UNIVERSITÀ
DEGLI STUDI
DI PADOVA

Università degli Studi di Padova

Dipartimento di Ingegneria dell'Informazione

Tesina di Laurea

PROGETTO DI DEMAPPER PER SISTEMI DVB-C2

Laureando: Bruno Cimoli

Relatore: Stefano Tomasin

Corso di Laurea Triennale in Ingegneria dell'Informazione

Anno Accademico 2011 / 2012

INDICE

INDICE DELLE FIGURE

INTRODUZIONE

Nelle telecomunicazioni digitali affinché un messaggio possa essere trasmesso deve prima essere modulato. Il processo di modulazione comprende la mappatura che associa al messaggio un segnale adatto alla trasmissione. Una volta arrivato a destinazione il segnale dovrà subire il processo inverso di demodulazione per ricostruire al meglio il messaggio originale con la conseguente demappatura del segnale tramite il "*demapper*".

Questa tesi propone e analizza alcuni algoritmi per il demapper delle trasmissioni di dati video digitali via cavo di seconda generazione DVB-C2 (Digital Video Broadcasting Cable-2), fra questi individua quelli più efficaci.

Nel primo capitolo sono introdotti i demapper con particolare attenzione alle definizioni dei vari elementi (standard, dispositivi, variabili, funzioni, modulazioni) necessari per capire e procedere nella realizzazione dei demapper.

Il secondo capitolo propone tre possibili algoritmi che calcolano le funzioni LLR (Log Likelihood Ratio) utilizzate dai demapper.

Nel terzo capitolo vengono verificate l'efficienza degli algoritmi e si analizzano i risultati ottenuti dai test.

Infine si riportano nelle appendici i programmi d'implementazione in linguaggio C degli algoritmi proposti, oltre gli acronimi e le variabili che compaiono nell'elaborato.

CAPITOLO 1

1 INTRODUZIONE AL SISTEMA DVB-C2

Il Digital Video Broadcasting (DVB) rappresenta l'insieme degli standard accettati a livello internazionale concepiti per lo sviluppo e per la trasmissione dei segnali televisivi digitali.

Il sistema DVB ha prodotto degli standard per ciascun mezzo trasmissivo (terrestre, satellite, via cavo), in particolare il DVB-C (Digital Video Broadcasting-Cable) per la trasmissione tramite cablatura.

Il DVB-C fu introdotto come standard europeo nel 1994, da allora le trasmissioni si sono evolute ottenendo risultati migliori sia in termine di efficienza, di correzione degli errori che di universalità dei dati (il DVB-C si concentra solamente sui formati MPEG). Per seguire questi miglioramenti nel 2008 si è introdotto il DVB-C2, dove il "2" indica "seconda generazione".

1.a Sistema DVB-C2

Il sistema DVB-C2 è riassumibile nello schema seguente:

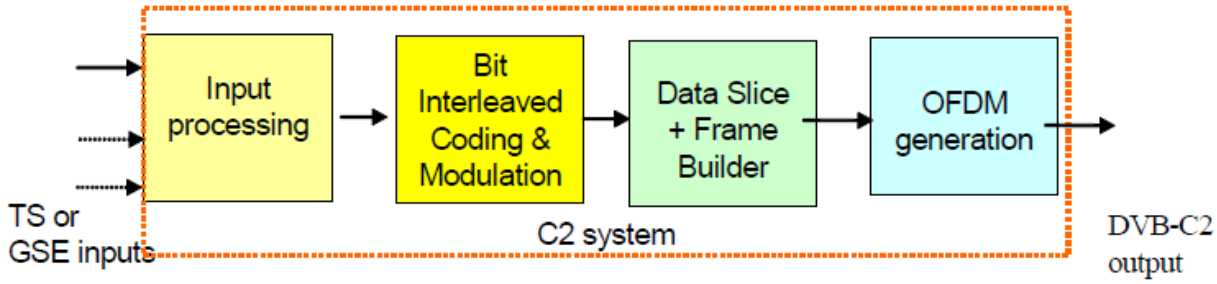

Figura 1.a.1 - Schema sintetizzato DVB-C2

Gli Input binari del sistema DVB-C2 sono prima elaborati in un processore (Input Processing), che non fa parte del sistema DVB-C2, in modo da renderli trattabili; usciti dal processore gli input sono introdotti nel Bit Interleaved Coding and Modulation (BICM) attraverso delle singole PLPs (Physical Layer Pipes), cioè dei canali digitali. Il flusso di dati è codificato in modo da ridurre le probabilità di commettere errori nella trasmissione (interleaved coding), per essere poi modulato, ovvero preparato alla trasmissione, e diviso in diversi frammenti (Frames) che saranno infine trasmessi tramite la tecnica Orthogonal Frequency-Division Multiplexing (OFDM).

Tutti i dati prima di essere scomposti in frame devono essere modulati, il sistema DVB-C2 per quest'operazione utilizza dei QAM (Quadrature Amplitude Modulation) che sarà spiegato nella seguente sottosezione del capitolo.

1.b Modulazione digitale QAM

Col termine modulazione s'intende il processo di trasformazione dell'informazione generata da una sorgente in un segnale che è consono alla trasmissione attraverso un canale fisico.

Nel caso delle trasmissioni digitali l'informazione è costituita da una sequenza di bit b_l di frequenza f_b e periodo T_b. La trasformazione della sequenza b_l in un segnale $s_{tx}(t)$ adatto alla trasmissione sul canale Ch è definita modulazione digitale e lo strumento che la esegue è il digital modulator.

Analogamente l'operazione inversa di ricostruzione della sequenza b_l', dal segnale $r(t)$ ricevuto dal canale, è definita demodulazione e lo strumento che la realizza è il digital demodulator.

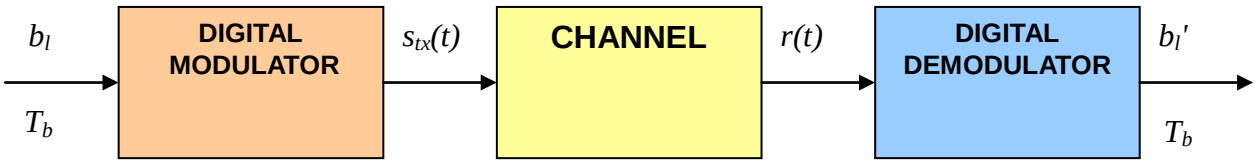

Figura 1.b.1 Schema modulazione digitale

1.b.1 Modulazioni Digitali

Il digital modulator, che si trova in trasmissione, trasforma una sequenza binaria b_l nel segnale $s_{tx}(t)$ continuo nel tempo e più adatto alla trasmissione. Per eseguire quest'operazione non banale si stabilisce un numero fisso n di bit che compongono le sequenze b_l e si applica una bit-map, cioè una relazione biunivoca tra tutte le possibili $M=2^n$ combinazioni di n bit e un set di altrettanti segnali $s_x(t)$ diversi tra loro.

Le possibili M combinazioni che compongono la bit-map sono definite codeword e i bit che le compongono sono identificati dall'indice i:

$$\text{codeword: } b_0\, b_1\, b_2\, b_3 .. b_i .. b_{n-1}$$

Il digital modulator è dotato di un componente detto encoder che applica una bit-map: associa alla generica sequenza b_l, che compone una codeword, un segnale $s_x(t)$ adatto alla trasmissione.

Il digital demodulator, che si trova in ricezione, ricostruisce la sequenza b_l' dal segnale ricevuto $r(t)$. Il digital demodulator è dotato di un decoder che esegue l'inverso della bit-map in trasmissione: decodifica il segnale continuo in una sequenza binaria.

I componenti del digital modulator e demodulator che si occupano di applicare la bit-map e la sua inversa sono definiti Bit-Mapper (BMAP) e Inverse Bit-Mapper (IBMAP) o più semplicemente: mapper e demapper.

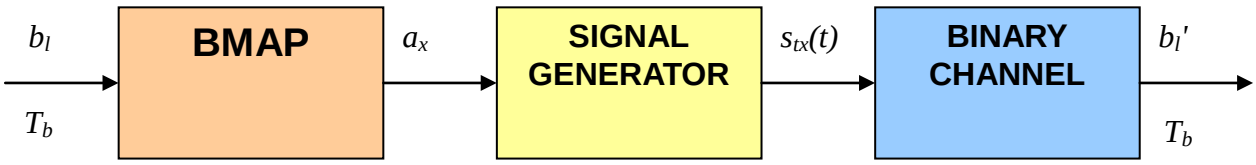

Figura 1.b.2 Digital modulator

dove ax è il valore associato alla codeword dal BMAP

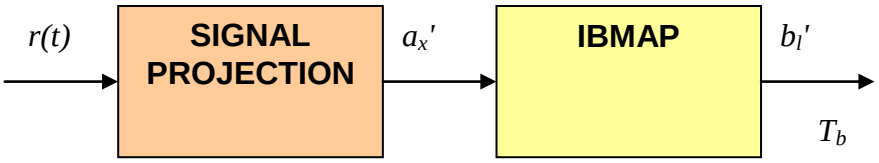

Figura 1.b.3 Digital demodulator

dove ax' è il valore ricostruito di ax dall'IBMAP

Purtroppo il canale e gli stessi dispositivi in trasmissione e ricezione sono soggetti a disturbi e attenuazioni che alterano il segnale $s_{tx}(t)$, in particolare gli si sovrappone il rumore, che è una variabile aleatoria. Il segnale $r(t)$ non è quindi una grandezza determinata ma aleatoria.

Per contrastare questa incertezza e per evitare eventuali errori è necessario scegliere un opportuno set di segnali $s_x(t)$ e un'opportuna bit-map tali che in ricezione si scelga la stessa codeword modulata in trasmissione.

Queste scelte determinano il tipo di modulazione digitale utilizzato per la trasmissione.

Alcune modulazioni digitali utilizzano un set segnali $s_x(t)$ tutti combinazioni lineari di due funzioni ortonormali $\varphi_1(t)$ e $\varphi_2(t)$ che costituiscono la base di tale set:

$$s_{tx}(t) = I\varphi_1(t) + Q\varphi_2(t)$$

I e Q sono i due coefficienti reali che identificano la particolare combinazione lineare a cui corrisponde $s_x(t)$. Scelte le coppie I_x e Q_x per ogni segnale del set è possibile stabilire una relazione biunivoca tra i numeri complessi $z=I_x+iQ_x=(I_x,Q_x)$ e i segnali $s_x(t)$; tale relazione può essere utilizzata per costruire una bit-map attraverso una costellazione:

Definizione: Dato un set di 2^n segnali associabili a vettori del tipo $x(Ix,Qx)$, sia inoltre associata a ciascuno di tali vettori una delle possibili 2^n codeword di n bit, si definisce costellazione del set di segnali la loro rappresentare vettoriale.

Le coppie di coefficienti I_x *e* Q_x nella costellazione individuano dei punti detti della costellazione e a ognuno di essi è associata la codeword corrispondente al segnale $s_x(t)$. Ecco alcuni esempi di costellazioni:

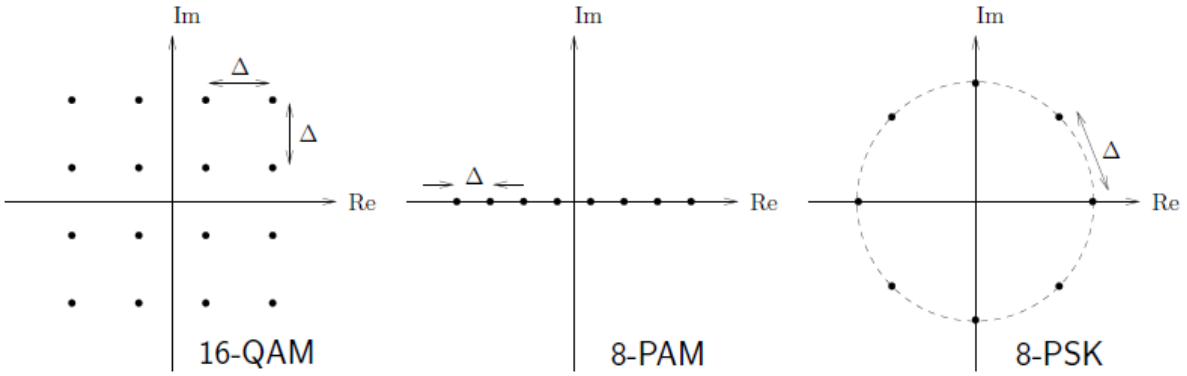

Figura 1.b.4 Costellazioni 16-QAM con n=4, 8-PAM e 8-PSK con n=3

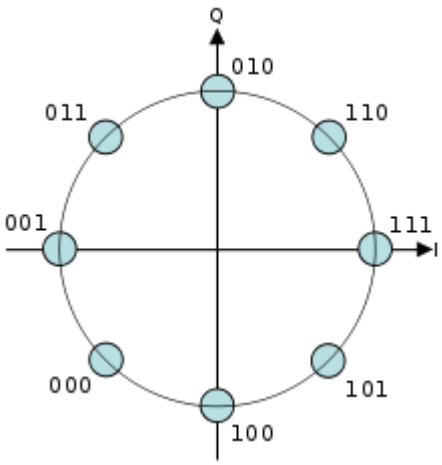

Figura 1.b.5 Costellazione 8-PSK con n=8 e codeword associate ai punti

Esistono numerose tipologie di modulazioni digitali, in particolare si definisce M-QAM (M-ary Quadrature Amplitude Modulation) quella modulazione che utilizza segnali di trasmissione $s_x(t)$ che sono combinazioni lineari di due funzioni $\varphi_1(t)$ e $\varphi_2(t)$ ortonormali tra loro:

$$s_x(t) = I_x\varphi_1(t) + Q_x\varphi_2(t)$$

$$\text{con } I_x, Q_x \epsilon \{-\sqrt{M}+1, -\sqrt{M}+2, \ldots . \sqrt{M}-1, \sqrt{M}-2, \}$$

Sono riportati alcuni esempi di costellazioni QAM che sono caratterizzate dalla disposizione a quadrato dei punti *x* della costellazione.

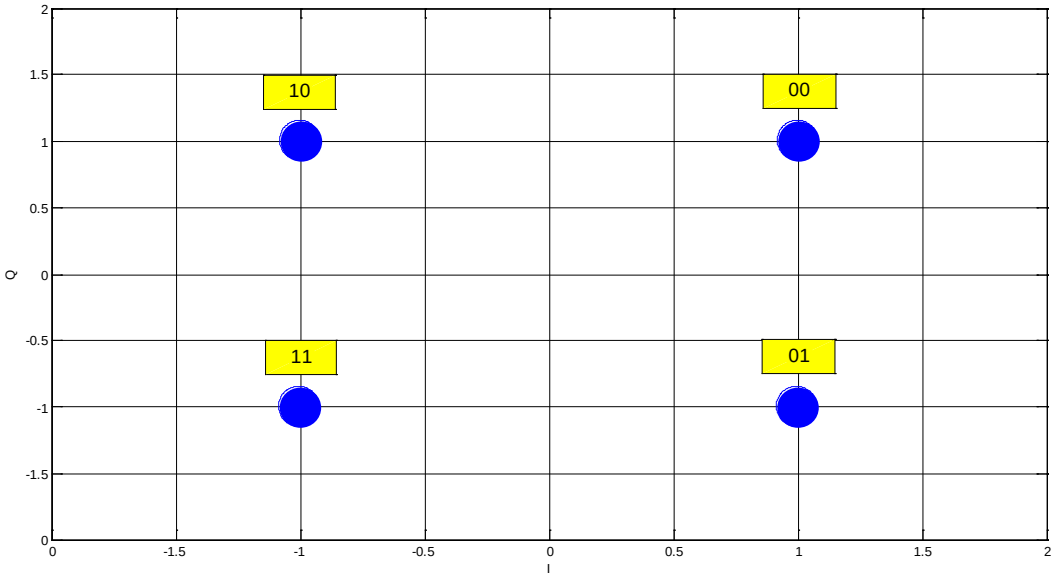

Figura 1.b.6 Costellazione 4-QAM, con n=2

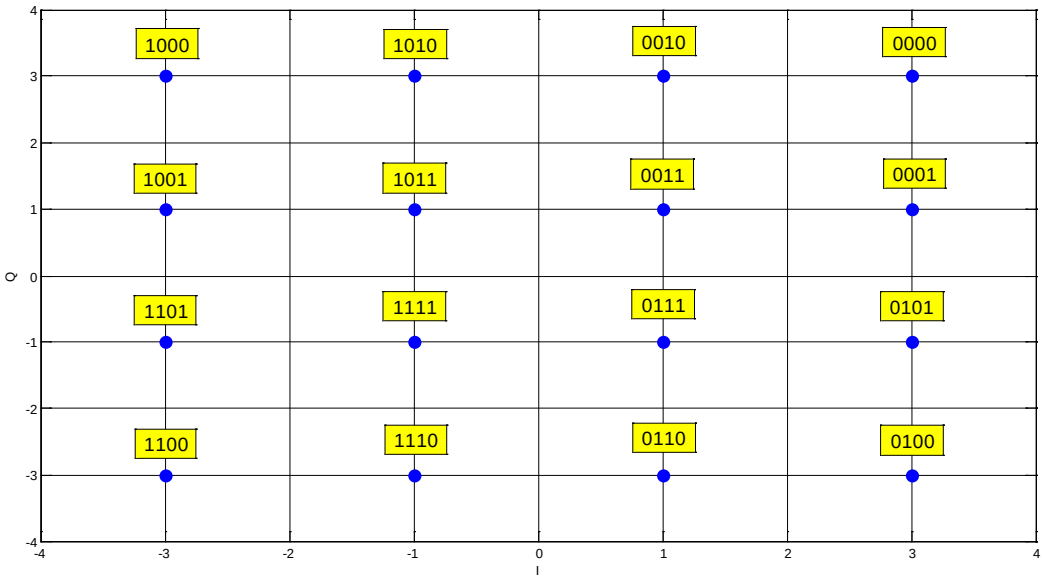

Figura 1.b.7 Costellazione 16-QAM , con n=4

In trasmissione il digital modulator riceve in ingresso una codeword di n bit e usando la bit-map decide quale degli $s_x(t)$ inviare; le coordinate degli $s_x(t)$ rispetto alla base $<\varphi_1(t),\varphi_2(t)>$ sono rappresentate nella costellazione dai punti x. Il segnale scelto per la trasmissione è indicato come $s_{tx}(t)$ e dal punto $tx(I_m,Q_m)$.

In ricezione il digital demodulator utilizza un decoder per risalire alla codeword che con maggiore probabilità aveva selezionato l'encoder. Il decoder riceve in ingresso il segnale *r(t)* (già filtrato e demodulato), che differisce da $s_{tx}(t)$ sostanzialmente per l'attenuazione del canale e per il rumore.

Nota l'attenuazione è possibile ottenere i coefficienti ρ_I e ρ_Q che rappresentano l'effetto dell'attenuazione su I_m *e* Q_m.

Il rumore invece agisce come il contributo *w(t)* (componente del rumore sulla banda di *r(t)*) che si somma al segnale trasmesso:

$$r_m(t) = I\varphi_1(t) + Q\varphi_2(t) = \rho_I(I_m + w_I)\varphi_1(t) + \rho_Q(Q_m + w_Q)\varphi_2(t)$$

$$\text{con } w(t) = w_\varphi(t) + w_\theta(t) = w_I\varphi_1(t) + w_Q\varphi_2(t) + w_\theta(t),$$
$$w_\varphi(t) = w_I\varphi_1(t) + w_Q\varphi_2(t)$$

La componente $w_\theta(t)$ del rumore è facilmente eliminabile tramite dei filtri e le attenuazioni ρ_I e ρ_Q sono anche loro eliminabili con un fattore correttivo.

Poiché ρ_I e ρ_Q sono semplificabili con un fattore correttivo su *r* si è deciso di semplificarli ponendoli uguali a 1.

La decodifica che consiste nello stabilire quale codeword è stata trasmessa osservando il segnale in ricezione, corrisponde a identificare quale $x(I_x,Q_x)$ coincide con $tx(I_m,Q_m)$ disponendo di $r(I,Q)$.

Per risolvere tale problema il decoder deve prendere una decisione: stabilire, ricevuto *r*, quale *x* sia con maggior probabilità coincidente con *tx*; per automatizzare e rendere più efficace tale scelta si divide la costellazione in regioni di decisione R_x a cui è associato un punto della costellazione *x*:

Definizione: Si definisce R_{x^*} regione di decisione del punto *x** l'insieme di tutti e i soli punti *r* della costellazione tali che le distanze:

$d(r,x^*) \leq d(r,x)$ per ogni *x* della costellazione

Ecco un esempio di due costellazioni, una QAM e l'altra PAM divise nelle loro zone di decisione:

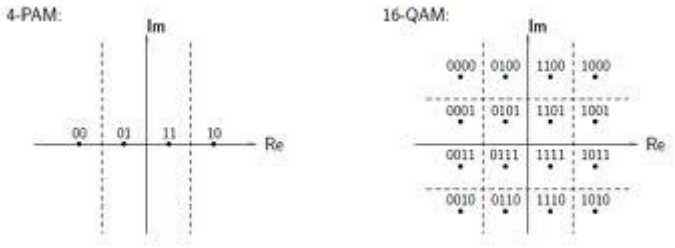

Figura 1.b.8 Costellazione 16-QAM con n=4 e 4-PAM con n=2

Se un punto $r(I,Q)$ cade nella regione Rx allora il decoder decide che il punto trasmesso $tx(I,Q)$ coincide con $x(I_x,Q_x)$ e di conseguenza trasmette in uscita la codeword associata a x.

L'utilizzo delle regioni di decisione non è l'unico criterio per stabilire le codeword in uscita, infatti ne esistono molti altri. I vari criteri di decisione sostanzialmente si dividono in due categorie:

1) Le hard-decision, come quello delle regioni di decisione, che stabiliscono direttamente dagli ingressi con quale codeword ricostruire il messaggio. Questo metodo di scelta è utile se i bit del messaggio sono indipendenti tra loro, altrimenti si perderebbero le informazioni contenute nelle eventuali dipendenze tra i bit.

2) Le soft-decision che non decidono subito con quali codeword ricostruire il messaggio basandosi esclusivamente sugli ingressi ma che integrano le informazioni ottenute dagli ingressi con quelle derivate dalle dipendenze tra i bit. Ovviamente se i bit sono tutti indipendenti tra loro questo tipo di decisione è inutile.

Nei sistemi che adottano il DVB-C2 il messaggio prima di essere modulato è soggetto all'interleaved coding, che elabora un nuovo messaggio codificato in cui compaiono varie dipendenze tra i bit, pertanto il criterio da adottare deve essere una soft-decision.

1.b.2 Codici Gray

In una qualsiasi modulazione digitale che utilizza costellazioni è rilevante anche l'assegnazione delle codeword ai punti della costellazione.

Dato un set ordinato di elementi a cui si vuole associare biunivocamente delle sequenze binarie, come delle codeword, il criterio o la regola con cui è stabilito a quale elemento deve essere associata una sequenza è definito codice o codifica binaria. Vi sono numerosi tipi di codici binari, uno dei più usati è quello Gray.

Definizione: Un codice binario si dice Gray se le sequenze che lo compongono sono a lunghezza fissa e sono ordinate in modo tale che due configurazione adiacenti differiscono per un unico bit; questa proprietà è nota come cambio 1.

Codice Gray a 3 bit
000
001
011
010
110
111
101
100

Si possono usare codici di Gray di tutte le lunghezze: il codice di lunghezza n è costituito da tutte le 2^n sequenze di n bit e consente di rappresentare tutti gli interi da 0 a $2^n - 1$.

Nella modulazione QAM è possibile disporre le codeword in modo da ottenere un codice Gray; per ogni punto x della costellazione la sua corrispondente codeword deve differire per un solo bit da quelle di tutti punti a distanza minima da x.

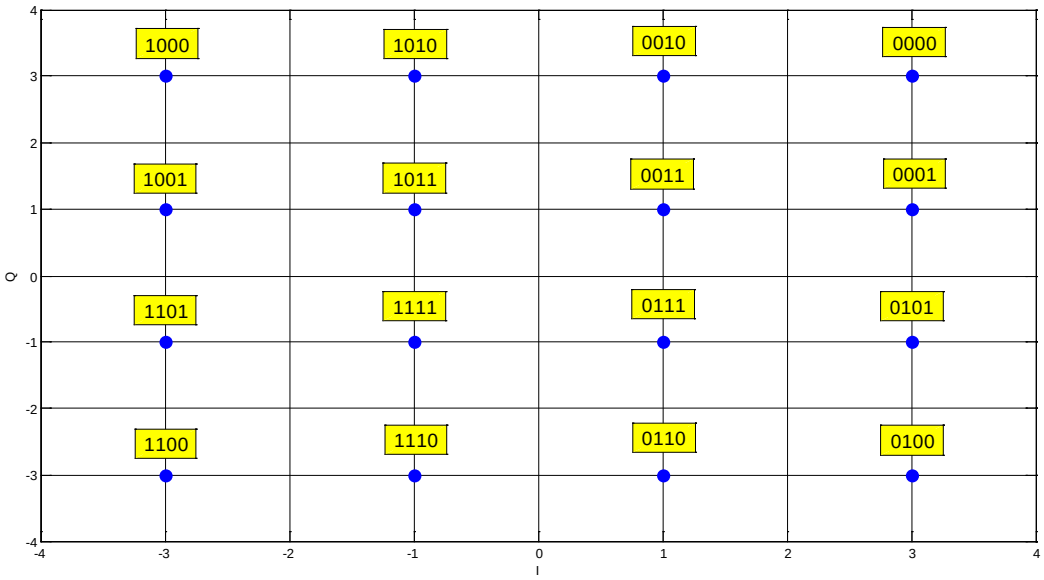

Figura 1.b.9 Costellazione 16-QAM con codce Gray adottato dal DVB-C2

Nel sistema DVB-C2 tutti i QAM mapper e demapper adottano codici Gray, perché grazie alla proprietà cambio 1 la probabilità di sbagliare più di un bit è molto bassa o almeno molto inferiore a quella di sbagliarne uno solo.

