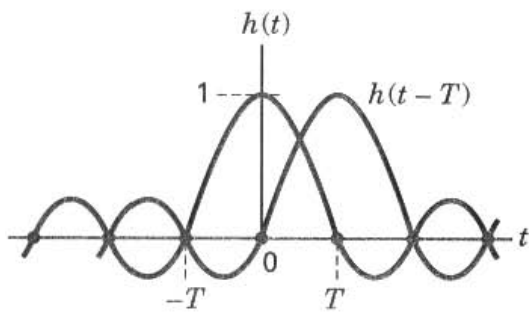


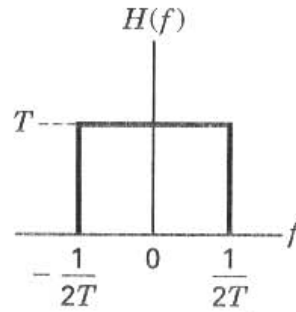
Spectra of pulse shapes

January-16-12
10:10 PM

Ideal Nyquist has minimal BW
the pulse shape is a sinc in time w/nulls at



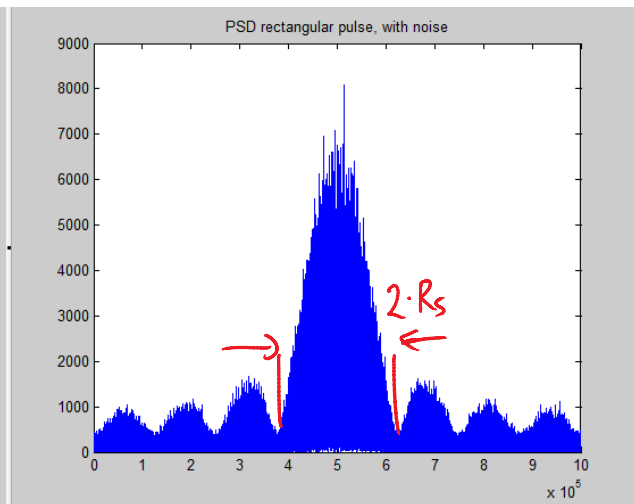
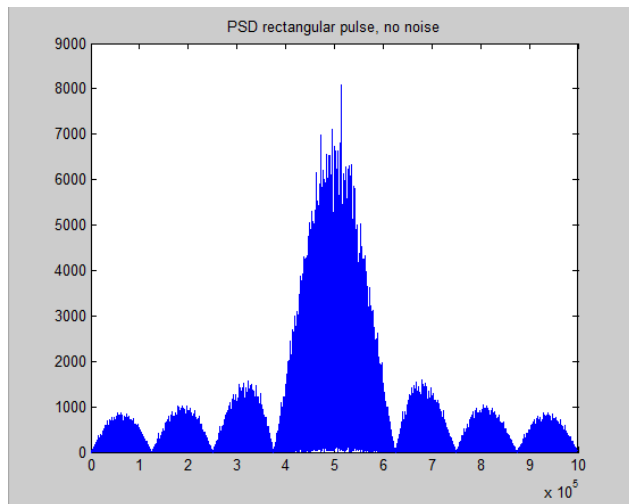
long leisurely
pulses w/ controlled
ISI...



