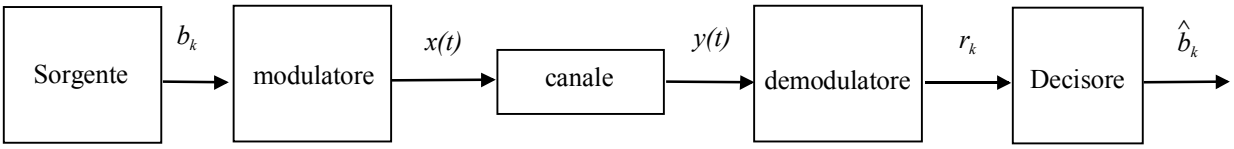
Fondamenti di Segnali e Trasmissione

Lezione di Laboratorio del 29/5/2009 (Alessandro Tomasoni)

Sistemi di trasmissione numerica

Fino ad ora abbiamo imparato ad utilizzare SCILAB per rappresentare, combinare, convolvere e trasformare segnali analogici. Oggi useremo parte di quello che abbiamo imparato per verificare in forma "sperimentale" alcuni concetti che avete visto a lezione, sulla trasmissione di segnali numerici (o digitali).
Trasmettere un segnale numerico significa trasmettere una sequenza di bit, tramite un segnale analogico, ovvero tramite una forma d'onda che varia con continuità nel tempo. Verificheremo tramite simulazione alcune nozioni legate ai segnali da usare, e a come prendere le decisioni sui segnali ricevuti. Costruiremo un modello SCILAB semplificato di un sistema di comunicazione, che ci permetterà di valutare la bontà delle nostre strategie.



Il sistema di trasmissione numerica semplificato che prenderemo in considerazione sarà costituito dai seguenti blocchi:

- **sorgente:** in un caso reale tale sorgente potrebbe rappresentare un file, quindi una sequenza di bit b_k che vogliamo trasmettere, o un segnale televisivo in uscita da una telecamera digitale quindi già sotto forma di sequenza di bit, oppure un segnale vocale trasformato in segnale elettrico da un trasduttore, campionato e convertito in formato numerico da un convertitore analogico/digitale (A/D).
- **modulatore:** trasforma la sequenza di bit b_k in un segnale continuo nel tempo *x(t)* che la rappresenta. Per esempio si trasmetterà una forma d'onda per comunicare uno zero ed un'altra, diversa dalla prima, per comunicare un uno.
- **canale:** durante il passaggio attraverso il canale di comunicazione, il segnale *x(t)* può subire delle modifiche dovute ad attenuazione, distorsione, interferenze, rumori. Quando si presenta al ricevitore sarà *y(t)*. Naturalmente nel caso di canale ideale non escludiamo la possibilità che sia $y(t)= x(t)$.
- **demodulatore:** tramite operazioni da decidere, trasforma il segnale continuo nel tempo *y(t)* in una sequenza di valori r_k, "parenti" dei bit trasmessi b_k.
- **decisore:** basandosi sulla sequenza r_k e su criteri da definire, sceglie quali bit b_k possono essere stati trasmessi.

N.B. i termini *modulatore* e *demodulatore* non corrispondono a traslazioni in frequenza! In questo contesto *modulatore* significa *generatore di forme d'onda* ("modulate" dai dati) e *demodulatore* significa *elaboratore del segnale ricevuto*.

Simulazione di una sorgente di bit

Il primo anello della catena è una sorgente di bit: come detto, in un caso reale tale sorgente potrebbe rappresentare tanto un file, quanto un segnale analogico convertito in formato numerico. Salvo casi particolari non si ha motivo di supporre che uno dei due valori (0 o 1) che può assumere un bit, sia più probabile dell'altro, pertanto la sequenza **b** in uscita dalla sorgente si può modellizzare tramite una sequenza di estrazioni da una variabile casuale binaria che assume i valori 0 o 1 con ugual probabilità p=1/2. In SCILAB possiamo usare la funzione **rand(R,C)**, che estrae una matrice RxC di valori indipendenti da una variabile casuale distribuita uniformemente tra 0 e 1

```
// sorgente.sce

b=round(rand(1,N));
```

```
--> N=10;
--> exec('sorgente.sce');
```