I sistemi DVB-C2 utilizzano un particolare codice Gray per ogni M-QAM, tale scelta non comporta una limitazione poiché la maggior parte dei codici utilizzati sono equivalenti.

Nella [1] è riportata la tecnica più diffusa per costruire un codice Gray su una costellazione QAM; essa consiste nel suddividere gli n bit in due gruppi: i primi $n/2$ per la componente in quadratura I e gli altri $n/2$ per la componente in fase Q. Si generano con i due gruppi di bit due codici Gray di

lunghezza *n/2* lungo i due assi. Combinando le codeword delle due componenti I e Q si ottiene una costellazione M-QAM.

Con questo procedimento si possono generare diversi codici Gray al variare della scelta dei due gruppi: quali e come i bit sono disposti in I o Q.

I codici Gray generati con questa tecnica sono equivalenti, infatti è sufficiente una permutazione dei bit per passare da un codice all'altro.

Anche il codice Gray dei sistemi DVB-C2 è realizzato con questa tecnica, in particolare:

$b_0\ b_2\ b_4 \ldots b_{n-2}$ codificano l'asse I

$b_1\ b_3\ b_5 \ldots b_{n-1}$ codificano l'asse Q

Non si perde quindi generalità se si utilizza lo specifico codice Gray dei sistemi DVB-C2.

1.c Funzioni LLR

Nel sistema DVB-C2 tutti i dati in trasmissione sono mappati utilizzando una bit-map del tipo QAM, in particolare secondo il tipo di dato sono utilizzati:

1) $n=2$, 4-QAM
2) $n=4$, 16-QAM
3) $n=6$, 64-QAM
4) $n=8$, 256-QAM
5) $n=10$, 1024-QAM
6) $n=12$, 4096-QAM

Il QAM demapper non applica il criterio standard delle regioni di decisione perché perderebbe l'informazione dell'interleaved coding; il demapper utilizza invece la funzione LLR (Log Likelihood Ratio) come soft-decision.

Definizione: Dato un sistema di trasmissione digitale che utilizza un M-QAM mapper e assunto che vi sia rumore nel canale di trasmissione, ricevuto il punto $r(I,Q)$ nella costellazione del QAM demapper, è possibile definire per ogni bit b_i della codeword come LLR:

$$LLR(b_i, I, Q) = log\left(\frac{P[b_i=1|I,Q]}{P[b_i=0|I,Q]}\right) \quad (1)$$

Con $P[bi=1|(I,Q)]$ e $P[bi=0|(I,Q)]$ probabilità composte, cioè le probabilità che il bit i-esimo della codeword in trasmissione valga 1 o 0 con la certezza di aver ottenuto il punto $r(I,Q)$.

Definiti i set C_{i1} e C_{i0} come gli insiemi di codeword, e dei corrispondenti punti della costellazione, in cui il bit i-esimo assume il valore 1 o 0.

La probabilità composta di ottenere in ricezione un punto $r(I,Q)$ dopo aver trasmesso il punto x della costellazione $P[(I,Q)|x]$, è data dalla formula:

$$P[I, Q|x\ trasmesso] = \frac{1}{2\pi\sigma^2} e^{-\frac{(I-I_x)^2+(Q-Q_x)^2}{2\sigma^2}} \quad (2)$$

Se si considerano invece le probabilità composte di ricevere il punto $r(I,Q)$, partendo dall'ipotesi che sia noto il valore di bit b_i della codeword trasmessa: $P[(I,Q)|b_i=1]$ e $P[(I,Q)|b_i=0]$ e supponendo che le probabilità di estrarre una delle codeword in cui b_i assume il valore noto sono equiprobabili, si possono ottenere le seguenti relazioni:

$$P[\, I,Q \mid bi = 1\,] \;=\; \sum_{x \epsilon C_{i1}} \frac{e^{-\frac{(I-I_x)^2+(Q-Q_x)^2}{2\sigma^2}}}{2^n \pi \sigma^2} \qquad (3)$$

Per semplificare le espressioni successive si pone:

$$P[(I,Q)|b_i = 1] = p_1 \; e \; P[(I,Q)|b_i = 0] = p_0$$

Applicando la definizione di probabilità condizionata:

$$P[bi = 1|(I,Q)] = \frac{P[bi = 1 \; \Lambda \; (I,Q)]}{P[(I,Q)]\Lambda} P[(I,Q)|bi = 1] =$$

$$= P[b_i = 1 \; \Lambda \; (I,Q)]/P[(b_i = 1)]$$

Per la regola di Bayes:

$$P[b_i = 1|(I,Q)] = P[(I,Q)|b_i = 1] \; \frac{P[bi = 1]}{P[(I,Q)]}$$

Lo stesso ragionamento è applicabile anche al caso $b_i=0$:

$$P[b_i = 0|(I,Q)] = P[(I,Q)|b_i = 0] \; \frac{P[bi = 0]}{P[(I,Q)]}$$

Passando alla formula del LLR(b_i):

$$LLR(b_i) = \log \left(\frac{\sum_{x \epsilon C_{i1}} e^{-\frac{(I-I_x)^2+(Q-Q_x)^2}{2\sigma^2}}}{\sum_{x \epsilon C_{i0}} e^{-\frac{(I-I_x)^2+(Q-Q_x)^2}{2\sigma^2}}}\right) \quad (4)$$

Si è supposto che $P(b_i=1)=P(b_i=0)$, che equivale a dire che l'estrazione di una delle codeword di C_{i1} è equiprobabile all'estrazione di una di quelle di C_{i0}.

Con la (4) è possibile calcolare gli LLR iterativamente senza dover calcolare le probabilità; in pratica è realizzabile un algoritmo per il calcolo degli LLR di tutti gli *n* bit per i QAM demapper del sistema DVB-C2 che utilizza la (4).

CAPITOLO 2

2 ALGORITMI E PROGRAMMI LLR

Nel capitolo precedente si è definito il sistema DVB-C2 e i suoi M-QAM mapper e demapper: 4-QAM, 16-QAM, 64-QAM, 256-QAM, 1024-QAM e 4096-QAM. Si è posta particolare attenzione alla funzione LLR adottata dai demapper.

In questo capitolo si realizzeranno tre algoritmi e tre programmi che li implementano, per calcolare i valori degli LLR(b_i) con la (4) dagli ingressi *r(I,Q)* e σ.

Si ricorda che i punti della costellazione M-QAM si dispongono in forma quadrata; si definisce con *m* il numero di punti che costituiscono un lato di questo quadrato tale che:

$$m = 2^{\frac{n}{2}}$$

2.a Stato dell'arte

Prima di introdurre i tre algoritmi è opportuno sintetizzare lo stato dell'arte sui demapper soft-decision che utilizzano gli LLR.

La maggioranza delle tecniche per il calcolo degli LLR non utilizzano la (4), definita nella [2] come "exact method"; adottano invece una sua semplificazione:

$$LLR(b_i) \cong \log\left(\frac{\max_{x\epsilon C_{i1}} e^{-\frac{|r-x|^2}{2\sigma^2}}}{\max_{x\epsilon C_{i0}} e^{-\frac{|r-x|^2}{2\sigma^2}}}\right) = \frac{1}{2\sigma^2}\left(|r-x_0|^2 - |r-x_1|^2\right) \quad (5)$$

$$\text{con } x_1 = \max_{x\epsilon C_{i1}} e^{-\frac{|r-x|^2}{2\sigma^2}} \text{ e } x_0 = \max_{x\epsilon C_{i0}} e^{-\frac{|r-x|^2}{2\sigma^2}}$$

La (5) è definita come "simplified method" [2] ed è una buona approssimazione nel caso di un canale con un rapporto segnale rumore SNR (Signal-to-Noise Ratio) elevato [3] e riduce notevolmente la complessità della (4).

Poiché la (5) non è altrettanto buona come approssimazione con valori più modesti del SNR e poiché sono già stati trovati dei metodi per calcolare gli LLR con la (5) o sue ulteriori semplificazioni si è stabilito di utilizzare la (4) nello svolgersi dell'elaborato.

Sono comunque elencati alcuni di questi metodi per il calcolo degli LLR prestando particolare attenzione agli aspetti utili all'obiettivo della tesi.

Il primo metodo esposto nella [2] utilizza una modulazione diversa dal QAM; infatti la (1) è definita anche per le modulazioni PAM e PSK e conseguentemente anche la (4) e la (5) sono valide nelle due modulazioni. E' noto che una costellazione M-QAM quadrata è sempre scomponibile in m costellazioni PAM. Sempre nella [1] è proposto un algoritmo che calcola un'approssimazione della (5) nelle costellazioni PAM; tale algoritmo può essere adoperato anche su una costellazione QAM applicandolo a quelle PAM che la compongono e selezionando il massimo tra gli LLR ottenuti dalle PAM si ottiene il valore più vicino alla (5).

Nella [3] è proposto un algoritmo che calcola un'approssimazione della (5) nelle costellazioni 16-QAM e 64-QAM con il codice Gray adottato dallo standard HIPERLAN/2 (High Performance Radio Local Area Network type2). L'algoritmo sfrutta una scomposizione dei set C_{it} in "subsets", composti di sole righe o colonne della codifica Gray dell'HIPERLAN/2; con la divisione in subset si può stabilire il set di appartenenza degli x e si possono sostituire le distanze $|r\text{-}x|$ con le loro proiezioni sugli assi I e Q.

Nella [4] è definito il metodo "Simplified Bit Metrics" su una costellazione M-QAM nei sistemi BICM SISO (Single Input and Single Output) e MIMO (Multiple Input and Multiple Output). Il "Simplified Bit Metrics" calcola un'approssimazione della (5) e utilizza le proprietà dei codici Gray descritte nella [1]; in particolare sfrutta quella sull'indipendenza di metà dei bit dalle colonne e dell'altra metà dalle righe riuscendo a migliorare le prestazioni temporali dell'algoritmo da $O(m^2)$ a $O(m)$.

2.b Calcolo LLR con codifica generica

Il metodo più elementare per calcolare gli LLR degli n bit della codeword con ingresso $r(I,Q)$ consiste nell'applicare direttamente per tutti i bit la (4).

La funzione LLR dipende dalle distanze di r rispetto ai punti x della costellazione, pertanto è necessario calcolare tali distanze che saranno poi inserite negli esponenziali della formula.

Eseguito il calcolo bisogna stabilire quali x stanno in C_{i1} e quali in C_{i0} per inserire l'esponenziale a numeratore o a denominatore. Per risolvere questo problema si adotta il procedimento più semplice, ma non il più efficiente, il quale consiste nel memorizzare in una matrice tutte le codeword per poi controllare direttamente il valore del bit i-esimo della codeword esaminata per stabilire se appartiene a C_{i1} o C_{i0}.

Si noti che in tale procedura non si è mai sfruttata l'informazione che il codice del QAM è di tipo Gray. Quest'omissione rende lenti gli algoritmi che implementano tale metodo, ma in compenso indipendenti dal codice della costellazione, sono pertanto validi per un codice qualsiasi. Tali algoritmi saranno indicati come LLR generici.

2.b.1 Algoritmo LLR generico

Un algoritmo per il calcolo degli LLR generico non utilizza l'informazione che il codice è di tipo Gray e si limita ad applicare la formula (4).

L'algoritmo LLR generico esegue essenzialmente tre semplici operazioni:

1) Memorizza le M codeword di n bit in una matrice B mxm di vettori di n bit; il vettore nella cella $B[0,0]$ coincide con il punto nell'angolo in basso a sinistra della costellazione (scelta non casuale), il primo indice l scorre lungo le righe e il secondo j lungo le colonne, vi è anche un terzo indice i che indica il bit i-esimo di cui si sta calcolando LLR.

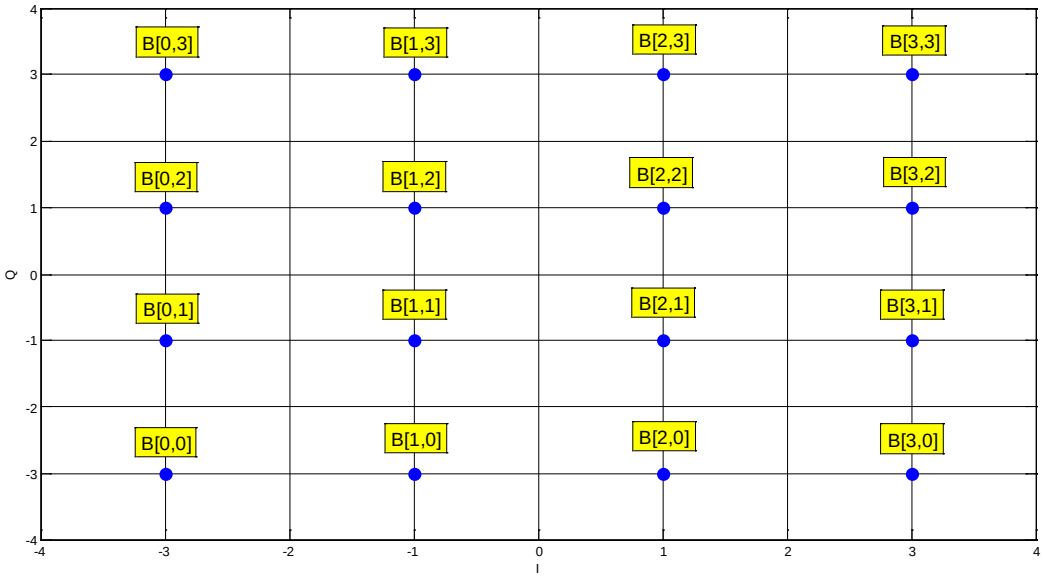

Figura 2.b.1 Costellazione con indici matrice B[l,j]

2) Calcola le distanze di $r(I,Q)$ da ogni punto della costellazione. Per ridurre le operazioni da svolgere si calcolano le proiezioni sull'asse reale e immaginario delle distanze:

   $d_{ix} = (I - I_x)$, $d_{qx} = (Q - Q_x)$ con I_x, Q_x coordinate del punto x

   Il vettore d'ingresso $r(I,Q)$ è l'unica variabile che cambia, mentre le coordinate I_x,Q_x di tutti i punti x della costellazione restano fissi.

   Le coordinate I_x,Q_x sono inoltre numeri interi dispari contenuti in $[-m+1, m+1]$.

   E' conveniente trovare un algoritmo che abbia già memorizzato le coordinate dei punti

   x al variare di n numero di bit; ecco un possibile algoritmo che soddisfa tali requisiti:

```
“
 //distanze() calcola le proiezioni su uno degli assi delle distanze da Vin degli x costellazione

 distanze(L,m)  //restituisce il vettore v, L può essere I o Q
 {
   vettore v  //dove memorizzo le distanze dlx

   p=(L-1+m)/2  //trasformazione che fa coincidere gli Lx all'indice i

   for(j=0,i<m,i++)          //calcolo distanze dlx
    { v[j]=2*(L-j)  }

    return v   }
”
```

Eseguendo tale metodo *con L=I, Q* si ottengono due vettori v_I e v_Q di lunghezza *m* che contengono tutti i valori di d_{ix} e d_{qx}. Si può osservare che le scelte nell'ordinamento delle codeword in *B* e delle distanze in v_I e v_Q implicano che le coordinate $v_I[l]$, $v_Q[j]$ individuano un punto *x* della costellazione e la sua codeword è memorizzata in $B[l][j]$.

3) Calcolar LLR(b_i) con i vettori v_I, v_Q e la matrice *B*. Siccome:

$LLR(bi) = \log\left(\frac{p_1}{p_0}\right)$ con $p_1 = P[(I,Q)|b_i = 1]$ e $p_0 = P[(I,Q)|b_i = 0]$

è necessario calcolare p_1 e p_0; per ogni punto *x* della costellazione si ha che:

$I_x = v_I[l]$, $Q_x = v_Q[j]$, $B[l][j]$ è la codeword di *x* con *k,,j*<*m*

tale proprietà ci premette di calcolare facilmente p_1 e p_0:

```
“

          //calcolo p1, p0 e LLR(bi)
           for(k=0,k<m,k++)    //scorro lungo le righe della costellazione
           for(j=0,j<m,j++)  //scorro lungo le colonne
              {
                 if(B[i][j][i]=1)  //condizione x sta in C1i
                   p1=p1+exp(-((vI(k))^2+(vQ(k))^2)/σ^2)  //calcolo p1, si veda formula (4)
                else                                          //altrimenti x sta in C0i
                   p0=p0+exp(-((vI(k))^2+(vQ(k))^2)/σ^2)
               }
           LLR[i]=log(p1/p0);  //calcolo LLR bit i-esimo
“
```

2.b.2 Implementazione in linguaggio C e analisi dell'algoritmo

Nell'appendice A è presentata una possibile implementazione dell'algoritmo LLR generico in linguaggio C. Si nota che l'inizializzazione della matrice B tramite il metodo codifica() è momentaneamente tralasciata perché sarà approfondita in seguito.

Gli ingressi $r(I,Q)$, n e σ vengono inseriti da prompt.

Dall'analisi teorica dei tempi di esecuzione si apprende che il programma compie tre operazioni critiche:

1) inizializzazione B: $O(2^n)$
2) calcolo distanze rispetto I e Q: $O(m) = O(2^{\frac{n}{2}})$
3) calcolo LLR: $O(nm^2) = O(n2^n)$

Il programma è un $O(n2^n)$ che ha una crescita esponenziale al variare di n numero di bit. Fortunatamente nel sistema DVB-C2 il massimo valore di n è 12, ciò rende quasi accettabile la prestazione di LLR generico, per lo meno per valori bassi di n.

2.b.3 Conclusioni

Si può affermare che data una costellazione a n bit, con una codifica qualsiasi, è sempre possibile scrivere un programma per il calcolo degli LLR dei bit applicando direttamente la (4). Esso però non ha buone prestazioni in termini di tempo, infatti è un $O(n2^n)$ con crescita esponenziale rispetto a n.

E' necessario per il nostro sistema DVB utilizzare l'informazione che tutte le codifiche delle modulazioni QAM sono di tipo Gray, in modo da sfruttare le simmetrie che essa comporta per migliorare le prestazioni del programma.

2.c Algoritmo LLR con codifica di tipo Gray

Per migliorare il programma precedente e velocizzarne l'esecuzione è necessario utilizzare l'informazione che il codice utilizzato dalla modulazione QAM è di tipo Gray. E' riportato un esempio di modulazione QAM a 4 bit ($b_0\ b_1\ b_2\ b_3$) con codifica Gray:

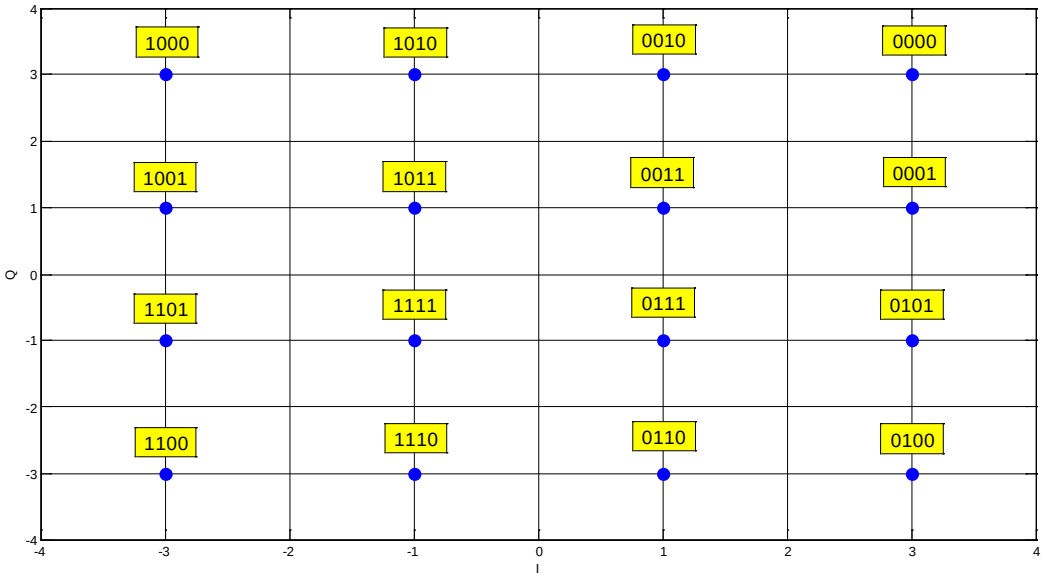

Figura 2.c.1 Costellazione 16-QAM con codifica Gray del sistema DVB-C2

Si nota che le codeword in una stessa colonna differiscono solo per i bit con indici dispari e analogamente le codeword di una stessa riga differiscono solo per i bit a indici pari.

Quest'osservazione rende quindi possibile per ogni bit i-esimo della codifica scomporre le regioni C_{i1} e C_{i0} in sottoinsiemi di codeword, che corrispondono alle righe della costellazione se i dispari o alle colonne se i pari:

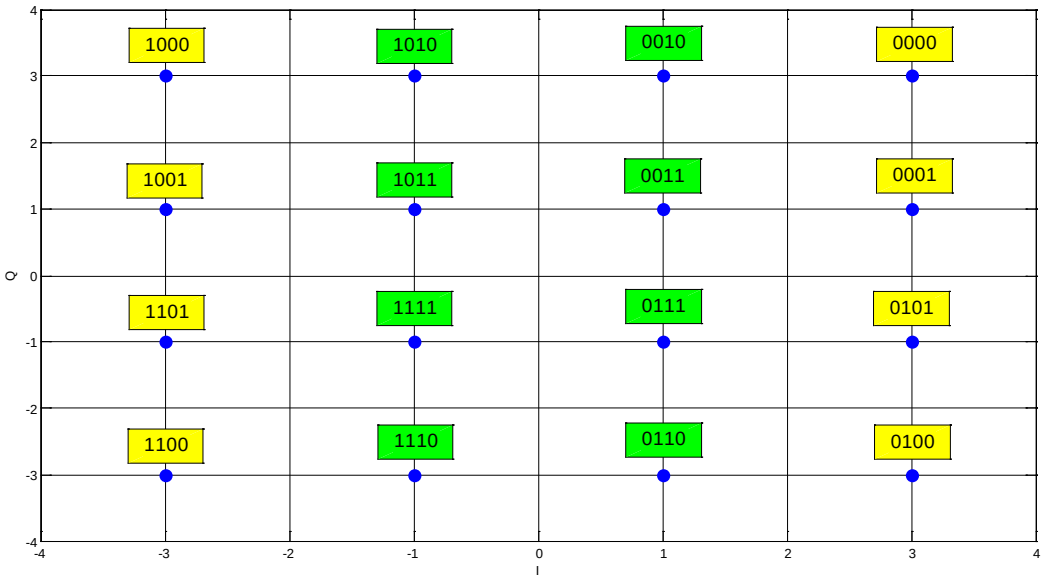

Figura 2.c.2 Costellazione 16-QAM, set di b2, verde C21 e giallo C20

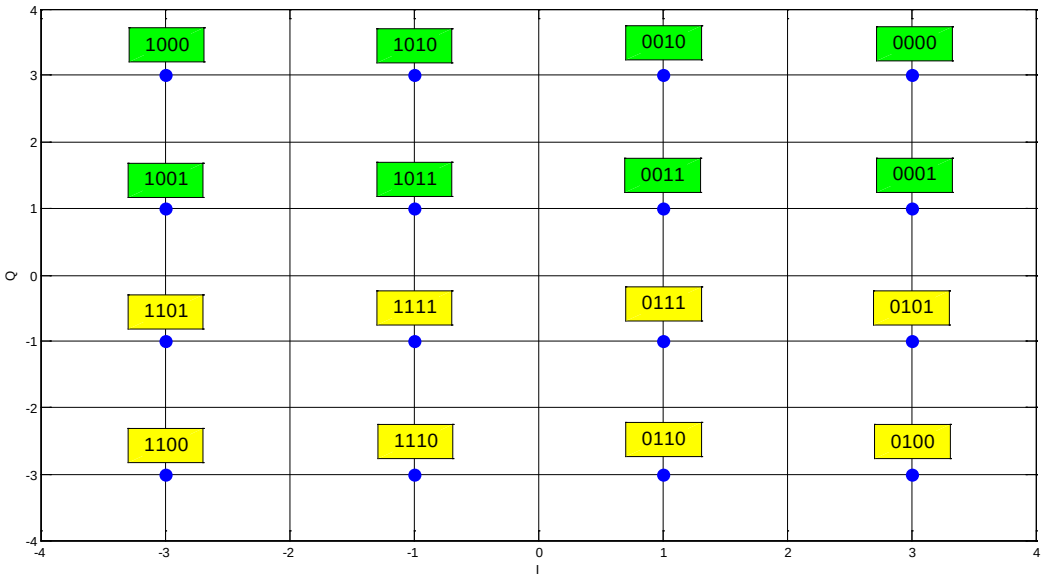

Figura 2.c.3 Costellazione 16-QAM, set di b1, verde C10 e giallo C11

Gli algoritmi per il calcolo dell'LLR di n bit con modulazione QAM che utilizzano tale proprietà della codifica Gray saranno indicati come LLR.

2.c.1 Algoritmo

Il nuovo algoritmo utilizza la proprietà vista nel paragrafo precedente che rende non più necessario il controllo di appartenenza ai set C_{i1} o C_{i0} per tutti gli x; è sufficiente un punto per ogni riga o colonna (dipende se il bit i-esimo è d'indice pari o dispari). Non è nemmeno più necessario memorizzare tutte le 2^n codeword dei punti della costellazione ma ne bastano m per rappresentare le righe e altrettante per le colonne.

Per evitare di utilizzare due vettori diversi per le righe e le colonne si possono prendere le codeword di una delle due diagonali principali del quadrato della costellazione e salvarle in un unico vettore D di lunghezza m; infatti a un cambio d'indice in D corrisponde un cambio sia di riga che di colonna.

