CS391L Machine Learning: HW3 MCMC Sampling

Joel Iventosch UTEID: jwi245 Email: joeliven@gmail.com

March 4, 2015

1 Introduction

The goal of this assignment was to learn about Monte Carlo Markov Chain Sampling methods. Specifically, this assignment involved using the Metropolis-Hastings algorithm (reduced to simply the Metropolis algorithm if the chosen proposal distribution was a Gaussian - barring the addition of adaptive variances) to sample from a one and two dimensional mixture of two Gaussian distributions. The intent of the assignment was to familiarize ourselves with the Metropolis-Hastings sampling method by gaining hands-on experience implementing the algorithm on an actual sampling problem - "test driving the algorithm" so to speak. Another key goal of the assignment was to understand the significant impact that the variance of the proposal distribution has on the quality of the samples obtained from the algorithm - namely, if the variance is too large, then the acceptance rate becomes very small because the proposal distribution has a wide spread and is thus more likley to suggest candidate samples that are in low probability density regions of the target distribution (particularly in higher dimensional spaces). On the other hand, if the variance of the proposal distribution is too small, then the acceptance rate will be very high, but the algorithm will tend to produce candidate samples from more or less the same region, causing very slow exploration of the state space, and thus unsatisfactory convergence times. This assignment involved implementing the algorithm for a specific (provided) one-dimensional Gaussian mixture target distribution, showing the results of using correct and incorrect variances for the proposal distribution, and then extending our results to a two-dimensional Gaussian mixture target distribution of our choosing. The remainder of this report sets out to acheive these three tasks, in addition to explaining several other techniqes and aspects of my experiments. (Reference note: several parts of this introductory paragraph are paraphrased from class lecture, notes, and the homework assignment handout.)

2 Methodology

The Metropolis-Hastings sampling method actually ended up being surprisingly straight-forward algorithm to implement (more so than I was expecting) - particularly if the chosen proposal distribution is symmetric (as is the case with a Gaussian), in which case the methodology reduces down to the basic Metroplis algorithm. Implementing the algorithm involves these basic steps (adapted from the Marsland textbook, chapter 15):

1. Obtain a starting value, x_0, generated randomly from the proposal distribution, $q(x)$. This is the starting sample.

2. Generate a candidate sample, x^*, from the proposal distribution, $q(x)$, centered at $x^{(t)}$ (which is x_0 for the first iteration).

3. Obtain a random comparison number, u, generated by sampling from the uniform distribution over the interval $(0, 1)$.

4. Calculate the acceptance rat, A, for the candidate sample, x^*, based on the following equation:

   $A(x^*|x^{(t)}) = min[1, p(x^*)q(x^{(t)}|x^*)/p(x^{(t)})q(x^*|x^{(t)})]$

   where $p(x)$ is the probability density function (pdf) of our target distribution and $q(x)$ is the pdf of our proposal distribution. Specifically, $q(x^{(t)}|x^*)$ is the proposal distribution pdf evaluated at $x^{(t)}$, when centered at x^*, and $q(x^*|x^{(t)})$ is the the proposal distribution pdf evaluated at x^*, when centered at $x^{(t)}$.

5. If $A(x^*|x^{(t)}) > u$ then we accept the candidate by setting $x^{(t+1)} = x^*$.
6. If $A(x^*|x^{(t)}) < u$ then we reject the candidate by setting $x^{(t+1)} = x^{(t)}$.
7. Repeat steps 2-6 for either a given number of iterations or until some predefined convergence criteria is met.

Note, that although I only used a Gaussian for my proposal distribution in this set of experiments, I still designed my code to be used for the full Metropolis-Hastings algorithm, so that it could easily be extended to use with other non-Gaussian, asymmetric proposal distributions in the future. I intended to experiment with an asymmetric distribution for this assignment as well, but did not have time to run full experiments on any non-Gaussians unfortunately. As such, the formula for my acceptance rate simplified down to the basic Metropolis acceptance rate equation, involving only the ratio of the target distributions, namely:

$$A(x^*|x^{(t)}) = min[1, p(x^*)/p(x^{(t)})]$$

Although leaving the second portion of the full Metropolis-Hastings acceptance ratio in my code might have cost me a little bit of time complexity on my calculations, it obviously had no impact on the results since $q(x^{(t)}|x^*) = q(x^*|x^{(t)})$ for any symmetric distribution. Furthermore, I felt it was worthwhile, given that it makes the code much more extensible to future experimentation using a variety of proposoal distributions. Furthermore, if I were to use this code for a larger scale problem where the added time complexitiy of the (unnecessary) "Hastings" portion of the acceptance rate calculation played a bigger negative factor, it would be easy enough to implement a conditional check to determine if the proposal distribution being used was symmetric - and then adjust which acceptance rate equation is used based on that.

Calculating Error Measurement: Throughout my experiments I measured and tracked the "Error" of my generated sample set in order to determine an approximate level of accuracy attained. I based this error measurement on a sum squared error (SSE) approximation, by taking a histogram of the generated sample data and then measuring difference between the value (i.e. count) of each bin edged and value of the target distribution pdf evaluated at that same bin edge. Since the pdf a multivariate distribution still returns a single scalar value, the difference between the count of a particular histogram bin and the pdf evaluated at that bin edge can be thought of as an approximation of "how far above or below" the pdf curve the generated set of samples is. By then squaring and summing these individual error measurements for each bin in the historgram, I obtained an approximate overall error for the generated sample set, which could be calculated at any given time step in the algorithm. (Reference Note: I implemented this SSE measurement and generated the code for it entirely on my own, but I did brainstorm with several colleagues on the general idea of using an SSE-like approach for error measurements in this model).

