4 的矩陣，代表表中 4-Tuple representation 的部分，我們還可以換成 10 進位，方便後續某些運算。步驟在 #pragma region construct_Galois_Field 的折疊區塊中，示意圖如第二張圖。

TABLE 2.8: Three representations for the elements of $GF(2^4)$ generated by $p(X)=1+X+X^4$.

Power representation	Polynomial representation	4-Tuple representation
0	0	(0 0 0 0)
1	1	(1 0 0 0)
α	α	(0 1 0 0)
α^2	α^2	(0 0 1 0)
α^3	α^3	(0 0 0 1)
α^4	$1+\alpha$	(1 1 0 0)
α^5	$\alpha+\alpha^2$	(0 1 1 0)
α^6	$\alpha^2+\alpha^3$	(0 0 1 1)
α^7	$1+\alpha+\alpha^3$	(1 1 0 1)
α^8	$1+\alpha^2$	(1 0 1 0)
α^9	$\alpha+\alpha^3$	(0 1 0 1)
α^{10}	$1+\alpha+\alpha^2$	(1 1 1 0)
α^{11}	$\alpha+\alpha^2+\alpha^3$	(0 1 1 1)
α^{12}	$1+\alpha+\alpha^2+\alpha^3$	(1 1 1 1)
α^{13}	$1+\alpha^2+\alpha^3$	(1 0 1 1)
α^{14}	$1+\alpha^3$	(1 0 0 1)

```
#pragma region construct_Galois_Field
//-----------------------------------------------------
//construction of Galois Field----------------------
//-----------------------------------------------------
GF_table_binary[15][0] = 0;
GF_table_binary[15][1] = 0;
GF_table_binary[15][2] = 0;
GF_table_binary[15][3] = 0;

for(int i = 0; i < coding_len; i++)
{
    //此表格為0、1、a、a^2... mod(prime_poly) 的結果
    //prime polynomial order = 4，則取mod的order不會 >=
    //也就是遇到 10000 (16)要做判斷

    if( (alpha_reg & coding_len+1) == coding_len+1 )
    {
        alpha_reg = alpha_reg ^ prime_poly;
        //printf("%d\n", alpha_reg ^ prime_poly);
    }
    GF_table_binary[i][0] =  alpha_reg     & 1;
    GF_table_binary[i][1] = (alpha_reg>>1) & 1;
    GF_table_binary[i][2] = (alpha_reg>>2) & 1;
    GF_table_binary[i][3] = (alpha_reg>>3) & 1;

    alpha_reg = alpha_reg << 1;
    //printf("%d\n", alpha_reg);
}

for (int i = 0; i <= coding_len; i++) //改成十進位方便後
{
    for (int j = 0; j < 3; j++)
    {
        GF_table_decimal[i] = GF_table_binary[i][0] +
    }
}
```

- **construct G_{sys}**

根據 $g(x)=1+x^4+x^6+x^7+x^8$，可以直接寫出 **G** 矩陣，但並非 systematic，所以多做高斯消去法，將 **G** 化成**[P | I]**的樣子即為 G_{sys}。後面 encode/decode 的時候，就會發現 15 個 bits 的 codeword 中，最後面 7 個 bits 就是原來的 message。步驟在 #pragma region construct_G 的折疊區塊中。

```
#pragma region construct_G
//-------------------------------------------------------------
//construct G matrix (size k*n = 7*15)-----------------------
//using Guass elimination to make G = [P|I] (systematic form)-----
//using systematic form for easier decoding-----------------------
//-------------------------------------------------------------

for(int i = 0; i < info_len; i++) //construct G matrix
{
    for(int j = 0; j <= coding_len-info_len; j++)
        G[i][j+i] = generator_poly[j];

}

for (int i = 0; i < info_len; i++) //Guass elimination G to Gsys
{
    for (int ii = i+1; ii < info_len; ii++)
    {
        if(G[ii][coding_len-info_len + i])
        {
            for (int k = 0; k < coding_len; k++)
            {
                G[ii][k] = G[i][k] ^ G[ii][k];
            }
        }
        else
            continue;
    }
}
```

- **產生 data→encode→BPSK→receive→hard decision**

用 rand % 2，產生 7 bits 的 message，乘上 G_{sys} 後以 BPSK(0 變-1、1 變 1)的方式來調變，傳送訊號長度為 15。接收的部分如同前面所述 $r[n]=y[n]+noise$。收到後做 hard decision(大於 0 為 1、小於 0 為 0)。步驟在以下各個摺疊區塊中。

```
#pragma region generate_random_7_bits_data···

#pragma region encode_data···

#pragma region generate_noise···

#pragma region BPSK_modulation···

#pragma region hard_decision···
```

