ECAI 2023
K. Gal et al. (Eds.)

doi:10.3233/FAIA230483

An Easy Rejection Sampling Baseline via Gradient Refined Proposals

Edward Raff[a,b,c,d], **Mark McLean**[a] **and James Holt**[a]

[a]Laboratory for Physical Sciences
[b]Booz Allen Hamilton
[c]Syracuse University
[d]University of Maryland, Baltimore County

Abstract. Rejection sampling is a common tool for low dimensional problems ($d \leq 2$), often touted as an "easy" way to obtain valid samples from a distribution $f(\cdot)$ of interest. In practice it is non-trivial to apply, often requiring considerable mathematical effort to devise a good proposal distribution $g(\cdot)$ and select a supremum C. More advanced samplers require additional mathematical derivations, limitations on $f(\cdot)$, or even cross-validation, making them difficult to apply. We devise a new approximate baseline approach to rejection sampling that works with less information, requiring only a differentiable $f(\cdot)$ be specified, making it easier to use. We propose a new approach to rejection sampling by refining a parameterized proposal distribution with a loss derived from the acceptance threshold. In this manner we obtain comparable or better acceptance rates on current benchmarks by up to 7.3×, while requiring no extra assumptions or any derivations to use: only a differentiable $f(\cdot)$ is required. While approximate, the results are correct with high probability, and in all tests pass a distributional check. This makes our approach easy to use, reproduce, and efficacious.

1 Introduction

Given a target distribution $f(\cdot)$ that we wish to draw samples from, rejection sampling provides one of the most popular strategies. The approach is often advertised as "easy," given the simplicity of its procedure. Given a proposal distribution $g(\cdot)$ that we do know how to sample from, and a value C such that $C \cdot g(x) \geq f(x) \forall x$, rejection sampling can be described succinctly in three steps:

1. Generate $x \sim g(\cdot)$ and $u \sim \mathcal{U}(0, 1)$, where $\mathcal{U}$ is the continuous uniform distribution.
2. Accept x as a valid sample of $f(\cdot)$ if $u \leq \frac{f(x)}{Cg(x)}$.
3. If not accepted, go back to step 1.

This also makes $1/C$ directly interpretable as the acceptance rate of the sampling procedure, and it guarantees independent and identically distributed (I.I.D.) samples. For this reason rejection sampling is often used in modeling applications where a physical processes is known, but produces bespoke distributions of few variables that need to be sampled [19, 26, 17, 23, 25, 27, 18]. Though algorithmically simple, this belays the non-trivial amount of work required by a user who wishes to draw samples. First one must perform considerable work to devise a distribution $g(\cdot)$ to sample from, find a method to determine C or find a small upper bound of C.

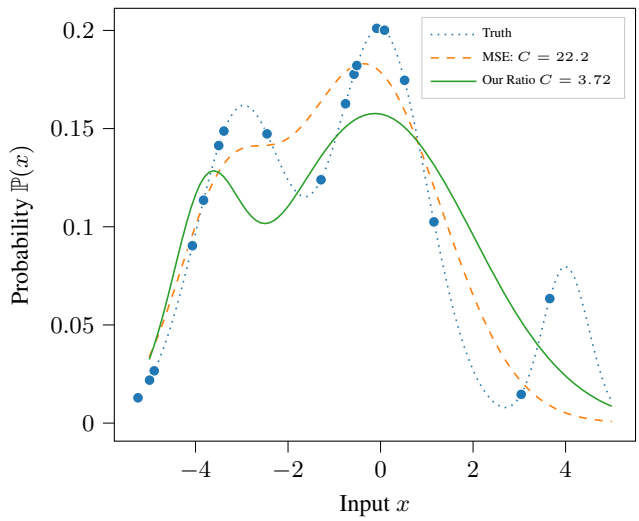

Figure 1: Example of estimating the true distribution $f(\cdot)$ (dotted blue) using a surrogate $g(\cdot)$ fit from a set of current samples. The MSE approach used by prior works is shown in orange (dashed). While MSE *looks* like a good fit, it does not directly relate to the ratio $f(x)/g(x)$ that determines the acceptance rate of rejection sampling. Our approach (solid green) has a 6× higher acceptance rate over the MSE based fit used by prior works.

Because of this difficulty, there has been considerable work over time attempting to perform rejection sampling with greater efficacy. These newer sampling strategies often suffer from additional constraints on the target distribution $f(\cdot)$, or additional work on the user to do additional derivations. In both cases the application of a rejection sampler is non-trivial for a user, and slows progress toward prototyping, building simulations, and various down-stream tasks like HMC and Gibbs samplers that may desire to leverage sampling for an intermediate step.

In this paper, we describe our novel Easy Rejection Sampler (ERS), where our goal is to make ERS easy for the user of rejection sampling. Rejection sampling is historically only done with differentiable functions, and so we apply modern automatic differentiation to the design of ERS. In doing so we build a system that accurately produces samples without requiring any specification of $g(\cdot)$, C, or any other values by the user or restrictions on $f(\cdot)$. This is obtained by

leveraging work in rejection sampling that allows estimating C while still producing valid samples with high probability. Since we require a proposal distribution $g(\cdot)$ that is easy to sample from, flexible, and parameterized, we use a Gaussian Mixture Model (GMM). We exploit the differentiable approach and parameterized GMM to develop a "refinement" operation that maximizes the acceptance rate by directly optimizing the ratio $f(\cdot)/g(\cdot)$ instead of Mean Squared Error (MSE) based measures used by prior works. As Figure 1 shows, this can significantly reduce the true value of C compared to current approaches. This refinement is run periodically as more samples are collected, allowing further minimization of C.

The rest of our paper is organized as follows. First we review work related to our own in section 2, including the two prior types of general purpose rejection samplers: optimization based and kernel density based. Next we will detail the design of our ERS approach in section 3 in three steps, presenting the primary mechanisms of ERS in the order they are used, with the final mechanism being our novel refinement based approach to proposal distributions that provides the key to higher acceptance rates over a wide array of challenging target distributions. These targets will be described in section 4 along with our experimental setup, followed by the results showing up to $7.3\times$ higher acceptance rates while imposing the fewest restrictions on $f(\cdot)$, and that our method is also faster by runtime estimates. These results also demonstrate the first example of obtaining improved rejection sampling runtime via the use of a GPU.

2 Related Work

Rejection Sampling is still widely used in multiple disciplines, often due to intrinsically low-dimensional problems or the need to perform simulations involving customized and hard to characterize integrals. This includes physics [19, 26], signal processing [21], catastrophe modeling [17, 23], computational finance [25, 27]. Though these works generally use off-the-shelf rejection sampling methods or design bespoke ones, their problems are not all amenable to current state-of-the-art samplers from machine learning, and often have high sample rates once a suitable $g(\cdot)$ and C is determined. Our work will focus on challenge-problems known to produce low sample rates for current methods so that a meaningful effect is detectable.

