CS136 Final Project

Ethan Nanavati

May 9th 2024

1 Introduction

Often, we want to calculate the posterior distribution given a prior and a likelihood. This is the core of Bayesian reasoning and is used in places from next-word prediction using a Dirichlet-Categorical model to performing posterior predictive estimation for model parameters. However, this task can be difficult– especially if the prior used is not conjugate– as it may require the integration of a complicated distribution.

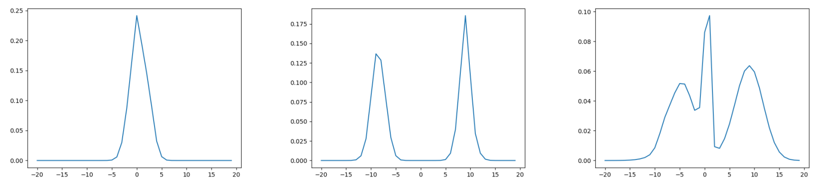
In this project, I use Chebyshev interpolation to rescale and represent a sampleable 'distribution' that does not integrate to 1 as a polynomial expression on an interval. In doing so, the method will approximate the underlying properly scaled distribution. This addresses the same problem in computing the posterior distribution from a sampleable likelihood and prior. I will analyze the accuracy of this Chebyshev method and compare it against the histogram generated by Metropolis sampling.

2 The Distribution (Data)

In selecting the target distributions for sampling, there were several considerations.

An important property of the distributions (these can be thought of as 'posteriors') that I use is that they are easily visualizable and evaluable. This allows a direct comparison of my model against the true distribution that it attempts to approximate. To that end, I created synthetic distributions for which closed-form solutions for the density are known.

Further, the distributions should be of differing difficulty and characteristics. Specifically, I wanted to choose a distribution that could emphasize pathologies in preexisting methods (in particular, Metropolis-Hastings sampling) to demonstrate a case that I believe this method will outperform. To that end, I included a "Gaussian Gaps"(GG) distribution. This is a 1D Gaussian Mixture Model (GMM) which has means that are far apart from each other. I also included 2 other distributions which should allow for the Chebyshev model to compare with the baseline in a more competitive setting. These distributions are "Gaussian Concentrated"(GC) (GMM where the means are very close) and "Gaussian



(a) GC (b) GG (c) GE

Figure 1: The 3 distributions that I will be evaluating the models on. GC means Gaussian Concentrated, GG means Gaussian Gaps, GE means Gaussian Eclectic.

Eclectic"(GE) (GMM where the means are chosen uniformly at random and variance is allowed to be higher).

All means for these distributions are selected in [-10,10] and it is assumed that all of their mass is in [-20,20]. This is reasonable since the standard deviation for each Gaussian was less than 3 and by simple calculation with the normal CDF, more than 99.9% of the mass falls within [-20,20]. More importantly, numerical integration revealed that less than .001% of the mass fell outside of [-20,20] in any of these distributions. These bounds are important for later in considering which points to interpolate at for Chebyshev Interpolation.

The goal of this project is to compare the approximations from the Chebyshev and baseline models to each of these distributions. One might expect a specialized method for computing the distance between distributions such as Wasserstein distance or KL-divergence. However, due to the nature of interpolation and sampling, we can expect local approximations to be 'vertical' rather than exhibiting horizontal shifts from the target distribution. Since I do not expect such pathologies to play a role in this project, I decided to use a simpler method where I compute the l1 absolute integration error between distributions and approximations with SciPy numerical integration.

3 Baseline Method

As a baseline, I will use Metropolis sampling to construct a histogram. The Metropolis algorithm will sample 5000 points after 5000 burnin samples from the posterior via a MCMC random walk, then a histogram over the samples will construct an approximation to the target distribution. I chose to use 5000 samples and 5000 burnins to give the model many points to approximate the target but not be too computationally intensive.

$p^*(x)$ is the target distribution which can be written as $p^*(x) = \sum_{k=1}^{N} \pi_k * Normal(x|\mu_k, \sigma_k^2)$ where π is a vector drawn from the N-dimensional probability simplex and μ_k and σ_k are some valid parameters to a normal distribution. $\tilde{p}(x)$ is the scaled distribution that I can sample from $\tilde{p}(x) = kp^*(x)$ (k a constant). $z^1, z^2, \ldots, z^s$ will be the samples in the Metropolis Markov chain, the first of which is randomly generated. z' is the proposed value at each step of the Metropolis algorithm. It will be drawn from $Q(\mu, \sigma)$ where Q is a normal distribution with mean μ as the previous entry in the Markov chain and some standard deviation σ (the proposal width). The number of bins in the histogram (K) is chosen via Sturge's rule ie. K = 1 + 3. 322 logN (N the number of kept samples). While other approaches may also have merit, I found that the method worked well with this one. I will use the standard Metropolis acceptance criteria, namely, accept z' over z_t (the previous entry in the Markov chain) if for u drawn from Uniform[0,1] $u \leq \frac{\tilde{p}(z')}{\tilde{p}(z_t)}$. Note that σ is considered known and serves as a hyperparameter to this model.

