電機專題結報

錯誤編碼偵測與更正

指導教授：

王忠炫 教授

學生:

何智興

專題動機

在現在這個物聯網、互聯網的時代，全世界有數億台行動裝置像是手機、電腦等都同時連到網路上，發射、接收訊號，因此對於訊號的傳輸的要求也越來越高。通訊技術也從過去的3G、4G到現在即將運用的5G技術，就是希望能更準確、更有效率的傳遞每個裝置的訊號。

通訊又分成類比與數位這兩種形式：類比訊號就是大自然中一切的訊號，像是聲音、影像等訊號皆屬於類比。以聲音為例，我們將聲音的大小轉成電壓的大小，得到的是連續的電壓變化，而這種連續的訊號就稱為類比訊號。數位訊號就是將連續的類比訊號加工成由0、1所組成的不連續訊號，也是目前多數電子產品訊號的表示方式，而數位訊號的優點包括方便儲存、傳送、加密與解密和容易偵錯、除錯，而這也是我們有興趣的主題。

通訊為我們生活帶來巨大方便，其中對於訊號的偵錯與除錯的技術最為感興趣，因此想更加地了解其背後原理和技術。

先備知識

編碼理論分成兩大區塊，資料壓縮和錯誤控制。資料壓縮嘗試壓縮資料的大小，讓傳輸和儲存資料時更加地有效率，像是我們常見的zip檔、rar檔皆是如此。錯誤控制則是增加位元到傳輸的資料之中來防止各種干擾，確保資料的穩定性。

在基本的通訊模型中，資料會在傳送端先經過編碼器之後傳入通道之中，到達接收端之後，經由解碼器還原出資料。在編碼端則是由三部分所組成的：source coding、 channel coding和modulation: source coding將資料轉換成數位的形式，而這樣的轉換是一對一的轉換，目的是消除冗餘的資訊，把不影響到資料傳輸結果的部分刪除，提升資料傳輸的效率；channel coding 則是增加冗餘資料來對抗有噪聲存在的環境；經source coding 和 channel coding 之後，資料是離散的形式，不利於傳輸，因此需要 modulation 將資料轉成連續的波，傳入通道至解碼端，而解碼端的運作方式與編碼端則完全相反，然而解碼完後的資料仍然有可能會與初始資料不相同，是因為訊號的失真和通道中的噪聲所造成，因此解碼器只能還原出資料的估計值。

現在廣泛使用數位訊號而不是類比訊號來傳輸的原因是因為數位訊號在傳輸上比類比訊號更加的可靠，因為當不可避免的干擾出現時，可以被偵測、還原成原本的資料；而數位訊號的表示方式是先將資料量化成數個等級，而每組資料則是以最相近的等級來表示，一種最簡單的方式是以四個位元來表示一組資料，前面三個位元是以二進位的方式來表示量值大小，最後一個位元是用來表示資料是正值或是負值，而每組資料被選中的機率相同，各組機率的加總和則為1。最優化的編碼方式是讓每組資料平均的位元數越少越好，而前提條件是不能讓各組資料的表示方式重複，滿足這樣條件的最小位元數稱為 least significant bit(LSB)。

除了上述提到的二進位表示方式之外，常見的資料表示方式還有八進位和十六進位，八進位的表示方式從0到7，十六進位的表示方式除了0到9之外，還加上了英文字母的A到F。

介紹完編碼的方式之後，現在介紹干擾的部分，我們從個位元發生錯誤的機會是獨立的機率開始，而這種個位元發生干擾的機率是獨立的模型稱為 white noise，雖然在真實情況中，會有特定的位元較為容易發生干擾，而且往往這些干擾會互相影響，但我們從較為簡單的 white noise開始探討干擾，再進一步地了解更複雜的情形。當一個位元發生錯誤的機率是p，而一組資料有l個位元，假設之中n個位元發生錯誤，機率為(l!)\*(p^n)\*((1-p)^(l-n))/(n!)\*((l-n)!)。

在學會如何計算一組資料發生幾個錯誤位元的機率後，介紹一種最簡單的偵測錯誤編碼，就是在各組資料的最後加入一個位元，使得整組資料各位元相加後對2取餘數為0，透過這種稱為 single parity check code，可以在一組資料中發生奇數個錯誤時偵測出來，而single parity check code的 redundancy rate 是 1/(n-1)，因此我們會直觀地認為當一組資料中包含越多的位元越有效率，但是相對地，它的可靠度就會降低，當然地，也不是越多的 redundancy bit 代表偵測到錯誤發生的能力一定越好，而是要設計出一種能有效地檢查方式，才能最有效率，因此要增加幾個和如何設計每個 redundancy bit 又是一門學問了。

