Project Implementation for Monte Carlo Simulator

R

January 28, 2026

Contents

Contents

1 Theoretical Basis

This simulator will be based on the Differential Evolution Monte Carlo approach, as described by (Cajo J. F. Ter Braak, 2006). We need to implement an uncertainty distribution using Bayesian priors. DEMC is popular since multiple chains run in parallel. We use DEMC because it finds an appropriate scale and orientation for the jumping distribution. **The jumps are a fixed multiple of the differences of two random parameter vectors currently in the population**. Selection process employs ”Metropolis ratio”, which defines the probability with which proposal is accepted. If uncertainty distributions are known, the efficiency of DEMC with respect to random walk Metropolis with optimal multivariate Normal jumps increases with population size. DEMC supports multidimensional updates in multi-chain ”Metropolis-within-Gibbs” sampling. It is simpler and faster than conventional models, even in the face of adverse properties like quasi col-linear parameters and multi-modal densities. This implementation uses multiple Markov chains, initialised from over-dispersed states, in parallel, and applying dynamic programming principles. Adaptive direction sampling solves the orientation problem but not the scale problem.

Section is based on (Chris Sherlock and Paul Fearnhead and Gareth O. Roberts, 2010). We aim to simulate a random walk Metropolis algorithm (RWM), where, for a given chain $\mathbf{X}$, we propose a jump given by $\mathbf{X}^*$, and define $\mathbf{Y}^* := \mathbf{X} - \mathbf{X}^*$. The jump $\mathbf{Y}^*$ is taken from the pre-specified Lebesgue density:

$$\tilde{r}(\mathbf{y}^*; \lambda) := \frac{1}{\lambda^d} r\left(\frac{\mathbf{y}^*}{\lambda}\right)$$

Where r(.) is a symmetrical function on $\mathbf{y}$, i.e. $r(\mathbf{y}) = r(-\mathbf{y})$. λ is always greater than 0 and governs the overall size of the proposed jump. λ is key to our efficiency. The proposal is accepted or rejected according to acceptance probability:

$$r(\mathbf{x}, \mathbf{y}^*) = \min\left(1, \frac{\pi(\mathbf{x} + \mathbf{y}^*)}{\pi(\mathbf{x})}\right)$$

If accepted, We have a new current value ($\mathbf{X}' \leftarrow \mathbf{X} + \mathbf{Y}^*$), otherwise the current value doesn't change. This leads to the effect that chains which come closer to a local mode are accepted, while proposals trending away from modes are accepted with probability equal to the distance from the posterior distribution at proposed and current values. Managing this probability determines how the chains diverge and remain around the posterior distribution. We define $P(\mathbf{x}, .)$ the transition kernel of the chain, which represents our proposal $\rightarrow$ acceptance / rejection process for jumps in the chain. The acceptance probability is chosen so the chain is **reversible** at equilibrium with stationary distribution $\pi(.)$. Reversibility is the property that $\pi(\mathbf{x})P(\mathbf{x}, \mathbf{x}') = \pi(\mathbf{x}')P(\mathbf{x}', \mathbf{x})$. This is valuable because reversible chains are easy to make with a pre-specified stationary distribution. For reversible geometrically ergodic chains you can also prove a central limit theorem. If we split the components of a target into *k* sub-blocks, we can split the generation into chains: $\mathbf{X} = (\mathbf{X}_1, ..., \mathbf{X}_k)$. We can write a single iteration of P(.) through the sub-blocks as:

$$\mathbf{x}_i^{(B)} := \mathbf{x}'_1, ..., \mathbf{x}'_{i-1}, \mathbf{x}_i, \mathbf{x}_{i+1}, ..., \mathbf{x}_k$$

$$\mathbf{x}_i^{(B)*} := \mathbf{x}'_i, ..., \mathbf{x}'_{i-1}, \mathbf{x}_i + \mathbf{y}_i^*, \mathbf{x}_{i+1}, ..., \mathbf{x}_k$$

Where $\mathbf{x}_j$ ' is the updated value. The acceptance probability is $\pi\left(\mathbf{x}^{(B)*}i \middle/ \pi\left(\mathbf{x}_i^{(B)}\right)\right)$. This algorithm is a generalised version of the RWM and Gibbs sampler, leading to the name **random walk Metropolis-within-Gibbs** or **RWM-within-Gibbs**. While RWM is reversible, RWM-within-Gibbs is not. Under reasonably general circumstances it can be shown the chains will converge on a stationary distribution.

