Principles of Digital Communication

April 30, 2021

Reading for next week : 5. [5, 6, 8]

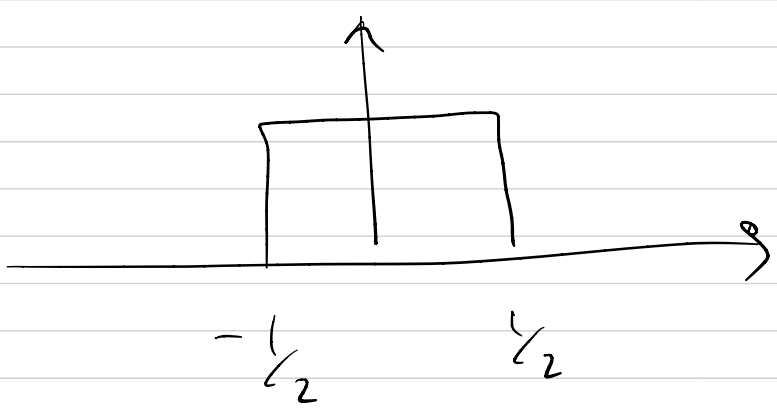
6. (1, 2, 3).

We create the sent signals

$$w_i(t) = \sum_{j=1}^{n} c_{ij}\, \psi(t-j)$$

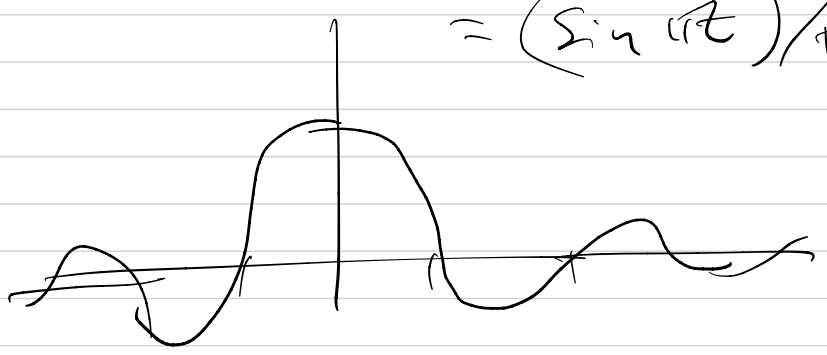
with ψ satisfying the Nyquist criterion.

Ex : $\Psi_F(f)$



$\psi(t) = \mathrm{sinc}(t)$

$= (\sin \pi t)/\pi t.$



$w(t)$ is sum :

Suppose $t \approx \frac{n}{2}$

$$w(t) = \sum_{j=1}^{n} c_j \, \mathrm{sinc}\left(t - \tfrac{n}{2} - j\right)$$

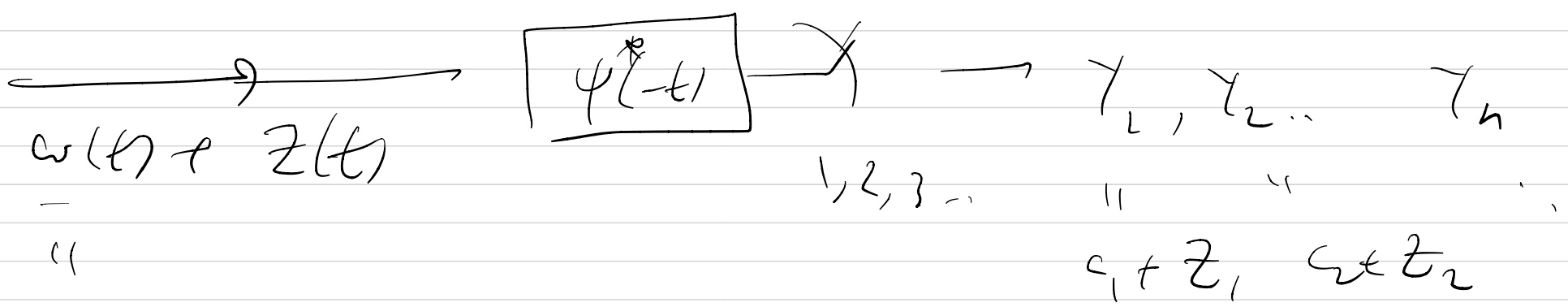
has magnitude $\approx \frac{1}{\left|\frac{n}{2} - j\right|}$

$|\mathrm{sinc}(x)| \approx \frac{1}{|x|}$

$\left|\frac{1}{x}\right|$ decays slowly in x so to approximate $w(t)$ at a particular $t = t_0$ may require summing many terms of the summation.