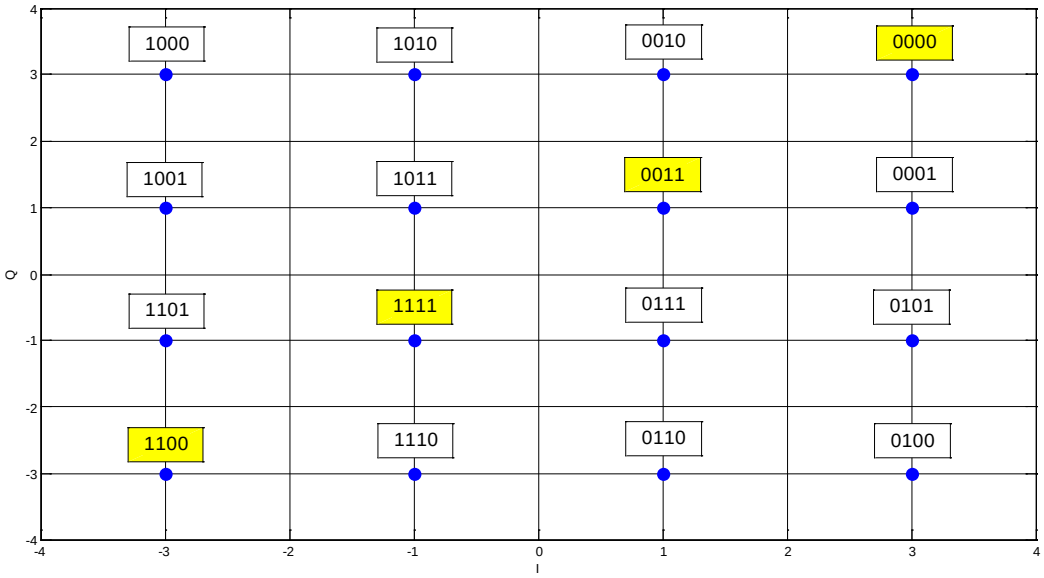

Figura 2.c.4 Costellazione 16-QAM diagonale principale evidenziata

Mantenendo le scelte degli indici dell'algoritmo LLR generico, si sceglie la diagonale che va dalla codeword salvata in $B[0][0]$ a quella in $B[m][m]$. Tale scelta comporta che il controllo sarà realizzabile con un unico indice j tale che:

$$D[j] = B[j][j] \quad \text{con } j=0,1,2...,m$$

La proprietà precedente ha naturalmente conseguenze, anche se non dirette, sul calcolo delle distanze dei punti della costellazione dall'ingresso $r(I,Q)$:

Proposizione: Per i bit d'indice pari le proiezioni sull'asse immaginario Q delle distanze sono ininfluenti sui valori finale degli LLR, analogamente per i bit d'indice dispari le proiezioni sull'asse reale I sono ininfluenti sui valori degli LLR.

Dimostrazione: Presi i punti x della costellazione, si stabilisce un ordine tra loro con l'ausilio degli indici l e j, già definiti nell'inizializzazione della matrice B.

Si ottiene una costellazione rappresentata da una matrice, i cui punti sono indicati con x_{ij}, tali che:

1. l indica la riga j e la colonna
2. x_{00} coincide col punto nell'angolo in basso a sinistra
3.

Siccome la codifica è di tipo Gray se il punto x_{ij} sta in C_{it}, con t booleano, allora:

$x_{lj} \epsilon C_{it}$ per ogni j se i pari

$x_{lj} \epsilon C_{it}$ per ogni l se i dispari

Scelto il caso i pari (per quello dispari cambiano solo gli indici). Posto:

$$T = \sum_{j=0}^{m-1} e^{-(Q-Q_x)^2}$$

si considera la probabilità composta p_1:

$$p1 = P[I, Q \mid bi = 1] = \sum_{x \epsilon C_{i1}} \frac{e^{-\frac{(I-I_x)^2+(Q-Q_x)^2}{2\sigma^2}}}{2^n \pi \sigma^2} =$$

Si separano le righe dalle colonne; per le colonne con punti in C_{i1} sono complete:

$$= \sum_{x \epsilon C_{i1} \wedge\ 0 \le l \le m-1} \sum_{0 \le j \le m-1} \frac{e^{-\frac{(I-I_x)^2+(Q-Q_x)^2}{2\sigma^2}}}{2^n \pi \sigma^2} =$$

fissato j le I_x sono costanti perché in una stessa colonna non cambia l'ascissa:

$$= \frac{\sum_{x \epsilon C_{i1} \wedge\ 0 \le l \le m-1} e^{-(I-I_x)^2} \sum_{0 \le j \le m-1} e^{-(Q-Q_x)^2}}{2^n \pi \sigma^2} =$$

la seconda sommatoria, i cui addendi provengono da tutti i punti in una stessa colonna, è uguale a T per definizione

$$= \frac{T \sum_{x \epsilon C_{i1} \wedge\ 0 \le l \le m-1} e^{-(I-I_x)^2}}{2^n \pi \sigma^2}$$

Analogamente per p_0 cambiano le colonne ma resta il termine T:

$$p0 = P[I, Q|bi = 0] = \frac{T \sum_{x \epsilon_{i0} \wedge\ 0 \le l \le m-1} e^{-(I-I_x)^2}}{2^n \pi \sigma^2}$$

Sostituendo le due espressioni nella (4) si ottiene:

$$LLR(b_i) = \log\left(\frac{\sum_{x \epsilon C_{i1}} e^{-\frac{(I-I_x)^2}{2\sigma^2}}}{\sum_{x \epsilon C_{i0}} e^{-\frac{(I-I_x)^2}{2\sigma^2}}}\right) \quad (6)$$

Con la (6) la proposizione è dimostrata. Si noti che si è potuto porre la seconda sommatoria uguale a L in entrambe le probabilità perché a differenza delle righe, tutti i punti della colonna stavano in C_{it}.

C.V.D.

La proposizione precedente non modifica l'algoritmo distanze() utilizzato con LLR generico, ma cambia il suo utilizzo all'interno del programma:

```
“
 //algoritmo che calcola LLR, m punti nel lato quadrato QAM

 vL = distanze(I,m)  //memorizzo su vL le proiezioni sull'asse reale

          for( i=0, i<m, i+2)  //calcolo LLR bit pari
             {   //calcolo p1, p0 e LLR(bi)
                for(j=0,j<m,j++)  //scorro lungo la diagonale
                    {
                       if(D[j][i]=1)  //condizione x sta in C1i
                           p1=p1+exp(-m*(vL(k))^2/σ^2)   //calcolo p1, formula (6)
                       else                                          //altrimenti x sta in C0i
                          p0=p0+exp(-(m*vL(k))^2/σ^2)
                     }
                LLR[i]=log(p1/p0);  //calcolo LLR bit i-esimo
              }

          //ripeto procedimento coi bit dispari usando Q
vL = distanze(Q,m) //memorizzo su vQ le proiezioni sull'asse immaginario

          for( i=1, i<m, i+2)  //calcolo LLR bit dispari
             {   //calcolo p1, p0 e LLR(bi)
                for(j=0,j<m,j++)  //scorro lungo la diagonale
                    {
                       if(D[j][i]=1)  //condizione x sta in C1i
                           p1=p1+exp(-m*(vL(k))^2/σ^2)   //calcolo p1, formula (6)
                       else                                          //altrimenti x sta in C0i
                          p0=p0+exp(-m*(vL(k))^2/σ^2)
                     }
                LLR[i]=log(p1/p0);  //calcolo LLR bit i-esimo
              } ”
```

Si può notare che i bit d’indice pari e dispari sono trattati separatamente in modo da evitare di utilizzare due vettori per le proiezioni delle distanze e le eventuali complicazioni nel calcolo degli

LLR su quale vettore adoperare.

Si è riuscito a completare un nuovo algoritmo che sfrutta l'informazione che la codifica sia di tipo Gray.

2.c.2 Implementazione in C e analisi dell'algoritmo

Nell'appendice B è proposta una possibile implementazione dell'algoritmo LLR in linguaggio C, si noti che l'inizializzazione della matrice D tramite il metodo codifica2( ) è momentaneamente tralasciata perché sarà chiarita in seguito:

Dall'analisi teorica dei tempi si deduce che il programma esegue tre operazioni critiche:

1) inizializzazione D: $O(m) = O(2^{\frac{n}{2}})$
2) calcolo distanze rispetto I e Q: $O(m) = O(2^{\frac{n}{2}})$
3) calcolo LLR: $O(nm) = O(n2^{\frac{n}{2}})$

Si ottiene che il programma è un $O(n2^{\frac{n}{2}})$ con una crescita esponenziale al variare di n numero di bit ma è ridotta di un fattore radice quadrata rispetto a quella dell'algoritmo LLR generico. Nel sistema DVB il massimo valore di n è 12, ciò rende decisivo il fattore $½$ all'esponente e sensibile il miglioramento delle prestazioni del programma rispetto a quello precedente.

2.c.3 Conclusioni

Si può affermare che il programma LLR non solo è realizzabile ma migliora notevolmente (almeno per i valori di n validi) il precedente LLR generico.

È molto improbabile, vista la complessità del problema, trovare un nuovo algoritmo che migliori a livello teorico le prestazioni temporali dei programmi LLR.

Si possono comunque apportare delle modifiche per perfezionarlo ulteriormente, in particolare è possibile sfruttare la conoscenza a priori della disposizione delle codeword nelle costellazioni dei QAM mapper e demapper dei sistemi DVB-C2.

Si può quindi pensare di scrivere un nuovo algoritmo che eviti di memorizzare le codeword.

2.d Algoritmo LLRp

L'algoritmo LLR studiato precedente è già una buona soluzione al problema del calcolo degli LLR degli n bit di una costellazione QAM con codifica di tipo Gray, ma è possibile migliorare tale algoritmo sfruttando che sono note quali codifiche Gray utilizza il sistema DVB-C2:

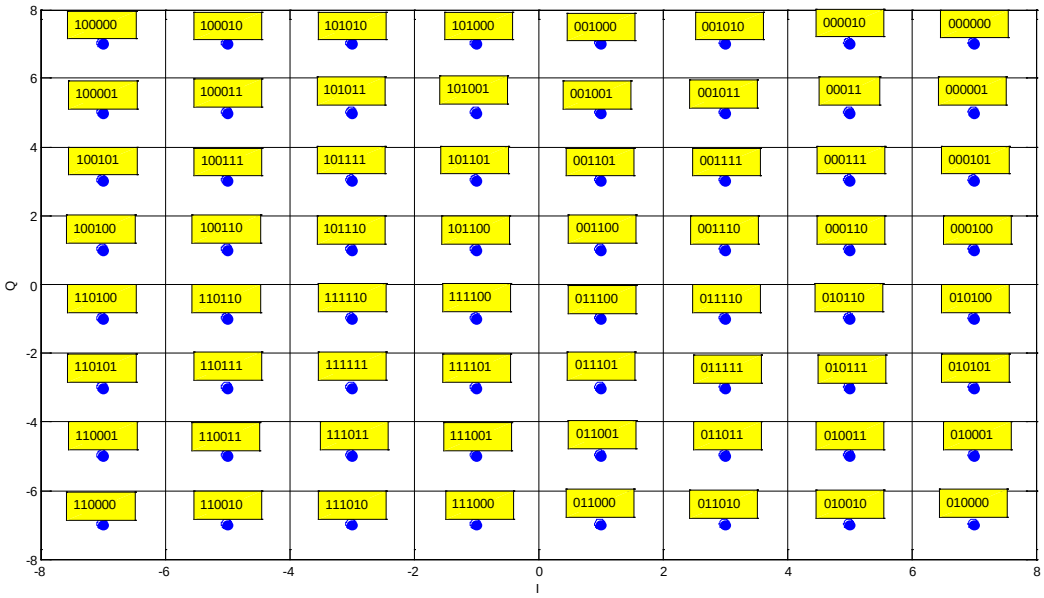

Figura 2.d.1 Costellazione 64-QAM con codifica Gray adottate dal DVB-C2

Oltre la proprietà cambio 1 data dalla definizione, la costellazione con codice Gray presenta importanti caratteristiche che riguardano i set C_{it}, con $t=1$ *o* 0:

Posto b_i come il generico bit degli n che compongono la codeword, per gli indici i pari tutti i punti nella stessa colonna della costellazione appartengono alla stessa C_{it}, simmetricamente per gli i dispari tutti i punti nella stessa riga appartengono allo stesso C_{it}.

I bit b_0 e b_1 (un QAM ha $n \geq 2$), ammettono come set C_{i1} e C_{i0} i due semipiani individuati dagli assi cartesiani, in particolare b_0 dall'asse Q e b_1 dall'asse I;

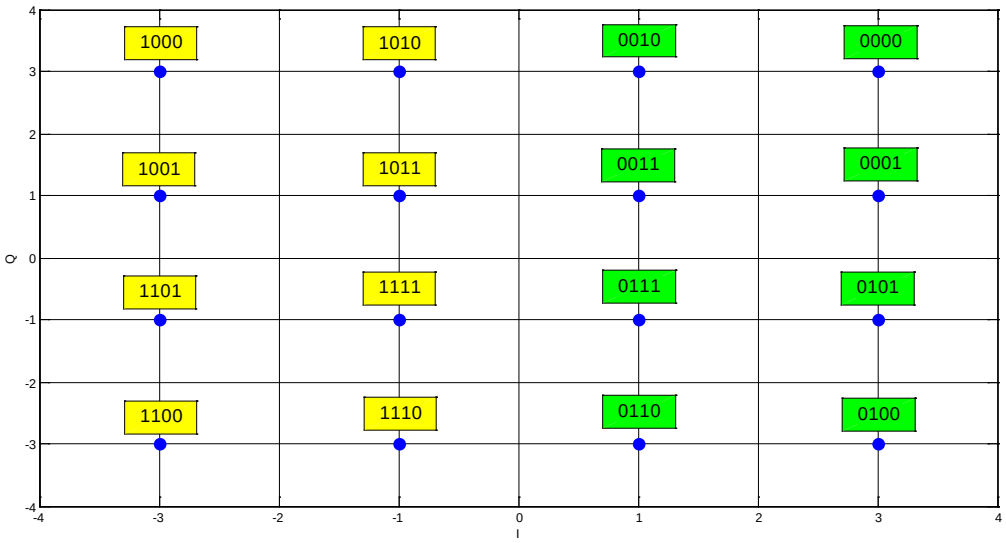

Figura 2.d.2 Costellazione 16-QAM, set di b_0, verde C_{00} e giallo C_{01}

i bit d'indice $i \geq 2$, se $n>2$, individuano come set C_{it} insiemi apparentemente diversi ma con strutture simili composte da miniblocchi:

Definizione: Data una costellazione M-QAM con codice Gray, si definisce miniblocco di b_i un insieme di colonne se i è pari, altrimenti di righe, adiacenti e appartenenti tutte allo stesso C_{it}.

Definizione: Si definisce miniblocco essenziale un miniblocco appartenente a C_{it} adiacente solo a righe o colonne (dipende se i pari o dispari) non appartenenti a C_{it}.

La costellazione QAM con codifica Gray è scomponibile in miniblocchi essenziali alternati tra loro:

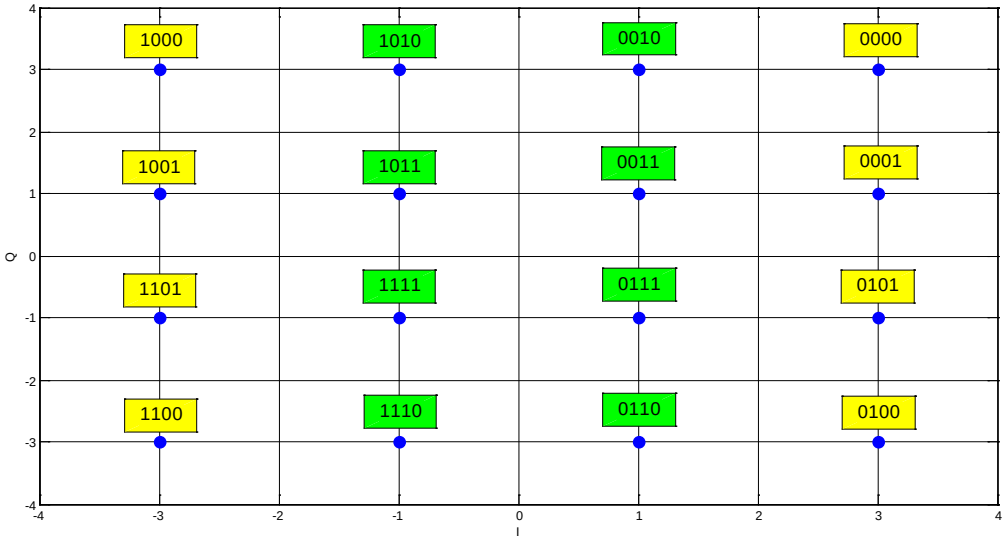

Figura 2.d.3 Costellazione 16-QAM set di b2 divisa in miniblocchi verdi e gialli

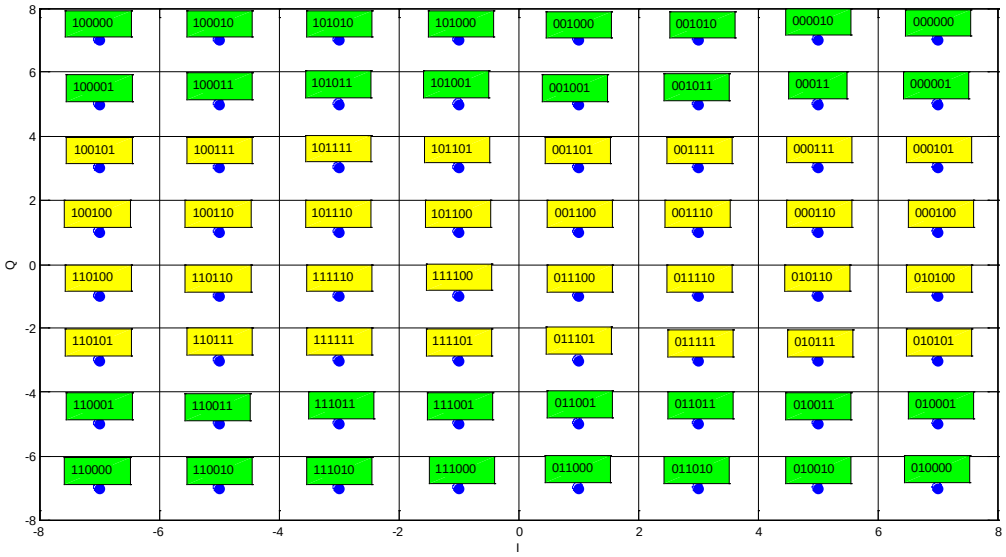

Figura 2.d.4 Costellazione 64-QAM set di b3 divisa in miniblocchi verdi e gialli

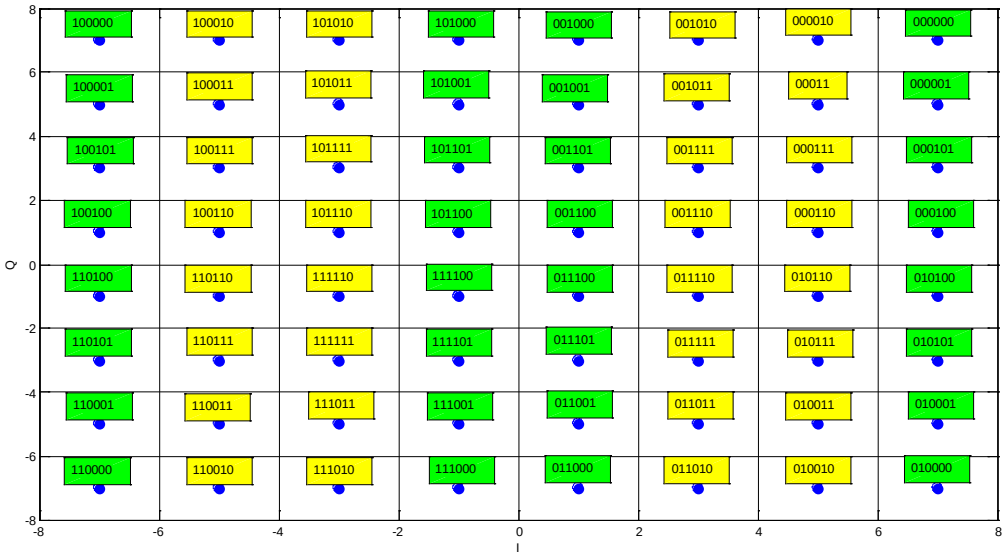

Figura 2.d.5 Costellazione 64-QAM set di b4 *divisa in miniblocchi verdi e gialli*

Si può notare che per ogni bit i miniblocchi essenziali sono tutti composti dallo stesso numero di righe (o colonne) k detta dimensione dei miniblocchi essenziali, ad eccezione del primo e dell'ultimo che sono di dimensione $k/2$.

Si stabilisce che questi due miniblocchi ne formano uno unico di dimensione k.

Si osserva inoltre che valori di k dipendono dallo specifico bit i-esimo:

Indice bit: i	Dimensione k
0,1	*m/2*
2,3	*m/2*
4,5	*m/4*
6,7	*m/8*
8,9	*m/16*
10,11	*m/32*

Tabella (1)

Ovviamente se $n<12$ i valori di k per $i>n-1$ non sono validi perché si riferiscono a bit che non esistono. I set dei due bit meno significativi sono composti da miniblocchi essenziali con $k=2$ per ogni valore di n. Si nota facilmente che escludendo i primi due bit, k parte da $m/2$ e continua a dimezzarsi allo scorrere a coppie dell'indice i.

La proprietà 1 è stata utilizzata nella realizzazione dell'algoritmo LLR, la 2 e la 3 possono invece sono servite per realizzare l'algoritmo LLRp (Log Likelihood Ratio perfective).

2.d.1 Algoritmo

L'obiettivo principale della realizzazione dell'algoritmo LLRp è di riuscire a eseguire il calcolo degli LLR senza memorizzare alcuna codeword e quindi senza eseguire alcun controllo diretto sulle codeword per stabilire a quale set C_{it} un punto della costellazione appartiene.

Per portare a termine tale obiettivo ci si avvale delle proprietà 2 e 3 viste nel paragrafo precedente; in particolare per i bit b_0 e b_1 si applicherà la 2 mentre per i restanti bit, se ce ne sono, si utilizzerà la 3. Grazie alla proprietà 1 è inoltre possibile eseguire un algoritmo per i bit pari delle codeword e poi riutilizzarlo per i bit dispari semplicemente invertendo le righe con le colonne.

Va inoltre ricordato che il calcolo delle distanze rimane identico a quello effettuato per l'algoritmo LLR; si utilizza quindi un unico vettore v_L di dimensione m da utilizzare prima per i bit pari e poi per quelli dispari.

Calcolo degli LLR dei due bit più significativi della codeword

Il primo bit b_0 della codifica, per qualsiasi valore di n, divide la costellazione in due semipiani che corrispondono ai set C_{01} e C_{00} (si veda la fig.2.c.2); in particolare i primi $m/2$ punti della costellazione stanno in C_{01} e gli altri in C_{00} ; calcolate le proiezioni sull'asse I delle distanze e salvate queste su v_L si può calcolare LLR di b_0:

```
“
//calcolo LLR di b0 sapendo che primi m/2 punti stanno in C01 e gli altri in C00
 p1=0;  //inizializzo p1 e p0
 p0=0;
 for(j=0, j<m/2, j++)
    {
       p1=p1+exp(-vD[j]^2/s^2);  //formula(6), primi m/2 punti in C01 e gli altri in C00
       p0=p0+exp(-vD[j+m/2]^2/s^2);
       LLR[0]=log(p1/p0);  //calcolo LLR di b0 tramite la formula(6)
    }  “
```

Per calcolare LLR di b_1 basta ricalcolare v_L partendo da Q e quindi si può ripetere il procedimento appena visto; oppure si possono utilizzare due vettori v_I e v_Q e calcolare i due LLR in un unico ciclo.

Calcolo degli LLR degli altri bit

Se $n>2$ vale la proprietà 3 della codifica Gray, pertanto esiste una successione k_i tale che per ogni bit b_i i set C_{it} sono composti da miniblocchi essenziali di dimensione k_i.

Data la dimensione k_i per ogni bit si può dedurre il numero di miniblocchi:

$$N°miniblocchi\ b_i = \frac{m}{k_i}$$

tale numero è sempre intero per costruzione di k_i.