1.1 Efficiency

Consecutive draws of an MCMC chain are correlated, with the marginal distributions converging to $\pi(.)$. For efficiency, we need to grasp convergence and mixing.

1.1.1 Convergence

For evaluating convergence, we can look at trace plots for different members in the chain. A Markov chain with a transition kernel P(.) is geometrically ergodic with stationary distribution π (.) if:

$$||P^n(\mathbf{x}, .) - \pi(.)||_1 \leq M(\mathbf{x})r^n$$

for some positive r < 1 and $M(.) \geq 0$ if M(.) is bounded above, then the chain is **uniformly ergodic**. Distances between measures use standard Euclidean distance. The efficiency of a geometrically ergodic algorithm is measured by the geometric rate of convergence, *r*, which is well approximated by the second largest eigenvalue of the transition kernel. Geometric ergodicity is a purely qualitative property unless $M(\mathbf{x})$ and r are known. For any geometrically ergodic reversible Markov chain satisfies a CLT for all functions with a finite second moment wrt π (.). Therefore, there exists a $\sigma_f^2 < \infty$ such that:

$$n^{1/2}\left(\hat{f}_n - \mathbb{E}_\pi\left[f(\mathbf{X})\right]\right) \Rightarrow N(0, \sigma_f^2)$$

This allows standard error calculations, which decrease with $n^{-1/2}$. When the second largest eigenvalue is also 1 a Markov chain is polynomially ergodic if:

$$||P^n(\mathbf{x}, .) - \pi(.)||_1 \leq M(\mathbf{x})n^{-r}$$

Convergence is very experimental in definition, essentially being when the next estimate only slightly wavers from the stationary average. An estimate of the expectation of a given function $f(X)$ is made more accurate than a simple average of all chains by only considering the chains after "burn-in", i.e. the transitional period between the posterior distribution and convergent estimates. Supposing that we burn in after *m* rounds, leaving *n* rounds till termination at *N*, the estimator becomes:

$$\hat{f}_n := \frac{1}{n}\sum_{m+1}^{N} f(\mathbf{X}_i)$$

1.1.2 Mixing

For a stationary chain, $\mathbf{X}_0$ is sampled from $\pi(.),\ \forall k > 0,\ i \geq 0$:

$$\text{Cov}\left[f(\mathbf{X}_k), f(\mathbf{X}_{k+i})\right] = \text{Cov}\left[f(\mathbf{X}_0), f(\mathbf{X}_i)\right]$$

or the autocorrelation at lag i. Assuming stationarity:

$$\sigma_f^2 := \lim_{n\to\infty} n \cdot \text{Var}\left[\hat{f}_n\right] = \text{Var}\left[f(\mathbf{X}_0)\right] + 2\sum_{i=1}^{\infty}\text{Cov}\left[f(\mathbf{X}_0), f(\mathbf{X}_i)\right]$$

Provided the sum exists. For our perfect sample, chains would be independent, which gives us an inefficiency estimate:

$$\frac{\sigma_f^2}{\text{Var}\left[f(\mathbf{X}_0)\right]} = 1 + 2\sum_{i=1}^{\infty}\text{Corr}\left[f(\mathbf{X}_0, f(\mathbf{X}_i))\right]$$

This is called the integrated autocorrelation time (ACT) and represents the number of independent sample equivalent to a single independent sample. To estimate this we can estimate the autocorrelation with:

$$\hat{\gamma}_i = \frac{1}{n-i}\sum_{j=1}^{n-i}\left(f(\mathbf{X}_j) - \hat{f}_n\right)\left(f(\mathbf{X}_{j+i}) - \hat{f}_n\right)$$

