Fourth Year Project Logbook

Week 1

30/09/13 - Exploring simple case with PAM modulation

I received the *PAM.pdf* file outlining the case where a signal is sent through a channel with AWGN and received with a timing error at the receiver. I read through the file several times to get an understanding of the underlying equations.

Leaving the Gram-Charlier series aside for the moment, I started getting to grips with Mathematica and implementing the transmission system model:

$$X = \omega_0 g_0 + \sum_{k=1}^{40} (\omega_{-k} g_k + \omega_k g_k) + \nu$$

where $g_k = g((\Delta + k)T)$, $g(t) = (u_T * h_l * u_R)(t) \times cos(\theta)$ and ν is a zero-mean Gaussian random variate with $\sigma_\nu^2 = N_0 \varepsilon_R$.

I learnt the basics of the interface, and began inplementing the filter and channel impulse responses (I.R.). I need to double-check the definition of the Root-Raised Cosine (RRC) Filter, as the impulse response wasn't as expected.

Later, I found the correct form for the RRC and double-checked it using Octave. The equation used is listed below. A plot showed that this equation is invalid at $t = \left[-\frac{T_s}{4\beta}, 0, \frac{T_s}{4\beta}\right]$, so I plan to find its limit at these points using Mathematica to obtain the complete solution.

$$h_{RRC}(t) = \frac{2\beta}{\pi\sqrt{T_s}} \frac{cos\left[(1+\beta)\frac{\pi t}{T_s}\right] + \frac{sin\left[(1-\beta)\frac{\pi t}{T_s}\right]}{\frac{4\beta t}{T_s}}}{1 - \left(\frac{4\beta t}{T_s}\right)^2}$$

01/10/13 - Implementing Raised Cosine functions

I implemented the function above in Mathematica, and using the `Limit` function found the value of the function at the following undetermined points:

$$h_{RRC}(t) = \begin{cases} \dfrac{4\beta + \pi(1-\beta)}{2\pi\sqrt{T_s}} & t = 0 \\ \dfrac{\beta}{2\pi\sqrt{T_s}}\left(\pi sin\left[\frac{(1+\beta)\pi}{4\beta}\right] - 2cos\left[\frac{(1+\beta)\pi}{4\beta}\right]\right) & t = \pm\frac{T_s}{4\beta} \\ \dfrac{2\beta}{\pi\sqrt{T_s}}\dfrac{cos\left[(1+\beta)\frac{\pi t}{T_s}\right] + \dfrac{sin\left[(1-\beta)\frac{\pi t}{T_s}\right]}{\frac{4\beta t}{T_s}}}{1-\left(\frac{4\beta t}{T_s}\right)^2} & \text{otherwise} \end{cases}$$

I also implemented the Raised Cosine function for the channel function, using the impulse response below[1]. I was unable however to convolve the receiver and transmitter filter functions using the `Convolve` function, even when I limited the impulse response using a `UnitBox`.

$$h_{RC}(t) = \frac{sinc\left(\frac{\pi t}{T}\right) cos\left(\beta\frac{\pi t}{T}\right)}{1-\left(2\beta\frac{t}{T}\right)^2}$$

I looked into Mathematica's treatment of the Gaussian distribution, and figured out how to generate random noise vectors following a Gaussian distribution, as well as how to generate a list of random binary symbols.

After discussing the convolution issue with David, he suggested that the channel should be initially modelled as ideal and therefore the overall channel and filter I.R. $g(t)$ can be defined as a Raised Cosine function, as defined above. I should therefore be ready to implement the simple ISI model tomorrow.

02/10/13 - Wrapping Up the Initial PAM Model

I pulled together the Raised Cosine function and random number generator to impiment the given simplified function for the PAM receiver output, given below. Playing around with the settings, I was able to show how the g_k function increases with the timing error. I decided to study the Mathematica environment a little more before carrying on with any programming.

$$X = \omega_0 g_0 + \sum_{k=1}^{40}(\omega_{-k}g_{-k} + \omega_k g_k) + \nu$$

[1] Proakis, "Digital Communications"

03/10/13 - Delving deeper into Mathematica

I devoted some time into looking through Michael Quinlan's notebooks and better understanding the workings of the `Table` functions and the various plotting options. Fortunately my notebook was corrupted so I was able to rewrite it and understand the model a bit more. I need to figure out what variance value the noise PDF should take on, as the noise appears to be overwhelming the timing error effects. Translating the resulting PDF's into patterns is another question that needs some thought.