Nelle costellazioni i miniblocchi sono composti da righe per i bit pari e colonne per quelli dispari e sono disposti in modo tale che un miniblocco essenziale di C_{i0} sia adiacente solo ad altri di C_{i1} e viceversa:

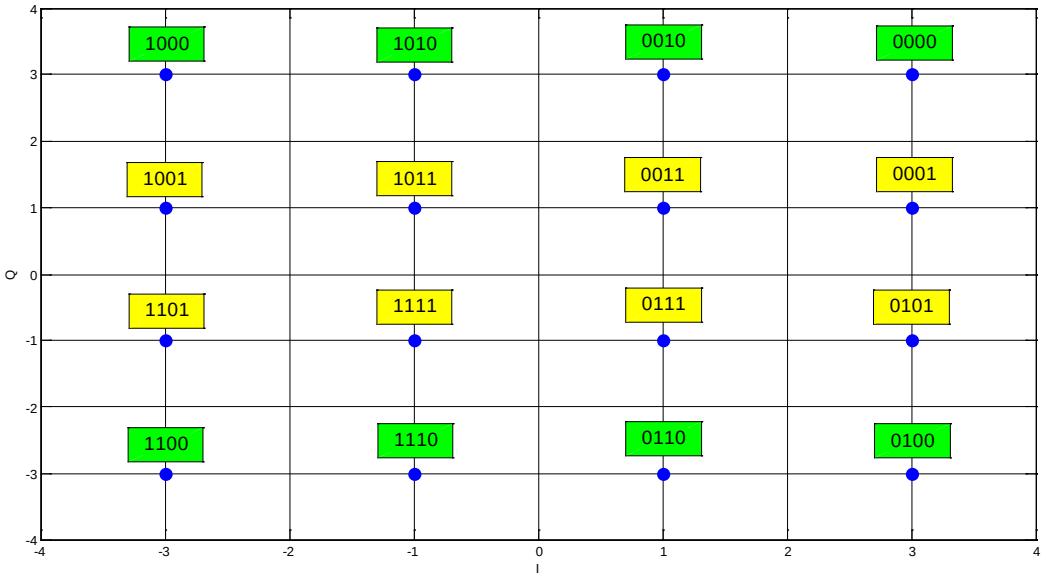

Figura 2.d.6 Costellazione divisa in miniblocchi, set C_{31} *verde* C_{30} *giallo*

Tralasciando momentaneamente che il primo miniblocco è composto da due parti separate, si può pensare che nota la k_i corrispondente al bit b_i si possono calcolare le relative p_1 e p_0:

Ordine dei miniblocchi di dimensione k_i.

```
“
//calcolo LLR del bit i-esimo con dimensione ki dei miniblocchi
  for(l=0, l<(m/ki), l+2)  //scorro lungo i miniblocchi a coppie
     {
          for(j=l, j<l+ki, j++) //scorro lungo le righe del miniblocco
             {
                 p0=p0+m*exp(-(vD(l)^2/s^2);  //primo miniblocco sta in Ci0
                 p1=p1+m*exp(-(vD(l+ki)^2/s^2);  //il miniblocco subito dopo sta in Ci1
              }
          LLR[i]=log(p1/p0);
     } “
```

in realtà i miniblocchi sono disposti col primo miniblocco diviso in due parti:

Si può risolvere il problema trattando il primo e l'ultimo miniblocco separatamente:

```
“
//come si corregge il ciclo for precedente
 for(l=3*ki/2, l<ki/2, l++)
            {
                p0=p0+m*exp(-(vD(m-l+ki)^2/s^2);  //seconda metà primo miniblocco
                p1=p1+m*exp(-(vD(m-l)^2/s^2);  //prima metà ultimo miniblocco
             }
 for(l=m-ki, l<m-ki/2, l++) //scorro lungo le righe del miniblocco
            {
                p0=p0+m*exp(-(vD(l-m+ki)^2/s^2);  //prima metà primo miniblocco
                p1=p1+m*exp(-(vD(l)^2/s^2);  //seconda metà ultimo miniblocco
             }
 for(l=0, l<(m/ki)-1, l+2)  //scorro lungo i miniblocchi a coppie, escludo una coppia
   {
       for(j=l+ki/2, j<l+ki, j++) //scorro lungo i miniblocchi intermedi, parto da ki/2
            {
                p0=p0+m*exp(-(vD(j)^2/s^2);  //primo miniblocco sta in Ci0
                p1=p1+m*exp(-(vD(j+ki)^2/s^2);  //il miniblocco subito dopo sta in Ci1
             }
       LLR[i]=log(p1/p0);
   } “
```

Attraverso i miniblocchi, l’algoritmo scorre in quest’ordine: nel primo ciclo for percorre i miniblocchi verdi, quelli viola nel secondo ciclo e i restanti miniblocchi intermedi nel terzo e ultimo ciclo for.

C0	C1	C0	C1	C0	C1	C1	C0

Risolto il problema di calcolare LLR del bit i-esimo conoscendo la rispettiva ki resta il compito di trovare: le dimensioni ki per eseguire iterativamente il calcolo degli LLR per tutti i bit.
Si ricorda che è sufficiente risolvere il problema solo per i bit a indice pari, per quelli dispari basterà ricalcolare i valori di v_L usando Q come ingresso e ripetere lo stesso algoritmo.

Dalla tabella (1) si deduce che:

1) $k_2=m/2$ ($k_2=k_3$)
2) $k_{i+1}=k_i/2$ per qualsiasi $1<i<n-2$ (altrimenti non esiste il bit (i+1)-esimo)
3) $k_{n-2}=k_{n-1}=2$ per qualsiasi valore di n

È quindi che per i bit i-esimi con $i>1$ si possono calcolare i relativi LLR: posto $k_i=k=m/2$ poi iterativamente si dimezza:

```
“
 k=m/2;  //dimensione k2

 for(i=2, i<n-1, i+2)   //solo bit pari escluso il primo
   {
     p1=0, p0=0;
      LLR(i,k)   //algoritmo precedente per il calcolo LLR bi con ki
      k=k/2  //per costruzione all'ultima iterazione k=1
   }  ”
```

Si è dimostrato che è possibile realizzare un algoritmo che calcola gli LLR di n bit di una costellazione QAM con codifica Gray senza memorizzare o controllare alcuna codeword.

2.d.2 Implementazione in linguaggio C e analisi dell'algoritmo

Nell'appendice C è presentata una possibile realizzazione dell'algoritmo LLRp in linguaggio C, si noti che il vantaggio di non dover realizzare alcuna codeword comporta una maggiore complessità del codice.

Realizzato il programma, si può passare all'analisi teorica dei tempi di esecuzione; il programma compie due operazioni critiche:

1) calcolo distanze rispetto I e Q: $O(m) = O(2^{\frac{n}{2}})$
2) calcolo LLR: $O(nm) = O(n2^{\frac{n}{2}})$

Si ottiene che il programma è un $O(n2^{\frac{n}{2}})$, con una crescita esponenziale al variare di n numero di bit come LLR. Il rapporto tra LLR e LLRp per quanto riguarda le prestazioni temporali è una costante K, cioè non dipende da n. Per stabilire quale tra i due programmi sia il più veloce occorre misurare direttamente i loro tempi di esecuzione.

2.d.3 Conclusioni

L'algoritmo LLRp è realizzabile, pertanto è possibile calcolare gli LLR di n bit di una costellazione QAM con codifica di tipo Gray senza memorizzare e confrontare alcuna codeword.

Purtroppo non è semplice stabilire se il nuovo programma sia più lento o veloce rispetto a LLR, poiché a livello teorico le prestazioni temporali restano invariate.

Per rispondere a tale quesito sarà quindi necessario eseguire un confronto più pratico: si testeranno entrambi i programmi su diverse quantità d'ingressi misurandone le prestazioni e si confronteranno i risultati.

CAPITOLO 3

3 ANALISI DELLE PRESTAZIONI E DEI RISULTATI

Nel capitolo precedente sono stati descritti i tre programmi: LLR generico, LLR e LLRp che implementano gli omonimi algoritmi per il calcolo degli LLR.

In questo capitolo saranno invece analizzati i risultati ottenuti dall'applicazione dei tre programmi sulle funzioni LLR, in particolare se esse sono consone alla realizzazione di un QAM demapper.

Prima di procedere nell'analisi è opportuno completare lo studio delle prestazioni temporali, affrontato solo a livello teorico nel capitolo precedente.

3.a Analisi prestazioni temporali algoritmi LLR

Nel capitolo precedente per ognuno dei tre programmi LLR generico, LLR e LLRp sono state studiate a livello teorico le prestazioni temporali e i risultati sono:

1) LLR generico $O(n2^n)$
2) LLR $O(n2^{\frac{n}{2}})$
3) LLRp $O(n2^{\frac{n}{2}})$

Per confermare le analisi teoriche e stabilire quale tra LLR e LLRp abbia migliori prestazioni, si scriveranno dei programmi TimeTester allo scopo di misurare i tempi di esecuzione per tutti i valori di n e per diverse quantità d'ingressi.

3.a.1 Programmi TimeTester

Per ottenere una misura accettabile dei tempi di esecuzione dei programmi non è sufficiente misurare il tempo di ogni singola esecuzione, perché i tempi così ottenuti sono molti bassi e quindi facilmente influenzabili da possibili rallentamenti casuali a cui è sottoposta la macchina che esegue il programma. È quindi più prolifico ripetere numerose volte l'esecuzione di uno stesso ingresso e misurarne il tempo totale, in questo modo i rallentamenti che compariranno su alcune esecuzioni avranno effetti meno significativi.

Inoltre la scelta di misurare il tempo complessivo di più esecuzioni si avvicina di più al caso reale poiché il demapper è utilizzato per enormi quantità d'ingressi, cioè per la trasmissione di un flusso

continuo di bit.

Ciò significa che è assai più interessante studiare le prestazioni rispetto a una gran quantità d'ingressi, flusso di bit, rispetto a una singola esecuzione da *n* bit.

Ecco una possibile struttura dei programmi TimeTester da applicare ai tre algoritmi:

```
“
 Vin[2] , s //ingresso, varianza
  L //numero esecuzioni
 n,m //numero bit codifica, numero punti lato costellazione

 time_start;  //parte il cronometro

 for(k=0;k<L,k++)
   LLRx(n,s,Vin); // programma che calcola LLR con Vin, s ingressi

 time_stop;  //ferma il cronometro
“
```

Con LLRx s'intende uno tra LLR generico, LLR e LLRp secondo quale programma TimeTester si sta realizzando.

3.a.2 Memorizzazione del codice Gray

Nei programmi LLR e LLR generico è necessario memorizzare alcune codeword, se non tutte, del codice.

La memorizzazione non deve necessariamente essere ripetuta a ogni esecuzione dei programmi LLR e LLR generico, anzi è preferibile compiere tale operazione un'unica volta all'inizio dei programmi TimeTester.

E' necessario definire la modalità della memorizzazione. Scrivere direttamente le codeword è un impegno dispendioso per valori di *n* grandi ed è preferibile trovare un algoritmo.

Una possibile soluzione consiste nell'applicare parte dell'algoritmo LLRp sfruttando il doppio ciclo for che percorre la struttura a miniblocchi e sostituendo le operazioni per il calcolo del LLR con il salvataggio dei bit delle codeword. Nell'appendice E è riportato un possibile programma in linguaggio C che implementa il metodo codifica() sfruttando la struttura a miniblocchi.

Il metodo codifica2() invece memorizza la diagonale principale D, ciò si traduce in termini di codice nel semplificare il doppio ciclo for in un unico ciclo che memorizza direttamente 1 o 0 su $D[j][i]$.

Con quest'algoritmo è possibile realizzare LLRTimeTester e LLRgenTimeTester; ovviamente il

contributo sul tempo totale del salvataggio del codice è trascurabile perché è eseguito una sola volta mentre il resto del programma sarà eseguito L volte, con $L>>1$.

3.a.3 Implementazione in C

Nell'appendice D è presentata una possibile realizzazione del programma LLRpTimeTester.

Gli altri due programmi sono simili tranne che per la memorizzazione della codifica e per il conseguente controllo delle codeword nel calcolo degli LLR.

3.a.4 Risultati

Con i tre porgrammi TimeTester è possibile calcolare i tempi di esecuzione scegliendo un opportuno L e confrontare i risultati. In quest'analisi si è posto: $L=10^7$

Si può quindi eseguire un confronto tra i programmi LLR, LLR generico e LLRp; si sono ripetute le misurazioni per tutti i valori di n con un ingresso r casuale affetto da rumore gaussiano. Si fa presente che il tempo di esecuzione non dipende dalla posizione del punto r nella costellazione.

NUMERO BIT n	LLR GENERICO	LLR	LLRp
2	8 s	7 s	7 s
4	29 s	16 s	15 s
6	112 s	28 s	26 s
8	454 s	61 s	55 s
10	2447 s	115 s	102 s
12	-	407 s	374 s

Tabella 2

Non si è riportato il tempo di esecuzione del programma LLRgenTimeTester nel caso n=12 con $L=10^7$ perché troppo elevato da misurare.

Come previsto l'algoritmo LLR generico è più lento degli altri due con una differenza che aumenta notevolmente all'aumentare di n.

Nel confronto tra LLRTimeTester e LLRpTimeTester risulta che quest'ultimo è più veloce, con un vantaggio che aumenta all'aumentare di n.

In conclusione si può affermare che tra i tre programmi realizzati, LLRp non solo permette di evitare l'operazione di memorizzazione del codice, ma garantisce anche le prestazioni migliori in termini di tempi di esecuzione.

3.b Impiego degli LLR nei demapper

Risolto il problema del calcolo degli LLR degli n bit di una costellazione QAM non rimane che utilizzare questi valori per la decodifica; si ricorda che la definizione di LLR di un bit i-esimo di una costellazione di tipo QAM dato l'ingresso $r(I,Q)$ è data dalla (1).

Il segno dell'LLR indica il valore più probabile del bit i-esimo del messaggio originale in trasmissione:

$$\boldsymbol{LLR} > 0 \rightarrow \boldsymbol{p_1} > \boldsymbol{p_0}$$

$$\boldsymbol{LLR} < 0 \rightarrow \boldsymbol{p_0} > \boldsymbol{p_1}$$

$$\boldsymbol{LLR} = \boldsymbol{0} \rightarrow \boldsymbol{p_1} = \boldsymbol{p_0}$$

Ottenuti gli n valori degli LLR si possono ipotizzare i valori dei bit della codeword decodificata in base ai segni degli LLR: se positivo $b_i=1$ e se negativo $b_i=0$.

Se il segno determina quale valore più probabilmente assume b_i, il modulo del LLR di quanto è più probabile un valore rispetto all'altro.

In particolare maggiore è |LLR| più è certo il valore di b_i qualunque esso sia, mentre minore è |LLR| più è incerto tale valore. Se LLR è nullo abbiamo una situazione di totale incertezza.

È evidente che se si vuole utilizzare gli LLR per decodificare un segnale in ricezione il risultato sarà più affidabile per valori grandi in modulo degli LLR.

Siccome il demapper utilizza gli LLR come una soft-decision questi non stabiliscono direttamente il valore dei b_i, ma sono integrati alle informazioni derivate dall'interleaved coding per stabilire in seguito i valori dei bit della codeword ricostruita. Pertanto valori prossimi a 0 degli LLR non implicano necessariamente l'impossibilità di prendere una decisione affidabile ma comunque non restituiscono informazioni utili per la decodifica.

3.b.1 Grafici degli LLR

Per calcolare quali valori può assumere LLR(b_i) e quali di questi sono affidabili è necessario studiare la funzione LLR.

Dalla (4) si deduce che la funzione LLR(b_i) dipende da $r(I,Q)$ e $\sigma^{2.}$

Fissato $\sigma = 1$ è possibile utilizzare i programmi del capitolo precedente per ottenere il grafico della funzione LLR al variare di I o Q, se si usano i programmi LLR o LLRp.

Si è dimostrato che con la codifica di tipo Gray gli LLR dei bit d'indice dispari dipendono esclusivamente da Q e quelli pari dipendono solo da I.

Pertanto è sufficiente studiare solo gli LLR dei bit a indice pari poiché quelli d'indice dispari hanno grafici identici se si sostituisce I con Q.

Utilizzando il programma LLRp e matlab si sono tracciati i seguenti grafici che rappresentano alcune funzioni LLR dipendenti da I, con $\sigma = 1$.

Sono proposti i grafici del LLR(b_0) con b_0 bit più significativo. Siccome C_{01} e C_{00} sono simmetriche rispetto l'asse Q è possibile supporre che LLR(b_0) sia una funzione dispari:

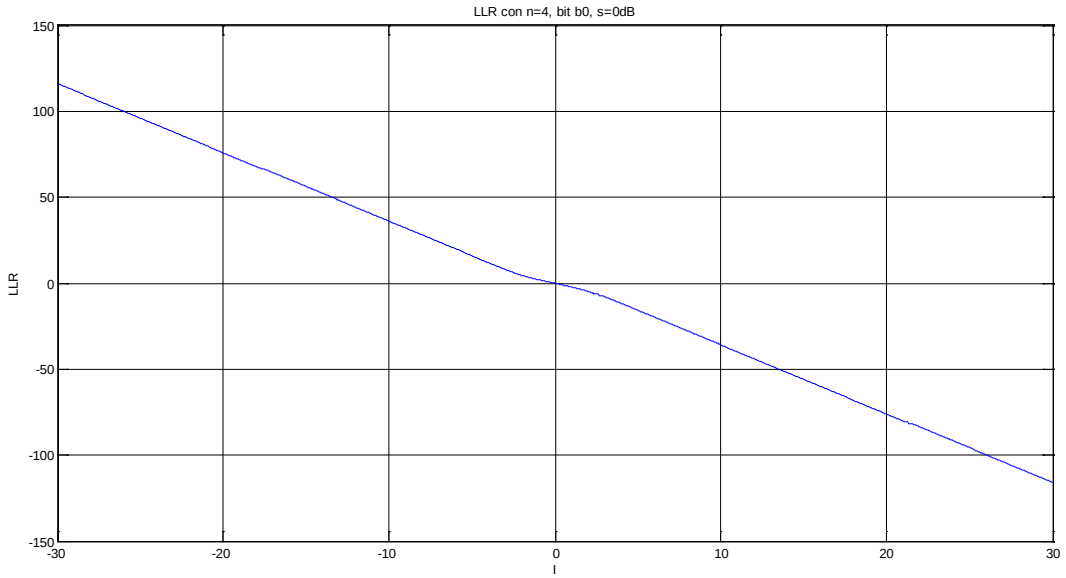

Figura 3.b.1 LLR(b_0,I) con $-30 \leq I \leq 30$ e n=4

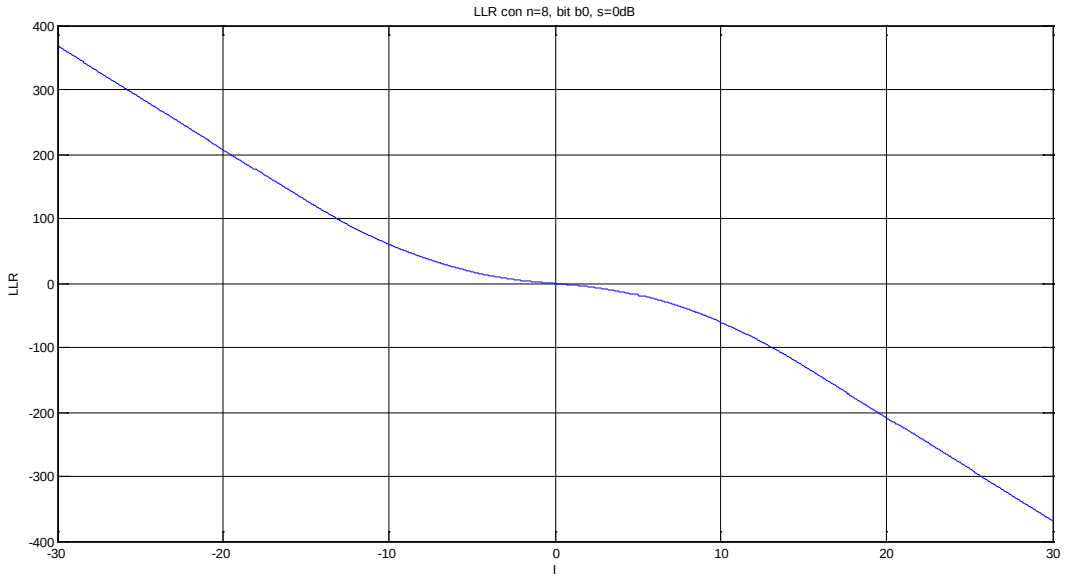

Figura 3.b.2 LLR(b_0,I) con-$30 \leq I \leq 30$ e n=8

Si noti che LLR(b_0) come previsto è dispari, passa per l'origine nel punto *0* ed è illimitata.

Per quanto riguarda gli altri bit invece i set C_1 e C_0 sono asimmetrici, ma con la struttura a miniblocchi, seguono alcuni esempi:

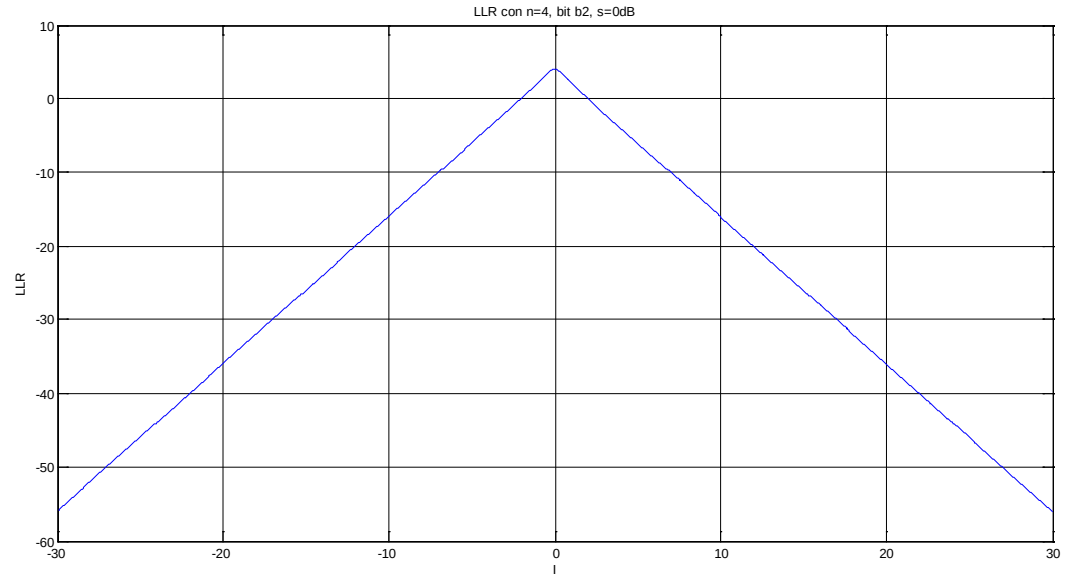

Figura 3.b.3 LLR(b_2,I) con -$30 \leq I \leq 30$ e n=4

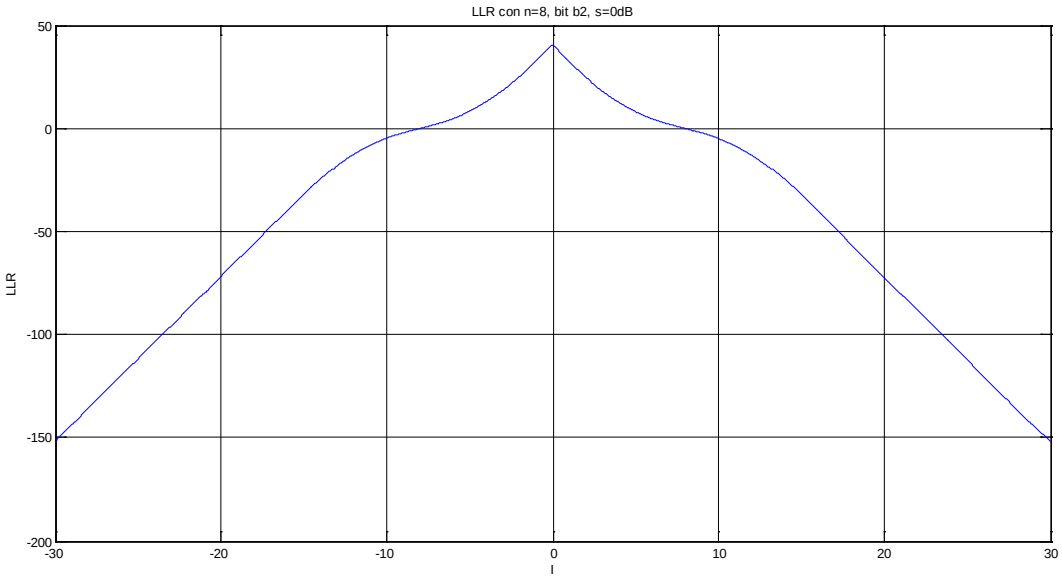

Figura 3.b.4 LLR(b_2,I) con $-30 \leq I \leq 30$ e n=8

I grafici di LLR(b_2) mostrano che la funzione è limitata superiormente con massimo in 0, ma non è limitata inferiormente.

Il grafico è coerente con la spartizione dei set C_{21} e C_{20} di b_2, il primo miniblocco, che compone C_{20}, è diviso in due parti all'esterno della costellazione; tale disposizione dei miniblocchi spiega perché LLR(b_2) è limitata solo superiormente: il picco in cui LLR è positivo coincide con l'unico miniblocco intermedio di C_{21}.

Si può supporre che all'aumentare di i (indice dei bit) si avranno grafici simili: gli LLR saranno limitati superiormente ma con un maggiore numero di picchi positivi e negativi corrispondenti ai miniblocchi intermedi:

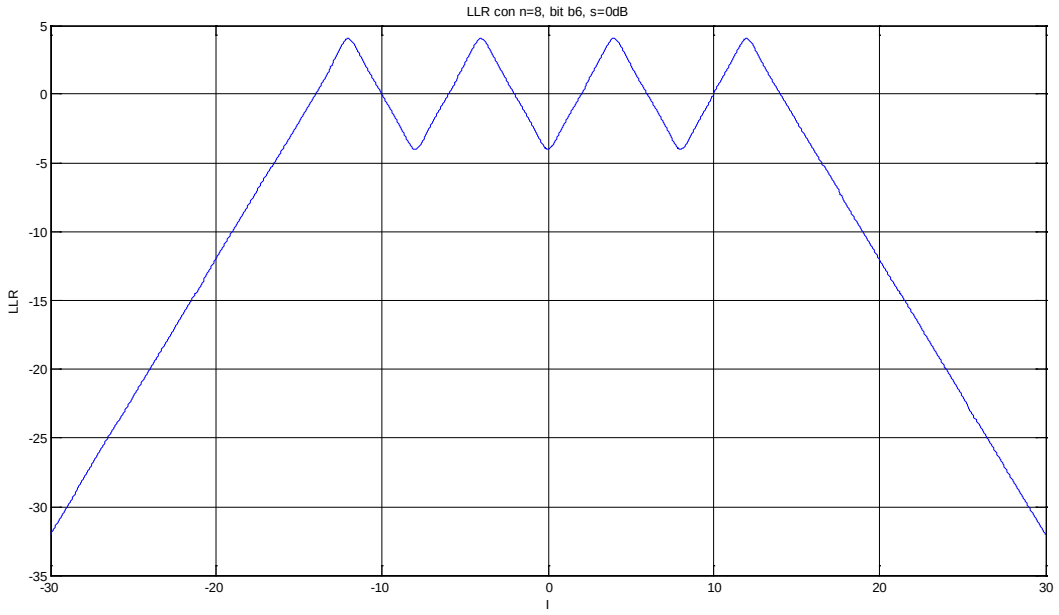

Figura 3.b.5 LLR(b_6,I) con $-30 \leq I \leq 30$ e n=8

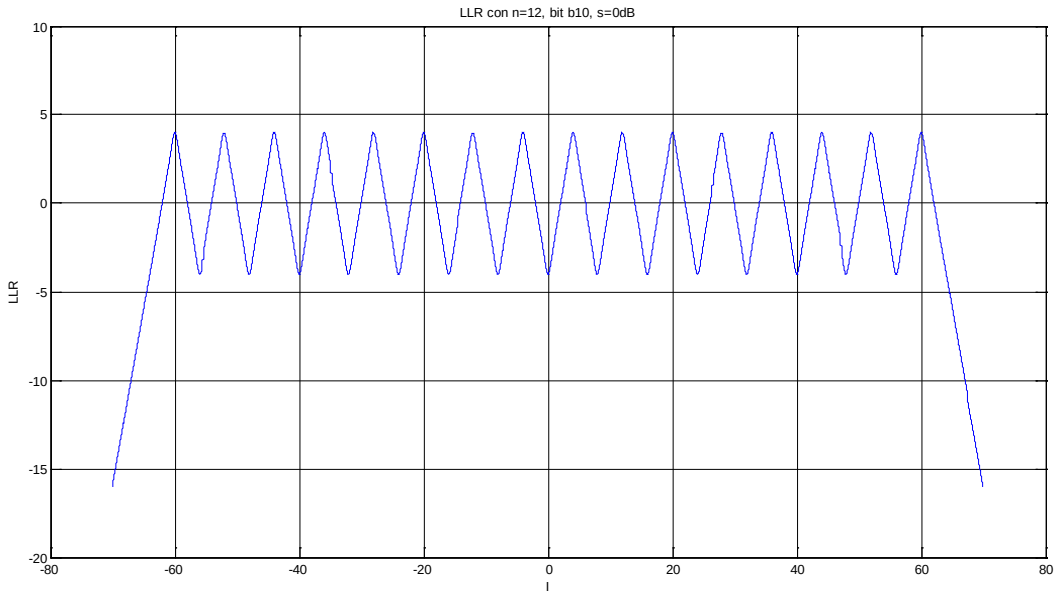

Figura 3.b.6 LLR(b_{10},I) con $-70 \leq I \leq 70$ e n=12

Le funzioni LLR sono riconducibili a due categorie:

LLR dei due bit più significativi, che sono dispari e illimitati.