3 Experiments and Results

To test the accuracy of my implementation of the Metropolis-Hastings algorithm for MCMC Sampling, I conducted two different sets of experiments: the first sampled from mixture of two Gaussians in one-dimension as the target distribution (as provided by part 1 of the hw3 assignment handout); the second sampled from a two dimensional mixture of two Gaussians that I created as the target distribution. These experiment sets are referred to from here forward as Series A and Series B.

SERIES A: 1-DIMENSIONAL

In this section I will summarize the results of my various experiments, and suggest some possible explanations for these results. Although throughout all of my experiments I used a Gaussian - and thus symmetric - proposal distribution, I was fairly pleased with the results of my experiments overall.

To give this MCMC Sampling method an initial

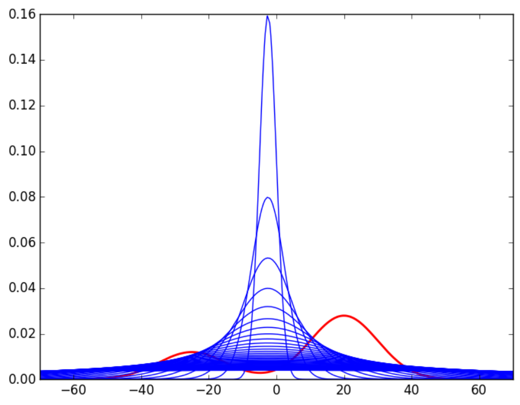


Figure 1: 1D target distribution in red with possible proposal distributions in blue

test drive (and to satisfy parts 1 and 2 of the hw3 assignment handout), I created and executed a Design of Experiments (DOE) that involved 93 experiments, tuning four key parameters of the model, namely:

1. the standard deviation of the proposal distribution, $sigma$
2. the number of iterations the algorithm was run for, n
3. the discard rate, d, defined as the percentage of final generated samples that are discarded
4. the initial burn quantity, b, defined as the number of initial samples that are discarded

Standard Deviation, $sigma$**:** The most time and experimental resources were spent trying to fine-tune the $sigma$ parameter of the model in the hopes of: **(a)** obtaining an optimal (or near optimal) standard deviation for this specific sampling problem at hand, and **(b)** gaining a better general understanding of the sampling results when a suboptimal $sigma$ parameter is chosen for the standard deviation of the proposal distribution. To achieve these two goals, I ran the first 32 Series A experiments adjusting $sigma$ and holding the other 3 parameters constant. More specifically, I started with $sigma = 2.5$ and incremented $sigma$ by 2.5 up through $sigma = 50$, at which point I incremented $sigma$ by 5 through $sigma = 80$. Additionally I tested for $sigma = 1, 90, 100, 125, 150, 200$ to see if anything interseting occurred at the far extremes on both the lower and upper ends. During these first 32 experiments, I held the other three parameters (number of iterations, n, discard rate, d, and initial burn quantity, b) constant at: $n = 5000$, $d = 0.0$, and $b = 1000$. Note that the decision to hold $n = 5000$ was based primarily on making the early (but numerous) experiments more time and computationally efficient, while still having n be substantially large enough to gain a quality (if not perfectly accurate) understanding of approximately what the optimal range for $sigma$ might be. On the other hand, the value of d was chosen as such based on early indications (from preliminary, non-listed experiments) that that might be the optimal value for d. The initial burn quantity, b, was chosen somewhat arbitrarily, although it utlimatley proved to be a fairly optimal value, as well.

The results of theses first 32 experiments was more or less as expected - very lowand very high values of $sigma$ both produced suboptimal results, with the optimal range appearing to be approximately between $sigma = [10, 30]$. Given that the provided target distribution was a mixture of two Gaussians, both with a standard deviation of 10 (and centered 45 units apart), this initial range for $sigma$ forthe proposal distirbution seems to make intuitive sense. Looking at the results more closely, as $sigma$ increased initially, the sampling accuracy (as measured by error - described in the methodology section above) improved dramatically. It then leveled out between $sigma = 10$ and $sigma = 30$, at which point it gradually began to increase again, becoming very poor by $sigma = 100$. The results of these first 32 experiments can be seen in figure 2, which plots $sigma$ vs. the final error of the sample set. Another interesting thing to note about the results is that, although the optimal range for $sigma$ is fairly evident, within that range the optimality (as seen in figure 2 by the error measurement) of $sigma$ is fairly unpredictable, as it oscillates up and down. My beliefe is that this is due to the stochasticity involved in the MCMC sampling algorithm used - namely, that because our initial starting

