Bayesian estimation of information content in models with different data sizes

Satnam Singh

Master of Science
Department of Statistics
The University of Auckland
New Zealand

A dissertation submitted in partial fulfilment of the requirements for the degree of Master of Science in Statistics

Supervised by Dr. Brendon Brewer

"The biggest risk is not taking any risk. In a world that's changing really quickly, the only strategy that is guaranteed to fail is not taking risks."

— Mark Zuckerberg

- Fortune favours the brave -

Abstract

There are several ways of measuring the information that is provided in a dataset. One of the ways we can measure this information is by calculating the entropy. In this dissertation, we look at how the information content varies as the size of the data changes. We also look at the posterior distribution of a set of hyperparameters in a time-series model, and how its' posterior variance change as the size of the dataset changes.

This project also looks at testing a hypothesis for a t-test model in a Bayesian setting, to explore the relationship between the entropy, posterior probability and the p-value. And finally, we have also explored some common covariance functions to generate Gaussians, which can be used to compute the hpyerparameters of an Autoregressive time-series model.

Acknowledgements

Firstly, I want to say thanks to my supervisor, Dr. Brendon Brewer for this project. It was because of Brendon's exceptional teaching skills that I was able to easily grasp a lot of the complicated theory, the code and the examples involved in this project. It helped me understand the topic a lot more which made it clear what the project was trying to achieve. Finally, thank you for fixing up my code, my formulas & my writing and thank you for looking after me.

Thank you to my mum for ... *just being mum.* And for the encouragements, the great food and for the mum support.

Staff at the 3/6 RNZIR (NZDF), especially Fereti and Sgt. Mills *(Millsie)*, who allowed for tremendous flexibility for my studies and for this project during the year *(and previous years)*, in making sure that trainings, trips and exercises are not interfering with my studies.

Thank you to the Statistics Department, especially friends from graduate room **303.299**:

- Shaun - *for providing lots of valuable advice, sharing his project knowledge and experiences. Plus helping out with code, having good banter and doing ocassional pushups in the lab*
- Sah & Lucas - *for the ever so delicious cakes, cookies, and the Lindt chocolates (aka morale boosters) throughout the semester, that kept me going*
- Theodora - *for helping me run my code on University's NeSI cluster*
- Chrystal - *for offering advice, food and for saying motivational quotes such as "you can do it" like a million times*
- Sara - *for proofreading and for providing an example .RNW file template on which this dissertation is written*
- Suba and Mugdha - *for giving invaluable advice on how to manage project, work & papers. And for keeping me sane in the time of great stress*
- Ben - *for listenning about my project, asking question, and providing feedback*
- David - *for helping with a likelihood function this one time*

Thank you all and thank you God.

Table of Contents

Contents

1 Metropolis-Hastings (MH) Algorithm **5**

1.1 Posterior Distribution 5

1.2 Proposal Distribution 5

1.3 Acceptance Probability 6

2 Autoregressive (AR) Models **7**

2.1 Estimating AR(1) model parameters using MH Algorithm 7

2.2 Simulated data for AR(1) likelihood function 8

2.3 MCMC estimated parameters 9

3 Posterior Expected Loss **10**

3.1 Posterior Variance 10

3.2 Expected Loss 10

3.3 Posterior Variances in an AR(1) model 11

3.4 Average Posterior Variances in an AR(1) model 11

4 Entropy **13**

4.1 Measuring Entropy in a t-test model 13

4.2 Shannon's Mutual Information 14

4.3 Generating datasets to calculate Posterior Probabilities and Entropies 14

4.4 But why stop at just one dataset? 14

4.5 A Simple Example 17

5 Posterior Entropy vs. Sample Size **21**

5.1 Testing prior t-test model against big datasets 21

5.2 Entropy vs Sample Size 22

5.3 Information content in each data group 24

6 AR(1) time series distribution observed with & without noise **26**

7 Covariance Function **28**

8 Known/Pre-defined Hyperparameters **29**

8.1 Drawing 5 Multivariate Normal Samples from a Gaussian, given a mean vector, μ and a Covariance Matrix, Σ 29

8.2 Larger set of observations (N = 500) 31

9 Absolute Covariance Function **34**

9.1 Posterior Distribution of θ 34

9.2 Determining Hyperparameters 34

9.3 Using Metropolis-Hastings algorithm to estimate the hyperparameters 35

9.4 Check Posterior for Convergence 36

10 Using Posterior estimates from MCMC to compute Gaussians **37**

10.1 Which posterior estimate to use? 37

11 Conclusion **39**

11.1 Discussion 39

11.2 Limitations 39

12 Bibliography **40**

13 Appendix **41**

1 Metropolis-Hastings (MH) Algorithm

The Metropolis-Hastings algorithm is a Markov Chain Monte Carlo (MCMC) algorithm, which is designed to sample any distribution that may be of interest. In this dissertation, we have used the MH algorithm to sample the **posterior distribution** of different models, so we can estimate its' parameters and therefore work out the entropy, posterior variance and the information content.

1.1 Posterior Distribution

If we have a current position in a parameter space *(or a current set of estimates)*, we can use the MH algorithm to generate a sequence of values within this parameter space to sample a **posterior distribution**. This is sometimes refered to the **target distribution** which we are trying to sample.

1.2 Proposal Distribution

In order to generate samples from this posterior distribution, the parameter space needs to be explored. This is done by generting a **proposal** or a **candidate point** for where the next position *might* be, depending on the density at that next proposal *(using some likelihood function)* and/or by some probability. The probability distribution for the next point where the random walk *might* move to is known as the **proposal distribution**.

For example, given that you are at a position *(irrespective to how you got there)*, you might want to propose a new move to a nearby parameter space from your proposal distribution. And depending on how near or far this proposal is, it always has some density associated with it. If the density at the proposed state $p(x)$ is low, then it is *likely* that it will be rejected. In which case the MH algorithm rejects the proposal and chooses another proposal that may be closer *(i.e. have a higher density)*, and hence be more acceptable to go to.

1.3 Acceptance Probability

If $p(x) \geq p(x^{t-1})$, then the proposal is accepted, and this becomes the new current state as $p(x)$ has greater or equal density.

If $p(x) < p(x^{t-1})$, then the proposal may *still* be accepted, but with an **acceptance probability**:

$$\frac{p(x)}{p(x^{t-1})}$$

where the final acceptance probability takes the value one, if the ratio of the above two densities exceeds one. So, the acceptance probability (α) is given by:

$$\alpha = \min\left(1, \frac{p(x)}{p(x^{t-1})}\right)$$

The next section looks at how the MH algorithm is used to explore the posterior distribution of the parameters of a time-series Autoregressive model.

2 Autoregressive (AR) Models

In time-series, an Autoregressive (AR) model is when a value from a time series is regressed on the previous value from the same time series. The term *auto*regression indicates that it is a regression for a variable, against the *same* variable in the past. For example, y_t regressed on y_{t-1}.

The notation AR(p) indicates an autoregressive model of order p. This AR(p) model is defined as:

$$y_t = \mu + \sum_{i=1}^{p} \phi_i (y_{t-1} - \mu) + \epsilon_t$$

In this dissertation, an autoregressive order one process AR(1) has been used.

$$y_t = \mu + \phi (y_{t-1} - \mu) + \epsilon_t$$

where,

μ = mean where that time series is centered around
ϕ_i = some constant *(proportion of the previous value remembered)*
y_{t-1} = previous value
$\epsilon_t \sim iid(0,\, \beta^2)$

2.1 Estimating AR(1) model parameters using MH Algorithm

In an AR(1) model, there are three unknown parameters which we can estimate using MCMC simulation in the MH code. That is, some constant mean (μ), proportion of the previous value remembered (ϕ), and the standard deviation of the innovation/jump (β).

The **Likelihood Function** used in the MCMC simulation is the product of the **first** values *(given simply by the Normal density)*:

$$f(y) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left[\frac{y_i - \mu}{\sigma}\right]^2}$$

and the remaining values by:

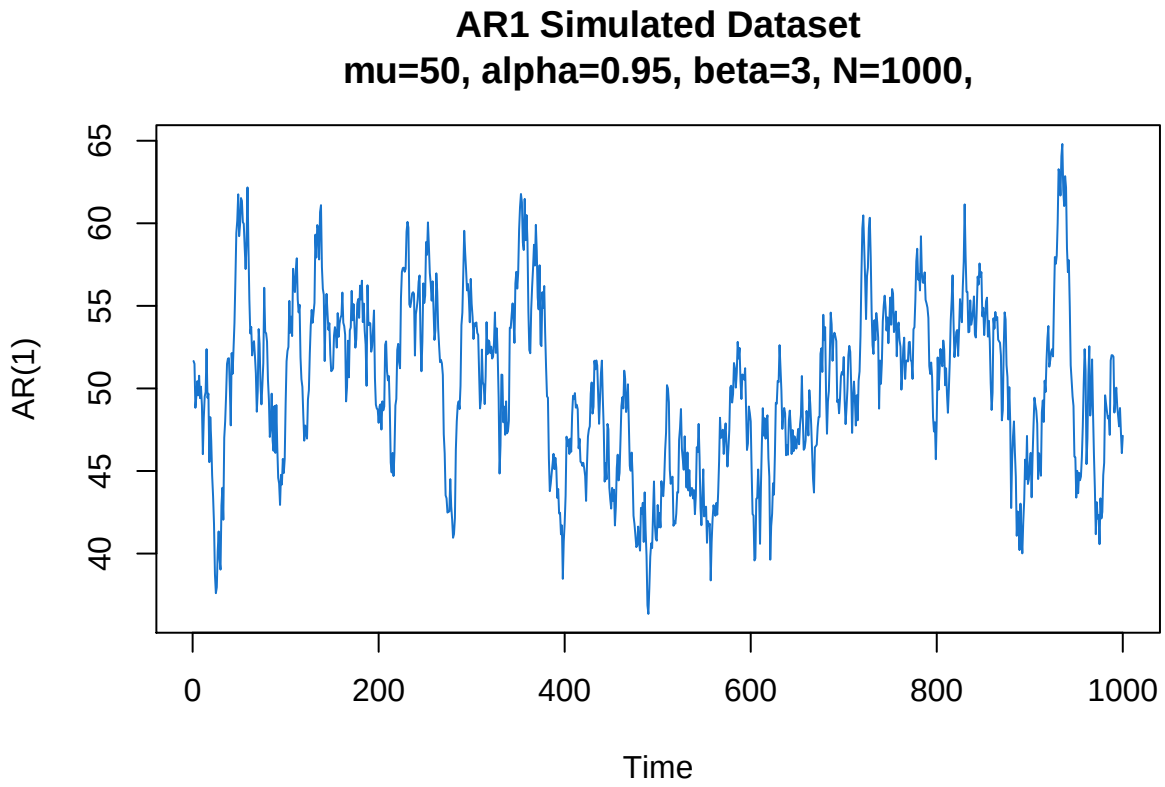
$$\prod_{i=2}^{N} \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left[\frac{y_i - [\mu + \phi(y_{i-1} - \mu)]}{\sigma}\right]^2}$$

where, $\sigma = \frac{\beta}{\sqrt{1-\alpha^2}}$

2.2 Simulated data for AR(1) likelihood function

The likelihood is a function of the parameters $\{\mu, \phi, \beta\}$, given some data. In this AR(1) model, the data is generated using the `arima.sim` R function in the Stats package. The parameters for this data will be predefined, so we can see the performance of the MCMC simulation in the Metropolis-Hastings code.

```
# GENERATING DATA FOR THE LIKELIHOOD
N = 1000 #Sample, data points
alpha = 0.95 #Proportion of the previous value remembered
B = 3 #The Jump, variance
mu = 50 #Mean

set.seed(3); ar1 = mu + arima.sim(model = list(ar=alpha), n = N, sd = sqrt(B))
ar1_data = as.numeric(ar1)

main = paste("AR1 Simulated Dataset\nmu=", mu, ", alpha=",
             alpha, ", beta=", B, ", N=", N, ",", sep = "")

plot(ar1_data, main = main, ylab = "AR(1)", xlab = "Time",
     type = "l", pch = 16, col = "dodgerblue3")
```



2.3 MCMC estimated parameters

The MCMC run has given a good estimate of the three *unknown* parameters, by exploring the full parameter space over a large number of iterations. This AR(1) model will be used to to calculate the **Expected Posterior Loss** or the **Average Posterior Variance** the in the next chapter.