While our method does not technically guarantee perfect sampling from the target distribution $f(\cdot)$, in all statistical tests our samples match the true distribution in every experiment. Further, approximate samples have been acceptable by users in practice[1], and our work inherits proofs that the samples will be correct with high probability. While Hamiltonian Monte Carlo (HMC) based sampling also uses a gradient and can handle much higher dimensions than rejection sampling, it also is not as 'turn key' and can require non-trivial work to ensure sample quality. Our study is focused only on rejection sampling and its standard uses: differentiable functions with $d \leq 3$ dimensions [19, 26, 17, 23, 25, 27, 18].

There is a broad family of *adaptive* rejection sampling methods first proposed by [12], where the sampling acceptance rate improves for larger total number of samples n. This seminal work has spawned many extensions, but which generally require the specification of $g(\cdot)$ and C (though they may ease the process), and more restrictive limitations on $f(\cdot)$ [13, 8, 11, 15]. Our work is instead focused on more general purpose rejection sampling techniques with weak or limited assumptions on $f(\cdot)$ that require little work to apply to new problems.

[1] e.g., the popular https://paulnorthrop.github.io/revdbayes/ uses an approximate sampler.

There are two different families of general purpose rejection sampling algorithms available today. The first family can be described as an optimization & sampling strategy that was first proposed with the OS* algorithm [18]. Their work introduced the use of rejected samples to improve the proposal $g(\cdot)$. The latter A* introduced a gumbel trick to further improve efficacy [20]. However these methods require additional derivations to be done by the user, requiring a function $i(\cdot)$ and $o(\cdot)$ such that $f(x) \propto \exp(i(x) + o(x))$. Our ERS also intermixes optimization with sampling, but uses gradient based approaches. In contrast ERS will not require any proposals or bounds to be specified by the user that OS* and A* require.

Pliable Rejection Sampling (PRS) [10] introduced the second strategy of using a Kernel Density Estimator (KDE) to estimate $g(\cdot)$ from the samples, a strategy refined by Nearest Neighbour Adaptive Rejection Sampling algorithm (NNARS) [1] to produce a near-optimal sampler under certain assumptions. However, both methods require bounded support to estimate the KDE and have non-trivial parameters to estimate analytically, or via cross-validation. The later is particularly challenging as it requires pre-existing samples, which significantly complicates practical usage. In contrast we impose no constraints on $f(\cdot)$ and require no other items to be specified by the user, and we use a GMM so that we may modify a reasonable number of parameters via gradient descent. Though PRS and NNARS are most similar to ERS in terms of $g(\cdot)$, they optimize a form of MSE to approximate $f(\cdot)$ where our novel ratio optimization provides considerably high acceptance rates.

We make note that the considerable requirements to use modern rejection samplers create reproducibility risks. The latter cited works often depend on transformations of the sampling distribution in order to convert unbounded support into finite support (i.e, changing the domain $\mathcal{X}$ from $[0, \infty]$ to $[0, 1]$) but do not specify what transformation is used, and do not specify many derivations. NNARS relies on a Holder constant H and associated $s \in [0, 1]$ such that $|f(x) - f(y)| \leq H\|x - y\|_\infty^s, \forall x, y \in \mathcal{X}$, but the H values are no stated for any experiments[2]. Similarly, PRS requires a cross-validation procedure with no further details. This missing information has been identified by many prior works as a significant barrier to replication [9, 14, 24, 28, 30, 31, 32]. For these reasons, we use the results as presented in these prior works, but note an additional benefit of our approach: by requiring no hyper-parameters, transformations or other constraints to be specified, the ability to reproduce works leveraging ERS is increased. Further, we provide source code at https://github.com/NeuromorphicComputationResearchProgram/EasyRejectionSampling.

3 Method

Our primary goal is to design a mechanism by which a user is not required to specify anything other than the function $f(\cdot)$, and optionally a domain $\mathcal{X}$ that they wish to draw samples from $x \sim f(\cdot) \in \mathcal{X}$. This is a more challenging problem than that addressed by prior methods, as the user, by specifying both the surrogate function $g(\cdot)$ and the ratio $C = \sup_{z \in \mathcal{X}} \frac{f(z)}{g(z)}$, is imposing a prior knowledge to the sampling method.

To build our approach, we must define a surrogate function $g(\cdot)$ and somehow determine the maximal ratio C during runtime. We tackle the latter by leveraging the little known method of "Empirical Supremum" based sampling [7]. Caffo et al. showed that by initializing an estimate of the supremum $\hat{C}_t = 1$, and then adjusting it

[2] Code for the method is available, not the experiments, and we were unable to replicate their results in our attempt.

to $\hat{C}_{t+1} = \max\left(\hat{C}_t, \frac{f(x_{t+1})}{g(x_{t+1})}\right)$, that there will be a finite number of errors when the support $\mathcal{X}$ is discrete, with a similar result for unbounded support. However, [7] still required the specification of $g(\cdot)$. We note that the Empirical Supremum was found to converge in under 100 iterations. We leverage this by performing a minimum of 500 samples per iteration, using the maximum across all samples before deciding an accept/reject decision. In practice we find that this results in convergence within 0.001 in one iteration, resulting in no difference between using $\hat{C}$ and the true C for our experiments.

Understanding that we will safely estimate C as the sampling runs allows us to specify the overall strategy of our approach. The description is subdivided into three critical design choices. Together, they will form our "Easy Rejection Sampling" approach.

1. We will use a batched iterated sampling procedure to alter the candidate distribution $g(\cdot)$ via a Gaussian Mixture Model after successive rounds of sampling.
2. We will specify a simple gradient based strategy for selecting an initial candidate distribution $g(\cdot)$ with few evaluations of $f(x)$.
3. We develop an approach to *refining* the distribution $g(\cdot)$ to *maximize acceptance rate*, where prior work has instead focused on maximizing the overall similarity of the distributions.

We note that the third refinement step, where we develop a loss function that targets the ratio $f(\cdot)/g(\cdot)$ directly, is critical to the success of our method. It is placed at the end due to readability, and it is the last step of each iteration. This is necessary as we use both accepted and rejected samples to inform the refinement, so running the refinement regularly can further improve the results by reducing C further.

Careful design allows us to perform significantly better than the theoretically optimal (in only a min-max sense[3]) NNARS algorithm while simultaneously requiring fewer constraints and no extra information from the user. This shows that the constant factors in NNARS are non-trivial, and full understanding of the limits of adaptive rejection sampling is not yet known. Our method further inherits the correctness results from [7], and additional empirical tests show no detectable difference in our sample quality from the true distribution.