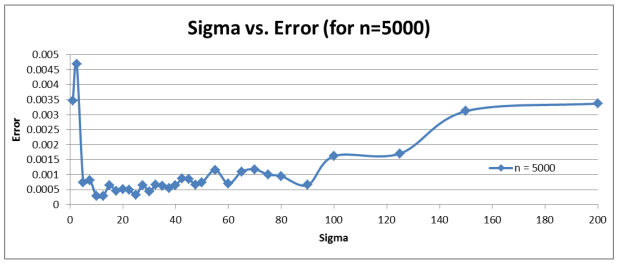


Figure 2: Series A, Exp 1-33: Sigma vs Error

sample is randomly generated within our proposal distribution, different experimental runs might have slightly (or maybe even significantly) different results, even given the exact same parameters. For instance, on one run the starting sample might lead to an "optimal" location within the target distribution, while on another run it might lead to a worse starting point. Although, the Metropolis-Hastings algorithm eventually produces a set of samples that converges on the target distribution regardless of the starting point, when using $n = 5000$ (and when making fine-grained optimality comparisons) the stochasticity of the initial starting sample can have an effect - and I believe that is one major factor that accounts for the oscillation apparent in figure 2. That said, further experimentation would be needed to prove this - for instance, repeating each of these first 32 experiments 10 times and taking their averages as results. However, given current time constraints, that is left as future work, and we will live with the oscillator factor for now.

Number of Iterations, n: In the next 29 Series A experiments I began to fine-tune the optimal number of iterations to run the experiments for - as a means to further fine-tune the *sigma* parameter. To do this, I selected the top values of *sigma* from the first 32 (which ended up being $sigma = [10, 30, stepsize = 2.5]$) experiments and ran them all again, still holding d and b constant at the same levels, but now with $n = 10000$, then $n = 25000$ and finally with $n = 50000$, reducing the number of possible *sigma* values considered by two at each step. (Note, the actual value of b was adjusted throughout, but this represented approximately the same burn *rate* as the previous 32 experiments.) The results of this next set of experiments can be seen in figure 3. Not surprisingly, the error diminishes as the num-

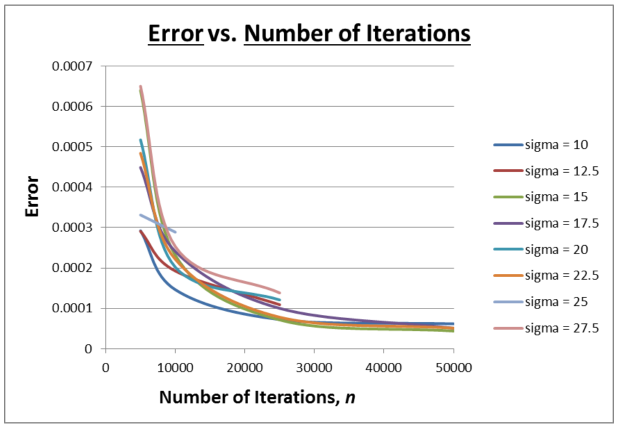


Figure 3: Series A: Error vs. Iterations

ber of iterations increases - and at approximately the same rate for each value of *sigma*. As such, it can be concluded with a fairly high level of confidence that $n = 50000$ was the optimal value for the number of iterations parameter - at least for this set of experiments.

Aside from helping determining the optimal number of iterations, Series A experiments 33-61 also provided informative results regarding the optimal value of *sigma* for our model. By comparing the results at each level of n, we can see from figures 4, 5, and 6 that the best values for *sigma* ranged from 10 to 22.5, depending on which subset of experiments we look at. Interestingly, although the different values of *sigma* finished in different ranking for different n's, there does not appear to be a definitive correlation between the value of n and which value of *sigma* performed best. This is informative in that it indicates that the optimal value of *sigma* for this model should produce the optimal sampling results regardless of how many iterations we run the experiment for - or at least we can make that generalization for those values of n that fall within the range that we tested (which was certainly not exhaustive). This result was somewhat surprising in that I thought certain *sigma* values might perform better in *short* extperiment runs,

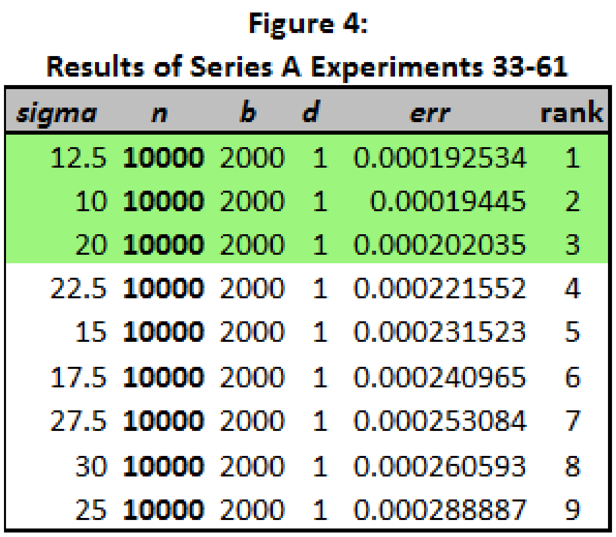


Figure 4:
Results of Series A Experiments 33-61

sigma	n	b	d	err	rank
12.5	**10000**	2000	1	0.000192534	1
10	**10000**	2000	1	0.00019445	2
20	**10000**	2000	1	0.000202035	3
22.5	**10000**	2000	1	0.000221552	4
15	**10000**	2000	1	0.000231523	5
17.5	**10000**	2000	1	0.000240965	6
27.5	**10000**	2000	1	0.000253084	7
30	**10000**	2000	1	0.000260593	8
25	**10000**	2000	1	0.000288887	9