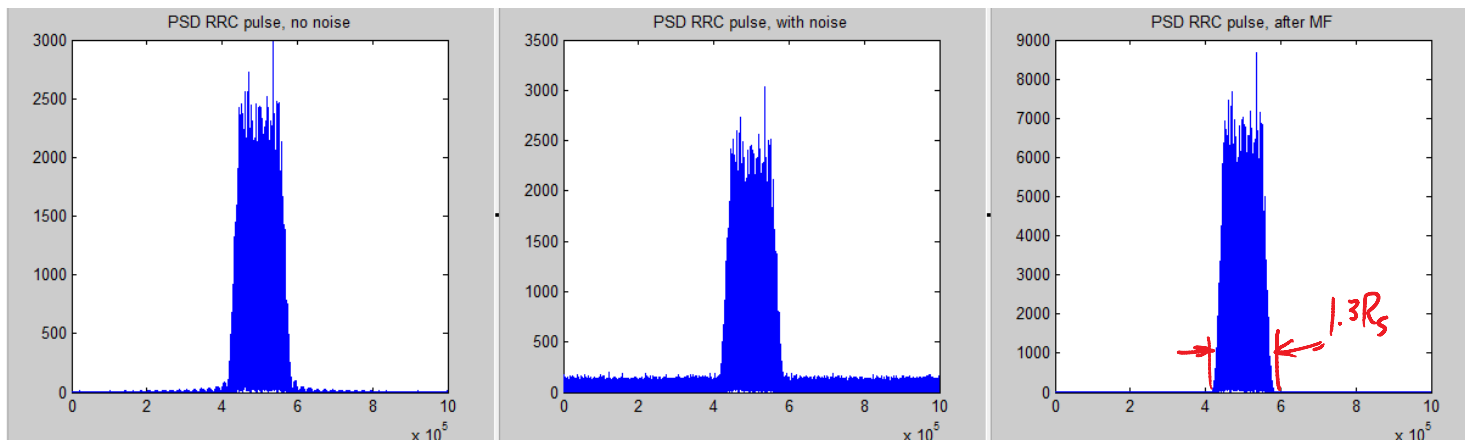
$$BW = \frac{1}{T} = R_s$$

MINIMAL
possible BW
noise equiv BW = R_s

NRZ null-to-null BW of $2/T$ - twice the minimum



RZ would have twice the Bandwidth of NRZ, 4x the minimum
RRC



$$BW = (\text{rolloff} + 1) \cdot \text{minimal BW} = (1 + \text{rolloff}) \cdot R_s$$

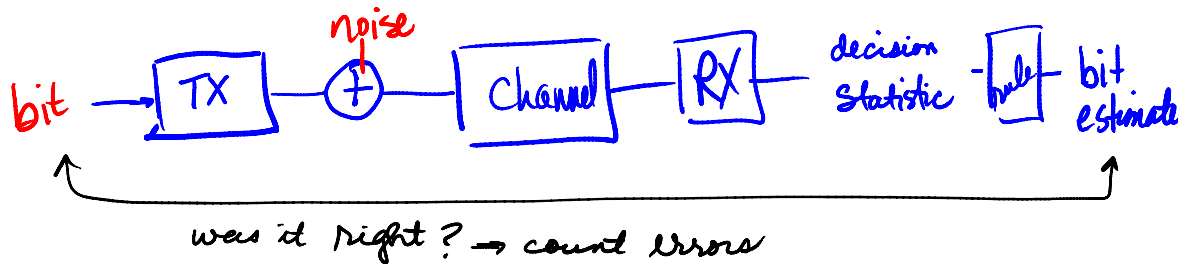
or $OSR < 1$

Monte Carlo simulations

January-13-12
5:40 PM

- Motivation

Suppose I had a coin and I wanted to test if the coin was "fair". I could estimate the probability of heads by tossing the coin many times and counting how many times a head appeared. This is called a Monte Carlo test. I estimate a probability by running repeated experiments. Intuitively I know that tossing the coin once or twice is not enough. The more I toss it, the better my estimate of the probability of heads will get. As a rule of thumb, you should run an experiment until at least 100 events of each type are counted in order to have a good estimate of the probability.



I can use this same approach in a numerical simulator (e.g., Matlab or Optisystem) to help me estimate the bit error rate in a nonideal communications system. For ideal circumstances we have equations to tell us the BER. It is much harder for non-ideal cases. Sometimes it is impossible to come up with closed form solutions for performance in these circumstances. A numerical simulator can be an invaluable tool to deal with the cases of non-idealities. Even when we cannot analyze the consequences we can build a simulator and simply "count errors". The non-ideal circumstances could be

- Non-Gaussian noise
- Non-linear channel
- Finite precision mathematics
- Quantification error
- Phase error
- Timing error
- Intersymbol interference

and the list goes on ...

