# Computational Statistics — Assessed Coursework

06049216

## Contents



## 1 Question 1

**Part a.**

We begin the question by performing a standard Random–Walk Metropolis–Hastings sampler with normally distributed noise, i.e.

$$X_{n+1} = X_n + \sigma\varepsilon_n, \qquad \varepsilon_n \sim N(0,1).$$

We will test this over a range of $\sigma$.

From inspecting $f$ in Figure 1, it is clear that numerical underflow will be an issue if the exponential terms are evaluated directly. Taking the logarithm does not solve the problem, as underflow occurs before the logarithm is applied.

Instead, we rewrite $\log f(x)$ in a numerically stable form:

$$\log f(x) \propto \begin{cases} -\infty, & x < 2, \\ \xi(x) + \log\big(e^{a_1(x)-\xi(x)} + e^{a_2(x)-\xi(x)} + e^{a_3(x)-\xi(x)}\big), & x \geq 2, \end{cases}$$

where

$$a_1(x) = -3(x-2), \qquad a_2(x) = -(x-30)^2, \qquad a_3(x) = -\frac{(x-20)^2}{0.01},$$

and

$$\xi(x) = \max\{a_1(x),\, a_2(x),\, a_3(x)\}.$$

This ensures that at least one of the terms inside the logarithm is equal to 1, preventing underflow.

Based on this, we run the RWMH algorithm (which will we also refer to as Algorithm $a$), with starting values being chosen as $X_0 \sim \text{U}(2, 40)$ to give variation in which of the three modes our chain will begin in. We will test out an evenly spaced sequence of 50 $\sigma \in [1, 50]$ as well as a more specific run where we will test 50 evenly spaced $\sigma \in [0.1, 10]$. For each of these runs, we will run $N = 30$ chains, discarding $\{X_1, \ldots, X_{2000}\}$ as burn-in. In other words, our sample will be made up of $X_n^{(N,\sigma)}$ for $n \in \{2001, \ldots, 10000\}$, $N \in \{1, 30\}$.

First, we can look into convergence. Convergence is well measured by the Gelman-Rubin statistic $\hat{R}$ and when $\hat{R} \approx 1$ this is is indicative of very good convergence. To achieve this we need to run multiple chains with the same sigma values, and we choose to run $N = 30$ chains for each $\sigma$ to compute their $\hat{R}$ values.

We also must consider mixing. Rosenthal (2011) found that optimal mixing for the one-dimensional case of RWMH as the acceptance rate $\alpha \approx 0.44$ (where acceptance rate is the empirical proportion of proposals accepted). Another useful diagnostic is effective sample size (ESS). In this case, for each $\sigma$ we have $(10000 - 2000) \times 30 = 240{,}000$ $X_i$ values, so we are looking at the proportion of effective i.i.d samples to actual samples.

Note: As ESS relies on the CLT, we prove this holds in part e.

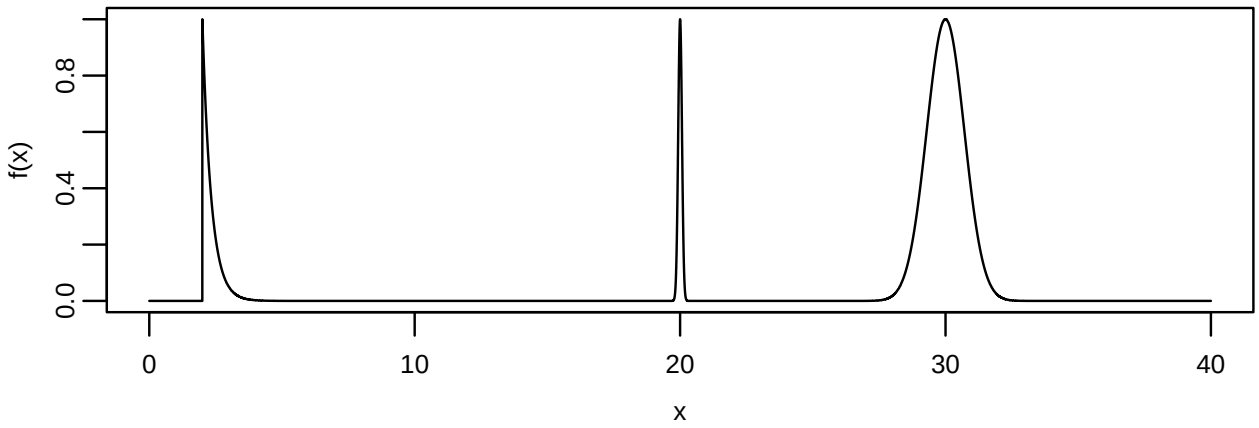


Figure 1: f(x) over x

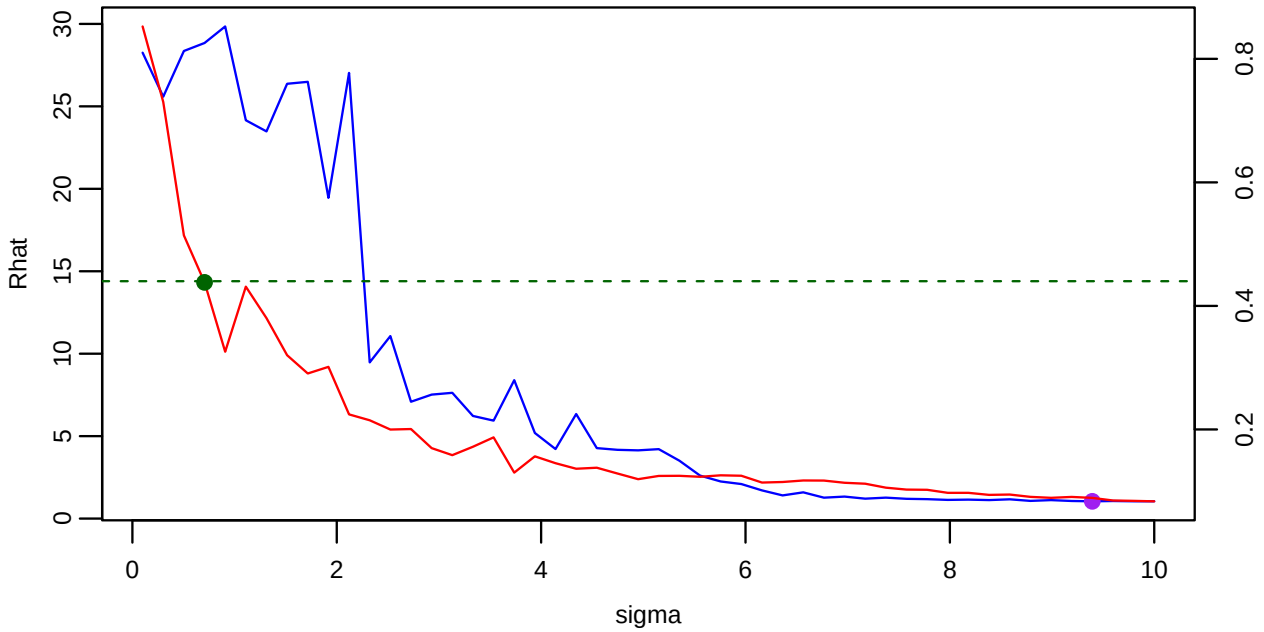


Figure 2: Acceptance rate (red), R hat (blue) over different variances for RWMH. Green line shows optimal acceptance rate, green point is the value that is closest to this. Purple point is the lowest GR statistic below 1.05