Simulazione di un modulatore

Essenzialmente abbiamo detto che il modo più semplice che viene in mente per trasmettere una sequenza di bit è quello di metterci d'accordo con il ricevitore che trasmetteremo una forma d'onda per comunicare uno zero ed un'altra diversa dalla prima per comunicare un uno. La prima idea che potrebbe venire in mente (forse non proprio la prima, ma la prima buona...) sarebbe quella di utilizzare una certa forma g(t) con ampiezza positiva, p.e. per lo zero, e la stessa, ma con ampiezza negativa, per l'uno. Dopo che la prima forma d'onda g(t) è esaurita, p.e. supponiamo che duri T secondi, passiamo a trasmettere il secondo bit in maniera analoga, e così via per tutti e N i bit. Risulterebbe allora:

$$x(t) = (-1)^{b_0} g(t) + (-1)^{b_1} g(t-T) + (-1)^{b_2} g(t-2T) + \ldots (-1)^{b_N} g(t-NT) = \sum_{k=0}^{N-1} (-1)^{b_k} g(t-kT) \qquad (1)$$

A questo punto dobbiamo decidere com'è fatta *g(t)* e quanto dura. Per la forma, forse la scelta più intuitiva sarebbe usare un rettangolo. Per la durata cominciamo ad incappare in qualche vincolo: p.e. se stiamo trasmettendo un file, possiamo scegliere la durata con una certa libertà (sempre tenendo presente che l'intera trasmissione richiederà un tempo *NT*), se stiamo trasmettendo un segnale televisivo o vocale, la nostra sorgente fornirà bit da trasmettere con un certo ritmo, e noi dobbiamo rispettare tale ritmo. Per ora fissiamo arbitrariamente una durata *T* di 1 s e vediamo come risulterebbe il nostro segnale *x(t):*

```
--> T=1;
--> dt=T/40;
--> tg=0:dt:2*T;
--> g=zeros(1,length(tg));
--> I=find(tg<T);
--> g(I)=1;
```

Ora dobbiamo sommare tanti di questi impulsi, traslati tra loro di *T* e pesati con ampiezze +1 o –1 a seconda che b_k sia 0 o 1, rispettivamente. Per realizzare quest'operazione ci sono diversi modi in SCILAB; ve ne propongo uno che utilizza la funzione **conv**:

```
--> Ncampioni=T/dt;
--> ak=zeros(1,Ncampioni*N);
--> ak(1:Ncampioni:(Ncampioni*N))=(-1).^b;
--> tak=(0:Ncampioni*N-1)*dt;
```

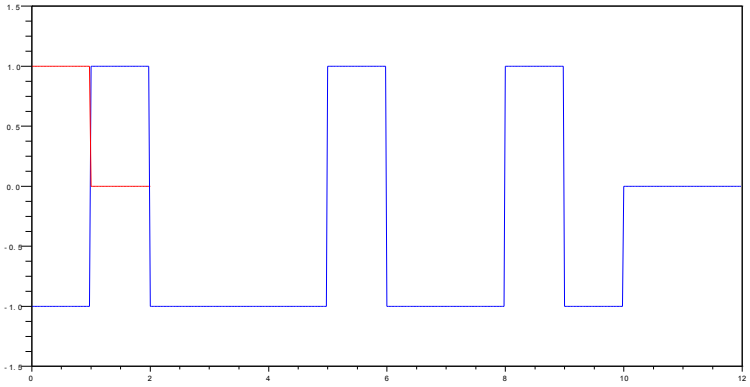
Ora dobbiamo applicare la funzione **convol** a *g* e a_k:

```
--> x=convol(ak,g);
--> t=(0:length(x)-1)*dt+tak(1)+tg(1);
```

Salviamo questo modulatore a impulsi rettangolari, ma prima controlliamo il risultato dei nostri sforzi

```
--> plot(tg,g,'r-');
--> plot(t,x,'b-');
```

```
// modulatore_rect

T=1;
dt=T/40;
tg=0:dt:2*T;
g=zeros(1,length(tg));
I=find(tg<T);
g(I)=1;

Ncampioni=T/dt;
ak=zeros(1,Ncampioni*N);
ak(1:Ncampioni:(Ncampioni*N))=(-1).^b;
tak=(0:Ncampioni*N-1)*dt;

x=convol(ak,g);
t=(0:length(x)-1)*dt+tak(1)+tg(1);
```

