SF1: Data Analysis - Statistical Signal Processing with Application in Audio

Final Report

Joseph Johnson

June 2021

Contents

1 Introduction

This final report explores parametric models in the context of audio restoration. Some important time-series models are studied, including simple polynomial models and the auto-regressive model. The majority of the theory utilises the General Linear Model (GLM) to infer parameters from data in both Frequentist and Bayesian interpretations. It extends into Bayesian model choice for finding signals buried in noise and, finally, restoring lost data in audio.

2 Week 3

2.1 Task 1

Week 3 begins with the simple case of a signal at an unknown dc level, θ, corrupted by noise. Throughout the week 3 work we will refer to the model parameters as θ, which in general represents a set of model parameters: $\theta = \{\theta_1, \theta_2, ..., \theta_N\}$. To sample a set of observations from this model we generate a single value of θ from a Gaussian distribution and corrupt this value N times, with N independent zero-mean Gaussian noise samples of variance σ_e^2. This forms a length N vector of observations. The GLM representation of our model is:

$$X = \left[X_1, X_2, \ldots, X_N\right]^T = \left[1, 1, \ldots, 1\right]^T \theta + \left[e_1, e_2, \ldots, e_N\right]^T$$
$$G = \left[1, 1, \ldots, 1\right]^T$$

The extensive notes in the course handout detail how to formulate maximum likelihood (ML) and maximum a posteriori (MAP) parameter estimates from our matrix G (and any priors on the parameters). Thus, performing inference is as simple as substituting this particular value of G into the expressions.

To start, $G^T X = \sum_n X_n$ and $G^T G = N$, so $(G^T G)^{-1} = \frac{1}{N}$. Then:

$$\theta^{ML} = (G^T G)^{-1} G^T X = \frac{1}{N}\sum_n X_n$$

and for the posterior, with prior parameters $m_\theta, \sigma_\theta^2$, such that $\Phi = N + \frac{\sigma_e^2}{\sigma_\theta^2}$, $\Theta = \sum_n X_n + \frac{\sigma_e^2}{\sigma_\theta^2} m_\theta$, then:

$$\mu_{post} = \frac{\sigma_\theta^2 \sum_n X_n + m_\theta \sigma_e^2}{N\sigma_\theta^2 + \sigma_e^2} \quad \text{and} \quad \sigma_{post}^2 = \frac{\sigma_e^2 \sigma_\theta^2}{N\sigma_\theta^2 + \sigma_e^2}$$

The MAP estimate is then simply $\theta^{MAP} = \mu_{post}$ and the posterior density is $p(\theta|X) = \mathcal{N}(\theta; \mu_{post}, \sigma_{post}^2)$.

In order to express a lack of prior knowledge about θ we let $\sigma_\theta^2 \to \infty$. It is easy to see in this case that $\mu_{post} \to \frac{1}{N}\sum_n X_n = \theta^{ML}$ and $\sigma_{post}^2 \to \frac{\sigma_e^2}{N}$. Therefore, for large prior variances the posterior density is a Gaussian centred at the ML solution, which becomes a point mass if $N \to \infty$.

To confirm this tends in distribution to the likelihood function we can use the Gaussian property of the noise to write:

$$p(X|\theta) \propto \exp(-\frac{1}{2\sigma_e^2}\sum_n (X_n - \theta)^2) \propto \exp(-\frac{1}{2\sigma_e^2}(\sum_n \theta^2 - 2X_n\theta))$$
$$= \exp(-\frac{1}{2\sigma_e^2}(N\theta^2 - 2\theta\sum_n X_n)$$

and so

$$p(X|\theta) = \mathcal{N}(\theta; \frac{1}{N}\sum_n X_n, \frac{\sigma_e^2}{N}) = \lim_{\sigma_\theta^2 \to \infty} p(\theta|X)$$

Figures (1) through (5) show plots of the prior, likelihood and posterior densities for a range of parameter values. Figure (1) demonstrates how an increasing number of observations reduces the variance of the likelihood, causing it to tend to a point mass. In turn, this drags the posterior density closer to the likelihood until the two are almost identical. Figure (2) shows the same thing, except the noise variance is greater meaning we now require more observations for the likelihood to dominate. Figure (3) shows this further, by increasing the noise variance until our likelihood function is almost flat - at which point the posterior tends to the prior distribution. Figure (4) shows the effect of changing the posterior mean. It simply changes the position of the prior density, and thus the position of the posterior. It does not impact whether the posterior favours the prior or likelihood. Finally, figure (5) shows an increasing prior variance. For very narrow priors, the posterior mostly ignores the likelihood, whilst for very wide (i.e. flat) priors, the posterior ignores the prior and tends to the likelihood.

2.2 Task 2

Task 2 considers a simple two parameter model with a dc and a trend component:

$$x_n = \theta_1 + n\theta_2 + e_n$$

The generator matrix, G, for this model is of the form:

$$G = \begin{bmatrix} 1, 1, \ldots, 1 \\ 0, 1, \ldots, N-1 \end{bmatrix}^T$$

For some example datasets, θ_1 and θ_2 were drawn from unit Gaussians and corrupted with unit Gaussian noise. The result was a jagged straight line. The value of the slope parameter θ_2 directly affected how jagged the line was (for constant noise variance) and also whether the line sloped upwards or downwards. Code for building G and finding the ML and Bayesian estimates is shown in figure (6). Three example datasets are shown in figure (7), along with three standard deviation confidence intervals for the Bayesian predictive distribution. The corresponding ML and MAP estimates are shown in figure (8). The confidence intervals tend to be wider for the smaller values of θ_2 which shows how the Bayesian approach has incorporated the uncertainty involved with the weaker SNR into the posterior distribution. MAP and ML estimates of θ_1 and θ_2 are shown in table (8).

2.3 Task 3

We now explore Bayesian model choice as a method for comparing potential models' ability to describe a given dataset. The approach is based on the model evidence $p(x|\mathcal{M}_i)$, which is the normalising constant of the posterior density in the Bayesian framework. Three models are considered here: pure Gaussian noise; an offset dc parameter with additive Gaussian noise; plus a linear trend model as in task 2. To test, one dataset was generated from each model, with constant noise variance $\sigma_e^2 = 1$. The resulting datasets are shown in figure (9). Table (10) contains the resulting log-model evidences for each potential model. The model evidence for the no parameter case, $p(X|\mathcal{M}_1)$, is simply the likelihood function of the data conditioned on the model choice, i.e. $\prod_n p_e(x_n)$, which is just Gaussian:

$$\log p(X|\mathcal{M}_1) = -\frac{N}{2}\log(2\pi\sigma_e^2) - \frac{1}{2\sigma_e^2}\sum_{n=0}^{N-1} x_n^2$$

The other two evidences were easily calculated using the result in the handout for the case of a Gaussian prior and a Gaussian likelihood.

Interestingly, the variance of the noise did not seem to impact the outcome significantly, provided we knew the true noise variance and could use it in our calculations. It affected the magnitude of the log-model evidence, but did so equally across potential models, meaning the true model was still the most favourable. Typically, poor estimates of the noise variance did have an impact on the results, and could lead to incorrect model choices in some cases. Models 1 and 2 were the most easily confused, especially for small values of θ where the datasets are almost the same. This method could differentiate model 3 data from the other two models very well.

If we wish to express a lack of prior knowledge about θ then we can let σ_θ^2 be very large. In this case, the model evidences for $\mathcal{M}_2$ and $\mathcal{M}_3$ tend to diverge to the value given by $\mathcal{M}_1$, i.e. the no parameter model. Therefore, our Bayesian approach is interpreting the diffuse prior as very strong evidence that our parameters are zero. The reason for this may be to do with how the posterior distribution tends to the likelihood function for diffuse priors, which in turn tends to a point mass for large enough N, which causes issues when we integrate over the full range of θ values for the model evidence.

2.4 Task 4

2.4.1 Bayesian model choice for signal detection

In task 4 we consider the problem of a known signal buried in noise at some unknown starting point, which we would like to infer. In this example we have a length 15 signal with an unknown offset, λ, with a discrete uniform distribution in $\lambda \in [0, 90]$. We also consider some attenuation or gain applied to the signal, γ, modelled by a unit Gaussian. For the purposes of inference we consider a set of 91 models $\Lambda = \{\mathcal{M}_\lambda : \lambda \in [0, 90]\}$ containing all possible offsets. Model $\mathcal{M}_\lambda$ then has parameter vector $\theta = \{\gamma\}$, and implicitly assumes the location of the offset. The model observations are then of the form:

$$y_n = \gamma x_n + e_n \quad \text{where} \quad x_n = \begin{cases} s_{n-\lambda} & \text{if} \quad \lambda \leq n \leq \lambda + 14 \\ 0 & \quad \text{otherwise} \end{cases}$$

where s_i is the i^{th} term in our signal sequence. This permits a GLM representation, for which the 100 observations under the model $\mathcal{M}_\lambda$ have G of the form:

$$G = \left[0, 0, \dots, s_0 \dots, s_{14}, \dots 0\right]^T$$

where s_0 starts at $n = \lambda$.

With this in hand it is easy to find the model evidence for $\mathcal{M}_\lambda$, and the posterior density for γ in each case using the results in the handout. The required expressions are $G^T G = \sum_{n=0}^{N-1} s_i^2$ and $G^T Y = \sum_{n=\lambda}^{\lambda+14} y_n s_{n-\lambda}$. The posterior is therefore Gaussian with:

$$\mu_{post} = \frac{\sigma_\gamma^2 \sum_{n=\lambda}^{\lambda+14} y_n s_{n-\lambda} + m_\gamma \sigma_e^2}{\sigma_\gamma^2 \sum_{i=0}^{14} s_i^2 + \sigma_e^2} \quad \text{and} \quad \sigma_{post}^2 = \frac{\sigma_e^2 \sigma_\gamma^2}{\sigma_\gamma^2 \sum_{i=0}^{14} s_i^2 + \sigma_e^2}$$

And the log model evidence is:

$$\log p(Y|\mathcal{M}_\lambda) = -\frac{N}{2}\log(2\pi) - \frac{N-1}{2}\log(\sigma_e^2) - \frac{1}{2}\log(\sigma_\gamma^2 \sum_{i=0}^{14} s_i^2 + \sigma_e^2) - \frac{1}{2\sigma_e^2}\left(\sum_{n=0}^{N-1} y_n^2 + m_\gamma^2 \frac{\sigma_e^2}{\sigma_\gamma^2} - \frac{(\sum_{n=\lambda}^{\lambda+14} y_n s_{n-\lambda} + m_\gamma \frac{\sigma_e^2}{\sigma_\gamma^2})^2}{\sum_{i=0}^{14} s_i^2 + \frac{\sigma_e^2}{\sigma_\gamma^2}}\right)$$

where $N = 100$.

For plotting purposes it will be useful to find the likelihood function. Again, from the GLM:

$$p(Y|\gamma, \mathcal{M}_\lambda) = \prod_{n=0}^{N-1} p_e(y_n - x_n) = \left(\prod_{n=0}^{\lambda-1} \mathcal{N}(y_n; 0, \sigma_e^2)\right)\left(\prod_{n=\lambda}^{\lambda+14} \mathcal{N}(y_n; \gamma s_{n-\lambda}, \sigma_e^2)\right)\left(\prod_{n=\lambda+15}^{N-1} \mathcal{N}(y_n; 0, \sigma_e^2)\right)$$

$$\propto \exp(-\frac{1}{2\sigma_e^2}(\sum_{n=0}^{\lambda-1} y_n^2 + \sum_{n=\lambda}^{\lambda+14}(y_n - \gamma s_{n-\lambda})^2 + \sum_{n=\lambda+15}^{N-1} y_n^2)$$

by expanding the middle term, we can write this as a function of γ as:

$$\propto \exp(-\frac{1}{2\sigma_e^2}(\gamma^2 \sum_{i=0}^{14} s_i^2 - 2\gamma \sum_{n=\lambda}^{\lambda+14} y_n s_{n-\lambda})$$

By comparing this to the standard Gaussian form, the likelihood as a function of γ is:

$$p(Y|\gamma, \mathcal{M}_\lambda) = \mathcal{N}(\gamma; \mu_L, \sigma_L^2) \quad \text{where} \quad \mu_L = \frac{\sum_{n=\lambda}^{\lambda+14} y_n s_{n-\lambda}}{\sum_{i=0}^{14} s_i^2} \quad \text{and} \quad \sigma_L^2 = \frac{\sigma_e^2}{\sum_{i=0}^{14} s_i^2}$$

We can also show by differentiation of the likelihood that $\gamma^{ML} = \mu_{post}$.