Had $|\psi(t)|$ decayed faster in $|t|$ then we would have needed fewer terms.

Moreover at the receiver



$$\sum_j c_j \psi(t - j)$$

output y the filter:

$$\left(w * \psi^*(-\cdot)\right)(t) + noise$$

$$= \sum_j c_j\, g(t-j)$$

$$g(t) = \psi(t) * \psi^*(-t)$$

$$g_F(f) = |\psi_F(f)|^2$$

in the case of $\psi(t) = \mathrm{sinc}(t)$ so is

$g(t)$, so when we sample the filter output at times $t = 1, 2, 3, \ldots$

the signal part gives $c_1, c_2, c_3, \ldots$

But if we made a small timing error and our samples are taken a times

$1+\delta, 2+\delta, 3+\delta, \ldots$ then the ℓ'th sample will be

$$\sum_j c_j\, \mathrm{sinc}(\ell + \delta - j)$$

$$= \frac{\sin \pi(\ell - j + \delta)}{\pi(\ell + \delta - j)} = \frac{(-1)^{\ell-j} \sin \pi\delta}{\pi(\ell + \delta - j)}$$

$$= \frac{(\sin(\pi\delta))}{(\ell + \delta - j)\pi}$$

So the ℓ'th sample

$$= c_\ell \left(\frac{\sin \pi\delta}{\pi\delta} \right) + c_{\ell-1} \frac{(-1)\sin \pi\delta}{-1+\delta}$$

$$\approx 1$$

$$+ c_{\ell+1} \frac{(-1)\sin \pi\delta}{1+\delta}$$

$$+ c_{\ell-2} \frac{\sin \pi\delta}{-2+\delta} + c_{\ell+2} \frac{\sin \pi\delta}{2+\delta}$$

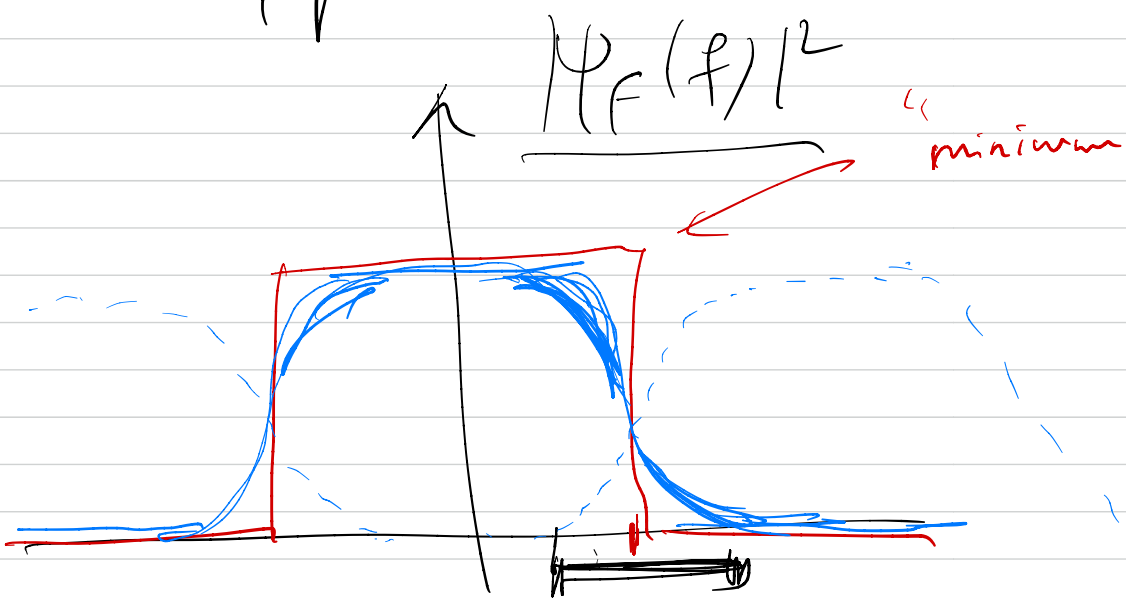
$$+ \dots$$

because $\mathrm{sinc}(\,)$ is slowly decaying the # of terms that contribute to the output is large