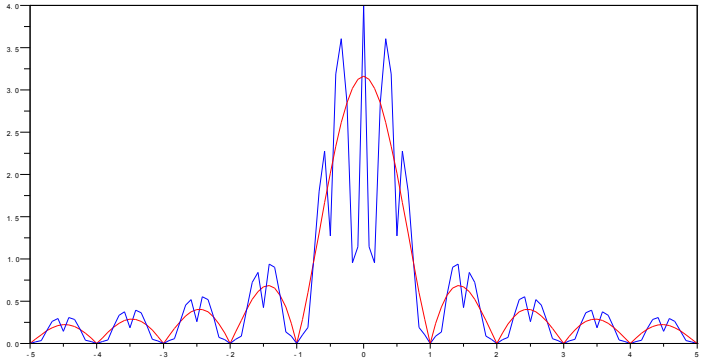


Sequenza di bit: 1 0 1 1 1 0 1 1 0 1

Ecco dunque il nostro segnale *x(t)*. Occorre osservare che fin'ora non abbiamo preso in considerazione per le nostre scelte che banda occupi il segnale *x(t)*. La banda infatti è una risorsa preziosa, ed in quasi tutti i sistemi di comunicazione costituisce un forte vincolo che può derivare da una limitazione fisica del canale (p.e. doppino telefonico) oppure dall'organizzazione della condivisone con più utenti dello stesso canale (p.e. radio, telefonia mobile). La banda occupata dipende fortemente dalla durata dell'impulso *T*, che come abbiamo già detto deve rispettare già altri vincoli, ma dipende anche dalla forma dell'impulso *g(t)*. Il rettangolo ha una trasformata (seno cardinale) con lobi laterali piuttosto alti, e quindi banda più larga rispetto ad altri segnali di pari durata ma con transizioni più dolci. La stessa cosa vale per sequenze di segnali con le stesse caratteristiche: confrontiamo la trasformata del rettangolo singolo ($G(f) = T\,sinc(fT)$) con quella del segnale *x(t):*

```
// TDF

Nx=length(x);
i0=find(t==0);
xshift=[x(i0:N) x(1:i0-1)];
Xshift=fft(xshift)*dt;
X=fftshift(Xshift);
v=1/Nx/dt;
if modulo(Nx,2)==0, // N pari
  f=(-Nx/2+(0:Nx-1))*v;
else
  f=(-(Nx-1)/2+(0:Nx-1))*v;
end
```

```
--> TDF
--> close
--> plot(f,abs(X))
--> G=T*sinc(%pi*T*f);
```



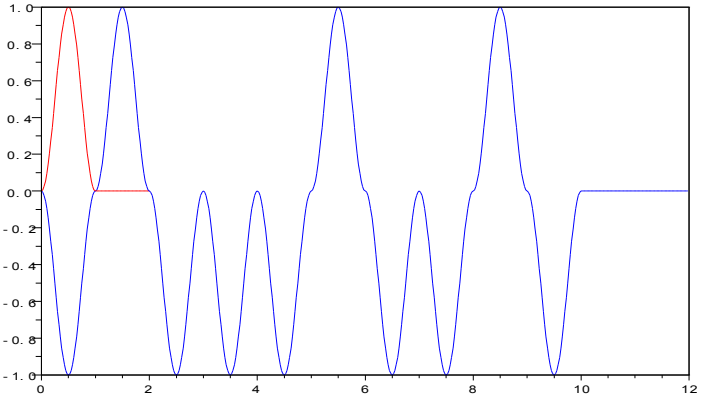
E' chiaro che *x* ha energia *N* volte più grande di *g*, per cui se vogliamo confrontarle a pari energia dobbiamo moltiplicare *G* per la radice di *N*:

```
--> plot(f,sqrt(N)*abs(G),'r--')
```

come si può notare la seconda è una versione

"frastagliata" della prima, per cui dove la prima è nulla, anche la seconda lo è: occupano la stessa banda. Allora scegliamo una forma *g(t)* più efficiente, di pari durata *T*: la caratteristica critica del rettangolo sono le transizioni ripide, per cui basta sostituirlo con una sua versione più addolcita, per esempio con una semi-onda sinusoidale al quadrato:

```
--> g=sin(%pi*tg/T).^2;
--> I=find(tg>T);
--> g(I)=0;
--> plot(tg,g,'r-')
```