Figure 4: Results of Series A Exp 33-61, $n = 10000$

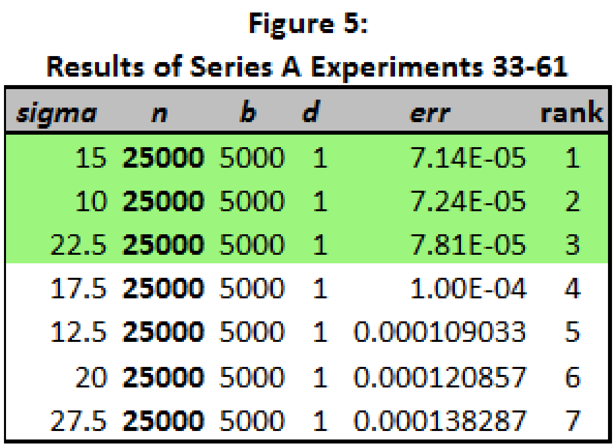


Figure 5:
Results of Series A Experiments 33-61

sigma	n	b	d	err	rank
15	**25000**	5000	1	7.14E-05	1
10	**25000**	5000	1	7.24E-05	2
22.5	**25000**	5000	1	7.81E-05	3
17.5	**25000**	5000	1	1.00E-04	4
12.5	**25000**	5000	1	0.000109033	5
20	**25000**	5000	1	0.000120857	6
27.5	**25000**	5000	1	0.000138287	7

Figure 5: Results of Series A Exp 33-61, $n = 25000$

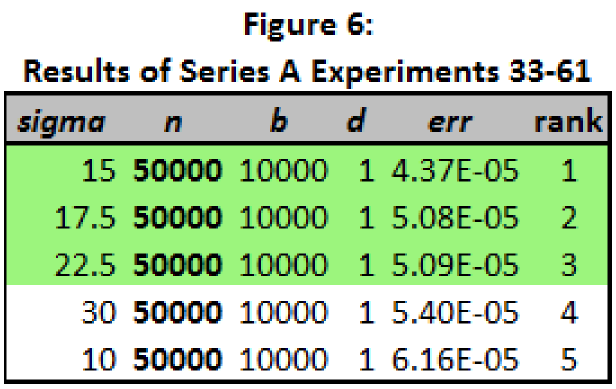


Figure 6:
Results of Series A Experiments 33-61

sigma	n	b	d	err	rank
15	**50000**	10000	1	4.37E-05	1
17.5	**50000**	10000	1	5.08E-05	2
22.5	**50000**	10000	1	5.09E-05	3
30	**50000**	10000	1	5.40E-05	4
10	**50000**	10000	1	6.16E-05	5

Figure 6: Results of Series A Exp 33-61, $n = 50000$

while other values performed better in *longer* experimental runs. Figure 7 shows the average of the top results for Series A experiments 33-61.

Discard Rate, d: In the next subset of experiments, Series A Experiments 62-75, I focused on fine-tuning the discard rate parameter of the model. To achieve this, I took the top three performing $sigma$ values from experiments 56-61 (in which optimal $n = 50000$ was used), namely $sigma = 15, 17.5, 22.5$ and ran four additional experiments for all three $sigma$ values. In all 12 experiments $n = 50000$ was held constant, as was the initial burn quantity, b. However, for each of the three values of $sigma$ I ran an experiment using a different discard rate. The motivation for this was the thought that by discarding most of the generated samples and only keeping every $m'th$ sample from the final set, for instance, (where m is based on the $discardrate$), we could reduce the autocorrelation that can be inherently present in successive samples generated by the Metropolis-Hastings algorithm - due to the fact that it is an MCMC method and thus inherits the Markov property that $P(x^{(t)})$ is based strictly on $P(x^{(t-1)})$ (Reference Note: Source 2). Choosing an optimal initial burn quantity, b, should ideally account for the autocorellation factor - but the thought was that this would be an interesting addition to the DOE to see if we could improve the accuracy of Metropolis-Hastings algorithm, by further reducing the autocorrelation through use of an optimal discard rate.

While this was a great thought, the results were completely contrary to the theory. In reality, the sampling accuracy decreased very precisely as the discard rate increased. In this set of expriments, I ran each of the three optimal $sigma$'s at optimal n number of iterations (50000), with the following discard rates: 0% (i.e. keep all samples), 80% (i.e. keep every 5th sample), 95%, and 98%. Figure 8 com-

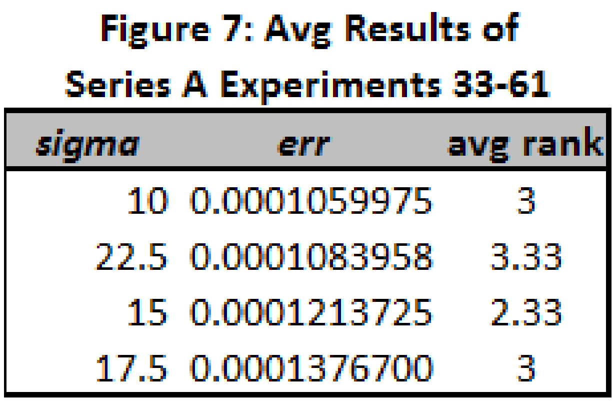


Figure 7: Avg Results of Series A Experiments 33-61

sigma	err	avg rank
10	0.0001059975	3
22.5	0.0001083958	3.33
15	0.0001213725	2.33
17.5	0.0001376700	3