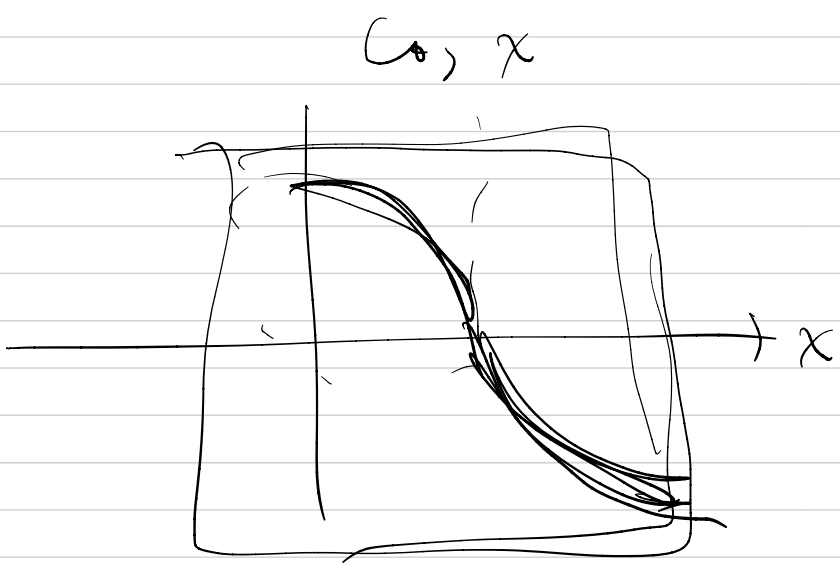
Conclusion from these considerations is that we would like a $\psi(t)$ that decays fast in $|t|$.

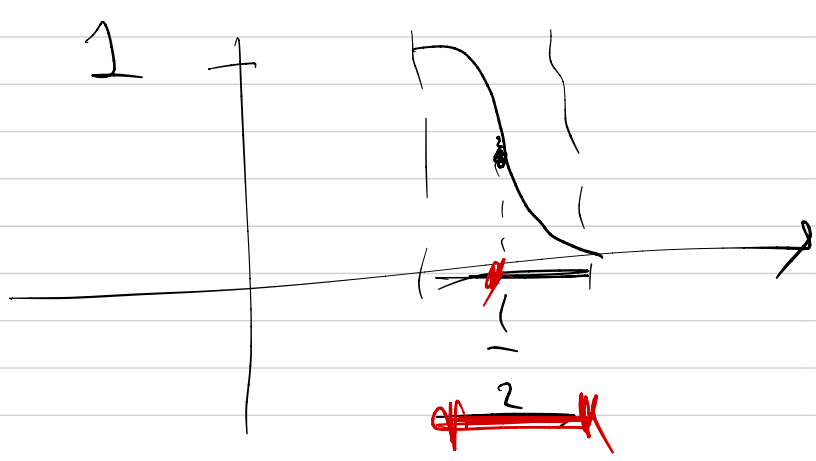
What can we do?

One popular choice is "Raised Cosine pulses".