Week 2

07/10/13 - Matrix manipulations

I decided to spend another day learning about the Mathematica environment, in particular matrix manipulation and generation. I looked into the `Apply`, `Map` and `Partition` functions and wrote some examples to figure out how to convert mathematical problems to Mathematica notation using matrices. I hope to convert the code to use matrices tomorrow to hopefully simplify and speed things up.

I also implemented David's equation for properly calculating the AWGN function variance from SNR[2], from last Friday's meeting.

08/10/13 - Fixed I.R. and Kernel Density Estimation

The first job was to rewrite the code to make use of the simple dot operator to calculate all the ISI components[3]. With the new code I was able to carry out many more runs and get much more detailed output. In addition, when I was rewriting the code I noticed a typo in the Raised Cosine I.R. that was degrading performance in the perfectly synchronised case. With both of these changes made, I decided to use Kernel Density Estimation to see what effects the timing offset has.

[2] See *davenotes.pdf*

[3] The ISI components are now calculated using:

$$\left[\sum_{k=0}^{k=40}\left(g_k\omega_k^1+g_{-k}\omega_{-k}^1\right)\cdots\sum_{k=0}^{k=40}\left(g_k\omega_k^m+g_{-k}\omega_{-k}^m\right)\right]=[g_{-40}\cdots g_{-1}g_1\cdots g_{40}]\bullet\begin{bmatrix}\omega_{-40}^1 & \cdots & \omega_{-40}^m\\ \vdots & & \vdots\\ \omega_{-1}^1 & \cdots & \omega_{-1}^m\\ \omega_1^1 & \cdots & \omega_1^m\\ \vdots & & \vdots\\ \omega_{40}^1 & \cdots & \omega_{40}^m\end{bmatrix}$$

where ω_k^j is the k'th ISI with the j'th timing offset.

Using offsets of 10^{-15}, 0.05, 0.1 & 0.15, the following values of $g_k, k \in \{-40 \cdots -1, 1 \ldots 40\}$ were calculated.

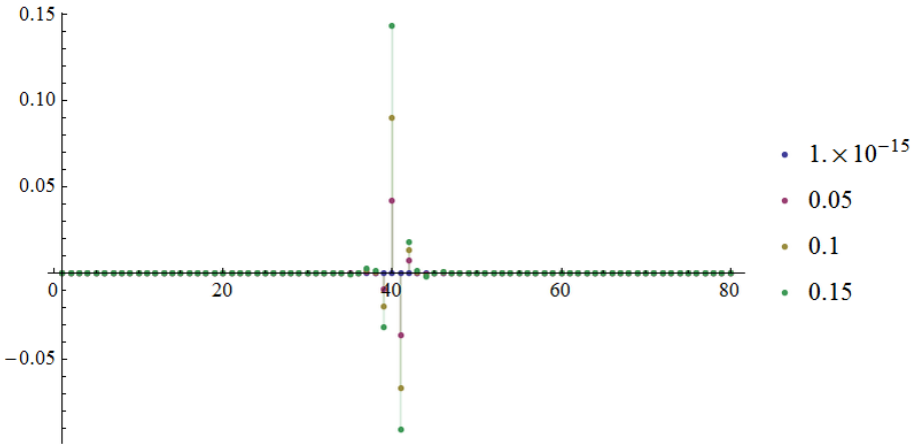

Figure 1: g_k linear plot

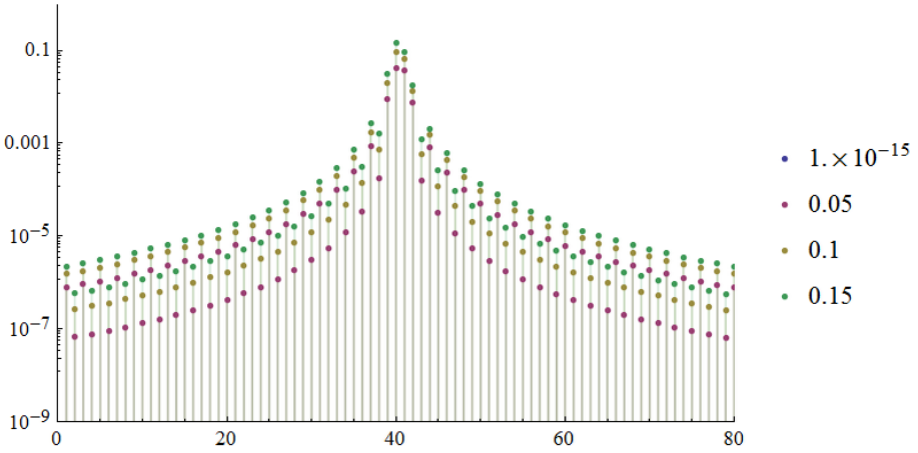

Figure 2: g_k log plot

Using `SmoothKernelDistribution` to perform Kernel Density Enstimation with 1 million points produced the following estimated PDFs for both possible transmitted values. As the timing error increases, we note that the PDF spreads out, but the mean remains steady.