Sequenza di bit: 1 0 1 1 1 0 1 1 0 1

Questa *g* ad ampiezza unitaria ha energia 3/8, si vede applicando il teorema di Parceval e integrando per parti:

$$\int_0^T \sin^4\left(\frac{\pi t}{T}\right)dt = \int_0^T \sin^3\left(\frac{\pi t}{T}\right)\sin\left(\frac{\pi t}{T}\right)dt = \int_0^T 3\sin^2\left(\frac{\pi t}{T}\right)\cos\left(\frac{\pi t}{T}\right)\cos\left(\frac{\pi t}{T}\right)dt = 3\int_0^T \sin^2\left(\frac{\pi t}{T}\right)dt - 3\int_0^T \sin^4\left(\frac{\pi t}{T}\right)dt =$$

$$= \frac{3}{2} - 3\int_0^T \sin^4\left(\frac{\pi t}{T}\right)dt$$

Quindi portando l'ultimo integrale al primo membro dell'equazione si ottiene il risultato. Verifichiamolo anche numericamente:

```
--> energia_g=sum(dt*g.^2)
```

Salviamo questo modulatore a impulsi a seno al quadrato, e vediamo allora com'è fatto il nostro nuovo segnale *x(t):*

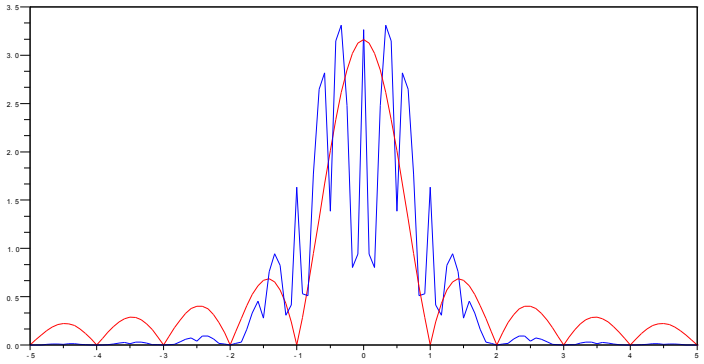
```
--> modulatore_sin2
--> plot(t,x,'b-')
```

```
// modulatore_sin2

T=1;
dt=T/40;
tg=0:dt:2*T;
g=sin(%pi*tg/T).^2;
I=find(tg>T);
g(I)=0;

Ncampioni=T/dt;
ak=zeros(1,Ncampioni*N);
ak(1:Ncampioni:(Ncampioni*N))=(-1).^b;
tak=(0:Ncampioni*N-1)*dt;

x=convol(ak,g);
t=(0:length(x)-1)*dt+tak(1)+tg(1);
```

Confrontiamo la TDF di questa nuova *x(t)* con quello vecchio. Per confrontarla con la vecchia *X* a pari energia dobbiamo dividere questa nuova *X* per la radice di 3/8:

```
--> TDF
--> plot(f,abs(X)/sqrt(3/8),'k')
```

```
--> plot(f,sqrt(N)*abs(G),'r--')
```



come si può notare l'energia del segnale x con questi nuovi impulsi è più concentrata a basse frequenze, e quindi x occupa una banda inferiore, cosa senz'altro apprezzabile.

Simulazione del canale

Ora che abbiamo il nostro segnale *x(t)* in uscita dal demodulatore occorre modellizzare il "maltrattamento" che questo subisce quando viaggia sul canale fino al ricevitore. Come detto prima, i canali possono presentare diverse non idealità che generano attenuazione, distorsione, interferenze e rumore, essenzialmente. In quest'esercitazione noi considereremo un canale affetto solo da rumore gaussiano bianco. Pur essendo un caso semplice è molto utilizzato e spesso anche interferenze da parte d'altri segnali possono essere modellizzate come un disturbo gaussiano bianco. Il canale presenta al demodulatore un segnale *y(t):*

$$y(t) = x(t) + n(t) \tag{2}$$

dove *n(t)* rappresenta un processo casuale con le seguenti caratteristiche:

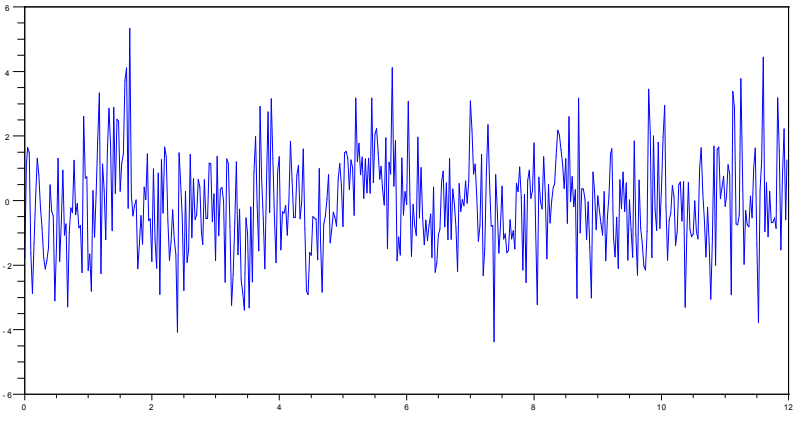
$$E[n(t)] = 0, \qquad E[n(t)n(t+\tau)] = \frac{N_0}{2}\delta(\tau), \qquad n(t)\ v.c.\ Gaussiana, \qquad \forall t \tag{3}$$