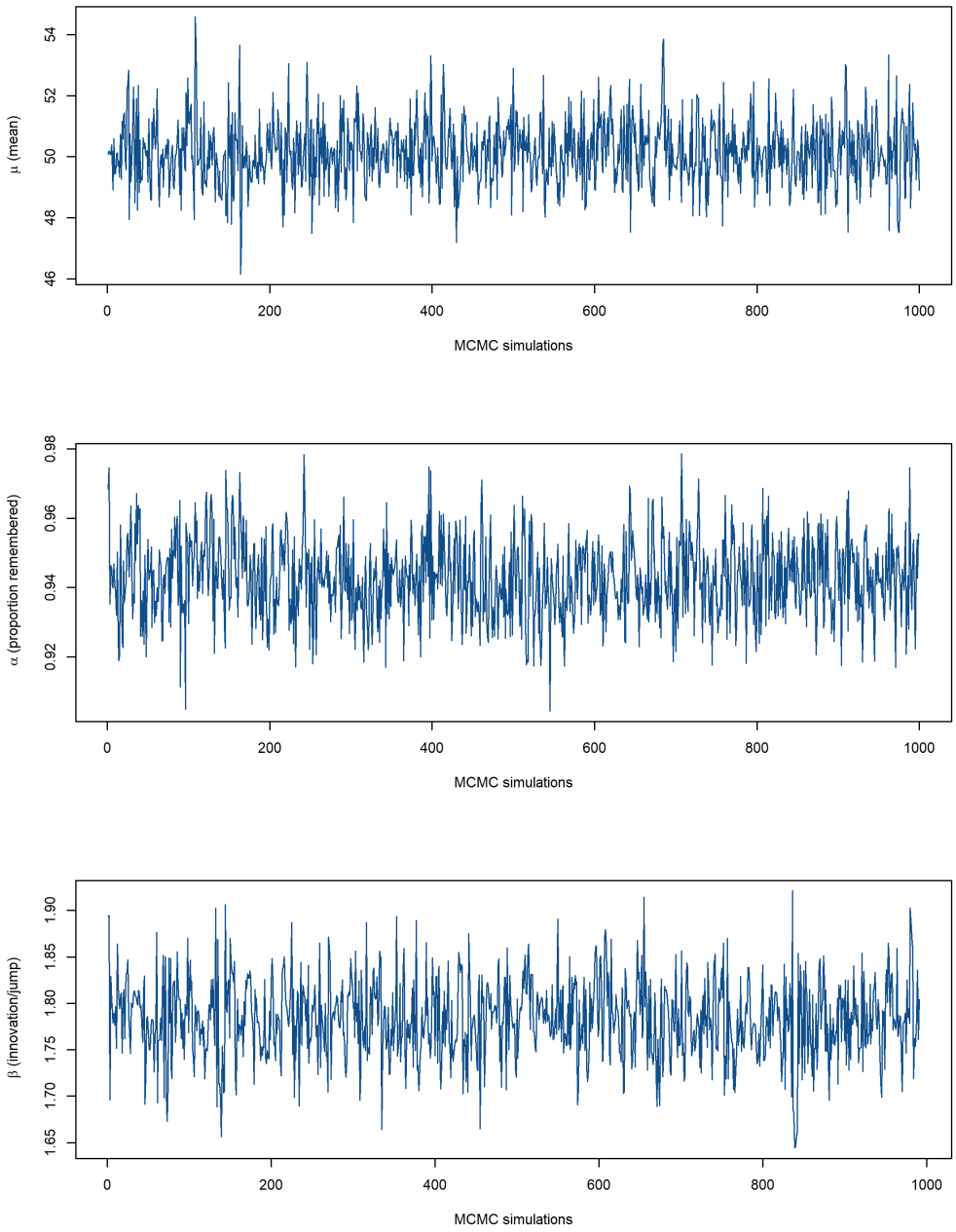


Figure 1: Posterior distribution of the hyperparameters of a simulated AR(1) model

3 Posterior Expected Loss

3.1 Posterior Variance

The Expected Loss *(before you get the data)*, is the average of the **Posterior Variance**. This means that we can calculate the expected loss by calculating the variance of the posterior distribution, averaged over all the possible datasets that we might observe. In another words, the posterior variance is a function of the dataset and we can calculate it by taking the expected value.

$$E\Big[L(\hat{\theta},\theta)\Big] = \int \Big(\theta - \hat{\theta}\Big)^2 p(\theta|x)d\theta$$

3.2 Expected Loss

Calculating the posterior variance is fairly straightforward to do as we can simply simulate a dataset given some prior and run MCMC to get our posterior. We can therefore do this over a period of time and with different sized data sets, and finally, take the average to get the **Expected Loss**.

In the end, we want to see if the average posterior variance gets smaller as the dataset gets bigger, allowing us to work out the information content of the data. In this dissertation, we have done this by measuring the expected loss.

What we expect:

With a bigger dataset, the posterior expected loss should go down with the size of the dataset. To be able to do this, a quadratic loss function is required. The posterior variance is a function of the dataset and it is given below:

3.3 Posterior Variances in an AR(1) model

	mu	alpha	beta	DataSize
1	5339.7986	4.7036	0.0051	100
2	1012.133	2.1856	0.0089	100
3	6526.5847	4.2246	0.0051	100
4	3665.9557	4.3212	0.005	100
5	3698.4434	5.1386	0.005	100
.	.	.	.	.
201	2750.9703	5.0401	0.0025	200
202	2057.0202	5.2738	0.0027	200
203	1533.3836	4.2238	0.0029	200
204	2243.7897	4.6185	0.0029	200
205	5103.9765	4.5777	0.0027	200
.	.	.	.	.
.	.	.	.	.
.	.	.	.	.
1996	1.0012	0.0499	9e-04	1000
1997	1.792	0.0606	5e-04	1000
1998	1.5166	0.0671	5e-04	1000
1999	4.0811	0.0919	5e-04	1000
2000	2.3209	0.071	0.0013	1000

Table 1: Set of 200 Posterior Variances for AR(1) models, per data size

3.4 Average Posterior Variances in an AR(1) model

```
avg_post_mean = aggregate(post_vars[, 1:3], list(post_vars$AR1_Data_size), mean)
colnames(avg_post_mean) = c("AR1_Data_size", colnames(avg_post_mean)[2:4])
avg_post_mean$AR1_Data_size = as.numeric(as.character(avg_post_mean[, 1]))

par(mfrow = c(3,1))
plot(mu ~ AR1_Data_size, data =  avg_post_mean, pch = 19, cex = 2.2,
     main = "Average Posterior Variance vs. data size,\nfor each Hyperparameter",
     ylab = expression(mu), xlab = "", col = "dodgerblue3")

plot(alpha ~ AR1_Data_size, data =  avg_post_mean, pch = 19, cex = 2.2,
     ylab = expression(phi), xlab = "", col = "deeppink2")

plot(beta ~ AR1_Data_size, data =  avg_post_mean, pch = 19, cex = 2.2,
     xlab = "Size of the AR(1) data",
     ylab = expression(beta), col = "springgreen3")
```

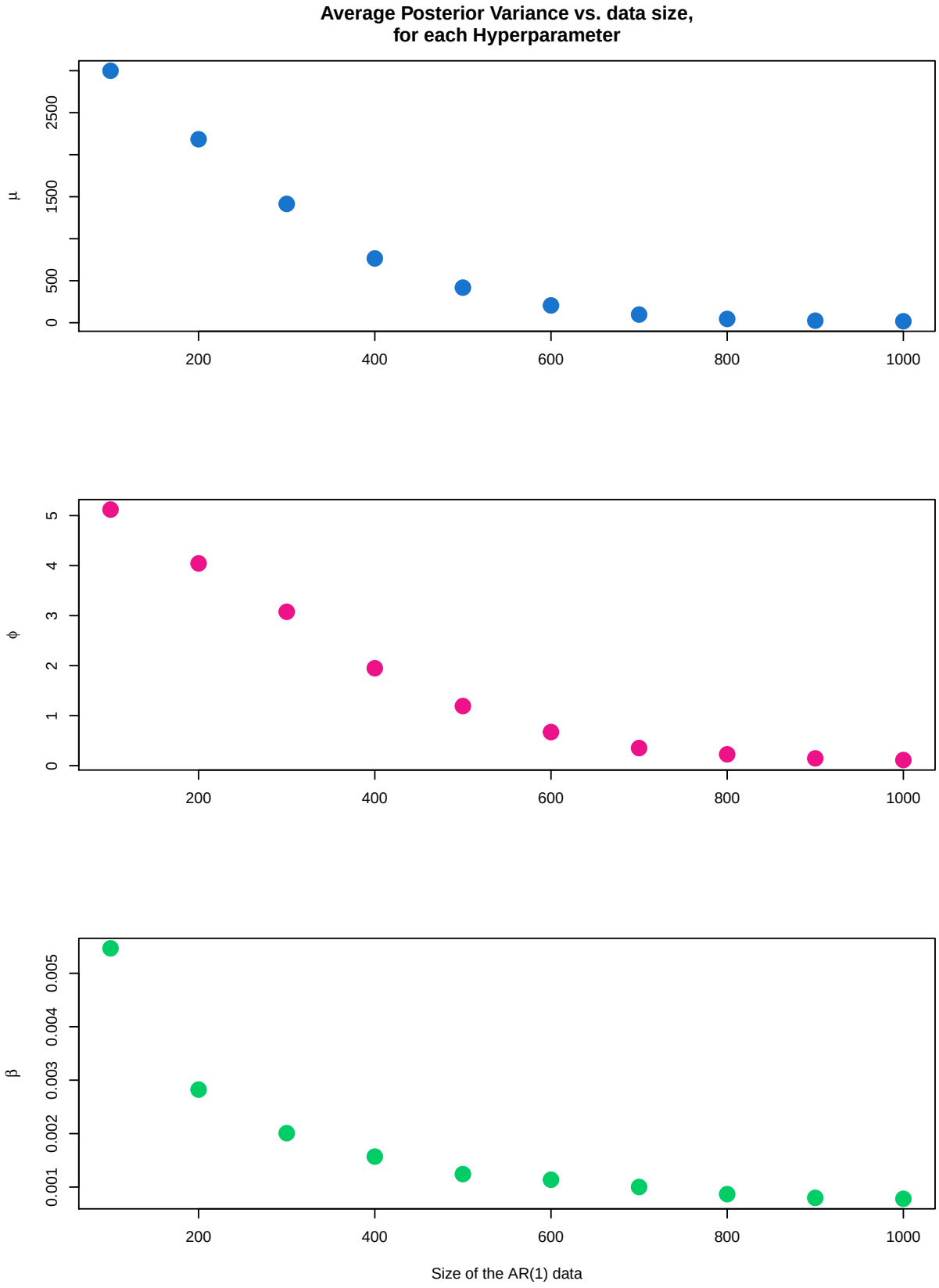


Figure 2: Change in posterior variances for each hyperparameter in a simulated AR(1) model

4 Entropy

There are different ways of measuring the amount of information that a data set will provide. One of the ways we can measure this information is by Entropy. A common definition of Entropy is that it is a measure of disorder in a system, and that it is always increasing. In Statistics, the Shannon Entropy is a measure of disorder or uncertainty in a probability distribution, given by:

$$H = -p_0 \log_2(p_0) - p_1 \log_2(p_1)$$

for N = 2 possible states, H_0 and H_1

4.1 Measuring Entropy in a t-test model

In Classical Statistics, a t-test allows us to test the hypothesis to see whether there is a significant difference between the means of two groups. The evidence against the null hypothesis is given by the *p-value*, under the assumption that the null is infact true.

$$H_0\text{: } \mu_1 = \mu_2$$
$$H_1\text{: } \mu_1 \neq \mu_2$$

However, in Bayesian Statistics, we have a **Joint Posterior Distribution** for μ_1 and μ_2 for this same hypothesis. The equivalent for this t-test model in Bayesian Statistics also allows for the two means to be exactly equal, however, rather than a p-value we get the **Posterior Probability** that $\mu_1 = \mu_2$, given the data. So, we get a posterior probability of H_0 and H_1.

