An introduction to modelling optical fibre links

The aim of this experiment is to develop a computer model of an optical fibre communications link that includes the effects of attenuation, dispersion and noise. Your model should be able to simulate eye-diagrams and you should use these to assess link performance. **Key result**: for suggested typical system parameters determine whether 1Gb/s and 10 Gb/s links are limited by attenuation or dispersion, and estimate at what length interval you will need to introduce regenerators for each bitrate.

When designing optical fibre networks it is often convenient to model or simulate the propagation of optical pulses through the complete fibre link to determine the effect of attenuation and dispersion in the fibre and noise in the detector. Attenuation and dispersion are the two factors that have most effect on the pulses; attenuation reduces the power of the pulse and dispersion caused the pulse to broaden. In high data rate optical networks these effects limit the distance that the pulses can propagate and limit the data rate itself. This exercise is designed to encourage you to investigate these effects, and to study their impact on optical fibre links, by writing a simple computer program to model pulse propagation.

You can use any method you wish to write programs to produce the results. In the past, students have used Excel, Maple, Matlab, Python or C++. When choosing what you use you should be careful to consider what your chosen system offers. While C++ offers the greatest speed it is not necessarily the most flexible as plotting tools are not automatically included. Python and Matlab may let you get things done much more quickly as plotting tools and a myriad of other useful functions are already built in - if you know how to use them already! Maple is a symbolic mathematics package that lets you plot easily by defining functions symbolically without having to worry about the details of things like setting up sets of *abscissa* values, but again may have a learning overhead. Finally Excel might be the most familiar to you but its programming capabilities are limited. You are strongly recommended to use Matlab as it is available for free through college on a student licence and as it is used elsewhere in the course you would be well advised to gain some familiarity with it.

This script is written as a set of assignments, the earlier ones in particular are to help structure your work and help you develop your models. However, do not take this script as a template for your report, which should concentrate instead on the more advanced elements of your study such as actually working out what links would work in practice in **assignment 4**. And do try to get on to the extension exercises at the end of the script, once you have a working model they would be easy to implement.

Assignment 1 - Dispersion

Write a program to model the propagation of Gaussian shaped pulses through a fibre link. Assume a pulse shape

$$P(t) = P_0 \exp(-t^2 / 2\tau_0^2),$$

where P_0 is the input power (peak), t is time, and τ_0 is the width of the Gaussian pulse. Take into account the pulse spreading by using the formula

$$\Delta\tau = D\,\Delta\lambda\,L$$

where D is a dispersion coefficient in units of [ns / (km.nm)], L is the length of the fibre, and $\Delta\lambda$ is the optical linewidth. $\Delta\tau$ is the dispersion induced pulse broadening such that the total pulse width is $\tau^2=\tau_o^2+\Delta\tau^2$, but remember that as the pulse gets wider its height must reduce to

maintain the same pulse energy. Plot the pulse shape for D = 15ps/nm/km, $\Delta\lambda$= 0.1 nm, τ_0 = 0.5 ns and for L = 10km, 50km and 100km.

Next plot, for the three lengths, a train of pulses initially of width 0.5ns separated (peak to peak) by a time interval of 2 ns. Compare the output of the detector when the pulses propagate through the fibre with the following set of parameters.

(a) A train of 10 pulses initially of width 0.5 ns separated by intervals of 2 ns.

(b) A train of 10 pulses initially of width 0.5 ns separated by intervals of 1 ns.

(c) A random sequence of pulses (eg 110010101000110) initially of width 0.5ns separated by intervals of 2 ns.

Assignment 2 – Eye diagrams

In a digital system it is important to be able to differentiate between a received 0 and a received 1. In the previous section you may have noticed that at some points it is difficult to be able to discern individual pulses that are next to each other, but actually it is still possible to see the difference between a pulse and no pulse. In practice a communications channel would be evaluated by considering the sort of random sequence of pulses as described in (c) and then overlaying many received such random signals aligned at a pulse centres to produce an *eye diagram*. The ability of the receiver to discern the difference between a 0 and a 1 is thus revealed in the "openness" of the eye at the aligned pulse centres.