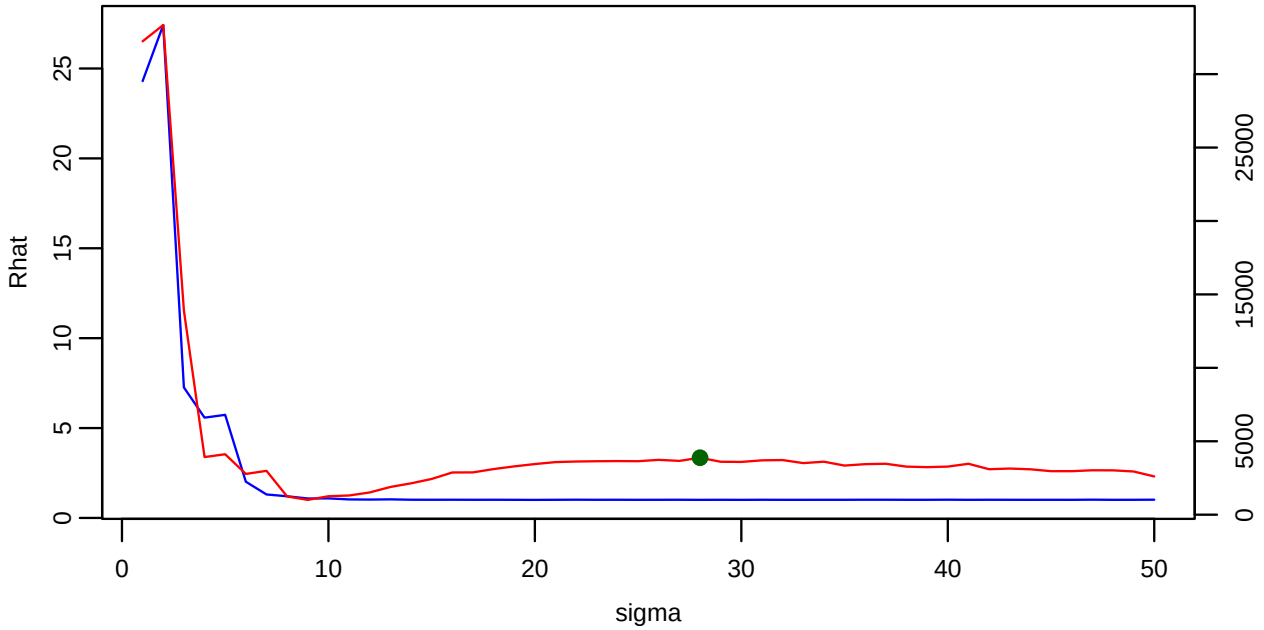


Figure 3: ESS (red) and R hat (blue) over different variances for RWMH. Green point is the highest ESS amongs acceptable GR statistic values

Figure 2 and Figure 3 shows us it is rather hard to achieve both good mixing and good convergence. For the $\sigma$ that mix well, they often converge badly and vice versa. It is something of an inherent problem with doing regular RWMH on $f$, either $\sigma$ is too large to explore intra-mode efficiently (poor mixing) or too small to explore inter-mode efficiently (poor convergence).Based on a simple RWMH, we could choose $\sigma = 28$ from Figure 3, as this is the highest ESS with a reasonable $\hat{R}$, but this is still very poor mixing, with an ESS of

less than 5000 from 240,000 iterations.

If we were limited to only doing RWMH, a way around this could be that the $\sigma$ for each step is chosen at random, such that it will either perform well inter-mode or intra-mode. Performing this with $\sigma_1 = 0.7$ and $\sigma_2 = 9.4$ (where $\sigma_1$ is the value from Figure 2 which is closest to achieving optimal $\alpha$ and $\sigma_2$ is the lowest value in Figure 2 which achieves $\hat{R} < 1.05$).

This however poses a problem, which is that each of the modes mix differently, as they each have more or less extreme peaks. Nevertheless, we decided to test this as previously with $n = 10000$, discarding 2000 burn in and $N = 30$, but using the mixed $\sigma$ values from Figure 2. Here, we can consider at each step $X_{n+1} = X_n + \sigma_n \varepsilon_n$ such that:

$$\sigma_n = 0.7 \cdot Y_n + 9.4 \cdot (1 - Y_n), \quad \text{where } Y_n \sim \text{Bernoulli}(0.5)$$

Looking at the trace plot on Figure 4, there are some signs of jumping between modes and mixing within modes, but it still looks far from optimal. Some reprise comes in the diagnostic testing, with $\hat{R} = 1.09$ (upper 95% C.I. 1.14) and an acceptance rate of $\alpha \approx 0.35$, but there is great room for improvement with different methodology (these results are shown in the supplementary material).

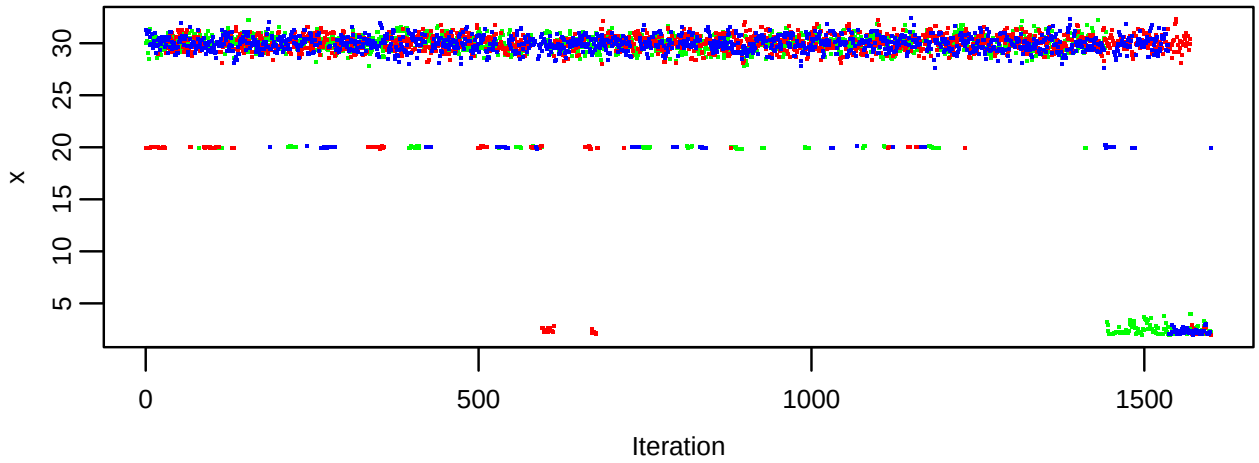


Figure 4: Trace plot of mixed RWMH switching between optimal mixing and optimal converge variance, each colour is a different chain. Thinned by a factor of 5 for clarity.

**Part b.**

The proposal we will choose is a mixture distribution

$$g(x) = w_1 g_1(x) + w_2 g_2(x) + w_3 g_3(x),$$

where

$$g_1(x) = \text{Exp}(x - 2 \mid 3) \qquad g_2(x) = \text{N}(x \mid 20,\, 0.1^2), \qquad g_3(x) = \text{N}(x \mid 30,\, 1).$$

The weights of the mixture were drawn as the relative weight of each component of $f(x)$, i.e:

$$w_1 = \frac{\int_2^4 f(x)dx}{\int_2^4 f(x)dx + \int_{19.5}^{20.5} f(x)dx + \int_{27}^{33} f(x)dx}, \quad w_2 = \frac{\int_{19.5}^{20.5} f(x)dx}{\int_2^4 f(x)dx + \int_{19.5}^{20.5} f(x)dx + \int_{27}^{33} f(x)dx}$$

$$w_3 = \frac{\int_{27}^{33} f(x)dx}{\int_2^4 f(x)dx + \int_{19.5}^{20.5} f(x)dx + \int_{27}^{33} f(x)dx}$$

These are shown graphically in Figure 5. Of course these are not exact, but will be close enough approximation to the true weights.