After drawing samples with Metropolis, I will bin a histogram across a restriction of the domain that includes all of the samples that the algorithm saw. This histogram will accurately approximate the distribution, given enough data, since Metropolis samples from the posterior. However, it may take a long time for convergence to occur. I am especially skeptical about the performance on the Gaussian Gaps dataset as for insufficiently large proposal width σ, Metropolis may not sample the whole distribution in a reasonable number of steps.

Fortunately, since I know the true distribution, it is straightforward to tell from the histogram if the Metropolis sampler converged. If the histogram well approximates the data, it converged, whereas if there are significant qualitative differences, convergence has not yet occurred. I also intuited this by calculating the absolute integration error (l1 error) between the histogram and target distribution. Low error was strong evidence of convergence, whereas high error indicated a lack of convergence. Because of these techniques, I did not need to use multiple chains to verify whether Metropolis had converged to the target.

I chose 3 different proposal widths σ =[1,5,10] for each dataset and performed analysis on the ones with lowest l1 error. I chose these hyperparameter values because I expected the model would behave differently on each. I thought that on the Gaussian Concentrated and Gaussian Eclectic datasets a moderate proposal width would be ideal as much of the data is connected (no large gaps with low density). On the other hand, including a proposal width of 10 (which is $\frac{1}{4}$ the total range [-20,20]) might be ideal for the Gaussian Gaps dataset given the large separation between peaks.

I implemented everything in Python. The Metropolis Histogram model was built from scratch using NumPy to generate pseudo-random numbers.

4 Proposed Upgrade

I will use Chebyshev Interpolation to construct the target distributions p^* Gaussian Eclectic, Concentrated, and Gaps.

After sampling the Chebyshev points from sample distribution $\tilde{p}(x) = kp^*(x)$ (k=.15 was used as an arbitrary scaling constant to mimic the process of rescaling a likelihood * prior to get the posterior) on [-20,20] and constructing the interpolating polynomial on those points, I analytically integrated the interpolant and scaled it so that it integrated to 1. The integration problem was then substantially easier as we needed only to integrate a polynomial on an interval.

Chebyshev interpolation works by sampling points $(x_i, p^*(x_i)_{i=1}^n$ from a target function $\tilde{p}(x)$ on an interval (Note that n is the number of points or nodes used in interpolation). It seeks to construct a polynomial of degree at most n-1 that goes through every $(x_i, p^*(x_i)$. A standard way to do this is using the Lagrange polynomials $l_i(x) = \prod_{j \neq i, j=1}^n \frac{x - x_j}{x_i - x_j}$ which convert the problem of finding the n node interpolant $(P_n(x))$ into $P_n(x) = \sum_{i=1}^n p^*(x_i) l_i(x)$. What makes Chebyshev interpolation interesting, though, is the sampling points that it uses. By choosing the x_i to be the Chebyshev points on [-20,20], $x_i = 20cos(\frac{(2i+1)\pi}{2n})$ we can ensure that the error at a point is bounded using

$$|p^*(x) - P_n(x)| \leq \frac{2 * 10^{n+1} * \max_{x \in [-20,20]}(f^{n+1}(x))}{(n+1)!} \tag{1}$$

(Note that $f^{n+1}(x)$ represents the n+1th derivative of f.)

Importantly, Chebyshev interpolation minimizes the uniform error on an interval for any nth degree interpolation scheme and is thus, in a sense, optimal (A. Tasissa, Chebyshev Interpolation, February 2024).

Chebyshev interpolation is interesting because of these analytical guarantees. Given assumptions on the smoothness of a distribution on a bounded domain (that $\max_{x \in [-20,20]}(f^{n+1}(x))$ is not too large), uniform error can be bounded above and decays rapidly with an increasing number of samples. This means that we can get an arbitrarily accurate estimate of the posterior distribution and enable a simple analysis of the target distribution through the resulting polynomial. Moreover, provided that $(f^{n+1}(x))$ does not increase too fast with n (faster than n!), convergence is super-exponential (due to the n+1!) so it should take little time and number of nodes to achieve high accuracy. This is in contrast to Metropolis which may not converge in a reasonable amount of time if the proposal width is poorly chosen. Additionally, Chebyshev does not require running multiple times, nor does it require any hyperparameter tuning beyond choosing the number of nodes so it may prove faster than Metropolis sampling.