This can now be applied to the hidden signal. The process for sampling data was: generate values of γ and λ from appropriate distributions; then generate the shifted, noisy data. The details for dealing with offsets greater than 85 (where the signal is not fully transmitted due to the data only having length 100) are contained in the Week 3 jupyter notebook. With the expressions defined above we iterate over all possible offset values and calculate the model evidence, then select the most probable value of λ as:

$$\hat{\lambda} = \underset{\lambda}{\text{argmax}}\, p(Y|\mathcal{M}_\lambda)$$

We can then calculate ML and MAP estimates of γ at $\hat{\lambda}$.

One other case to consider is that of no signal at all, say $\mathcal{M}_\infty$. This model evidence is the same calculation as that of $\mathcal{M}_1$ in task 3 and so will not be detailed here. Code for calculations related to task 4 is shown in figure (14).

The resulting graphs and data are shown in figures (11) through (15). For small values of γ relative to the noise variance, the analysis was less reliable, as can be seen in figure (13). There is no single, distinct peak. However once γ was larger than about 0.2 or so, the analysis almost always correctly identified the correct λ. For the largest values, there was a very clear peak at the true λ value. This is all relative to the noise variance, so if the noise variance increases then the values of γ must also be larger to get the same reliability.

The model never favoured the null hypothesis of no signal over the alternative, even for the strongest attenuation values.

The model was robust against changes in the prior variance, almost always giving correct results for wide and narrow zero mean priors. However, if the prior mean was too far from the true value of γ then the model could sometimes give the wrong estimate of λ. I tried setting the prior mean to $-\gamma_{true}$ so that the prior was far away - this usually gave correct results if γ was less than about 1.5, meaning that the prior mean could be a distance of around 3.0 away from the true value and still provide reliable estimates. Naturally the best results came when using the true prior, but I found that for cases where the mean of the generating distribution for γ had a non zero mean (e.g. 1), a zero mean prior still worked reliably.

2.4.2 The matched filter

Another example of a signal detection method is the matched filter. The principle of this filter is to maximise SNR of the filter output at the beginning of the true signal. More of this detailed in appendix B.

3 Week 4

3.1 Task 5

3.1.1 The Autoregressive model

Week 4 is a longer task centred around the autoregressive (AR) model. We consider AR(P) models of the form:

$$X_n = \sum_{i=1}^{P} X_{n-i} + e_n$$

which have a matrix G that depends on the observed data points. Consequently, we cannot generate observations directly from G and must instead iteratively calculate them from some initial state. The code for this is shown in figure (16). Figures (17) through (19) show plots of three different AR processes. The first two plots demonstrate stable AR process with different characteristics: the first process exhibits oscillatory behaviour as the negative AR coefficients try to restore the system to zero; the second shows more sluggish AR behaviour as the system tries to stay roughly where it was previously. The third example is an unstable process as there is a filter pole with magnitude greater than 1. The system diverges to infinity as it is allowed to grow continuously. In all three cases the stability can be determined from the pole zero diagram.

The figures also show the squared DFT magnitudes for each process. As explained in 3F3, the DFT of the process will have peaks at the same frequency as the filter power spectrum, with some random perturbation. This is exactly what we see as the DFT plots have two peaks corresponding to conjugate pairs of filter poles. In the third case the poles are strictly real so the peaks are centered at zero. The instability of the filter means the peak magnitude is far greater than the random variations, so the plot appears smooth.

3.1.2 Parameter estimation

As stated in the previous section, G is dependent on the process observations. Therefore, once we have generated the process we can build G to be used for parameter estimation. G is an $(N-P)\times(P)$ matrix, which can be built as shown in figure (20). Once G is built, the estimation process is identical to previous tasks as we continue to use Gaussian noise and priors.

Maximum likelihood estimation is the most simple method since there is no need to specify the residual variance, which is unknown in the case of real audio. Therefore we might favour this method in later sections where we do not initially wish to iteratively estimate the residual variance. For now we can explore these methods without worrying about this as we specify the residual variance when generating the process. As shown in figure (21) the ML estimate returned a reasonable fit to the data, and is capable of following the general trend well. It also looks very similar to the MAP estimate when we use the true coefficient prior for estimation, which is to be expected as the number of datapoints increases.

These two methods fall short on the amplitude of the estimated process, and they tend to look like scaled down versions of the true process. I found that you could remedy this somewhat by adjusting the prior mean of the AR coefficients by hand. A larger prior mean lead to a process with a larger amplitude and so by trial and error it was possible to find a value of m_α that fit the data very well by eye. In the case of the process shown in figure (21) I was able to achieve a significant reduction in the mean square residual from roughly 1 (the true value used to generate to process) to nearly 0.35, well below the true residual value. This was in general very difficult to do though, and was highly sensitive to the specific sample and accuracy of the residual variance estimate. Therefore this prior tweaking method would in practice not be a viable method for modelling real audio.

In order to explore the behaviour of mean square residuals, the ML estimate was generated across a range of model orders and the mean square estimation error:

$$\frac{1}{N}\sum_{n=1}^{N}(X_n - \hat{X}_n^{ML})^2$$

was plotted. The results are in figure (22). There is an obvious 'knee' in the graph around the true model order, the error falls sharply as the true order is approached, beyond which it stays relatively constant - decreasing an insignificant amount compared to the increase in model complexity.

3.1.3 Bayesian model evidence

We proceed to examine the Bayesian model evidence method introduced in week 3 for these AR models. Again, as G is already defined, there is no difference in the code implementation of the log model evidence versus that of week 3. The required expression is given in the handout and the relevant code is in figure (23). Plots of the log model evidences for the $P = 30$ and $P = 150$ models are shown in figure (24). In these plots, the true prior parameters have been used and the residual variance is correct. The model evidence was generally close to a concave function, but it had many local optima meaning that optimising the model evidence is a very difficult problem. It was also very sensitive to the choice of parameters specified - if the residual variance was over-estimated then the model evidence was a decreasing function of model order; if the residual variance was under-estimated the model evidence was an increasing function of model order. Clearly the estimate of residual variance is very important. Further, there was a similar sensitivity to the prior parameters, for example a flat prior tended to push the maximum of the model order to higher values. Despite this method providing accurate estimates for correctly specified prior parameters, it is very difficult to apply this to real audio samples as tuning the three different parameters quickly becomes very difficult.

Given these findings, estimating model order from the residual error is likely more practical than Bayesian model choice in the later sections.

3.1.4 Real Audio

I initially tried applying the Bayesian model selection method to real audio samples, in order to confirm the findings of the previous section. I had difficulty doing this as the estimated model order typically was the largest value in the range of model orders I considered, no matter how large, due to insufficient accuracy in the prior parameters. I tried small and large values for the prior parameters and residual variance, as well as trying to hone in on the global optimum by narrowing the range slowly (similar to the golden section method), but this quite often just pushed to model orders in the thousands, or found a local optimum. I could not get iterative adjustments of the noise variance as suggested in the handout to work, the estimated model order still grew very large. The iterative method would not affect the parameter estimates insofar as the calculation does not depend directly on the residual variance, only the model order.

I had much more success by roughly estimating the model order from the residuals. Figure (25) shows log magnitude spectra of estimates of steady notes, transients, vowels and consonants for real audio extracts, along with the corresponding model order estimates. These were very short extracts of single sounds, so often the model orders were very small as there were no long term characteristics to capture that we might expect from a full audio excerpt. The estimated spectra matched the true spectra very well, capturing the majority of the well defined peaks. The roll-off was, however, slightly slower for the estimated spectrum meaning higher frequencies appeared more strongly in the estimated processes.

Steady notes (including vowels) had lower model orders, which seems reasonable given that they change less with time when compared to transients (and consonants). This must of course be taken as only approximate due to our model orders being so small, but it does support the conclusions of week 1's report.

3.1.5 Lost packet concealment

One useful application of these AR models is in lost packet concealment for tarnished audio signals. Here we consider signals in which degradation manifests as a continuous burst of zero signal level at random (or non-random) points. Due to the non-stationarity of long audio signals, we cannot simply apply one model to the whole signal and assume it will work well. As explained in previous reports, it may be valid to assume that the signals are stationary over short time intervals (e.g. 50ms). To test this I found the optimal model order for the long section of audio using the residuals method from the previous sections, then broke the audio up into short sections and trained an AR model of that order for each section. I then plotted a histogram of the AR coefficients to see if there was any commonality. Plots of histogram coefficients are shown in figure (26), in which there seems to be grouping of coefficients around a mean (note that in the plot some colours have been reused as there were too many coefficients. The important point is that each coefficient was closely distributed to others of the same order, despite being from different sections of audio). I therefore decided to proceed with only one model being trained for the whole sample, though with more time I would've liked to explore models in which I had a separate model for each chunk.

The main process for interpolation was: select an optimal model order (the model orders that appeared to be optimal were shorter than I had expected for longer audio samples); find the maximum likelihood forwards AR coefficients; find the maximum likelihood backwards AR coefficients; extract the missing sections to be interpolated; run the forwards and backwards AR models in prediction mode; then output each final prediction as a weighted sum of the forwards and backwards predictions.

I first tried fixing the two examples given as part of the project, for which the code is shown in figures (27) and (28). For `armst_37_orig.wav` the estimated optimal model order was roughly 55. I ran the interpolation process with and without random noise and decided to judge the provided examples qualitatively before doing some more quantitative analysis on my own examples. I found that the processed audio sounded better without noise: weak noise affected the result insignificantly, whilst strong noise degraded the quality further. I also tried models of higher and lower order than 55: shorter models were unable to remove the popping sound as well as P=55, longer models introduced their own distortion.

For the `grosse_original.wav` example I found that the residual variance settled around model order 77. I repeated the experiments as in the previous paragraph and achieved similar results. The processed examples for these files are included in my submission. The processed audio samples are definitely an improvement over the test examples, as the severity of the degradation is reduced - however it is still noticeable. Table (30) shows the reduction in MSE of each method.

In the more complicated case where the original audio sample is not available and parameters must be estimated from the missing data, the literature suggests the use of a Kalman filter to produce state estimates plus bounds for the missing sections[1]. This offers the opportunity for ML estimation methods which can be maximised through gradient ascent/descent or, more likely, the expectation maximisation algorithm.

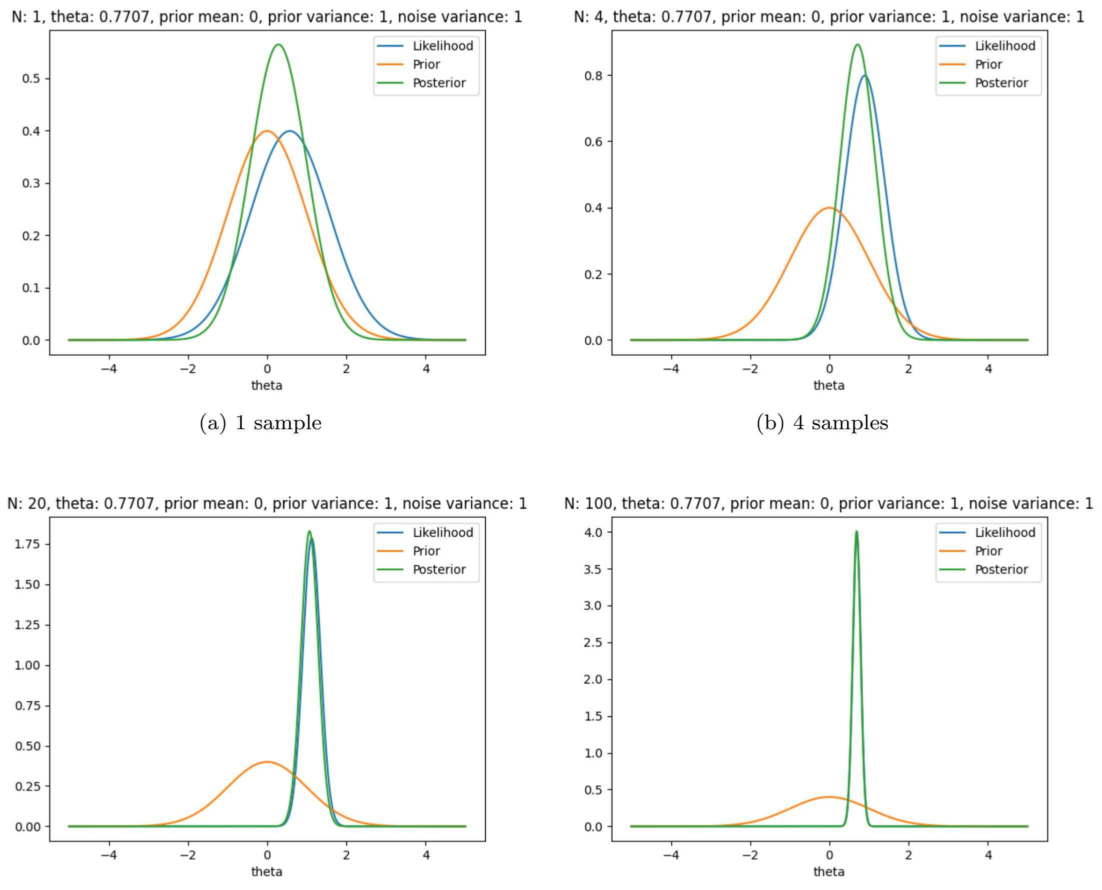
Some more quantitative results are shown in figure (31). This data was generated by removing successively longer chunks of audio, running the interpolation process and measuring mean squared error. Clearly, the ability of the process to reduce MSE is highly dependent on how much data is missing, and falls very quickly until it is almost insignificant at about 30% missing. The results also depended on how the noise was distributed. For example, at 10% missing data, the MSE reduction was better when the corruption was two spread out bursts of 50 missing samples every 1000 datapoints, vs one burst of 100. Further, the use of appropriately scaled noise could actually improve MSE, though I found it did not improve audible quality. Noise that was too weak did not change the graph whilst strong noise increased MSE.