A proposal draw $Y$ is generated by

$$Y \sim \begin{cases} \text{Exp}(3) + 2, & \text{with probability } w_1, \\ N(20,\, 0.1^2), & \text{with probability } w_2, \\ N(30,\, 1^2), & \text{with probability } w_3. \end{cases}$$

Given the current state $X_t$, generate a proposal $Y_{t+1} \sim g$.
The acceptance probability is

$$\alpha(X_t, Y_t) = \min\left(1,\ \frac{f(Y_t)\, g(X_t)}{f(X_t)\, g(Y_t)}\right).$$

Let $U \sim \text{Uniform}(0, 1)$. Then the next state of the chain is:

$$X_{t+1} = \begin{cases} Y, & \text{if } U < \alpha(X_t, Y_t), \\ X_t, & \text{if } U \geq \alpha(X_t, Y_t). \end{cases}$$

As in the previous section, we generate 30 independent chains.
Chain $i$ has length 10,000, starting at

$$X_0^{(i)} \sim \text{Uniform}(2, 40).$$

For each chain, the first 2,000 iterations are discarded as burn–in.
The retained part of chain $i$ is

$$\{\, X_{2001}^{(i)}, \ldots, X_{10000}^{(i)} \,\}.$$

Each chain contributes 8,000 post burn–in samples.
The final sample used for inference is the concatenation

$$\{X_{2001:10000}^{(1)},\ X_{2001:10000}^{(2)},\ \ldots,\ X_{2001:10000}^{(30)}\},$$

giving a total of $30 \times 8000 = 240{,}000$ draws from the approximate target distribution.

We will refer to this process as Algorithm $b$.

Since our normal components are not truncated, there is an argument that $Y_t < 2$ is a possibility. However, this is reasonably small to regard as negligible, specifically we have $\int_{-\infty}^{2} g(x)\, dx < 10^{-150}$, so a proposition of $Y_t < 2$ is reasonably improbable (result in Supplementary Material).

We have a few options to assess convergence performance. Firstly, we can look at trace plots. If we have good convergence, we expect that the chain will jump around modes, regardless of where it started, which we can observe in Figure 8. We can also view the empirical density of our MCMC samples to see if they have converged to something resembling $f$. Figure 6 shows us a very close match which is a positive sign. Additionally, Figure 7 demonstrates $\hat{R}$'s rapid convergence to 1. Finally, from Table 1 we observe that the proportion of our samples that are within each mode are resembling the corresponding $w_i$ values. These metrics provide strong evidence that our algorithm has converged to $f$ very well.

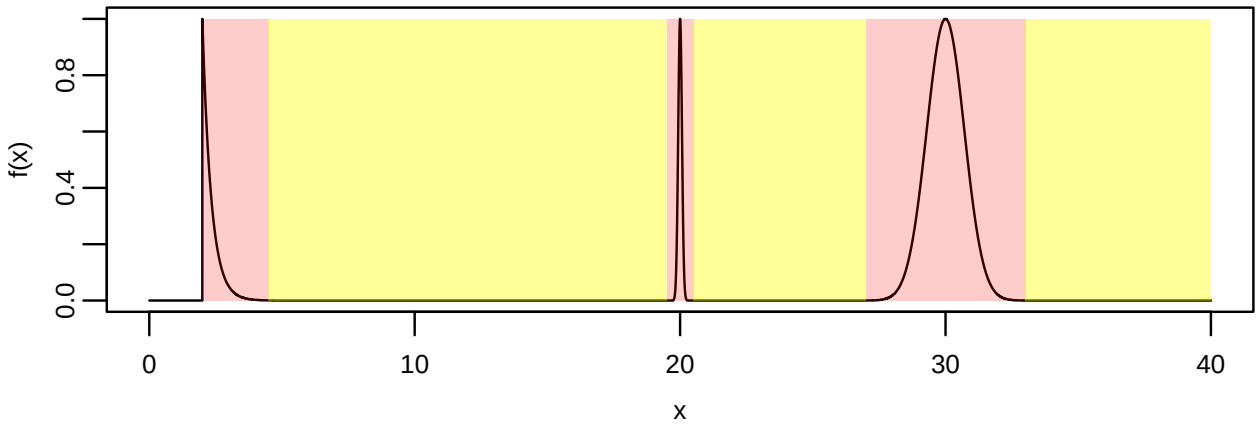


Figure 5: f(x), with red rectangles showing w1,w2,w3

Table 1: Proportion of samples in each mode in 240,000 samples of Algorithm b and proportional area of each mode

| Proportion | $w_i$ |
|---|---|
| 0.1453 | 0.1459 |
| 0.0771 | 0.0776 |
| 0.7775 | 0.7764 |