另一種常見的codeword 是運用在電腦上面的ASCII code，每組codeword 由7個bit 組成，因此共能表示128種符號，但電腦中一個byte有8個bit，所以我們會在最後面用 single parity check code 來讓它具有 error detecting 的功能。

接著討論到較為複雜的情形，就是把 burst error 考慮進來之後，single parity check code無法那麼有效地偵測出是否有錯誤，因此學習了 simple burst error detecting code 的方式來處理當 burst error 的長度小於一組資料長度的情況。Simple burst error detecting code 就是把每個位置的bit相加，再對2取餘數，因此還會再產生一組資料，使得每個位置的bit相加之後對2取餘數為0，而這也是解碼端用來驗證是否有錯誤發生的方式。還有另一種方式，weighted code。就是當全部的符號數為質數(這裡表示為n)時，我們給每個符號一個相對應的數字，從0到 n-1，接著讓各符號對應的值乘上它們的權重然後加總，而每個符號的權重就是看他們是從最後一個符號開始數的第幾個符號加上一，接著再用n-(加總後的值對n取餘數)，並找出這個值所對應的符號，放在原本那些要傳送的符號後面，讓解碼端能夠用相同的方式來算出加總各符號的加權值對n取餘數是否為0的方式來偵測錯誤。

在介紹完 error detecting code之後，接著介紹 error correcting code，而資料中提到 repetition code 和 Hamming code這兩種方式。在repetition code中，以 three times repetition code 為例，就是每個bit總共重複了3次，是一種 single error correcting code，雖然大大地確保了資料的正確性，但是卻十分的沒有效率。Hamming code 是由G矩陣(generator matrix)和H矩陣(parity check matrix)所組成：G矩陣的功能就是將k個bit的資料轉換成n個bit(稱作codeword)，因此G矩陣的大小為k\*n；而H矩陣乘上codeword的轉置矩陣之後，能檢查這組codeword是否有受到干擾，如果乘完的結果為零向量，則這組codeword並未受到干擾而發生錯誤；若有發生錯誤，我們則能夠從H乘上codeword的轉置矩陣所得到的向量來推測最有可能是哪個位元發生了錯誤。而推測到底是哪個位元發生錯誤的方式，也可以透過 Venn diagram，如果一個圓之中有偶數個1，那麼這個圓裡代表的位元原則上是沒有問題，反之，若是有奇數個1，那麼這個圓有些位元是錯的，在算出每個圓的對與錯之後，就能輕易地找到是哪個位元發生了錯誤；除此之外，也會發現，表示圓是否正確的向量(正確的圓以0表示，有錯誤位元的圓以1表示)與H乘上codeword的轉置矩陣所得到的向量相同。我們已知H乘上codeword的轉置矩陣所得到的向量不為零向量代表存在著錯誤，而更正的方式就是找出H矩陣中哪個列向量與H乘上codeword的轉置矩陣所得到的向量一樣，然後在codeword所對應的位元加上1然後再對2取餘數來修正；而(7,4)Hamming code最多只能修正一個錯誤位元，這也是我們接著要討論的主題。

一種編碼方式最多能夠修正幾個位元的錯誤取決於Hamming distance，就是兩個codeword之間有幾個位元數是不相同的，假設任兩個codeword間的Hamming distance至少為3，那麼這樣的編碼方式能夠偵測到2個位元的錯誤(算法：最小的Hamming distance-1)，能夠更正1個位元的錯誤(算法：最小的((Hamming distance-1)/2)。而剛剛舉例的(7,4)Hamming code能夠更正一個錯誤，能偵測部分的兩個錯誤，但有些部分兩個錯誤的codeword則會在修正之後產生更多的錯誤，而(8,4)Hamming code就是在codeword的最後面多加一個位元，而這個位元的產生方式就是如果前面七個位元有偶數個1，那麼就補0；反之，如果有奇數個1，那麼則補1；(8,4)Hamming code優於(7,4)Hamming code的地方在於它除了能更正一個錯誤位元之外，它能在兩個位元發生錯誤時正確地偵測到，不會像(7,4)Hamming code可能更正出偏差更多的codeword。我們可以從這個例子發現到，增加redundant bit可以增加error correcting 的能力，但是會降低傳輸有效資料的效率。