For communications there are random elements: the data and the noise. These two inputs can be simulated. We can generate random variables according to a known probability distribution. For example we could generate equally probable bits and Gaussian noise. We can also run numerical models of our components that are non-ideal. We can make these models very sophisticated so they are very true to reality. Then we could manipulate these random inputs in a numerical simulation of what the communications devices would do for those inputs. In a numerical simulator we could generate the decision statistics and make the bit estimates just like the receiver would.

If we repeat this experiment many times we can estimate the bit error rate. I send one bit and see if the estimate is correct. I do this again and again and count how many times the estimate is wrong. This forms an estimate of the bit error rate. I mentioned earlier that estimating a probability requires that I count about 100 errors to have a good estimate of the probability. What does this mean if the ostensible BER is 10^{-4} ? 10^{-9} ? 10^{-12} ? This could be a lot of simulation time and is one of the disadvantages of the Monte Carlo method. If it is a real hardship, I can back off to only counting 10 errors, but my estimate of the BER is not going to be terribly accurate.

- Validation

The theoretical expressions for bit error rate are based on ideal components and do not take into account all the myriad of other effects that come from non-ideal conditions. You may ask, given the limited situation when the theoretical equations hold, what place do they have in a simulations course? The theoretical curves are very important for validating your Monte Carlo simulations. Today we are talking about basic choices in your system that will (eventually) be part of larger, more complex simulations. The validation of your results. Being sure that the values you have selected for noise are correctly scaled and normalized is extremely important for the validity of your conclusions and results. Exploiting the theoretical curves when non-idealities are "turned off" in the simulator can help.

Example - ideal threshold. The expression we use for the ideal threshold could be used to validate a numerical threshold estimator algorithm.

If you do not know the true BER to start with, how do you know how many times you need to run your MC iterations? Intuitively we know the more errors we count the more confident we are of your BER estimate. This is true whether the "experiment" is run in the laboratory or in a numerical simulator. You will hear various rules of thumb for counting errors - anywhere from 10 to 100... How do you quantify the confidence you have in your estimate of the BER? This was discussed by Michel Jeruchim in an article I have placed on the class web page. He develops the following plot for the confidence intervals of the BER estimation.