We can plug this estimate for the autocorrelation into the inefficiency estimate to estimate the ACT, but these terms contributions to autocorrelation are theoretically white noise, and the sum of these terms can dominate the deterministic effect of note. An alternative solution comes from the sum truncated from the first lag, *l*, for which $\hat{\gamma} < 0.05$, yielding the estimator:

$$\text{ACT}_{\text{est}} := 1 + 2\sum_{i=1}^{l-1} \hat{\gamma}_i$$

Due to high inter-run variance in the ACT, we can consider another efficiency measure called the Mean Square Euclidean Jump Distance (MSEJD):

$$S^2_{Euc} := \frac{1}{n-1} \sum_{i=1}^{n-1} ||\mathbf{x}^{i+1} - \mathbf{x}^i||_2^2$$

The expectation of S^2_{Euc} at stationarity is called the expected square Euclidean jump distance (ESEJD). A single component with variance $\sigma_i^2 := \text{Var}(X_i),\ \forall i \in N$. Since the sample is stationary, $\mathbb{E}[X'_i - X_i] = 0$, so:

$$\mathbb{E}\left[(X'_i - X_i)^2\right] = \text{Var}[X'_i - X_i] = 2\sigma_i^2(1 - \text{Corr}(X_i, X'_i))$$

When the chain is stationary and posterior variance is finite, maximising the ESEJD is equivalent to minimising a weighted sum of the autocorrelations up to a lag value $l - 1$. We can observe how autocorrelation varies against different rate parameter choices (Figure 1).
The distance (MSJD)

$$S^2_d := \frac{1}{n-1} \sum_{i=1}^{n-1} \left(\mathbf{x}^{i+1} - \mathbf{x}^i\right)^t \cdot \sum^{-1} \left(\mathbf{x}^{i+1} - \mathbf{x}^i\right)$$

is proportional to the unweighted sum of the $l - 1$ autocorrelations over the principle components of the ellipse.

1.2 Algorithmic idea

A common jumping distribution in $\mathbb{R}^d$ is the multivariate normal distribution, for which we need to identify a covariance matrix. The *d* variances and the $d(d-1)/2$ covariances must be chosen to balance progress in each step and a reasonable "acceptance rate" (the square-root of the variance relates to the relevant scale of each parameter and the correlations relate to the orientation). These are traditionally calculated experimentally. If parameters are highly correlated, special precautions must be taken to avoid singularity of the covariance matrix. We will run *N* chains in parallel, and the jumps for a current chain are derived from the remaining $N - 1$ chains. We can define a simple strategy that balances understanding and using the space: we take the differences of vectors of two randomly chosen steps, multiply the difference with factor γ, and add the result to the vector of the current chain. The difference vector contains scale and orientation information.

Each proposal is shown to define a Metropolis step, **in which each jump is as likely as the reverse jump, given the present state of the remaining chains**. The N-chain is a single random walk Markov chain on an $N \times d$ dimensional space. The core of the method is around 10 lines, it requires a random number generator and a function to