For example, if the means for the two datasets looks very different, then the posterior probability for H_0 will be low and hence, the posterior probability for H_1 will be 1-Posterior probability of H_0. Which means we end up getting both of them.

$$p(H_0|x) + p(H_1|x) = 1$$

4.2 Shannon's Mutual Information

Mutual information measures the relationship between two random variables, which are sampled together. It measures, on average, how much information one random variable provide for another.

In our testing prior t-test model, our two random variables are the prior entropy and the average posterior entropy. Therefore, Shannon's Mutual information, in terms of the transfer of information content is the difference between prior entropy and the average posterior entropy.

$$I(\Theta, X) = H(\Theta) - H(\Theta|X)$$

4.3 Generating datasets to calculate Posterior Probabilities and Entropies

The posterior probability for H_0 and H_1 is correlated with the **Posterior Entropy**. If we generate a dataset to use in a t-test model which strongly supports the null *(i.e. has no evidence against the H_0 in classical terms)*, we can safely say that there is no significant difference against the two group means being the same.

What this also translates to [in Bayesian terms] is, that there is no remaining *uncertainty*. Hence, the posterior entropy is zero *(or very close to zero)*, as we have pretty much answered the question that the hypotheses posed for this one dataset.

4.4 But why stop at just one dataset?

To see how the overall relationship looks like between the posterior probability and the posterior entropy, we can generate a large number of datasets with varying sample sizes. Some of which supports the supports H_0 *(roughly 50% of the time)*, and the remaining that supports H_1. This will be known as our **Testing Prior**.

So basically, the reason we have done this is because of our prior belief that the null hypothesis is actually possible. I.e. Given, we believe that there is a very good chance that the hypothesis $\mu_1 = \mu_2$ could infact be true, we can implement this so that our prior datasets reflect that belief - and that's what we have done in the testing prior t-test model.

The testing prior t-test model specifications:

The datasets for the t-test model are generated with the specifications such that it assigns:

- 50% probability to the prior hypothesis that $\mu_1 = \mu_2$ and,
- 50% prior probability that $\mu_2 = \mu_1$+**difference**

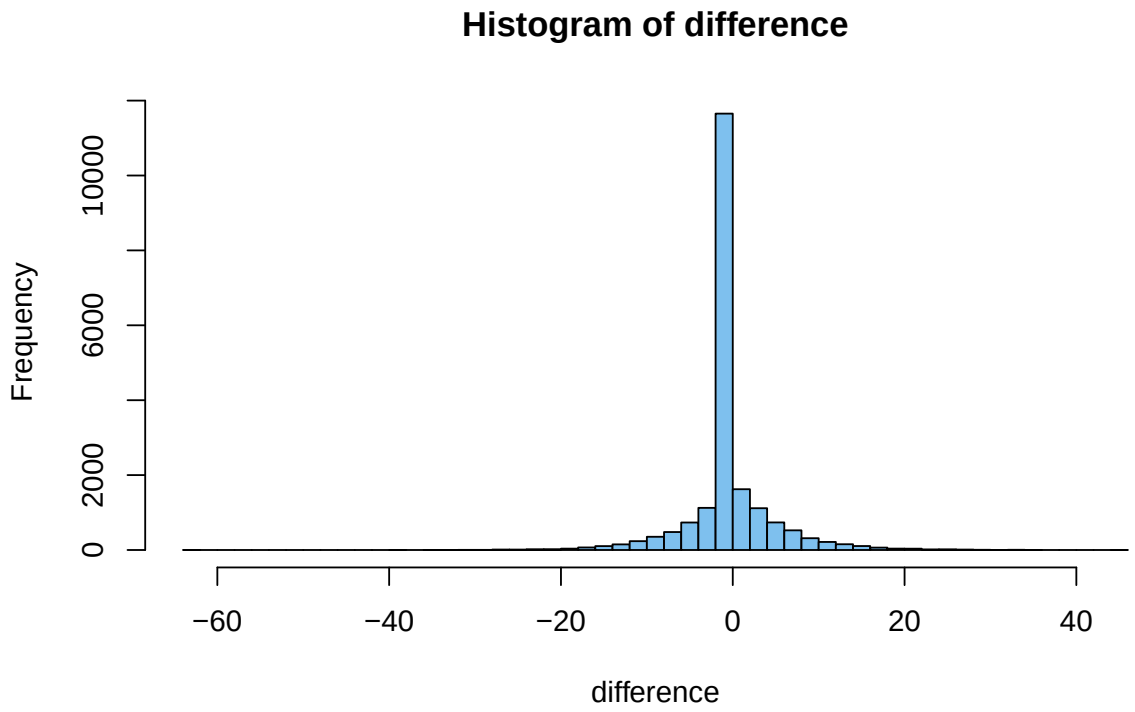
and, the size of this difference $= -L \times log(1 - u)$

where,
$L = 5$ *(length of the exponential prior)*
$u \sim u[-1, 1]$

and finally, this difference is turned into either being positive or negative with equal probability.

The priors for the t-test model is implemented in JAGS, which is a program used for analysing Bayesian models that uses MCMC. The code for this is provided in the Appendix and is also availabe on GitHub along with all other relevant code. A simple illustration of how the prior datasets are generated in R, showing this difference is given below:

```r
L = 5
n = 10000

diff_size <- numeric(n)

for (i in 1:n){
  u = runif(1)
  diff_size[i] = -L * log(1 - u)
}

# Converting the difference into either a negative or a positive
difference = diff_size * sample(c(-1, 1), size = n, replace = TRUE)

# A quick way to simply add a difference of 0,
# ...which we know happens rougly 50% of the time
difference <- c(difference, rep(0, 10000))
hist(difference, breaks = 40, col = "skyblue2")
```



Once we have done this for a large number of datasets, we can work out the posterior probabilities of H_0 and H_1 and work out the information content by calculating the ***average*** posterior entropy across different sample sizes. This is shown in the next section.

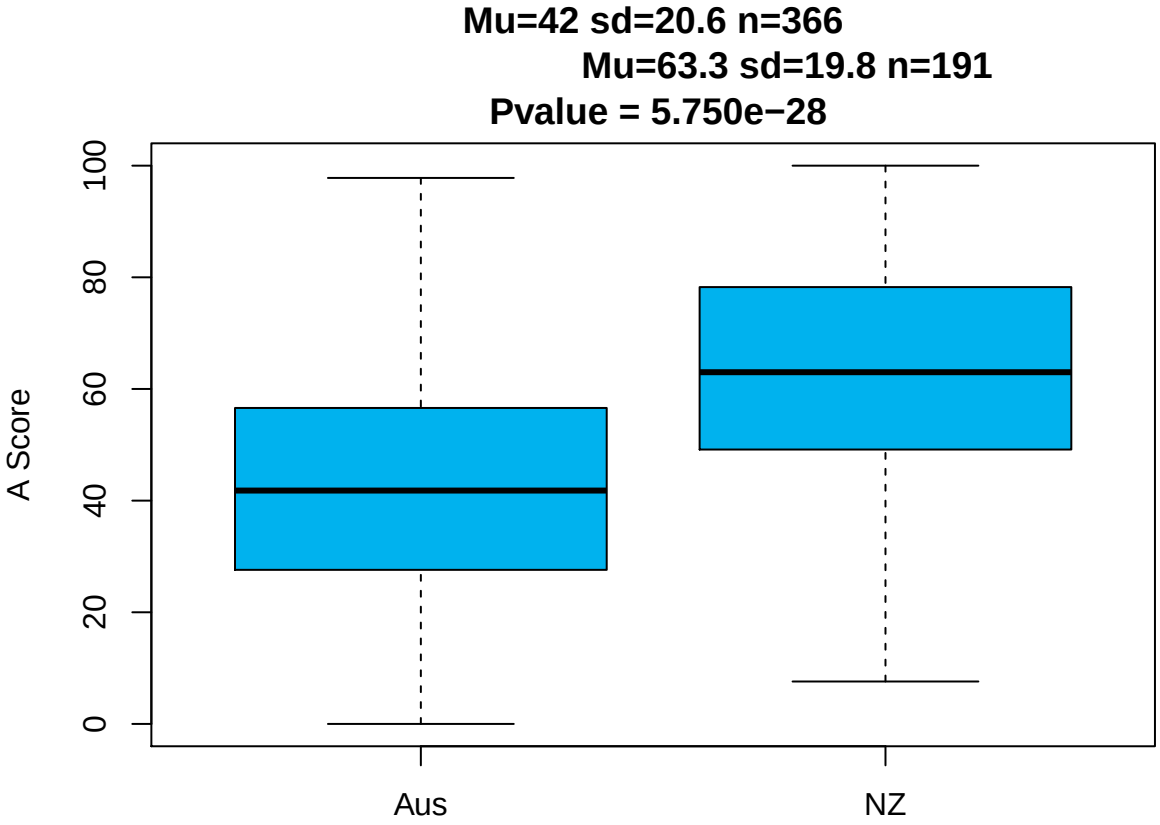
4.5 A Simple Example

Before we implement our **testing prior** t-test model *(where we specify in the prior that $\mu_1 = \mu_2$, roughly 50% of the time)*, we can generate a set of *simpler* datasets to see the relationship between the *p-value*, *entropy* and the *posterior probabilities*. These two groups in these datasets have a pre-defined constant standard deviation ($\sigma = 20$), and random mean between a range of 30.

A prior dataset for this simple example:

The boxplot below shows how a single data set might look like. Recall, that it does not have the testing prior implemented in it which is described in the previous subsection.

```
set.seed(2)
AusMean <- sample(x = 35:65, size = 1)
Aus <- rnorm(n = sample(x = 50:500, size = 1), mean = AusMean, sd = 20)

NzMean <- sample(x = 35:65, size = 1)
Nz <- rnorm(n = sample(x = 50:500, size = 1), mean = NzMean, sd = 20)

# Two side-by-side boxplots
exam <- data.frame(score = c(Aus, Nz),
                   country = c(rep("Aus", length(Aus)),
                               rep("NZ", length(Nz))))

pval = t.test(score ~ country, data = exam)

boxplot(score ~ country, data = exam, ylab = "A Score",
        main = paste("Mu=", round(mean(Aus), 1),
                     " sd=", round(sd(Aus), 1),
                     " n=", length(Aus) ,
                     "\n Mu=", round(mean(Nz), 1),
                     " sd=", round(sd(Nz), 1),
                     " n=", length(Nz),
                     "\n Pvalue = ", ifelse(pval$p.value < 0.001,
                                            formatC(pval$p.value,
                                                    format = "e",
                                                    digits = 3),
                                            format(pval$p.value,
                                                   digits = 3)),
                     sep = ""), col = "deepskyblue2")
```



The data:

A set of 10,000 datasets were generated to implement this t-test model in JAGS *(again, without the testing priors)*, and the classical p-value was recorded along with the calculation of the posterior entropy and the posterior probability of H_0.

The following are two plots generated using the `smoothScatter` function in the graphics package in R. They show the relationship between entropy and the posterior probability of H_0, against the classical P-value.

In the first graph, we can see that the Entropy is the highest when the p-value is around ~ 0.08, indicating the highest *uncertainty*. This is re-assuring as in a classical case, this p-value would result in *"weak"* evidence against the null hypothesis. Meaning the null hypothesis could be true, but we have *weak* evidence against this hypothesis. And likewise, the alternative could also be true. Since we only have two states, and the p-value of 0.08 does not support either of them strongly, we get the result where the uncertainty/entropy is the greatest.

In the second graph, the posterior probability of H_0 is monotonically increasing with the p-value, which also makes sense as an increase in the p-value indicates less and less evidence against H_0 being true. In Bayesian terms, this translates to a greater posterior probability for H_0.