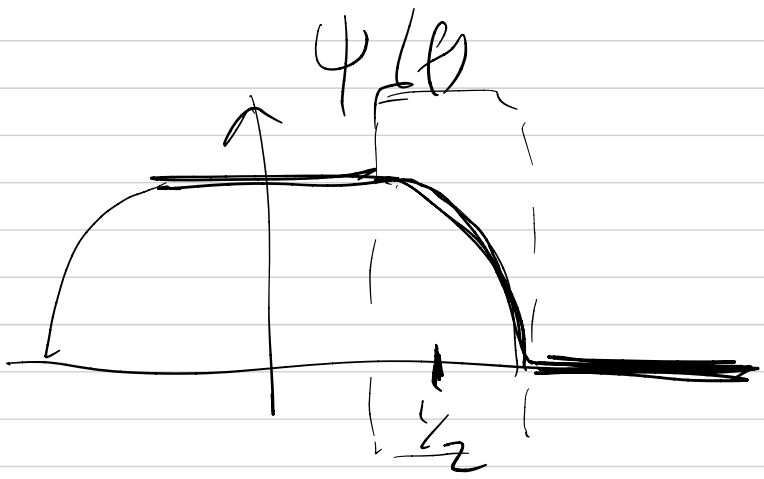
$\frac{1+\cos x}{2}$



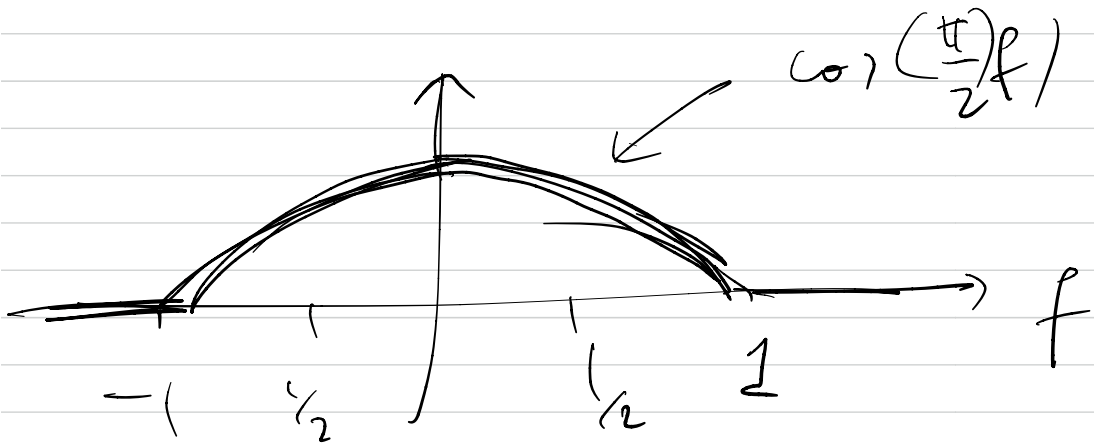


$\frac{1+\cos x}{2} = \cos^2 \frac{x}{2}$ this allows for a

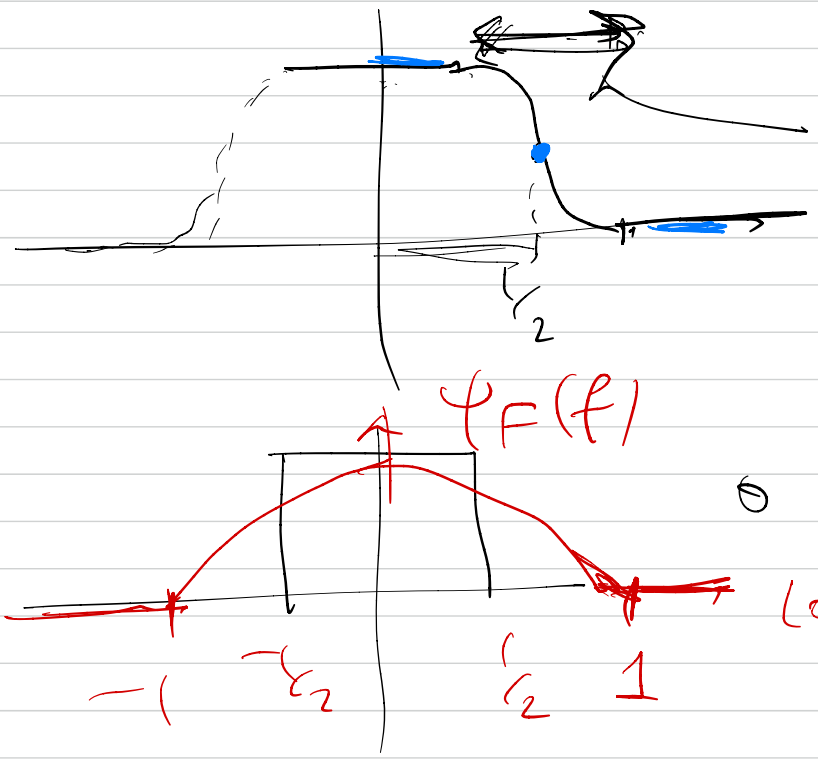
nice analytic expression for $\Psi_F(f)$



$\longleftrightarrow$ corresponds $\psi(t)$



The $\psi(t)$'s one obtain by this method are called "Root raised cosine" pulses, they are parametrized by the "excess bandwidth"



