Lecture 8: Pulse Detection and Matched Filters

John M Pauly

February 23, 2026

Digital Communications: Pulse Detection and Matched Filters

In communications we have

- Signals
- Noise
- Interference

We want to detect a signal, and minimize noise and interference.

A full analysis requires

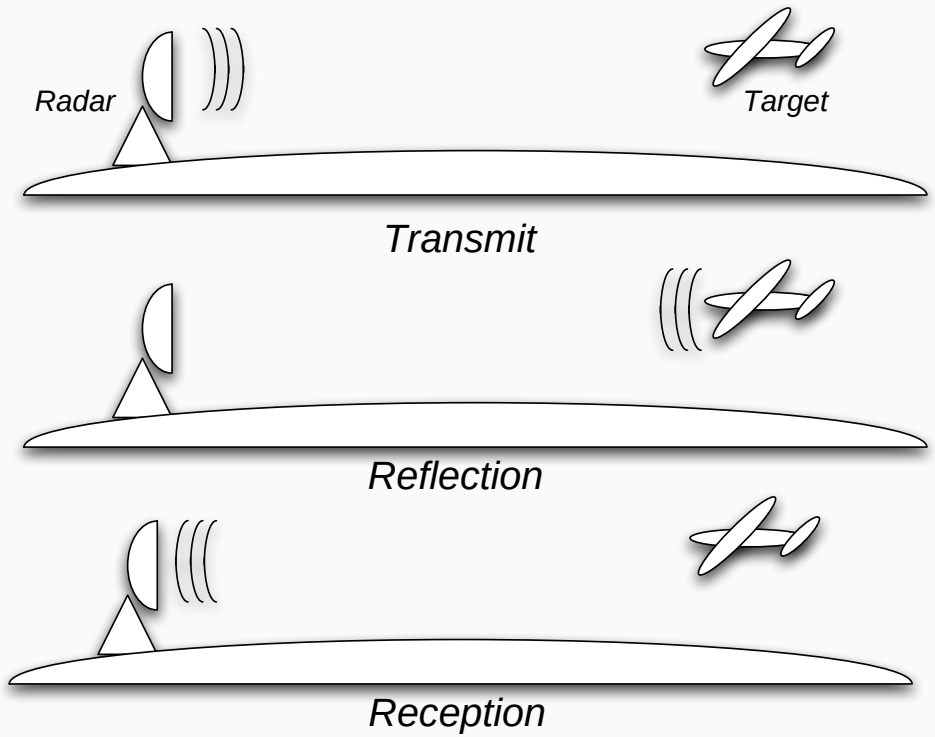
- Probability (EE 178)
- Stochastic Processes (EE 278)

For this class we will use the idea of distance and mean square error to solve this problem, and then relate this to probability.

Same result as for additive white Gaussian noise (AWGN) that is normally assumed.

Radar Problem

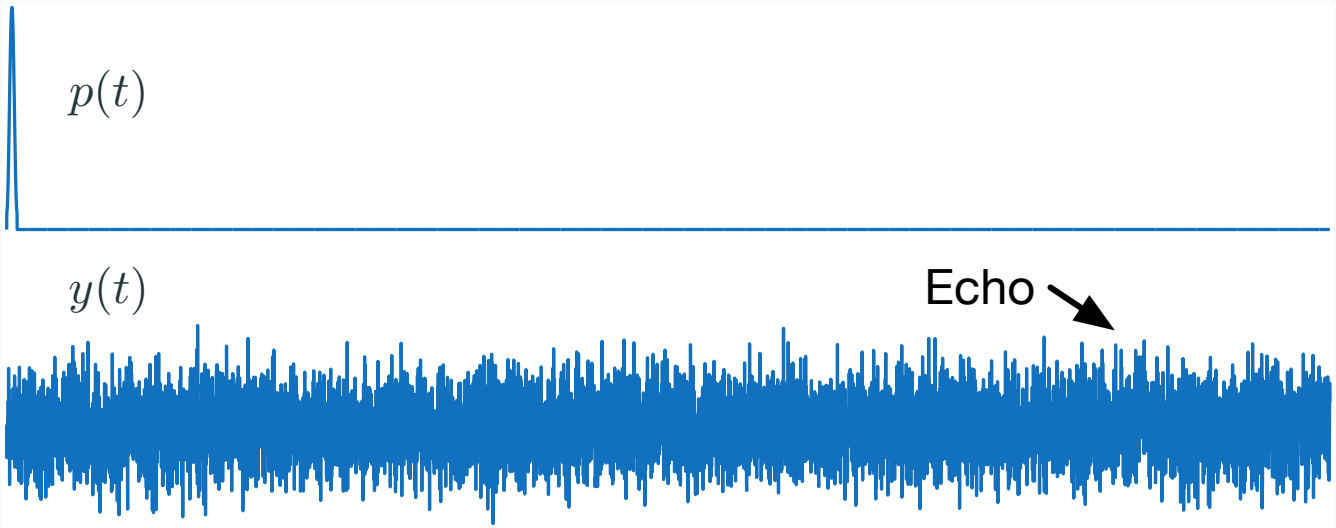
We'll start with the problem of detecting aircraft with radar, where many of these ideas first appeared.



A signal is transmitted, reflected, and then received. We want to process the signal to see if there was a reflection.

Radar Signal

The radar signal might look like this.



A pulse $p(t)$ is transmitted, and a signal $y(t)$ is received, along with additional noise.

In this case, it is difficult to tell if there really is a return!

How do I tell if there is a signal there?

Matched Filter Idea

If there was a return at some time t_r, what would it's amplitude be?

We do a least-squares fit at each time. If this is bigger than what we'd expect from noise, we have an echo.

To find the amplitude a we find the value that minimizes the squared distance between the transmit pulse and the received signal.

$$Q = \int_{-\infty}^{\infty} \left(y(t) - ap(t - t_r)\right)^2 dt$$

where we've assumed all of the signals are real valued. We can find a by differentiating Q with respect to a, and setting the result to zero. In your homework you will show that the solution is

$$a = \frac{\int_{-\infty}^{\infty} y(t)p(t - t_r)dt}{\int_{-\infty}^{\infty} p^2(t)dt}$$

Matched Filter Idea (cont.)

If we choose $p(t)$ to be normalized,

$$\int_{-\infty}^{\infty} p^2(t)dt = 1$$

then

$$a(t_r) = \int_{-\infty}^{\infty} y(t)p(t - t_r)dt$$

where we've explicitly noted that this amplitude is for a time t_r.