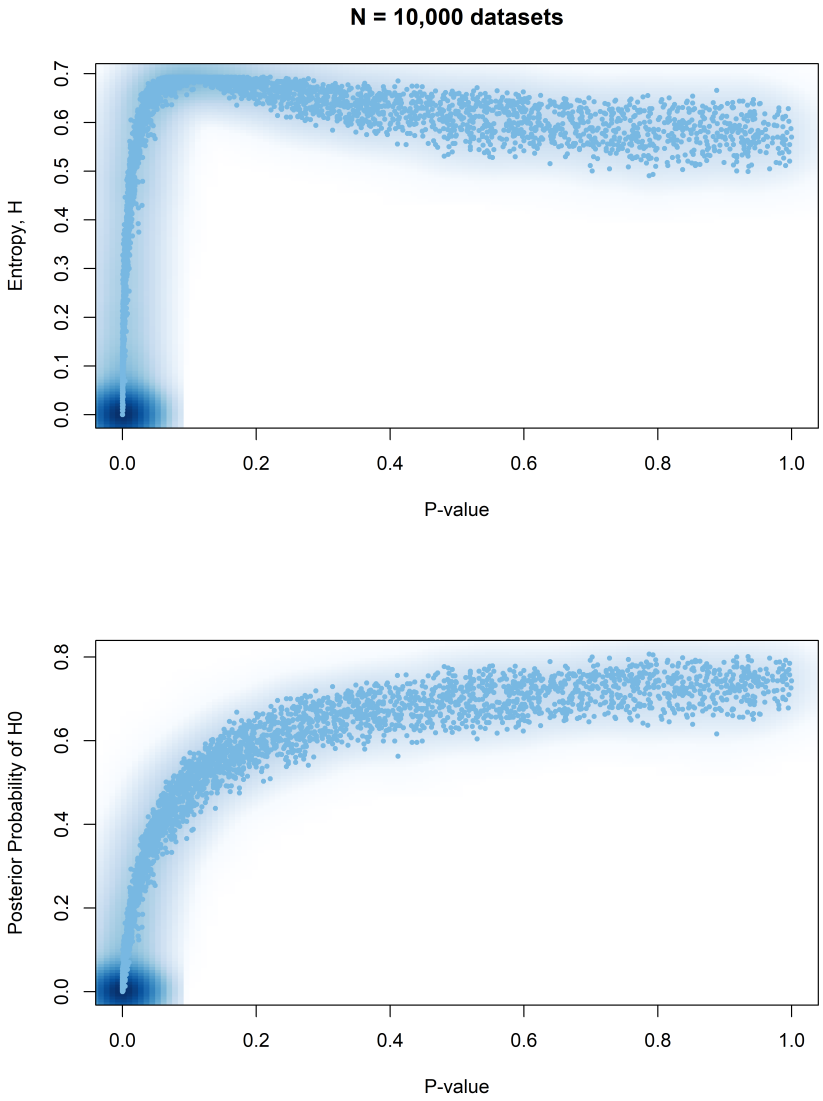


Figure 3: Relationship between the entropies, posterior probabilities and p-values generated in the t-test model

Recording probabilities for each generated dataset in the t-test model:

```
# Posterior probability of Ho
  p0[i] = mean(results$mu2 == results$mu1)

  # Posterior probability of H1
  p1[i] = 1-p0[i]

  # Posterior probability that Nz scores are better than Aus
  post_mu2_gt_m1[i] <- mean(results$mu2 >= results$mu1)

  # Classical P-value
  pvalue[i] = data$pval

  # Calculating Entropy
  H_entropy = (-p0[i] * log(p0[i])) - (p1[i] * log(p1[i]))
  H[i] = ifelse(is.nan(H_entropy), 0, H_entropy)

  # Just a quick correction to the format of the p values.
  # Looks nice this way.
  pvalue1 = as.numeric(ifelse(pvalue < 0.001,
                              formatC(pvalue, format = "e", digits = 3),
                              format(pvalue, digits = 3)))
  t_testData <- data.frame(p0 = p0,
                           p1 = p1,
                           post_mu2_gt_m1 = post_mu2_gt_m1,
                           H = H,
                           pvalue1 = pvalue1)
```

Average posterior probabilities and the average posterior entropy:

```
  mean(t_testData$p0) # Average Posterior Probability of Ho

   [1] 0.1686068

  mean(t_testData$p1) # Average Posterior Probability of H1

   [1] 0.8313932

  mean(t_testData$H) # Average Posterior Entropy

   [1] 0.2083943
```

5 Posterior Entropy vs. Sample Size

In the previous section, we have seen the relationship between entropy and posterior probability of H_0, against the p-value, with relatively similar sized datasets. This sections explores the effect of introducing larger sets of data on the posterior entropy and on the information content.

As mentioned in the previous section, entropy is the amount of uncertainty or the amount of information that is missing, given a probability distribution. Computing entropies given different sized data sets will allow us to answer queries such as "*how big of a dataset do we need to answer a particular question*". Meaning, how big does our sample size need to be for the entropy to be zero *(or close to zero)*. So we can minimise the... *uncertainty*.

When the Entropy is zero, that means there is no remaining uncertainty, and we have pretty much answered the question. So, back our t-test model, how big of a data size would we need for the entropy to become zero, on average?

5.1 Testing prior t-test model against big datasets

The testing prior t-test model in this section uses the same approach to calculate posterior probabilities. However, there are eight different categories for the sizes of data sets which we have specified for our the t-test to be able to use *(N = 100, 200, 300, 500, 1000, 5000, 10000, 25000)*.

In total, almost 15,000 t-tests were performed using the JAGS model. Each test randomly chose a sample size group *(such as N=300)*, and this sample represented the number of observations each group will have for that one [out of 15,000] t-test.

The means for each group is the same as per the model specifications in the previous section:

- 50% probability to the prior hypothesis that $\mu_1 = \mu_2$
- 50% prior probability that $\mu_2 = \mu_1$+**difference**

5.2 Entropy vs Sample Size

The graph below shows the computed entropies for each individual t-test model, for all eight sample size groups. The `jitter` function was used to add noise to each point, in order to better see where these entropies were landing, especially for bigger data size groups.

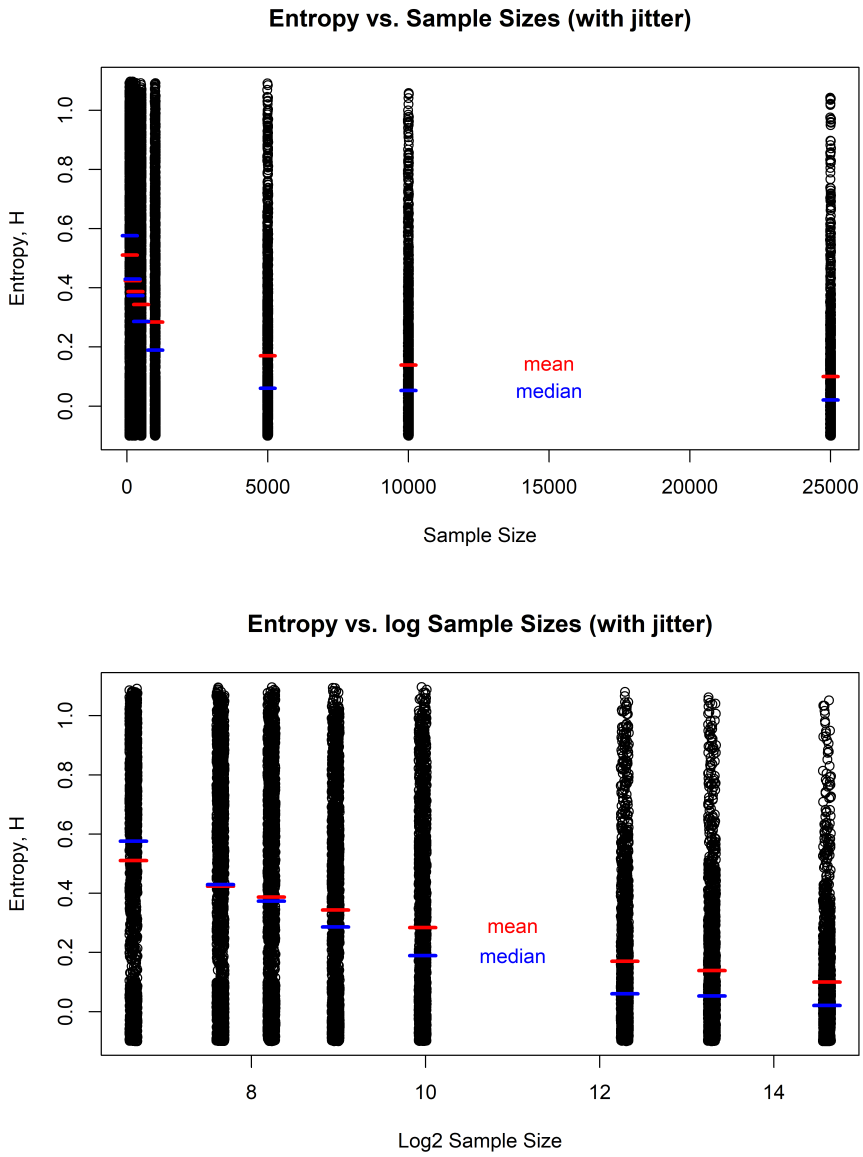


Figure 4: Relationship between the sample size and entropy. The bars show the estimated average of the entropy, and how that varies along with the data size

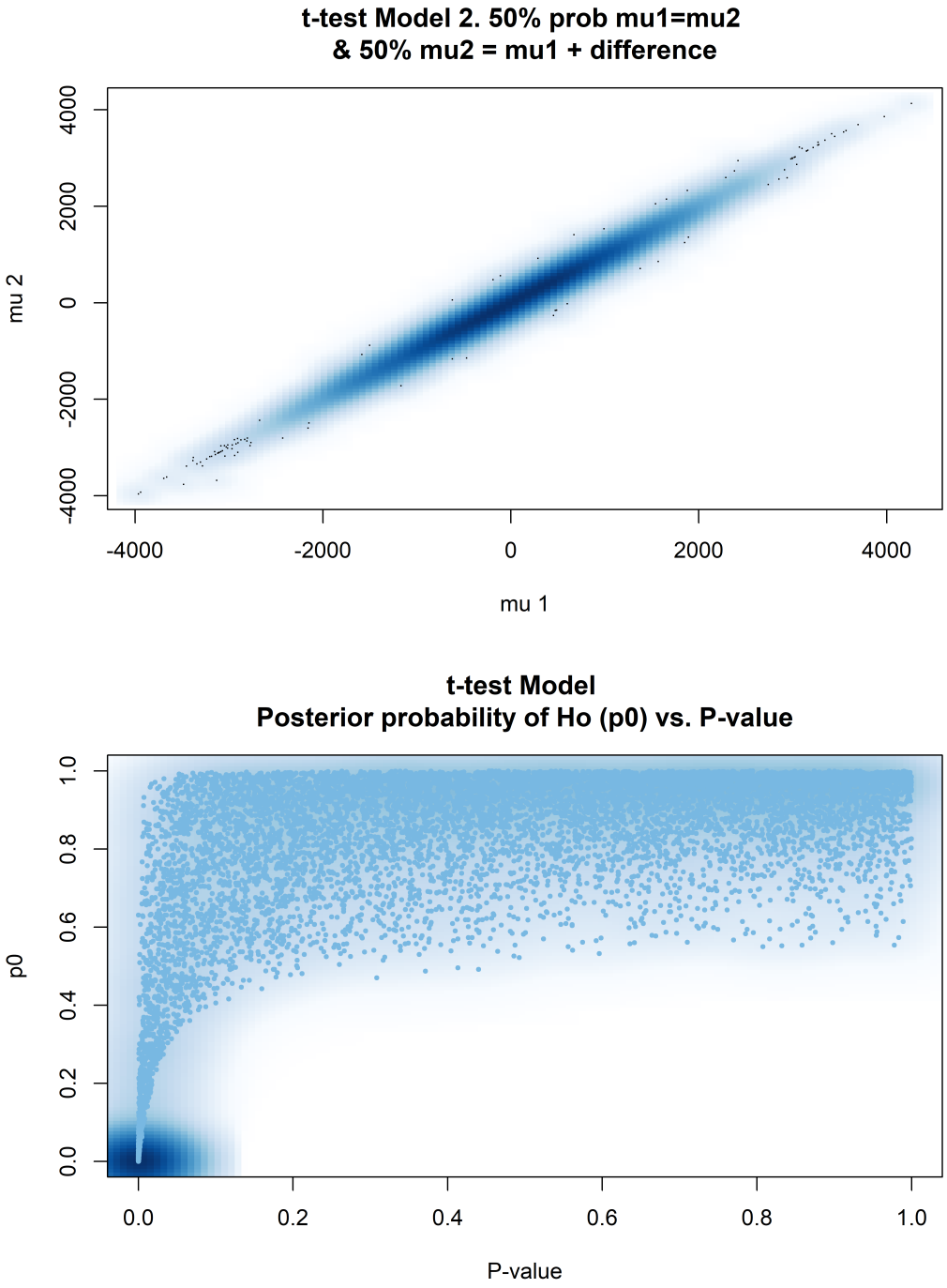


Figure 5: Prior relationship between the two means (above). Posterior probability of Ho, against the P-value (below)