Figure 7: Avg Results of Series A Exp 33-61

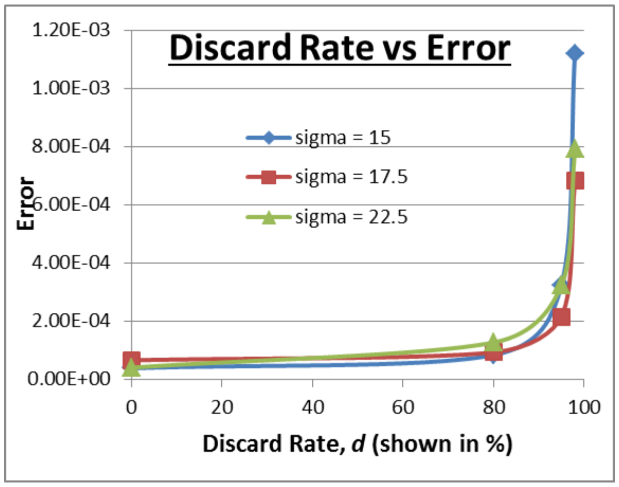


Figure 8: Series A Exp 62-75, Discard Rate vs. Error

pares the Discard Rate vs. the Error for each value of *sigma*. As the figure suggests, error increases sharply along with the discard rate. In hindsight, one possible explanation for this is that the effective (i.e. total resulting) number of samples collected decresed at d increased, since the number of iterations was held constant at the "optimal" level. If I were to run that portion of the experiment again, I would have n increase proportionally with d, thus holding the *effective* number of total samples constant. However, with $d = 80+\%$ for three of the four trials, n would have to be prohibitively large to hold the total number of effective samples constant, given the exisiting limits on time and computational resources. **Initial Burn Quantity, b:** The last of the four model parameters that I tuned in the Series A experiments was the *initial burn quantity*. To achieve this I utilzed a very similar methodology to that which I used to tune the discard rate. However, the results were very inconclusive, as can be seen in the scatter plot of the data shown in figure 9. **Overall Results:** Overall, I was very pleased with the results of the Series A experiments. My final 6 experiments ran comparisons of the top three candidate *sigma* values established thus far by the preceeding experiments, using the optimally tuned parameters of $n = 50000$, $d = 0\%$, and $b = 5000$ (not necessarily optimal, since the results of tuning b were inconclusive). I ran each of the three *sigma* values twice. The results showed that the optimal values of *sigma* were $sigma = 15, 17.5, 22.5$ in that order, corresponding to error rates of $0.005897\%, 0.007078\%, and 0.007238\%$ respectively. Figure 10 shows one of the experimental run results for $sigma = 15$, plotting a histogram of the samples produced and the curve of the the target pdf. As you can see, the histogram matches the pdf very closely. The best overall error rate I was able to for the 1-dimensional case across all 96 experiments for $sigma = 15$, as well, attaining an error of 0.0039%. The second best was for $sigma = 22$, obtaining an error of 0.00412%. Figure 11 shows the plot for $sigma = 15$ and figure 12 shows the plot for $sigma = 22.5$. Figures 13 and 14 show plots for two of the worst occuring results - figure 13 when the standard deviation for the proposal distribution was too small ($sigma = 2.5$), and figure 14 the standard deviation was too large ($sigma = 70$). As you can see in figure 13, the small *sigma* leads to a very high acceptance rate (91.15%), thus the samples are all spread out and the peaks of the target distribution are not properly filled in the histogram, causing the convergence time to be much longer. In figure 14, the large *sigma* causes the acceptance rate to be very low (28.6%), thus causing the samples to be clustered very closely together and the histogram bars to exceed the two peaks in the target Gaussian mixture

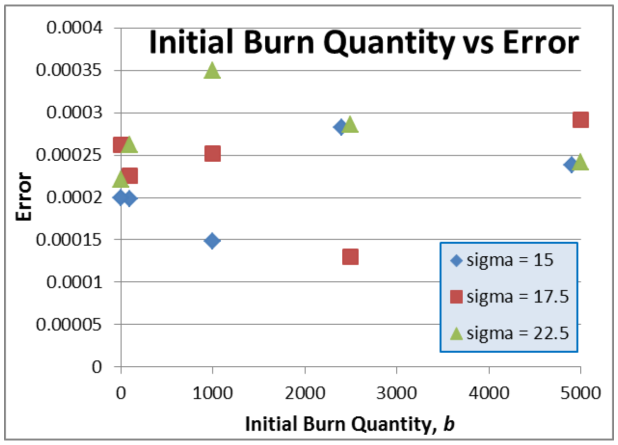


Figure 9: Series A Exp 76-90, Initial Burn Quantity vs. Error (very inconclusive results)

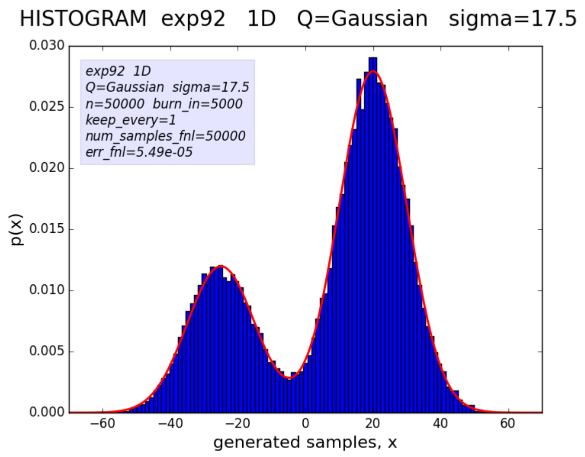


Figure 10: Series A Exp 92: Target pdf and histogram of samples for optimal standard deviation, $sigma = 15$

distribution.