LLR degli altri bit che sono limitati superiormente e presentano un andamento oscillante nell'intervallo di I o Q che contiene i punti della costellazione $[-m+1,m-1]$.

Queste informazioni seppure utili non bastano a determinare l'affidabilità di un decoder che utilizza gli LLR; è necessario studiare anche i valori $r(I,Q)$ ricevuti.

L'ingresso r dipende anche da una variabile aleatoria gaussiana: il rumore, in particolare quello del canale di trasmissione, che già influisce direttamente su LLR con la sua varianza.

La varianza σ^2 determina in media quanto il rumore su $r(I,Q)$ si discosta dalla sua media, che si suppone nulla; pertanto indica anche quanto $r(I,Q)$ si discosta in media dal punto $tx_m(I_m,Q_m)$ della costellazione scelto in trasmissione.

Siccome i valori delle funzioni LLR dipendono da I e Q la varianza influisce anche su quali valori possono assumere gli LLR e con una certa probabilità.

Sia $r(I,Q)$ che gli LLR possono essere considerati variabili aleatorie composte del rumore.

3.b.2 PDF e CDF degli LLR

Per stabilire per quali di LLR(b_i) si può compiere una scelta di decodifica affidabile occorre stabilire una soglia di affidabilità M_i, sotto la quale non è proponibile utilizzare LLR(b_i) nella decodifica.

Consultando le Figure $3.b.1,2,3,4,5,6$ si osserva che per ogni bit e per ogni n esiste almeno un intorno di $I: [a,b]$ con $a, b \epsilon \mathbb{R}$ la cui immagine è un intorno di zero. Quest'osservazione, giustificata anche dal fatto che le funzioni LLR sono continue e assumono valori positivi e negativi, indica che tali funzioni possono sempre assumere valori prossimi a zero se non nulli. È pertanto impossibile fissare una soglia M_i di affidabilità.

Tuttavia nell'affermazione precedente si è tralasciato che LLR(b_i) è una variabile aleatoria, in particolare non si è tenuto conto della probabilità che LLR(b_i) assuma un certo valore.

Le funzioni LLR possono essere considerate affidabili per la realizzazione di un decoder se sono sufficientemente basse la probabilità di ricevere ingressi $r(I,Q)$ i cui LLR siano vicini a zero.

La funzione che determina con quale probabilità una variabile aleatoria $a(x)$ assume valori inferiori a $M>0$ è la sua distribuzione, nota anche come CDF (Comulative Distribution Function). La funzione densità o PDF (Probability Density Function), definita anche come la derivata della CDF, indica in un intervallo $[a,b]$, $a, b \epsilon \mathbb{R}$ il rapporto tra la probabilità che $a(x)$ assuma un valore contenuto nell'intervallo e la lunghezza di quest'ultimo $(b-a)$.

3.b.3 Programmi per trovare le PDF e le CDF

Per realizzare i grafici delle CDF e PDF degli LLR si utilizza, come per quelli precedenti, il programma LLRp.

Il programma LLRpTesterPDF esegue il calcolo degli LLR su F ingressi $r(I,Q)$ diversi; per ottenere questi ingressi si è deciso di scomporli in due parti: la prima col metodo random() per realizzare le coordinate (I_m,Q_m) delle codeword scelte in trasmissione, in teoria equiprobabili; per la parte del rumore gaussiano invece si sono realizzati tre vettori in matlab col metodo randn() con media nulla e due diversi valori della varianza: σ^2=*0 dB, -10 dB, -20 dB.*

Nell'appendice F è riportato il programma LLRpTesterPDF in C utilizzato per calcolare gli LLR di $F=10^5$ $r(I,Q)$ diversi; gli ingressi del programma sono il numero n di bit e la varianza s, ottenuti da prompt, il vettore contenente $2x10^5$ rumori gaussiani come il file di testo rumor.txt.

Per realizzare la CDF in matlab su un campione discreto (F valori di LLR) si utilizza tale formula:

$$P_{LLR}[x]=P[LLR<x]=N_x/N_{tot} \quad \text{con } N_x=n°\ campioni:\ LLR<x,\ N_{tot}=totale\ campioni=F=10^5$$

Per la realizzazione della PDF che in teoria non ha significato su un dominio discreto come gli L campioni di LLR si è deciso di realizzare una sua approssimazione tramite il metodo hist() di matlab; la formula utilizzata è:

$$p_{LLR}[x]=(P_{LLR}[a]-P_{LLR}[b])/(b-a)=(Na-Nb)/(b-a)Ntot \quad \text{per ogni } x \in a,b$$

Per semplificare si scelgono gli intervalli $[a,b]$ tali che: $b-a=1$ e si ricorda che i grafici dei bit con indice dispari sono uguali o almeno molto simili a quelli a indice pari.

3.b.4 Grafici con varianza unitaria

In questo paragrafo sono riportati alcuni grafici ottenuti utilizzando il programma LLRpTesterPDF e Matlab.

Si pone σ^2= 0dB per poter confrontare i grafici delle CDF e PDF con quelli delle figure 3.b.1,2,3,4,5,6.

Si può iniziare da $LLR(b_0)$ (uguale a $LLR(b_1)$) poiché è la funzione con la divisione in set C_{it} più particolare:

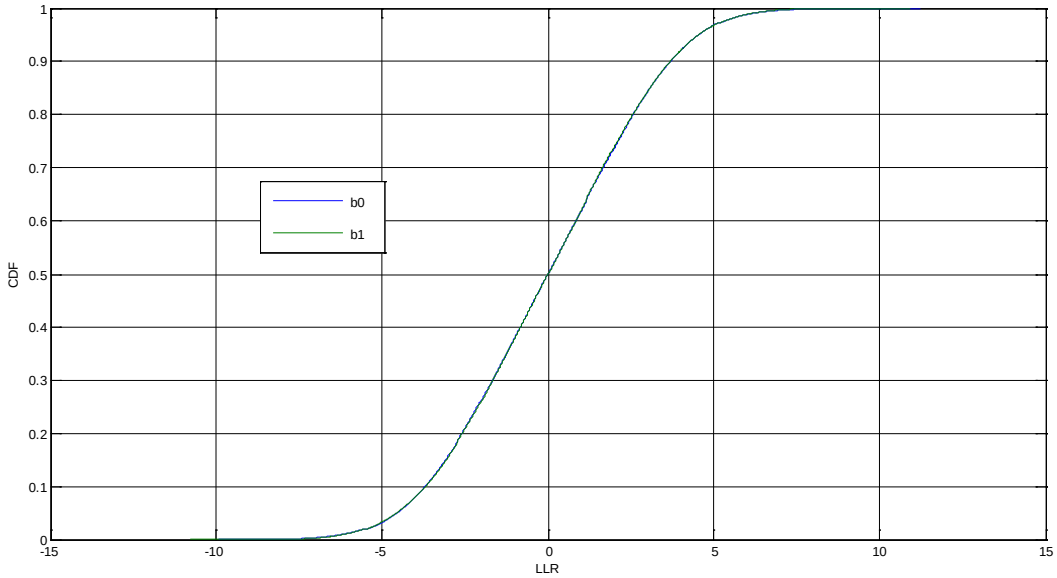

Figura 3.b.7 CDF degli LLR(b_0) e LLR(b_1) con n=2 e s=0dB

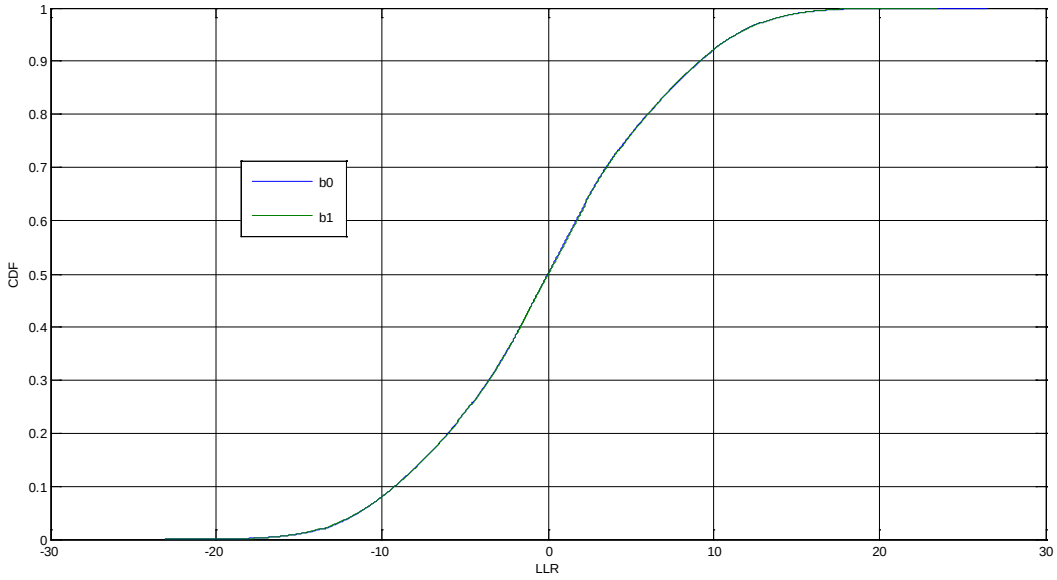

Figura 3.b.8 CDF degli LLR(b_0) e LLR(b_1) con n=4 e s=0dB

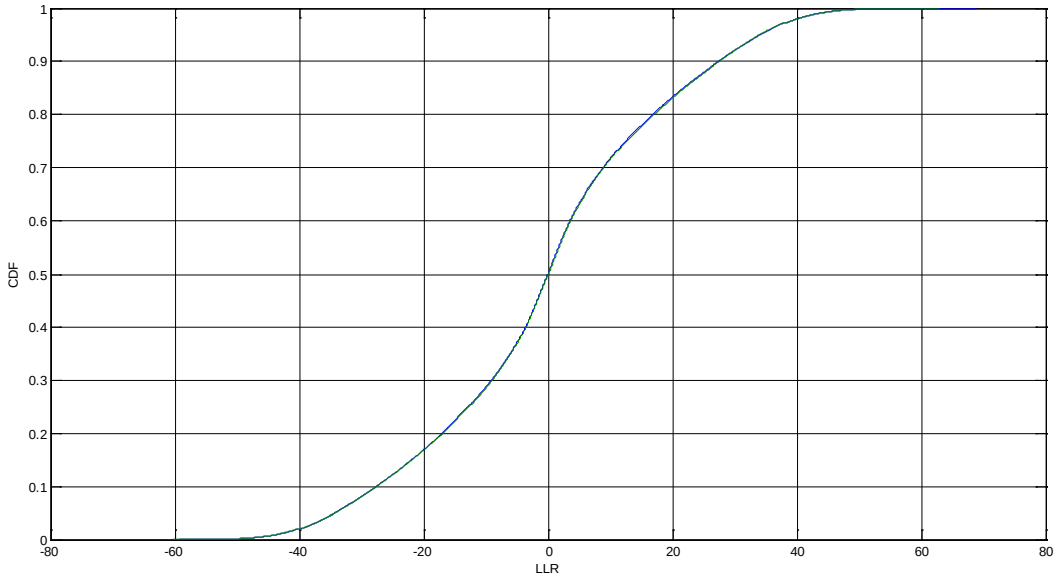

Figura 3.b.9 CDF degli LLR(b_0) e LLR(b_1) con n=6 e s=0dB

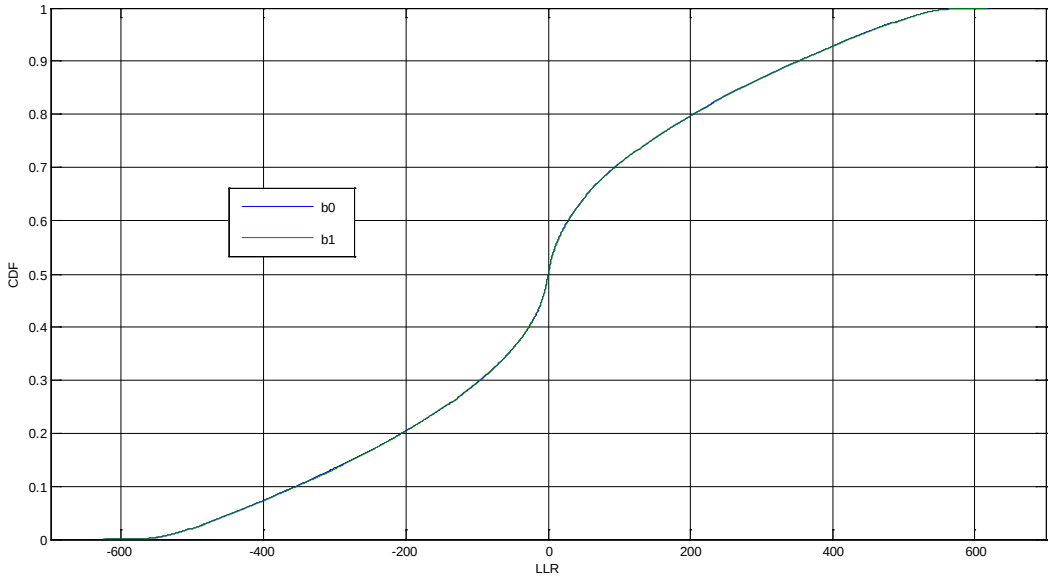

Figura 3.b.10 CDF degli LLR(b_0) e LLR(b_1) con n=10 e s=0dB

I grafici delle CDF di b_0 sono compatibili con quelli delle LLR(b_0):

1. $CDF_{LLRb0}(0)=0.5$, cioè in zero c'è la massima incertezza
2. le CDF_{LLRb0} sono simmetriche rispetto a $(0,0.5)$, coerente con la simmetria di LLR(b_0)

Ecco invece le rispettive PDF:

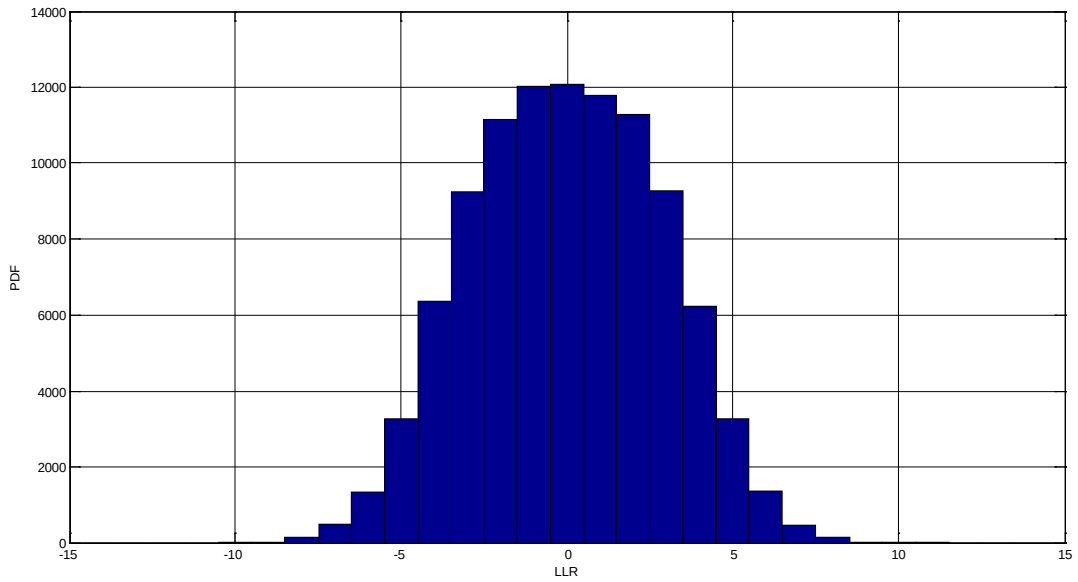

Figura 3.b.11 PDFx100000 del LLR(b_0) con n=2 e s=0dB

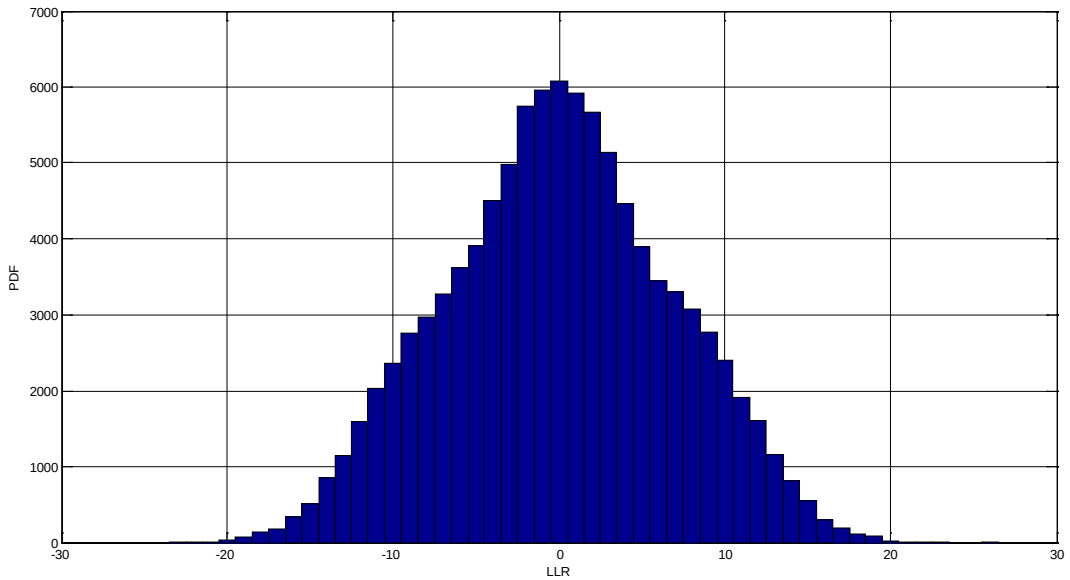

Figura 3.b.12 PDFx100000 del LLR(b_0) con n=4 e s=0dB

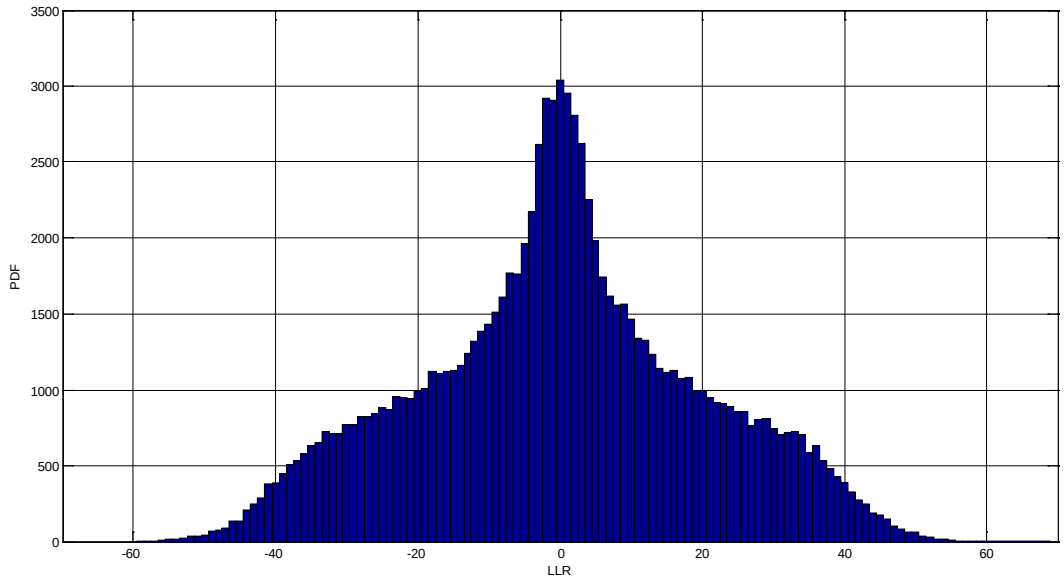

Figura 3.b.13 PDFx100000 del LLR(b_0) con n=6 e s=0dB

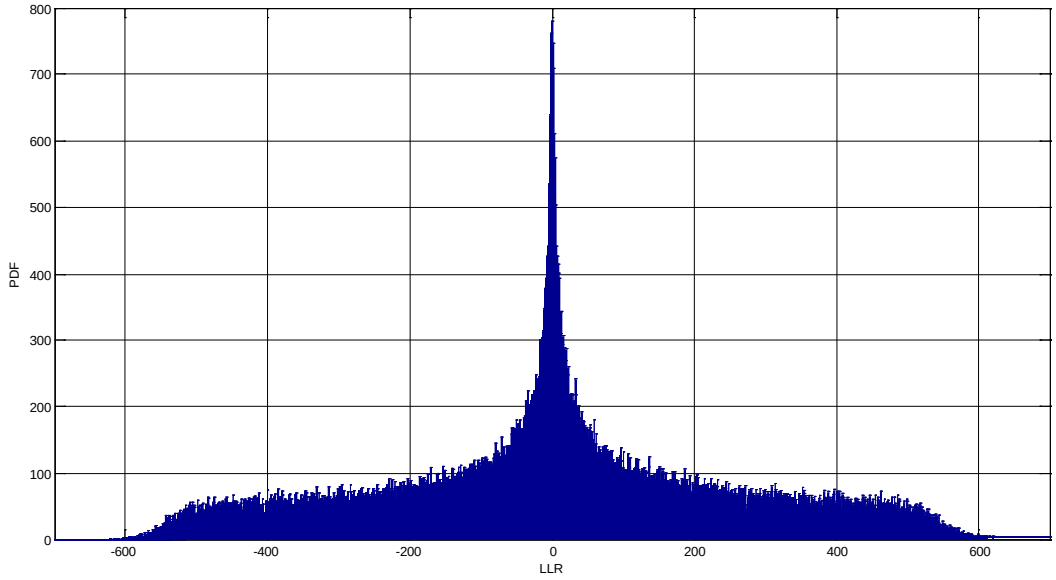

Figura 3.b.14 PDFx100000 del LLR(b_0) con n=10 e s=0dB

Le PDF mostrano chiaramente che la maggior parte dei campioni LLR sono nulli o assumono valori vicino a zero e tendono a diminuire di numero all'aumentare dei valori dell'LLR. Questa caratteristica si accentua all'aumentare di n.

Con una tale varianza le funzioni LLR(b_0) e LLR(b_1) sono poco affidabili per decodificare.