[1] https://www.frontiersin.org/articles/10.3389/fbuil.2019.00109/full

4 Conclusion

In this report we have demonstrated an implementation and some results relating to important topics in parametric signal modelling. We have shown how different forms of signal degradation can be understood from a statistical perspective, and consequently improved. In particular, we have explored the wide ranging applications of the General Linear Model and shown how the same theory can be applied in several quite disparate scenarios. Each topic has been supported with appropriate python code as well as qualitative and quantitative results for comparing performance.

5 Appendix A: Figures



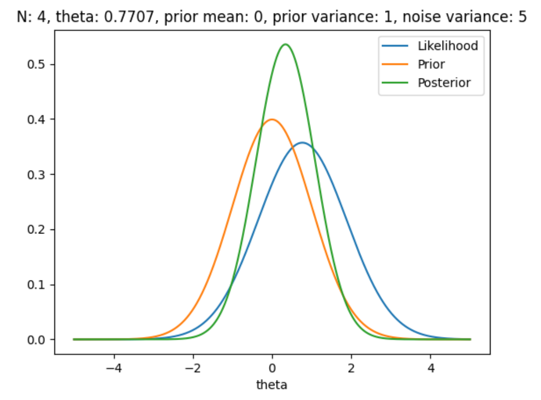
(a) 1 sample

(b) 4 samples

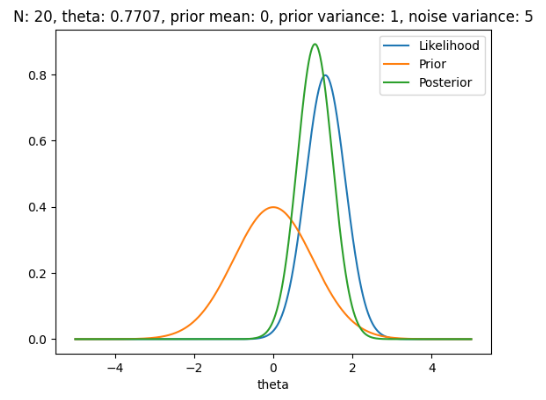
(c) 20 samples

(d) 100 samples

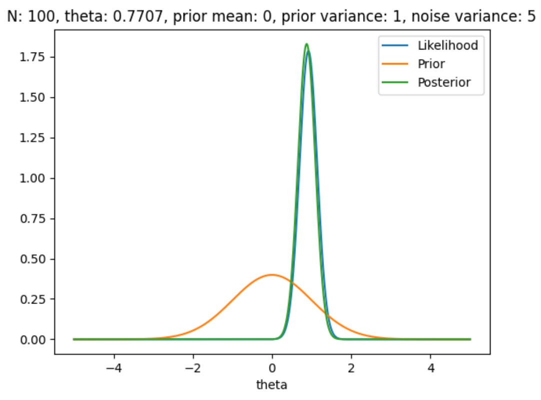
Figure 1: Effect of increasing the number of data samples in weak noise



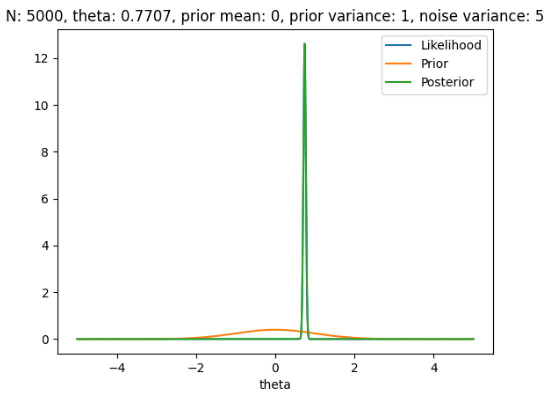
(a) 4 samples



(b) 20 samples

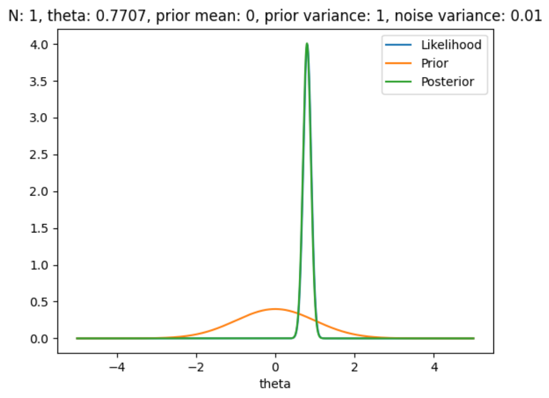


(c) 100 samples

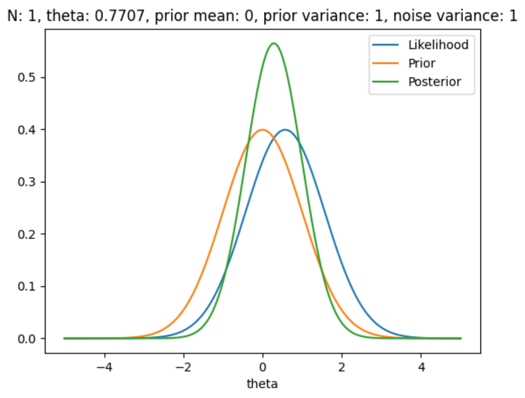


(d) 5000 samples

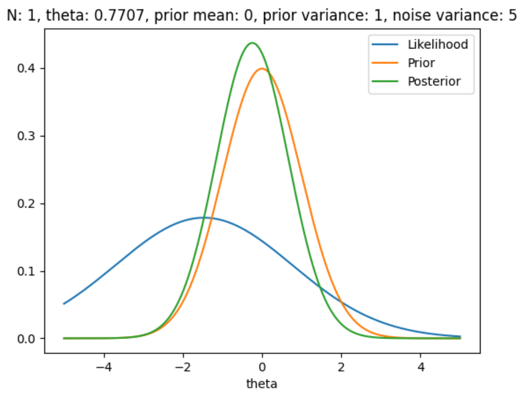
Figure 2: Effect of increasing the number of data samples in stronger noise



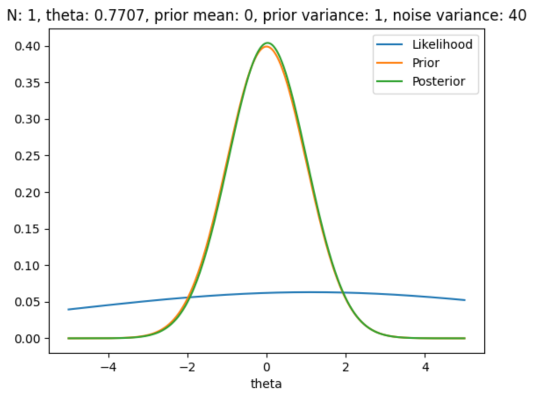
(a) Very weak noise



(b) Weak noise

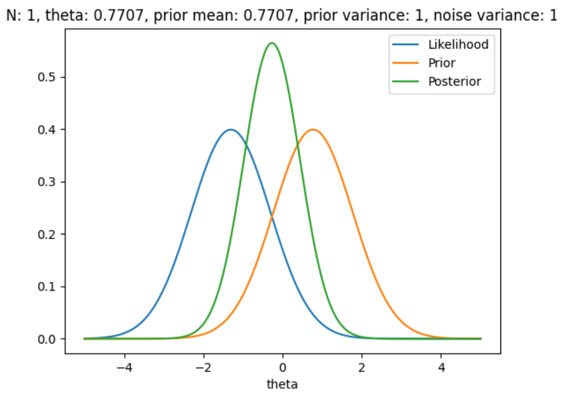


(c) Stronger noise

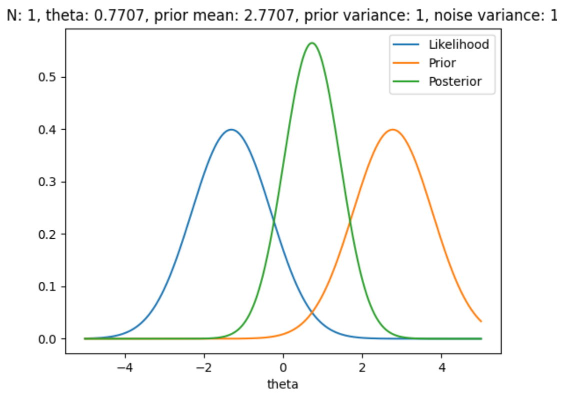


(d) Very strong noise

Figure 3: Effect of strengthening of noise on the likelihood

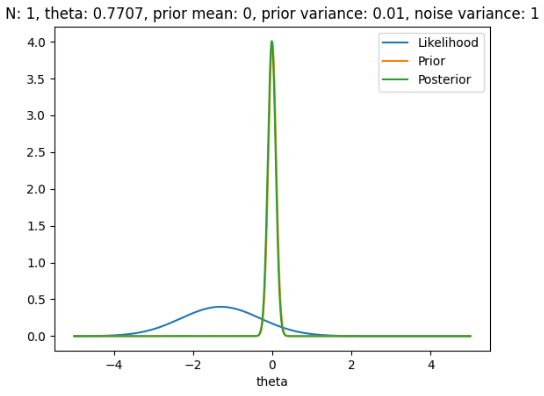


(a) Posterior mean at the true parameter value

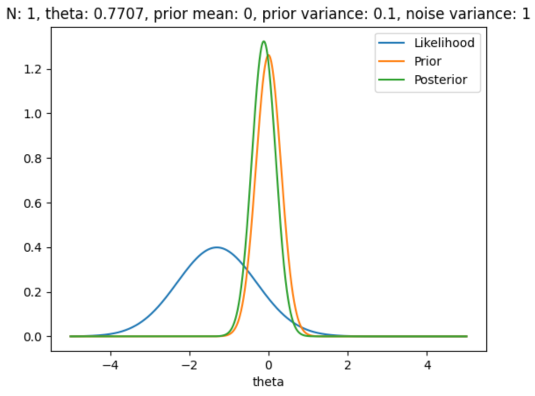


(b) Posterior mean above the true parameter value

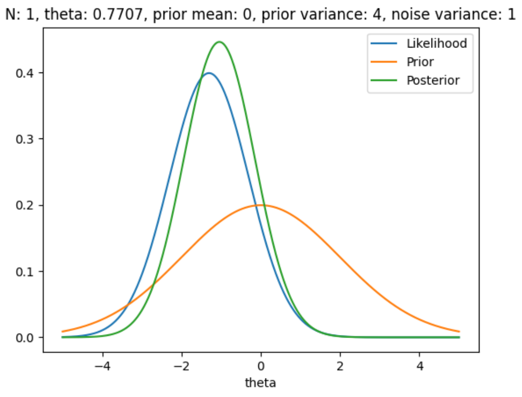
Figure 4: Effect of different posterior means