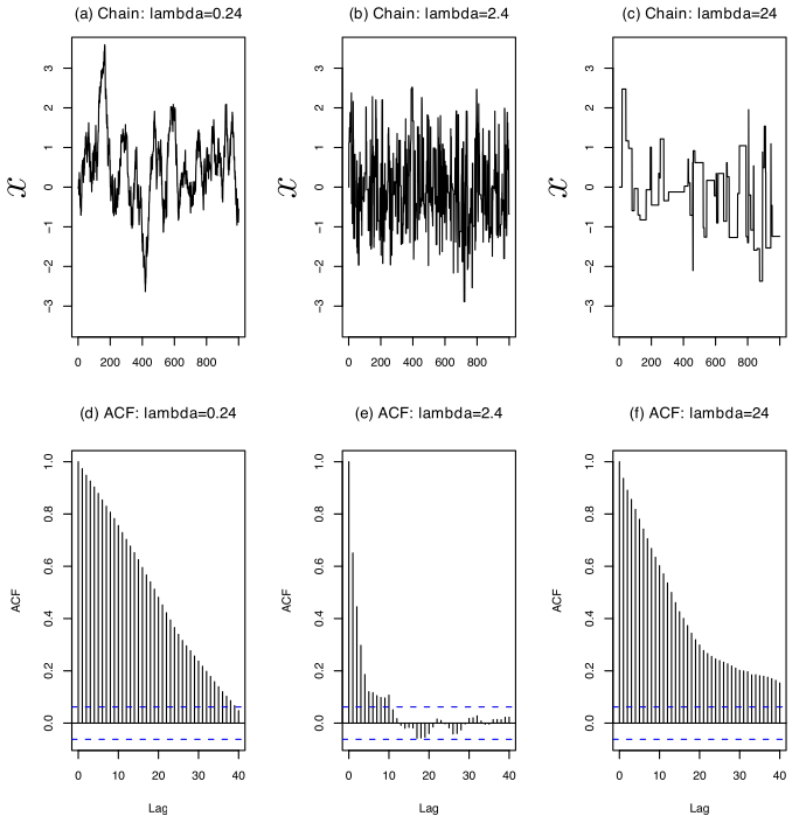

Figure 1: Trace plots and autocorrelation plots for a standard Gaussian initiailised with $x = 0$ and using an RWM with a Gaussian proposal algorithm for 1000 iterations. The scale parameters used were 0.24, 0.26, and 24 respectively

calculate the fitness of each proposal vector. This can be used for block updating in a multi-chain Gibbs sampler and provide DE variants of simulated annealing and simulated tempering. This method is tested by the authors on normal, student, normal mixtures, and two Bayesian analysis examples. The authors have given the following C-style pseudo-code for the DEMC and simulated tempering and annealing variants:

```
for (s = 0, s < N_generation, s++) { // through generations
  // randomly select 2 different numbers R1 and R2 != i
  do {R1 = floor(Uniform(0,1)*N);} while (R1 == i);
  do {R2 = floor(Uniform(0,1)*N);} while (R2 == i, R2 == R1);

\\ following Storn and Price, 1995 (DEI)

  for (j = 0; j < d; j++) {
    x_p[j] = X[i][j] + c * (X[R1][j] - X2[R2][j]) + Uniform(-b,
         b);

    r = fitness (x_p) / fitness(X[i]);


    // selection process: accept if Metropolis ratio r >
        Uniform(0,1)

    if (log(r) > Temperature * log(Uniform(0,1))): swap (X[i],
        x_p);
    // if X[i] is a draw from the target density, even if x_p
        is rejected
  } // cycled through all members of population

  Record (X);

} // end cycle through generations
// summarise recorded sample of draws
```

X is an $N \times d$ matrix with elements X[i][j] and X[i] = $\mathbf{x}_i$, the ith member chain of the population. x_p is the proposal d-vector $\mathbf{x}_i$, and fitness(.) = π (.) , c = γ. Record(X) collects the draws. CoolingSchedule() = 1 for DEMC but otherwise for simulated tempering or annealing versions. A random walk Metropolis algorithm (RWM) is a generic algorithm to sample form a d-dimensional target distribution with probability density function $\pi(.)$. We implement with the multivariate normal, centred at the current point, with variance equal to the covariance matrix. It repeatedly updates a single d-dimensional parameter vector $\mathbf{x}$ with proposal $\mathbf{x}_p = \mathbf{x} + \epsilon$ where $\epsilon \sim N(0, \widetilde{\sum})$ is selected by $\mathbf{x} = \mathbf{x}_p$, with probability $\min(1, r)$ or continuing with $\mathbf{x}$ otherwise. This creates a Markov chain with stationary distribution $\pi(.)$. In Bayesian theory, $\pi(.) \propto$ prior × likelihood. The choice of $\widetilde{\sum} = c^2 \sum, \sum = \text{Cov}_\pi(\mathbf{x})$, the covariance of the target distribution, and c as the fraction of acceptances around 0.23 for large d. For a multivariate normal target, $c = 2.38/\sqrt{d}$.