Al solito, in SCILAB, noi rappresenteremo *n(t)* tramite una sua versione campionata con lo stesso passo **dt** degli altri segnali, e quindi come un processo casuale bianco discreto. Per simulare il comportamento di un processo con densità costante pari a $N_0/2$ su tutto l'asse delle frequenze, possiamo utilizzare campioni di un processo bianco con densità $N_0/2$ nella banda -1/(2**dt**) - 1/(2**dt**). I campioni n_i, avranno allora le seguenti caratteristiche:

$$E[n_i] = 0, \qquad E[n_i^2] = \sigma^2, \qquad E[n_i\, n_{i+k}] = 0, \qquad \sigma^2 = \frac{N_0}{2\,dt}, \qquad n_i\ v.c.\ Gaussiana, \qquad \forall i \tag{4}$$

Un vettore di *N* campioni estratti da una variabile casuale gaussiana a media nulla e varianza unitaria si ottiene in SCILAB tramite **rand(1,N,'normal')**. Questi campioni sono indipendenti (e quindi incorrelati), e a valor medio nullo. Per avere la sequenza n_i che serve a noi, manca solo da imporne il valore quadratico medio a σ^2. Prima però bisogna calcolarlo.
Fissiamo il valore di densità spettrale di rumore in presenza del quale ci interessa condurre l'esperimento. Per noi, in maniera arbitraria ho scelto N_0=0.1 W/Hz.

```
--> N0=0.1;
--> sigma=sqrt(N0/2/dt);
--> n=sigma*rand(1,length(x),'normal');
```



L'effetto della trasmissione di x(t) sul canale è quello di ricevere un segnale y che è la somma di x e n:

```
--> y=x+n;
--> plot(t,y)
```

Questo è il segnale con il quale il ricevitore dovrà cimentarsi per cercare di ricostruire la sequenza di informazione b_k.

```
// canale

N0=0.1;
sigma=sqrt(N0/2/dt);
n=sigma*rand(1,length(x),'normal');
y=x+n;
```

Simulazione del ricevitore

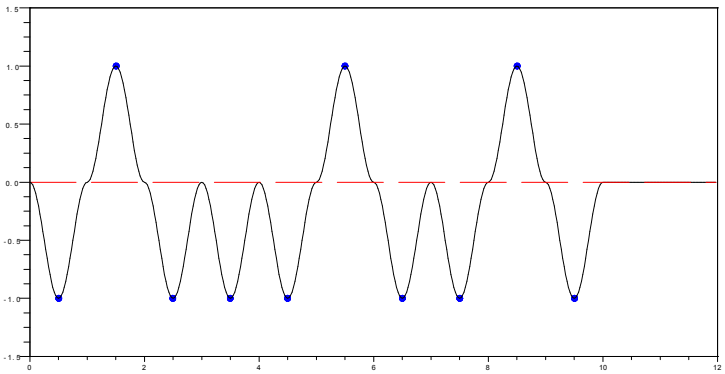
Nel ricevitore abbiamo distinto due blocchi, un demodulatore ed un decisore. Tale distinzione potrà apparire artificiosa ma intende separare due compiti ben precisi: il demodulatore deve fornire al decisore una sequenza di simboli che sarà "legata" al segnale trasmesso, senza cancellare dell'informazione contenuta nel segnale ricevuto. Sulla base di questa sequenza il decisore dovrà scegliere in base a criteri da stabilire, qual è la sequenza b_k che con maggior probabilità è stata trasmessa. Valuteremo la bontà del ricevitore in base alla sua probabilità di sbagliare decisione, ad un certo rapporto segnale-rumore.
Iniziamo dal caso di canale ideale e quindi privo di rumore. In tal caso risulta *y(t)=x(t),* e quindi le regole di demodulazione e decisione possono essere banalmente:

$$\text{demodulatore} \quad r_k = y\left(kT + T/2\right)$$

$$\text{decisore} \quad \begin{cases} \hat{b}_k = 1 & se \quad r_k < 0 \\ \hat{b}_k = 0 & altrimenti \end{cases} \tag{5}$$