% excess bandwidth

$0 \leftrightarrow 100\%$

0 - excess

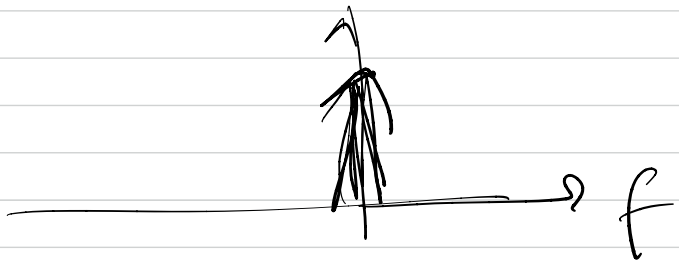
100% excess

The above discussion comprises (5.5).

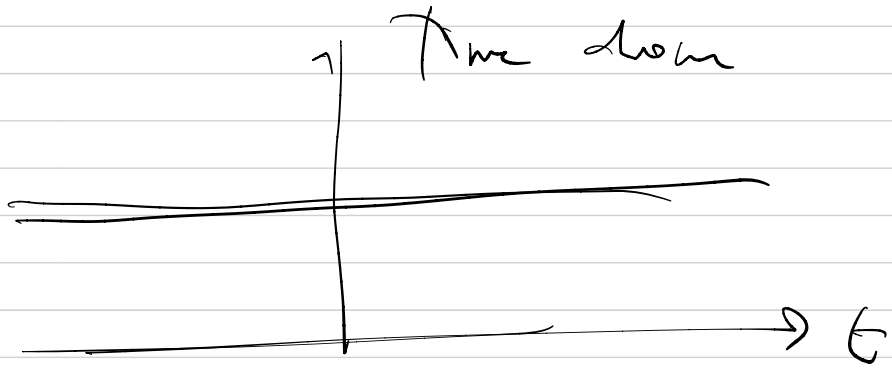
Why this particular sinous transition from 1 to 0 rather than other transitions?

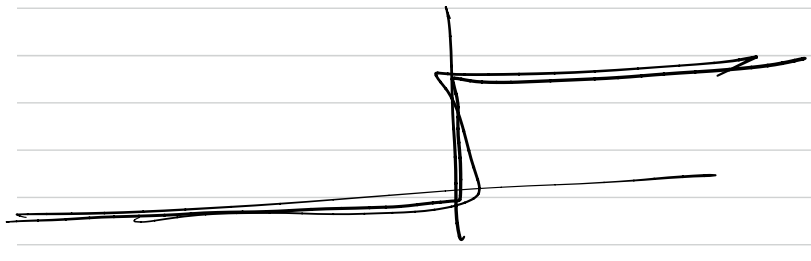
Let us examine the relationship between the Fourier transform & how fast a function decays.