Each of the following sub-sections match our stated design choices, and will detail the relevant approach with the justification for why the approach was taken. We note that, as a matter of engineering, the below procedures currently include hard-coded hyper parameters that we do not tune. Their purpose is simple and intuitive (e.g., increase a value by some small amount so we are not comparing to a worst case), and do not require complex math like the Holder constant used by NNARS. In extended testing we found that these coefficients did not have a meaningful impact on the results.

Our implementation is done in JAX [6] and works on $\log(f(\cdot))$ and $\log(g(\cdot))$ in order to be numerically stable. Our description will remain at the $f(\cdot)$ and $g(\cdot)$ level for clarity.

3.1 Easy Rejection Sampling Algorithm

The overall procedure of our approach can be succinctly described as fitting a GMM to the current data as the proposal distribution $g(x)$, sampling, and then re-fitting the proposal $g(x)$ at various intervals. Our use of vectorized operations makes the multiple fittings of $g(x)$ computationally reasonable, and when factoring out differences in acceptance rate is $\geq 2\times$ faster on CPU, and shows the first speedup for a rejection sampler on a GPU at $4\times$.

Our approach is shown in pseudo-code in Algorithm 1, where A and R contain the set of currently accepted and rejected samples, with their evaluations against $f(\cdot)$ cached for posterity. The K current mean, covariance, and weights of the GMM are $\boldsymbol{\mu}_{1,\dots,K}, \Sigma_{1,\dots,K}$, and $\boldsymbol{w}$ respectively; the subscript $1,\dots,K$ will be dropped if the value of K is not changing. Our two primary operations are fitting a GMM (initialized with k-means++ [2, 29]), and a "refine" step that we discuss in subsection 3.3.

Algorithm 1 Easy Rejection Sampling

Require: Target function to sample $f(\cdot)$, limited to a domain $\mathcal{X}$. A target number of samples to draw T

1: $\boldsymbol{\mu}_{1,\dots,K}, \Sigma_{1,\dots,K}, \boldsymbol{w} \leftarrow$ INITIALIZE($f, \mathcal{X}$) ▷ $g(x)$ is short for $\sum_{i=1}^{K} w_i \cdot \mathcal{N}_{\mathcal{X}}(\boldsymbol{\mu}_i, \Sigma_i)$, see Algorithm 2. Note: K is the number of initial means determined by the Initialize function.
2: $A \leftarrow []$ ▷ Accepted samples
3: $R \leftarrow []$ ▷ Rejected samples
4: $\hat{C} \leftarrow -\infty, C_{low} \leftarrow \infty$
5: refine $\leftarrow$ *False*
6: **while** $|A| < T$ **do**
7: $X^{n\cdot\log(K+1),T} \sim g(\cdot)$ ▷ Draw $n \cdot \log(K+1)$ candidate samples
8: $\tilde{C} \leftarrow \max_i \left(\frac{f(x_i)}{g(x_i)}\right)$ ▷ Batch supremum
9: refine $\leftarrow \tilde{C} > \hat{C} \vee \tilde{C} > C_{low}$
10: $\hat{C} \leftarrow \max(\hat{C}, \tilde{C})$ ▷ Empirical supremum
11: $C_{low} \leftarrow \min(C_{low} \cdot 1.05, \tilde{C})$
12: $u_i \sim \mathcal{U}(0, 1)$
13: Accept to A all $u_i \leq \frac{f(x_i)}{\hat{C} \cdot g(x_i)}$, all rejected to R ▷ $f(x_i)$ is cached for *all* samples
14: $\tilde{\boldsymbol{\mu}}, \tilde{\Sigma}, \tilde{\boldsymbol{w}} \leftarrow \boldsymbol{\mu}, \Sigma, \boldsymbol{w}$
15: **if** Last GMM call was more than $|A|/1.5$ acceptances ago or $\log(|A|) > 2 \cdot K$ **then**
16: $\boldsymbol{\mu}_{1,\dots,K'}, \Sigma_{1,\dots,K'}, \boldsymbol{w} \leftarrow$ fit GMM to $A \cup R$ with $K' = \min(\log_2(|A|), |A|/(d \cdot 15))$ clusters
17: refine $\leftarrow$ *True*
18: **if** refine **then**
19: $\boldsymbol{\mu}, \Sigma, \boldsymbol{w} \leftarrow$ REFINE($\boldsymbol{\mu}, \Sigma, \boldsymbol{w}, A, R$) ▷ Refine initial and GMM, taking the best (lowest $\bar{C}$), see Algorithm 3
20: $\bar{C} \leftarrow \max_i \left(\frac{f(\boldsymbol{x}_i)}{g(\boldsymbol{x}_i)}\right), \quad \forall \boldsymbol{x}_i \in A \cup R$
21: **if** $\bar{C} \leq \hat{C}$ **then**
22: $\hat{C} \leftarrow \bar{C}$ ▷ We keep current refinement
23: **else**
24: $\boldsymbol{\mu}, \Sigma, \boldsymbol{w} \leftarrow \tilde{\boldsymbol{\mu}}, \tilde{\Sigma}, \tilde{\boldsymbol{w}}$ ▷ We reject current refinement, and revert to previous model.
25: refine $\leftarrow$ *False*

We sample $n = 500$ items at a time, and increase the value multiplicatively with the log of the number of components K in the mixture to ensure sampling is computationally effective. We note on line 4 that we initialize the empirical supremum $\hat{C}$ to $-\infty$ instead of 1, because it allows our approach to work with both unnormalized $f(\cdot)$ and unnormalized $g(\cdot)$. The former reduces the complexity of the implementation, especially for intractable normalization terms that would require estimation. The latter helps with bounded support $\mathcal{X}$, avoiding more costly and complex evaluations of $g(\cdot)$. This is possi-

[3] Their proof is with respect only to families of the Hölder density, in worst-case situations. This does not inform a limit on the rejection rate for non-worst-case densities within that family, let alone those beyond it. For example, densities with infinite support already are outside the scope of these proofs. This is not to diminish the value of their theoretical contributions, but to elaborate that significant room for improvement is still possible.

ble because $\hat{C}$ is simply determining the maximum observed ratio, and so any missing normalizing terms in either function roll into $\hat{C}$ multiplicatively (i.e, $\frac{1}{\hat{C}} \cdot \frac{f(\cdot)/z_1}{g(\cdot)/z_2} = \frac{z_2}{z_1 \hat{C}} \frac{f(\cdot)}{g(\cdot)}$).

Lines 7-13 perform the rejection sampling, where $\tilde{C}$ is the *batch* supremum, which in most all cases converges (or closely approaches convergence) in a single iteration. We also keep track of C_{low}, a running estimate of the lowest batch supremum seen inflated by 5% per round. If the current batch supremum is above C_{low}, we flag the model for refinement. While violating C_{low} does not change the empirical supremum $\hat{C}$, it is an indicator that we may have acquired new—relatively harder—samples that will benefit the refinement optimization later. A refinement is also done if the bound $\hat{C}$ occurs, but this is rare and usually occurs due to samples occurring out in the tail of the distribution (e.g., $f(x)/g(x)$ *could* be larger for an x such that $\max(f(x), g(x)) \lessapprox 1/n$).