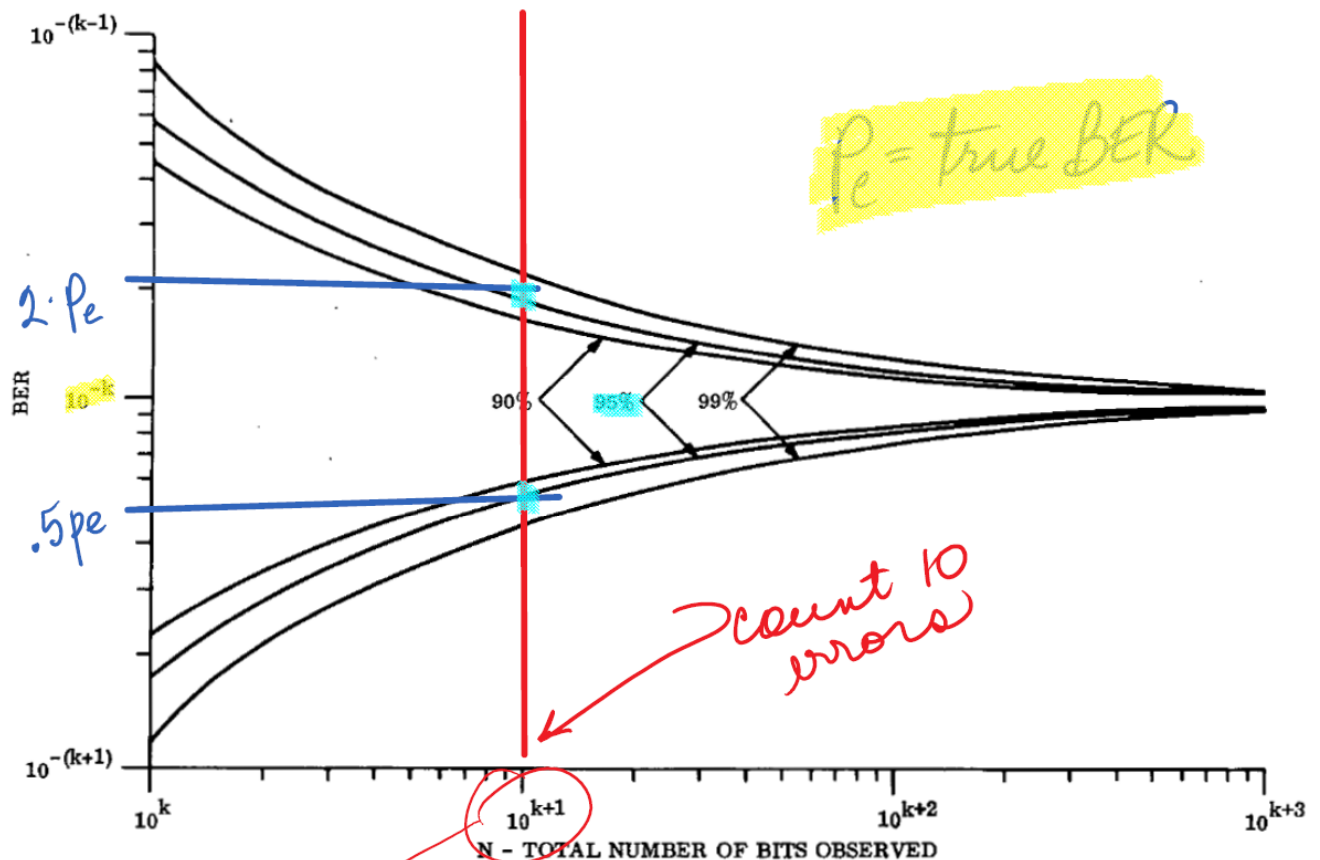


Fig. 2. Confidence bands on BER when observed value is 10^{-k} (Monte Carlo technique).

This is a plot of the probability of the BER estimate falling in confidence intervals about the TRUE VALUE,

as a function of the number of MC iterations.

In this plot the true value of the BER is 10^{-k} for any value of k (not necessarily integer). The confidence intervals are centered on this value. The curves marked 90%, for example, tell you the extremes of the 90% likely estimates about the true value.

Suppose we run enough iterations to expect about 10 errors. That is, if we run $10 \cdot 1/p_e$, the expected number of errors is 10. That is the red bar.

The red bar intersects the 95% confidence curves at two times the true BER and at $1/2$ the true BER. An estimate of BER that is 95% likely to be within a multiple of two of the true value is considered an acceptable estimate. Of course, more iterations are better.

The problem is that we do NOT KNOW THE TRUE BER. Obviously. That is why we are trying to estimate it. Therefore I cannot say I will run $10 \cdot 1/p_e$ iterations - I don't know p_e . One approach is to count the errors. If I measure 10 errors, then maybe I have run $10 \cdot 1/p_e$ iterations. Of course there is uncertainty in that statement - the stopping criteria (10 errors) is random itself. Therefore our confidence interval may not be as good as we think. Hence we really want to target 30 or 100 to have a good estimate.

The number of errors counted is a compromise of complexity and certainty.

Matlab has a routine called BERTools that is specifically designed to help with this quandary. It manages the sweeping across simulations of a broad swath of BER values. Your BER curve can span from 10^{-2} to 10^{-6} , four decades of values. You do not want to run $10 \cdot 10^6$ iterations if your bit error rate is 10^{-2} . BERTools sets a block size that is reasonable chunk of compute time. You run that block size and see if you have any errors. Then you iterate around that block size to achieve the lower BER numbers. These choices are problem specific and can be tweaked by the researcher to compromise complexity and certainty.