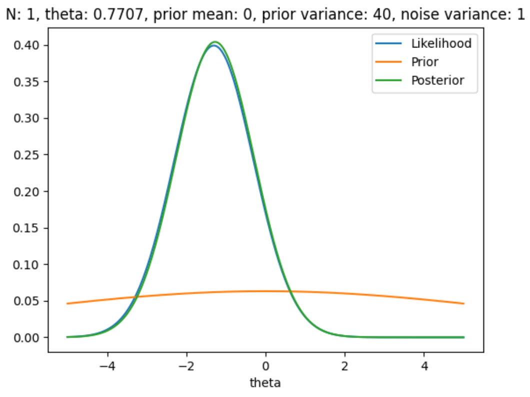
(a) Very narrow prior



(b) Narrow prior



(c) Wide prior

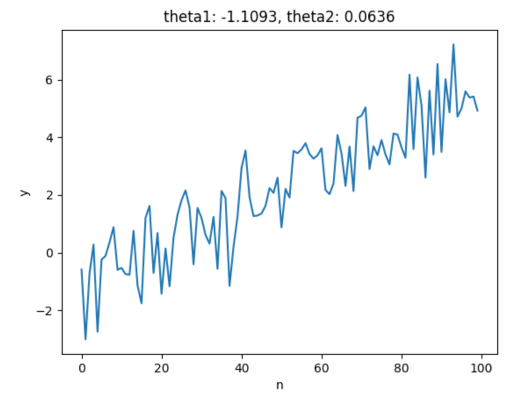


(d) Flat prior

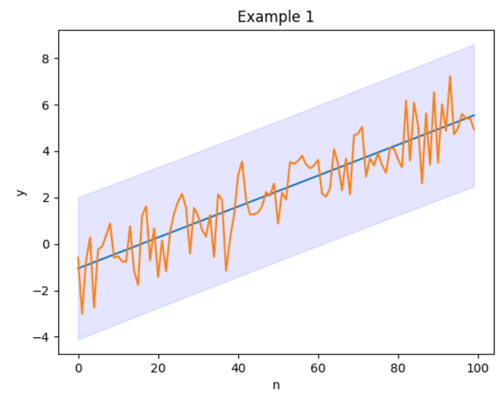
Figure 5: Effect of increasing prior variance

```
def polynomial_G(N_samps, N_params):
    # returns generator matrix G for a polynomial model with P coefficients and N
        observations
    G = np.zeros((N_samps, N_params))
    for i in range(N_params):
        G[:,i] = np.power(np.arange(N_samps), i)
    return G

def get_theta_ML(x, G):
    # least squares result
    return np.matmul(np.linalg.inv(np.matmul(G.T, G)), np.matmul(G.T, x))

def get_posterior(x, G, prior_mean, C_inv, noise_variance):
    # terms as in handout
    phi = np.matmul(G.T, G) + noise_variance * C_inv
    Theta = np.matmul(G.T, x) + noise_variance * np.matmul(C_inv, prior_mean)
    mean = np.matmul(np.linalg.inv(phi), Theta)
    covar = noise_variance * np.linalg.inv(phi)

    return mean, covar
```

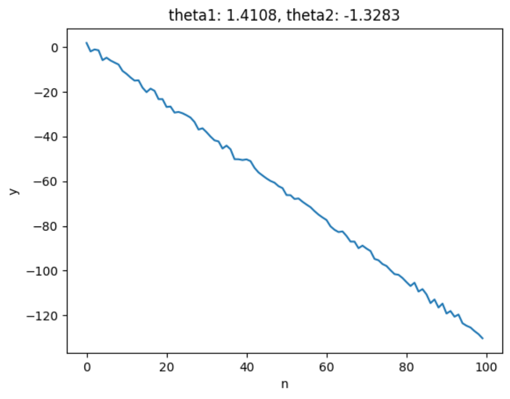
Figure 6: Python code for ML and Bayesian estimates



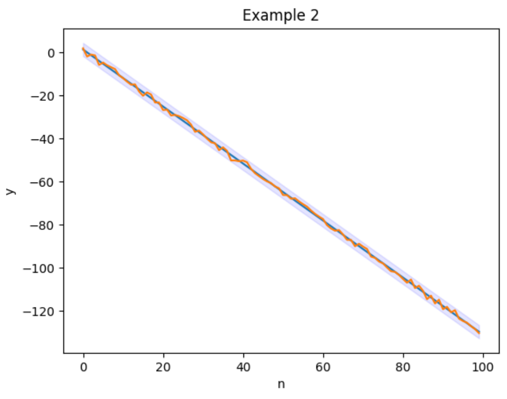
(a) Dataset 1



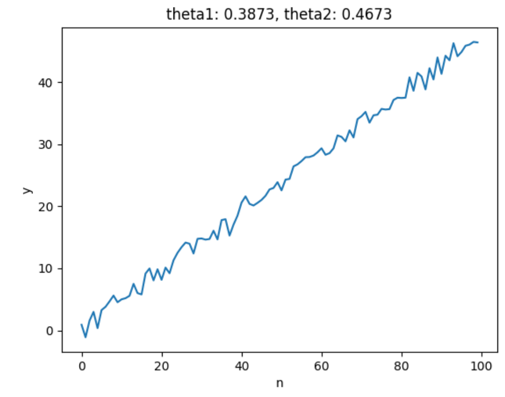
(b) Bayesian predictive distribution



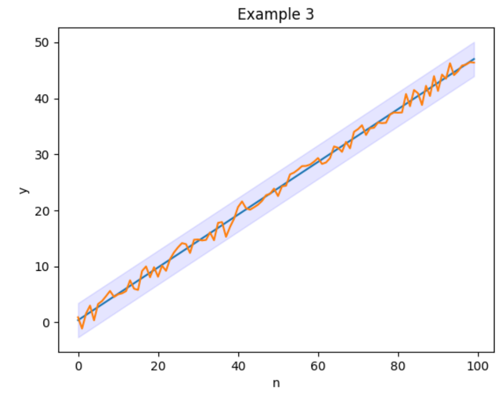
(c) Dataset 2



(d) Bayesian predictive distribution



(e) Dataset 3

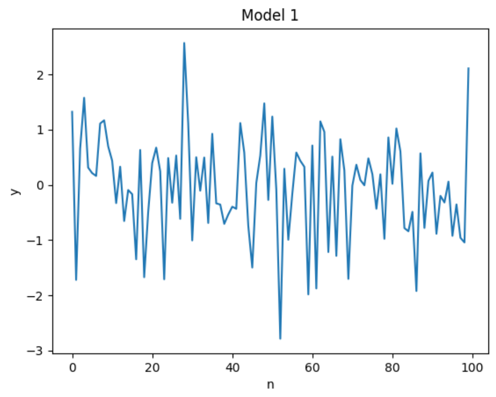


(f) Bayesian predictive distribution

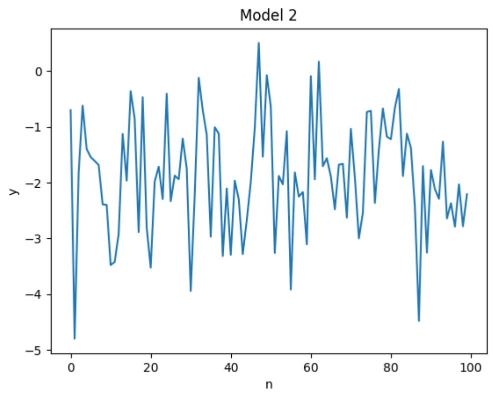
Figure 7: Linear trend datasets, plus three standard deviation confidence intervals

	True		Maximum Likelihood		Maximum a Posteriori	
Dataset	θ_1	θ_2	θ_1	θ_2	θ_1	θ_2
1	-1.1093	0.0636	-1.097	0.0672	-1.0553	0.0665
2	1.4108	-1.3282	1.423	-1.325	1.368	-1.3238
3	0.3873	0.4673	0.3997	0.4709	0.3848	0.4712

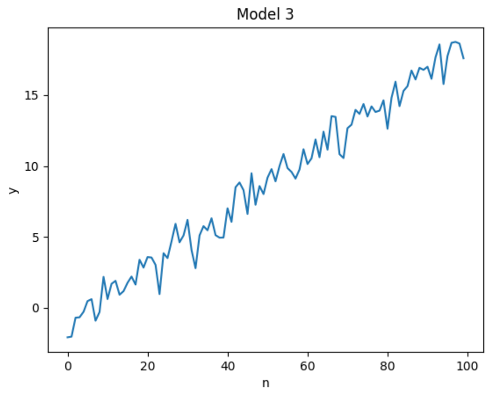
Figure 8: True, ML and MAP values of θ for three sets of one hundred samples



(a) Model 1



(b) Model 2

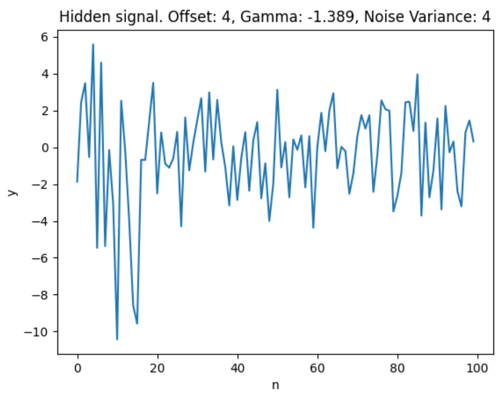


(c) Model 3

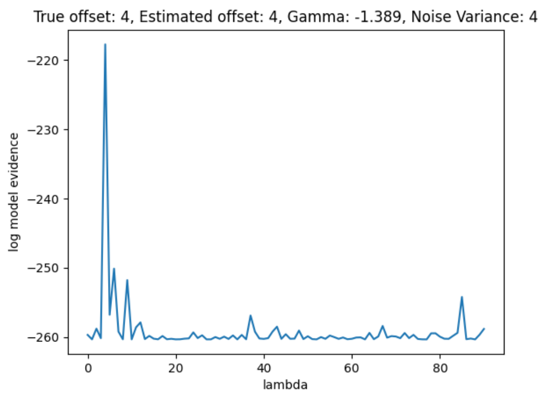
Figure 9: Datasets from each of the three models

True model	$\log p(X\|\mathcal{M}_1)$	$\log p(X\|\mathcal{M}_2)$	$\log p(X\|\mathcal{M}_3)$
$\mathcal{M}_1$	-135.51	-137.71	-142.41
$\mathcal{M}_2$	-321.70	-148.07	-153.72
$\mathcal{M}_3$	-5529.73	-1825.47	-140.78

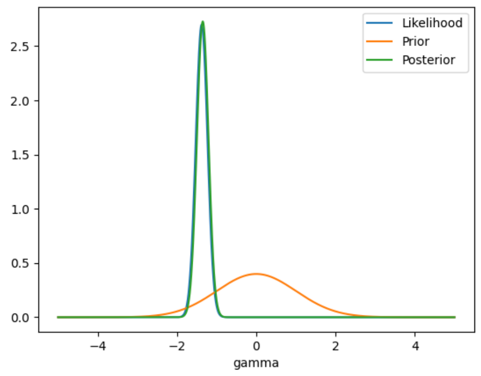
Figure 10: Log model evidences for data generated from each model