A *potential* GMM is fit whenever the number of accepted samples has increased by 50%, to avoid excessive training of GMMs. We do not require fitting an accurate distribution, but rather one that can be refined to a maximal acceptance rate, and do not require finding the "optimal" number of mixtures. As such we use the standard expectation maximization approach with diagonal covariance, and simply make the number of mixtures in the new GMM grow logarithmically with the accepted sample size. The diagonal covariance is used because it is faster to sample from by a significant margin, especially when a compact support $\mathcal{X}$ is given where samples can be drawn in $\mathcal{O}(1)$ time, as the general case for an arbitrary covariance is highly non-trivial [5]. We use $\mathcal{N}_{\mathcal{X}}$ to denote the normal distribution truncated to the domain of $\mathcal{X}$, which is user specified (e.g., a function in the domain $[0, \infty]$).

Why not full rank covariance? Using a full-rank Σ is problematic when we have compact support $\mathcal{X}$ (i.e., $x \in [0, \infty]$) because there is no closed form method of sampling from the truncated distribution $\mathcal{N}_{\mathcal{X}}$, which thus requires its own sampling scheme to drawn from [33]. This is extremely expensive, and we found that the Gaussian clusters that occur at the edge of the support $\mathcal{X}$ are very challenging to sample from and would increase run-time by $10,000\times$ just due to the cost of sampling from the resulting $g(\cdot)$. Thus diagonal Σ is preferred due to exact and fast sampling from it's truncated distribution.

A key optimization is that the GMM is fit to both accepted and rejected samples, by weighting each data point x_i by its true evaluation $f(x_i)$. This is essentially free, as we cache all calls to $f(\cdot)$, and requires at most doubling the memory use. Heuristically, we multiply the weight of accepted samples by a factor of 10 to reflect their greater importance to the underlying distribution, which allows our method to still work even when initial sampling rates are low. These factors all occur in lines 15-17.

Finally, we perform the refinement on lines 18-24. This is a non-convex problem, and so does not always succeed. If a GMM was attempted on lines 15-17, both the current model and the candidate GMM will be refined. We can compute the empirical supremum using the cached $f(\cdot)$ values again, to determine if the refinement has provided a new distribution $g(\cdot)$ that is better than prior solutions.

We can formally show the proofs of [7] still apply, using their notation:

Theorem 3.1. *Algorithm 1, given a fixed $g(\cdot)$, and a sequence of i samples draw thus far, converges to the same or better (fewer false samples) solution as [7], and thus retains $\mathcal{O}(i^{-1})$ convergence rate of correctness.*

Proof. Let $\widetilde{\tau}_i = \min\left\{j \in \mathbb{N} \mid U_{ij} \leqslant \frac{f(X_{ij})}{\widehat{C}_i g(X_{ij})}\right\}$ define the sequence of samples drawn from $g(\cdot)$ that define the index of the i'th sample $X_{i,\widetilde{\tau}_i}$ sampled by Empirical Rejection Sampling, and τ_i the result from using the true supremum C. If samples are selected B at a time by Algorithm 1, then the sampled index $\tau_i^B = \min\left\{j \in \mathbb{N} \mid U_{ij} \leqslant \frac{f(X_{ij})}{\widehat{C}_{i+(B-i \mod B)} g(X_{ij})}\right\}$. By definition $\widehat{C}_i \leq \widehat{C}_{i+1}$, and so a simple recurrence shows that $\widehat{C}_i \leq \widehat{C}_{i+(B-i \mod B)}$. Thus it must be the case that $\tau_i^B \leq \widetilde{\tau}_i$. Since $\widetilde{\tau}_i$ controls the acceptance of samples and forms the proof of correctness for [7], then Algorithm 1 also satisfies the proof for a fixed $g(\cdot)$. □

The proof that you can alter $g(\cdot)$ is of the same form by "restarting" the sequence when $g(\cdot)$ changes, and using the previous $X_{i,j}$ values to pick an initial $\widehat{C}_1$ that is valid (true by definition, as its the maximum ratio of $f(\cdot)/g(\cdot)$ observed so far), and beginning a new convergence of rate $\mathcal{O}(i^{-1})$ at the warm-started solution $\widehat{C}_1$.

This proves that our Algorithm 1 will converge to a solution of reasonable quality, it *does not guarantee that no erroneous samples will occur*. The probability of this can be described, but not easily quantified, as the likelihood the i'th sample x_i being drawn incorrectly is the situation that $\frac{f(x_i)}{Cg(x_i)} < u_i < \frac{f(x_i)}{\widehat{C}_{i+(B-i \mod B)} g(x_i)}$, which simplifies to answering the probability that $\mathbb{P}\left(C < \frac{\widehat{C}_{i+(B-i \mod B)} f(x_i)}{f(x_i) - \widehat{C}_{i+(B-i \mod B)} g(x_i) u_i}\right)$. This is hard to quantify due to the joint dependence on a future iteration's estimate of $\hat{C}$ (because we get samples in batches), the curvature of $f(\cdot)$ and $g(\cdot)$, and the uniform random value of u_i. We instead use the final value of $\hat{C}$ to check that empirically, we do not appear to have falsely accepted any samples. All of our experiments passed this test.

3.2 Initial Proposal Distribution

Now that we have specified the overall iterative strategy of our rejection sampler, we specify how the initial distribution $g(\cdot)$ is chosen. Congruent with standard practice, we aim for our initial proposal to be wider than the underlying distribution $f(\cdot)$, and let subsequent iterations of our algorithm narrow the proposal once samples have been obtained. Note that we count all $f(\cdot)$ evaluations in this stage against the total number of calls for computing sample acceptance rate, so it is important that we find a reasonable choice with a limited number of $f(\cdot)$ evaluations.

The procedure is outlined in Algorithm 2, and contains three primary steps: 1) handling finite support, 2) apparently unimodal distributions, 3) multi-modal distributions.

First, if the distribution was indicated to have a bounded support, we simply set $\boldsymbol{\mu}$ to have the center of the min/max bounds, and set the covariance to be wide enough to cover the entire space. This occurs on lines 3-7, and handles the finite support case. On lines 8-9 we have potentially infinite locations for the distribution, and so use random sampling to find a location that has a non-zero probability. This provides greater flexibility for our approach.

Once an initial point is selected, on lines 10-13 we sample a small number of points in the space around the initial point, and then use Stochastic Gradient Descent (SGD) to maximize the value of $f(\cdot)$. Because we are trying to empirically find some number of modes of $f(\cdot)$, we use fast converging FISTA [3] to quickly reach local maxima as implented in jaxopt [4]. These modes are used to determine whether or not the distribution is multi-modal.