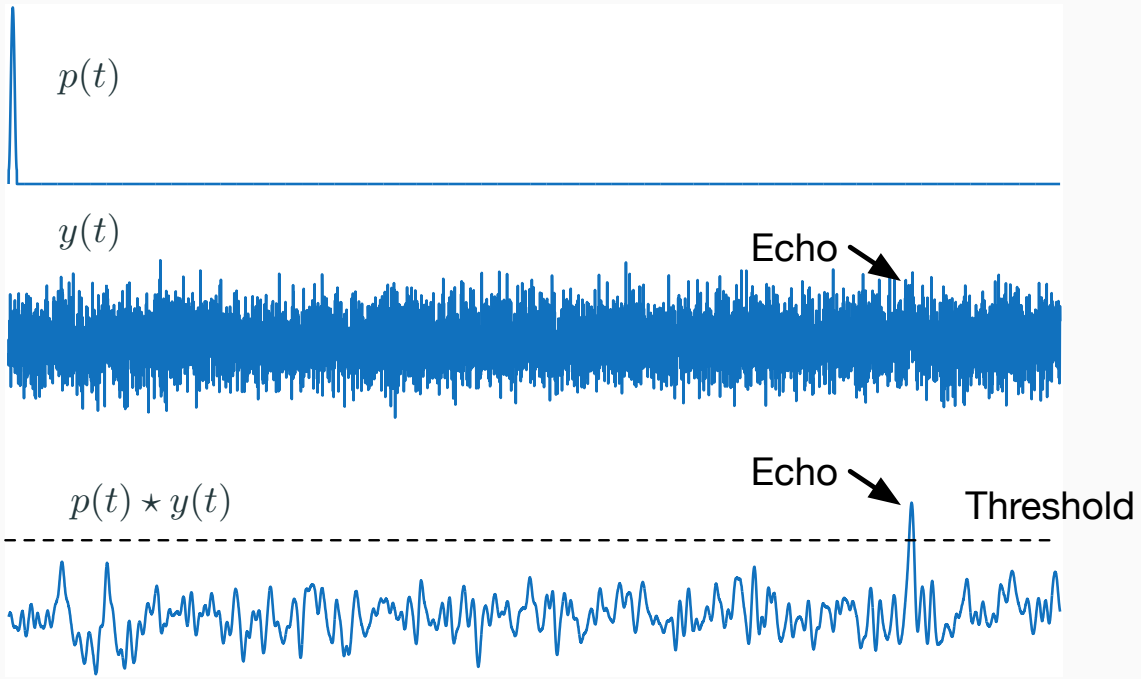
This operation is a *cross correlation*. It looks a lot like a convolution, except neither signal is reversed.

Basic idea:

- For each t_r, shift $p(t)$ to the right
- Multiply point-by-point with $y(t)$
- Integrate the result.

Matched Filter Idea (cont.)

For the previous example, the result looks like this:



If the cross correlation is larger than the threshold we'd expect from noise, we declare a detected echo. We'll get back to setting that threshold in a bit.

This is called a *matched filter* because the transmit pulse and the received echo are matched (the same shape). This is optimal for AWGN.

Cross Correlation

Cross correlation is an important operation that has it's own notation, It can be written either as

$$\begin{aligned} R_{xy}(\tau) &= \int_{=\infty}^{\infty} x(t-\tau)y(t)dt \\ &= \int_{=\infty}^{\infty} x(t)y(t+\tau)dt \end{aligned}$$

In the first we delay $x(t)$ by τ. In the second we advance $y(t)$ by τ. Either way only the difference in time matters.

Common notation is

$$R_{xy}(t) = x(t) \star y(t)$$

similar to that for convolution. In your homework you will show that

$$R_{xy}(t) = R_{yx}(-t)$$

Correlation and Convolution

Correlation and convolution are closely related.

In your homework you will show that

$$x(t) \star y(t) = x(-t) * y(t)$$

This means we can perform a correlation by first reversing the first signal, and then doing a convolution.

This is very frequently done. Efficient implementations of convolution are quite common, while implementations of correlation are less often available.

Cross Correlation: Complex Signals

In general, our signals will be complex. The same derivation holds, with the result that

$$a(t_r) = \int_{-\infty}^{\infty} p^*(t - t_r) y(t) dt$$

The conjugate cancels the phase of the return in $y(t)$. Without this the phases would add, and the result will often be small.

You will also derive this result in the homework.

Matlab's `corr()` includes the conjugate automatically.

If the two signals are the same, this gives the *autocorrelation* which we will see frequently.

$$\begin{aligned} R_{xx}(\tau) &= \int_{=\infty}^{\infty} x^*(t - \tau) x(t) dt \\ &= \int_{=\infty}^{\infty} x^*(t) x(t + \tau) dt \end{aligned}$$

Correlation Spectrum

The spectrum of the cross correlation is

$$\mathcal{F}\{R_{xy}(t)\} = S_{xy}(f) = X^*(f)Y(f)$$

This is call the *cross-power spectrum*, and has two subscripts.

The spectrum of the autocorrelation is

$$\mathcal{F}\{R_{xx}(t)\} = S_x(f) = |X(f)|^2$$

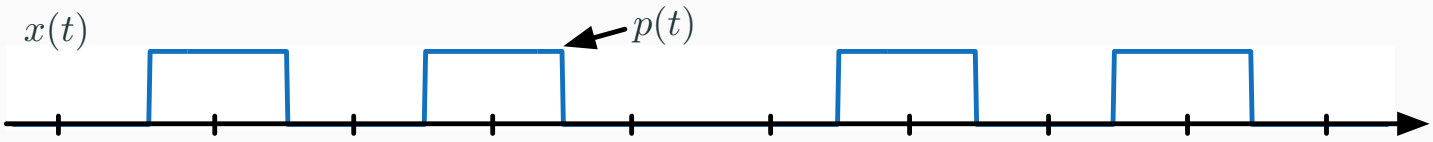
$S_x(f)$ is called the *power spectrum* of $x(t)$, and will be useful for characterizing the bandwidth of a signal transmitted over a channel. It only has one subscript.

Communications and Matched Filters

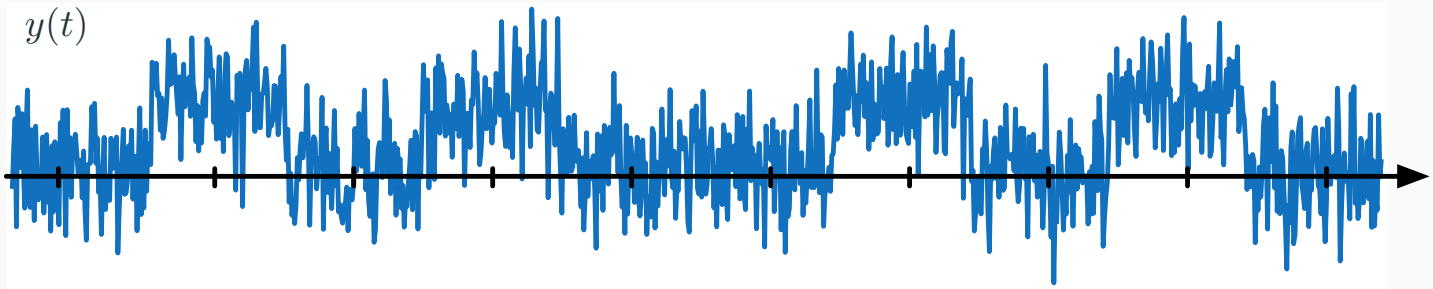
In radar, we don't know if there is an echo, or where it is.

