# 1 Correlated Noise

If we start with our usual uncorrelated expression for $\chi^2$, we have

$$\chi^2 = r^T \mathrm{N}^{-1} r$$

where $r = d - A(m)$, the residual between our model and data. In this case, the noise matrix N is diagonal, and we know that $\langle r_i^2 \rangle = \mathrm{N}_{ii}$. Now let's introduce an orthogonal matrix V so $\mathrm{V}\mathrm{V}^T = \mathrm{V}^T\mathrm{V} = \mathrm{I}$. Without changing $\chi^2$, we can add a few copies of V into $\chi^2$:

$$\chi^2 = r^T \mathrm{V}^T \mathrm{V}\mathrm{N}^{-1}\mathrm{V}^T\mathrm{V} r$$

We can rewrite this as

$$\chi^2 = (Vr)^T (\mathrm{V}\mathrm{N}\mathrm{V}^T)^{-1} (Vr)$$

Let's make the definitions $\tilde{r} \equiv \mathrm{V}r$ and $\tilde{\mathrm{N}} \equiv \mathrm{V}\mathrm{N}\mathrm{V}^T$. That leaves:

$$\chi^2 = \tilde{r}\tilde{\mathrm{N}}^{-1}\tilde{r}$$

which looks just like the expression we started with, but now our rotation means that $\tilde{\mathrm{N}}$ is no longer diagonal. By inspection, we can see that the eigenvalues and eigenvectors of $\tilde{\mathrm{N}}$ are the diagonal entries of N and V, respectively.

Fortunately, we can work out what $\tilde{\mathrm{N}}$ looks like. Let's start by taking $\langle \tilde{r}_i \tilde{r}_j \rangle$. We know that $\tilde{r}_i = \mathrm{V}_{:,i}^T r$, the $i^{th}$ column of V dotted against the residual $r$. We can now write the expectation as $\left\langle \mathrm{V}_{:,i}^T r r^T \mathrm{V}_{:,j} \right\rangle$. We know what $\left\langle r r^T \right\rangle$ is, though - it's just N. That means $\langle \tilde{r}_i \tilde{r}_j \rangle = \mathrm{V}_{:,i}^T \mathrm{N} \mathrm{V}_{:,j}$. If you look at $\tilde{\mathrm{N}}_{ij}$, though, it is exactly this expression.

We now have what we need to write down $\chi^2$ if the noise in our data points is Gaussian, but correlated between data points. We can keep our usual expression for $\chi^2$, but we replace our diagonal copy of N by a non-diagonal N where $N_{ij} = \langle r_i r_j \rangle$. We got here by starting with a diagonal model, but if we know how to write down $\langle r_i r_j \rangle$, we can calculate $\chi^2$ from our non-diagonal N. Everything we've done to *e.g.* model-fit, estimate error bars, *etc.* carries straight through as long as we use the correct N.

# 2 Sampling from a Covariance Matrix

Let's say we had a correlated noise matrix and wanted to create a realization of what the noise might look like? There are many reasons you might want to do this, say Monte Carlo simulations of data. One case of particular interest is when carrying out Markov Chain Monte-Carlo (MCMC) parameter estimation. It's quite natural for an MCMC to give you a covariance matrix between parameters, and as we'll see we can make the MCMC run much faster if our sample steps match up with this covariance matrix.

For a diagonal noise/covariance matrix, it's straightforward to generate realizations of $r$, since $r_i$ is just a Gaussian number with zero mean and variance $\mathrm{N}_{ii}^{1/2}$. We know that $\tilde{r} = \mathrm{V}r$, so we can generate a realization of $\tilde{r}$ by generating a realization of $r$ and multiplying by V.

The eigenvalues/eigenvectors of our covariance matrix are one way to generate realizations of correlated noise, but we can do even better. Rather than using an orthogonal matrix V to rotate a diagonal noise matrix, we could instead start with a noise matrix of I, and a lower-triangular matrix L as the basis of our rotation. If we carry that through, you'll see that $\tilde{r} = \mathrm{L}r$ where $r$ now has $\sigma = 1$, and $\tilde{\mathrm{N}} = \mathrm{L}\mathrm{L}^T$. Cholesky runs $\sim$15 times faster than eigh on my laptop, so if you have a large matrix, using Cholesky will be much faster than eigenvalues. You are free to choose either method, though.

# 3 Stationary Noise

One special but common case is where $\mathrm{N}_{ij}$ is a function only of $i - j$. In this case, it turns out that the to multiply the data by the eigenvector matrix, you just need to Fourier transform the data. The eigenvalues are just the average of the square of the Fourier transform (which we call the power spectrum). If you're careful with phases and normalizations, you can generate a random vector, scale by the square root of the power spectrum, and take the inverse Fourier transform. Since, as we'll see, the Fast Fourier transform (FFT) runs in order $n \log(n)$ time, as opposed to a direct matrix cost of $n^3$, we can run to very much larger problems this way.

Finally, the discrete Fourier transform (DFT), wraps the end of our data around to the front. This is often undesirable - a common case would be taking a piece out of a much longer data stream. If you have a matrix that does not wrap around, you can either factor it directly (look up Toeplitz matrices - this can be done in $n^2$ instead of $n^3$), or you can generate a much longer data series that does wrap around using the DFT, and snip out a piece. This isn't exactly correct, but becomes so in the limit of the whole series being much longer than our desired piece. In practice, the DFT is so much faster (and, it turns out, numerically more stable as well) that I am hard pressed to think of a case where I would generate correlated data with a Toeplitz instead of a long FFT.