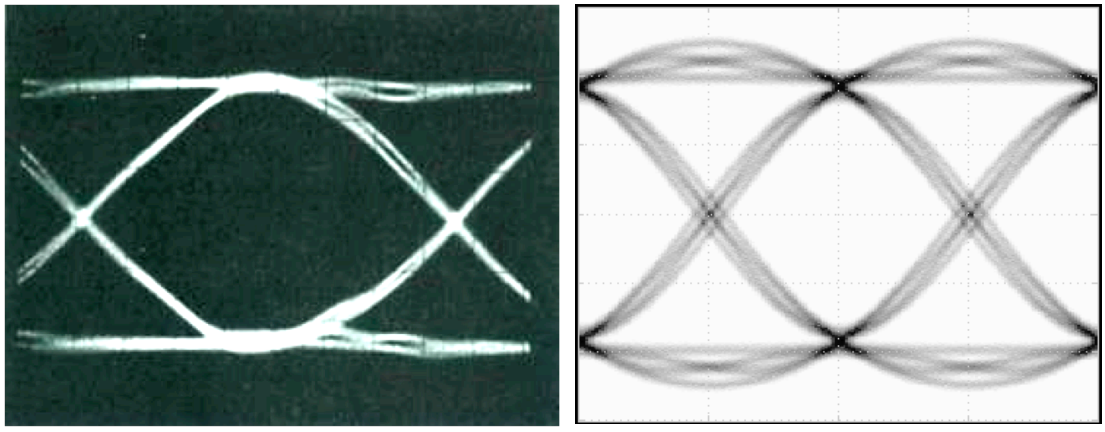

Figure 1: Eye diagrams

Write a program that produces an eye diagram for the above systems by overlaying signals corresponding to all possible 3-bit transmitted signal patterns: {000, 001, 010, 100, 011, 101, 110, 111}. You could also extend this program to considering 5 bit patterns and compare the results.

Use your eye diagrams to think about what might be a sensible Gaussian pulse width to use for a given bit-rate.

Assignment 3 – Noise

The optical detection process is subject to random noise associated with several different physical processes. This can lead to errors in interpretation of data. For example, a binary "0" can be mis-interpreted as a "1". Most links are tested to evaluate their sensitivity to noise

and a bit error rate (BER) is specified. Typical values of error rates are 1 in 10^9 or even 1 in 10^{12}.

Consider and discuss how you can extend the simulations carried out in assignment 1 and 2 to include the effects of random noise. Remember in particular that due to the bandwidth of the receiver electronics, the noise is effectively low pass filtered. You could either try to find a way to simulate low pass filtered noise, or perhaps you could consider how a sampling oscilloscope might display many overlayed *sampled* values of the received signal as discrete points rather than a continuous line (the second eye diagram in figure 1 is simulated in this way).

In each case there are 2 main types of noise in the system that you should consider:

(a) Thermal noise in the receiver electronics is usually the limiting factor in an optical communications system and for a given receiver this is characterized by the Noise Equivalent Power - the received optical power that would present the same signal level at the detector as that due to the inherent noise in the receiver. To complicate this further, some receivers might simply quote a value for NEP in units of power (W) for a fixed detector bandwidth, and others might use units of power per square root of bandwidth (W/√Hz) to take account of the typical variation in noise as the bandwidth (B) of the receiver is changed. Your program should be able to accept values in either and should also take account of the bandwidth that you require to receive your signal [remember the minimum bandwidth required for a NRZ signal with bitrate, b, is B=b/2].