In communications, we know where the symbols are (more on that later)

We want to know *which* symbol was sent at each time. If the originally transmit



we might receive this



How do we determine what sequence of "1's and "0"'s was transmitted?

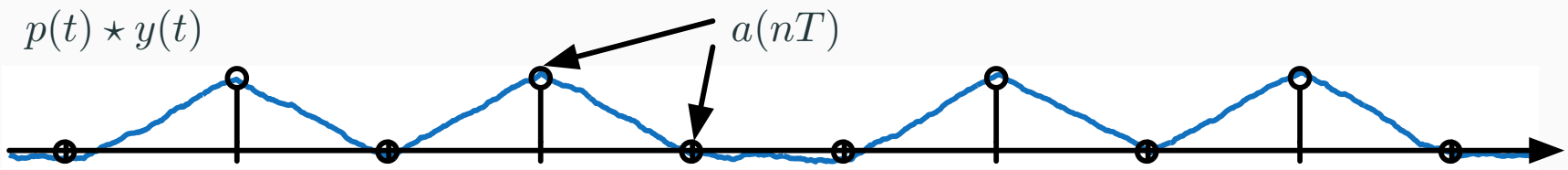
Communications and Matched Filters (cont.)

Assume the timing is synchronized with the symbols. If the symbols are spaced by T, they are centered at $y(nT)$.

For each symbol we estimate its amplitude with

$$a(nT) = \int_{-\infty}^{\infty} p^*(t - nT)y(t)dt$$

The result is



The cross correlation suppresses much of the noise. The original bits can be detected with a simple threshold.

Communications and Matched Filters (cont.)

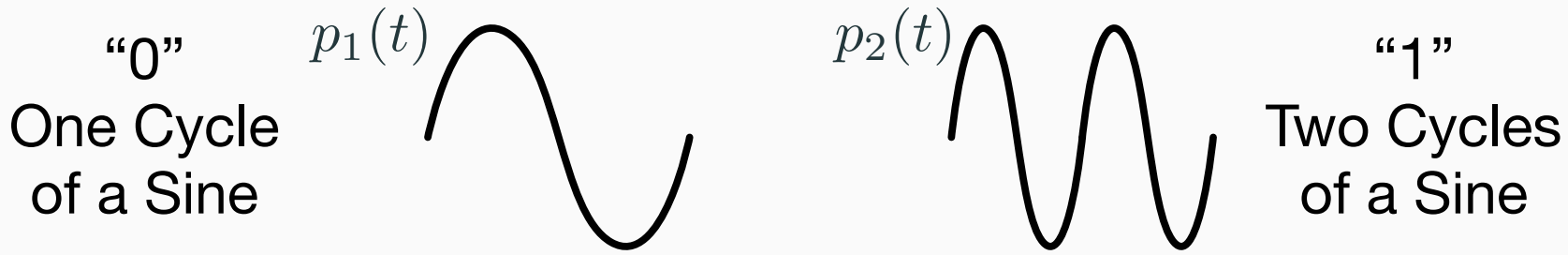
If A is the value of the signal for a "1", we decide on whether we received a "1" or a "0" by

$$d[n] = \begin{cases} 1 & a(nT) > \frac{A}{2} \\ 0 & a(nT) < \frac{A}{2} \end{cases}$$

We pick the closest answer for the decoded bit $d[n]$. This is optimal for AWGN.

Frequency Shift Keying Example

Another example is frequency shift keying or FSK. Here we transmit sinusoids at two different frequencies for the two different bits.



We send one cycle of a sine for a "0", and two cycles for a "1".

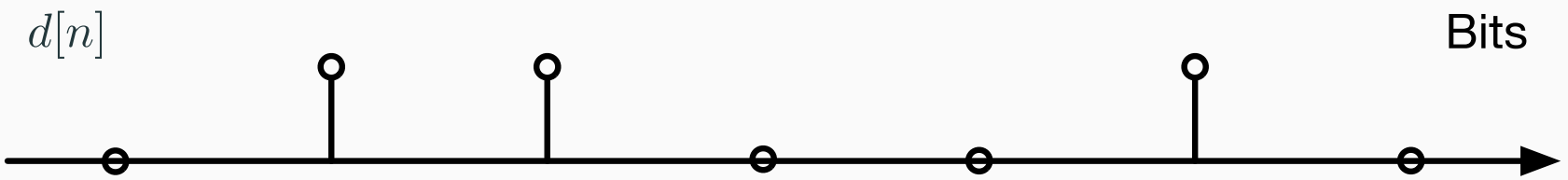
Note that

$$\int_{-\infty}^{\infty} p_1(t)p_2(t)dt = 0$$

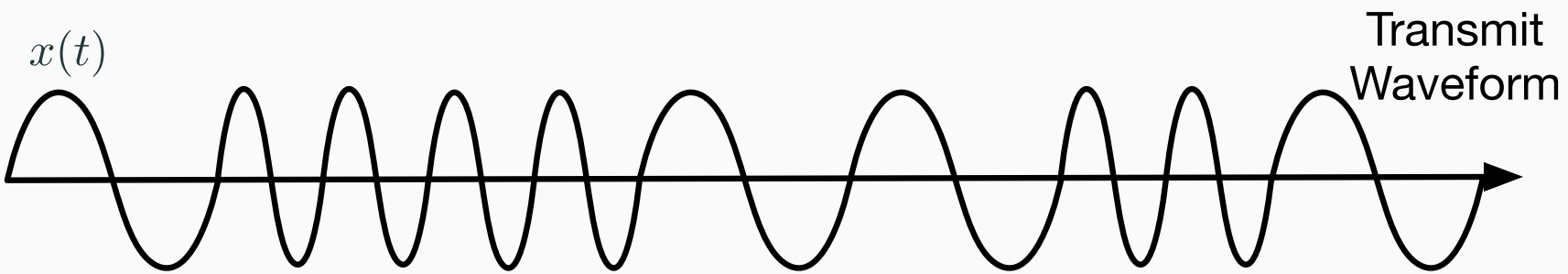
The two symbols are orthogonal, so that when we detect one, we should have a zero response for the other. We will often use this.

Frequency Shift Keying Example (cont.)

If we want to send a sequence of bits $d[n]$,



The waveform we would send is



How do we detect the two symbols?

Frequency Shift Keying Example (cont.)

We use a matched filter for each symbol.