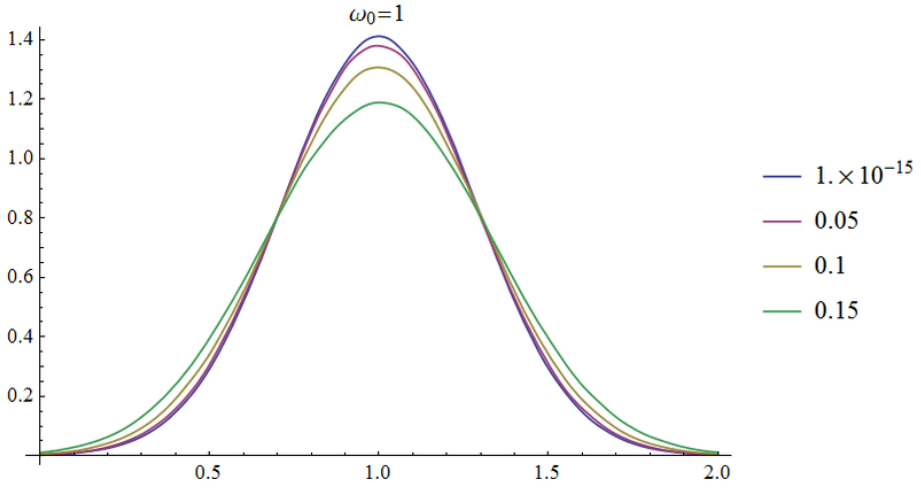

Figure 3: kernel density estimation $\omega_0 = 1$

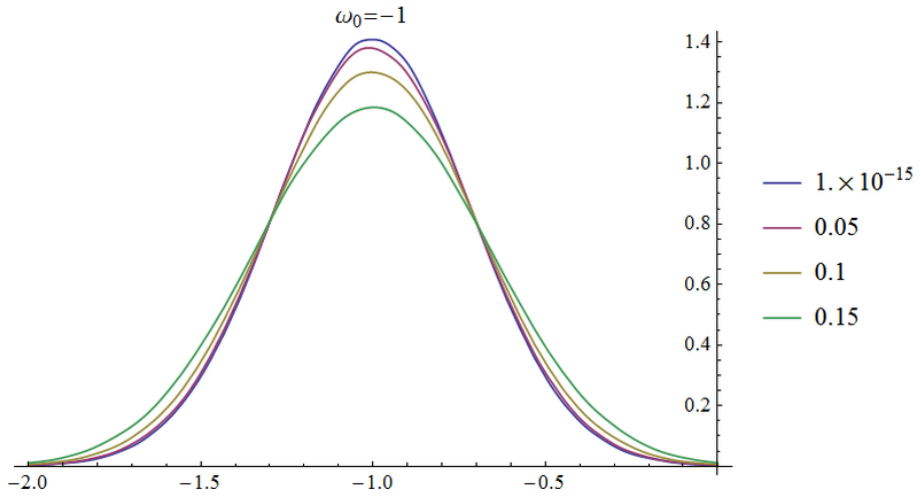

Figure 4: kernel density estimation $\omega_0 = -1$

09/10/13 - Setting up Digital Comms Lab PC

With Ger's help, I set up an account on `Digital Comms Lab 1` & `Digital Comms Lab 2` and got the internet working. Mathematica 8 is installed and working on both machines, we will have to consider whether an upgrade to Mathematica 9 would be useful or not. Git and VNC or similar have to be installed next. A request was made to the Boole cluster for access for this machine, however the email given (`bcrisupport@bcri.ucc.ie`) was invalid.

10/10/13 - Probing the Elec Eng network

After finding out the Boole cluster was no more, I used today to examine what hardware I had available to me. I got access from Ger to the public `UEPC004` server, and from there I am able to access machines on the elec eng network. I set up a *Remote Desktop Protocol* link to `Digital Comms Lab 1` through this server, allowing me to control the machine from any location. I am also able to log remotely into EDA lab machines, and run Mathematica 5 on those machines.[4] Ger has been known to tweak machines in response to personal requests, so if asked nicely he may let me use two or three of these machines concurrently.