Algorithm 2 Initialization of first proposal distribution $g(x) = \sum_{i=1}^{K} w_i \cdot \mathcal{N}(x|\mu_i, \Sigma_i)$

procedure INITIALIZE($f, \mathcal{X}$)
  $x \leftarrow \vec{0}$
  if $\mathcal{X}$ is compact **then**
    $\mu_1 \leftarrow$CENTER($\mathcal{X}$) ▷ Place $g(x)$ at the center.
    $\Sigma_1 \leftarrow$RANGE($\mathcal{X}$)/3 ▷ Set radius to cover the whole space with 3 σ
    $w \leftarrow [1]$ ▷ Uni-modal
    return $\mu_1, \Sigma_1, \boldsymbol{w}$
  while $f(x) = 0$ **do** ▷ Maybe discontinuous?
    $x \sim \mathcal{N}(0, I)$
  for $i \in [1, d+3]$ **do**
    $modes_i \leftarrow x+ \sim \mathcal{N}_{\mathcal{X}}(0, I)$
    $modes_{d+4} \leftarrow x$
  Run SGD on $modes$ to minimize $\sum_i -f(modes_i)$
  if Covariance of $modes_{\dots} < \epsilon$ **then** ▷ We need to estimate a covariance
    $\mu_1 \leftarrow \frac{1}{K}\sum_i^K modes_i$
    for $i \in [1, d \cdot 2 + 10]$ **do**
      $spread_i \leftarrow x+ \sim \mathcal{N}_{\mathcal{X}}(0, I)$
    Run SGA on $\sum_i \log\left(f(x) - spread_i - 5\right)^2$
    $\Sigma_1 \leftarrow$COV($spread_{\dots}$)
    $w \leftarrow [1]$
    return $\mu_1, \Sigma_1, \boldsymbol{w}$
  else ▷ Estimate multi-modal coverage
    Select K values $\mu_1, \mu_2, \dots, \mu_K$ from $modes$ via k-farthest selection, stopping when $\|\mu_K - \mu_{K+1}\|_2 < \epsilon$
    Set $\Sigma_i \leftarrow I \cdot (\max_{j,z} \|\mu_j - \mu_z\|_2 / K), \quad \forall i \in [1, K]$
    $\boldsymbol{w} \leftarrow \vec{1}/K$
    return $\mu_{\dots}, \Sigma_{\dots}, \boldsymbol{w}$

We check for unimodality by checking the covariance of the found modes, and if smaller than some ϵ, that means all points are on top of each other and thus the distribution appears unimodal (if that was an erroneous decision, it will eventually be corrected by the sampling and GMM refits). To estimate a covariance matrix, we again use SGD to find points that have a different in log probability of -5 lower, which is many orders of magnitude smaller. The covariance is estimated from these points, which—being overly far from the mode of the distribution—will have heavier tails than $f(\cdot)$ and thus allow good sampling coverage.

If the covariance is non-zero, we begin selecting the 2 farthest pairwise points, and iteratively looking at the next point that is farthest from the current set until the distance becomes $< \epsilon$, indicating that we are not selecting any new modes. This gives us a set of K modes, and we use the maximum pairwise distance divided by the number of modes as the covariance for all modes. This again provides an overestimate of the true covariance and thus helps ensure coverage.

Combined, these give us a strategy for selecting the initial distribution $g(\cdot)$ that will quickly identify empirical maximum ratios $f(\cdot)/g(\cdot)$, and allow fast convergence of our methods.

3.3 Refinement

The final, and most significant improvement of our method is the refinement operation, where we alter the GMM to improve the acceptance rate of samples drawn. Our key insight is that the kernel density estimates used by NNARS and PRS are attempting to minimize losses of the form $\int_{\mathcal{X}} |f(x) - g(x)|^p \, dx$. This is intuitive: the closer $f(\cdot)$ and $g(\cdot)$ are, the higher the acceptance rate will be. Yet, this is not what the core rejection sampling procedure addresses. Instead we can seek to refine the initial proposal model $g(\cdot)$ to minimize $\sup_{z \in \mathcal{X}} \frac{f(z)}{g(z)}$ in a direct fashion, such that we should expect high acceptance rates.

Algorithm 3 Refinement of $\boldsymbol{\mu}_{1,\dots,K}, \Sigma_{1,\dots,K}, \boldsymbol{w}$ using accepted and rejected samples A and R

procedure $\ell(\boldsymbol{\mu}, \Sigma, \boldsymbol{w}, A, R)$
  $\alpha_i \leftarrow \log(f(\boldsymbol{x}_i)/g(\boldsymbol{x}_i)), \quad \forall \boldsymbol{x}_i \in A$
  return $\langle$SOFTMAX($\boldsymbol{\alpha}$), $\boldsymbol{\alpha}\rangle$
while Max iterations not reached **do**
  Alter $\boldsymbol{\mu}$, Σ, and $\boldsymbol{W}$ using $\frac{\partial \ell}{\partial \boldsymbol{\mu}}$, $\frac{\partial \ell}{\partial \Sigma}$, and $\frac{\partial \ell}{\partial \boldsymbol{w}}$ via automatic differentiation
  if Current solution is $\leq$ than $\hat{C}$ **then**
    return $\boldsymbol{\mu}$, Σ, $\boldsymbol{w}$
return original $\boldsymbol{\mu}$, Σ, $\boldsymbol{w}$

Our approach to doing so is simple and detailed in Algorithm 3. We use the existing samples to compute the log ratio $\alpha_i = \log(f(\boldsymbol{x}_i)) - \log(g(\boldsymbol{x}_i))$, giving us a vector of empirical results. While we could select the maximal α_i to use as the loss to minimize, this is undesirable, as many α_i may be large and the optimization will take longer (especially when many mixture components K exist, using the maximum will generally influence only one of K components). Instead we use the approximate maximum computed by SOFTMAX$(\boldsymbol{\alpha})^\top \boldsymbol{\alpha}$, which allows a better behaved gradient to impact multiple components K in proportion to their log-ratios. Optimizing SOFTMAX$(\boldsymbol{\alpha})^\top \boldsymbol{\alpha}$ still directly tackles the sampling ratio $f(\cdot)/g(\cdot)$, and so results in higher acceptance rates via lower C values. Crucially this requires no further calls to $f(\cdot)$ because it is performed only on the current samples (accepted and rejected) which have already cached $f(\cdot)$ values.