In this case we compute two amplitudes using cross correlation,

$$\begin{aligned} a_1(nT) &= \int_{-\infty}^{\infty} p_1(t-nT)y(t)dt \\ a_2(nT) &= \int_{-\infty}^{\infty} p_2(t-nT)y(t)dt \end{aligned}$$

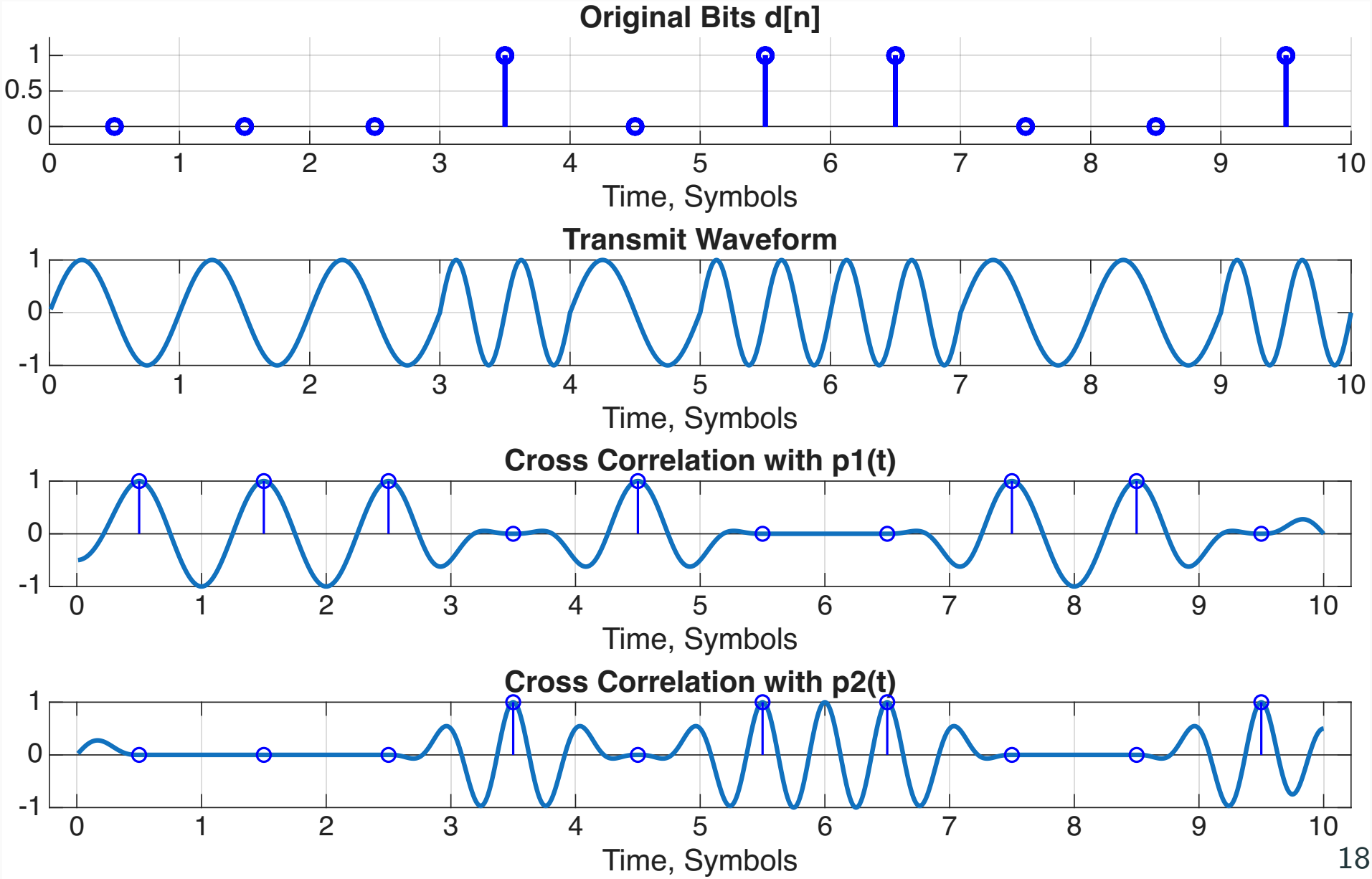
At each symbol time, one of the two should be "1" and the other "0"

The detection rule is which symbol is closer to the received waveform

$$d[n] = \begin{cases} 1 & a_2(nT) > a_1(nT) \\ 0 & a_2(nT) < a_1(nT) \end{cases}$$

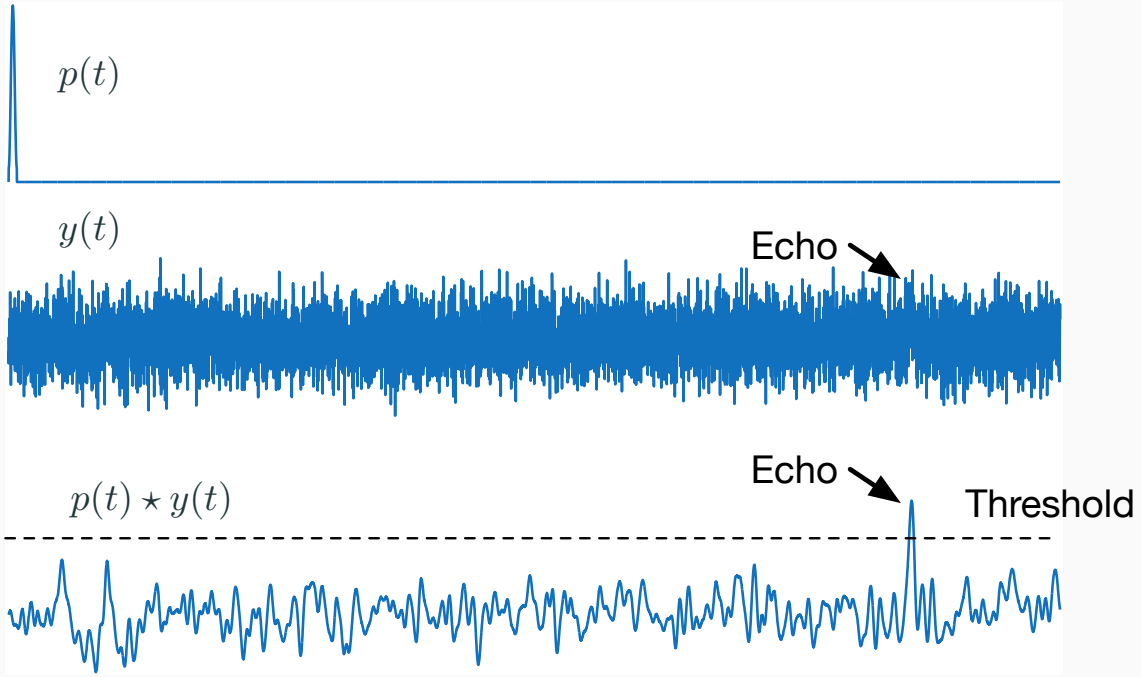
We could also just decide based on either one of the two, since each is sufficient. That has worse noise performance.