SERIES B: 2-DIMENSIONAL Extending the Metropolis-Hastings algorithm to the 2D case was pretty straight forward from a code/design standpoint, but more difficult to experiment on due to the signifcantly increased time complexity associated with running the actual algorith on higher-dimensional data. Not only did each iteration of the algorithm take slightly longer to perform (although not dramatically, given that the data was only in 2D, as oppsed to n-Dimensional), but I found that many more iterations were required in order to cover the state space. Adding to this difficulty is the fact that the $sigma$ parameter of the model in 2D is the$2x2$ covariance matrix, which has four entries - hence four parts of the parameter to tune. As such, I was unable to conduct a thorough, in depth DOE for the 2D case. However, I performed experiments with a variety of different proposal covariance matrices, achieving mixed results.

For the 2D case, I chose my target distribution to be a mixture of two 2D Gaussian distributions with means: $mu_1 = [4.0, 2.0]$ and $mu_2 = [-7.0, 5.0]$; and with covariance matrices: cov_1=[[8.0, 15.0], [5.0, 25.0]] and cov_2=[[12.0, 5.0], [0.0, 3.0]]. I spent a fair amount of time manipulating these parameters and then plotting the pdfs of the various 2D Gaussian mixtures in a 3D plot in Matplotlib in Python. This was very valuable in and of itself in that it was a great visual demonstration of how altering the mean and covariance parameters of a mixture of Gaussians effect the 2D distribution when projected into the 3rd dimension via its pdf. Ultimately I ended up settling on the parameters that I chose because it created a distribution that I felt would be somewhat challenging to sample from, given its varying peaks and unaligned orientations along its eliptical skews. Although I did not attain as good of results as in the 1D case, I still able to obtain decent results - and whatever inadequacies might have remained were likely due more to a lack of fine-tuning the parameters to the same extent as was done for the 1D case - not due to inadquecies in the algroithm (or my implementation of it) itself. It is also worth noting that I used a normalization constant in the calculation of the SSE for the 2D case to account for the dramatically larger number of bins in the histogram...22500 for the 2D case vs. 100 bins for the 1D case. However, by dividing each individual error term within the SSE by the nomralization constant, the 2D SSE can be compared direclty with the 1D SSE.

In total I ran approximately 20 experiments for the 2D case, attaining a maximum accurracy when the $sigma$ parameter of my proposal distribution (which was a 2D Gaussian distribution with mean mu always

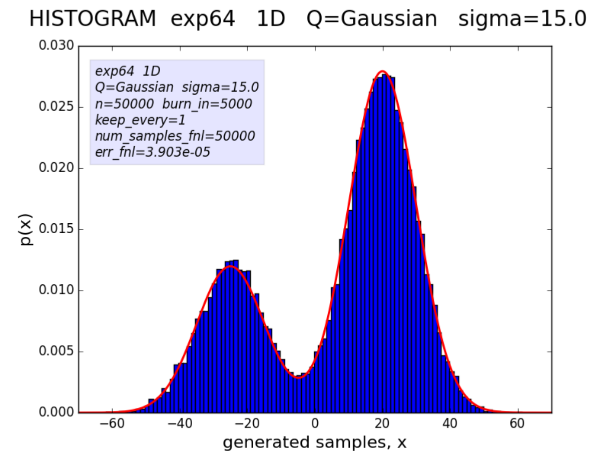


Figure 11: Series A Exp 64: Target pdf and histogram of samples for optimal standard deviation, $sigma = 15$

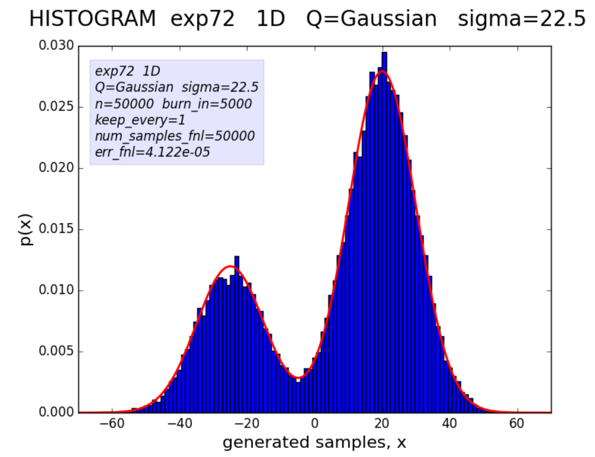


Figure 12: Series A Exp 72: Target pdf and histogram of samples for near-optimal standard deviation, $sigma = 22.5$

centered at the current time step) was a covariance matrix of: $[[120, 0], [0, 120]]$, resulting in an error of 0.003065%. Figure 15 and 16 show two different angles of a 2D histogram of my best 2D experiment, projected onto a 3D plot next to the 3D projection of my target distribution (the target 2D distribution is the blue surface plot, while the 2D histogram is comprised of the many bars). Figure 17 shows the same 2D histogram as a heat map on a flat 2D plot. Figures 18 and 19 show the same but for my second best 2D experiment, with a covariance matrix of $[[70, 0], [0, 85]]$ and an error of 0.007156%. Similarly, figures 20 and 21 show the same, but for one of the worse experimental results for the 2D case, in which the elements of the covariance matrix were far too small.