This approach may be seen as an adaption of the strategy used for gumbel softmax sampling trick of [16], but instead of reparameterizing a target function we are creating a biased approximation that is advantageous in practice. In our context, the "correct" target function is to replace line 3 of algorithm 3 with $\max_{\forall i} \alpha_i$. This uses the maximal value (of $\frac{f(\cdot)}{g(\cdot)}$), but also means that $\forall j \neq i$, the gradient through α_j is exactly equal to 0 (i.e., the max operation returns a non-zero gradient only to the index that was selected). This is problematic when the ratio $\frac{f(\cdot)}{g(\cdot)}$ is multimodal, because only one mode of $g(\cdot)$ will get a meaningful gradient because only one point $\boldsymbol{x}_i$ contributed to the final loss calculation (via α_i). By using the softmax each mode of $\frac{f(\cdot)}{g(\cdot)}$ we get a gradient proportional to its scale, and so progress over the whole domain is made instead of just one location. Thus we replace the target function of interest with a *biased* approximation (i.e., Softmax$(\boldsymbol{\alpha})^\top \boldsymbol{\alpha} \neq \max_i \alpha_i$) because it results in better learning behavior. In comparison, [16] need a well-defined gradient through a stochastic function, and so require the softmax for their approach to define that gradient that is unbiased. But we both use the softmax to get a well-behaved gradient.

Because this is a non-convex optimization problem, we use the AdaBelief[34] optimizer with an initial learning rate of 0.1, and perform 800 total gradient update steps per refinement call. We check the quality of the solution at 100, 200, 400, and 800 steps and select the best (lowest maximum ratio). Efficacy could be improved by checking at every iteration, but is not advantageous from a computational perspective due to the interpreter overheads of Python.

4 Experiments and Results

Having defined our approach, we first specify our experiments, followed by results showing that ERS is competitive with or significantly better than prior methods while simultaneously requiring less information from the user. We will use several standard benchmarks used for rejection sampling problems that each exercise a different kind of challenge for which a sampler may suffer low acceptance rates. The primary measure of interest is the acceptance rate, which indicates what percentage of samples $g(\cdot)$ are accepted divided by the total number of evaluations of $f(\cdot)$. This means the gradient descent operations used in subsection 3.2 reduce the total rate of ESR. The gradient evaluations in subsection 3.3 do not count because the values of $f(\boldsymbol{x}_i)$ are cached during the initial sample acceptance/rejection evaluation, so no additional evaluations of $f(\cdot)$ are needed. Consistent with prior work, we use $n = 10^5$ target accepted samples and 10 runs to compute the mean and standard deviation of results. In all cases we report the NNARS, PRS, A*, and OS* results from prior works as we were unable to replicate their success.

Three primary and challenging problems are used as the target function $f(\cdot)$. The first is the "peakiness" problem proposed by [20] in Equation 1, where a controls how "peaky" the distribution is and the domain $\mathcal{X} \in (0, \infty)$. As $a \to \infty$ the peakiness gets higher, making sampling more difficult.

$$f(x) \propto \frac{e^{-x}}{(1+x)^a} \tag{1}$$

The next problem tests the impact of scaling the dimension size of the problem and is given for a general d-dimensional distribution in Equation 2 as proposed by [10]. Here the support is compact with $\mathcal{X} \in [0, 1]$. We note that this distribution is highly multi-modal, making it especially challenging. Most rejection sampling focuses on one-dimensional problems due to the difficulty of specifying a useful $g(\cdot)$ and C.

$$f(\boldsymbol{x}) \propto \prod_{i=1}^{d} \left(1 + \sin\left(4\pi x_i - \frac{\pi}{2}\right)\right) \tag{2}$$

The last test we consider is the "clutter" problem first posed by [22], and is given in Equation 3. Here θ indicates a set of centers that are selected from 10 points spaced uniformly in the range of $[-5, -3]$ and $[2, 4]$. This creates a distribution with two very strong peaks that are separated by a far distance, and has been a challenging sampling distribution for over two decades [22].

$$f(\boldsymbol{x}) \propto \prod_{i=1}^{N} r\left((2\pi)^{-d/2} e^{-(\boldsymbol{x}-\theta_i)^2/2}\right) + (1-r)\left((2\pi)^{-d/2} e^{-(\boldsymbol{x}^2/100^2)/2} 100^{-d}\right) \tag{3}$$

For all tests where $d \leq 2$ we ran a two-sample Kolmogorov–Smirnov (KS) test comparing ERS's samples with those of $A*$ as implemented in the original code, or with the true distribution. In no case was a difference in ERS's samples and the target distribution detected. Because the KS test is not well defined for $d > 2$, in these situations we used a two-sample Carmer test, which also found no significant difference.

We note that reproducibility of NNARS and PRS is limited. For NNARS and PRS Equation 1 results are given, though the distribution is not compact – and no transformation $[0, \infty] \to [0, 1]$ is specified. Similarly for PRS results on the clutter task Equation 3 are given, and were obtained by artificially clipping the distribution to a range that contained all samples[4]. Both methods have hyper-parameters that are not specified and would be altered by a chosen transformation, further complicating our replication of their results. Since NNARS presents the state-of-the-art results, we use their reported results as our comparison numbers though we are unable to reproduce them.

4.1 Empirical Acceptance Rates

Results will be presented in the same order as the problems were specified. Results also present a "Simple Rejection Sampling" (SRS) baseline of manually specifying $g(\cdot)$ and C as reported by prior work.

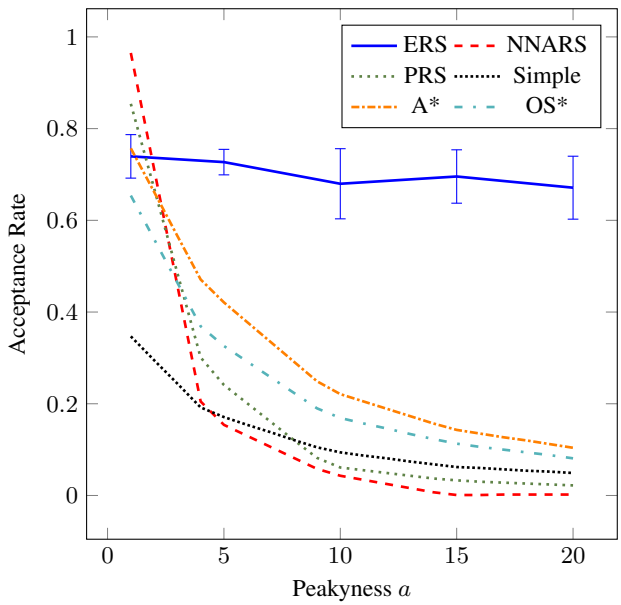

Figure 2: Results on the peakiness problem of Equation 1, where $a = 1$ indicates minimal peakiness and easier samping, and $a = 20$ is higher peakiness and more challenging to sample from. NNARS and PRS only perform better for $a = 1$, where all approaches perform well. Our ERS suffers only a 1% point drop in mean acceptance rate as a increases at each step, where all other approaches degrade quickly.