1.3 Genetic algorithms

Several Markov chains are simulated in parallel. The state of a single chain is given in the d-dimensional vectors, where these vectors are N vectors $\mathbf{x}_1 ..., \mathbf{x}_N$. The vectors become a population $\mathbf{X}$, an $N \times d$ matrix. In Bayesian analysis the initial population

can be drawn from a prior distribution. DE is a simple genetic algorithm for optimisation in real parameter spaces. For N > 4, the default proposal for ith member $\mathbf{x}_i$ is:

$$\mathbf{x}_p = \mathbf{x}_{R0} + \gamma(\mathbf{x}_{R1} - \mathbf{x}_{R2})$$

Where our parameters are randomly selected without replacement from the population (without $\mathbf{x}_i$). The proposal vector is retained if the fitness of $\mathbf{x}_p$ is higher than the fitness of $\mathbf{x}_i$. If the fitness function is $\pi(.)$, then the proposal is accepted if $r = \pi(\mathbf{x}_p)/\pi(\mathbf{x}_i) > 1$. Typically, $0.4 < \gamma < 1$. We turn DE into a Markov chain by balancing the proposal and acceptance scheme with respect to $\pi(.)$. To ensure the entire parameter space can be reached, we modify our expression into:

$$\mathbf{x}_p = \mathbf{x}_i + \gamma(\mathbf{x}_{R1} - \mathbf{x}_{R2}) + \mathbf{e}$$

Where **e** is drawn from a symmetric distribution with a small variance compared to the target, this has unbounded support ($\mathbf{e} \sim N(0, b)^d$), where *b* is small. The (Cajo J. F. Ter Braak, 2006) paper adds a probabilistic acceptance rule to DE: proposal (2)is accepted with probability $\min(1, r)$ where $r = \pi(\mathbf{x}_p)/\pi(\mathbf{x}_i)$. This algorithm is called DEMC. To this end, they write the following theorems:

> Theorem 1: DEMC yields a Markov chain, with a unique stationary distribution that has pdf $\pi(.)^N$.
>
> **Proof:** We have two parts a. $\pi(.)$ is a stationary distribution of the ith chain, since the chain is reversible (the jumps in each chain satisfy detailed balance wrt $\pi(.)$ at each step). For the ith member, the probability from the jump of $\mathbf{x}_i$ to $\mathbf{x}_p$ is equal to the reverse jump
>
> $$\mathbf{x}_i = \mathbf{x}_i - \gamma(\mathbf{x}_{R1} - \mathbf{x}_{R2}) - \mathbf{e} = \mathbf{x}_p = \mathbf{x}_i + \gamma(\mathbf{x}_{R2} - \mathbf{x}_{R1}) - \mathbf{e}$$
>
> Since the pairs are equally likely, the distribution of **e** is symmetric. If $\mathbf{x}_i \sim \pi(.)$, then balance is achieved point-wise by accepting the proposal with probability $\min(1, r), r = \pi(\mathbf{x}_p)/\pi(\mathbf{x}_i)$. The Jacobian of the transformation of (2) is 1 in absolute value. Detailed balance also holds over arbitrary measurable sets. Conditionally on the other chains, π (.) is a stationary distribution of the ith chain. As the conditional stationary distribution does not depend on the state of the other chains and is identical for all chains, $\pi(\mathbf{x}_1, ..., \mathbf{x}_N) = \pi(\mathbf{x}_1) \times ... \times \pi(\mathbf{x}_N)$ is a joint stationary distribution.
>
> b. The stationary distribution is unique if the chain is aperiodic, not transient, and irreducible. The first two conditions are generally satisfied because DEMC generates random walk for each member chain. For the third condition, it is required that any state can be reached with positive probability, and this is guaranteed by unbounded support of the distribution of **e**. Each component therefore has a unique stationary distribution, from (a), we know this must be π (.).

Because the joint stationary PDF of the N chains factorises out across the states, the individual chains are independent at any generation after DEMC has reached independence from the initial value. This allows us to derive the Gelman $\hat{R}$ statistic (they suggest below 1.2).