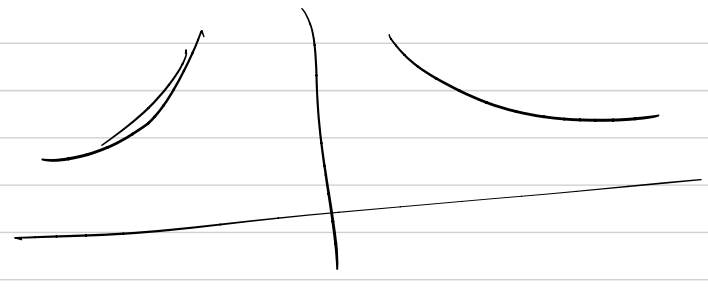
Fourier down



Time down

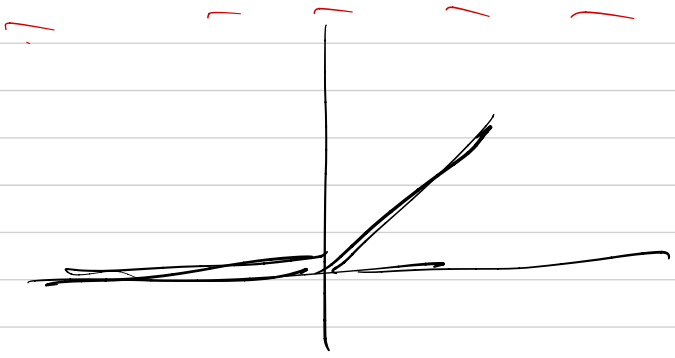


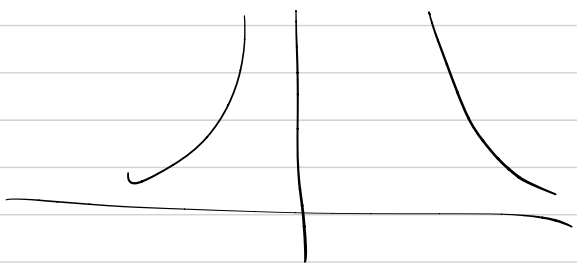




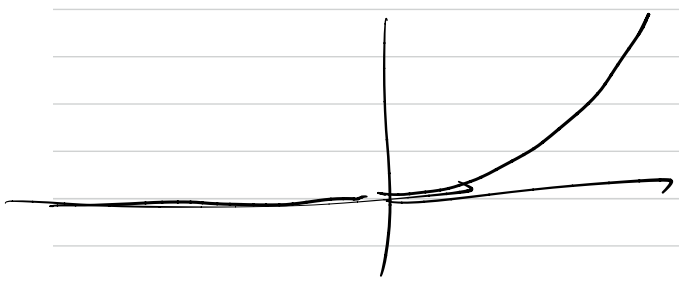
Jump discontinuity in Freq. Dom

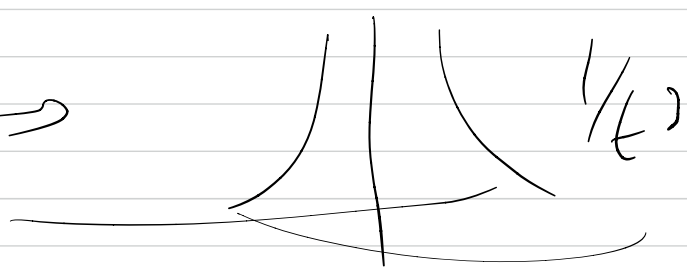
$\longleftrightarrow$ "$1/t$" behavior in time.





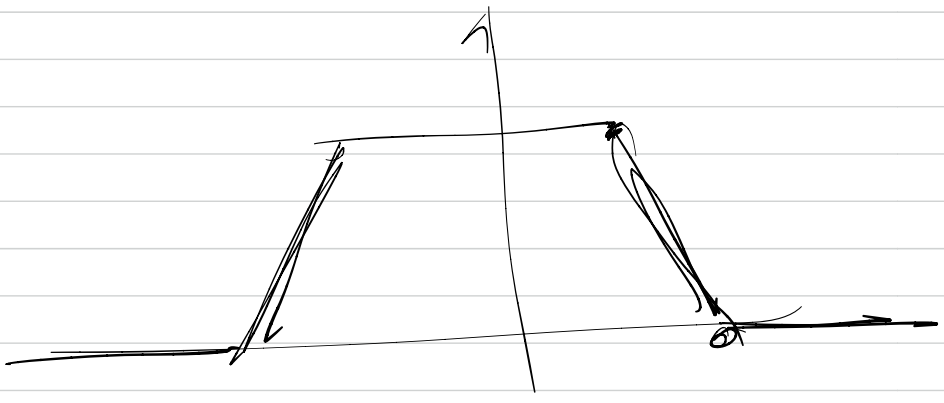
jump discont in the derivative $\longleftrightarrow$ $1/t^2$ behavior.





jump disc. in the 2nd derivative $\longleftrightarrow$ $1/t^3$ behavior.

So $|\Psi_F(f)|^2 =$



would have led to $1/t^2$ beh. in time of $g(t)$

But the "RRC family" is also continous in the derivative of $|\Psi_F(f)|^2$ so $g(t)$ will decay like $(1/t)^3$.

Chapt 6:

How to choose the signal constellation $\{c_1, c_2, \ldots, c_m\}$.