Si osservano ora alcune PDF di LLR(b_2), si ricorda che i set di b_2 e b_3 sono composti da solo due miniblocchi di dimensione $k=m/2$:

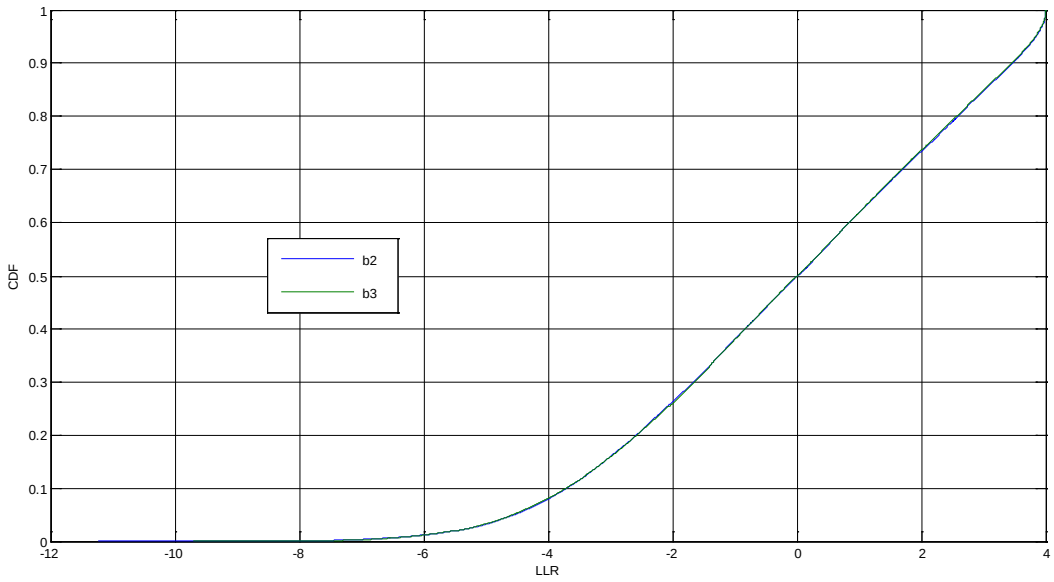

Figura 3.b.15 CDF degli LLR(b_2) e LLR(b_3) con n=4 e s=0dB

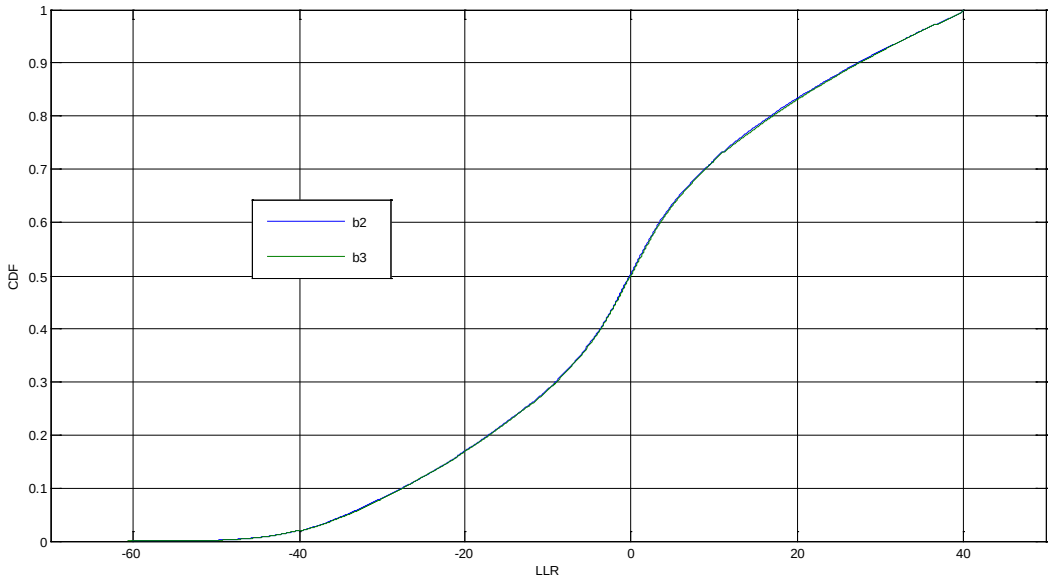

Figura 3.b.16 CDF degli LLR(b_2) e LLR(b_3) con n=8 e s=0dB

Anche in questo caso le CDF di b_2 confermano i rispettivi grafici LLR(b_2) del paragrafo 3.b.1; infatti la CDF raggiunge il valore 1 in un punto esatto, che corrisponde al massimo della LLR.

Per quanto riguarda le PDF:

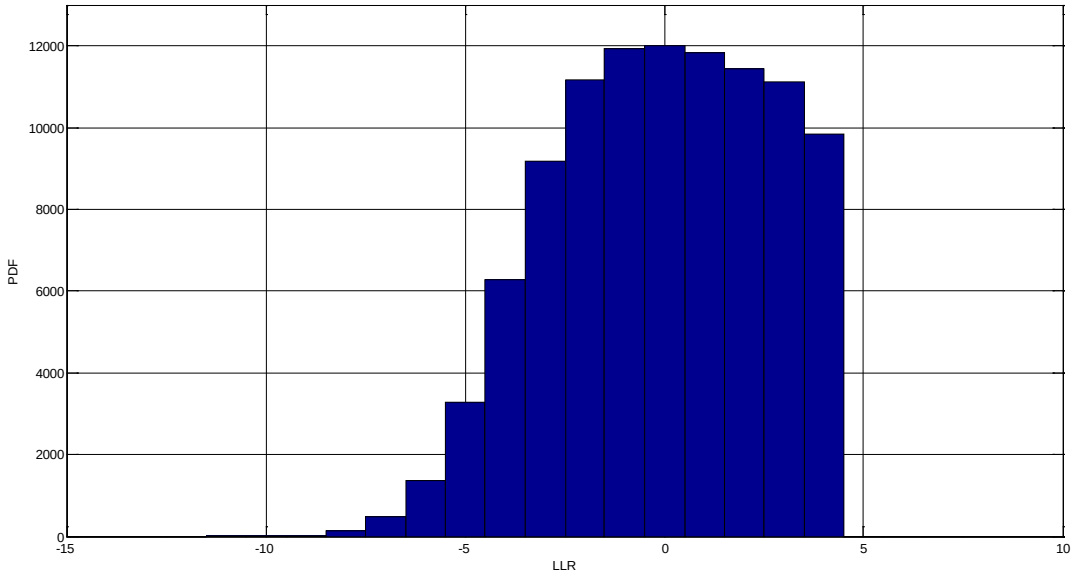

Figura 3.b.17 PDFx100000 del LLR(b_2) con n=4 e s=0dB

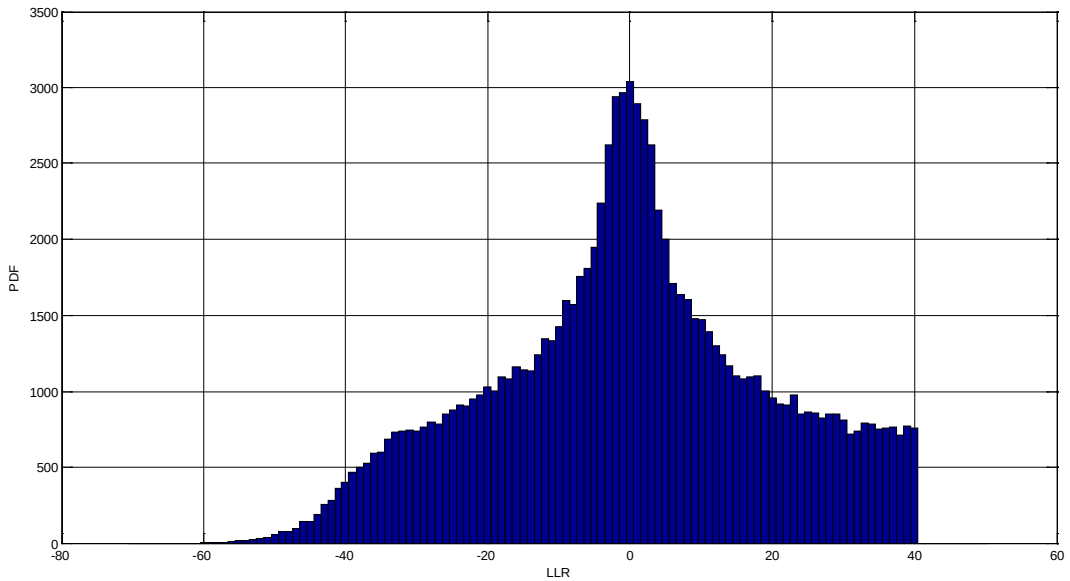

Figura 3.b.18 PDFx100000 del LLR(b_2) con n=8 e s=0dB

Anche in questo caso la maggior parte dei campioni si trova nei valori di LLR vicino a zero.

Si nota inoltre che mentre il numero di campioni che assume valori negativi decresce lentamente al tendere del LLR a -∞, mentre quelli positivi subiscono un brusco arresto al valore massimo.

Per quanto riguarda gli altri bit i grafici sono simili a quelli di b_2:

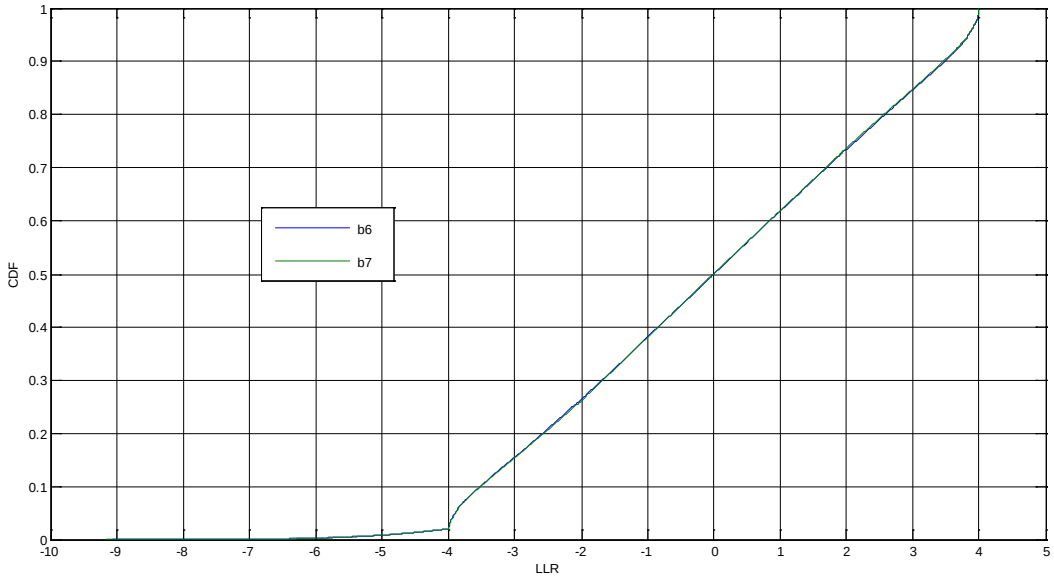

Figura 3.b.19 CDF degli LLR(b_6) e LLR(b_7) con n=8 e s=0dB

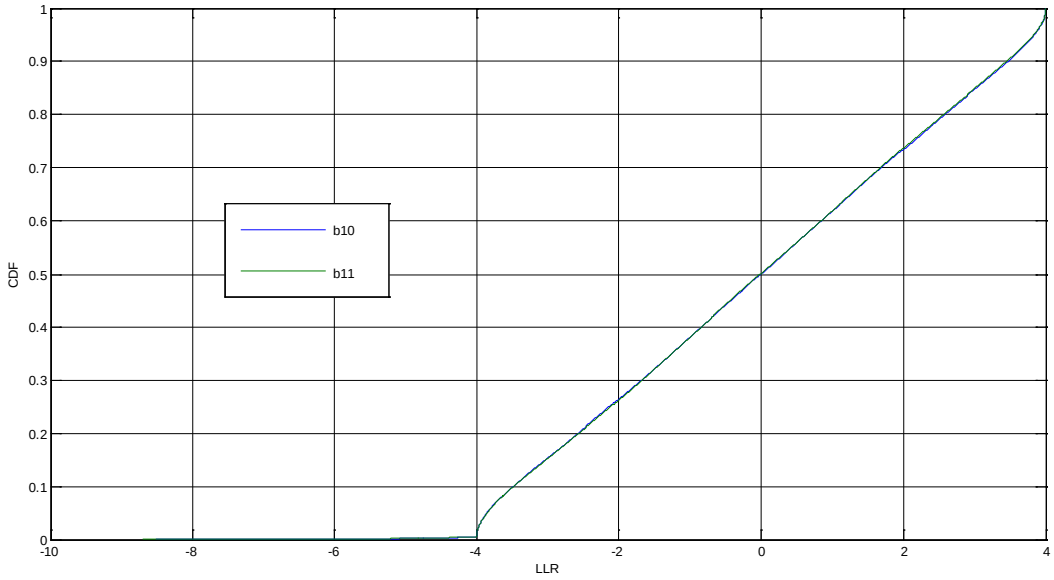

Figura 3.b.20 CDF degli LLR(b_{10}) e LLR(b_{11}) con n=12 e s=0dB

e per quanto riguarda le PDF:

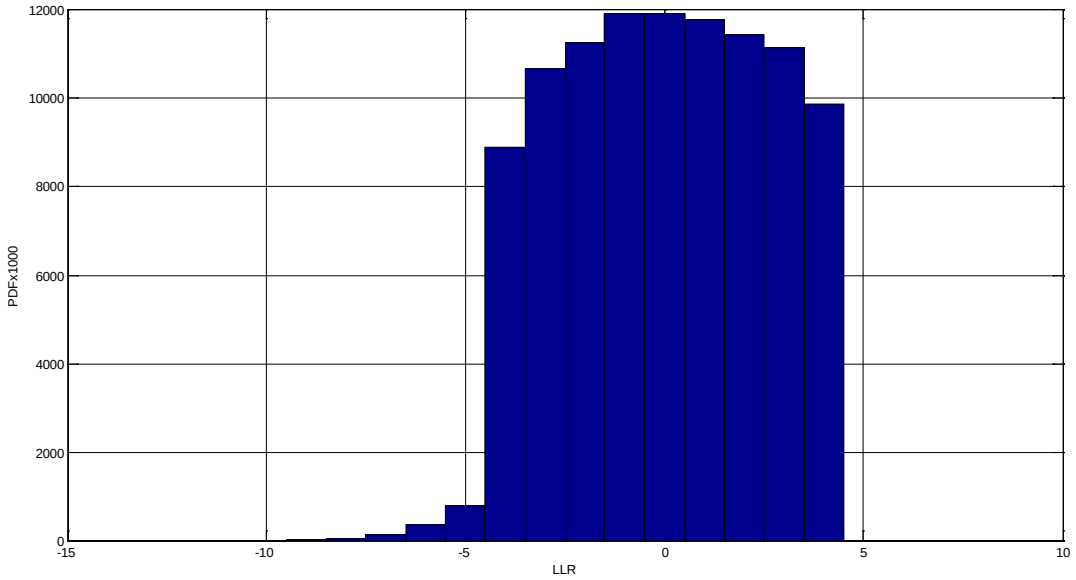

Figura 3.b.21 PDFx100000 del LLR(b_6) con n=8 e s=0dB

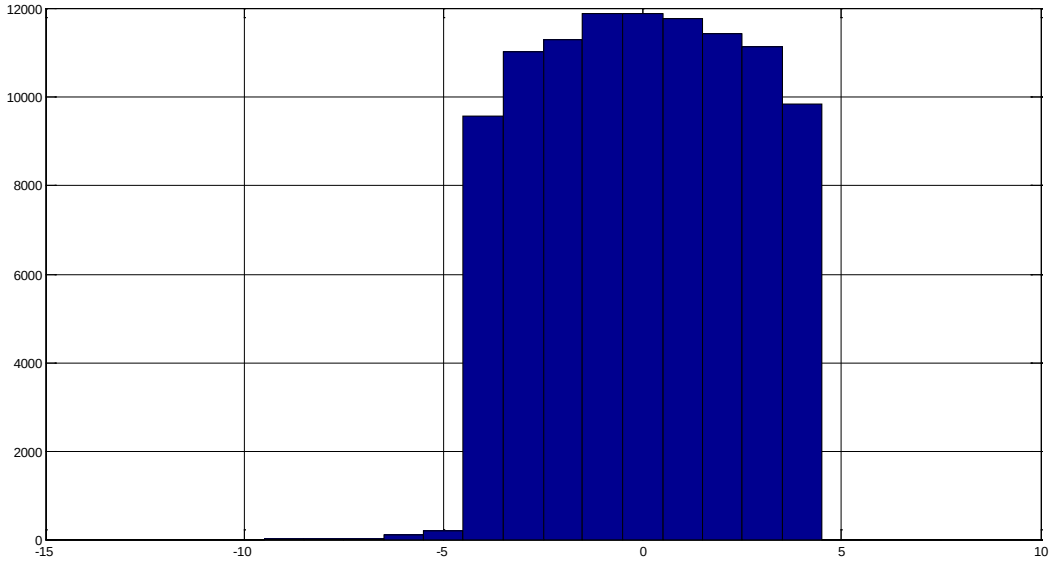

Figura 3.b.22 PDFx100000 del LLR(b_{10}) con n=12 e s=0dB

I grafici delle PDF degli altri bit sono simili a quelli di b_2; la concentrazione di campioni attorno a zero aumenta con l'indice i dei bit, le funzioni LLR sono quindi tutte poco affidabili.

Pertanto l'affidabilità della decisione finale dipenderà molto dalla qualità dell'interleaved coding.

3.b.5 Grafici con varianze minori

Nel paragrafo precedente sono stati realizzati i grafici delle PDF e CDF delle funzioni LLR nel caso $\sigma^2 = 0\text{dB}$ e i risultati si sono rivelati negativi, le LLR sono inaffidabili.

Per quanto riguarda il caso $\sigma^2 = -10\text{dB}$ i grafici delle PDF e CDF sono alquanto diversi da quelli del caso precedente:

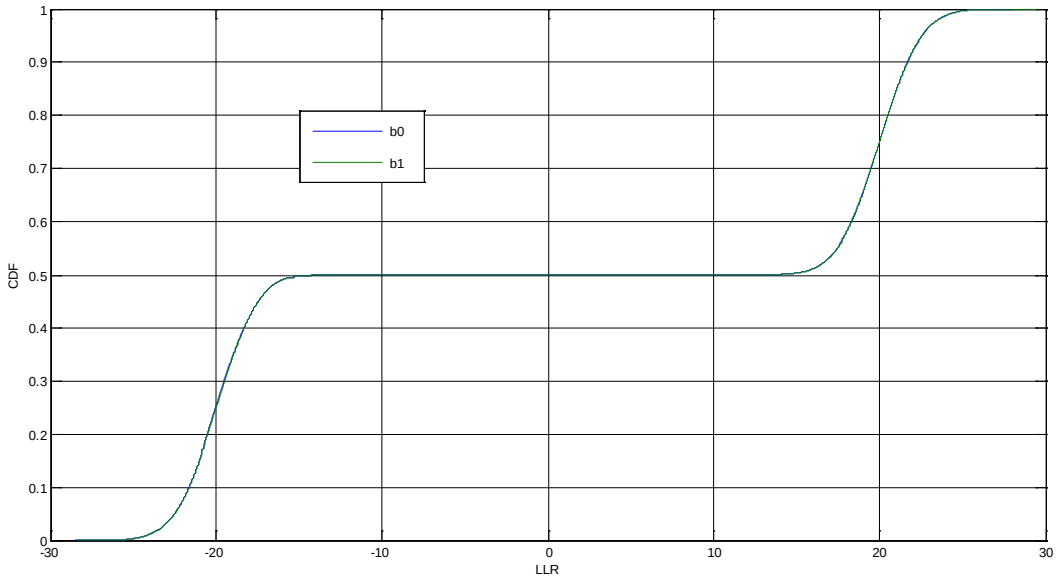

Figura 3.b.23 CDF degli LLR(b_0) e LLR(b_1) con n=2 e s=-10dB

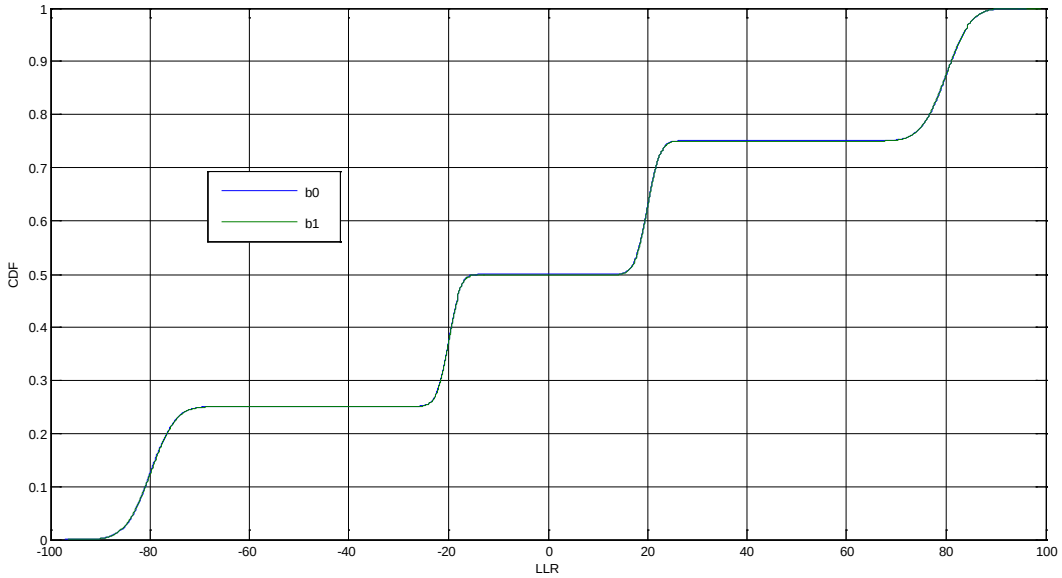

Figura 3.b.24 CDF degli LLR(b_0) e LLR(b_1) con n=4 e s=-10dB

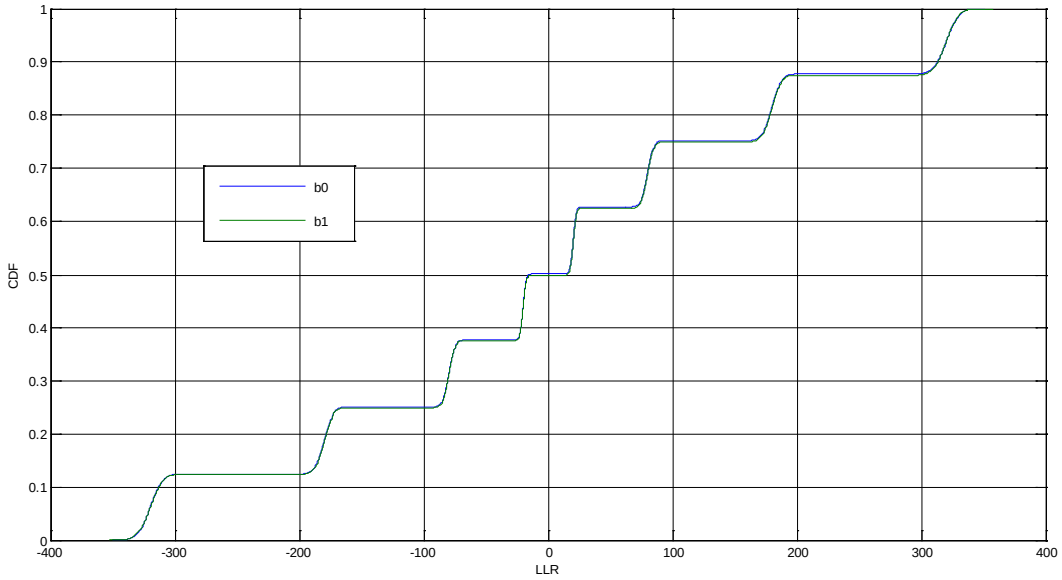

Figura 3.b.25 CDF degli LLR(b_0) e LLR(b_1) con n=6 e s=-10dB

Il grafico delle CDF degli LLR(b_0) assomiglia a quello di una variabile aleatoria discreta. Per quanto riguarda le PDF:

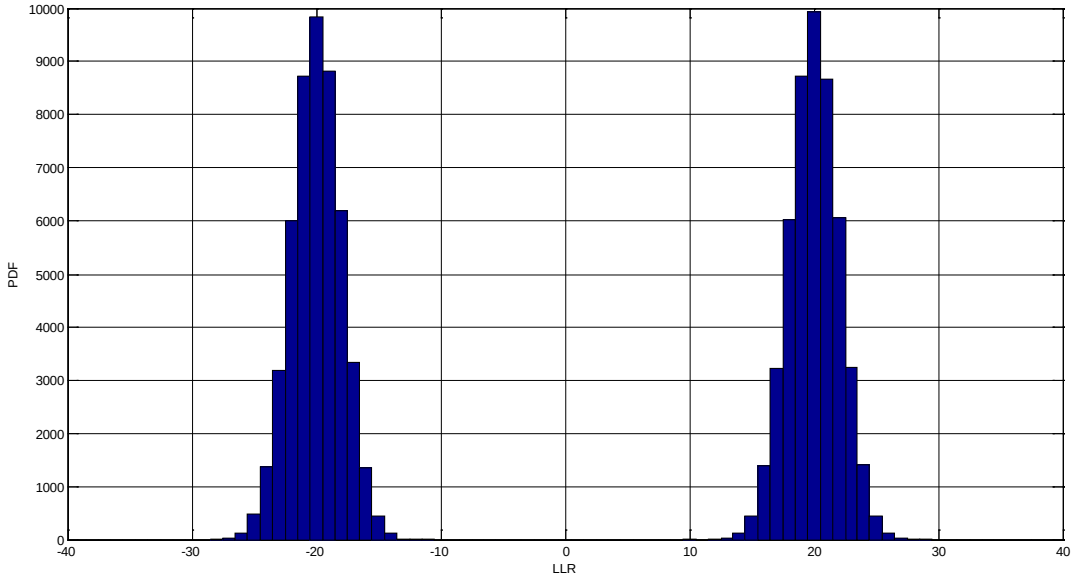

Figura 3.b.26 PDFx100000 del LLR(b_1) con n=2 e s=-10dB

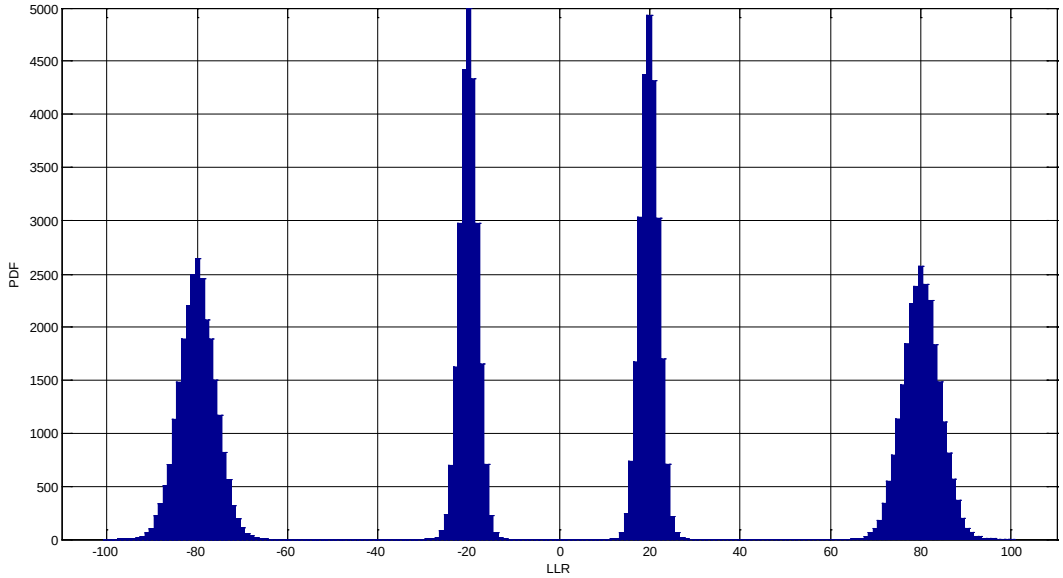

Figura 3.b.27 PDFx100000 del LLR(b_0) con n=4 e s=-10dB

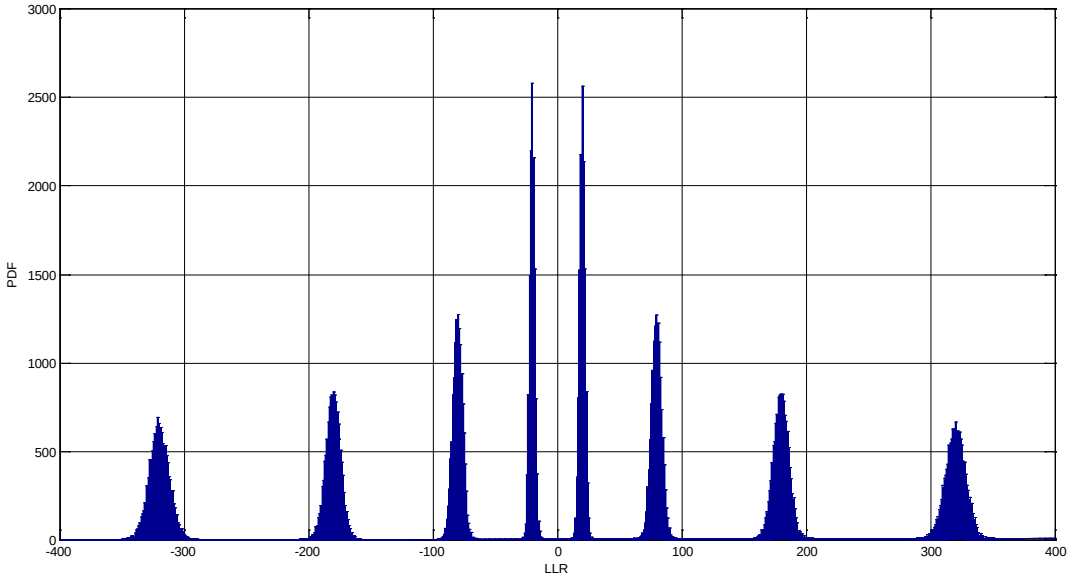

Figura 3.b.28 PDFx100000 del LLR(b_0) con n=6 e s=-10dB

Le PDF di LLR(b_0) sono pari e hanno un supporto composto dall'unione di più picchi a supporto compatto. Si nota che il numero di picchi corrisponde a m; tale osservazione unita alla parità della funzione suggerisce che a tali picchi corrispondano le colonne (righe se i dispari) della costellazione.