Thus we start with the peakiness problem, which has historically favored the optimization & sampling approaches of OS* and A*. Our results with standard deviation are shown in Figure 2. While ERS has a lower acceptance rate for the easiest case of $a = 1$, ERS is almost uninhibited by increased peakiness as $a \to 20$, which quickly turns into a large and dramatic advantage of 56.7 percentage points. This makes ERS the most robust to the challenge by a large margin, and we argue superior to PRS and NNARS that, while better for $a = 1$, quickly drop in efficacy down to an acceptance rate of 2% and 0.2% respectively.

We wish to point out that ERS works on Equation 1 directly, where NNARS and PRS must have applied an unknown transformation o convert Equation 1 to one with finite support. We suspect that, given the same transformation, ERS is likely to have comparable or better acceptance rate in the $a = 1$ case.

We next consider the dimension scaling problem, which historically favors KDE based approaches. The results are shown in Figure 3.

[4] This was indicated by the author when asked over email, though they do not remember the clipping value. We appreciate their responsiveness and valuable information that helped us determine replicability.

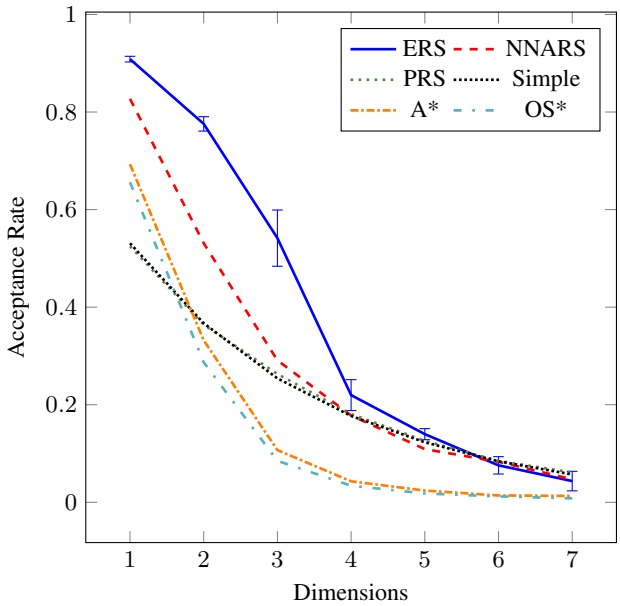

Figure 3: Results for Equation 2 as the number of dimensions d increases. In this case ERS provides superior acceptance rates for low dimensionality, but suffers from the curse of dimensional similarly to prior KDE based methods — its results becoming statistically indistinguishable as $d = 7$ is reached.

Here we see a somewhat inverted behavior, where ERS dominates for $d \leq 4$, but becomes statistically equivalent to PRS and NNARS as the dimension increases. Thus we conclude that ERS has equal or better performance in all cases for this problem.

The dimension scaling and peakiness results combined are particularly significant in that ERS performs overall best for both tasks, where previously performance favored only one between two different styles of adaptive rejection samplers. That we perform well across both tasks directly speaks to the original design goal: an easy to use sampler that can be an initial solution applied to problems and obtain effective results. In particular, we imagine that running ERS for even larger n can be an effective way of determining how challenging a problem may be to devise a better rejection sampler for, since it performs well with no tuning.

Last we consider the clutter problem, with results presented in Table 1. Here we see that ERS has a 5.6% point advantage over A* in the $d = 1$ case, and a larger 36.33% advantage in the $d = 2$ case. Though ERS's advantage comes at a mild increase in the variance of the results, the large gap in acceptance rate more than makes up for the variance.

Table 1: Clutter problem Equation 3 acceptance rate results for 1 and 2D data, including standard deviation (σ) of results.

	ERS	PRS	A*	SRS
1D	**95.0**	79.5	89.4	17.6
σ	0.7	0.2	0.8	0.1
2D	**92.4**	51.0	56.1	<0.0
σ	1.0	0.4	0.5	<0.0

4.2 Runtime Considerations

Our implementation uses JAX, making it easy to add acceleration as well as use current Python tools. This is relevant because ERS is the only vectorizable algorithm under consideration. Multiple samples and computations are collected concurrently, allowing the potential for acceleration. All models were benchmarked in a Google Colab instance, which provided a Tesla V100 GPU and a TPU. We consider only the A* and OS* for runtime comparisons at baseline since the author's code is available and implements the Clutter problem. This way we are directly comparing against the original implementation details.

The results can be found in Table 2, where we see that ERS is the fastest in terms of runtime. The acceptance rate difference does not explain the difference in runtimes. Scaling the A* runtime by the difference in acceptance rates would give a speed of 2,476.4 seconds to produce 10^5 samples, which is still $2.16\times$ slower than ERS for $d = 1$. Adding a GPU increases the speed advantage to $3.9\times$. We note that the TPU appears to be slightly slower than CPU, and is likely due to the design of TPUs to work on larger batches of data at one time for larger neural networks.

Table 2: Runtime of ERS compared to the faster A* and OS* algorithms on the Clutter problem. All times reported in seconds.

Method.	Clutter
ERS: CPU	1148.5
ERS: GPU	634.2
ERS: TPU	1398.6
A*	2631.5
OS*	3348.4

We note that the GPU has very low utilization in our testing, in part because 10^5 $d = 1$ samples is only 6.4 MB of data to be processed, which is not enough to fully leverage the compute capacity of these devices. As such we would anticipate even larger speedups for problems that required larger sample sizes, and efficacious GPU/TPU utilization for rejection sampling is an avenue for further research.

4.3 Ablating Hard-Constants

While our approach has a number of "magic numbers", each is set with a simple intuition e.g., only run the GMM when 50% more items have been sampled because there must be enough new data to get a different result. We ablate these by modifying each constant with 4 values in the range of 50% smaller to 100% larger than specified. We then run ESR for all combinations of these constants on the peakyness task with $a = 20$ as it has the most variance of any of our tests. We use the same seed for the generated samples because we want to see what the impact of these constants are, and so all runs getting the same generated sequence of samples allows us to isolate that factor. In doing so we observe a minimum acceptance rate of 75.5% and a maximum of 75.9%, indicating only a 0.4% variation due to the hard constants.

5 Conclusion

Our approach of refining a parameterized proposal distribution $g(\cdot)$ represents a new approach to defining general purpose rejection samplers. It requires fewer parameters, functions, and derivations to be specified—requiring only the target function $f(\cdot)$ to be specified, while simultaneously returning up to $7\times$ higher acceptance rates and $4\times$ lower runtime even after accounting for the difference in acceptance rates. While we do not resolve the limitations of rejection sampling to higher dimensional data, our method enables a strong and effective baseline.