4 References

1. class lecutures, notes, handouts, etc.
2. http://en.wikipedia.org/wiki/Metropolis%E2%80%93Hastings_algorithm
3. CODE REFERENCE NOTE: the second half of one of my functions for plotting in 3D was adapted from an online source: http://matplotlib.org/examples/mplot3d/hist3d_demo.html
4. CODE REFERENCE NOTE: this technique for setting the colormap was adapted from matplotlib's online demo repository: http://matplotlib.org/examples/...

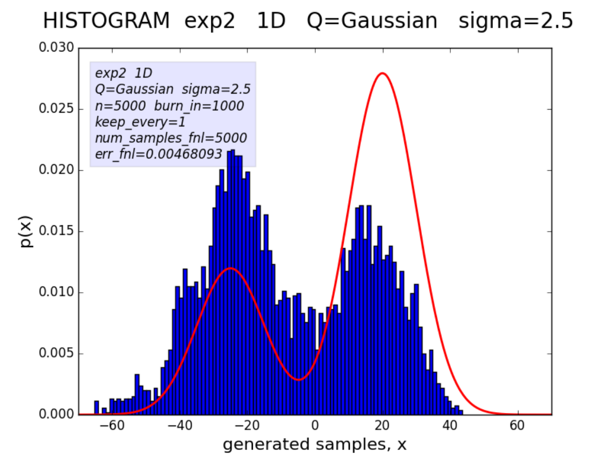


Figure 13: Series A Exp 2: Target pdf and histogram of samples for too small of a standard deviation, $sigma = 2.5$

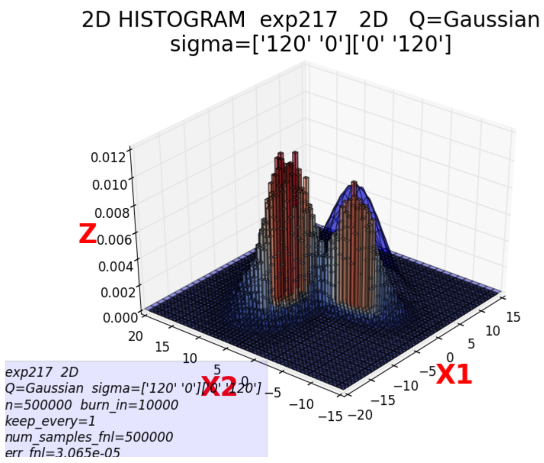


Figure 15: Series B Exp 217: Target 3D pdf and histogram of samples for optimal covariance

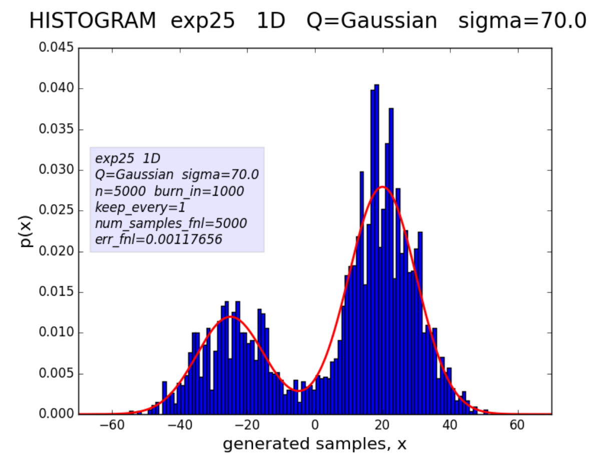


Figure 14: Series A Exp 25: Target pdf and histogram of samples for too large of al standard deviation, $sigma = 70$

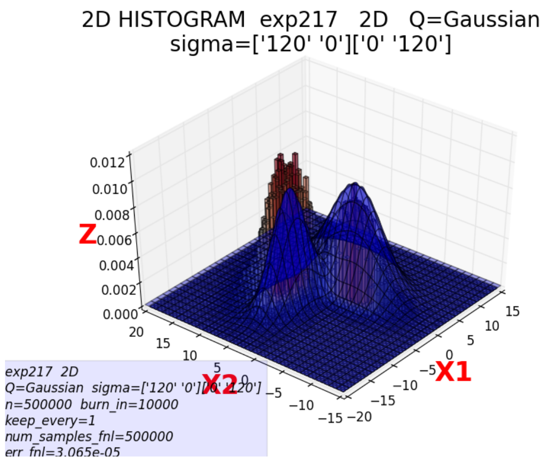
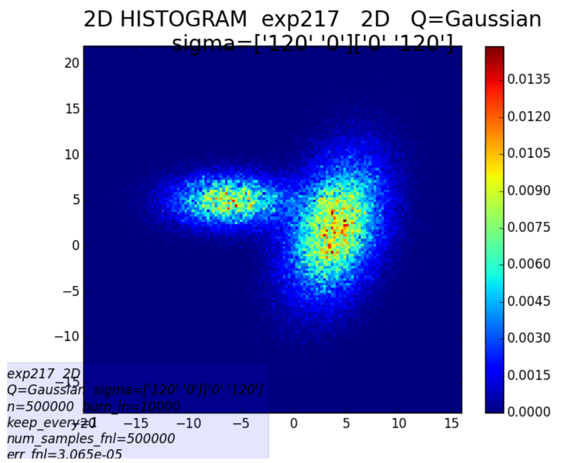
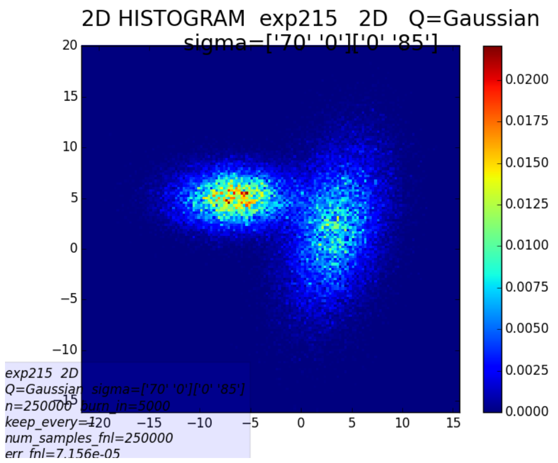