5.3 Information content in each data group

Recall that the Shannon's mutual information is the difference between prior entropy and the average posterior entropy.

$$I(\Theta, X) = H(\Theta) - H(\Theta|X)$$

For our testing prior t-test model, the prior entropy is:

$$H = -p_0 \log_2(p_0) - p_1 \log_2(p_1)$$
$$H = -0.5 \log_2(0.5) - 0.5 \log_2(0.5)$$
$$H = 1$$

as the prior probabilities for H_0 and H_1 are P = {0.5, 0.5}
This means that we can calculate the posterior entropies for each t-test performed, across different sized data. This will allow us to calculate the information content and for us to see how it changes with the size of the data. Entropy calculation for each of the 15,000 t-tests are again done using JAGS code. The following table shows the values from 10 such t-tests (out of 15,000):

```
ttest$information = 1 - ((-ttest$p0 * log2(ttest$p0)) -
                           (ttest$p1 * log2(ttest$p1)))

xtable(head(ttest[, c("sample_size", "p0", "p1", "H", "information")], 10))
```

	sample_size	p0	p1	H	information
1	10000	0.00	1.00	0.00	
2	25000	0.00	1.00	0.00	
3	10000	0.98	0.02	0.13	0.87
4	25000	0.00	1.00	0.00	
5	25000	0.90	0.10	0.48	0.52
6	10000	0.96	0.04	0.22	0.78
7	10000	0.00	1.00	0.00	
8	25000	1.00	0.00	0.03	0.97
9	10000	0.99	0.01	0.11	0.89
10	10000	0.00	1.00	0.00	

Table 2: Posterior probabilities and entropy at each simulated data

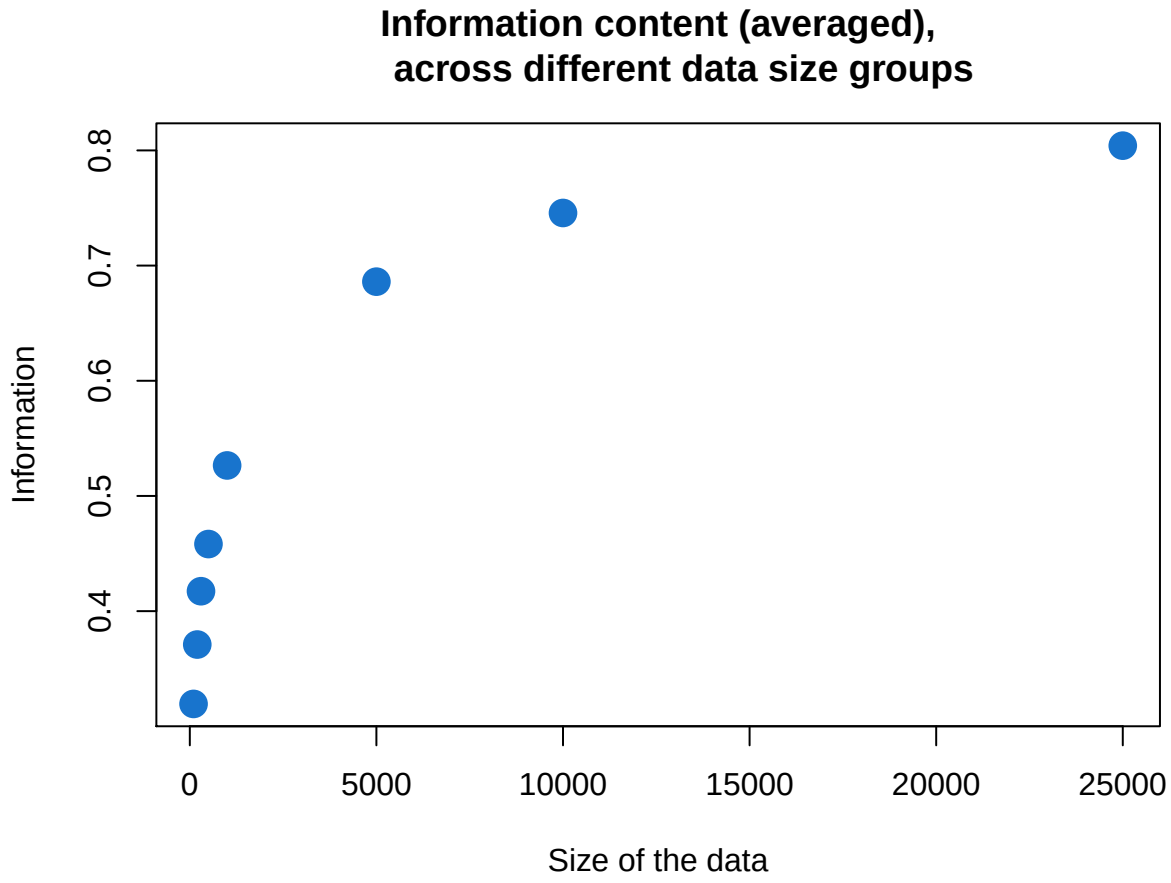


Figure 6: Average information content for the t-test model, across different data sizes

And finally, an easy way to look at the all the calculated information from each t-test, across each data size group is simple take the average and plot it against their respective sample sizes.

```
# Ignoring the NaN values, due to log(0) and the H being 0
ttest <- ttest[which(!is.nan(ttest$information)), ]

avg_info <- tapply(ttest$information, ttest$sample_size, mean)

plot(avg_info ~ as.numeric(names(avg_info)),
     main = "Information content (averaged),
     across different data size groups",
     ylab = "Information",
     xlab = "Size of the data",
     col = "dodgerblue3", pch = 16, cex = 2)
```

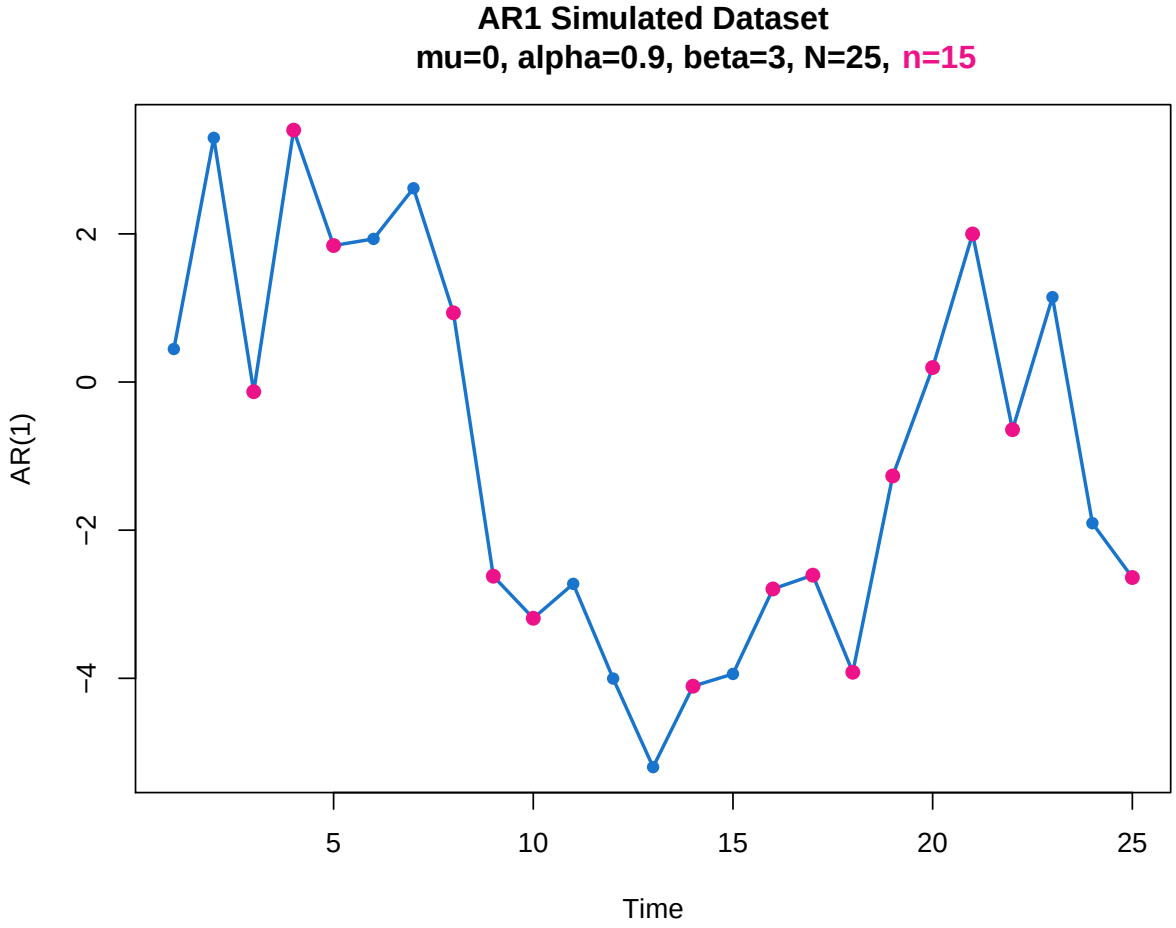
6 AR(1) time series distribution observed with & without noise

From a given underlying distribution, say, an AR(1) time series distribution with 25 points, we might only observe a subset of values (n=10). We might want to know that, given a Gaussian distribution, what is the posterior probability of the missing 15 points?

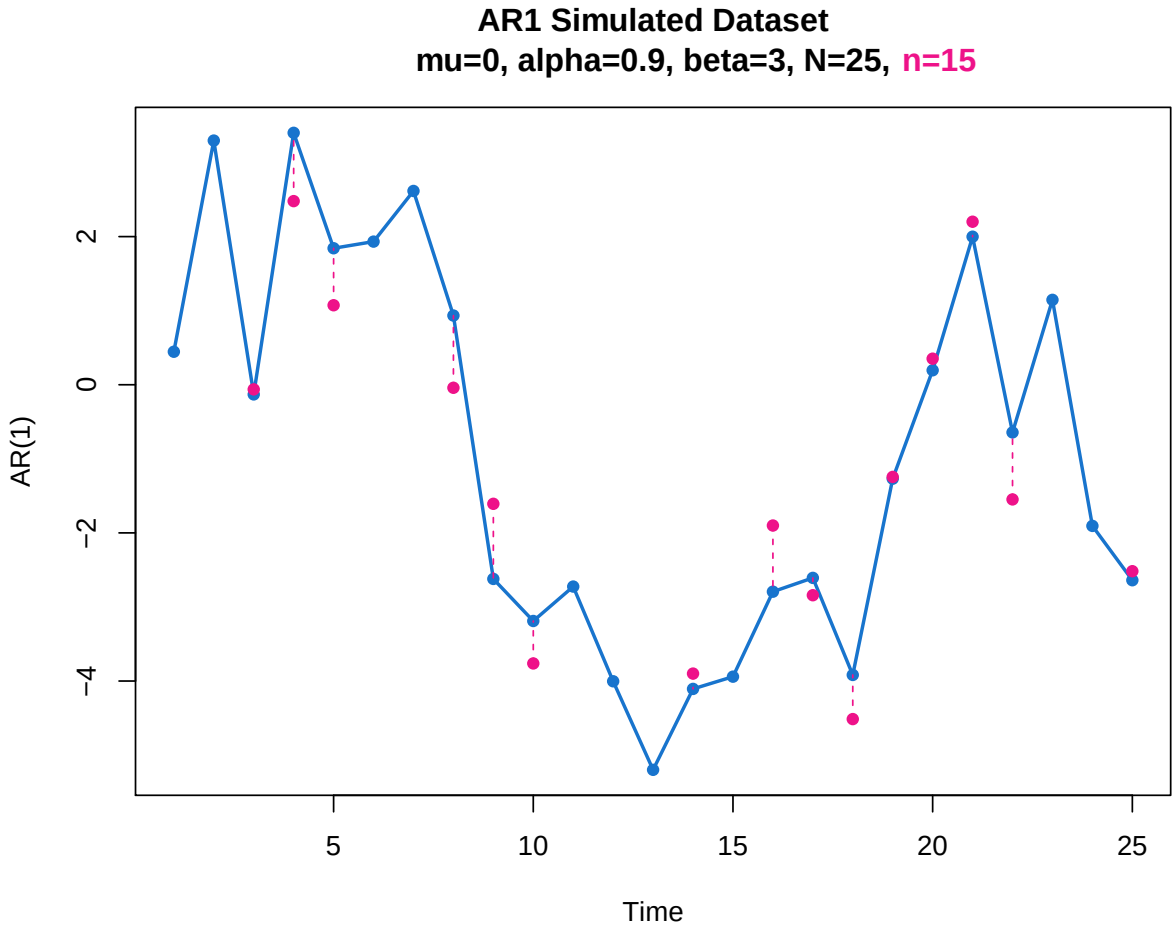
In another words, condition on those 10 points that we know know, what is the posterior distribution of all the remaining 15 points?