I perform Chebyshev Interpolation for 8, 16, 32, and 100 nodes in order to get different accuracy interpolants in a computationally feasible way. For each of these I evaluate performance using l1 error due to its ease and to maintain the same standard across both models.

I hypothesized that when compared to a Metropolis histogram, Chebyshev interpolation should decrease l1 error in less time, especially if the proposal width for Metropolis is chosen poorly on the Gaussian Gaps distribution. This is because there may be insufficient exploration in Metropolis, but Chebyshev Interpolation has analytical guarantees on performance regardless of target function.

The Chebyshev Interpolation model heavily leveraged the symbolic math and integration of the SymPy package. In particular, it was helpful for expanding, integrating, and evaluating the Chebyshev interpolants. Throughout the project, I also used NumPy to generate pseudo-random numbers for reproducibility and SciPy to perform numerical integration for l1 error.

5 Results and Analysis

I ran Metropolis for 5000 burnin and keep samples on the proposal widths described previously (1,5,10) and presented findings on the histograms corresponding to the proposal widths that minimize l1 error for each of the 3 distributions. I then performed Chebyshev interpolation with normalization on each of the sample distributions for 8,16,32, and 100 nodes. I presented findings on the interpolants that minimized l1 error for the 3 distributions (see Figure 2).

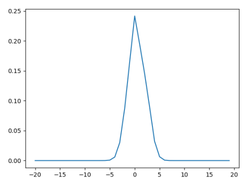
For reference, in my numerical experiments it seems that if l1 error is below .4, the approximation resembles the distribution with a good degree of accuracy. However, if the error exceeds .7 there is very little connection between the target distribution and approximation.

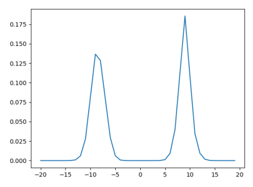
Performance for both of these methods differed dramatically given different proposal widths (for Metropolis) and numbers of nodes (for Chebyshev). As expected, Metropolis performed well for all proposal widths on the Gaussian Eclectic and Gaussian Concentrated distributions, but required a sufficiently large width (around 5) to converge to the Gaussian Gaps distribution. However, given an appropriate proposal width, the Metropolis method performed surprisingly well, having an l1 error of less than .15 on Gaussian Gaps. For 32 nodes, Chebyshev interpolation also proved very nice. The approximation of GC and GG had l1 error of less than .2 and GE had a higher, but still reasonable, error of .219. For these runs, performance was comparable and the time to run was actually shorter for Metropolis than for interpolation (contrary to my hypothesis).

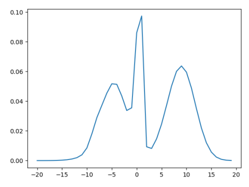
From these results, one might conclude that the methods perform comparably at approximating target distributions. However, while I expected convergence of Chebyshev interpolation to be fast and increase with the number of nodes, surprisingly, when I tried interpolating at 100 nodes, there was a catastrophic error of nearly 1. I tried a number of other node values and got a similarly poor result where the density at the endpoints was incredibly high and nearly 0 everywhere else. This is illustrative of the Runge phenomena–a characteristic of interpolants wherein the error at the boundary is large for high-degree polynomials. While Chebyshev interpolation is designed to minimize this, it is not effective in all cases. Recall the Chebyshev interpolation error.

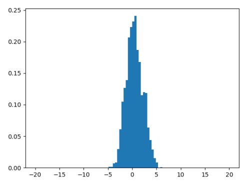
$$|p^*(x) - P_n(x)| \leq \frac{2 * 10^{n+1} * \max_{x \in [-20,20]}(f^{n+1}(x))}{(n+1)!} \tag{2}$$

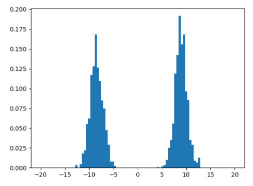
For a function whose derivatives grow sub-exponentially, this error will converge at all points in the interval. However, for a Gaussian (and by extension a GMM) that is not the case. As you take higher derivatives, the term