Recall: # of bits $= k = \log_2 m$

# of dim $= n =$ (Duration) $\times$ (Bandwidth).

Energy per bit $= E_b$

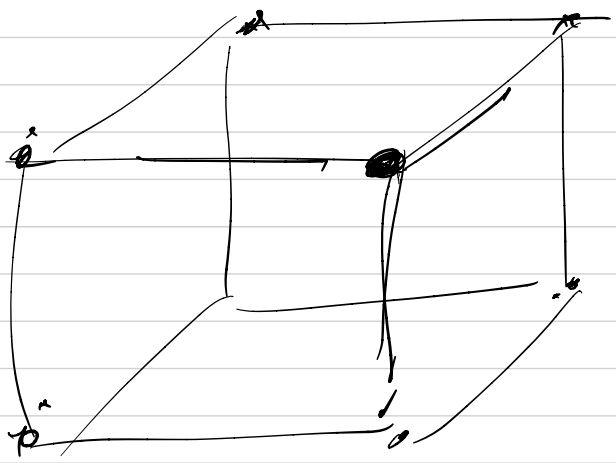
$\equiv \|c_i\|^2 \simeq k E_b = E$

$=: n E_s$

We saw that if $k/n \nearrow$ we will have large error probability; at the same time we want to use our dimensions efficiently, so we avoid $k/n \searrow 0$. Looks like we should aim for (k/n) being a constant.

E.g. $k = n$, we need $m = 2^k = 2^n$ points in n dim space., for example c_i's form $(\pm\sqrt{E_b}, \pm\sqrt{E_b} \ldots, \pm\sqrt{E_b})$.

(4.4.2).



our Eisdl constellation from the 2^n corners of an n-dimensional hypercube.

The problem with this constellation is:

$d_{min} = 2\sqrt{\mathcal{E}_b}$, and each point has

n neighbors at this distance.

Suppose $c = (\sqrt{\mathcal{E}_b}, \sqrt{\mathcal{E}_b}, \ldots, \sqrt{\mathcal{E}_b})$ is

sent. $\gamma = (\sqrt{\mathcal{E}_b} + z_1, \ldots, \sqrt{\mathcal{E}_b} + z_n)$

will be received.

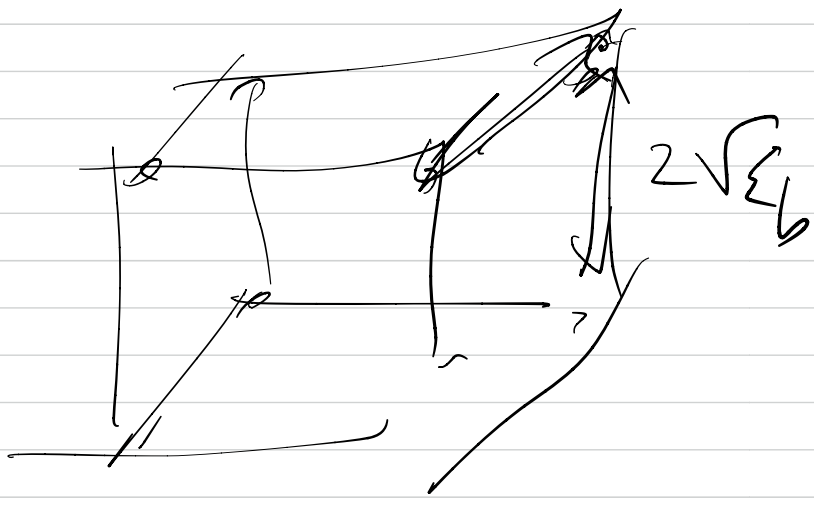
$+ \; + + \; - \; + \; + \; \ldots$

$\Rightarrow \hat{c} = (\sqrt{\mathcal{E}_b} \ldots -\sqrt{\mathcal{E}_b}, -\sqrt{\mathcal{E}_b}, \ldots)$

Since each coordinate has $Q\left(\frac{\sqrt{\mathcal{E}_b}}{\sigma}\right)$ chance of being negative, and since we have many coordinates, there is an excellent chance that an error will be made.

One way to rectify this problem: give up a little on coordinate-efficiency. Instead of $k = n$, aim for, e.g. $k = n/2$.

Instead of 2^n points in n dims we need $2^{n/2} = \sqrt{2^n}$



e.g.: $n = 20$

instead of $2^{20} = 10^6$

we need $2^{10} = 10^3$

points

This may allow us to put the chosen points far apart, and thus reduce the probability of error.