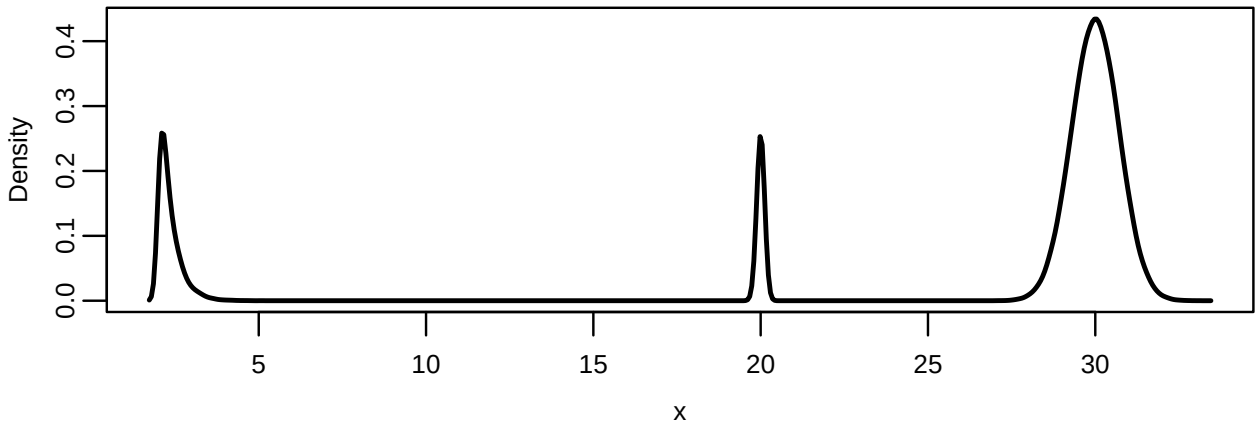


Figure 6: Empirical sample density of 240,000 samples from Algorithm b

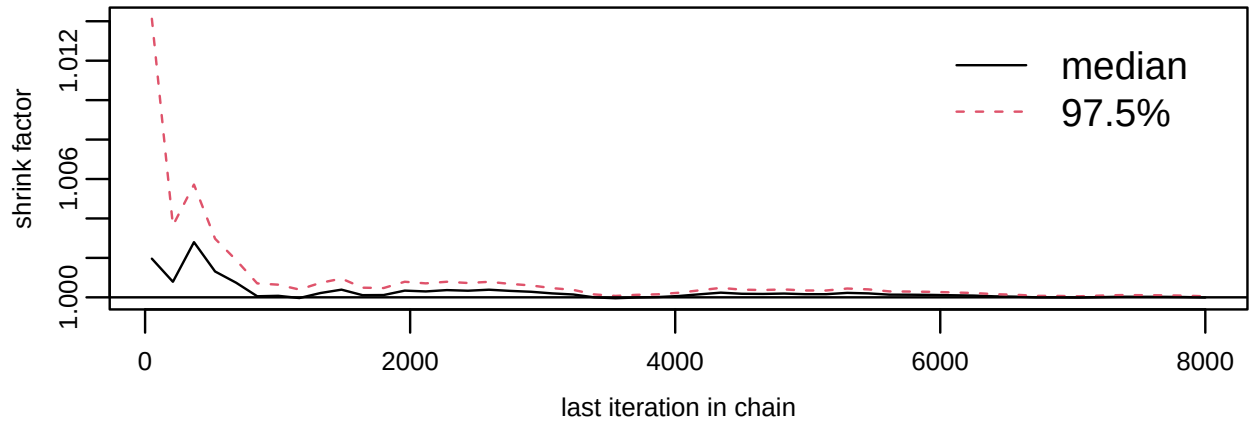


Figure 7: Gelman-Rubin statistic and chain length for 30 chains using Algorithm b

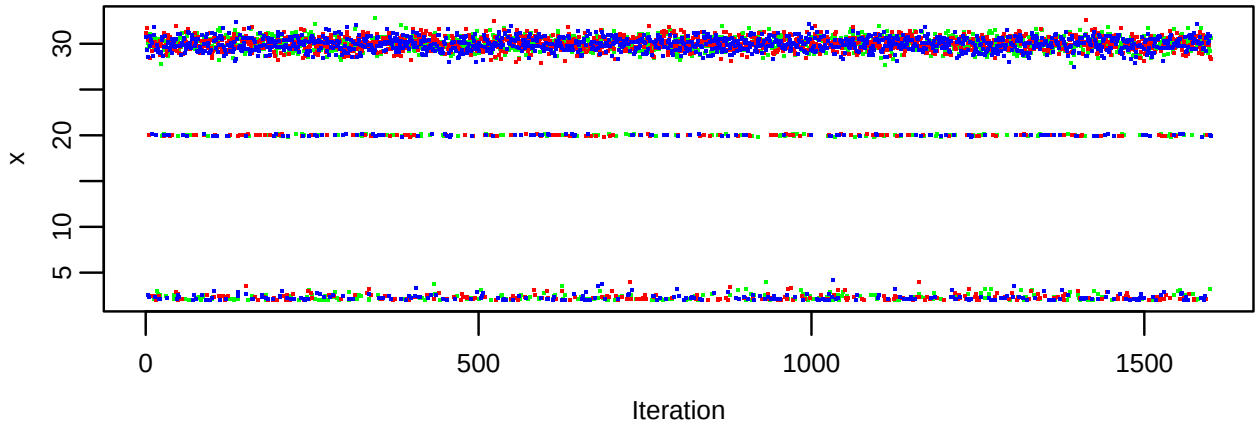


Figure 8: Trace plot of three chains of MCMC samples, Algorithm b. Thinned for clarity.

**Part c.**
Since we have a proposal which factors in the trimodality of our target, parallel tempering seemingly offers little benefit. A pseudocode example is as follows, which we will call Algorithm $c$:

**Algorithm c**

**Input:**

1. Target $f(x)$.
2. Proposal $g(x)$.
3. Temperature vector $T$ with length $m$
4. Number of replications $N$.
5. Iterations per run: $n$.
6. Burn-in discard proportion $B$.

---

- For $N' \in \{1, \ldots, N\}$:
  1. Draw initial states for all $m$ temperature chains:

$$X_{0,j}^{(N')} \sim \text{Uniform}(2, 40), \qquad j = 1, \ldots, m.$$

  2. For $t = 1$ to $n$:
     (a) For each chain $j = 1$ to $m$:

– Draw $Y \sim g(Y \mid X_{t-1,j})$.
– Calculate the Metropolis-Hastings acceptance probability:

$$\alpha_{t,j} = \min\left\{1,\ \frac{f(Y)^{1/T_j}\, g(X_{t-1,j})}{f(X_{t-1,j})^{1/T_j}\, g(Y)}\right\}.$$

– Set $X_{t,j} = Y$ with probability $\alpha_{t,j}$, otherwise $X_{t,j} = X_{t-1,j}$.

(b) For each chain $j = 1$ to $m$:
– Choose neighbour chain $k$ according to:

$$\mathbb{P}(k \mid j) = \begin{cases} 1, & j = 1,\ k = 2, \\ \frac{1}{2}, & 1 < j < m,\ k \in \{j-1, j+1\}, \\ 1, & j = m,\ k = m-1, \end{cases}$$

– Compute the swap acceptance probability:

$$a_t(j,k) = \min\left\{1,\ \frac{f(X_{t,k})^{\,1/T_j - 1/T_k}}{f(X_{t,j})^{\,1/T_j - 1/T_k}}\right\}.$$

– Draw $u \sim \text{Uniform}(0,1)$. If $u \le a_t(j,k)$, swap $X_{t,j} \leftrightarrow X_{t,k}$.

**Output:** The concatenated sequence of post-burn-in samples from $T = 1$:

$$\left\{X^{(1)}_{B\cdot n+1:n,1}, \ldots, X^{(N)}_{B\cdot n+1:n,1}\right\}$$

We input the following into the algorithm (as well as the $f(x)$ defined in the question):

$$g(x) = w_1 g_1(x) + w_2 g_2(x) + w_3 g_3(x)$$
$$T = \{1, 2, 5, 10, 20, 40, 70\}, \quad N = 30, \quad n = 10000, \quad B = 0.2$$

We can see the effect that tempering has on $f$ in Figure 9. Implementing Algorithm $c$, we can indeed verify with Figure 10 and Figure 11 that the target distribution is being well sampled from.

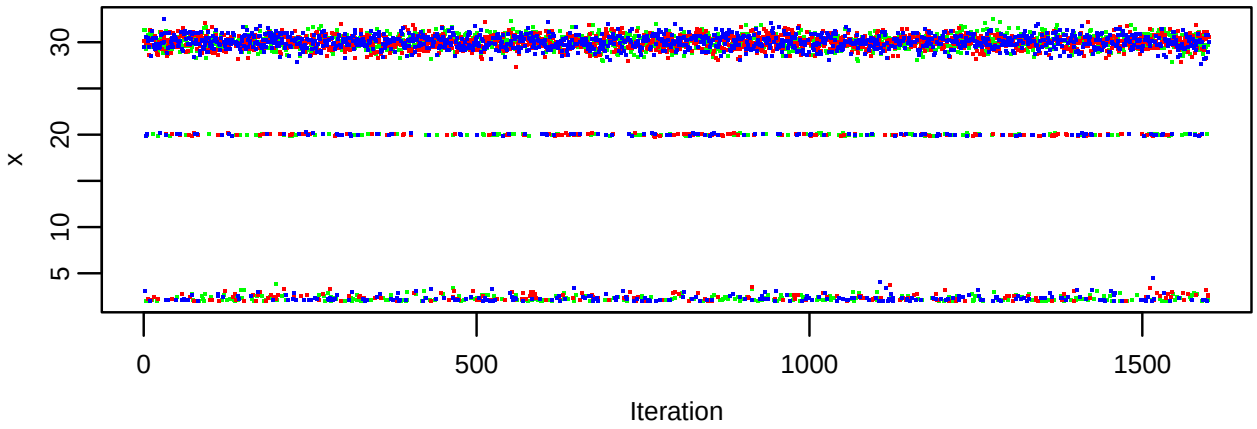


Figure 10: Trace plot of three chains of MCMC samples, Algorithm c. Thinned for clarity.