Hence, these 10 points allows us to see what the posterior distribution for $y_1, y_2, y_6, y_7, y_{10}$,..., y_{25} might look like.

First case:
This is when we observe these 15 observations **without** noise. (Remember, in reality we do not know the underlying blue curve of the graph below, only the observed 15 pink points)



Second case: This is when we observe these 15 observations **with** noise.



7 Covariance Function

The variance of a random variable measures how spread out the observations are from its mean, on average. And similarly, the Covariance function calculates how "rough" the surface is from the mean vector, on average or how strong the correlation is between points. For a $n \times d$ matrix X, $K(X)$ denotes the $n \times d$ matrix whose (i, j) element is $K(x_i, x_j)$.

Using Gaussian Processes to find the posterior distribution of our missing points Gaussian processes assumes that the output for any sets of input points follows a multivariate normal distribution. Just like a Gaussian distribution is fully specified by its mean and covariance matrix, a Gaussian process is specified by a mean vector and a covariance function. The **Covariance matrix** is constructed from the **Covariance function**:

$$K_{ij} = K(x_i, x_j)$$

There are two common types of covariance functions that we can use to make a covariance matrix, Σ.

1. Squared Exponential Covariance Function *(or the radial basis)*:

$$\mathrm{K}(x, x') = \sigma^2 e^{\left[-\frac{1}{2}\left(\frac{x-x'}{l}\right)^2\right]}$$

2. Absolute Covariance Function:

$$\mathrm{K}(x, x') = \sigma^2 e^{\left[-\left|\frac{x-x'}{l}\right|^{\nu}\right]}$$

where ν is a *roughness* parameter and $0 < \nu \leq 2$

8 Known/Pre-defined Hyperparameters

If the hyperparameters of the above two Covariance functions are known, then we can simply take posterior samples from the multivariate Gaussian distribution to estimate the distribution of our missing observations. If, however, these hyperparameters are not known, then we have to estimate them using MCMC techniques, given some prior knowledge and a likelihood function.

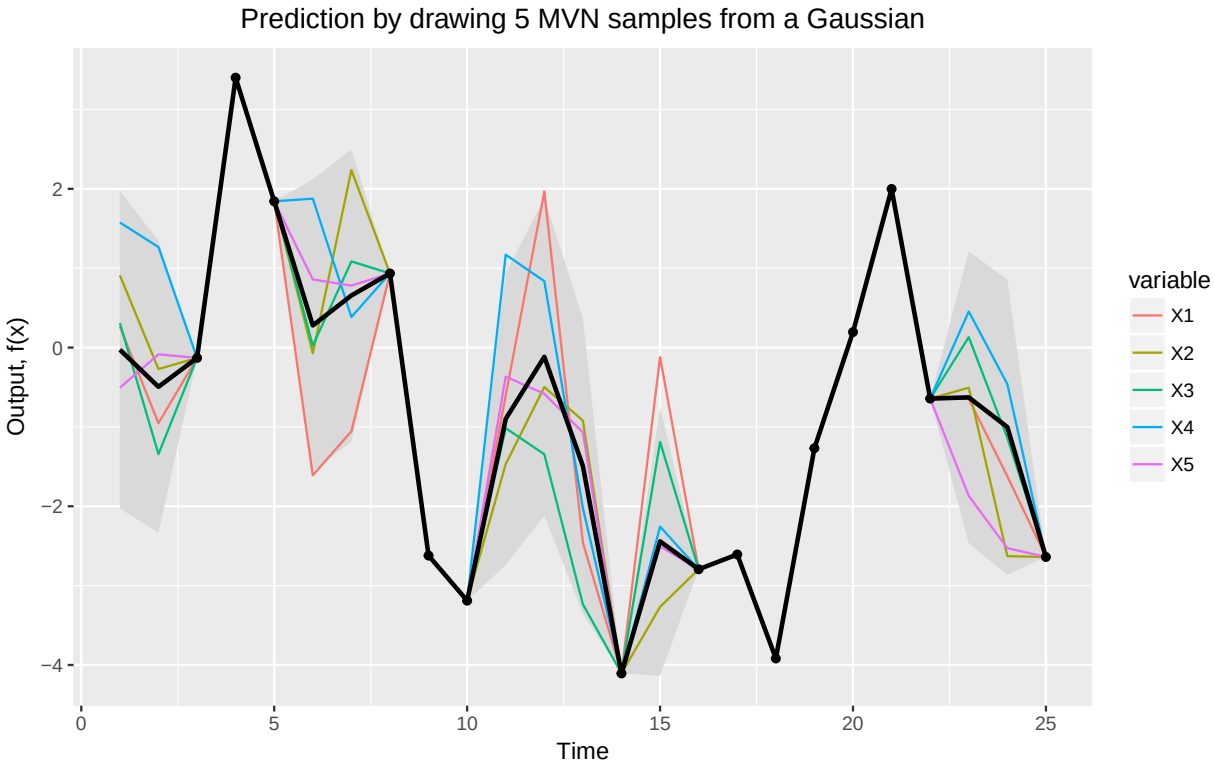
Let's suppose we know these hyperparameters and we draw multivariate normal samples from a Gaussian using the **Absolute Covariance Function**

8.1 Drawing 5 Multivariate Normal Samples from a Gaussian, given a mean vector, μ and a Covariance Matrix, Σ

This prediction of those 15 [unobserved] points is just, a **Gaussian Distribution**.

```
# THE DATA
obs <- data.frame(x = samp_index, y = samp)

# This may or may not make a difference, but lets order it
obs <- obs[order(obs$x), ]

# FUNCTIONS
set.seed(2)
x_predict <- attributes(ar1)$tsp[1:2][1]:attributes(ar1)$tsp[1:2][2]
l <- 1
nu <- 2
sigma <- 1

# Absolute Covariance Function - Using absolutes insead of Squared distances
SE <- function(Xi, Xj, l) {sigma^2 * exp(-(abs(Xi - Xj)/l)^nu)}
cov2 <- function(X, Y) {outer(X, Y, SE, l)}

## Taking the inverse of the Covariance Matrix
cov_xx_inv <- solve(cov2(obs$x, obs$x))

# Mean Vector
Ef <- cov2(x_predict, obs$x) %*% cov_xx_inv %*% obs$y

# Covariance Matrix
Cf <- cov2(x_predict, x_predict) - cov2(x_predict, obs$x)  %*% cov_xx_inv %*%
  cov2(obs$x, x_predict)

# Replacing the negatives with 0 in the diagonals of the Covariance Matrix
diag(Cf)[diag(Cf) < 0] <- 0
```

```r
no_mvrSample <- 5
values <- mvrnorm(no_mvrSample, Ef, Cf)

dat_original <- data.frame(x = x_predict, t(values))
dat <- melt(dat_original, id = "x")

ymins = rep(Ef - 2*sqrt(diag(Cf)), no_mvrSample)
ymaxs = rep(Ef + 2*sqrt(diag(Cf)), no_mvrSample)

ggplot(dat, aes(x = x, y = value)) +
  geom_ribbon(aes(ymin = ymins, ymax = ymaxs), fill = "grey85") +
  geom_line(aes(color=variable)) +
  geom_point(data = obs, aes(x = x, y = y)) +
  geom_line(data = NULL, aes(x = rep(x_predict, no_mvrSample),
                             y = rep(Ef, no_mvrSample)), size = 1)+
  scale_y_continuous(name = "Output, f(x)") +
  xlab("Time") +
  ggtitle("Prediction by drawing 5 MVN samples from a Gaussian") +
  theme(plot.title = element_text(hjust = 0.5))
```



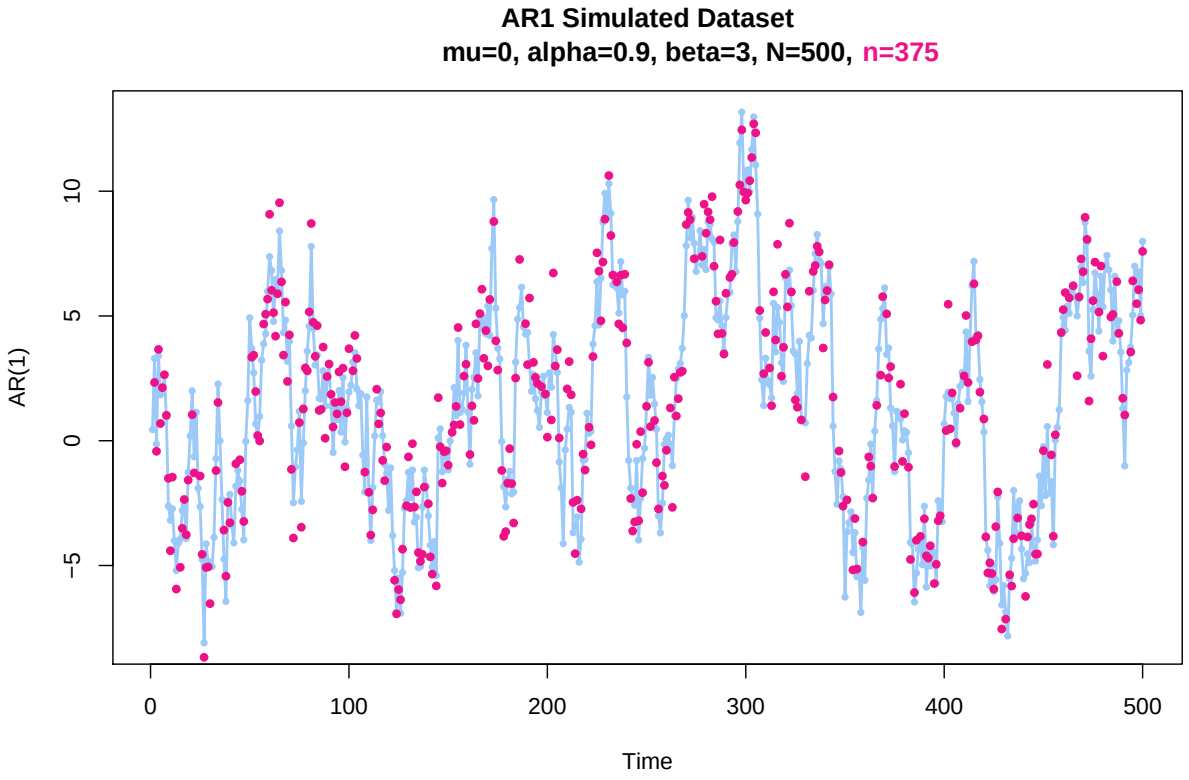
8.2 Larger set of observations (N = 500)