Intorno a zero non vi sono campioni, ciò significa che è quasi nulla la possibilità che di ottenere valori del LLR prossimi a zero.

Per quanto riguarda gli altri bit:

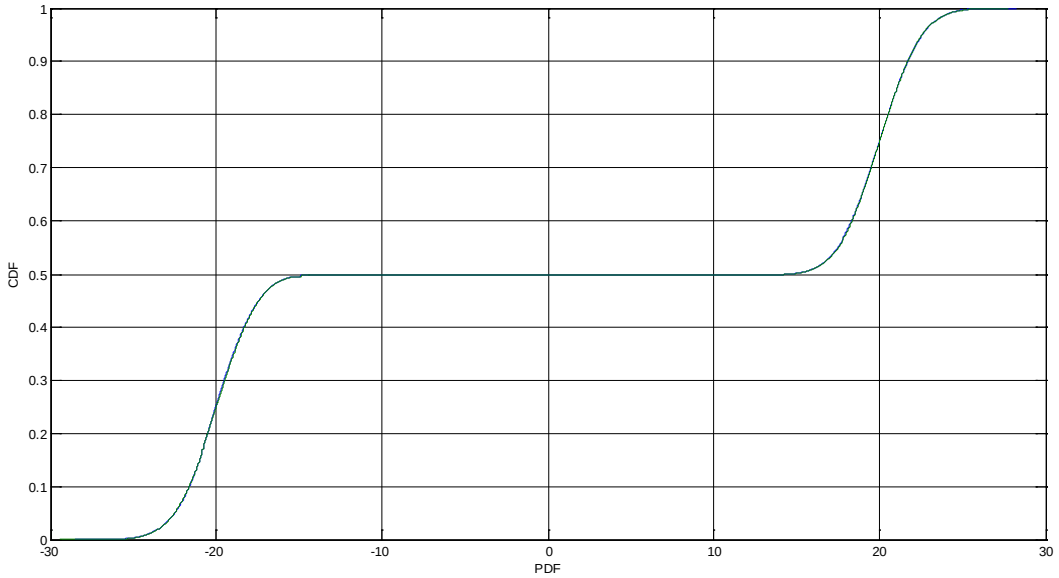

Figura 3.b.29 CDF degli LLR(b_2) e LLR(b_3) con n=4 e s=-10dB

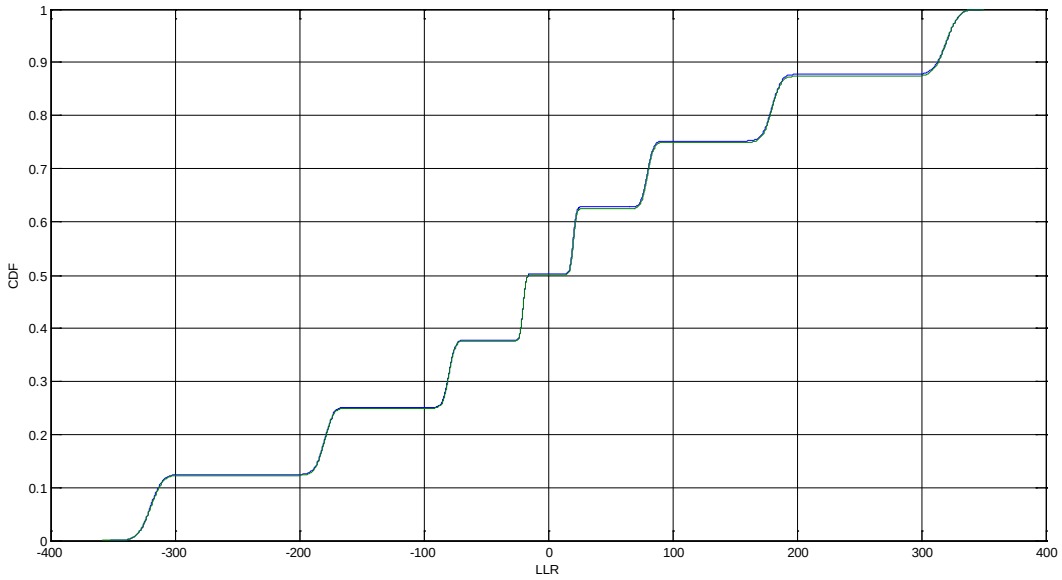

Figura 3.b.30 CDF degli LLR(b_2) e LLR(b_3) con n=8 e s=-10dB

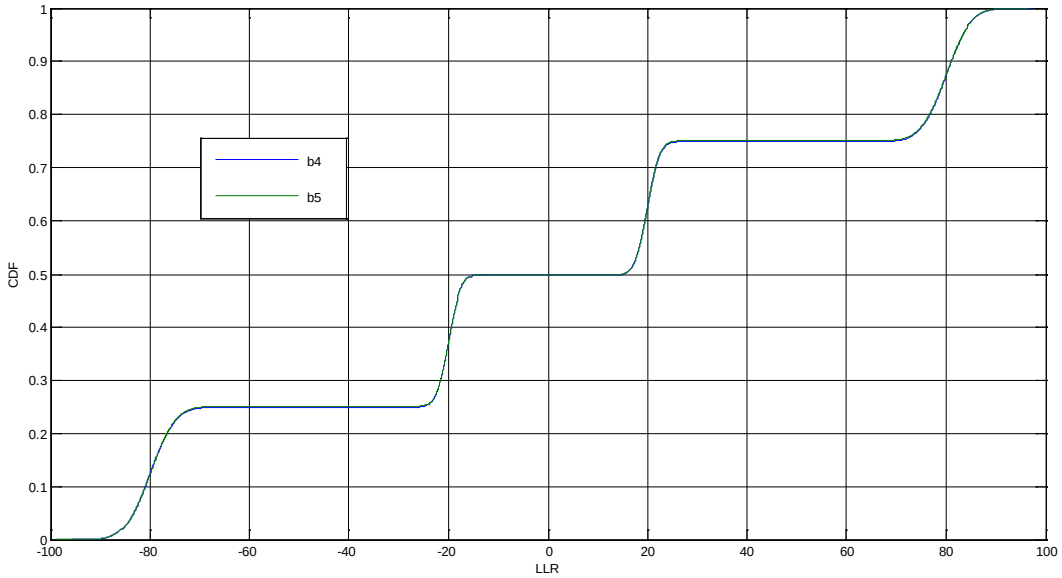

Figura 3.b.31 CDF degli LLR(b_4) e LLR(b_5) con n=8 e s=-10dB

Le CDF sono simili a quelle di b_0 e quindi simili alle distribuzioni delle variabili aleatorie discrete.

In particolare i bit più significativi hanno il maggior numero di gradini e quelli meno significativi un unico gradino.

Per quanto riguarda le PDF:

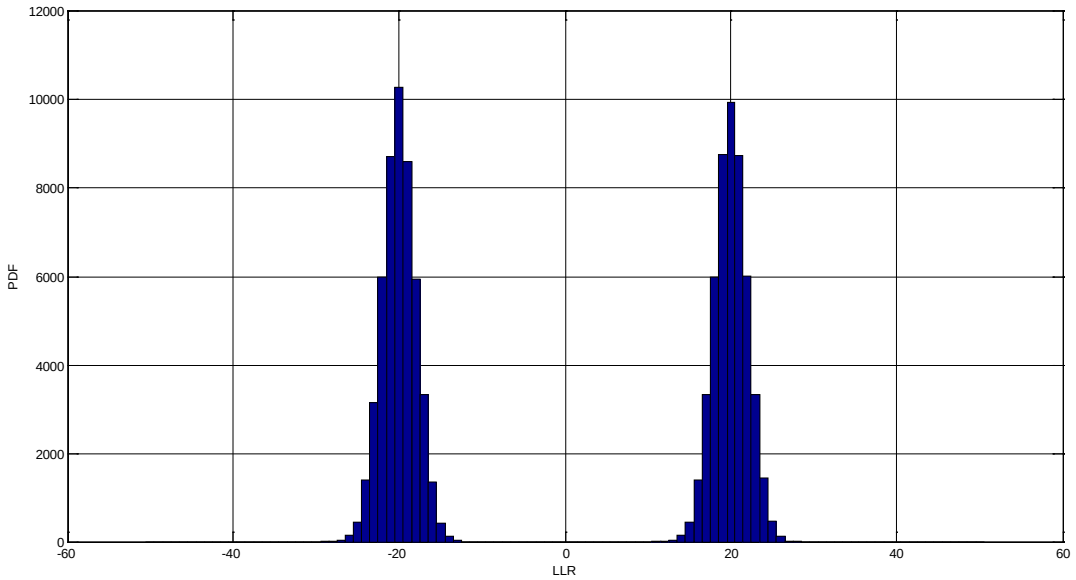

Figura 3.b.32 PDFx100000 del LLR(b_2) con n=4 e s=-10dB

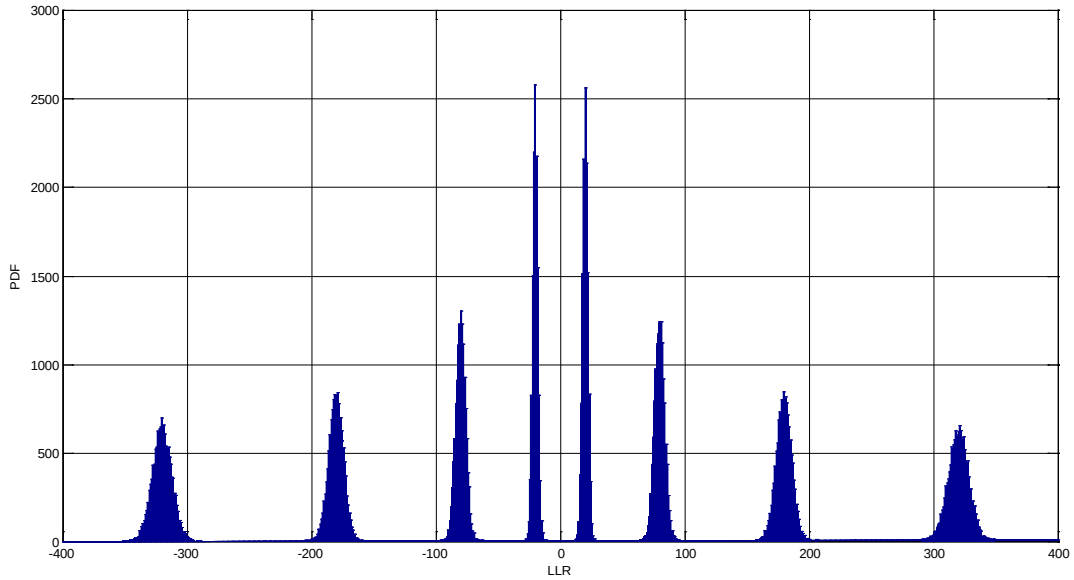

Figura 3.b.33 PDFx100000 del LLR(b_2) con n=8 e s=-10dB

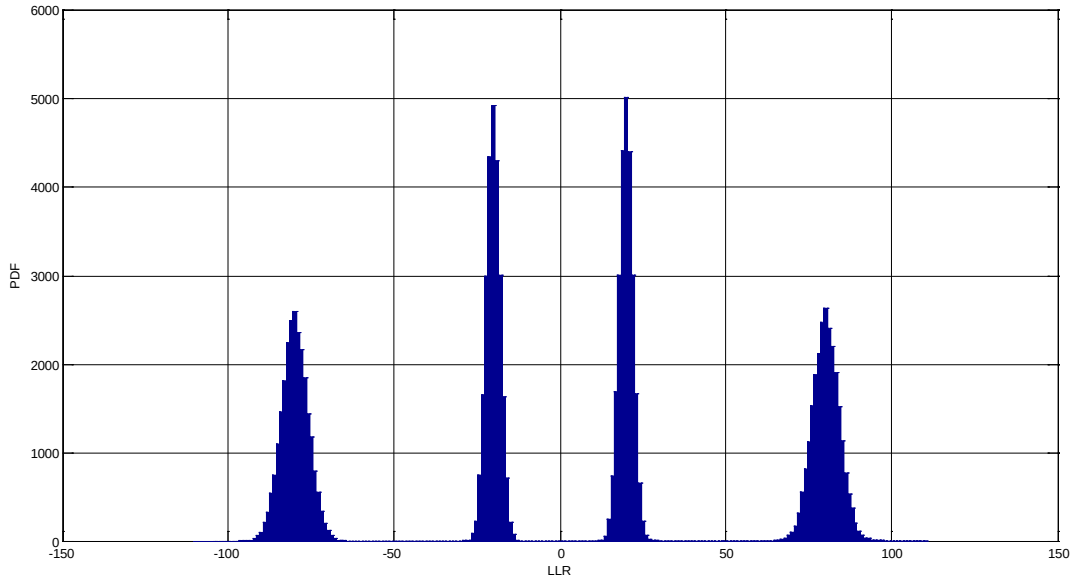

Figura 3.b.34 PDFx100000 della LLR(b_4) con n=8 e s=-10dB

Anche le PDF assomigliano a quelle di LLR(b_0), in particolare intorno a zero non vi sono campioni.

Con questo particolare valore della varianza è possibile fissare gli M_i, ad esempio $M_i=10$; ciò implica che si può supporre |LLR|>10 e la probabilità che ciò non sia verificato è molto bassa:

$$p_1 = e^{10}\, p_0 \quad \text{o viceversa} \quad p_0 = e^{10}\, p_1$$

In ogni caso le probabilità sono nettamente una maggiore dell'altra.

Si può affermare che nel caso $\sigma^2 = -10\text{dB}$ le funzioni LLR assumono valori prossimi a zero con una probabilità quasi nulla, e quindi sono quasi sempre utili e decisivi per decodificare un segnale in ricezione.

Vengono infine riportati alcune CDF e PDF degli LLR nel caso $\sigma^2 = -20\text{dB}$:

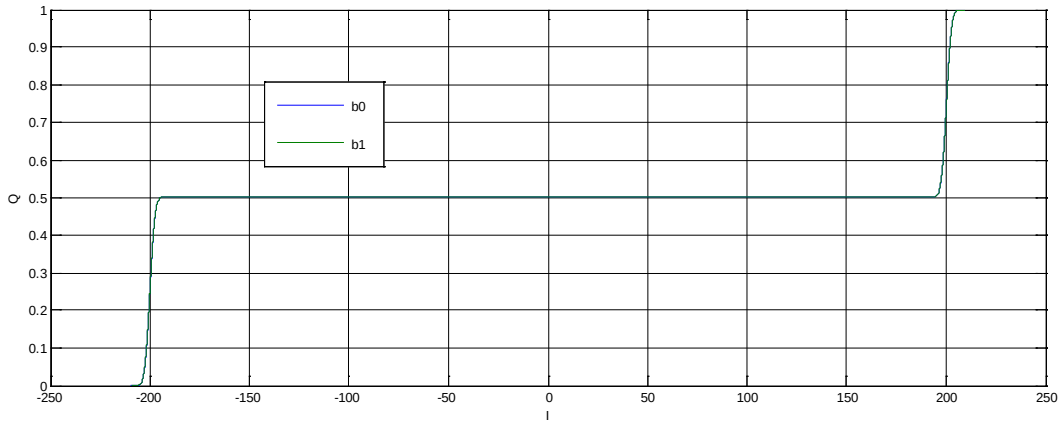

Figura 3.b.35 CDF degli LLR(b_0) e LLR(b_1) con n=2 e s=-20dB

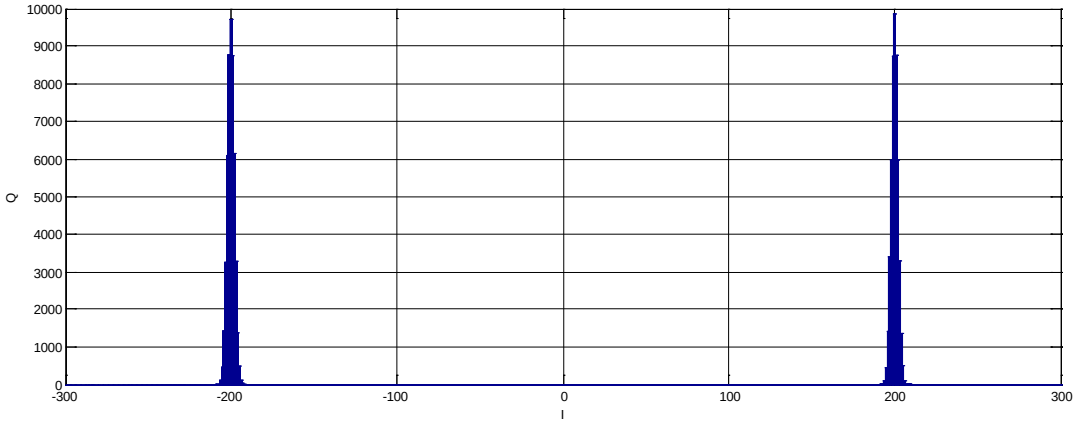

Figura 3.b.36 PDFx100000 di LLR(b_0) con n=2 e s=-20dB

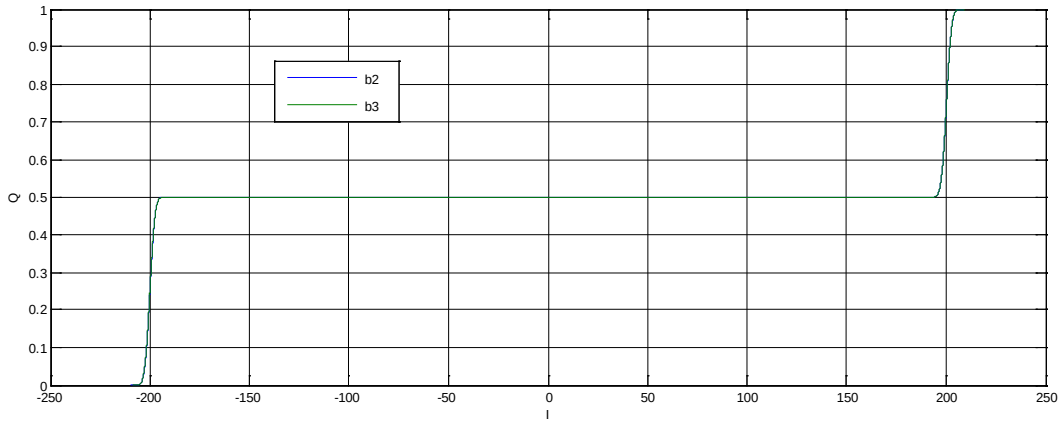

Figura 3.b.37 CDF degli LLR(b_2) e LLR(b_3) con n=4 e s=-20dB

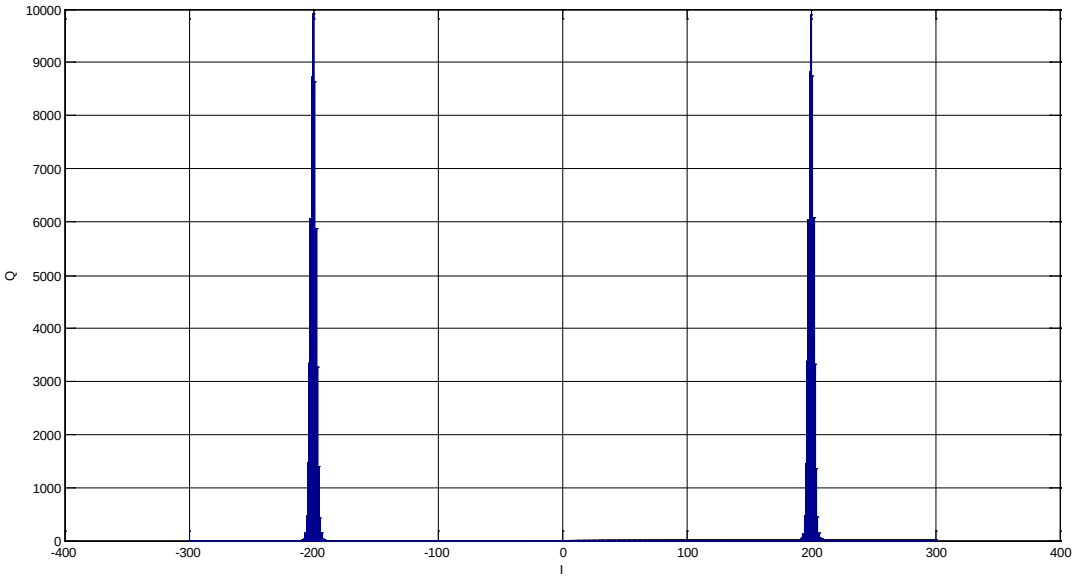

Figura 3.b.38 PDFx100000 di LLR(b_3) con n=4 e s=-20dB

L'affidabilità del demapper migliora se diminuisce l'effetto del rumore sul segnale ricevuto espresso da σ.

CAPITOLO 4

4 CONCLUSIONI

Nei capitoli precedenti sono stati realizzati dei modelli di QAM demapper per i sistemi di telecomunicazioni video digitali che si rifanno allo standard DVB-C2. Questi modelli utilizzano le funzioni LLR come criterio di decisione dei singoli bit.

Si sono realizzati tre diversi algoritmi LLR generico, LLR e LLRp per il calcolo degli LLR; il primo LLR generico applica direttamente la formula (4); il secondo LLR invece sfrutta la proprietà (1) della codifica Gray che permette di applicare la formula (6) riducendo i tempi di esecuzione e l'uso di memoria; il terzo LLRp applica le tre proprietà della codifica Gray evitando l'operazione di memorizzazione della codifica.

Grazie alle realizzazioni dei tre algoritmi è stato possibile confrontare i tempi di esecuzioni di questi e analizzare le funzioni LLR, in particolare quali valori assumono con più frequenza al variare di σ.

BIBLIOGRAFIA

[1] E. Akay e E. Ayanoglu, «Low Complexity Decoding of Bit-Interleaved Coded Modulation for M-ary QAM,» in *IEEE International Conference on Comunications, ICC 2004, vol. 2*, 2004, pp. 901-905.

[2] D. Lin, Y. Xiao e S. Li, «Low Complexity Soft Decision Technique for Gray Mapping Modulation,» *Wireless Pers Commun,* 2010.

[3] F. Tosato e P. Bisaglia, «Simplified Soft-Output Demapper for Binary Interleaved COFDM with Application to HIPERLAN/2,» *Imaging Systems Laboratory - HP Laboratories Bristol,* 2001.

[4] R. Ghaffar e R. Knopp, «Low Complexity Metrics for BICM SISO and MIMO systems,» in *Vehicular Technology Conference (VTC 2010-Spring), 2010 IEEE 71st*, 2010.

[5] C. Gianfranco e G. Pierobon, Teoria della probabilità e dei processi aleatori, Pàtron editore, 2006.

[6] N. Benvenuto, R. Corvaja, T. Erseghe e N. Laurenti, Comunication Systems Fundamentals and Desing Methods, John Wiley&Sons Ltd, 2007.

[7] ETSI, ETSI EN 302 769 V1.2.1, 2011-04.

[8] ETSI, DVB BlueBook A133, 2012.