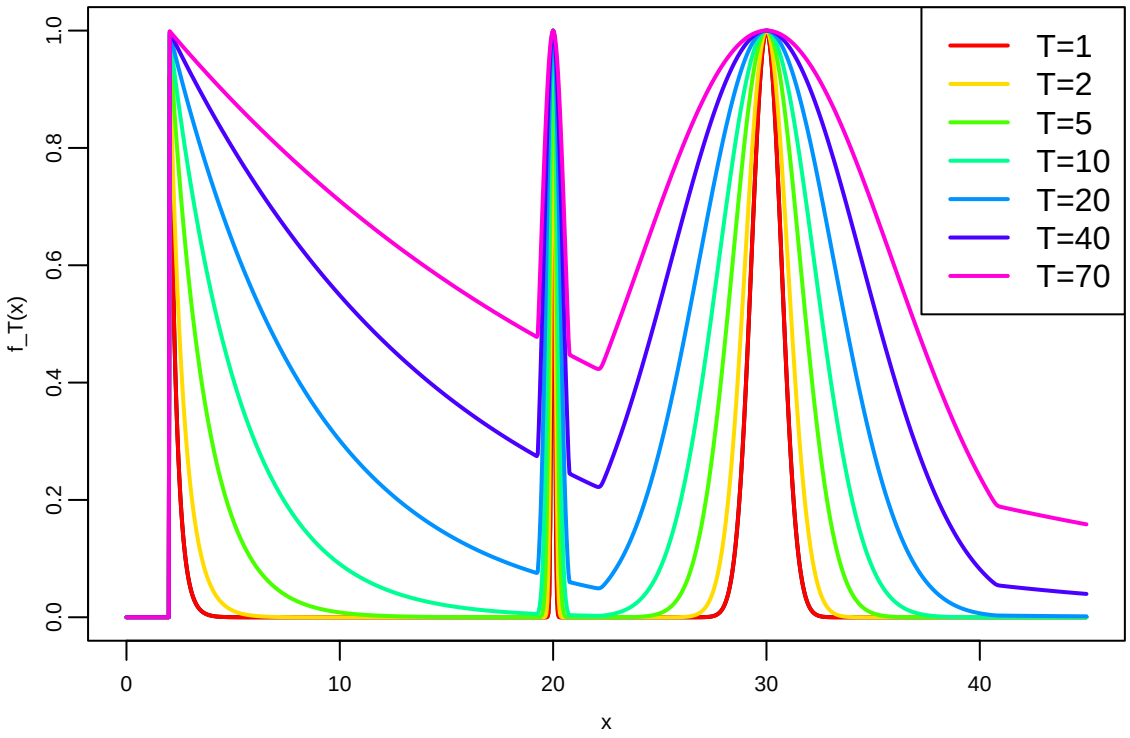


Figure 9: Differing temperatures of f

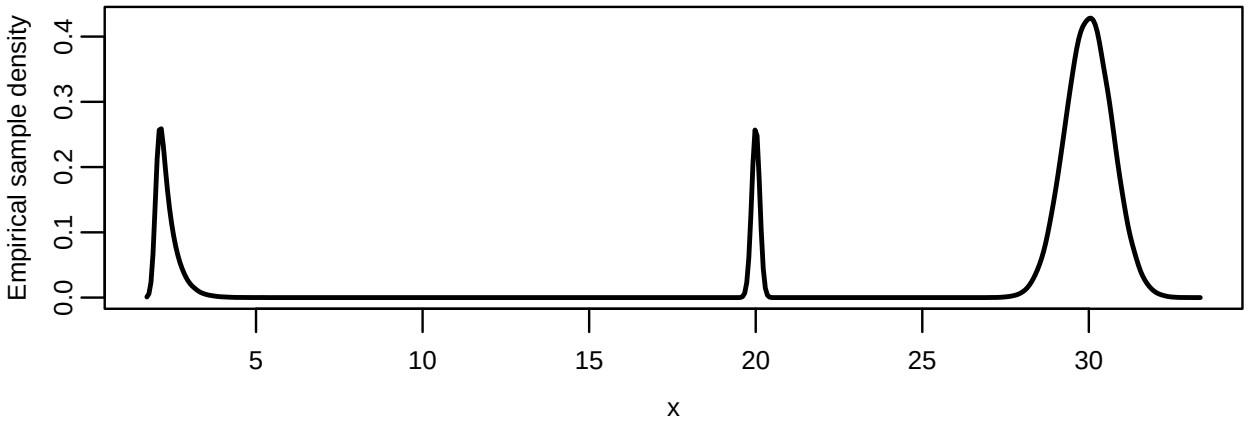


Figure 11: Empirical sample density of 240,000 samples from Algorithm c

**Part d.**

We will run Algorithms $a$,$b$,$c$ with $n = 1000$ (800 when accounting for burn-in), and run 30 chains of each, giving us a concatenated sample length of 24,000. This lower value for chain length will allow the differences in convergence to materialise. We will use the same $T$ values as previously.

From Table 2, it is clear that Algorithms $b$,$c$ outperform Algorithm $a$ in convergence, with a lower Gelman-Rubin $\hat{R}$ and upper bound. However, in mixing Algorithm $b$ pulls ahead, as it is more efficient to run than the costly extra chains involved in Algorithm $c$. Conceptually this makes sense, our proposal was already very well matched to the target distribution, the extra chains was adding very little improvement in terms of mixing (seen in an approximately 10% increase in ESS) at a huge computational cost (approx 1000% increase)

Table 2: Convergence and efficiency summaries for Algorithms a, b, and c. 30 chains of length 800

| Algorithm | a | b | c |
|---|---|---|---|
| Gelman-Rubin statistic | 1.06 | 1.00 | 1.00 |
| Gelman-Rubin upper CI | 1.09 | 1.00 | 1.00 |
| ESS | 460 | 18152 | 20277 |
| Time (seconds) | 0.13 | 0.69 | 6.67 |
| ESS per second | 3604 | 26347 | 3041 |

**Part e.**

For our methodology, we will need the CLT to hold. Technically this is a requirement for evaluating ESS, so it is worth confirming for each of our algorithms. Therefore, we will prove the CLT holds for Algorithms $a$,$b$,$c$. We could show this through showing 2 finite moments, but we will demonstrate it through published proofs.

We will prove each chain is geometrically ergodic, which implies the CLT holds as per Jones (2004). For Algorithm $a$, we can recognise that out of the three modes, the heaviest tail is exponential and Jarner and Hansen (2000) proves that as long as $f$ is strictly positive for all $\mathbb{R}$ and has at most exponentially heavy tails, then the RWMH will be geometrically ergodic.

For Algorithm $b$, we turn to Mengersen and Tweedie (1996) who state that as long as $\sup(f(x)/g(x) < \infty$ for a MH algorithm, uniform ergodicity holds which is a form of geometric ergodicity. In our case, we only have one tail to consider, and as the distributions are almost identical it is clear this holds. Inputting large values of $x$ into $f(x), g(x)$ actually shows $g(x)$ is slightly heavier tailed, as desired.

Since Algorithm $c$ tempers $f$, this result is trivial since we have already proved it for when $T = 1$, and as temperature increases $f$ becomes only lighter tailed.

Since we have proved geometric ergodicity, the CLT follows for each of these algorithms, and we can carry this result forwards.