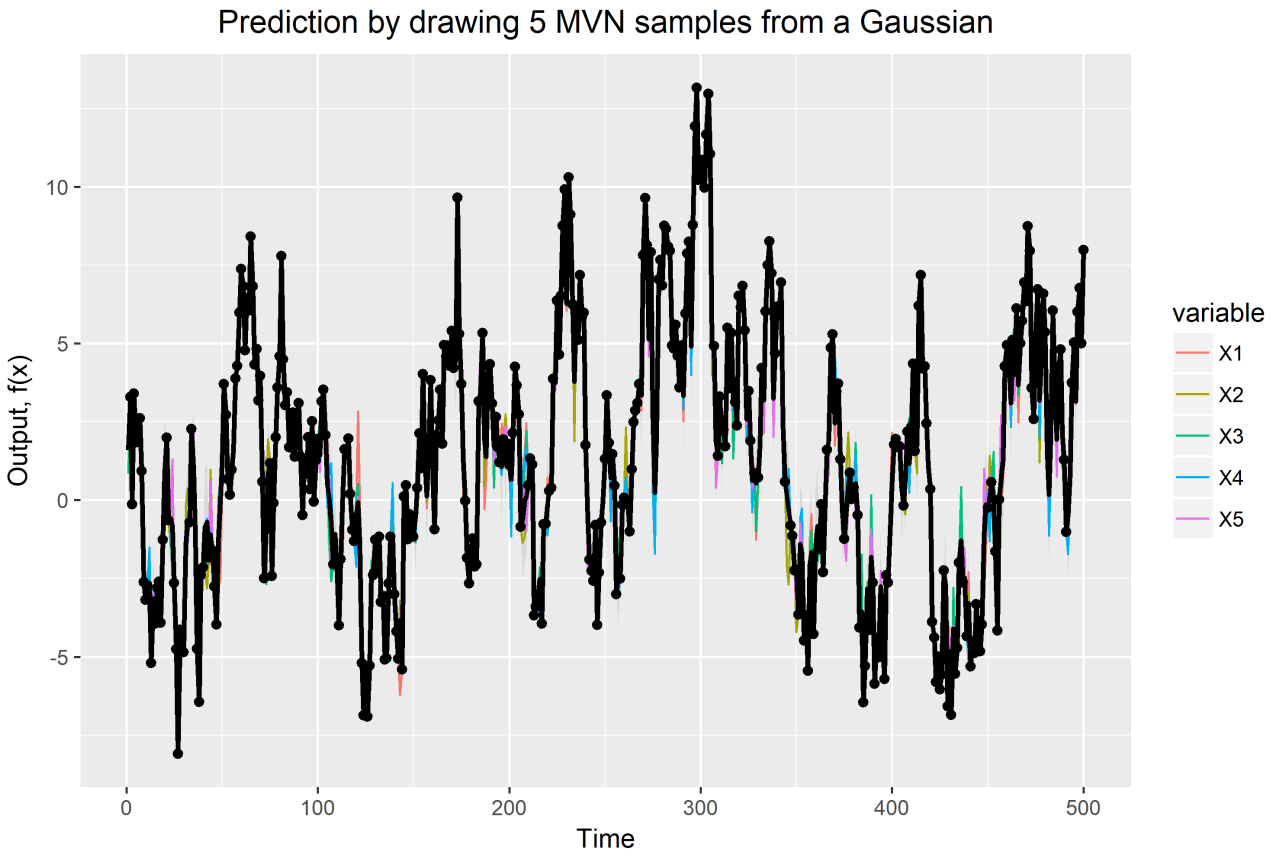
Similarly, we can apply the above to an AR(1) simulated dataset, that contains N = 500 observations. Again, we can figure out that, given we know the specifications of the hyperparameters, how well does our posterior samples from a multivariate Gaussian distribution compare to the underlying model.

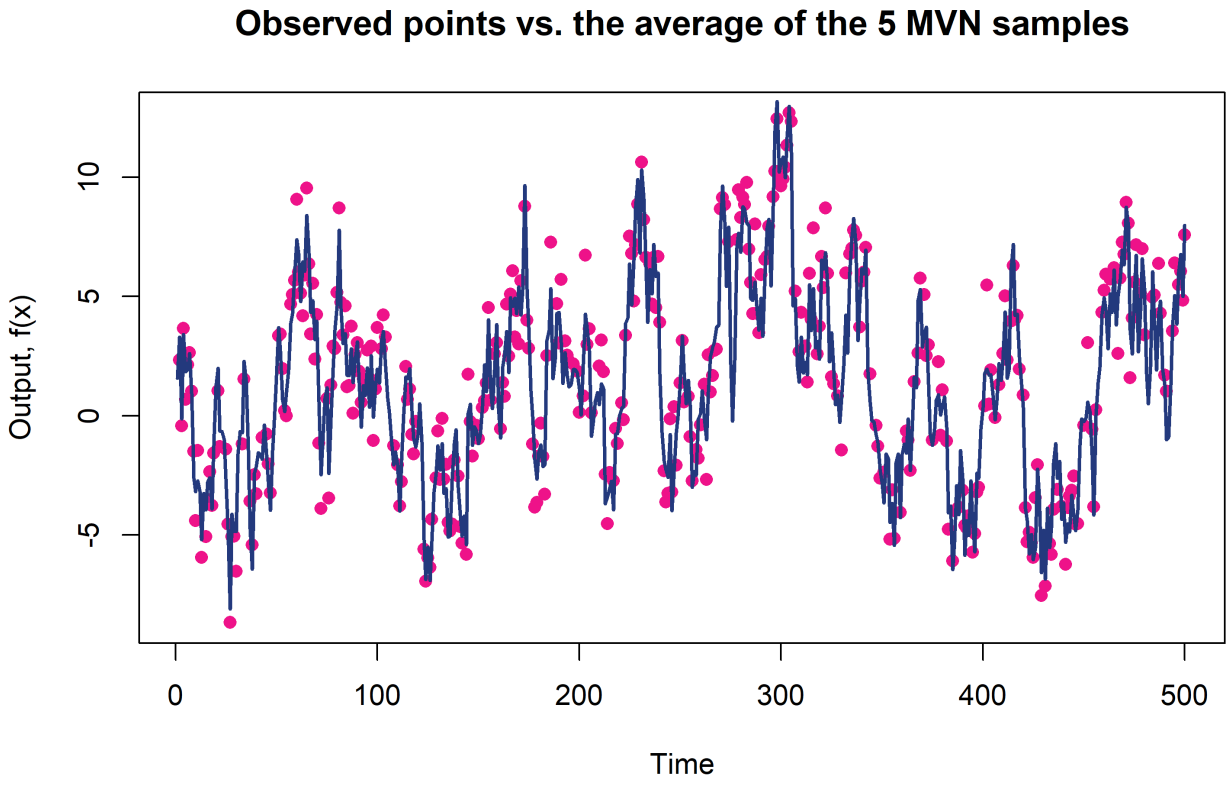
Below is a similar graph of an AR(1) time series model, with similar parameters from which we only observe 75% of the data with noise. Keep in mind that, in reality, we do not actually observe the full underlying curve *(in light blue)* as it is not known. So, we are only working with the 375 observations that were observed.

```
library(MASS)
#### PARAMETERS
N = 500
alpha = 0.90
B = 3
mu = 0
####

set.seed(2);ar1 = mu + arima.sim(model = list(ar=alpha), n = N, sd = sqrt(B))

## Data observed with noise
set.seed(2);samp_index = sort(sample(x = 1:N, size = floor(length(ar1) * 0.75),
                                     replace = FALSE))
samp = ar1[samp_index]

# Data observed with noise
set.seed(3);data_observed_noise = samp + rnorm(n = length(samp), mean = 0, sd = 1)
```

```
# Plotting
main = paste("AR1 Simulated Dataset\nmu=", mu, ", alpha=", alpha, ",
             beta=", B, ", N=", N, ",", sep = "")
plot(ar1, main = main, ylab = "AR(1)", type = "o", pch = 16, col = "#9ccaf8",
     lwd = 2, cex = 0.7)
title(paste("\n", paste(rep(" ", nchar(main)+15), collapse = ""), "n=",
            length(samp_index), sep = ""), col.main = "deeppink2")

points(x = samp_index, y = data_observed_noise, pch = 16,
       col = "deeppink2", cex = 0.85)
```


AR1 Simulated Dataset
mu=0, alpha=0.9, beta=3, N=500, n=375
AR(1)
10
5
0
−5
0
100
200
300
400
500
Time


Prediction by drawing 5 MVN samples from a Gaussian
Output, f(x)
10
5
0
-5
0
100
200
300
400
500
Time
variable
X1
X2
X3
X4
X5


Observed points vs. the average of the 5 MVN samples
Output, f(x)
10
5
0
-5
0
100
200
300
400
500
Time

9 Absolute Covariance Function

9.1 Posterior Distribution of θ

$$P(\theta|y_{1:N}) \propto P(y_{1:N}|\theta)P(\theta)$$

9.2 Determining Hyperparameters

The hyperparameters are often estimated by finding the parameters such that they maximise the likelihood function given the data we have observed. In the covariance function, there are three hyperparameters $\{\sigma^2, l, \nu\}$.

The prediction of the y given θ, the likelihood of observing the data that we have seen, $P(y|\theta)$, is just a Gaussian Distribution. This likelihood of this is:

$$P(y|\theta) = \frac{1}{[\det\ 2\pi\mathbf{K}(\theta)]^{1/2}} e^{-\frac{1}{2}y^T \mathrm{K}^{-1}(\theta)y}$$

The Gaussian distribution is fully specified by its mean, μ, and covariance matrix, K, and therefore the log-likelihood function is just the log of the probability density function:

$$\log\ \mathcal{L}(y|\sigma^2, l, \nu) = -\frac{\mathrm{N}}{2}\log(2\pi) - \frac{\mathrm{N}}{2}\log(\det\ \mathbf{K}) - \frac{1}{2}\left[(y-\mu)^T\mathbf{K}^{-1}(y-\mu)\right]$$

where y is an N x 1 data vector, μ is a mean vector

9.3 Using Metropolis-Hastings algorithm to estimate the hyperparameters

As previously seen with the t-test and the AR(1) model, the hyperparameters are approximated using the Metropolis-Hastings algorithm (MCMC method), for generating samples from a posterior distribution. In this case, that distribution is the Gaussian distribution and the function is the absolute covariance function.

Recall, that the data is still that 75% noisy data (n = 375) which is observed with noise at different points in time. The log-likelihood function, which is a function of these three hyperparameters, and which involves the Covariance Matrix K, the data matrix y, the the mean vector μ, gets evaluated given this data.

So basically, the purpose of the likelihood function is choosing the hyperparameters $\{\sigma^2, l, \nu\}$, that best fits the data, which we have observed, and which we know is fixed.

9.4 Check Posterior for Convergence

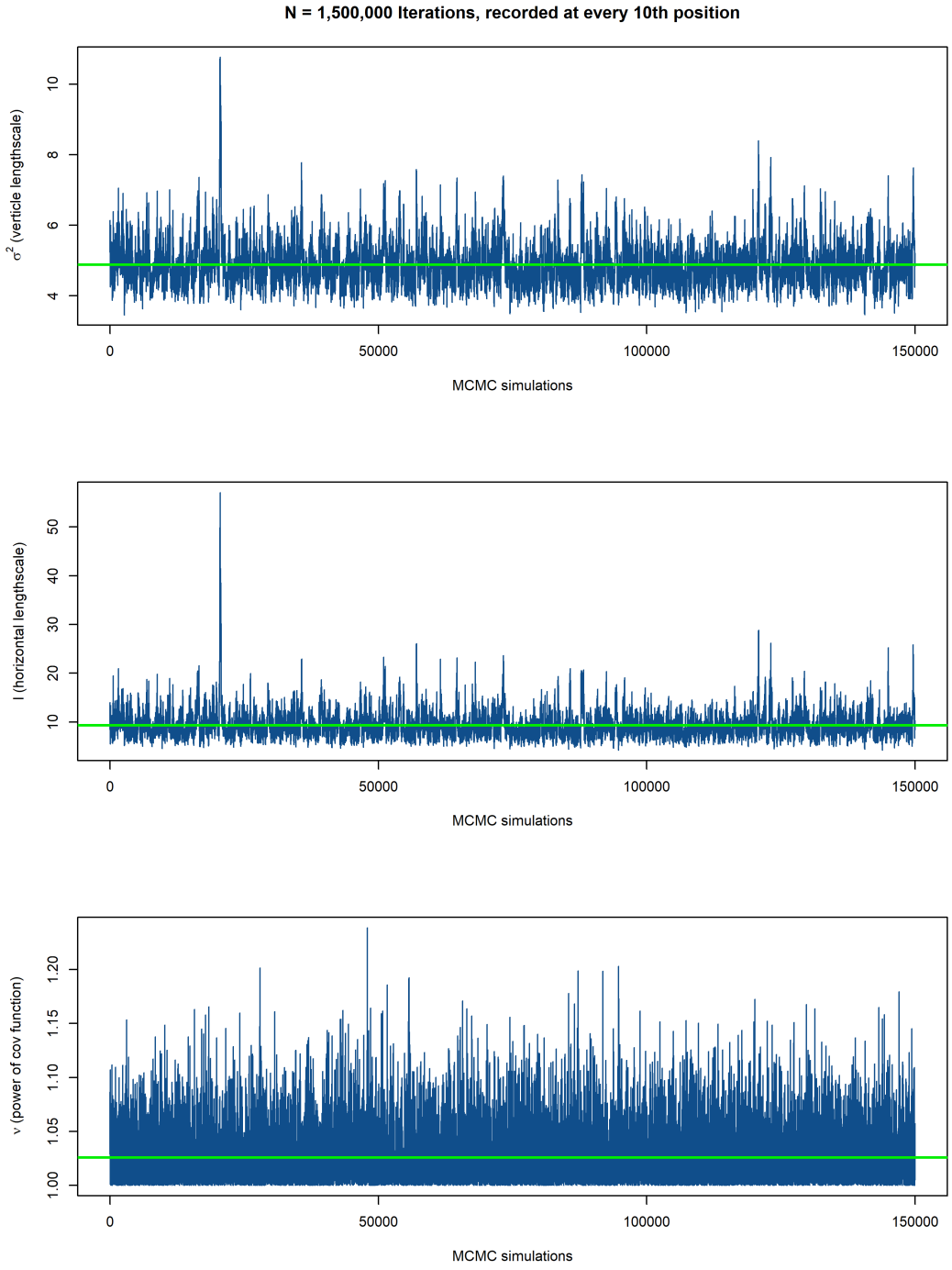


Figure 7: Estimates of the hyperparameters using the Absolute Covariance Function