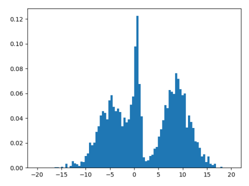

(a) True GC

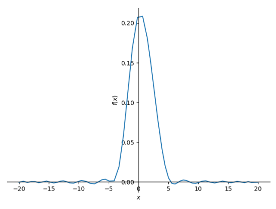

(b) True GG

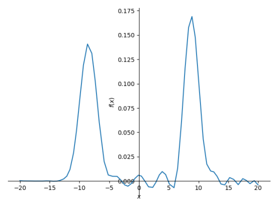

(c) True GE


(d) Metropolis GC


(e) Metropolis GG


(f) Metropolis GE


(g) Chebyshev GC


(h) Chebyshev GG

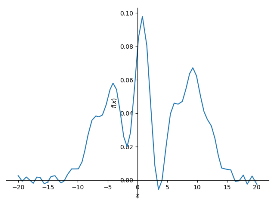

(i) Chebyshev GE

Figure 2: GC means Gaussian Concentrated, GG means Gaussian Gaps, GE means Gaussian Eclectic. The first row shows the true target distribution, the second shows the Metropolis histograms, and the third shows rescaled Chebyshev interpolants. (d) proposal width is 5. l1 error is .100. (e) proposal widthis 5. l1 error is .148 (f) proposal width is 5. l1 error is .124. (g) 32 nodes. l1 error is .107. (h) 32 nodes. l1 error is .153. (i) 32 nodes. l1 error is .219.

$\max_{x\in[-20,20]}(f^{n+1}(x))$ explodes on finite intervals for a Gaussian. Indeed, while the negative exponential in the Gaussian pdf eventually decreases growth to 0, $f^{n+1}(x)$ takes the form $\Theta(x^n) * e^{-\Theta(x^2)}$ where the polynomial term of degree x^n can overpower the exponential on a finite interval about 0. I believe that this was the issue that occurred during interpolation and that for more than about 40 nodes, Chebyshev Interpolation cannot approximate Gaussian mixtures with any accuracy.

Another consideration from these results is that the 32 point interpolants dropped below 0 on the [-20,20] interval. This is somewhat expected, considering that interpolation simply seeks to fit a polynomial (which can be negative) to data. However, this is problematic for approximating a distribution as density should be consistently non-negative.

Surprisingly, Metropolis sampling also took less time to run than Chebyshev interpolation. This could be for any number of reasons, from the few samples that I drew in Metropolis to a slow Python implementation of Chebyshev interpolation. It is possible that there are well behaved distributions that Chebyshev interpolation will approximate accurately and in shorter time than it takes to run Metropolis to convergence, but the results of this project do not give evidence of that.

6 Conclusion

Chebyshev interpolation and Metropolis Histograms provided similar levels of accuracy for less accurate methods (<32 nodes and $<$ 5000 samples respectively), but, on the datasets I used in this project, the latter scaled much better with number of samples than Chebyshev for increasing nodes. This was in opposition to my expectation that Chebyshev interpolation would reliably outperform Metropolis for any reasonably high number of nodes.

I hypothesized that the reason for this discrepancy was that the repeated derivatives of a Gaussian mixture grow too fast and affect the uniform guarantees on the interpolant's approximation.

If this is true, the results are likely heavily influenced by the choice of distributions in this project. Future work could seek to evaluate the performance of Chebyshev on distributions whose derivatives grow slower (for example, exponential distributions with $\lambda < 1$). One might also explore methods that ensure that the interpolant is non-negative on the region of integration (for example, setting the interpolant to 0 in regions where it was negative and then rescaling).

Nonetheless, given the ubiquity of Gaussians, the results of this project suggest that the Chebyshev interpolation method has substantially less practical utility than expected– Illustrating Runge phenomenon qualitatively and through l1 error. Moreover, these results lend more evidence to the Metropolis model being an effective way to construct a rescaled distribution.

7 Bibliography

References

[1] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020.

[2] Aaron Meurer, Christopher P. Smith, Mateusz Paprocki, Ondřej Čertík, Sergey B. Kirpichev, Matthew Rocklin, AMiT Kumar, Sergiu Ivanov, Jason K. Moore, Sartaj Singh, Thilina Rathnayake, Sean Vig, Brian E. Granger, Richard P. Muller, Francesco Bonazzi, Harsh Gupta, Shivam Vats, Fredrik Johansson, Fabian Pedregosa, Matthew J. Curry, Andy R. Terrel, Štěpán Roučka, Ashutosh Saboo, Isuru Fernando, Sumith Kulal, Robert Cimrman, and Anthony Scopatz. Sympy: symbolic computing in python. *PeerJ Computer Science*, 3:e103, January 2017.

[3] Abiy Tasissa. Lecture 10: Chebyshev interpolation, 2024. Unpublished lecture notes.

[4] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020.

[5] WSU. Chebyshev interpolation.

[1] [2] [4] [5] [3]