- **Find symdrome**

hard decision 後的訊號，以 α^1、α^2、...、α^{2t} 代入找出 syndrome，我們的 t=2，會有 4 個 syndromes。計算多項 α^n 相加時，用的是十進位表示之間的 exclusive or 運算。算出所有項相加完的十進位表示後，可以用建好的 GF(16)反推回原本在 table 中是 α 的幾次方，例：$3=(0011)=1+\alpha=\alpha^4$。次方運算用次方項相加即可。

```cpp
int find_symdrome(vector<int> y_hat, int alpha_power, vector<int> G) //fi
{
    int symdrome = 0;
    vector<int> alpha_temp;

    for (int i = 0; i < coding_len; i++)
    {
        if(y_hat[i] == 1)
            alpha_temp.push_back( (i * alpha_power) % coding_len );

        /*
        for (int i = 0; i < alpha_temp.size(); i++)
            cout << alpha_temp[i] <<" ";
        cout << endl;
        */
    }

    for (int i = 0; i < alpha_temp.size(); i++)
    {
        //cout << symdrome <<"^"<< GF_table_decimal[alpha_temp[i]]<<"=";
        symdrome = symdrome ^ GF_table_decimal[alpha_temp[i]];
        //cout << symdrome <<" "<<endl;
    }

    return symdrome;
}
```

```cpp
#pragma region find_symdrome

//-------------------------------------------------------------------
//find symdrome -----------------------------------------------------
//-------------------------------------------------------------------
fill(symdrome_decimal.begin(), symdrome_decimal.end(), 0);
fill(symdrome_power.begin(), symdrome_power.end(), 0);

for (int i = 0; i < 2*error_correct; i++)
{
    symdrome_decimal[i] = find_symdrome(y_hat, i+1, GF_table_decimal);
    //cout << symdrome_decimal[i] <<" " ;
}
//cout << endl;
for (int i = 0; i < 2*error_correct; i++)
{
    for (int j = 0; j <= coding_len; j++)
    {
        if (symdrome_decimal[i] == GF_table_decimal[j])
            symdrome_power[i] = j;
    }
    //cout << symdrome_power[i] <<" " ;
}
//cout << endl;
```

- **Berlekamps**

因為 BCH code 是 binary code，所以我們可以用簡易版的 Berlekamps 來做運算。其中當 syndrome 全為 0 時，直接跳過更正的步驟。找 d_{u+1} 、ρ 、$\sigma^{u+1}(x)$ 的運算寫成子 function 在程式最下方。完整步驟則在第三張圖所示的各個折疊區塊當中，大致上就是將表格中所有的變數算出來，然後依據最終的$\sigma^{t}(x)$，將 α^0、α^2、...、α^{14} 一一帶入來找出 codeword 錯誤的位置。最後翻轉該位置的 bit 即可完成更正。

μ	$\sigma^{(\mu)}(x)$	d_μ	ℓ_μ	$2\mu-\ell_\mu$
-1/2	1	1	0	-1
0	1	S_1	0	0
1				
2				
.				
.				
t				

```cpp
int find_next_du(int current_step, vector<int> si

int find_rho(int current_step, vector<int> u2_lu,

vector<int> find_next_sigma_x(int current_step, i
                              vector<int> symdr
```

```cpp
if( symdrome_power ==  fifteen_vector_for_check)
{
    //cout << "no error!!!!!!!!!!!!!!!!!!!!!!!!!!!!!" <<endl;
}
else
{
    #pragma region Berlekamps…

    #pragma region find_error_position…

    #pragma region correct_error…

    #pragma region cout_for_checking_Berlekamps…

}
```

三、 Result

```
[Running] cd "c:\Users\CTLAB\Desktop\ECC\Final Project\2022\
SNR : 0   BER : 0.139089
SNR : 0.5   BER : 0.119177
SNR : 1   BER : 0.09965
SNR : 1.5   BER : 0.08161
SNR : 2   BER : 0.0645157
SNR : 2.5   BER : 0.04873
SNR : 3   BER : 0.0366886
SNR : 3.5   BER : 0.02542
SNR : 4   BER : 0.0164743
SNR : 4.5   BER : 0.0107314
SNR : 5   BER : 0.00639
SNR : 5.5   BER : 0.00355143
SNR : 6   BER : 0.00194429
SNR : 6.5   BER : 0.000881429
SNR : 7   BER : 0.000304286
SNR : 7.5   BER : 0.000162857
SNR : 8   BER : 5.2635e-005
SNR : 8.5   BER : 1.66871e-005
SNR : 9   BER : 3.31604e-006
SNR : 9.5   BER : 7.87527e-007
SNR : 10   BER : 1.51777e-007
SNR : 10.5   BER : 0
SNR : 11   BER : 0
SNR : 11.5   BER : 0
SNR : 12   BER : 0
```