除了從幾何的角度判斷一組編碼方式能夠偵測和更正幾個錯誤位元之後，接著從代數的角度來理解。假設一種編碼方式，它的information bit有k個位元，codeword有n個位元，我們令r=n-k，那麼它的特徵向量共有2^r種表示方式，而當我們希望這種編碼方式能夠更正一個錯誤位元時，那麼2^r要大於等於n+1，原因是因為當codeword沒有錯誤產生時，需要一組特徵向量來表示，加上當codeword有一個位元發生錯誤時，由於總共有n個位元，所以總共需要n組特徵向量來表示，因此更正一個錯誤位元至少需要n+1組相異的特徵向量來表示。當n=2^r-1，k=2^r-r-1時，那我們就可以稱之為(n,k)Hamming code。

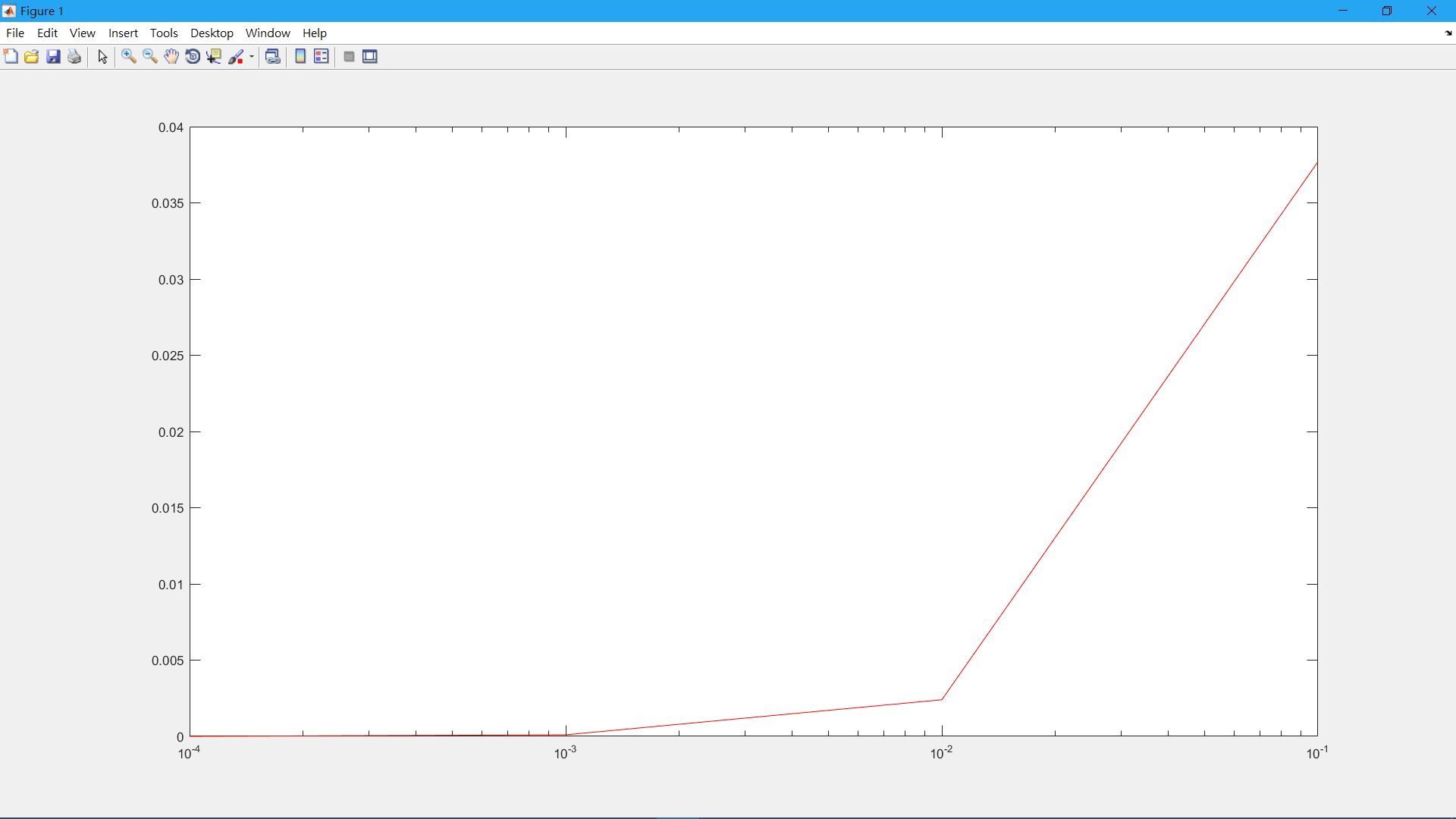
接著討論當我們希望能夠更正兩個錯誤位元的情況，當codeword沒有錯誤產生時，需要一組特徵向量來表示，加上當codeword有一個位元發生錯誤時，由於總共有n個位元，所以總共需要n組特徵向量來表示，再加上當codeword有兩個位元發生錯誤時，由於總共有n\*(n-1)/2種可能，所以總共需要n\*(n-1)/2組特徵向量來表示，因此要更正兩個錯誤位元，2^r要大於等於(n\*(n-1)/2)+n+1。