1.4 Practical application of DEMC

If they exist, $\mu = \mathbb{E}(\mathbf{x}), \sum = \text{Cov}(\mathbf{x})$, the expectation and covariance of the target distribution. After convergence, for each population member i and j:

$$\mathbb{E}\left[(\mathbf{x}_i - \mathbf{x}_j)(\mathbf{x}_i - \mathbf{x}_j)^T\right] = 2\sum$$

with expectation across generalisations. The average across the population at each generation converge for large N to the expectation and covariance of the target distribution.

$$\tilde{\mu}(\mathbf{x}_i) \to \mu,\ \tilde{\mu}\left[(\mathbf{x}_i - \mathbf{x}_j)(\mathbf{x}_i - \mathbf{x}_j)^T\right] = 2\sum$$
$$N \to \infty$$

Where we take the average across the pairs of population members for a paired mean $\tilde{\mu}$. For large N and small b, the proposal looks like $\mathbf{x}_p = \mathbf{x}_i + \gamma\epsilon$ with $\mathbb{E}[\epsilon] = \mathbf{0}$ and $\text{Cov}[\epsilon] = 2\sum$, the covariance matrix of the target. Specifically in the case where $\pi(.)$ is the multivariate normal, then $\gamma\epsilon \sim N(0, 2\gamma^2\sum)$ such that DEMC is expected to behave like RWM. The optimal choice of γ then becomes $2.38/\sqrt{2d}$. This choice of γ is expected to give an acceptance probability of 0.44 for d = 1, 0.28 for d = 5, and 0.23 for large d. If the initial distribution is drawn from the prior, then DEMC translates the prior into the posterior populations. In the situation $N \leq d$, all proposals lie in an N-1 space when $\mathbf{e} = \mathbf{0}$. This convergence depends on $\mathbf{e}$, so convergence could take a long time if variance is small.
The authors used $\mathbf{e} \sim \text{Unif}[-b, b]^d, d = 10^{-4}$ for their simulations. For their multivariate normal and student distribution (3 dof) estimations they used zero means, and set their covariance matrix such that the variance of the jth variable was equal to j and the pairwise correlations were all 0.5. They also simulated bimodal distributions using two normal mixtures. They used the default $\gamma = 2.38/\sqrt{2d}$. In the sequel, draws count the number of proposal evaluations (each using one evaluation of $\pi(.)$) and generations refer to cycles through the population.

1.4.1 Multivariate Normal

We can observe that with more members (higher N), the evolutionary process is far smoother (Figure 2). Convergence also seems to occur faster in populations with lower variance. Mean convergence happens within generally less than 100 generations but standard deviation of the last variable and covariance take a long time to converge, likely due the large number of estimates required. In all normal distribution test cases however, the estimates eventually converged around the true value, even in the presence of bad priors. They find N = 200 fastest in practice. The $\hat{R}$ -statistic crosses 1.2 for all 100 parameters after 900-1000 generations for all tested population sizes. The acceptance fraction varied (binomially) around 0.2 for N = 101. But after convergence, for around $N \geq 200$ the mean fraction was 0.23. We see steady increases in acceptance rates after time, though not stable enough to build convergence around. They also show experimentally that their efficiencies increase with N/d
We can identify our covariance matrix in a known distribution by simulating different values for *d* and *N*. The efficiency metric the authors use is
$100 \times \text{MSE}_{\text{RWM}}/\text{MSE}_{\text{DEMC}}$. The statistics were the empirical percentiles, taken from the sample for the first and *d*-th dimensional targets in a *d*-dimensional simulation.