Frequency Shift Keying Example (cont.)



Radar, and Setting the Detection Threshold

Recall that after matched filtering our signal looked like this:



How do we choose the threshold?

- Too small: we detect noise and get false alarms
- Too high: we miss too many real targets

Hypothesis Testing

At each time t_r we want to decide whether there is a pulse present

- H0: $y(t) = n(t)$
- H1: $y(t) = p(t - t_r) + n(t)$

where $n(t) \sim N(0, \sigma^2)$ is white Gaussian noise. Then

H0:

$$\hat{a}(t_r) = \int_{-\infty}^{\infty} n(t)p(t - t_r)dt = n_p$$

H1:

$$\begin{aligned} \hat{a}(t_r) &= \int_{-\infty}^{\infty} \left(ap(t - t_r) + n(t)\right) p(t - t_r)dt \\ &= \int_{-\infty}^{\infty} ap^2(t - t_r)dt + \int_{-\infty}^{\infty} n(t)p(t - t_r)dt = a + n_p \end{aligned}$$

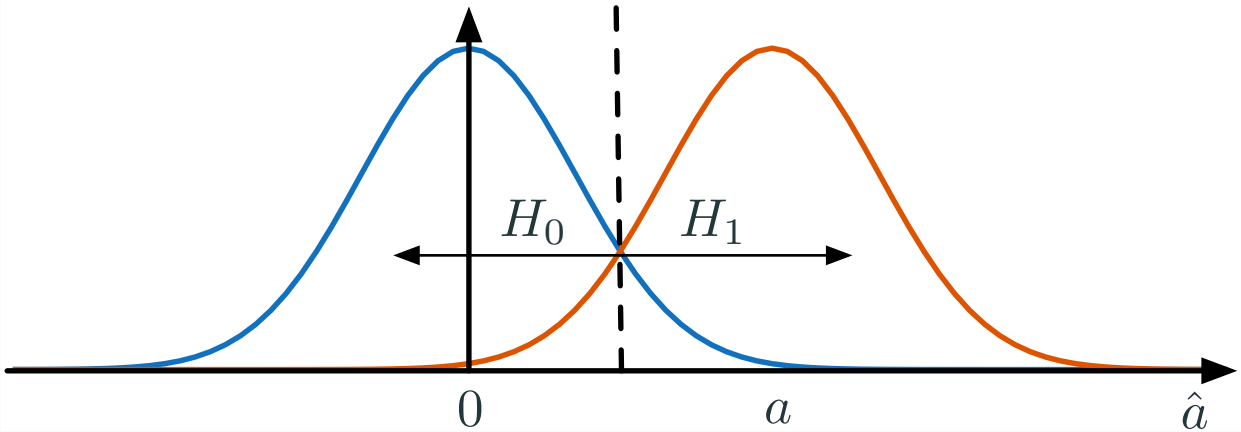
where $n_p \sim N(0, \sigma_p^2)$, the reduced noise after the cross correlation. How do we choose one of these?

Maximum Likelihood

For a given $\hat{a}$, choose the most likely:

$$p(\hat{a}|0) = \frac{1}{\sqrt{2\pi\sigma_p^2}} e^{-\frac{\hat{a}^2}{2\sigma_p^2}}$$

$$p(\hat{a}|a) = \frac{1}{\sqrt{2\pi\sigma_p^2}} e^{-\frac{(\hat{a}-a)^2}{2\sigma_p^2}}$$

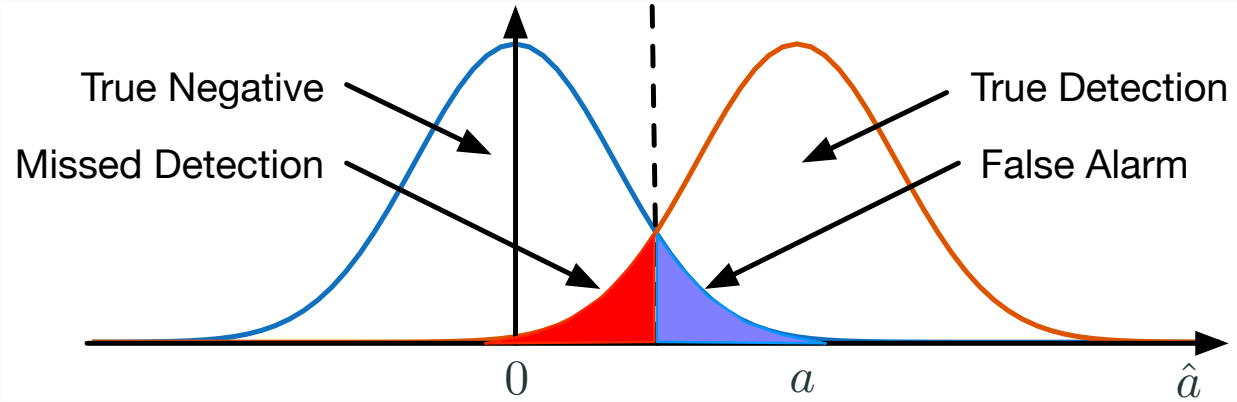


Here $\hat{a}$ is the estimate of the pulse height, corrupted by noise. We assume a is known, and are just trying to decide if the pulse is present or not.

False Alarms and Missed Detections

When we choose a threshold we can make two types of errors

- False Alarm: We say there is a pulse, when it is really noise
- Missed Detection: We say it is noise when a pulse is actually present



Often in communications we care equally about both (is the bit a 1 or a 0?), and just set the threshold in the middle.