At first glance, we could approach this problem by attempting to calculate the 95% CI for a Monte-Carlo estimate can be derived as follows for iid samples:

$$\left[\hat{I}_n + \frac{1}{\sqrt{n}} S z_{0.025}, \hat{I}_n + \frac{1}{\sqrt{n}} S z_{0.975}\right]$$

Where $z_\alpha$ is the $\alpha$ quantile of a standard normal distribution, and $S$ is the sample standard deviation, i.e. $S = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(X_i - \hat{I}_n)^2}$. However, since our MCMC samples are not iid, we could replace $n$ with $n'$ where $n'$ is our ESS. This is effective for working out CIs at set values of $n$, but as the calculation on ESS is based on calculating large autocorrelation lags and very computationally expensive, plotting a continuous CI is not feasible with this method.

As we have shown the CLT holds, batch means becomes a viable alternative.

Firstly, we will divide our samples into batches of size $b$, where we set our batch size according to the rule found by Liu, Vats, and Flegal (2019) as $b = \lfloor n^{1/3} \rfloor$. We set $m = \lfloor n/b \rfloor$. We discard any $X_i$ values such that $bm < i \leq n$. The batch means CI is given by $\hat{I}_n \pm t_{1-\delta/2}\frac{s_m(n)}{\sqrt{m}}$, where $t_\alpha$ is the $\alpha$ quantile for a $T_{m-1}$ distribution, and $s_m(n)$ is the sample standard deviation for the means of the batches.

We will run all 3 algorithims with $n = 35000, N = 1$ and starting values $X_0 \sim \text{Uniform}(2, 40)$

Using the segmentation from Figure 5 we can use numerical integration to calculate $k = \frac{1}{\int_2^\infty f(x)\,dx} \approx 0.434$ and use the normalised $f(x)$ to find

$$E[X] = \int_2^\infty x f(x)\, dx \approx 25.18$$

We have to use segmentation in both cases, otherwise R's numerical integration misses the second mode.

Comparing this to the batch means CI in Figure 12 it is clear that all 3 algorithms do a reasonable job at estimating $E(X)$, with Algorithms $b$,$c$ outperforming Algorithm $a$ with the width of their CIs. Table 3 confirms that our algorithms are effective at estimating $E(X)$.

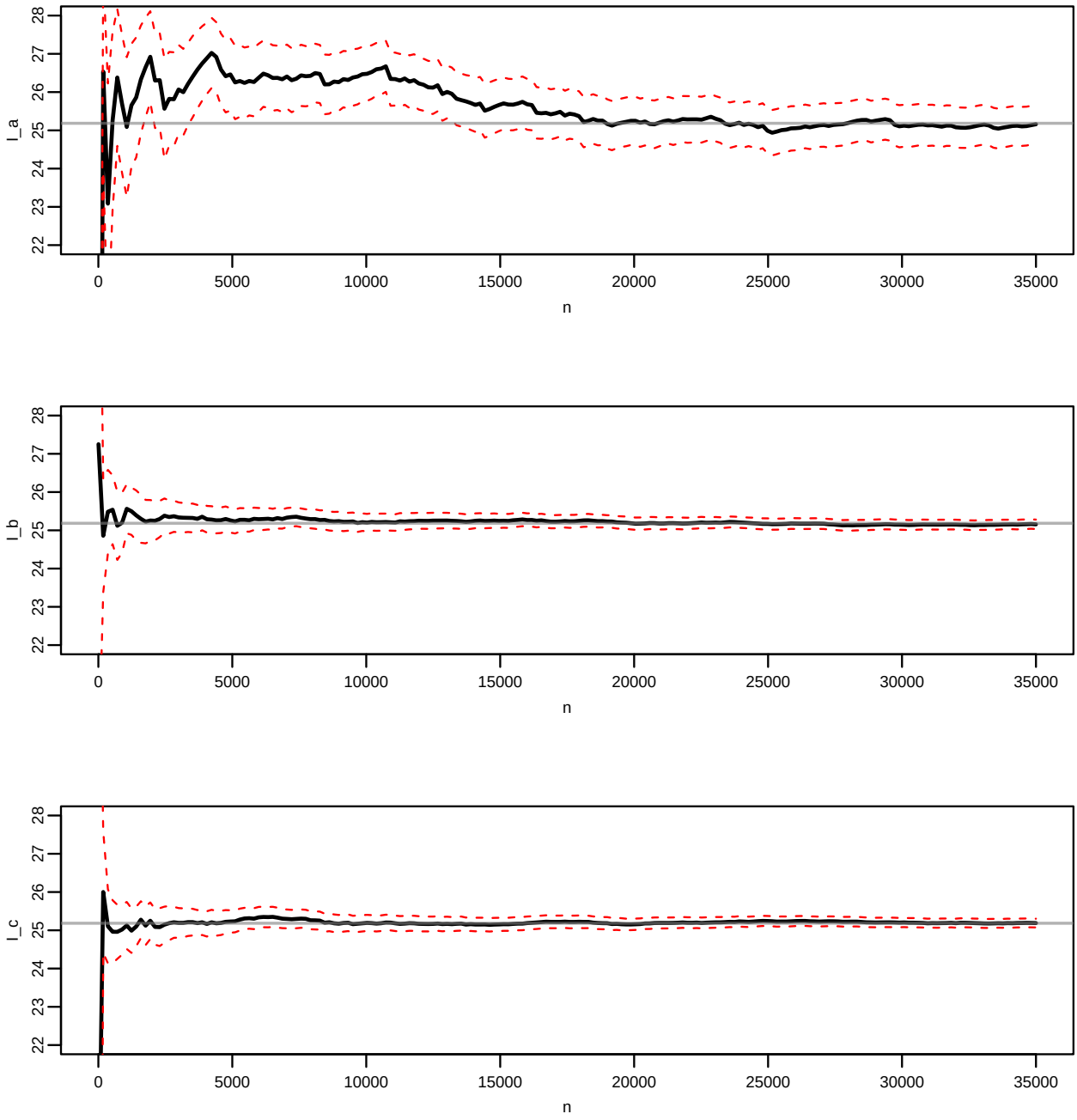


Figure 12: MC estimates from Algorithms a,b,c with upper and lower CI. Horizontal line shows true expectation value.

Table 3: MC estimates and batch means C.I.s for Algorithms a, b, c. True value is 25.18.

| Algorithm | Lower bound | MC estimate | Upper bound | Width |
|---|---|---|---|---|
| a | 24.656 | 25.160 | 25.664 | 1.007 |
| b | 25.033 | 25.157 | 25.280 | 0.247 |
| c | 25.075 | 25.188 | 25.302 | 0.228 |

**Additional notes for Q1.**

From the inference, it seems Algorithm $b$ is the strongest choice, but we must make some comments about this. Firstly, it is absolutely possible that Algorithm $c$ was slightly inefficient due to how the function was setup, but this almost certainly doesn't account for all of the discrepancy, this is down to having 7x the number of chains running. Perhaps simulated tempering could have proved more efficient.

No acf plots were used. This was largely down to ESS proving largely the same information. If it were ever not stated, assume that $X_1 \sim \text{Uniform}(2, 40)$ for any of these algorithms, as this was always the case.

For parallel tempering, I am unsure if duplicate swapping was an issue. By that I mean if a point is swapped and it iterates to the next chain and it gets swapped again, which I understand isn't valid. I think this may be a possibility in my algorithm, though I got somewhat lost in the complexity. Also the pseudocode for this algorithm does not allow for a $g$ which is conditional on $X_t$, as in the RWMH case. The code version was modular enough to allow for this.