APPENDICE A

PROGRAMMA LLR generico

“

```
//programma LLR generico
 #include <stdio.h>
 #include <math.h>

 #define N 12
 #define M 64
 #define T 10000000

 main()         //calcola LLR di n bit con una qualsiasi codifica (es Gray)
 {
   int n,m,i,l,j;   //n°bit,lato quadrato,indici: bit, riga, colonna
   double r[2],Md[M][M],s,vI[M],vQ[M],LLR[N],p1,p0,p; //input,distanze,output
   bool B[M][M][N]; //codifica, parte dal punto basso a sn,
                    //ordine: righe, colonne, bit

   scanf("%d%lf%lf%lf",&n,&s,&r[0],&r[1]);  //ottengo gli ingressi da prompt
   m=(int)pow(2,n/2);  //lato quadrato
   s=1/(s*s);

  codica(B);  //metodo che salva la codifica utilizzata in B

  p=(r[0]+m-1)/2;  //calcolo quadrati delle distanze sull'asse I
  for(i=0;i<m;i++)
     vI[i]=(p-i)*(p-i);

  p=(r[1]+m-1)/2;  //calcolo quadrati delle distanze sull'asse Q
  for(i=0;i<m;i++)
     vQ[i]=(p-i)*(p-i);

  for(i=0;i<m;i++)   //calcolo exp delle distanze di tutti i punti costellazione
    for(j=0;j<m;j++)
       Md[i][j]=exp((vI[j]+vQ[i])*-2*s);  //formula (4)

  for(i=0;i<n;i++)        //per tutti i bit calcolo LLR
  {
```

```
    p1=0;
    p0=0;
    for(l=0;l<m;l++)
      {
        for(j=0;j<m;j++)
          {
            if(B[l][j][i]==1)      //controllo se sta C1 con B
               p1=p1+Md[l][j];
            else
               p0=p0+Md[l][j];
          }
      }

     LLR[i]=log(p1/p0);

     if(LLR[i]>T)  //nel caso LLR sia troppo elevato in modulo o indeterminato
       LLR[i]=T;
    else if(LLR[i]<-1*T)
       LLR[i]=-1*T;
    else if(p1==0 & p0==0)
       LLR[i]=0;


   }

   return 0;
}  “
```

APPENDICE B

PROGRAMMA LLR

“

```
//programma LLR
 #include <stdio.h>
 #include <math.h>

 #define N 12
 #define M 64
 #define T 10000000

 main()   //calcola LLR di n bit guardando le codeword con una codifica Gray
 {
   int n,m,i,l;      //n°bit,lato quadrato,indici: bit, d
   double r[2],s,vD[M],LLR[N],p1,p0,p;   //input,distanze,output
   int d[M][N];    //codifica diagonale QAM

   scanf("%d%lf%lf%lf",&n,&s,&r[0],&r[1]); //ottengo gli ingressi da prompt
   m=(int)pow(2,n/2);  //lato quadrato
   s=1/(s*s);
   k=m/2;

  codica2(d);  //metodo che salva la codifica utilizzata in D

  p=(r[0]+m-1)/2;  //calcolo exp delle distanze sull'asse I
  for(l=0;l<m;l++)
     vD[l]=exp(-2*(p-l)*(p-l)*s);

  for(i=0;i<n;i=i+2)         //per tutti i bit pari calcolo LLR
  {
    p1=0;
    p0=0;
    for(l=0;l<m;l++)
      {
           if(d[l][i]==1)      //controllo se C1 con d
              p1=p1+vD[l];
           else
              p0=p0+vD[l];
      }
```

```
    LLR[i]=log(p1/p0);

    if(LLR[i]>T)  //nel caso LLR sia troppo elevato in modulo o indeterminato
       LLR[i]=T;
    else if(LLR[i]<-1*T)
       LLR[i]=-1*T;
    else if(p1==0 & p0==0)
       LLR[i]=0;
  }

  p=(r[1]+m-1)/2;  //calcolo exp delle distanze sull'asse Q
  for(l=0;l<m;l++)
     vD[l]=exp(-2*(p-l)*(p-l)*s);

  for(i=1;i<n;i=i+2)         //per tutti i bit dispari calcolo LLR
  {
    p1=0;
    p0=0;
    for(l=0;l<m;l++)
      {
            if(d[l][i]==1)      //controllo se C1 con d
               p1=p1+vD[l];
            else
               p0=p0+vD[l];
      }
    LLR[i]=log(p1/p0);

    if(LLR[i]>T)  //nel caso LLR sia troppo elevato in modulo o indeterminato
       LLR[i]=T;
    else if(LLR[i]<-1*T)
       LLR[i]=-1*T;
    else if(p1==0 & p0==0)
       LLR[i]=0;

  }

   return 0;
 }
"
```

APPENDICE C

PROGRAMMA LLRp

“

```
#include <stdio.h>
#include <math.h>

#define N 12
#define M 64
#define T 10000000

main()  //calcola LLR di n bit con codifica Gray senza guardare le codeword
{
  int n,m,i,j,k,l; //n°bit,lato quadrato, indici bit, riga o colonna, dim e miniblocco
  double r[2],vD[M],LLR[N],p1,p0,p,s;  //input,distanze,output,probabilità

  scanf("%d%lf%lf%lf",&n,&s,&r[0],&r[1]); //ottengo gli ingressi da prompt
  m=(int)pow(2,n/2); //lato quadrato
  p0=0;
  p1=0;
  s=1/(s*s);

   p=(r[0]+m-1)/2;  //calcolo gli exp delle distanze sull'asse I e LLR di b0
  for(i=0;i<m;i++)
    {
      vD[i]=exp(-2*(p-i)*(p-i)*s);  //calcolo LLR b0
      if(i<m/2)
         p1=p1+vD[i];
      else
         p0=p0+vD[i];
    }
   LLR[0]=log(p1/p0);

   if(LLR[0]>T)  //nel caso LLR sia troppo elevato in modulo o indeterminato
      LLR[0]=T;
   else if(LLR[0]<-1*T)
      LLR[0]=-1*T;
   else if(p1==0 & p0==0)
      LLR[0]=0;

  k=m/2;          //inizializzo dimensione miniblocchi
```

```
for(i=2;i<n;i++)        //per tutti gli altri bit pari calcolo LLR
{
  p1=0;
  p0=0;
  for(l=m-3*k/2;l<m-k;l++)  //divido il primo e l'ultimo miniblocchi in 2 parti
   {
       p1=p1+vD[l];    //seconda meta' ultimo miniblocco
       p0=p0+vD[l+k];  //prima metà primo miniblocco
   }

  for(l=m-k;l<m-k/2;l++)
   {
       p1=p1+vD[l];        //prima meta' primo miniblocco
       p0=p0+vD[l-m+k];     //seconda metà ultimo miniblocco
   }

  for(j=0;j<m/(2*k)-1;j++)  //scorro lungo i miniblocchi di lunghezza k
  {
     for(l=k*(1+4*j)/2;l<k*(3+4*j)/2;l++)  //sono in un miniblocco intermedio
        {
          p1=p1+vD[l];
          p0=p0+vD[l+k];
        }
   }

  LLR[i]=log(p1/p0);

  if(LLR[i]>T)  //nel caso LLR sia troppo elevato in modulo o indeterminato
    LLR[i]=T;
 else if(LLR[i]<-1*T)
    LLR[i]=-1*T;
 else if(p1==0 & p0==0)
    LLR[i]=0;

  k=k/2;    //aggiorno indici e dimensione miniblocchi
  i=i+1;
}

//per i bit dispari si ripete l'algoritmo sostituendo r[0] con r[1]
p1=0;
p0=0;
p=(r[1]+m-1)/2;  //calcolo exp delle distanze sull'asse Q e di b1
for(i=0;i<m;i++)
```

```
    {
      vD[i]=(p-i)*2;
      vD[i]=exp(-2*(p-i)*(p-i)*s);
      if(i<m/2)      //calcolo anche LLR di b1 sfruttando il ciclo for
        p1=p1+vD[i];
      else
        p0=p0+vD[i];
    }
LLR[1]=log(p1/p0);

if(LLR[1]>T)  //nel caso LLR sia troppo elevato in modulo o indeterminato
    LLR[1]=T;
 else if(LLR[1]<-1*T)
    LLR[1]=-1*T;
 else if(p1==0 & p0==0)
    LLR[1]=0;

k=m/2;
for(i=3;i<n;i++)        //per tutti gli altri bit dispari calcolo LLR
{
  p1=0;
  p0=0;

  for(l=m-(3*k)/2;l<m-k;l++)  //divido il primo e l'ultimo miniblocco in 2 parti
    {
        p1=p1+vD[l];    //seconda meta' primo miniblocco
        p0=p0+vD[l+k];   //prima metà ultimo miniblocco
    }

  for(l=m-k;l<m-k/2;l++)
    {
        p1=p1+vD[l];     //prima metà primo miniblocco
        p0=p0+vD[l-m+k];  //seconda metà ultimo miniblocco
    }

  for(j=0;j<m/(2*k)-1;j++)  //scorro lungo i miniblocchi di lunghezza k
  {
     for(l=k*(1+4*j)/2;l<k*(3+4*j)/2;l++)  //sono in un miniblocco intermedio
        {
          p1=p1+vD[l];
          p0=p0+vD[l+k];
        }
    }
```

```
    LLR[i]=log(p1/p0);

    if(LLR[i]>T)  //nel caso LLR sia troppo elevato in modulo o indeterminato
      LLR[i]=T;
   else if(LLR[i]<-1*T)
      LLR[i]=-1*T;
   else if(p1==0 & p0==0)
      LLR[i]=0;

    k=k/2;
    i=i+1;
  }

  return 0;
}
```

”

APPENDICE D

PROGRAMMA LLRpTimeTester

“

```
#include <stdio.h>
#include <math.h>
#include <time.h>

#define N 12
#define M 64
#define L 10000000 //10^7numero esecuzioni
#define T 10000000 //valore massimo LLR

main()  //calcola LLR di n bit con codice Gray senza guardare le codeword L volte
{
  int n,m,i,l,j,k,b;   //n°bit,lato quadrato,indici: bit,riga,colonna,dim,miniblocco
  double r[2],vD[M],LLR[N],p1,p0,p,s,diff;
  //input,distanze,output,probabilità,varianza,tempo
  time_t start,end;  //inizio e fine cronometro

  scanf("%d%lf%lf%lf",&n,&s,&r[0],&r[1]);  //ottengo gli ingressi da prompt
  m=(int)pow(2,n/2);  //lato quadrato
  s=1/(s*s);

  time(&start);  //parte il cronometro
  for(b=0;b<L;b++)  //eseguo L volte il calcolo LLR
 {
  p0=0;
  p1=0;
  p=(r[0]+m-1)/2;  //calcolo gli exp delle distanze sull'asse I e di b0
  for(i=0;i<m;i++)
     {
      vD[i]=exp(-2*(p-i)*(p-i)*s);
      if(i<m/2)
         p1=p1+vD[i];
      else
         p0=p0+vD[i];
     }
   LLR[0]=log(p1/p0);
```

```
 if(LLR[0]>T)  //nel caso LLR sia troppo elevato in modulo o indeterminato
   LLR[0]=T;
 else if(LLR[0]<-1*T)
   LLR[0]=-1*T;
 else if(p1==0 & p0==0)
   LLR[0]=0;

k=m/2;
for(i=2;i<n;i++)        //per tutti i bit pari calcolo LLR
{
  p1=0;
  p0=0;
  for(l=m-3*k/2;l<m-k;l++)  //divido il primo miniblocchi in 2 parti
   {
       p1=p1+vD[l];
       p0=p0+vD[l+k];
   }

  for(l=m-k;l<m-k/2;l++)
   {
        p1=p1+vD[l];
        p0=p0+vD[l-m+k];
   }

  for(j=0;j<m/(2*k)-1;j++)  //scorro lungo i miniblocchi di lunghezza k
  {
     for(l=k*(1+4*j)/2;l<k*(3+4*j)/2;l++)  //sono in un miniblocco intermedio
        {
          p1=p1+vD[l];
          p0=p0+vD[l+k];
        }
   }

  LLR[i]=log(p1/p0);

  if(LLR[i]>T)  //nel caso LLR sia troppo elevato in modulo o indeterminato
    LLR[i]=T;
  else if(LLR[i]<-1*T)
    LLR[i]=-1*T;
  else if(p1==0 & p0==0)
    LLR[i]=0;

  k=k/2;
```

```
  i=i+1;
}


p1=0;
p0=0;
p=(r[1]+m-1)/2;  //calcolo exp delle distanze sull'asse Q e di b1
for(i=0;i<m;i++)
  {
    vD[i]=(p-i)*2;
    vD[i]=exp(-2*(p-i)*(p-i)*s);
    if(i<m/2)
       p1=p1+vD[i];
    else
       p0=p0+vD[i];
  }
LLR[1]=log(p1/p0);

if(LLR[1]>T)  //nel caso LLR sia troppo elevato in modulo o indeterminato
   LLR[1]=T;
else if(LLR[1]<-1*T)
   LLR[1]=-1*T;
else if(p1==0 & p0==0)
   LLR[1]=0;

k=m/2;
for(i=3;i<n;i++)        //per tutti i bit dispari calcolo LLR
{
  p1=0;
  p0=0;

  for(l=m-(3*k)/2;l<m-k;l++)  //divido il primo miniblocchi in 2 parti
    {
        p1=p1+vD[l];
        p0=p0+vD[l+k];
    }

  for(l=m-k;l<m-k/2;l++)
    {
         p1=p1+vD[l];
         p0=p0+vD[l-m+k];
    }
```

```
    for(j=0;j<m/(2*k)-1;j++)  //scorro lungo i miniblocchi di lunghezza k
     {
        for(l=k*(1+4*j)/2;l<k*(3+4*j)/2;l++)  //sono in un miniblocco intermedio
            {
              p1=p1+vD[l];
              p0=p0+vD[l+k];
            }
      }
    LLR[i]=log(p1/p0);

    if(LLR[i]>T)  //nel caso LLR sia troppo elevato in modulo o indeterminato
      LLR[i]=T;
   else if(LLR[i]<-1*T)
      LLR[i]=-1*T;
   else if(p1==0 & p0==0)
      LLR[i]=0;

    k=k/2;
    i=i+1;
  }
 }

  time(&end);   //fermo il cronometro
  diff=difftime(end,start);  //calcolo il tempo totale

  return 0;
}
 ”
```

APPENDICE E

PROGRAMMA codifica

“

```
#include <stdio.h>
#include <math.h>

#define N 12
#define M 64

main()   //genera e salva una costellazione QAM a n bit gray su una matrice B
{
  int n,m,i,l,j,k,b;   //n°bit,lato quadrato,indici: bit,riga,colonna,dim,miniblocco
  double p; //input,distanze,output
  bool B[M][M][N]; //codifica, parte da punto basso a sn
                        // ordine: righe, colonne, bit

  scanf("%d,&n); //ottengo l'ingressi da prompt
  m=(int)pow(2,n/2); //lato quadrato

  //inizializzo la matrice B con codifica Gray
  k=m/2;
  for(l=0;l<k;l++)  //primi 2 bit b0 e b1 trattati separatamente
     {
       for(j=0;j<m;j++)   //inizializzo 2 bit alla volta
       {
         B[l][j][1]=true;   //b1
         B[l+k][j][1]=false;
         B[j][l][0]=true;    //b0
         B[j][l+k][0]=false;
       }

     }

  for(i=2;i<n;i=i+2)   //inizializzo la codifica
   {
     for(l=m-(3*k)/2;l<m-k;l++)  //divido il primo e l'ultimo miniblocchi in 2 parti
      {
          for(j=1;j<m;j++)   //inizializzo 2 bit alla volta
           {
            B[l][j][i+1]=true;          //b(i+1) dispari
```

```
            B[l+k][j][i+1]=false;
            B[j][l][i]=true;        //bi pari
            B[j][l+k][i]=false;
           }
      }

     for(l=m-k;l<m-k/2;l++)  //seconda meta' miniblocco
      {
          for(j=1;j<m;j++)   //inizializzo 2 bit alla volta
           {
             B[l][j][i+1]=true;      //b(i+1) dispari
             B[l-m+k][j][i+1]=false;
             B[j][l][i]=true;            //bi pari
             B[j][l-m+k][i]=false;
           }
      }

     for(b=0;b<m/(2*k)-1;b++)  //scorro lungo i miniblocchi di lunghezza k
     {
       for(l=k*(1+4*b)/2;l<k*(3+4*b)/2;l++)  //sono in un miniblocco intermedio
           {
             for(j=0;j<m;j++)  //inizializzo 2 bit alla volta
              {
                B[l][j][i+1]=true;       //b(i+1) dispari
                B[l+k][j][i+1]=false;
                B[j][l][i]=true;           //bi pari
                B[j][l+k][i]=false;
              }
           }
      }
     k=k/2;
   }

  return i;
}
“
```

APPENDICE F

PROGRAMMA LLRTesterPDF

“

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <ctype.h>
#include <time.h>

#define N 12
#define M 64
#define F 200000  //doppio numero campioni
#define T 10000000 //limite LLR

main()
{
  //variabili utilizzate
  int n,m,i,j,k,l,b1,b2,d,segno;   //n°bit,lato quadrato,indici
  double r[2],sg[2],s,vD[M],LLR[N],p1,p0,p,diff; //input,rumore,distanze,output
  FILE *fp,*out;  //puntatore file da aprire
  char stringa[F];  //stringa dove salvare i dati estratti dal file

  out=fopen("output.txt","w");  //apro il file degli LLR, da scrivere
  scanf("%d%lf",&n,&s); //ottengo gli ingressi da prompt

  fp=fopen("rumor.txt","r");  //apro file coi rumori, da leggere

  m=(int)pow(2,n/2); //lato quadrato
  s=1/(s);  //inverto s varianza per non dover ripetere divisione
  k=m/2;

  b1=getc(fp);  //elimino primo spazio vuoto nei file

  //leggo da file il rumore gaussiano
  while(((b1=getc(fp))!=EOF)&&(b2!=EOF))
  {
     segno=1;  //estraggo s[0] rumore asse I
     if(b1=='-')
       {
          segno=-1;  //correggo il segno
```

```
      b1=getc(fp);
    }

    j=0;
    while(!isspace(b1))
     {
      stringa[j]=b1;
      b1=getc(fp);
      j++;
     }
    stringa[j+1]='\0';
    sg[0]=segno*atof(stringa);

          //estraggo s[1] rumore asse Q
  b2=getc(fp);
  segno=1;
  if(b2=='-')
    {
      segno=-1;  //correggo il segno
      b2=getc(fp);
    }

  j=0;
  while(!isspace(b2))
     {
      stringa[j]=b2;
      b2=getc(fp);
      j++;
     }
  stringa[j+1]='\0';
  sg[1]=segno*atof(stringa);

  r[0]=sg[0]+2*(rand()%(m))-m+1;  //ingressi con random
  r[1]=sg[1]+2*(rand()%(m))-m+1;  //aggiungo il rumore gaussiano


  p0=0;  //inizializzo probabilità
  p1=0;
      p=(r[0]+m-1)/2;  //calcolo gli exp delle distanze sull'asse I
      for(i=0;i<m;i++)
        {
           vD[i]=exp(-2*(p-i)*(p-i)*s);
           if(i<m/2)            //calcolo LLR di b0
```

```
            p1=p1+vD[i];
          else
            p0=p0+vD[i];
      }
  LLR[0]=log(p1/p0);

  if(LLR[0]>T) //nel caso LLR sia troppo elevato in modulo o indeterminato
     LLR[0]=T;
 else if(LLR[0]<-1*T)
     LLR[0]=-1*T;
  else if(p1==0 & p0==0)
     LLR[0]=0;

  k=m/2;
  for(i=2;i<n;i++)        //per tutti i bit pari calcolo LLR
   {
     p1=0;
     p0=0;
     for(l=m-3*k/2;l<m-k;l++)
      {
           p1=p1+vD[l];    //prima meta' ultimo miniblocco
           p0=p0+vD[l+k];   //seconda meta' primo miniblocco
      }

     for(l=m-k;l<m-k/2;l++)
      {
           p1=p1+vD[l];      //seconda meta' ultimo miniblocco
           p0=p0+vD[l-m+k];   //prima meta' primo miniblocco
      }

     for(j=0;j<m/(2*k)-1;j++)  //scorro lungo i miniblocchi di lunghezza k
      {
        for(l=k*(1+4*j)/2;l<k*(3+4*j)/2;l++)  //sono in un miniblocco intermedio
              {
                 p1=p1+vD[l];
                 p0=p0+vD[l+k];
              }
      }

    LLR[i]=log(p1/p0);  //calcolo LLR

    if(LLR[i]>T) //nel caso LLR sia troppo elevato in modulo o indeterminato
          LLR[i]=T;
```

```
   else if(LLR[i]<-1*T)
          LLR[i]=-1*T;
   else if(p1==0 & p0==0)
          LLR[i]=0;

    k=k/2;
    i=i+1;
 }


p1=0;
p0=0;
p=(r[1]+m-1)/2;  //calcolo exp delle distanze sull'asse Q
for(i=0;i<m;i++)
  {
        vD[i]=(p-i)*2;
        vD[i]=exp(-2*(p-i)*(p-i)*s);
        if(i<m/2)                 //calcolo LLR b1
          p1=p1+vD[i];
        else
          p0=p0+vD[i];
  }
LLR[1]=log(p1/p0);

if(LLR[1]>T)  //nel caso LLR sia troppo elevato in modulo o indeterminato
   LLR[1]=T;
else if(LLR[1]<-1*T)
   LLR[1]=-1*T;
else if(p1==0 & p0==0)
   LLR[1]=0;

k=m/2;
for(i=3;i<n;i++)        //per tutti i bit dispari calcolo LLR
 {
    p1=0;
    p0=0;

    for(l=m-(3*k)/2;l<m-k;l++) //divido il primo e l'ultimo miniblocchi in 2
      {
        p1=p1+vD[l];    //prima meta' ultimo miniblocco
        p0=p0+vD[l+k];   //seconda meta' primo miniblocco
      }
```

```
            for(l=m-k;l<m-k/2;l++)
              {
                 p1=p1+vD[l];              //seconda meta' ultimo miniblocco
                 p0=p0+vD[l-m+k];    //prima metà primo miniblocco
              }

            for(j=0;j<m/(2*k)-1;j++)  //scorro lungo i miniblocchi di lunghezza k
             {
               for(l=k*(1+4*j)/2;l<k*(3+4*j)/2;l++)//sono in un miniblocco intermedio
                    {
                         p1=p1+vD[l];
                         p0=p0+vD[l+k];
                    }
             }
           LLR[i]=log(p1/p0);  //calcolo LLR

           if(LLR[i]>T) //nel caso LLR sia troppo elevato in modulo o indeterminato
                  LLR[i]=T;
           else if(LLR[i]<-1*T)
                  LLR[i]=-1*T;
           else if(p1==0 & p0==0)
                  LLR[i]=0;

            k=k/2;
            i=i+1;
        }

      for(i=0;i<n;i++)
       fprintf(out,"%lf\t",LLR[i]);   //salvo su file gli LLR
       fprintf(out,"\n");
         //fine ciclo calcolo LLR ripeto con nuovo rumore e input

    }

    fclose(fp);    //chiudo i file rimasti aperti
    fclose(out);

    return 0;
  }
"
```

APPENDICE G

ACRONIMI

DVB	Digital Video Broadcasting
DVB-C	Digital Video Broadcasting for Cable
DVB-C2	Digital Video Broadcasting for Cable 2 (second generation)
MPEG	Moving Picture Experts Group
BICM	Bit Interleaved Coding and Modulation
PLPs	Physical Layer Pipes
OFDM	Orthoginal Frequency-Division Multiplexing
BMAP	Bit-Mapper
IBMAP	Inverse Bit-Mapper
QAM	Quadrature Amplitude Modulation
PAM	Pulse Amplitude Modulation
PSK	Phase Shift Keying
LLR	Log Likelihood Ratio
SNR	Signal-to-Noise Ratio
HIPERLAN/2	HIgh PErformance Radio LAN type 2
SISO	Single Input and Single Output
MIMO	Multiple Input and Multiple Output
LLRp	Log Likelihood Ratio perfective (algorithm)
CDF	Comulative Density Function
PDF	Probability Density Function

APPENDICE H

SIMBOLI

A	Può indicare I o Q
A_x	Può indicare I_x o Q_x
B	Matrice nxn che memorizza codeword
b_l	Sequenza binaria
b_l'	Sequenza binaria ricostruita
b_i	Bit i-esimo di una codeword
C_{i1},C_{i0}	Set di codeword in cui b_i=*1 o 0*
C_{it}	Set di codeword in cui $b_i=t$
D	Vettore che memorizza le codeword della diagonale principale
d_{ix}, d_{qx}	Proiezioni sugli assi delle distanze di r dagli x
F	Numero campioni programma LLRpTesterPDF
f_b	Frequenza sequenze binarie
I, Q	Coordinate punto costellazione ricevuto r
I_m, Q_m	Coordinate punto costellazione trasmesso tx_m
I_x, Q_x	Coordinate punto x della costellazione
i	Indice bit nella codeword
j	Indice colonne matrici B e
k	Dimensione miniblocco
k_i	Successione dimensioni miniblocchi
L	Numero esecuzioni programmi TimeTester
l	Indice righe matrici B e D
M	Costante positiva per l'affidabilità LLR
M_i	Successione costanti affidabilità LLR(b_i)
m	Numero punti in un lato della costellazione QAM
n	Numero di bit della modulazione
r	Punto costellazione in ricezione
$r(t)$	Segnale in ricezione
p_1,p_0	Probabilità: $P[I,Q\|b_i=1]$, $P[I,Q\|b_i=0]$
ρ_I, ρ_Q	Attenuazioni nella costellazione
s	Varianza del rumore nei programmi e algoritmi
s_{tx}	Segnale in trasmissione

σ	Deviazione standard del rumore (radice della varianza)
t	Indice booleano
T_b	Periodo sequenze binarie
tx	Punto della costellazione estratto in trasmissione
v_I, v_Q,v_L	Vettori delle proiezioni delle distanze
φ_1, φ_2	Funzioni ortonormali che generano i segnali s_{tx}
w	Rumore
x	Punto della costellazione