References

[1] Juliette Achddou, Joseph Lam-Weil, Alexandra Carpentier, and Gilles Blanchard, 'A minimax near-optimal algorithm for adaptive rejection sampling', in *Proceedings of the 30th International Conference on Algorithmic Learning Theory*, eds., Aurélien Garivier and Satyen Kale,

volume 98 of *Proceedings of Machine Learning Research*, pp. 94–126. PMLR, (2019).
[2] David Arthur and Sergei Vassilvitskii, 'k-means++: The Advantages of Careful Seeding', in *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, volume 8, pp. 1027–1035, (2007).
[3] Amir Beck and Marc Teboulle, 'Mirror descent and nonlinear projected subgradient methods for convex optimization', *Operations Research Letters*, **31**(3), 167–175, (2003).
[4] Mathieu Blondel, Quentin Berthet, Marco Cuturi, Roy Frostig, Stephan Hoyer, Felipe Llinares-López, Fabian Pedregosa, and Jean-Philippe Vert, 'Efficient and Modular Implicit Differentiation', *arXiv*, 1–25, (2021).
[5] Z I Botev, 'The normal law under linear restrictions: simulation and estimation via minimax tilting', *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, **79**(1), 125–148, (1 2017).
[6] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018.
[7] Brian S. Caffo, James G. Booth, and A. C. Davison, 'Empirical Supremum Rejection Sampling', *Biometrika*, **89**(4), 745–754, (2002).
[8] George Casella, Christian P. Robert, and Martin T. Wells, 'Generalized Accept-Reject sampling schemes', 342–347, (2004).
[9] Jesse Dodge, Suchin Gururangan, Dallas Card, Roy Schwartz, and Noah A. Smith, 'Show Your Work: Improved Reporting of Experimental Results', in *Proceedings of EMNLP*, number 2, pp. 2185–2194, (2019).
[10] Akram Erraqabi, Michal Valko, Alexandra Carpentier, and Odalric-Ambrym Maillard, 'Pliable Rejection Sampling', in *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, p. 2121–2129. JMLR.org, (2016).
[11] M. Evans and T. Swartz, 'Random Variable Generation Using Concavity Properties of Transformed Densities', *Journal of Computational and Graphical Statistics*, **7**(4), 514, (12 1998).
[12] Alan Genz, 'Numerical Computation of Multivariate Normal Probabilities', *Journal of Computational and Graphical Statistics*, **1**(2), 141, (6 1992).
[13] Dilan Görür and Yee Whye Teh, 'Concave-Convex Adaptive Rejection Sampling', *Journal of Computational and Graphical Statistics*, **20**(3), 670–691, (1 2011).
[14] Odd Erik Gundersen and Sigbjørn Kjensmo, 'State of the Art: Reproducibility in Artificial Intelligence', *Proceedings of the 32nd AAAI Conference on Artificial Intelligence (AAAI-18)*, 1644–1651, (2018).
[15] Wolfgang Hörmann, 'A rejection technique for sampling from T - concave distributions', *ACM Transactions on Mathematical Software*, **21**(2), 182–193, (6 1995).
[16] Eric Jang, Shixiang Gu, and Ben Poole, 'Categorical reparameterization with gumbel-softmax', in *International Conference on Learning Represenations*, (Aug 2017).
[17] Stephen Jewson, Clair Barnes, Stephen Cusack, and Enrica Bellone, 'Adjusting catastrophe model ensembles using importance sampling, with application to damage estimation for varying levels of hurricane activity', *Meteorological Applications*, **27**(1), e1839, (2020). _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/met.1839.
[18] A I Jul, Marc Dymetman, Guillaume Bouchard, Simon Carter, and De Maupertuis, 'The OS* algorithm: a Joint approach to Exact Optimization and Sampling', *arXiv*, 1–21, (2012).
[19] Noureddine Kermiche. Total Rejection Sampling and the Reduction of the Wave Function, July 2022.
[20] Chris J Maddison, Daniel Tarlow, and Tom Minka, 'A* Sampling', in *Advances in Neural Information Processing Systems*, eds., Z Ghahramani, M Welling, C Cortes, N Lawrence, and K Q Weinberger, volume 27. Curran Associates, Inc., (2014).
[21] Luca Martino and Joaquín Míguez, 'Generalized rejection sampling schemes and applications in signal processing', *Signal Processing*, **90**(11), 2981–2995, (November 2010).
[22] Tom Minka, 'Expectation oropagation for approximate Bayesian inference', in *Uncertainty in Artificial Intelligence*, (2001).
[23] Kirsten Mitchell-Wallace, Matthew Jones, John Hillier, and Matthew Foote, *Natural Catastrophe Risk Management and Modelling: A Practitioner's Guide | Wiley*, May 2017.
[24] Kevin Musgrave, Serge Belongie, and Ser-Nam Lim, 'A Metric Learning Reality Check', in *ECCV*, (2020).
[25] Nguyet Nguyen and Giray Ökten. The acceptance-rejection method for low-discrepancy sequences, March 2014. arXiv:1403.5599 [q-fin].
[26] Maris Ozols, Martin Roetteler, and Jérémie Roland, 'Quantum rejection sampling', in *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, pp. 290–308, (January 2012). arXiv:1103.2774 [quant-ph].
[27] Spassimir H. Paskov and Joseph F. Traub. Faster Valuation of Financial Derivatives, August 1998.
[28] Edward Raff, 'A Step Toward Quantifying Independently Reproducible Machine Learning Research', in *NeurIPS*, (2019).
[29] Edward Raff, 'Exact acceleration of k-means++ and k-means‖', in *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, p. 2928–2935, (2021).
[30] Edward Raff, 'Research Reproducibility as a Survival Analysis', in *The Thirty-Fifth AAAI Conference on Artificial Intelligence*, (2021).
[31] Edward Raff, 'Does the market of citations reward reproducible work?', in *Proceedings of the 2023 ACM Conference on Reproducibility and Replicability*, ACM REP '23, p. 89–96, New York, NY, USA, (2023). Association for Computing Machinery.
[32] Edward Raff and Andrew L. Farris, 'A siren song of open source reproducibility, examples from machine learning', in *Proceedings of the 2023 ACM Conference on Reproducibility and Replicability*, ACM REP '23, p. 115–120, New York, NY, USA, (2023). Association for Computing Machinery.
[33] Stefan Wilhelm and B G Manjunath, 'tmvtnorm : A Package for the Truncated Multivariate Normal Distribution Generation of random numbers computation of marginal densities', *The R Journal*, **2**(1), 25–29, (2010).
[34] Juntang Zhuang, Tommy Tang, Yifan Ding, Sekhar C Tatikonda, Nicha Dvornek, Xenophon Papademetris, and James Duncan, 'AdaBelief Optimizer: Adapting Stepsizes by the Belief in Observed Gradients', *Advances in Neural Information Processing Systems*, **33**, (2020).