Given these resources, I feel there are three ways I could continue:

- I could upgrade to the latest version of Mathematica on all machines, and set up a Mathematica cluster with `Digital Comms Lab 1` as the front end and the EDA Lab PCs as remote nodes. With this setup, all machines would act as one (as in a traditional cluster). This would be the easiest to use, but would require considerate work to set up.
- I could use the `MathLink` interface to acheive a similar, lower-level version of the former, with the EDA Lab machines as independent, remote slaves and `Digital Comms Lab 1` sending commands to these slaves and collating the replies. This setup is distributed computing with a star topology, and would be easier to setup. The downside is that the code needs to manually divide the task between each of the nodes, and needs to be well designed to minimise network delays.
- I could simply run the code in parrallel on each of the machines available to me, dumping the results to text files, and collate the data at the end. This would require no setup, and code written on any machine would only require porting to another version of Mathematica. Additionally this seems like it would deal best with hiccups such as machines going down and it does not depend on a connection between the machines. The downside is there would be some overhead with collecting the results afterwards.

[4] The GUI does not work when using `ssh` to access the EDA Lab machines, but using the command `math` to start and operate Mathematica kernels does.

Week 3

14/10/13 - Running longer scripts on the EDA machines

Today I spent some time figuring out how to build and run scripts on the EDA machines. I found that defining a module in a text file and copying the Mathematica code into the module allows the code to be called with input arguments, and writing the output to a text file and placing the module in a loop allows each pass to be recorded for later parsing[5]. After running the code overnight, this system appears to work, and is scaleable over multiple machines. The main disadvantage is the size of these files (7.7MB per 400,000 values), so I must either figure out how to transfer them over the network or see whether reducing the precision of the output values will reduce the file sizes.

15/10/13 - Reducing output size

Given the 15GB of samples produced the night before was far too much to pull off the machine, I copied 20 million of the samples and plotted them to make sure the script had worked in practice[6]. I then looked into how I could reduce the size of the output produced, and decided to replace the `SmoothKernelDistribution` function (which came in after Mathematica 6.0 and therefore couldn't be used on the EDA machines) with a fine-grained histogram function[7]. This allowed me to add the probabilities generated in each sweep to those generated before and keep the output to a handful of 1kB files. I ran the simulation overnight to check it.

16/10/13 - Moving onto 4-PAM

Checking the output from the night before, I get a similar PDF plot as with the `SmoothKernelDistribution` function. I therefore modified the code to examine all 3 decision region boundaries in a 4-PAM system and ran the simulation for 100 million samples per condition. The resulting distributions shown below show increased probability of error with timing error, as expected, but decision region boundaries in this case remain the same.

I could imagine finding a value for the probability of error and moving onto PSK systems as the next steps in the process.

[5] Using the `Get` and `Put` methods. The `DumpSave` method is supposed to be more efficient, but was added after Mathematica 6.

[6] For the record, I could only use a fraction of them, as loading all 20 million samples crashed the machine for over an hour.

[7] I am assuming that both approach the true PDF as $N \to \infty$

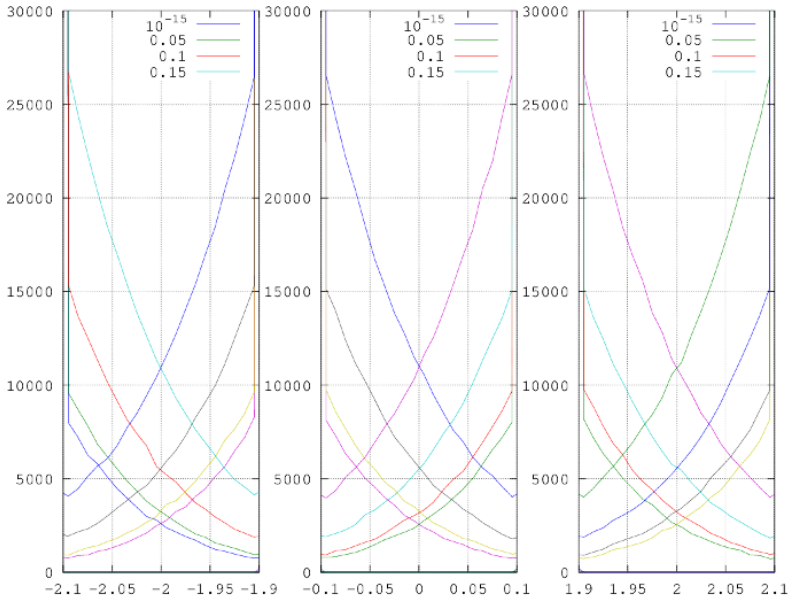

Figure 5: PDF for 4-PAM, $\omega_0 \in -3, -1, 1, 2$, 10^8 samples