OptiSystem takes advantage of the Matlab tool to sweep through simulations. Matlab sets the block size, calls the simulator, and manages the estimation of the BER and running of needed iterations.

Hints

Running Monte Carlo in Matlab is more efficient when using memory instead of for loops. This is what BERTools does well. It encourages/facilitates your use of appropriately sized blocks.

As you sweep through parameters that will cause BER to swing use blocks of data in memory as they are efficient in Matlab. Loop through blocks to gather more iterations. Have a stopping criteria of 100 bits are maximum number of iterations. The best block size is a function of your RAM memory. Try large values and see how it works 10^5 when the BER is high. If it is still slow, back off the block size.

AWGN command is useful ... but be careful you are using it correctly; always validate your noise normalization (use 'measured' switch in the command)

Rectpulse and intdump can easily generate NRZ and the matched filter (actually correlator)

Symerr and biterr commands count errors (and keeps track of bits vs. symbols for non-binary modulation)

Winzer paper

January-17-12
9:51 AM

In the homework you will contrast performance using a matched filter receiver with a low pass filter with fixed bandwidth. Why a fixed bandwidth low pass filter? That is what is used in the SONET standard for example. Theoretically for an AWGN the matched filter is optimal for BER - whatever pulse shape you use. However you should match to the RECEIVED pulse which has been convolved with the channel impulse response and the front end impulse response.

$$h_p(t) * h_c(t) * h_{FE}(t)$$

You may not know the channel or it may be time varying. In wireless, for example, adaptive filters and equalizers are used to optimize BER. With the advent of optical coherent detection this is becoming the norm for optical receivers also. However for IM/DD this would be prohibitively expensive. For these systems an analog filter is most economical. What analog filter? Rule of thumb led to Bessel filter with 0.75 x bit rate as a FWHM bandwidth.

Like any rule of thumb it is not actually optimal, but a good compromise over a myriad of practical impairments in real systems. Remember that all our analytical formulas assume ideal components and AWGN. Your components may or may not support all frequency bands equally, etc.

Consider the homework assignment. The system you are simulating at 10 Gb/s is probably very close to a purely AWGN channel. However at 40 Gb/s you should see optical filtering effects that introduce ISI. The theoretical equations do not take this into account if you do not match to this channel effect.

Winzer et al. examined filtering effects in a 2001 paper where the EDFA is used as a pre-amplifier at the receiver. It is a simulation not unlike our homework assignment.