Possiamo verificare che tali regole sono ottime nel senso che danno probabilità d'errore nulla:

```
--> [massimo,centro]=max(g);
--> rk=x(centro+0:Ncampioni:Ncampioni*N);
--> trk=t(centro+0:Ncampioni:Ncampioni*N);
--> plot(t,x,'k-');
--> plot(trk,rk,'b.')
```



Ed ora il decisore:

```
--> bstimati=zeros(1,length(rk));
--> bstimati(rk<0)=1;
--> err=N-sum(bstimati==b);
--> Perr=err/N;

Perr =
    0.
```

Sequenza di bit: 1 0 1 1 1 0 1 1 0 1

In un caso generico, per poter dire che la probabilità d'errore è nulla, occorrerebbe simulare la trasmissione di una sequenza ben più lunga di 10 bit, ma in questo caso si può facilmente intuire che questo ricevitore non sbaglierà mai...
Passiamo al caso con rumore, e salviamo sia demodulatore sia decisore:

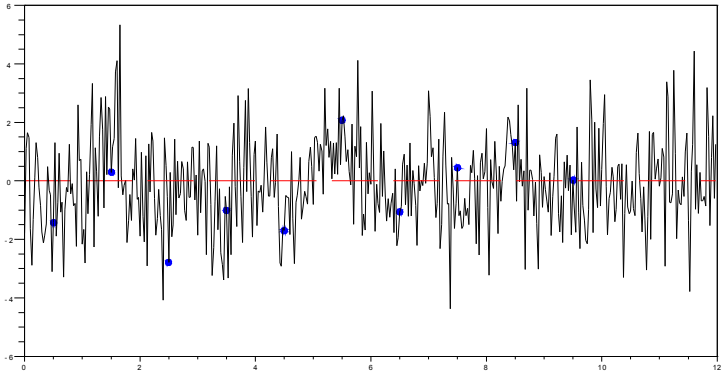
```
// demodulatore_ymaxg

[massimo,centro]=max(g);
rk=y(centro+0:Ncampioni:Ncampioni*N);
trk=t(centro+0:Ncampioni:Ncampioni*N);
```

```
// decisore

bstimati=zeros(1,length(rk));
bstimati(rk<0)=1;
err=N-sum(bstimati==b);
```

```
Perr=err/N;
```

Visualizziamo la sequenza r_k che ci restituisce ora il demodulatore.

```
--> exec('demodulatore_ymaxg.sce');
--> exec('decisore.sce')

Perr =
    0.2

--> plot(t,y,'k-');
--> plot(trk,rk,'b.');
```



Sequenza di bit: 1 0 1 1 1 0 1 1 0 1

Le prestazioni senz'altro poco soddisfacenti. Si può fare di meglio? Si può migliorare il demodulatore.

Per migliorare il demodulatore diamo un'occhiata al segnale ricevuto *y(t)* e proviamo a decidere a occhio i bit trasmessi. Un'osservazione che viene in mente abbastanza rapidamente è che sembra stupido prendere una decisione sulla base della lettura di *y(t)* in un solo istante. Se dobbiamo decidere a occhio ci viene istintivo prendere in considerazione *y(t)* in tutto l'intervallo p.e. 0-1 *s* e vedere se sta prevalentemente sopra o sotto l'asse *t*. Facendo così per esempio, rispetto ai campioni r_k evidenziati, ci verrebbe da cambiare le decisioni dell'ottavo e del decimo bit.
Guarda caso, correggeremmo gli unici due bit che ha sbagliato il ricevitore che abbiamo implementato prima. Proviamo a formalizzare la regola e ad implementare questo nuovo ricevitore: *vedere se sta prevalentemente sopra o sotto l'asse*, da un punto di vista matematico significa fare una media di *y(t)* in quell'intervallo, e poi decidere in base al segno di tale media. Teniamo allora lo stesso decisore di prima, e implementiamo un demodulatore che fornisce come sequenza r_k, la media dei campioni di *y* nell'intervallo *(k-1)T-kT*.

$$\text{demodulatore} \quad r_k = \frac{1}{N_g}\sum_{i=1}^{N_g} y\big((k-1)T + idt\big) \tag{7}$$