10 Using Posterior estimates from MCMC to compute Gaussians

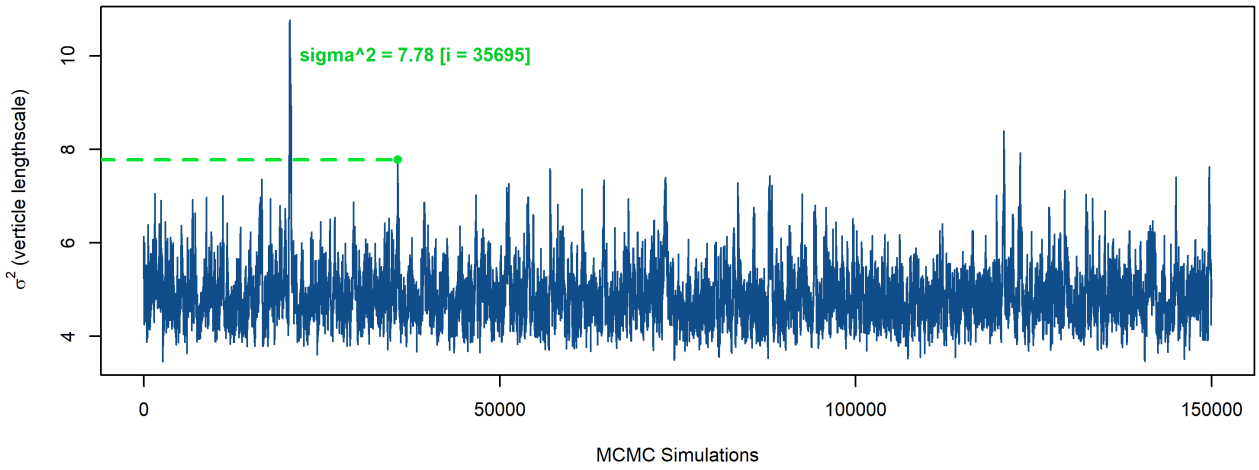
After running the MCMC iteration, we are able to see how well the posterior distribution of the hyperparameters are mixing. The MCMC sampler seems to be exploring the parameter space efficiently, and it does not accept or reject too many proposals. If too many proposals are rejected, we need a lot of simulations to generate a sufficient number of parameter samples. If too many proposals are accepted, then we don't gain much information about the underlying distribution. In the above case the acceptance probability was... meaning on average the MCMC is accepting and rejecting new proposals at a reasonable rate - which is exactly what we want.

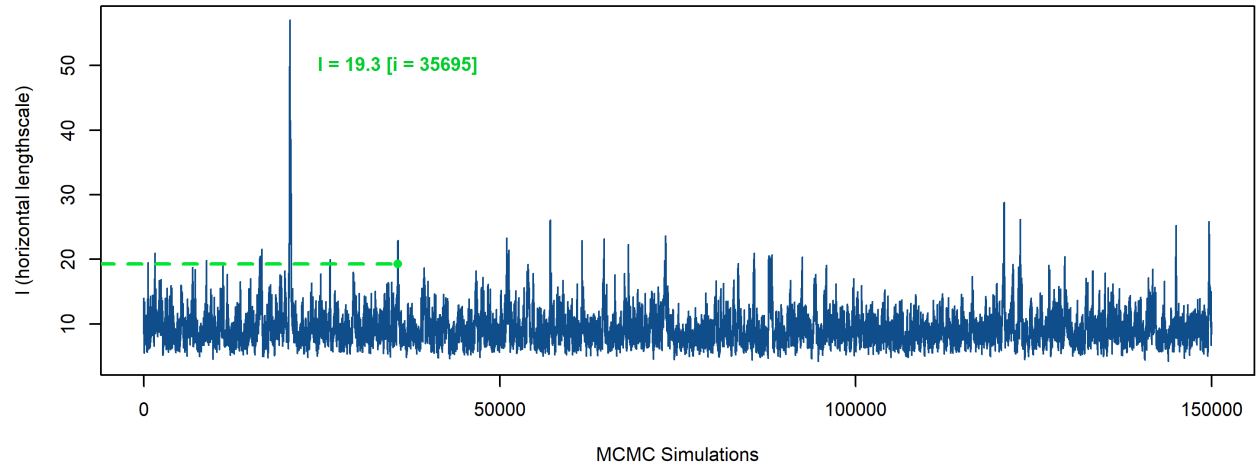
10.1 Which posterior estimate to use?

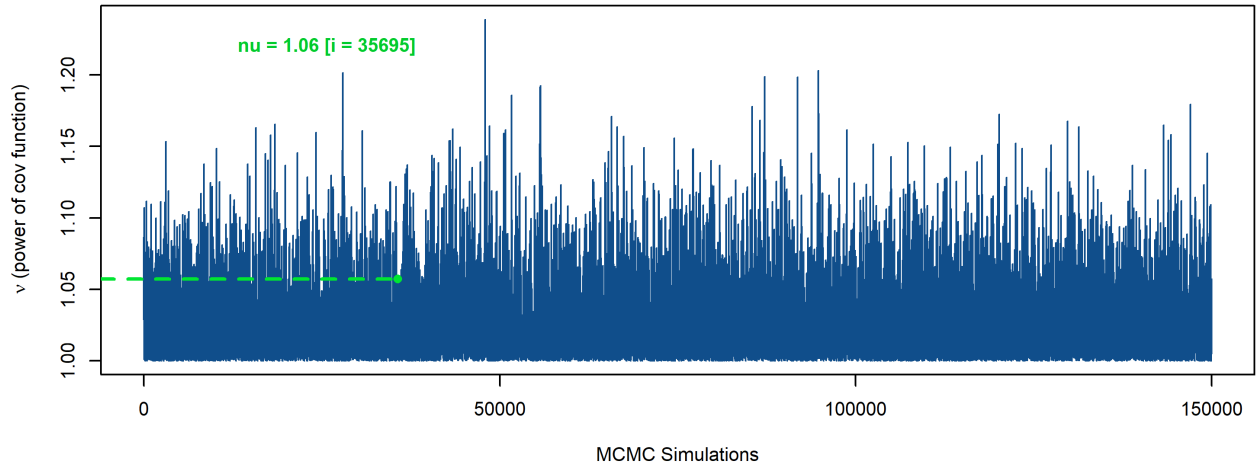
The above posterior distribution explores a range of values that our parameters can take. For example, looking at the first parameter {σ^2, the vertical lengthscale}, the MCMC sampler seems to be mixing quite well between 4 and 7. Similar signs of mixing can be seen for the other two hyperparameters throughout the entire simulation process. This means that ***any*** of these estimates at a given iteration can be the true estimate for our unknown hyperparameters.

sigma^2	l	nu
5.3	14.033	1.06
5.3	8.8874	1.06
5.3	8.8812	1.06
5.3	8.8812	1.06
5.3	8.8812	1.06
5.3	10.7966	1.06
.	.	.
.	.	.
.	.	.
4.8548	9.0964	1.0408
4.8538	9.0938	1.0408
4.896	9.0933	1.0431
4.8504	9.0893	1.0431
4.8484	9.0893	1.0468
4.8411	9.0486	1.0464

Table 3: 150,000 sets of estimates using the Absolute Covariance Function


sigma^2 = 7.78 [i = 35695]
σ² (verticle lengthscale)
4
6
8
10
0
50000
100000
150000
MCMC Simulations


l = 19.3 [i = 35695]
l (horizontal lengthscale)
10
20
30
40
50
0
50000
100000
150000
MCMC Simulations


nu = 1.06 [i = 35695]
ν (power of cov function)
1.00
1.05
1.10
1.15
1.20
0
50000
100000
150000
MCMC Simulations

11 Conclusion

11.1 Discussion

The aim of this project was to measure the information content, posterior variances and the entropy of datasets. This was done mainly with an AR(1) dataset using R, and a t-test model which was implemented in JAGS. The Metropolis-Hastings algorithm was used to do the MCMC simulations which helped compute the posterior distributions of the unknown parameters in these models.

As we were trying to implement the testing prior t-test model and run simulations on it using different sized datasets, what we also found interesting was to record the p-values. We did this for every t-test data that was generated and were also able to see and compare the relationship of the outputs that come from a Bayesian setting *(posterior entropy)*, to the outputs in a classical statistics setting *(p-value)*.

11.2 Limitations

Running simulations on the universities NeSI cluster limited the range of datasets which we wanted to explore. For e.g the covariance function uses the X data matrix as an input, for which several inversions happen. This was therefore computationally very time consuming, once the length of the data vector X became more than 500. So we had to stop at N=1000 in our models.

Recall, that we were able to use MCMC to calculate the distribution of the hyperparameters in our AR(1) time series model, however, it would have been good to do the hyperparameter estimation with varying data sizes. This would have also allowed us to observe how the information content changed as well as how the posterior variances of the hyperparameters changed, for different sized data points.

Finally, the correct estimation of the hyperparameters for the AR(1) time-series model using Gaussian processes, can also allow us to develop tools for adaptive scheduling of observations using Bayesian methods. This can be used in the detection of exoplanets and planetary orbit estimation.

12 Bibliography

References

[1] James O Berger. *Statistical decision theory and Bayesian analysis*. Springer Science & Business Media, 2013.

[2] Brendon J Brewer. Computing entropies with nested sampling. *Entropy*, 19(8):422, 2017.

[3] W Keith Hastings. Monte carlo sampling methods using markov chains and their applications. *Biometrika*, 57(1):97–109, 1970.

[4] M. G. Kendall, A. Stuart, and J. K. Ord, editors. *Kendall's Advanced Theory of Statistics*. Oxford University Press, Inc., New York, NY, USA, 1987.

[5] David J. C. MacKay. *Information Theory, Inference & Learning Algorithms*. Cambridge University Press, New York, NY, USA, 2002.

[6] David JC MacKay. Introduction to gaussian processes. *NATO ASI Series F Computer and Systems Sciences*, 168:133–166, 1998.

[7] Liam Paninski. Estimation of entropy and mutual information. *Neural computation*, 15(6):1191–1253, 2003.

[8] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2005.

[9] Christian P Robert and George Casella. The metropolishastings algorithm. In *Monte Carlo Statistical Methods*, pages 231–283. Springer, 1999.

[10] Murray Rosenblatt. *Gaussian and non-Gaussian linear time series and random fields*. Springer Science & Business Media, 2012.

[11] Ilker Yildirim. Bayesian inference: metropolis-hastings sampling. *Department of Brain and Cognitive Sciences, Univ. of Rochester, Rochester, NY*, 2012.

13 Appendix

The modelling code used in this dissertation is available publically on GitHub at: *https://github.com/satnamakl/statsproject*