(a) Noisy hidden signal

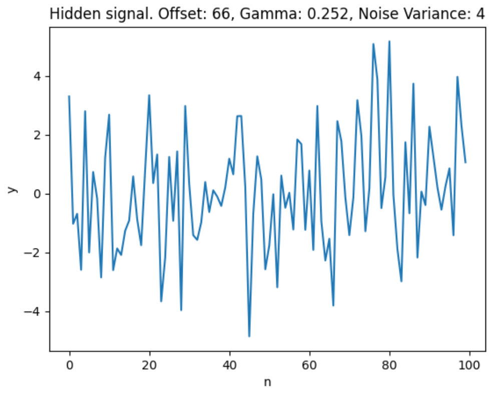


(b) Model evidences for all possible values of λ

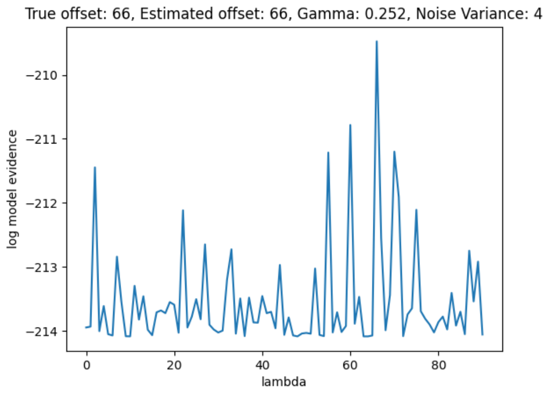


(c) Prior, likelihood and posterior

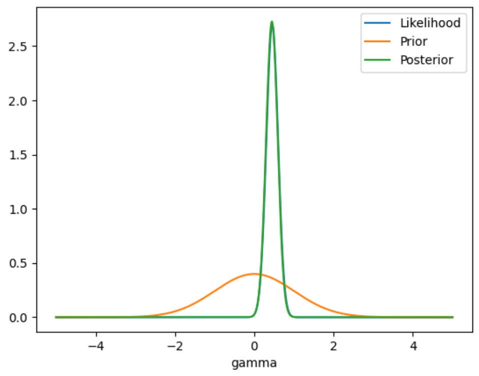
Figure 11: Large value of γ



(a) Noisy hidden signal

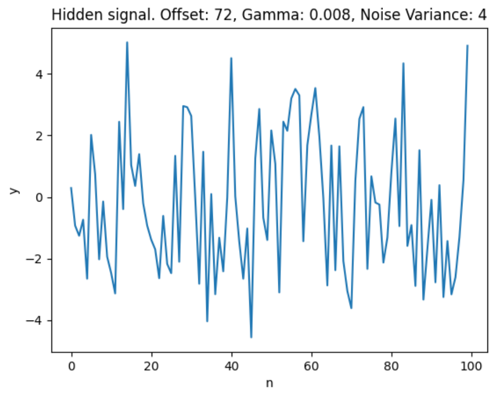


(b) Model evidences for all possible values of λ

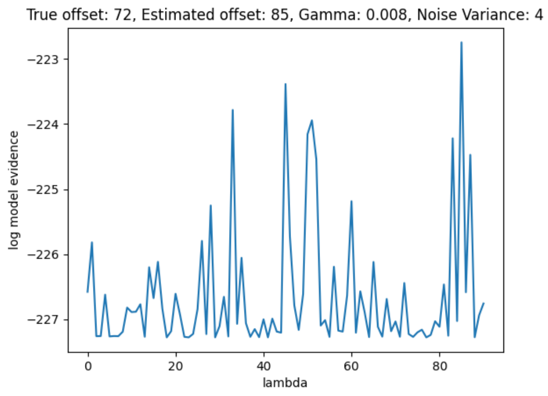


(c) Prior, likelihood and posterior

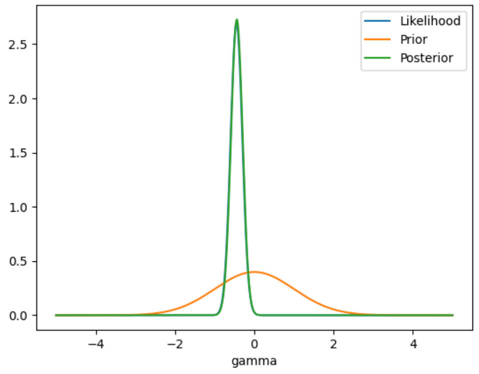
Figure 12: Moderate value of γ



(a) Noisy hidden signal



(b) Model evidences for all possible values of λ



(c) Prior, likelihood and posterior

Figure 13: Small value of γ

```
def get_log_model_evidence(data, lamb, sig_vec, noise_variance, prior_mean,
    prior_variance):
    # extra code to account for the case where the signal extends beyond the end of the
        observations
    Num_samps = data.shape[0]
    signal_shift = np.pad(sig_vec, (lamb, max(Num_samps-lamb-15, 0)))[:Num_samps]

    t1 = -Num_samps*np.log(2*np.pi)/2
    t2 = -(Num_samps-1)*np.log(noise_variance)/2
    t3 = -np.log(prior_variance*(np.sum(np.square(sig_vec))+noise_variance))/2

    t4 = np.sum(np.square(data))
    t5 = noise_variance*np.square(prior_mean)/prior_variance
    t6 = noise_variance*prior_mean/prior_variance + np.sum(data*signal_shift)
    t7 = np.sum(np.square(sig_vec)) + noise_variance/prior_variance

    return t1+t2+t3 - (t4+t5-(np.square(t6))/t7)/(2*noise_variance)

def get_log_null_hypothesis(data, noise_variance):
    Num_samps = data.shape[0]
    return -Num_samps*np.log(2*np.pi*noise_variance)/2 - np.sum(np.square(data))/(2*
        noise_variance)

def get_gamma_ML(data, lamb, sig_vec):
    Num_samps = data.shape[0]
    signal_shift = np.pad(sig_vec, (lamb, max(Num_samps-lamb-15, 0)))[:Num_samps]
    numer = np.sum(signal_shift*data)
    denom = np.sum(np.square(sig_vec))
    return numer/denom

def get_gamma_posterior(data, lamb, sig_vec, noise_variance, prior_mean, prior_variance):
    Num_samps = data.shape[0]
    signal_shift = np.pad(sig_vec, (lamb, max(Num_samps-lamb-15, 0)))[:Num_samps]

    pos_mea = (noise_variance*prior_mean/prior_variance + np.sum(data*signal_shift))/(np.
        sum(np.square(sig_vec)) + noise_variance/prior_variance)
    pos_var = (noise_variance)/(np.sum(np.square(sig_vec)) + noise_variance/
        prior_variance)
    return pos_mea, pos_var


def get_gamma_likelihood(data, lamb, sig_vec, noise_variance):
    Num_samps = data.shape[0]
    signal_shift = np.pad(sig_vec, (lamb, max(Num_samps-lamb-15, 0)))[:Num_samps]

    lik_mea = np.sum(data*signal_shift) / np.sum(np.square(sig_vec))
    lik_var = noise_variance / np.sum(np.square(sig_vec))
    return lik_mea, lik_var
```

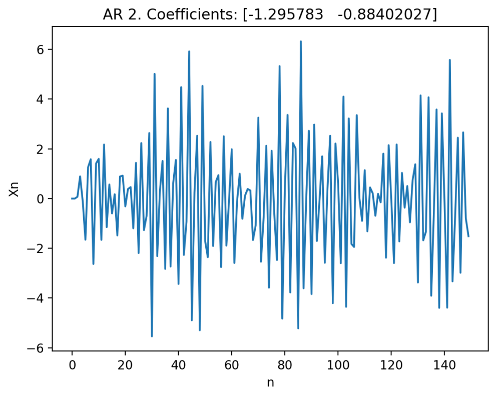
Figure 14: Python code for calculations in task 4

	γ			Model evidence		
Dataset	True	ML	MAP	λ_{true}	$\lambda_{estimate}$	Null
1	-1.3888	-1.3801	-1.3506	-217.7061	-217.7061	-320.9814
2	0.2520	0.4536	0.4439	-209.4769	-209.4769	-228.4558
3	0.0080	-0.4500	-0.4403	-226.4427	-222.7458	-254.84527

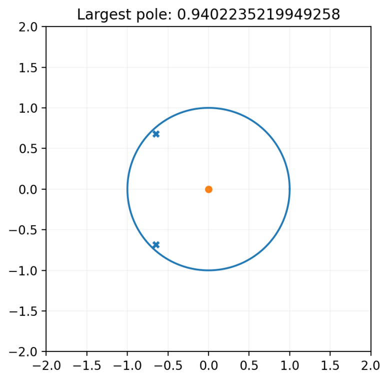
Figure 15: Estimates of γ and model evidences. When the model evidences for λ_{true} and $\lambda_{estimate}$ are equal, the analysis gave a correct estimate

```python
def AR(alpha, sigma, num, is_init_vals=False, initial_values=None):
    # order of AR model
    P = alpha.shape[0]
    # observation vector
    data = np.zeros(num)
    # first P values must be defined for the model to work correctly
    # default to zero initial values
    if is_init_vals:
        data[:P] = initial_values
    else:
        data[:P] = np.zeros(P)
    # iterate over all data points following the initial values
    for i in range(P, num):
        term = 0
        # sum over all P coefficients
        for j in range(1, P+1):
            term += alpha[j-1]*data[i-j]
        # add innovation term
        term += sigma*np.random.normal(size=1)
        data[i] = term
    return data
```

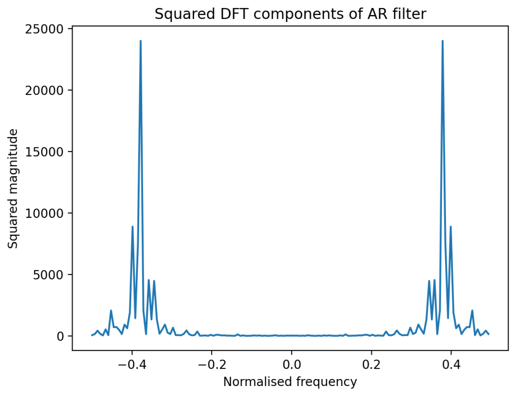
Figure 16: Python code to iteratively generate the AR(P) model



(a) Generated data

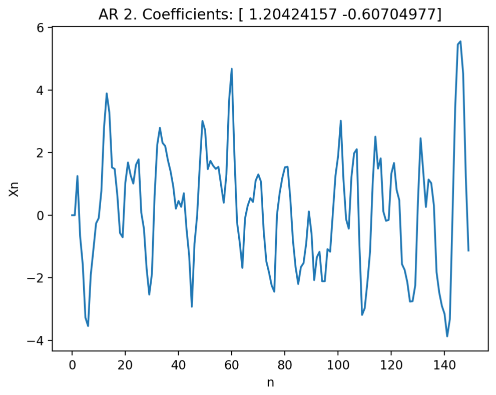


(b) Filter poles

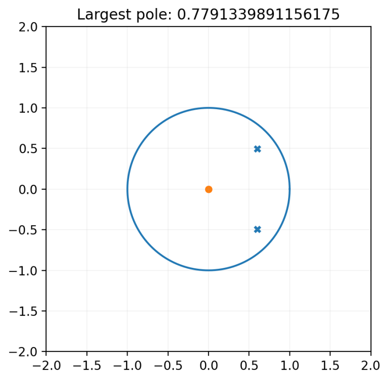


(c) Filter spectrum

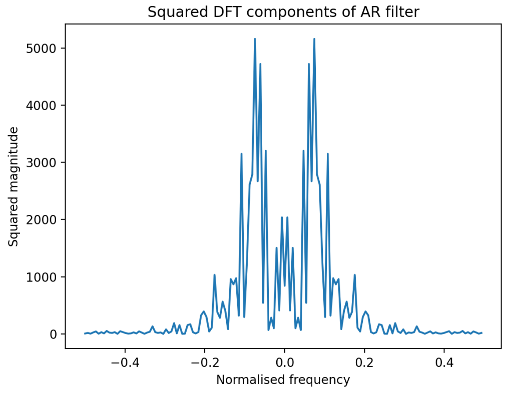
Figure 17: Oscillating behaviour for negative AR(2) coefficients



(a) Generated data

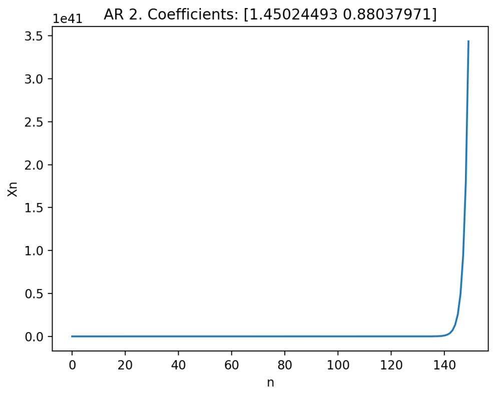


(b) Filter poles

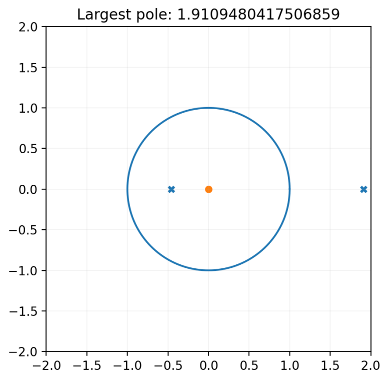


(c) Filter spectrum

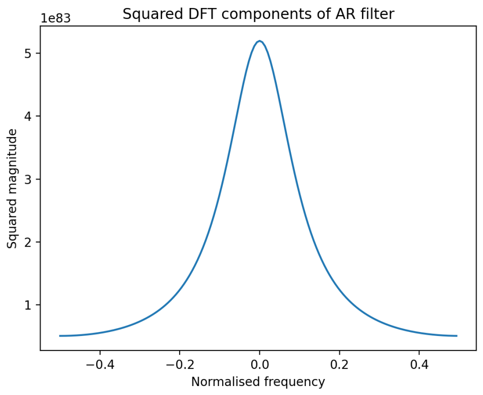
Figure 18: Sluggish behaviour for positive AR(2) coefficients