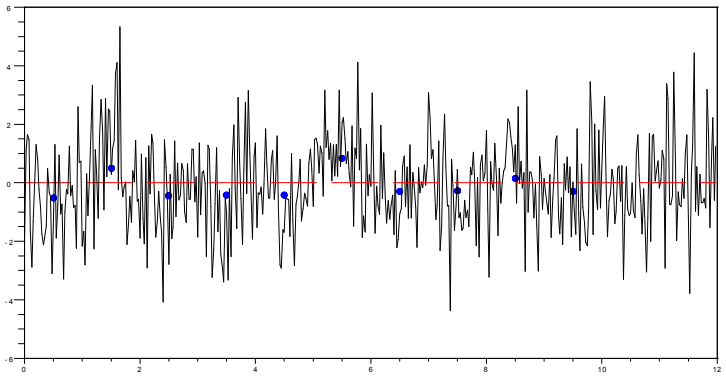
$$\text{decisore} \qquad \begin{cases} \hat{b}_k = 1 \quad se \quad r_k < 0 \\ \hat{b}_k = 0 \quad altrimenti \end{cases}$$

Notate che fare la media significa sommare i campioni e poi dividere per il numero di addendi. La divisione per un numero positivo non ne cambia il segno, quindi operativamente avremmo anche potuto semplicemente osservare il segno della sommatoria, senza calcolare la media. Eseguiamo la divisione solo per disegnare un grafico paragonabile ai precedenti.

```
--> for n=1:N,  rk(n)=sum(y([1:Ncampioni]+(n-1)*Ncampioni)); end
--> [massimo,centro]=max(g);
--> trk=t(centro+0:Ncampioni:Ncampioni*N);
```

Il decisore non dovrebbe commettere errori, verifichiamolo:

```
--> decisore
--> err, Perr
        err =     0
        Perr =    0
```



Sequenza di bit: 1 0 1 1 1 0 1 1 0 1

Ora, qual è la probabilità d'errore di questo ricevitore, con questo segnale e questa potenza di rumore? Non sarà 0 in realtà, quindi per misurarla dovremo simulare la trasmissione di un numero di bit molto più grande di 10. Iniziamo a salvare questo demodulatore:

```
// demodulatore_sumy

for n=1:N
  rk(n)=sum(y([1:Ncampioni]+(n-1)*Ncampioni));
end
[massimo,centro]=max(g);
trk=t(centro+0:Ncampioni:Ncampioni*N);
```

E ora rilanciamo tutta la simulazione, modificando il numero di bit *N* trasmessi; iniziamo con *N*=1000 bit:

```
--> clear
--> N=1000;
--> sorgente
--> modulatore_sin2
--> canale
--> demodulatore_sumy
--> decisore
--> err, Perr
        err =    12
        Perr =    0.0120
```

Ora dobbiamo chiederci quanto affidabile sia questa stima della probabilità d'errore. E' chiaro che l'affidabilità aumenta con *N*, ed è chiaro che dipende anche dalla probabilità d'errore che si deve stimare. In pratica si ottiene che essa dipende solo dal numero d'errori (indipendenti) che si conteggiano, ed una regola empirica è che tale numero deve ammontare almeno a qualche decina, meglio un centinaio, ma bisogna anche tenere in conto le possibilità della macchina. Noi ne abbiamo raccolti 12, per cui per una stima più precisa potremmo moltiplicare *N* per 5 (*N*=5000):

```
--> clear
--> N=5000;
--> sorgente
--> modulatore_sin2
--> canale
--> demodulatore_sumy
--> decisore
--> err, Perr
        err =    75
        Perr =    0.015
```

Tanto per curiosità, vediamo cosa avrebbe fatto il vecchio ricevitore, quello basato su **demodulatore_ymaxg**:

```
--> demodulatore_ymaxg
--> decisore
--> err, Perr
        err =   1136
        Perr =    0.2272
```