(b) The fundamental limit to noise generation in an optical detector is set by the noise generated by quantum processes – your detector will detect individual photons in the optical signal and, in the simplest case, these will tend to arrive at random times. This is known as shot noise. For a constant photocurrent the mean square variation of current (I) generated by shot noise is given by

$$\langle I^2 \rangle = 2\, e \langle I \rangle B$$

where e is the electronic charge, and B is the *electrical* bandwidth of the detector. The noise is Poisson distributed and clearly related to the expected or mean current received, $\langle I \rangle$. So you would actually expect the brighter parts of your signal (the ones) to have a higher noise than the dimmer parts (the zeros).

And there are 2 different ways you could model the noise

(a) **Noise in a sampled signal:** Here you are trying to mimic the signal that you would see on a sampling oscilloscope where the individual samples are taken at quite different time points from different pulses in your signal and so are completely uncorrelated. Use the descriptions of how to calculate thermal and shot noise above to come up with an expected noise level at each point in your signal (as a function of the expected signal current at that point in time when considering shot noise) and then use a random number generator to calculate a noise that you can add to the sample of your signal. The Matlab function `randn()` is a good way to go about this.

You can very quickly generate many samples and then plot them on a graph. Remember though that your samples are uncorrelated so it does not make sense to join up the points in your plot. In fact you might want to consider plotting your data as a 3D histogram using the Matlab function `hist3()`. The example code below should give you some ideas of how to start going about this.

```
n=100000;       % Number of samples to take
t=0:10/n:10;    % A set of time points spanning 10 nsec
sig=exp(-(t-5).^2/(2*0.5^2));   % A gaussian pulse of width 0.5ns
noise1=randn(size(sig));        % A set of random gaussian distributed
                                % noise values (standard deviation = 1
noiseT=0.1*noise1;              % A thermal noise signal (does not depend
                                % on current)
noise2=randn(size(sig));        % Another set of random gaussian distributed
                                % noise values (standard deviation = 1
noiseS=0.1*sqrt(sig).*noise1;   % A shot noise signal (depends on
                                % sqrt of current)
sign=sig+noiseT+noiseS;         % Noisy signal
figure(1);
plot(t,sign,'.');               % Remember to not join up the dots
figure(2);
h=flipud(hist3([t;sign]',{0:0.05:10;-0.5:0.01:1.5}));
                                % Create the histogram - a bit of
                                % manipulation needed to make it display
                                % correctly
imshow(h,[]);
colormap('hot');
```

At the optical detector the optical power is converted, linearly, to a current and the current generated from 1 mW of optical power incident on the detector is *R* mA, where *R* is the responsivity of the detector. A typical value of *R* would be 0.7 mA/mW. Using ideas from your discrete sampled noise analysis generate a random current noise and add it to the simulations shown in assignment 2. (Note, you can choose an arbitrary distribution for the noise). Comment on the results. What would constitute a "bit-error" in your simulation and how could you determine the BER?

(b) **Noise in a continuous signal:** This is the sort of thing that you might find displayed on a normal analog oscilloscope. Here you should be careful to accurately represent the frequency content of the noisy signal, which will always be reduced as much as possible to reduce the noise level as much as possible, but still retain the signal information. Typically you need a bandwidth B that is half the bitrate, ie b/2. A useful way to filter signals in the digital domain (in your a model for instance) is to use what is known as an Infinite Impulse Response (IIR) filter. These are very easy to implement and mimic real world filters and electronics very well.

In the simplest case to mimic a low pass resistor/capacitor (RC) filter, starting with a sampled signal x(i) at time points t(i) then to calculate the filtered signal y(i) you compute:

$$y(i) = (1-\alpha)y(i-1) + \alpha x(i)$$

As you can see the filter computation is recursive in that you need to compute y(i-1) before you can compute y(i). In this case the filter mimics an RC filter with a time constant of $-t_s/\ln(1-\alpha)$ where t_s is the time between samples, this filter would have a cutoff frequency of $f_c = -\ln(1-\alpha)/(2\pi t_s)$.