For part (e) we have compared the estimates at CIs over $n$ values, but it could be argued that it would be far fairer to do this based on computational cost, as this is a better assessment of how efficient an algorithm is.

# 2 Question 2

**Part a.**

Firstly, we sample $n_{\text{train}} = \lfloor 0.7 \times n \rfloor$ values from $\{1, \ldots, n\}$ without replacement, where $n$ is the number of observations in the `Boston` dataset. We then extract the `mdv`, `lstat`, `rm` values from `Boston` whose indices align with the sampled values and denote these as the training set, and those which do not have sampled indices (i.e `mdv`',`lstat`',`rm`') will be the test set with length $n_{\text{test}} = n - n_{\text{train}}$. More formally:

$$X_{\text{train}} = \begin{pmatrix} 1 & \texttt{lstat}_1 & \texttt{rm}_1 \\ 1 & \texttt{lstat}_2 & \texttt{rm}_2 \\ \vdots & \vdots & \vdots \\ 1 & \texttt{lstat}_{n_{\text{train}}} & \texttt{rm}_{n_{\text{train}}} \end{pmatrix}, \qquad Y_{\text{train}} = \begin{pmatrix} \texttt{medv}_1 \\ \texttt{medv}_2 \\ \vdots \\ \texttt{medv}_{n_{\text{train}}} \end{pmatrix}$$

$$X_{\text{test}} = \begin{pmatrix} 1 & \texttt{lstat}'_1 & \texttt{rm}'_1 \\ 1 & \texttt{lstat}'_2 & \texttt{rm}'_2 \\ \vdots & \vdots & \vdots \\ 1 & \texttt{lstat}'_{n_{\text{test}}} & \texttt{rm}'_{n_{\text{test}}} \end{pmatrix}, \qquad Y_{\text{test}} = \begin{pmatrix} \texttt{medv}'_1 \\ \texttt{medv}'_2 \\ \vdots \\ \texttt{medv}'_{n_{\text{test}}} \end{pmatrix}$$

$$\beta_{\text{init}} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

$$f_\tau(x) = \beta_{0,\tau} + \beta_{1,\tau}\, x_1 + \beta_{2,\tau}\, x_2.$$

$$\ell_\tau(y,q) = \begin{cases} \tau(y-q), & y > q, \\ (\tau-1)(y-q), & y \le q. \end{cases}$$

$$R_\tau = \frac{1}{n_{\text{train}}} \sum_{i=1}^{n\text{train}} \ell_\tau(Y_i, f_\tau(X_i;\beta))$$

$$\tau \in \{0.1,\ 0.5,\ 0.9\}.$$

**QR Algorithm**

**Input:**

1. Model $f_\tau$
2. $\tau = (\tau_1, \ldots, \tau_m)$
3. Design matrix and response, split into test and train data $\{X_{\text{train}}, Y_{\text{train}}\}, \{X_{\text{test}}, Y_{\text{test}}\}$
4. Loss function $\ell_\tau$
5. Risk function $R_\tau$
6. Initial values $\beta_{\text{init}} = (\beta_0, \ldots, \beta_j)$. These will be used for each $\tau$.

- For each $\tau \in (\tau_1, \ldots, \tau_m)$:
  1. Use R's built-in `optim` (BFGS) function to optimise $R_\tau$ over $\beta$ given $X_{\text{train}}, Y_{\text{train}}, \tau, \beta_{\text{init}}$. Label the output as $\hat{\beta}_\tau$.
  2. Set $Q_\tau = X_{\text{test}}\hat{\beta}_\tau$ and calculate empirical loss as:
  $$\mu_\tau = \frac{1}{n_{\text{test}}} \sum_{i=1}^{n_{\text{test}}} \ell_\tau(y_i,\ q_i),$$
  where $y_i$ and $q_i$ denote the $i$th components of $Y_{\text{test}}$ and $Q_\tau$, respectively.
  3. Calculate the coverage as:
  $$\widehat{C}_\tau = \frac{1}{n_{\text{test}}} \sum_{i=1}^{n_{\text{test}}} \mathbf{1}\{y_i \le q_i\},$$
  where $n_{\text{test}}$ is the number of rows in $X_{\text{test}}$.

**Output:**

$$\widehat{C} = (\widehat{C}_{\tau_1}, \ldots, \widehat{C}_{\tau_m}), \quad \hat{\beta} = (\hat{\beta}_{\tau_1}, \ldots, \hat{\beta}_{\tau_m}), \quad \mu = (\mu_{\tau_1}, \ldots, \mu_{\tau_m})$$

Inputting our values into the algorithm we get the output seen in Table 4. We can observe that the $\widehat{C}$ values are within a reasonable tolerance of the corresponding $\tau$ values.

Table 4: QR output for the linear model

| tau | 0.1 | 0.5 | 0.9 |
|---|---|---|---|
| beta_0 | 0.6150 | -2.1678 | 0.7327 |
| beta_1 | -0.7527 | -0.6356 | -0.6247 |
| beta_2 | 4.1966 | 5.0373 | 5.9912 |
| empirical_loss | 0.6964 | 1.8500 | 1.0506 |
| empirical_coverage | 0.1250 | 0.4737 | 0.9211 |

**Part b.**

We alter the following values (any not mentioned stay the same):

$$X_{\text{train}} = \begin{pmatrix} 1 & \texttt{lstat}_1 & \texttt{rm}_1 & \texttt{lstat}_1^2 & \texttt{rm}_1^2 & \texttt{lstat}_1\,\texttt{rm}_1 \\ 1 & \texttt{lstat}_2 & \texttt{rm}_2 & \texttt{lstat}_2^2 & \texttt{rm}_2^2 & \texttt{lstat}_2\,\texttt{rm}_2 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & \texttt{lstat}_{n_{\text{train}}} & \texttt{rm}_{n_{\text{train}}} & \texttt{lstat}_{n_{\text{train}}}^2 & \texttt{rm}_{n_{\text{train}}}^2 & \texttt{lstat}_{n_{\text{train}}}\,\texttt{rm}_{n_{\text{train}}} \end{pmatrix}.$$

$$X_{\text{test}} = \begin{pmatrix} 1 & \texttt{lstat}'_1 & \texttt{rm}'_1 & \texttt{lstat}'^2_1 & \texttt{rm}'^2_1 & \texttt{lstat}'_1\,\texttt{rm}'_1 \\ 1 & \texttt{lstat}'_2 & \texttt{rm}'_2 & \texttt{lstat}'^2_2 & \texttt{rm}'^2_2 & \texttt{lstat}'_2\,\texttt{rm}'_2 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & \texttt{lstat}'_{n_{\text{test}}} & \texttt{rm}'_{n_{\text{test}}} & \texttt{lstat}'^2_{n_{\text{test}}} & \texttt{rm}'^2_{n_{\text{test}}} & \texttt{lstat}'_{n_{\text{test}}}\,\texttt{rm}'_{n_{\text{test}}} \end{pmatrix}.$$

$$\beta_{\text{init}} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

$$f_\tau(x) = \beta_{0,\tau} + \beta_{1,\tau}x_1 + \beta_{2,\tau}x_2 + \beta_{3,\tau}x_1^2 + \beta_{4,\tau}x_2^2 + \beta_{5,\tau}x_1x_2.$$

Applying this to the previously defined methodology, we can observe the output in Table 5. Interestingly, though for $\tau = 0.9$ the coverage has improved, for $\tau = \{0.1, 0.5\}$ the coverage has worsened. Perhaps a size of overfitting, we will explore this in more detail later.