Possiamo quindi essere abbastanza "orgogliosi" del nostro ricevitore; eppure si può fare ancora meglio. Ridiamo un'occhiata al segnale *y*: è chiaro che la scelta del ricevitore su **demodulatore_ymaxg**, aveva una sua logica: quella di andare a leggere *y* nell'istante in cui il segnale aveva ampiezza massima, e quindi nell'istante in cui occorreva un campione di rumore più grande per far commettere un errore al decisore. Certamente sarebbe stato molto più ingenuo andare a leggere *y* in un altro istante, con *x(t)* più basso (nonostante anche questo ricevitore darebbe 0 errori su canale privo di rumore!). Il fatto che il rumore abbia valor medio zero, fa sì che il ricevitore basato su **demodulatore_sumy** abbia prestazioni migliori. Però possiamo pensare di migliorare anche quest'ultimo introducendo il principio che i campioni di *y*, nei quali la *x* assume i valori più grandi (in modulo), siano più considerati degli altri nella somma. In

pratica significherebbe fare una somma "pesata" dei campioni di *y*, nell'intervallo *(k-1)T-kT*, a seconda della forma di *g(t)*:

somma pesata con g(t).

$$\text{demodulatore} \quad r_k = \sum_{i=1}^{N_g} g(i\,dt)\, y((k-1)T + i\,dt) \qquad (8)$$

$$\text{decisore} \quad \begin{cases} \hat{b}_k = 1 & se \quad r_k < 0 \\ \hat{b}_k = 0 & altrimenti \end{cases}$$

Valutiamo le prestazioni di questo nuovo ricevitore con la nostra *y(t)*.

```
--> for n=1:N,  rk(n)=sum(y([1:Ncampioni]+(n-1)*Ncampioni).*g(1:Ncampioni)); end
--> decisore
--> err, Perr
        err =    21
        Perr =    0.0042
```

Salviamo questo nuovo demodulatore che promette molto bene. Dovremmo moltiplicare per 5 il numero di campioni, ma inizia a diventare un po' pesante per il pc e comunque il miglioramento, eseguendo più prove, sembra evidente:

```
//demodulatore_gy

for n=1:N
  rk(n)=sum(y([1:Ncampioni]+(n-
1)*Ncampioni).*g(1:Ncampioni));
end
[massimo,centro]=max(g);
trk=t(centro+0:Ncampioni:Ncampioni*N);
```

Il ricevitore *a somma pesata con g(t)* è di gran lunga il migliore tra i tre che ci sono venuti in mente. Si può dimostrare che questa coppia demodulatore-decisore costituisce il miglior ricevitore possibile per il nostro sistema di trasmissione. Le sue prestazioni (P_e) si possono calcolare anche teoricamente e vedremo che dipendono dal rapporto segnale rumore E_g/N_0 in base all'espressione:

$$P_e = Q\left(\sqrt{\frac{2E_g}{N_0}}\right) = \frac{1}{2}\,erfc\left(\sqrt{\frac{E_g}{N_0}}\right) \qquad (9)$$

Nel nostro esempio si ha:

```
--> Eg=sum(g.^2)*dt;   // 0.3750
--> Peteorica=0.5*erfc(sqrt(Eg/N0));  // N0 = 0.1

        Peteorica =    0.0031
```

L'accordo con la **Perr** ottenuta tramite simulazione è abbastanza buono, se si tiene anche conto del fatto che abbiamo raccolto meno di 30 errori.

Si noti infine che l'espressione (9) non dipende dalla forma di *g(t)*, ma solo dalla sua energia; la forma di *g(t)* può quindi essere scelta in modo da minimizzare la banda occupata senza condizionare le prestazioni. L'unica cosa che noi abbiamo imposto in entrambi gli esempi, è che le *g(t)* avessero durata *T*, per non sovrapporsi tra loro, in modo da essere certi che non diano la cosiddetta *interferenza inter-simbolica* (ISI). Si può dimostrare che questa condizione è sufficiente ma non necessaria per non avere ISI: esistono forme d'onda *g(t)* particolari che anche sovrapponendosi non danno ISI, e quindi danno le medesime prestazioni, e poiché occupano la banda minima possibile (o poco più), sono utilizzate in molti sistemi reali.

Esercizi:

1. Realizzare un modulatore a impulsi triangolari di durata T e ripetere la simulazione con il ricevitore ideale, ed un numero di bit N opportuno. Calcolare l'energia dell'impulso E_g e verificare tramite il calcolo teorico (9) la **Perr** ricavata dalla simulazione.

2. Ripetere l'esercizio 1 scalando gli impulsi triangolari di un'ampiezza opportuna in modo da riottenere E_g=3/8 come per l'impulso a seno al quadrato. Ripetere la simulazione con ricevitore ideale, e verificare che si ottiene la stessa **Perr** ottenuta a lezione con l'impulso a seno al quadrato (a meno della tolleranza dovuta all'incertezza della stima).