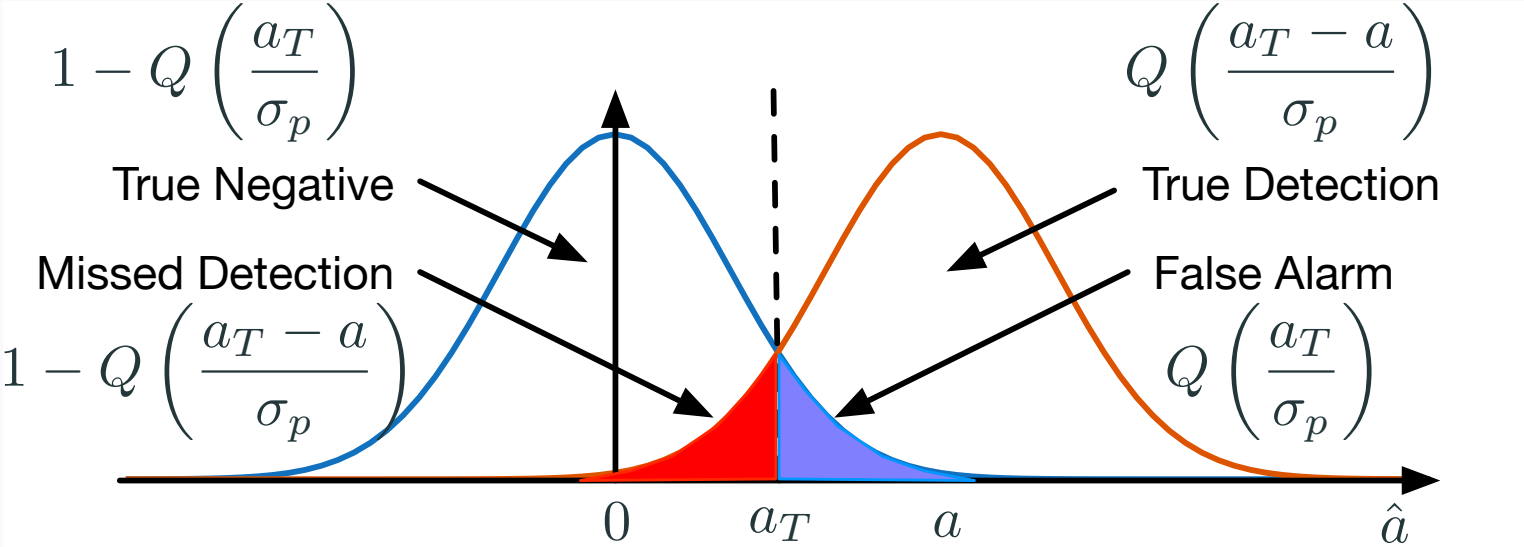
Other times one outcome is more important than the other. In radar, we want to limit false alarms so that the operator isn't overwhelmed, while still making sure we see true targets.

Q Function

The probability of a false alarm is just the integral of the tail of the Gaussian. We'll define this for a normalized Gaussian as:

$$Q(x) = \frac{1}{\sqrt{2\pi}} \int_x^\infty e^{-t^2/2} dt$$

Then the probabilities of both errors, and well as true detections, are

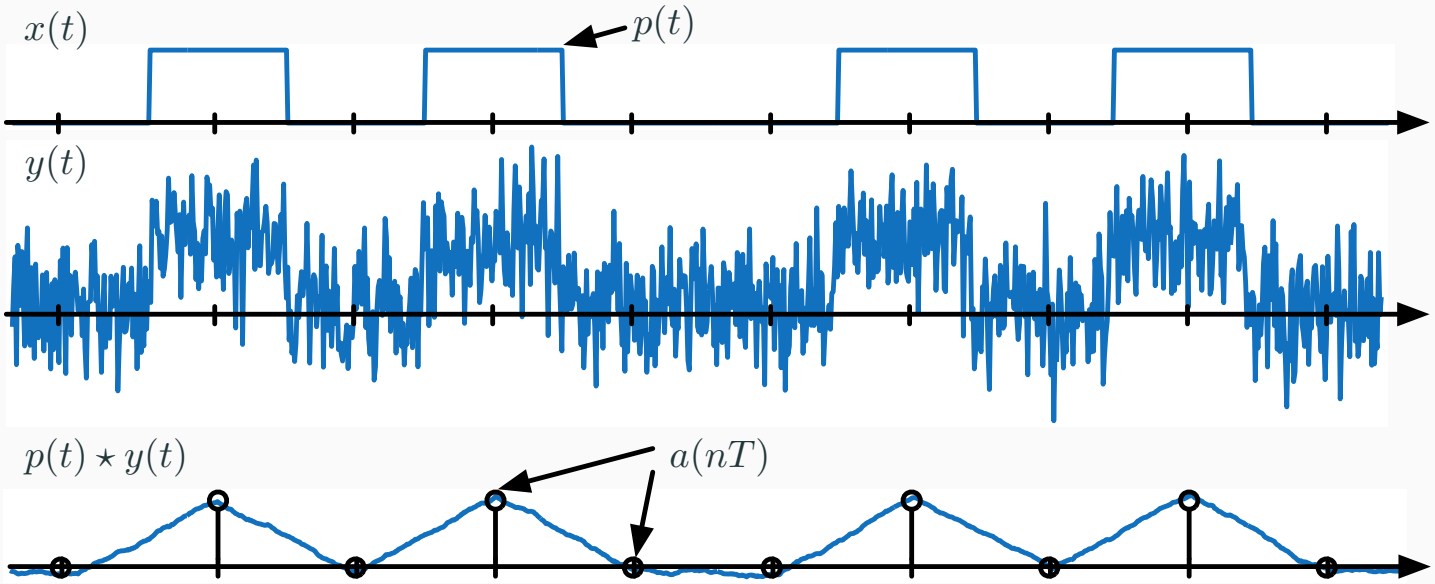


a_T is the decision threshold level. Dividing by σ_p normalizes the Gaussian.

Probability of Error for On-Off Keying

For this class we're more concerned about communications, and errors in decoding symbols.

Earlier we talked about on-off keying, and using a matched filter.



After cross correlation, we need to decide at each sample whether there was a "1" or a "0". Note the much reduced noise after the cross correlation, $\sigma_p << \sigma$.

Computing Q

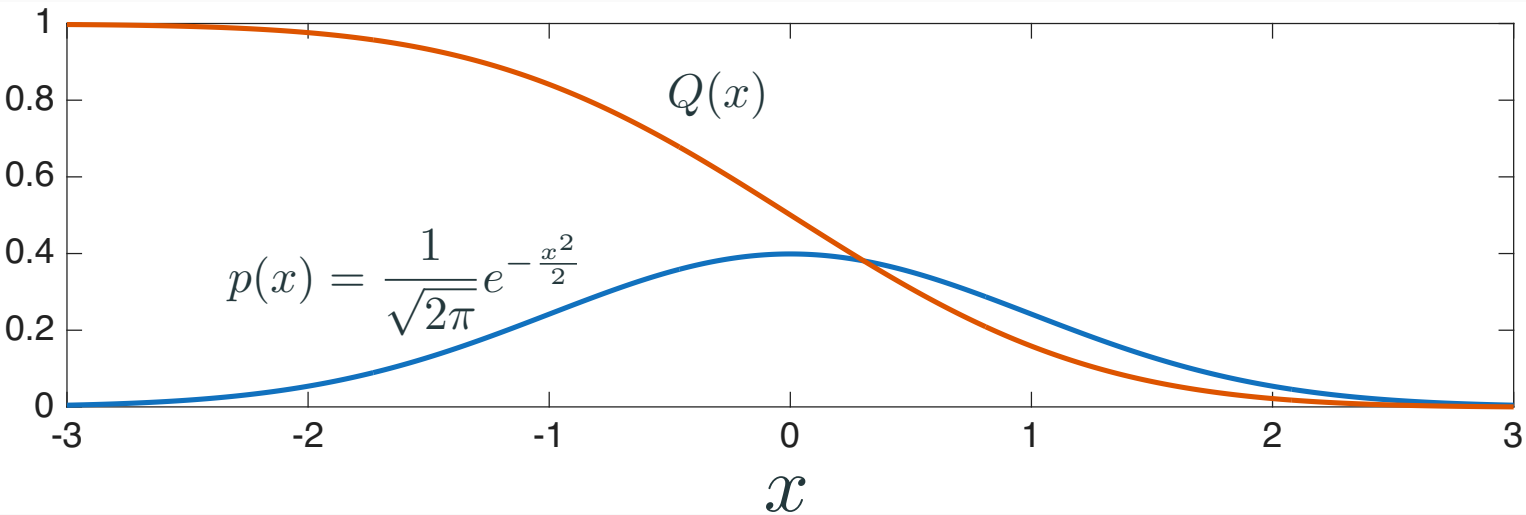
The Q function is in the communications toolbox at `qfunc()`.