(a) Generated data



(b) Filter poles



(c) Filter spectrum

Figure 19: Divergent behaviour for unstable process

```
def build_AR_matrix(data, p):
    # retrieve n for matrix dimension
    n = data.shape[0]
    # initialise G as (n-p)x(p) matrix
    mat = np.zeros((n-p,p))
    # build matrix column-wise, taking observations from data
    for i in range(p):
        mat[:,i] = data[p-i-1:n-i-1]
    return mat
```

Figure 20: Python function for building G

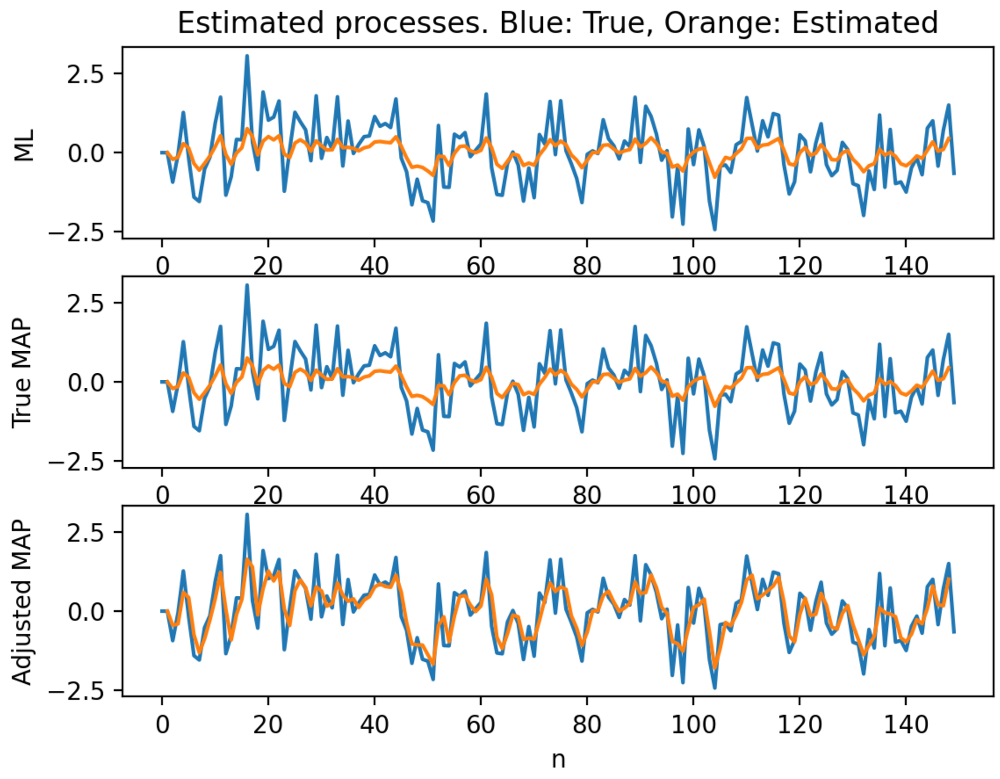
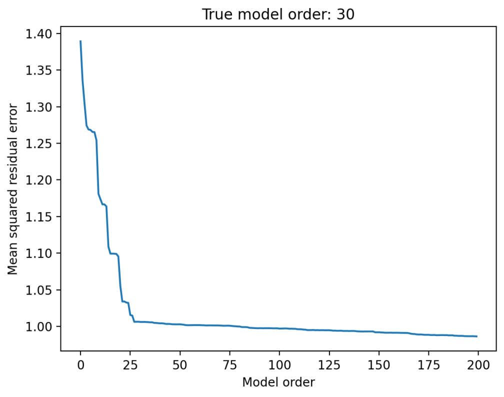
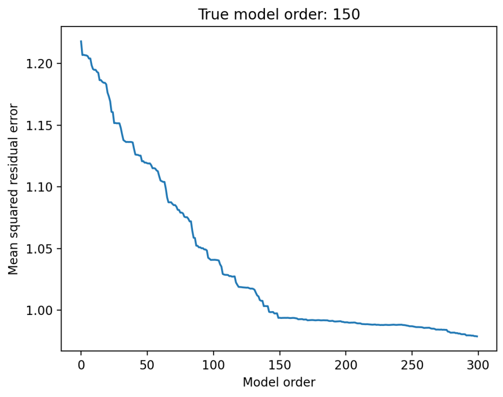


Figure 21: Estimated process for three different cases. Top: ML, Middle: MAP with the true prior, Bottom: Tweaked MAP estimate



(a) True order = 30

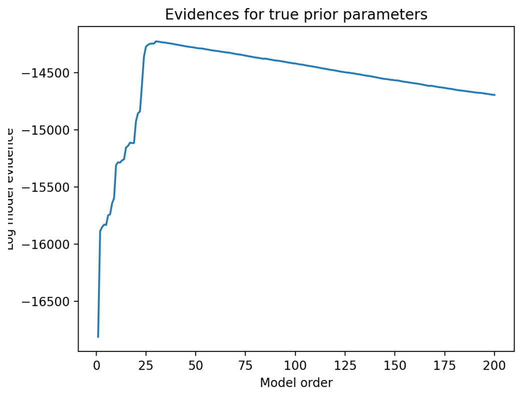


(b) True order = 150

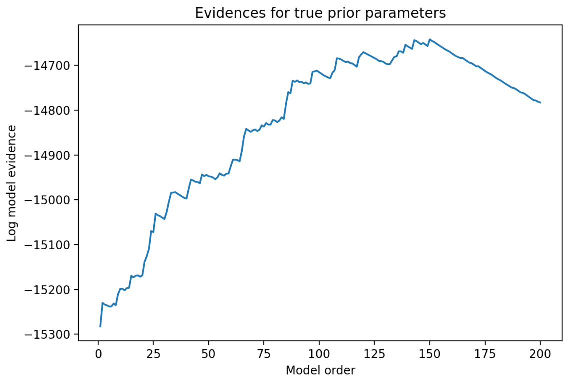
Figure 22: Squared errors for varying estimated model order

```
def get_log_model_evidence(data, noise_variance, prior_mean, inv_prior_covar, Gen):
    p_est = Gen.shape[1]
    data_trunc = data[p_est:]
    Num_samps = data_trunc.shape[0]
    phi = np.matmul(Gen.T, Gen) + noise_variance * inv_prior_covar
    theta = np.matmul(Gen.T, data_trunc) + noise_variance * np.matmul(inv_prior_covar,
        prior_mean)
    th_map = np.matmul(np.linalg.inv(phi), theta)
    sign_C, log_det_C_inv = np.linalg.slogdet(inv_prior_covar)
    sign_P, log_det_phi = np.linalg.slogdet(phi)

    t1 = -Num_samps*np.log(2*np.pi)/2
    t2 = sign_C*log_det_C_inv/2
    t3 = -sign_P*log_det_phi/2
    t4 = -(Num_samps-2)*np.log(noise_variance)/2

    t5 = np.sum(np.square(data_trunc))
    t6 = noise_variance * np.linalg.multi_dot((prior_mean, inv_prior_covar, prior_mean))
    t7 = np.matmul(theta.T, th_map)


    return t1+t2+t3+t4 - (t5+t6-t7)/(2*noise_variance)
```

Figure 23: Python code for calculation of task 5 model evidences

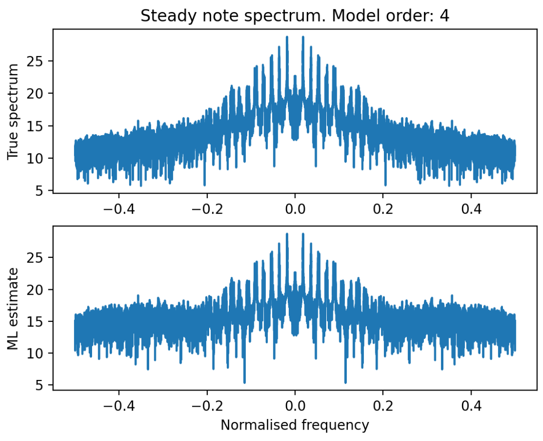


(a) True order = 30

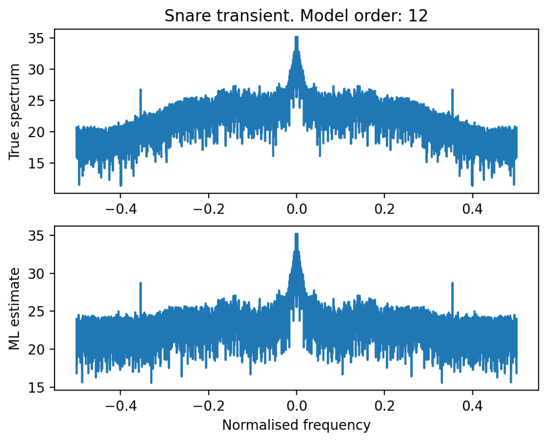


(b) True order = 150

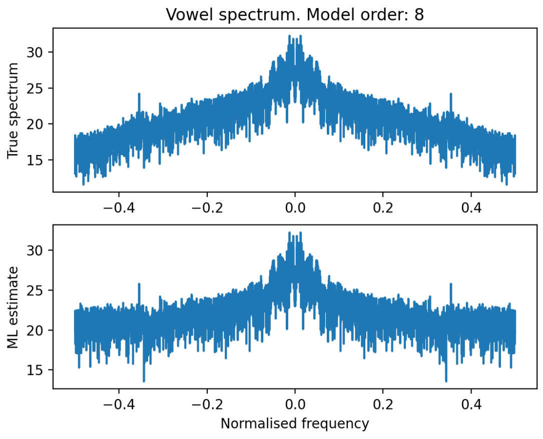
Figure 24: Model evidences for varying estimated model order



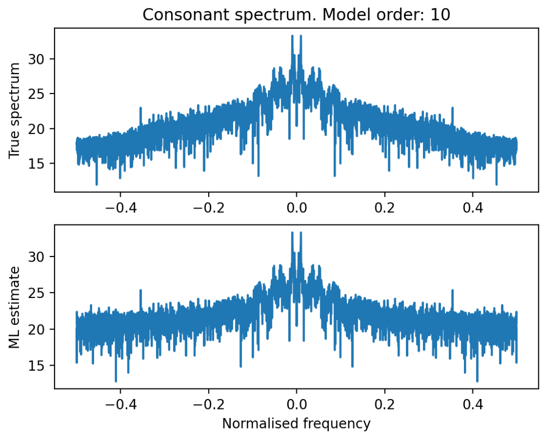
(a) Steady note



(b) Snare transient



(c) Vowel



(d) Consonant

Figure 25: Log spectra for different audio excerpts

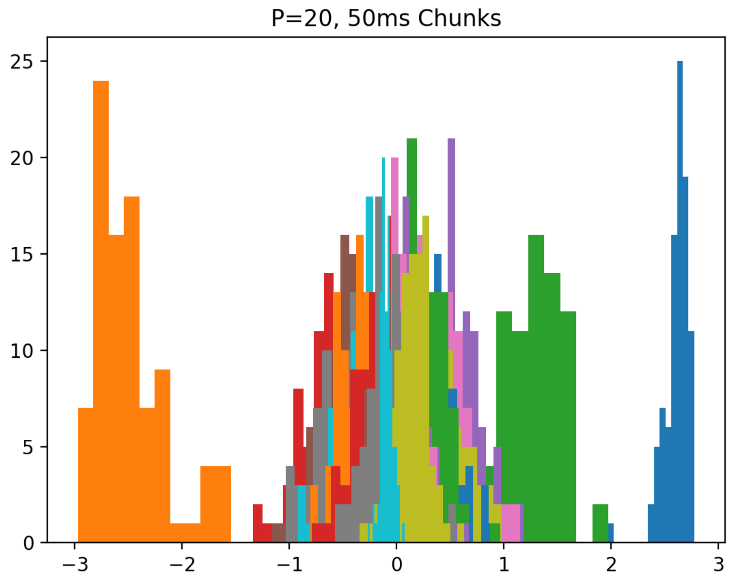


Figure 26: Histogram of AR(20) coefficients for 50ms chunks of `grosse_orig.wav`. The groupings are clear (especially away from zero), but note that colours have been reused.

```
def get_missing_indices_ranges(data):
    # two step process to retrieve all bursts of degradation longer than 2 samples
    # missing samples are zeroed out
    missing_i = np.array([i for i in range(data.shape[0]) if data[i] == 0])
    # to find ranges, we must find the first and last indices of each burst
    beginning_i = np.array([missing_i[i] for i in range(missing_i.shape[0]) if not
        missing_i[i-1] == missing_i[i]-1])
    ending_i = np.array([missing_i[i] for i in range(-1, missing_i.shape[0]-1) if not
        missing_i[i+1] == missing_i[i]+1])
    ending_i = np.roll(ending_i, -1)
    # zip to give ranges
    missing_r = np.array(list(zip(beginning_i, ending_i)))

    # have to repeat process to remove single points in the audio
    broken_i = np.empty(0)
    for item in missing_r:
        if (item[1]-item[0])<=1:
            broken_i = np.append(broken_i, item[0])
            broken_i = np.append(broken_i, item[1])

    # rerun the initial process with the singular points in the audio removed
    missing_i = np.array([item for item in missing_i if not (item in broken_i)])
    beginning_i = np.array([missing_i[i] for i in range(missing_i.shape[0]) if not
        missing_i[i-1] == missing_i[i]-1])
    ending_i = np.array([missing_i[i] for i in range(-1, missing_i.shape[0]-1) if not
        missing_i[i+1] == missing_i[i]+1])
    ending_i = np.roll(ending_i, -1)

    missing_r = np.array(list(zip(beginning_i, ending_i)))

    return missing_i, missing_r
```