模擬與成果

透過C++模擬出一個包含編碼、解碼、產生隨機錯誤等過程的 (7,4)Hamming code，盡可能地讓模擬貼近現實的狀況。首先，程式先讀入G矩陣(大小：4\*7)，然後呼叫 G\_identity 這個函式判斷這個G矩陣是否為一個合法的矩陣，而一個合法的矩陣，矩陣的左方或是右方之中會包含一個單位矩陣，G\_identity函式會回傳一個值，如果這個單位矩陣存在於G矩陣的左方，回傳1；若在G矩陣的右方，回傳2；若G矩陣不存在單位矩陣，則回傳-1。接著用find\_H函式透過G矩陣來算出H矩陣(大小：3\*7)，當G矩陣的單位矩陣在G矩陣的左方時，H矩陣的單位矩陣在H矩陣的右方；相反地，當G矩陣的單位矩陣在G矩陣的右方時，H矩陣的單位矩陣在H矩陣的左方，而H矩陣扣除單位矩陣的部分則是G矩陣扣除單位矩陣部分的轉置矩陣。

接著建立在解碼端所需要用到的資料，因為要從codeword轉回去變成information bit需要乘上G的轉置矩陣再乘上G乘G轉置矩陣的反矩陣，所以先用G\_G\_trans函式來算出G乘G轉置矩陣，再將結果傳至find\_inverse函式之中，求出它的反矩陣。為了降低運算時間和成本，先利用build\_syndrome這個函式，對H乘上加入經干擾的codeword的轉置向量所可能產生的特徵向量，建立對應的修正方式，讓H乘上每組codeword後得到的特徵向量，能迅速的從這個已經建立的表找到對應的修改方法，而不用再跟每個H矩陣的列向量逐一比較，提升效率。

使用者能夠自己輸入每個位元發生錯誤的機率為多少，測試在各機率下一萬組資料經過這樣模擬的錯誤率，進行比較，而這一萬組資料的information bit是由電腦隨機產生所形成，產生完每組information bit之後送至to\_codeword的函式和G矩陣相乘得到對應codeword，接著將各組的codeword送至rand\_generator來決定哪些位元發生錯誤，發生的機率則是先前使用者所輸入的機率，而這個過程也是隨機的。把這些已經發生隨機錯誤的codeword送至get\_syndrome的函式中，得到每組codeword的特徵向量，然後利用先前build\_syndrome這個函式所建立的表，找到每組codeword的修正向量，進行修正。每組codeword修正完之後，用back\_to\_information\_bit的函式算出解碼後的information bit，接著與原始產生的information bit進行比較，看看是否相同。最後，在一萬組資料測試完之後，算出錯誤率。



橫軸代表解碼成information bit 後的錯誤率

縱軸代表每個位元發生錯誤的機率

收穫與展望

雖然在這個專題中學習到了許多編碼的基本原理以及如何去偵錯與更正，但是這些僅僅是編碼理論中最入門的小小一部份，仍然有許多更複雜和新穎的理論與模型需要去了解，才能更加了解編碼的全貌。

透過模擬模型的過程，將知識轉換成實際可運用的程式，除了開始思考如何降低運算成本、節省時間之外，對於設計程式的邏輯有很大改變，像是學會先建立資料庫，再去抓取資料，而不是每次重新運算，讓整個設計有更佳的效率。這樣的實務經驗並不是能從書本中獲得的，而是需要動手操作才能獲得的寶貴經驗。

有了這次專題的經驗之後，對於學問的研究也有了不一樣的認知，了解到做研究的過程中往往許多的知識都需要靠自己來發掘，然後了解，和一般地上課考試有很大地不一樣，不會有問題列在考卷上，而是要透過自己不斷的思考，而這也是讓我覺得很有趣的地方，因此如果之後還有機會，會想要繼續修專題，探索更多的知識。

程式碼