It is closely related to

- the error function `erf()`
- and the complimentary error function `erfc()`

that you are already probably familiar with.

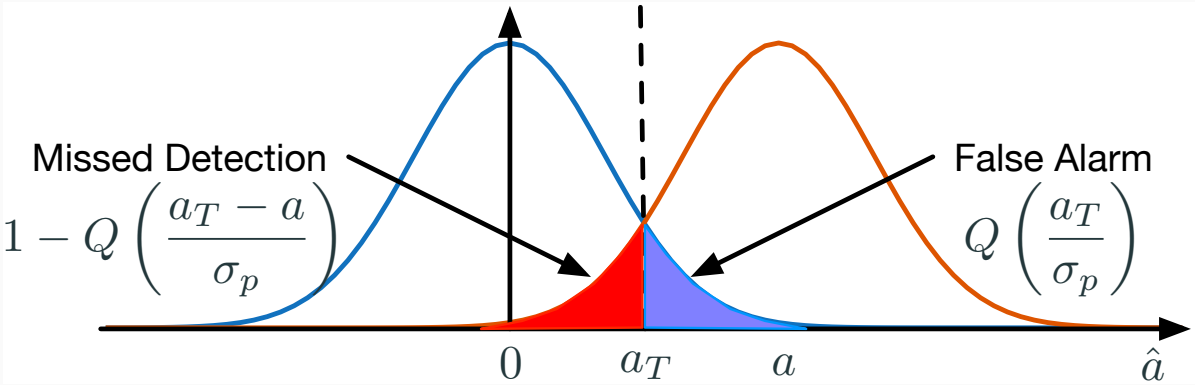
The Gaussian $p(x)$ and Q function look like this:



Error Rate on On-Off Keying

An error is either due to

- Detecting a 1, when a 0 was sent
- Detecting a 0, when a 1 was sent



So the error for a signal bit is

$$P_b = Q\left(\frac{a_t}{\sigma_p}\right)$$

since Q is symmetric, and the errors have the same probabilities. We have two events, each with a probability of 1/2.

Error Rate on On-Off Keying, SNR

The SNR is

$$SNR = \frac{a^2}{\sigma_p^2}$$

where $a = 2a_t$, twice the decision threshold. The probability of error is then

$$P_b = Q\left(\frac{a_t}{\sigma_p}\right) = Q\left(\frac{a/2}{\sigma_p}\right) = Q\left(\frac{1}{2}\sqrt{SNR}\right)$$

The probability that we have no errors in a packet with N bits is

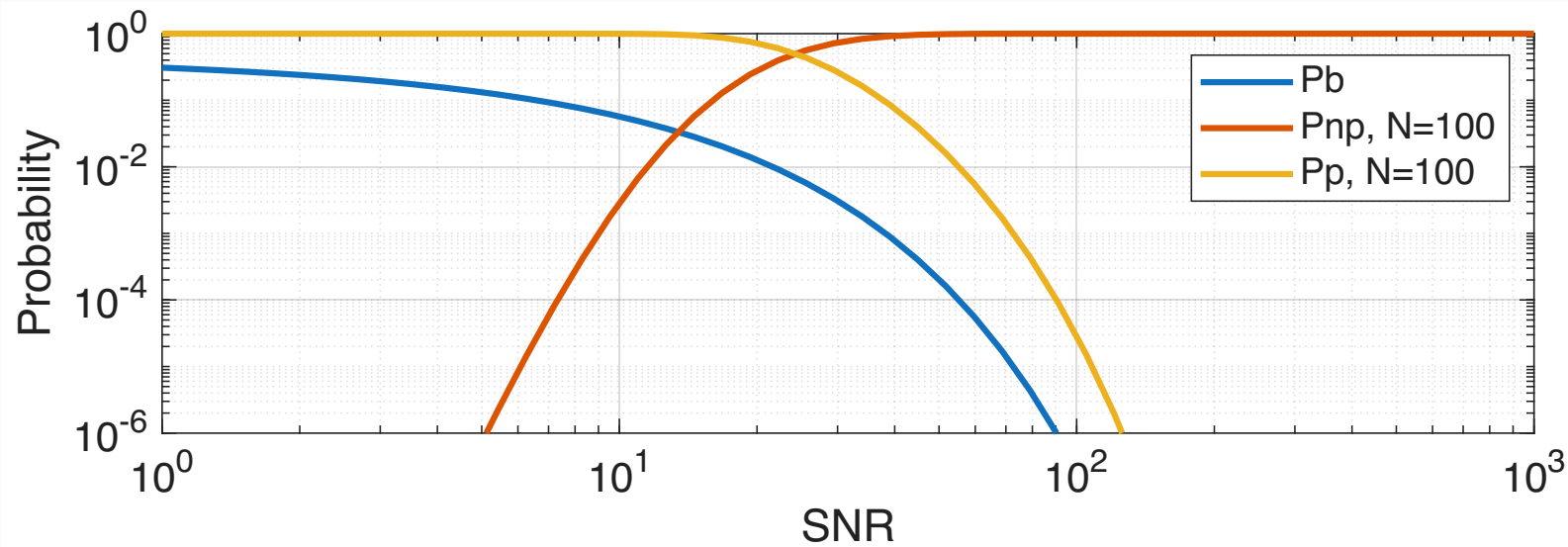
$$P_{np} = (1 - P_b)^N$$

and the probability of a packet with errors is

$$P_p = 1 - P_{np} = 1 - (1 - P_b)^N$$

Error Rate on On-Off Keying, SNR

The probability of a bit error P_b and packet error P_p as a function of SNR looks like this for $N = 100$ length packet