Actually Matlab has functions already built in to apply IIR filters using the `filter()` function, and in general this can compute much more complex filters of the general form:

$$a_1 y(i) = \sum_{j=2}^{n_a} a_j y(i-j+1) + \sum_{j=1}^{n_b} b_j x(i-j+1)$$

The calculation of the required a and b coefficients can be quite complex but there are some built in functions that will work them out for you for simple cases: for instance `butter()` will design butterworth filter coefficients. A good start would be to try a second order butterworth filter. An example showing the effect this has on a square pulse is given below:

```
n=1000;
tstep=10/n;                 % The step between time points (in nsec)
t=0:tstep:10;               % A set of time points
fs=1/tstep;                 % The sampling frequency
sig=1.0.*(t>4).*(t<6);      % Generate a square pulse
figure(3);
clf;
plot(t,sig,'b');            % Plot the input signal
hold;
fc=0.25;                    % The cutoff frequency of the filter (in GHz)
[b,a]=butter(2,fc/(fs/2))   % Generate filter coefficients, note that the
                            % frequency that you give is normalised by the
                            % nyquist frequency, which is half your
                            % sampling frequency (fs/2)
sigfb=filter(b,a,sig);      % Filter the data
plot(t,sigfb,'r');          % And plot it, now you can join up the dots!
```

You can now use the filter to see what happens to shot noise by remembering that your ideal signal tells you what the expected optical power should be. From this you should be able to work out how many photons you would expect to detect in any given time slot, and then use another Matlab function `poissrnd()` to calculate a poisson distributed random number based on that expected number of photons to calculate an actual detected photon signal. The following additional section of code shows this and the resulting filtered output.

```
sigphot = poissrnd(sig);    % Generate some poisson distributed photons
plot(t,sigphot,'g');        % Plot the photon signal
sigf=filter(b,a,sigphot);   % Filter the signal
plot(t,sigf,'m');           % And plot it
```

Note that in this example we are expecting to see on average 1 photon per sample in the one part of the square pulse, and zero otherwise. With 200 samples per pulse this would give 200 photons in our optical bit. You would need to set these levels appropriately for the real physical system you are looking at.

If you want to model band limited thermal noise as well then you can use a similar approach, by just filtering a set of random numbers generated with randn() again, but you should be careful to get the level of the noise correct (see NEP above). Try generating some filtered signals and check that they have the levels that you want.

From the signals that you used to model eye diagrams above calculate a few examples of eye diagrams where you are showing the full effects of the frequency bandwidth limits of your detector, on both the signal and the noise. What happens when you start taking the bandwidth B below b/2? [Note that a real filter will still let some information through above its cutoff frequency].

Assignment 4 – Using your program

Simulate the operation of a typical optical communications system using your programs. You should try to demonstrate its use for a range of scenarios that demonstrate systems that are limited by both loss and by dispersion, and by receiver and shot noise. For example consider the following system:

A fibre with loss 0.3dB/km, Dispersion D=-20ps/nm/km, using a source of power 1mW and bandwidth $\Delta\lambda$=0.2 nm, and a detector with a NEP=2 pW/√Hz.

For operation at (a)1Gb/sec and (b)10Gb/sec, try to determine whether the fibre link it is loss or dispersion limited in each case using your simulations.

Extensions…

(a) It is obviously best to use a very narrow linewidth source for your communications system, but you can still run into problems as the optical linewidth of the modulated signal is still determined by the Fourier transform relationship between the pulse width (in time) and the optical linewidth (in optical frequency). This will determine the optical linewidth you need to use in the formula for $\Delta\tau$, What difference to your answers in assignment 4 will there be if such a system is used?

(b) Up until now you have simulated a 2-level system where the expected value at the receiver should have one of 2 levels and can thus convey 1 bit/symbol. Consider now a 4-level system (how many bits per symbol can this convey?) and plot its eye diagram.