Figure 27: Python function for locating missing indices

```
def ar_interpolation(data, alphas_f, alphas_b, missing_i, missing_r, noise_variance):
    prediction_f = np.copy(data) # arrays passed by reference
    # iterate over missing indices
    for index in missing_i:
        term = 0
        # find forward prediction of the process at missing index, checking
            # if the array bounds are violated
        for i in range(alphas_f.shape[0]): # equivalent to zero initial conditions
            if index-i-1 >=0:
                term += alphas_f[i] * prediction_f[index-i-1]
        # assign forward prediction term along with optional noise
        prediction_f[index] = term + np.sqrt(noise_variance)*np.random.normal(size=1)[0]

    # find reverse prediction by iterating over missing data backwards
    prediction_b = np.copy(data)
    for index in reversed(missing_i):
        term = 0
        for i in range(alphas_b.shape[0]):
            if index+i+1 < data.shape[0]:
                term += alphas_b[i] * prediction_b[index+i+1]
        prediction_b[index] = term + np.sqrt(noise_variance)*np.random.normal(size=1)[0]

    # locate each gap in the data and calculate the weighted sum at each point
    prediction_c = np.copy(data)
    for gap in missing_r:
        gap_length = gap[1]-gap[0]
        for i in range(gap_length+1):
            rho = i/(gap_length-1)
            prediction_c[gap[0]+i] = (1-rho)*prediction_f[gap[0]+i] + rho*prediction_b[gap
                [0]+i]

    # return combined prediction
    return prediction_c
```

Figure 28: Python function for interpolating the missing data

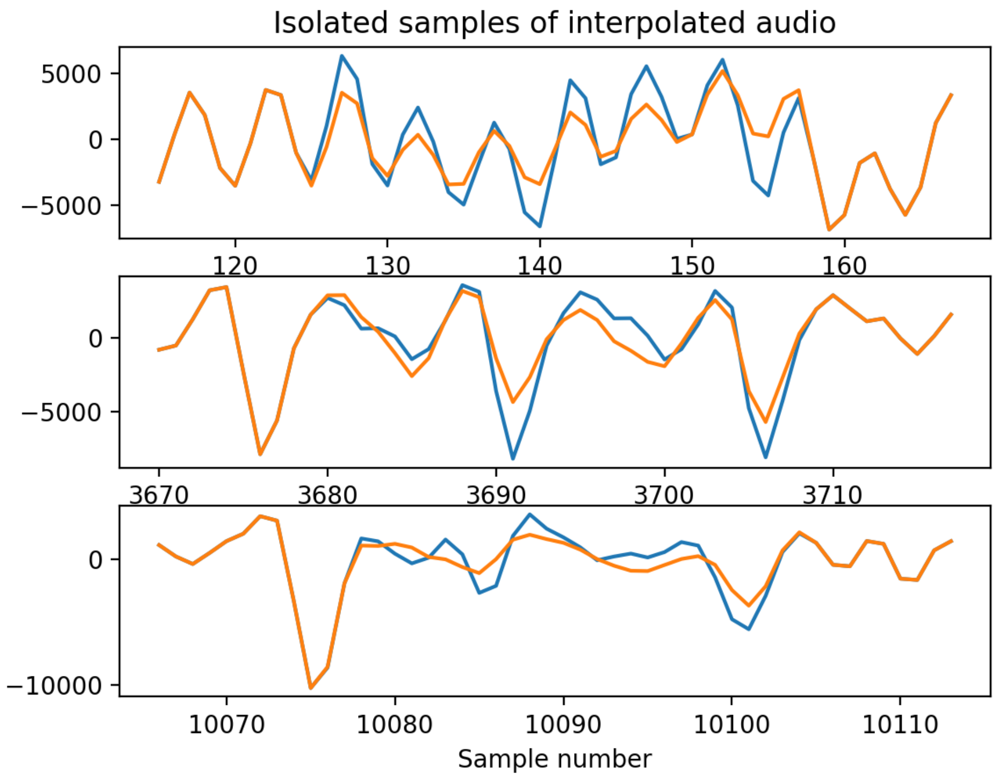
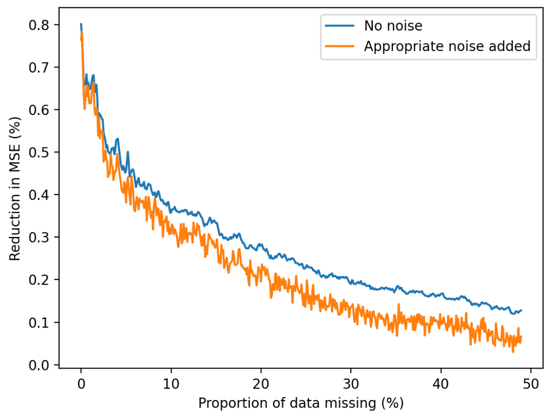


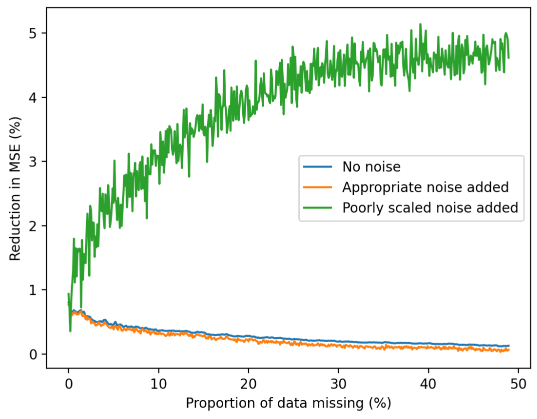
Figure 29: ML estimates of `armstrong_37_orig.wav` for three isolated intervals. Blue: True, Orange: Estimated.

	Mean squared error		
Sample	Before processing	After processing	Reduction (%)
Armstrong	2,893,496.42	1,163,987.53	59.8
Grosse	23,384.45	6,445.21	72.4

Figure 30: MSE values for provided test samples



(a) Without poorly scaled noise



(b) With poorly scaled noise

Figure 31: Noise reduction as a function of percentage of missing data

6 Appendix B: Task 4 - the matched filter

Another common method of signal detection is the matched filter. A matched filter is chosen such that a known signal being passed through the filter, will have maximum output energy at the beginning of the known signal. For constant noise, this is the same as maximising the SNR at time λ. As stated in task 4, the Bayesian detection problem is of the form:

$$\hat{\lambda} = \underset{\lambda}{\operatorname{argmax}}\, p(Y|\mathcal{M}_\lambda)$$

$$\hat{\lambda} = \underset{\lambda}{\operatorname{argmax}} \left\{ \mathcal{C}_\lambda + \frac{1}{2\sigma_e^2} \frac{(\sum_{n=\lambda}^{\lambda+14} s_{n-\lambda} y_n + m_\gamma \frac{\sigma_e^2}{\sigma_\gamma^2})^2}{\sum_{i=0}^{14} s_i^2 + \frac{\sigma_e^2}{\sigma_\gamma^2}} \right\}$$

where $\mathcal{C}_\lambda$ contains all terms in $p(Y|\mathcal{M}_\lambda)$ that do not depend on λ. For algebraic ease we will only consider the case where $m_\gamma = 0$ and $\sigma_\gamma^2 \to \infty$, but the analysis can be extended to include this prior information. Then:

$$\hat{\lambda} = \underset{\lambda}{\operatorname{argmax}} \left\{ \frac{1}{2\sigma_e^2} \frac{(\sum_{n=\lambda}^{\lambda+14} s_{n-\lambda} y_n)^2}{\sum_{i=0}^{14} s_i^2} \right\} = \underset{\lambda}{\operatorname{argmax}} \left\{ \Big(\sum_{n=\lambda}^{\lambda+14} s_{n-\lambda} y_n \Big)^2 \right\}$$

Given that $s_{n-\lambda}$ is only non-zero for $0 \le n - \lambda_{true} \le 14$, maximising this sum is clearly equivalent to having as many non-zero terms in the sum as possible - i.e setting $\lambda = \lambda_{true}$. We can also show that the estimator is maximising the output power. Using the fact that $y_n = \gamma s_{n-\lambda} + e_n$ for $\lambda_{true} \le n \le \lambda_{true} + 14$, and $y_n = e_n$ otherwise:

$$\hat{\lambda} = \underset{\lambda}{\operatorname{argmax}}\, J = \underset{\lambda}{\operatorname{argmax}} \left\{ \Big(\sum_{n=\lambda}^{\lambda+14} s_{n-\lambda}(\gamma s_{n-\lambda} + e_n) \Big)^2 \right\} = \underset{\lambda}{\operatorname{argmax}} \left\{ \Big(\gamma \sum_{n=\lambda}^{\lambda+14} s_{n-\lambda}^2 + \sum_{n=\lambda}^{\lambda+14} s_{n-\lambda} e_n \Big)^2 \right\}$$

However, as the squared terms are strictly non-negative, $\sum_{n=\lambda}^{\lambda+14} s_{n-\lambda}^2 \le S$ where S is the total hidden signal power, with equivalence when $\lambda = \lambda_{true}$. If we expand the sum to be maximised we get:

$$J = \gamma^2 \Big(\sum_{n=\lambda}^{\lambda+14} s_{n-\lambda}^2 \Big)^2 + 2\gamma \Big(\sum_{n=\lambda}^{\lambda+14} s_{n-\lambda}^2 \Big) \sum_{n=\lambda}^{\lambda+14} s_{n-\lambda} e_n + \Big(\sum_{n=\lambda}^{\lambda+14} s_{n-\lambda} e_n \Big)^2$$

Now, taking expected values:

$$\mathbb{E}\,J = \gamma^2\,\mathbb{E}\big(\sum_{n=\lambda}^{\lambda+14} s_{n-\lambda}^2\big)^2 + 2\gamma\big(\sum_{n=\lambda}^{\lambda+14}\mathbb{E}\,s_{n-\lambda}^2\big)\sum_{n=\lambda}^{\lambda+14}\mathbb{E}\,s_{n-\lambda}\,\mathbb{E}\,e_n + \mathbb{E}\big(\sum_{n=\lambda}^{\lambda+14} s_{n-\lambda}e_n\big)^2$$

We assume that the noise is iid, zero mean and uncorrelated with the data such that:

$$\begin{aligned}\mathbb{E}\,J &= \gamma^2\,\mathbb{E}\big(\sum_{n=\lambda}^{\lambda+14} s_{n-\lambda}^2\big)^2 + \sum_{n=\lambda}^{\lambda+14}\mathbb{E}\,s_{n-\lambda}^2\,\mathbb{E}\,e_n^2 \\ &= \gamma^2\,\mathbb{E}\big(\sum_{n=\lambda}^{\lambda+14} s_{n-\lambda}^2\big)^2 + \sigma_e^2\sum_{n=\lambda}^{\lambda+14}\mathbb{E}\,s_{n-\lambda}^2\end{aligned}$$

All terms are positive so:

$$\mathbb{E}\,J \leq \gamma^2 S^2 + \sigma_e^2 S = \gamma S(\gamma S + \frac{\sigma_e^2}{\gamma}) \quad \text{with equivalence at} \quad \lambda = \lambda_{true}$$

i.e $\mathbb{E}\,J$ is bounded above by the output signal power of the detection filter. Therefore, our decision rule is choosing $\hat{\lambda}$ such that the output power is maximised, which occurs on average when $\lambda = \lambda_{true}$.