So: instead of

k bits: $(b_1 \ldots b_k) \longrightarrow (c_1 \ldots c_n)$

$n = k$

$c_i = (-1)^{b_i} \sqrt{\varepsilon_b}$

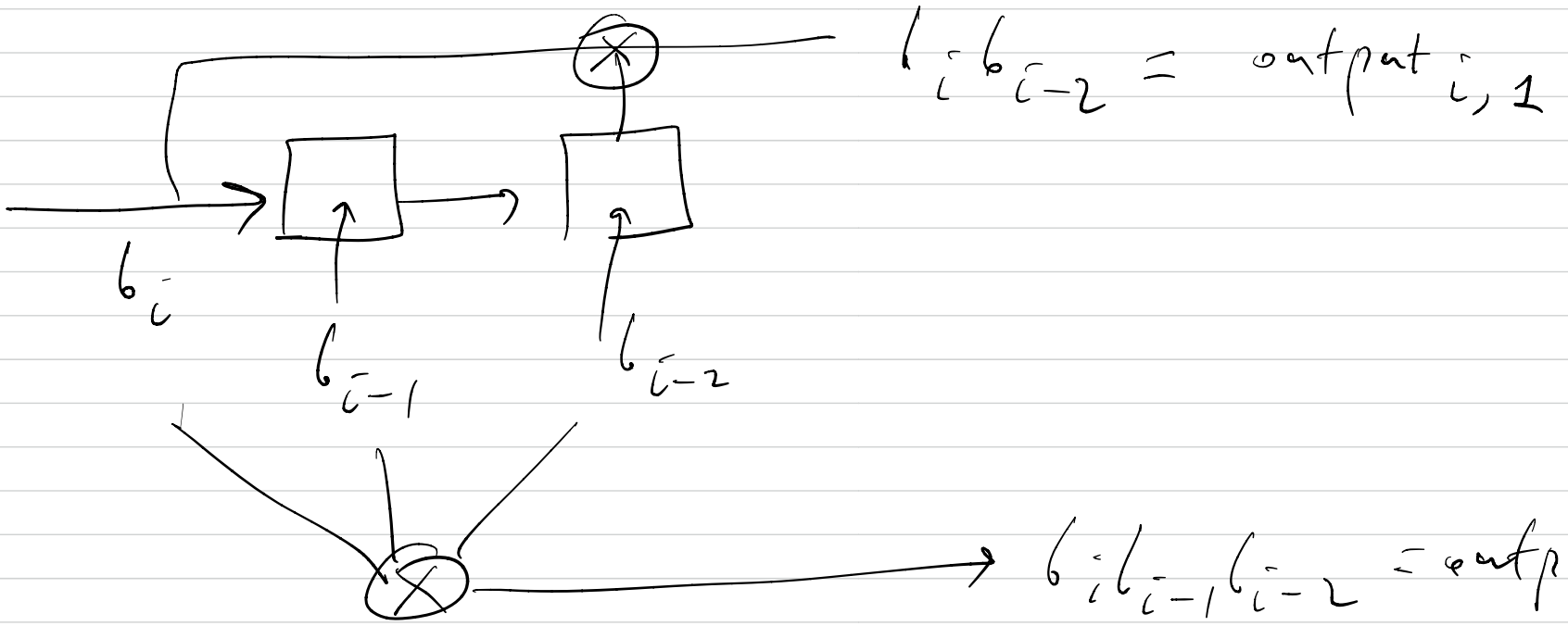
we do a more sophisticated map from

$(b_1 \ldots b_k) \rightarrow \langle\langle c_1 \ldots c_n \rangle\rangle$

$n = 2k.$

Let us represent our bits b_i by $\{+1, -1\}$ instead of $\{0,1\}$.

An example of the transformation from $b_1 b_2 \ldots b_k$ to $c_1 c_2 \ldots c_n$ is



$b_1 \ldots b_k \rightarrow (\underbrace{out_{11}\ out_{12}\ out_{21}\ out_{22} \ldots out_{k1}\ out_{k2}}_{2k})$

We see that the transformation from $\langle\langle b \rangle\rangle \rightarrow \langle\langle c \rangle\rangle$ is fairly simple: the machine does a finite # of operations per bit.

The question is how to reconstruct $(b_1 \dots b_k)$ from $y = c + z$?

Brute-force: 2^k possible b's. For each of them I compute $\|y - c(b)\|$, choose the b for which $\|y - c(b)\|$ is smallest.

Possible in principle, terrible in practice: 2^k computations to recover k bits.

However (6.3) we will see that we don't need 2^k operations to find which b is most probable if our transformation from $b \to c$ is of the kind above. The amount of computation turns out to be linear in k.