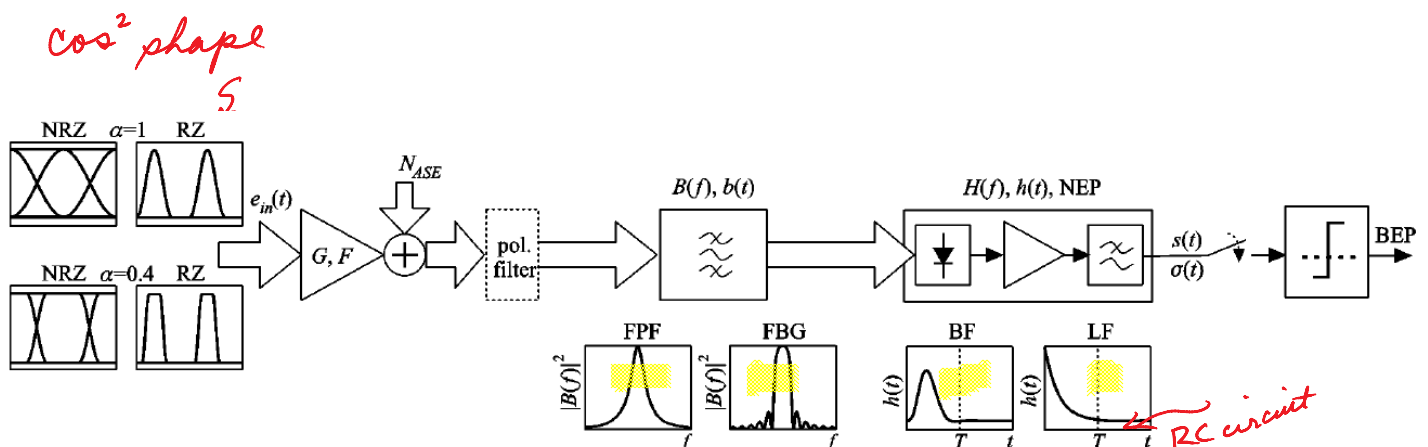


Fig. 1. Optically preamplified receiver structure. The optical (NRZ) input field $e_{in}(t)$ (cf. optical power rate diagram) is optically preamplified (gain G , noise

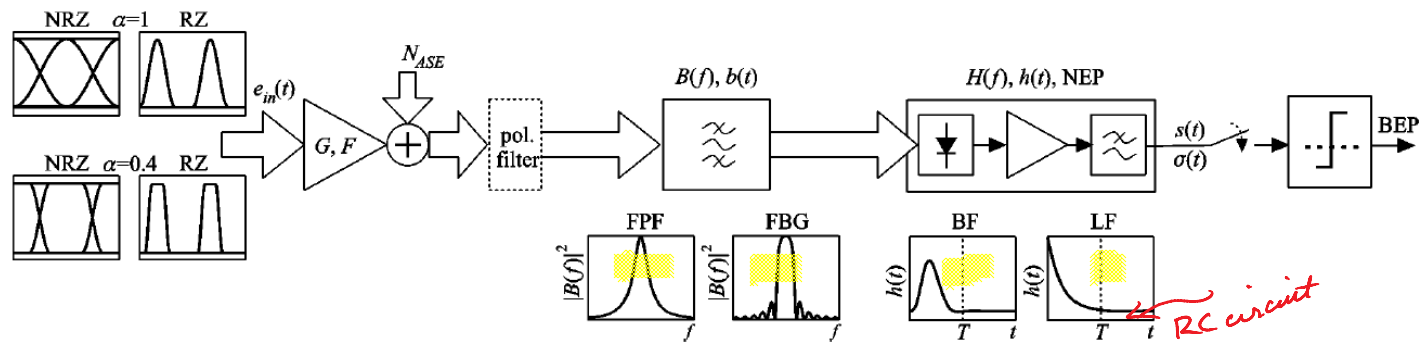


Fig. 1. Optically preamplified receiver structure. The optical (N)RZ input field $e_{in}(t)$ (cf. optical power eye diagrams) is optically preamplified (gain G , noise figure F) and corrupted by amplified spontaneous emission (ASE). In addition to an optical bandpass filter [Fabry-Pérot filter (FPF), or fiber Bragg grating (FBG)] with power transmission $|B(f)|^2$, a polarization filter may be employed to reduce ASE. The impulse response of the detection chain is assumed to have fifth-order Bessel (BF) or first-order RC low-pass (LF) characteristics [impulse response $h(t)$]. Electronic noise is accounted for by a noise equivalent power (NEP). Sampling and threshold decision of the electric signal with mean $s(t)$ and variance $\sigma^2(t)$ leads to a bit error probability (BEP).

SONET standard OOK NRZ w/ 0.75R fourth order Bessel He(f)

Experiments with denser packing of wavelengths - narrower, sharper optical filters - showed that 1.2R with realistic NRZ shape yielded better BER. When using RC low pass filters (not raised cosine) .9R was better. Therefore the filtering effect was very visible in these systems. The paper simulates various pulse shapes and electrical filter shapes and varies Be to find the best BER - an excellent example of the value of simulation and very close to our homework assignment. Modifying our code could recreate their results...

While this example is for OOK, filtering effects have large impact in coherent systems as well. Baud rates push the frequency response of components and create impairments. DSP is enlisted to help...