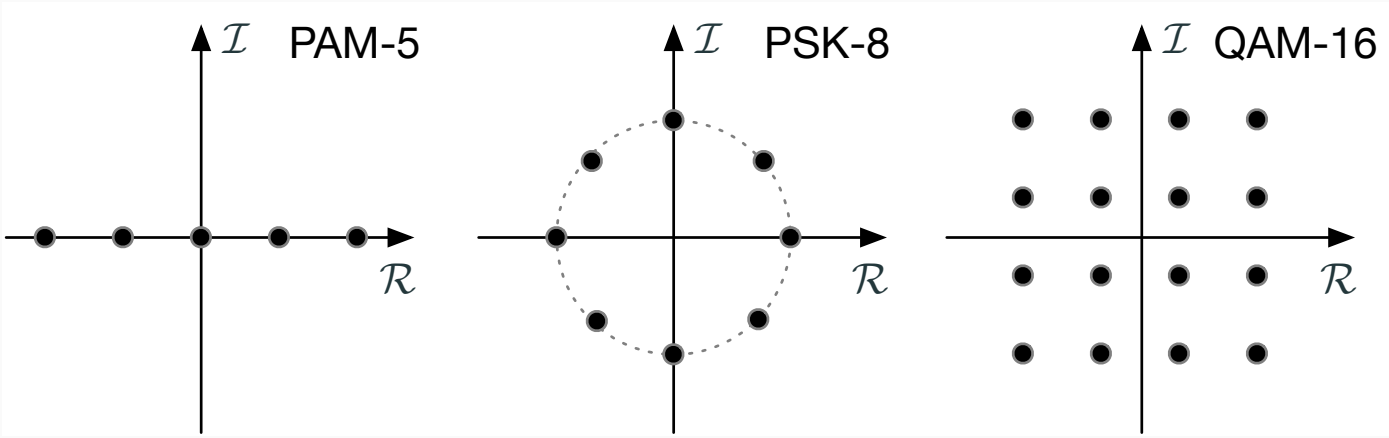


This is roughly the length of your ADSB packets. You need an SNR of about 55 for 1% packet errors.

Again, this is the SNR after the matched filter.

Optimal Detection with a Symbol Constellation

Assume we have a constellation of symbols, as in the last lecture,



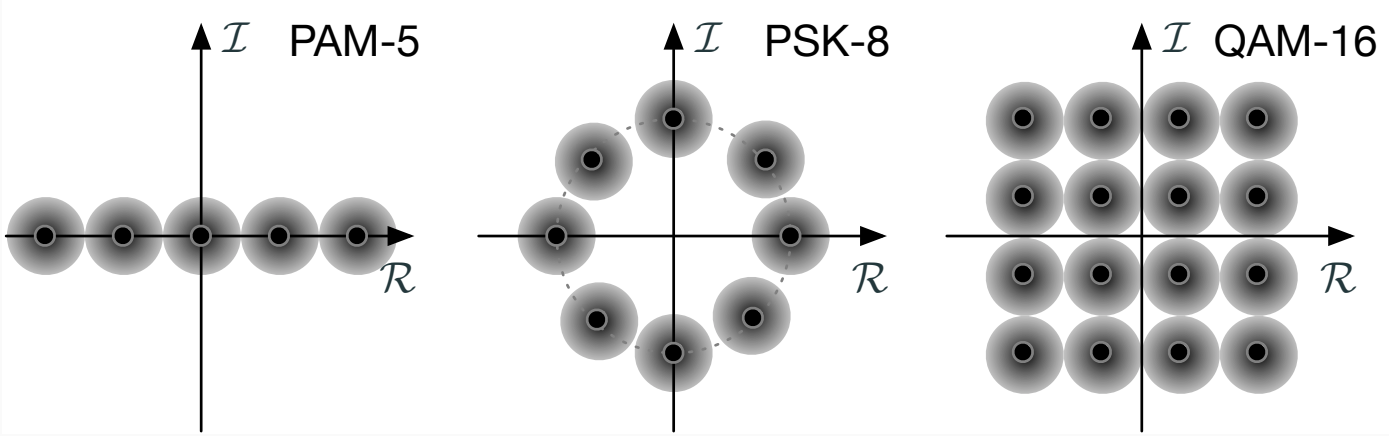
If we assume

- All symbols are equally likely
- The noise is AWGN

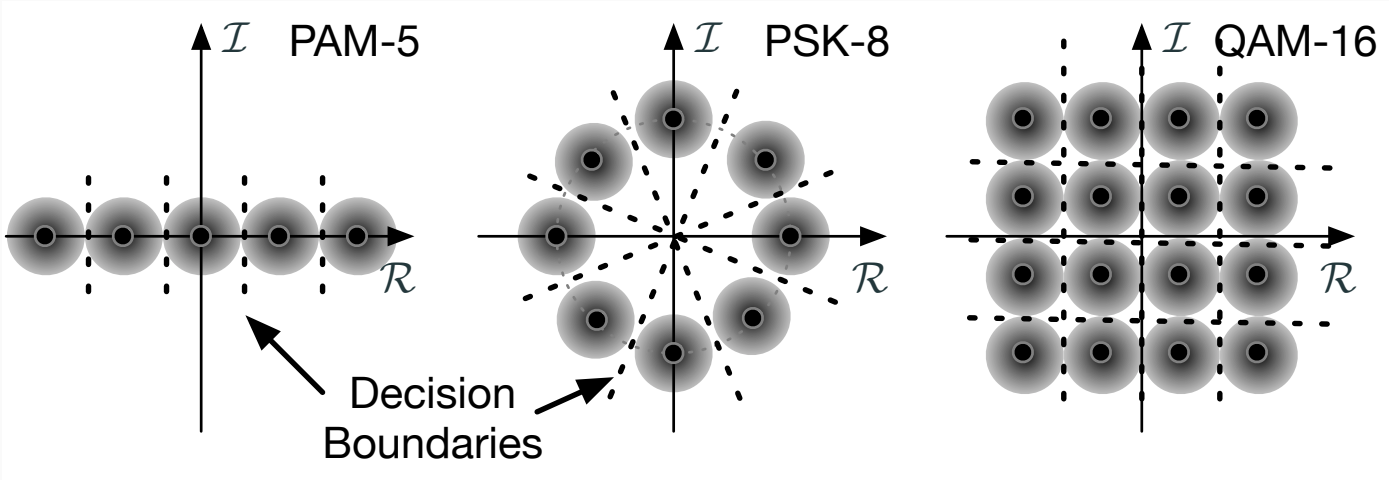
Then the optimal detector simply chooses the closest point in the constellation

Optimal Detection with a Symbol Constellation, (con't)

With noise, each symbol we acquire will have an uncertainty



The optimal choice is to choose the closest one



Designing Constellations

Things we'd like

- Minimum signal amplitude and power for efficiency
- Maximum separation between symbols for detection
- Minimum bandwidth
- Maximum bit rate
- Minimum decoding complexity (timing, frequency offset, etc)

We'll look at comparing some of the major constellations next week

Conclusion

A matched filter is a good way to detect the presence of signals in noise

For communications,

- Synchronize to the symbols
- Use a matched filter for each symbol
- Choose the symbol that most closely matches the received signal

This maximizes performance for noise and interference rejection.

More on synchronization later.