7 Appendix C: Task 5 - least squares interpolation

With the time I had left I decided to explore the least squares interpolator for packet recovery. I used the AR model trained in previous sections to build a coefficient matrix A, then performed appropriate partitioning of A and the degraded signal x. I created a function that takes in x and an AR model, and returns the fully interpolated signal. I tested this function on the armstrong sample, as well as my own test sample in which I removed a block of 600 samples from `armstrong.wav`. The results are included in my submission, and the interpolation function plus a graph of the estimated signal are shown in figures (32) and (33).

```
def least_squares_interpolator(data, alphas):
    missing_i, _ = get_missing_indices_ranges(data)
    # initialise matrices
    A_f = np.zeros((data.shape[0]-alphas.shape[0], data.shape[0]))
    Ai_f = np.zeros((data.shape[0]-alphas.shape[0], missing_i.shape[0]))
    Ami_f = np.zeros((data.shape[0]-alphas.shape[0], data.shape[0] - missing_i.shape[0]))
    x_mi_f = np.zeros(data.shape[0] - missing_i.shape[0])

    # coefficients vector
    a_vec_f = np.pad(-alphas, (0, 1), mode='constant', constant_values=1)
    # build A
    for i in range(A.shape[0]):
        A_f[i,:] = np.pad(a_vec_f, (i, A_f.shape[1]-i-a_vec_f.shape[0]))
    # build Ai
    for i in range(missing_i.shape[0]):
        Ai_f[:, i] = A_f[:, missing_i[i]]
    # build Ami
    c = 0
    for i in range(data.shape[0]):
        if i not in missing_i:
            Ami_f[:, c] = A_f[:, i]
            x_mi_f[c] = data[i]
            c += 1
    # preliminary calculation
    mat = np.matmul(np.linalg.inv(np.matmul(Ai_f.T, Ai_f)), Ai_f.T)
    ls = -np.matmul(mat, np.matmul(Ami_f, x_mi_f))
    # apply least squares solution in the correct places
    c2 = 0
    ls_data = np.copy(data)
    for i in range(data.shape[0]):
        if i in missing_i:
            ls_data[i] = ls[c2]
            c2+=1
    return ls_data
```

Figure 32: Python function for least squares interpolation

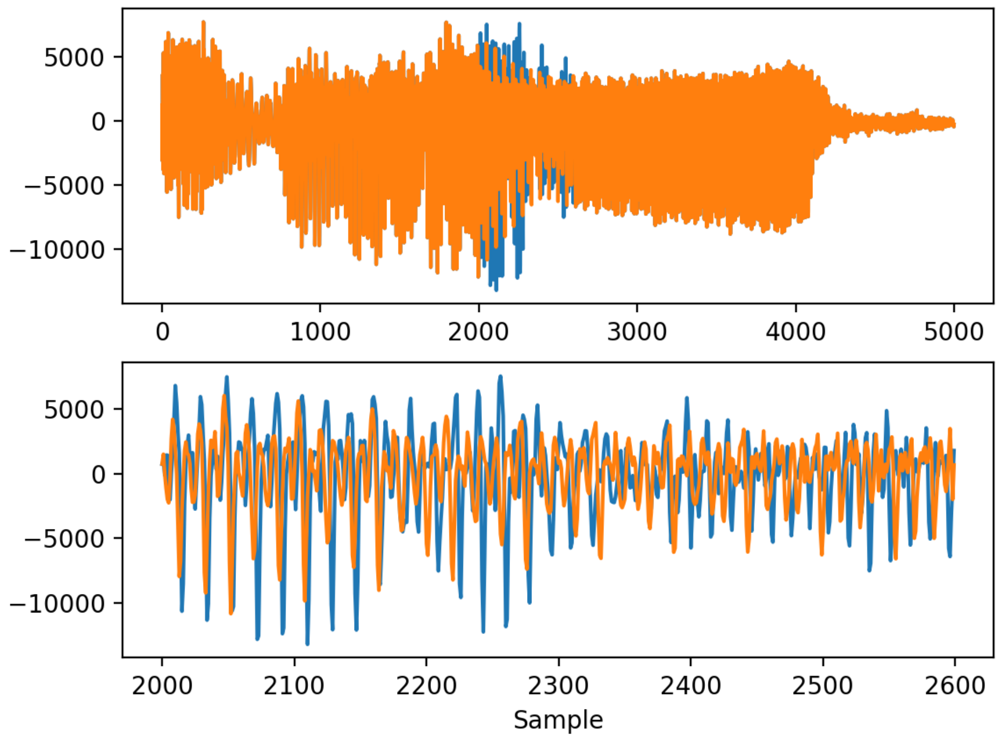


Figure 33: Least squares interpolation of missing block of 600 samples

8 Appendix D: Task 6 - J sinusoids model

In this section we consider the J sinusoids model described in the project handout. This model allows for a ML and Bayesian interpretation of the DFT. The terms used here are the same as in the handout. For $J = 1$, G is of the form:

$$G = \begin{bmatrix} \mathbf{c} & \mathbf{s} \end{bmatrix}$$

where $\mathbf{c}$ and $\mathbf{s}$ are column vectors containing sine and cosine waves evaluated at integer multiples of some frequency ω. If me evaluate the log likelihood evaluated at its maximum, given by the parameters:

$$\theta^{ML} = (G^T G)^{-1} G^T X$$

then, we see that (dropping any terms that do not depend on ω):

$$-\log p(X|\theta) = \theta^T G^T X + X^T G\theta - \theta^T G^T G\theta$$

$$= X^T G(G^T G)^{-1} G^T X \quad \text{at this ML solution}$$

We also have:

$$G^T X = \begin{bmatrix} \underline{\mathbf{c}}^T \underline{\mathbf{x}} \\ \underline{\mathbf{s}}^T \underline{\mathbf{x}} \end{bmatrix}$$

and

$$G^T G = \begin{bmatrix} \underline{\mathbf{c}}^T \underline{\mathbf{c}} & \underline{\mathbf{c}}^T \underline{\mathbf{s}} \\ \underline{\mathbf{c}}^T \underline{\mathbf{s}} & \underline{\mathbf{s}}^T \underline{\mathbf{s}} \end{bmatrix}$$

then

$$(G^TG)^{-1} = \Delta \begin{bmatrix} \underline{\mathbf{s}}^T\underline{\mathbf{s}} & -\underline{\mathbf{c}}^T\underline{\mathbf{s}} \\ -\underline{\mathbf{c}}^T\underline{\mathbf{s}} & \underline{\mathbf{c}}^T\underline{\mathbf{c}} \end{bmatrix} \quad \text{where} \quad \Delta = \det(G^TG)$$

Due to the fact that our sine and cosine waves form an orthonormal basis at a DFT bin frequency[2], $\frac{2\pi p}{N}$:

$$\underline{\mathbf{s}}^T\underline{\mathbf{s}} = \underline{\mathbf{c}}^T\underline{\mathbf{c}} = \frac{N}{2}$$

and

$$\underline{\mathbf{c}}^T\underline{\mathbf{s}} = 0$$

Therefore our log likelihood is of the form:

$$-\log p(X|\theta) = \frac{N\Delta}{2} \begin{bmatrix} \underline{\mathbf{c}}^T\underline{\mathbf{x}} & \underline{\mathbf{s}}^T\underline{\mathbf{x}} \end{bmatrix} I \begin{bmatrix} \underline{\mathbf{c}}^T\underline{\mathbf{x}} \\ \underline{\mathbf{s}}^T\underline{\mathbf{x}} \end{bmatrix}$$

Therefore:

$$-\log p(X|\theta) \propto (\underline{\mathbf{c}}^T\underline{\mathbf{x}})^2 + (\underline{\mathbf{s}}^T\underline{\mathbf{x}})^2$$

Similarly, if we consider the p^{th} DFT component:

$$X_p = \sum_{n=0}^{N-1} x_n e^{-jn\frac{2\pi p}{N}} = \underline{\mathbf{c}}^T\underline{\mathbf{x}} - j\underline{\mathbf{s}}^T\underline{\mathbf{x}}$$

Then:

$$|X_p|^2 = (\underline{\mathbf{c}}^T\underline{\mathbf{x}})^2 + (\underline{\mathbf{s}}^T\underline{\mathbf{x}})^2$$

Clearly then:

$$\log p(X|\theta^{ML}) \propto |X_p|^2$$

The code required to implement the J-sinusoids GLM is shown in figure(34). The functions for finding the posterior distribution and the model evidence are the same as in other sections, so are not included here. Figure (35) shows the estimated spectrum from the GLM for normalised frequencies ranging from 0.0001 to 0.5. There are many clear frequency spikes, which I attempted to extract by eye. I then reconstructed these frequencies in a sine wave generator and recorded the results - the reconstruction is included in the sample audio folder of my submission (note the recording is very loud so playback should be at low volumes). The quality of the reconstruction is reasonable as it seems to capture enough frequencies from `organ.wav` to be recognisable, however there are definitely still some elements of the original sound missing. With more time I would've liked to explore the possibility of incorporating priors into the GLM to improve my estimation.

[2]https://stats.stackexchange.com/questions/249198/from-a-statistical-perspective-fourier-transform-vs-regression-with-fourier-bas

```
def build_sinusoid_G(omegas, n):
    # number of frequencies
    J = omegas.shape[0]
    # 2 components for each freq, 1 row for each data point
    mat = np.zeros((n, 2*J))
    counter=0
    # build matrix of sinusiods at the normalised frequencies
    for i in range(J):
        freq = omegas[i]
        cosines = np.cos(np.arange(1, n+1)*freq)
        sines = np.sin(np.arange(1, n+1)*freq)
        mat[:, counter] = cosines
        counter += 1
        mat[:, counter] = sines
        counter += 1
    return mat

def get_theta_ML(x, Gmat):
    # standard least squares expression
    return np.matmul(np.linalg.inv(np.matmul(Gmat.T, Gmat)), np.matmul(Gmat.T, x))

def get_theta_amp(thetas, omegas):
    thetas_amp = np.zeros(omegas.shape[0])
    # extract amplitude at each frequency as the root squared sum of a and b
    for i in range(omegas.shape[0]):
        thetas_amp[i] = np.square(thetas[2*i]) + np.square(thetas[2*i+1])
    return thetas_amp
```

Figure 34: Python code for J-sinusoids

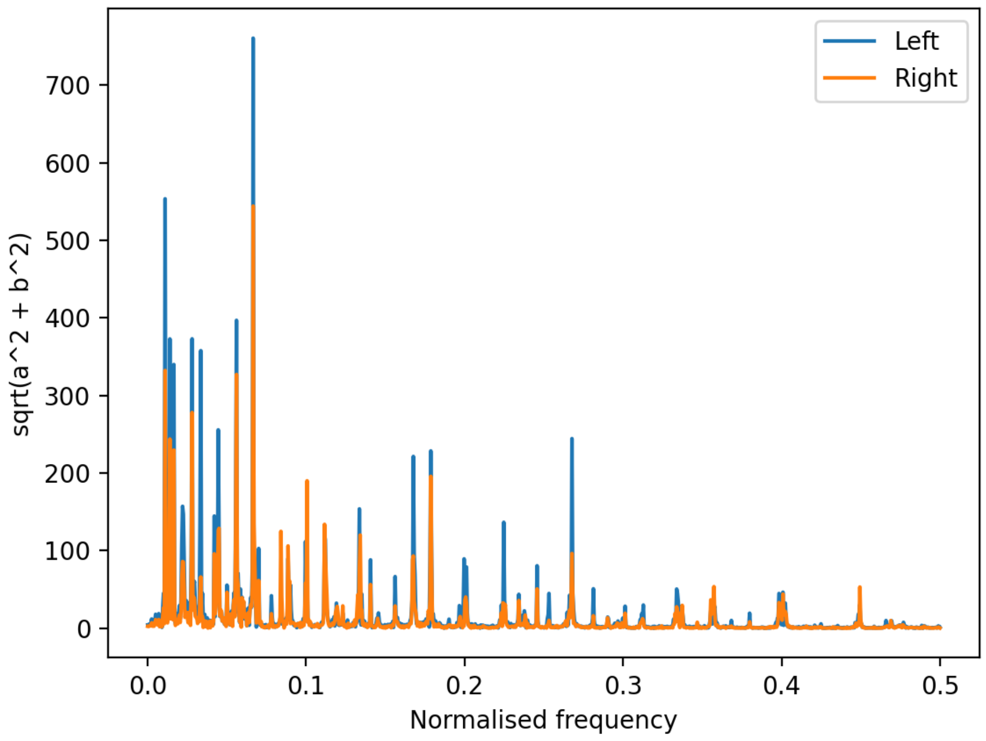


Figure 35: Maximum likelihood estimate of frequency spectrum for `organ.wav`