Figure 16: Series B Exp 217: Target 3D pdf and histogram of samples for optimal covariance



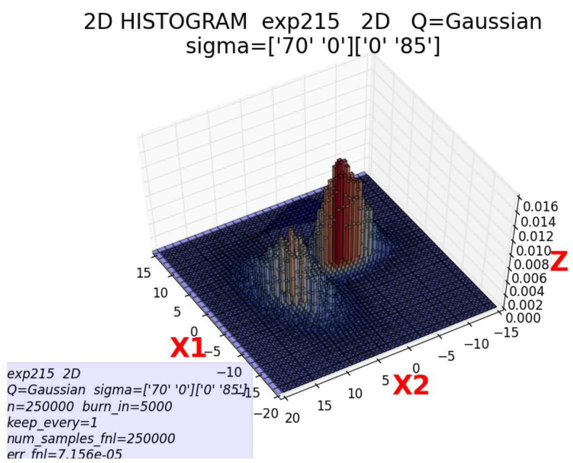
exp217 2D
Q=Gaussian sigma=['120' '0']['0' '120']
n=500000 burn_in=10000
keep_every=1
num_samples_fnl=500000
err_fnl=3.065e-05

Figure 17: Series B Exp 217: 2D heatmap histogram of samples for optimal covariance



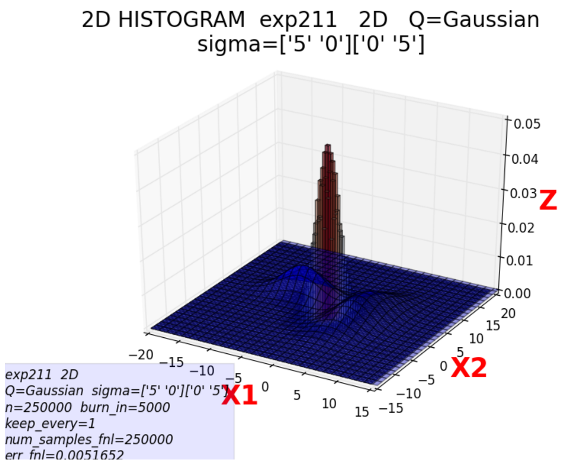
exp215 2D
Q=Gaussian sigma=['70' '0']['0' '85']
n=250000 burn_in=5000
keep_every=1
num_samples_fnl=250000
err_fnl=7.156e-05

Figure 19: Series B Exp 215: 2D heatmap histogram of samples for optimal covariance



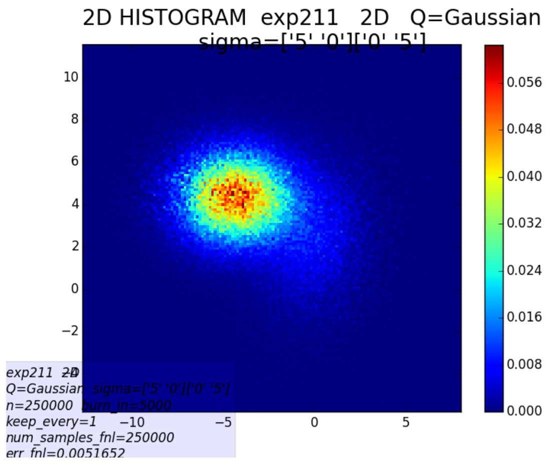
exp215 2D
Q=Gaussian sigma=['70' '0']['0' '85']
n=250000 burn_in=5000
keep_every=1
num_samples_fnl=250000
err_fnl=7.156e-05

Figure 18: Series B Exp 215: Target 3D pdf and histogram of samples for optimal covariance



exp211 2D
Q=Gaussian sigma=['5' '0']['0' '5']
n=250000 burn_in=5000
keep_every=1
num_samples_fnl=250000
err_fnl=0.0051652

Figure 20: Series B Exp 211: Target 3D pdf and histogram of samples for too small covariance



exp211 2D
Q=Gaussian sigma=['5' '0']['0' '5']
n=250000 burn_in=5000
keep_every=1
num_samples_fnl=250000
err_fnl=0.0051652

Figure 21: Series B Exp 211: 2D heatmap histogram of samples for too small covariance