Workshop 2

The aim of workshops is open time to work on, and get help progressing your project. The format for the workshop is not fixed, but each week I would like you to follow the advice I would give any research student:

1. **read** something;
2. **do** something; and
3. **write** something.

In the previous weeks we looked at sinusoids. On of the topics touched on was that even though the sinusoids themselves are deterministic, we can generate functions that look random. But purely random signals aren't that interesting. Often we want some structure as well.

A common example is in generating *textures*. In computer generated images (CGI) we often want to put realistic textures onto objects, or generate parts of a scene such as clouds which are "noisy".

On appealing approach to this problem is called $1/f$ noise (or sometimes "pink" noise[1]), because the amplitude (or power) of the signal in the frequency domain (on average) is divided by the frequency.

Please read a little more about pink noise at

- https://en.wikipedia.org/wiki/Pink_noise
- http://mansquito.com/pages/what-is-pink-noise.html
- https://www.bhphotovideo.com/explora/content/pink-noise-and-rhythms-our-brains

We can generate such noise (approximately) by various approaches, but one appealing way in MATLAB is to start by generating a Fourier domain signal, and transforming it back to the time domain.

Start by generating a completely random Fourier transform, using Gaussian variables, and remembering also to have random *phase*, *i.e.,* the frequency content is described by complex numbers, distributed randomly around the unit circle in the complex plane, *e.g.,*

```
N = 256;                    % the number of frequencies to generate
magnitude = randn(1, N/2); % Normal (Gaussian) random numbers
phase = 2*pi*rand(1, N/2); % uniformly random phases
f_half = magnitude .* exp( i*phase ); % complex values for frequency components
```

If we transformed this into the time domain, we would get "white" noise, so called because of the analogy to white light which is (incorrectly) thought of as equally composed of all visible light frequencies.

To make pink noise we filter so that the magnitude squared is inversely proportional to frequency:

```
f_half = f_half ./ sqrt(1:N/2);
```

Transforming back has two steps: the first is to make sure we have a *Hermitian* signal, which we do by pasting a second copy of the conjugate of `f_half` onto the first, but flipped around (also making sure we put a 0 into the zero frequency slot. The second step is just to take the inverse Fourier transform

[1] See Colours of Noise.

```
f_full = [0, f_half, fliplr(conj(f_half))];
x = real(ifft(f_full))
```

We shouldn't need to use the `real` function here, but I like to make sure that the output is real in case there are any numerical errors.

Then plot the result. You should see something a bit like Figure 1, though it won't be the same because it is built using random numbers.

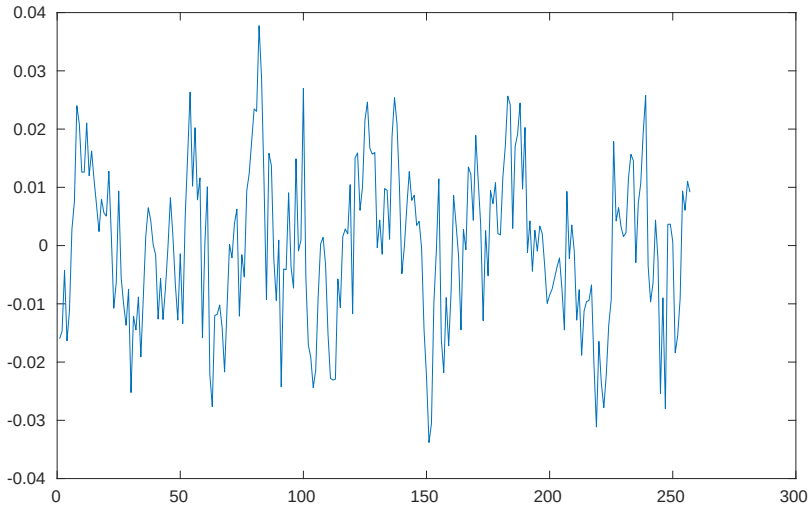

Figure 1: $1/f$ noise example.

Play around, increasing the value of N to see how it changes. Note that it is random, but not completely – there seem to be patterns in the randomness. Perhaps the result looks something like the profile of a mountain range?

Now

1. Build that code into a function, and have a play with it. Look up how to set random "seeds", *e.g.,* see au.mathworks.com/help/matlab/examples/controlling-random-number-generation.html, and au.mathworks.com/help/matlab/math/generate-random-numbers-that-are-repeatable.html so that you can create repeatable patterns of random numbers.

2. Extend the code to allow $1/f^{\alpha}$ noise. That is, change the filtering function so that it has a *configurable* parameter α as an extra input the function. Experiment with other values of α between 0.0 and 2.0. The former is "white" noise, and the latter "brown" noise (Brownian motion). You can explore even further too.

3. Download the code `one_on_f_noise2.m` from my web page, which will generate a 2D version of the above. You can plot the results using the `surf` function, and interactively rotate them to get the best view.

4. A texture map can do more that this. It generates a noise field, which can be used to alter other maps to generate textures such as those seen in marble and wood.

You **do not** have to hand anything up. However, you should not limit your exploration of this topic to 50 minutes. You should expect (in the long run) that you will need to put in a couple of hours work (towards your project) for each workshop. But today we are just starting out.

You can also ask questions about your practical, or any other aspects of the course.