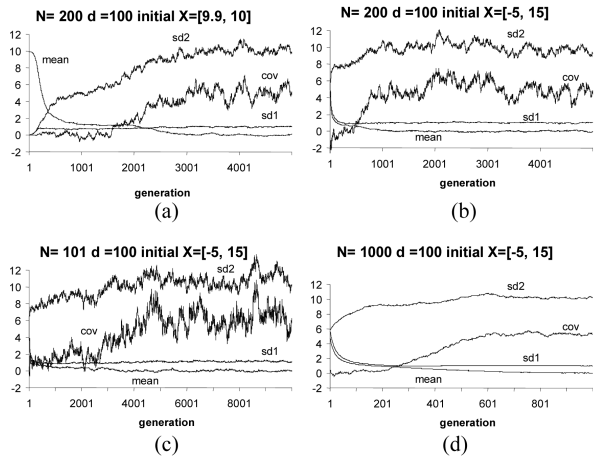

Fig. 3 How the mean and (co)variance of a Population of N members convergence to true values for a 100-dimensional Normal target in relation to N and initial population $\mathbf{X}$. Shown are the mean of the first variable, the standard deviations (sd1 and sd2) and covariance (cov) of the first variable and the last variable. The true values are 0, 1, 10 and 5, respectively. (a) narrow initial population, Uniform$[9.9,10]^{100}$; (b)–(d) broad initial population, Uniform$[-5,15]^{100}$; (a)–(b) $N = 200$; (c) $N = 101$; (d) $N = 1000$.

Figure 2: Testing Simulating Known Normal Distributions

Theoretical MSEs for the normal distribution are equal for 2.5 and 97.5 percentiles, their estimated MSEs were averaged for the MSE_{DEMC} statistic. Experimentally found that DEMC had high efficiency, exceeding 70% (Figure 3). They did not choose to change γ for the normal distribution after testing, and found that $c = 3.0$ was a good selection across dimensional sizes. Even with 10^5 burn in and 10^6 draws for an initial distribution of Unif [-5, 15]$^{\text{d}}$ that neither RWM nor DEMC converged based on Gelman's $\hat{R}$. However, after shifting to an initial distribution with a normal distribution with the true mean and covariance they had no convergence problems.

	Normal						Student t_3			
	$d = 5$		$d = 50$		$d = 100$		$d = 5$		$d = 50$	
N	P50	P2.5	P50	P2.5	P50	P2.5	P50	P2.5	P50	P2.5
$2d$	82	82	91	81	71	74	68	70	88	147
$3d$	100	87	85	80	92	91	86	96	102	191
$10d$	113	86	131	84	127	100	92	99	129	501

Note. The estimated MSEs per draw of RWM were, in column order, 20, 59, 174, 396, 335, 823, 12, 962, 121 and 41604. P2.5 is a pooled efficiency for the 2.5 and 97.5 percentiles.

Figure 3: Estimated efficiency for simulations at different percentiles

1.4.2 Normal Mixture Target

In this test, they targeted a mixture f two normal distributions:

$$\pi(\mathbf{x}) = \frac{1}{3}N_d(-\mathbf{5}, \mathbf{I}_d) + \frac{2}{3}N_d(\mathbf{5}, \mathbf{I}_d)$$

If it's bold it's a vector across our d-dimensions.

2 Random Ideas

1. We need to figure out how exactly we implement an uncertainty distribution.
2. Implement jumping distribution, probably multivariate normal.
3. It sounds like we'll confront threading now.
4. Should research Metropolis ratio and Metropolis-Within-Gibbs.
5. For randomness, a Mersenne twister will probably do the job, just have to read up on the documentation. This can feed a uniform random variable sampler.
6. We can static cast to (int) for a floor function.
7. How do we find the correlations for the covariance matrix?
8. OpenGL, Metal, and Vulkan

3 References

Cajo J. F. Ter Braak (2006). *A Markov Chain Monte Carlo version of the genetic algorithm Differential Evolution: easy Bayesian computing for real parameter spaces*, Statistics and Computing.
Chris Sherlock and Paul Fearnhead and Gareth O. Roberts (2010). *The Random Walk Metropolis*, Statistical Science.
Their citation for DE comes from Price and Storn, 1997. Gilks and Roberts, 1996 Roberts and Rosenthal (2001) Storn and Price, 1997 Robert and Casella, 2004 Roberts and Rosenthal, 2001

Table 1: Clock summary at *[2026-01-27 Tue 20:46]*

Headline	Time	
Total time	**1:26**	
Theoretical Basis	1:26	
Practical application of DEMC		1:26