Table 5: QR output for the quadratic and interaction model

| tau | 0.1 | 0.5 | 0.9 |
|---|---|---|---|
| beta_0 | 0.1167 | 0.0972 | 0.3493 |
| beta_1 | 0.8892 | 0.3359 | 0.6687 |
| beta_2 | 0.3925 | 0.1723 | 1.0683 |
| beta_3 | 0.0061 | 0.0181 | 0.0182 |
| beta_4 | 0.6257 | 0.7651 | 0.7395 |
| beta_5 | -0.2847 | -0.2320 | -0.2839 |
| empirical_loss | 0.6633 | 1.7348 | 0.7925 |
| empirical_coverage | 0.1316 | 0.4605 | 0.9079 |

**Part c.**

**Algorithm for optimisation of lambda**

**Input:**

1. Model $f_\tau$
2. $\tau$
3. Design matrix and response, split into test and train data $\{X_{\text{train}}, Y_{\text{train}}\}, \{X_{\text{test}}, Y_{\text{test}}\}$
4. Loss function $\ell_\tau$
5. Risk function $R_\tau$
6. Initial values $\beta_{\text{init}} = (\beta_0, \ldots, \beta_j)$
7. Range of $\lambda_{\text{init}}$ values to consider
8. $k$ number of folds

- Sample $n_{\text{train}}$ times from $\{1, \ldots, k\}$ with replacement as $\xi = (\xi_1, \ldots, \xi_{n_{\text{train}}})$.
- Take 30 equally spaced values across the range of $\lambda_{\text{init}}$ as $\{\lambda_1, \ldots, \lambda_{30}\}$.
- For each $\lambda \in \{\lambda_1, \ldots, \lambda_{30}\}$:
- Define $w$ as an empty vector with length $k$.
  - For each $j \in \{1, \ldots, k\}$:
    - Set validation set to all $X_i, Y_i$ where $\xi_i = j$ and training set where $\xi_i \neq j$.
    - Using R's `optim` (BFGS), optimise $R_\tau$ over $\beta$ given inputs. Label output $\hat{\beta}_\tau$.
    - Set $w_j = R_\tau(\hat{\beta}_\tau, X_{\text{val}}, Y_{\text{val}}, \lambda)$.
  - Define $W_\lambda = \frac{1}{k} \sum_{j=1}^{k} w_j$.

**Output:**

$$\lambda^* = \underset{\lambda \in \{\lambda_1, \ldots, \lambda_{30}\}}{\arg\min} \; W_\lambda.$$

Our first consideration is the number of folds to perform for k-fold validation. Kohavi (1995) found that in general $k = 10$ is a reliable and often optimal choice ($\sqrt{n}$ folds was mentioned in the lecture as appearing in some papers, but I was unable to confirm this).

The only value we will change from those defined in (b) is :

$$R_\tau = \frac{1}{n_{\text{train}}} \sum_{i=1}^{n_{\text{train}}} \left[ \ell_\tau(Y_i, f_\tau(X_i; \beta)) + \lambda \sum_{p=1}^{j} \beta_p \right]$$

and add:

$$k = 10$$

$$\lambda_{\text{init}} = [10^{-4}, 10^{-1}]$$

Note that we have decided to drop $\beta_0$ from the sum, as it is illogical to penalise a model based on its intercept. A good example of this is considering only slightly correlated data whose response is shifted greatly away from the observed data. For this hypothetical, a good model would expect $|\beta_0|$ to be large and the other $|\beta|$ values to be far lower, but ridge regression including $\beta_0$ would incorrectly punish it. In any case, we return to this example.

We find $\lambda_\tau^*$ for each $\tau \in \{0.1, 0.5, 0.9\}$, and run our QR Algorithm defined in (a), inputting the ridge regression risk. The output can be seen in Table 6.

Taking plots for each $\lambda$ within our $\lambda$ optimisation algorithm, we can observe Figures 13, 14, 15.

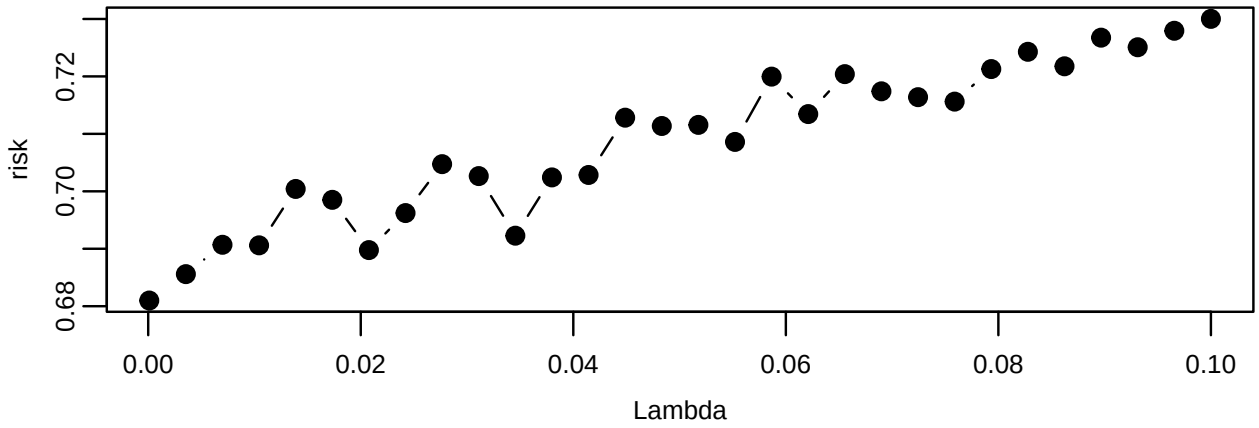


Figure 13: Line plot of cross validation ridge regression risk for tau = 0.1 over different lambda values.

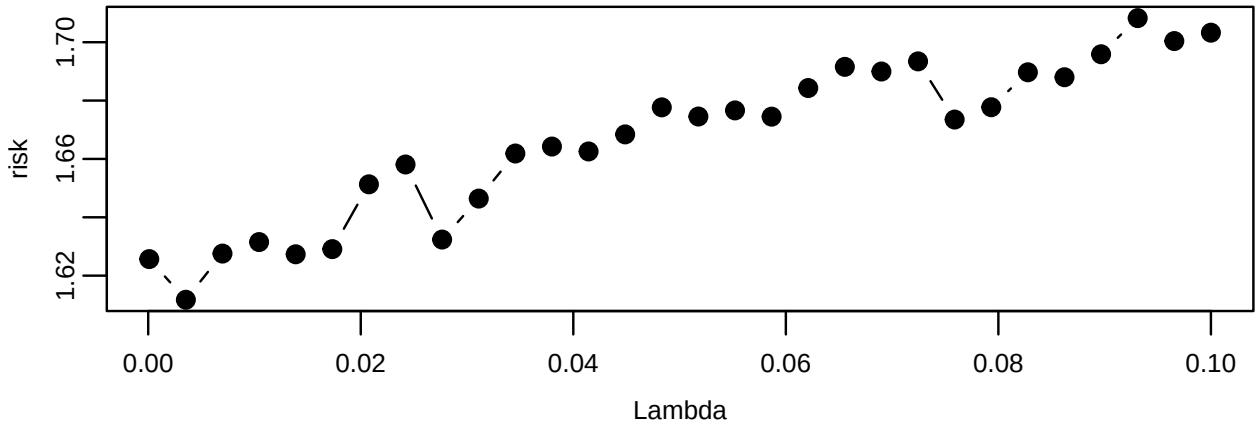


Figure 14: Line plot of cross validation ridge regression risk for tau = 0.5 over different lambda values.

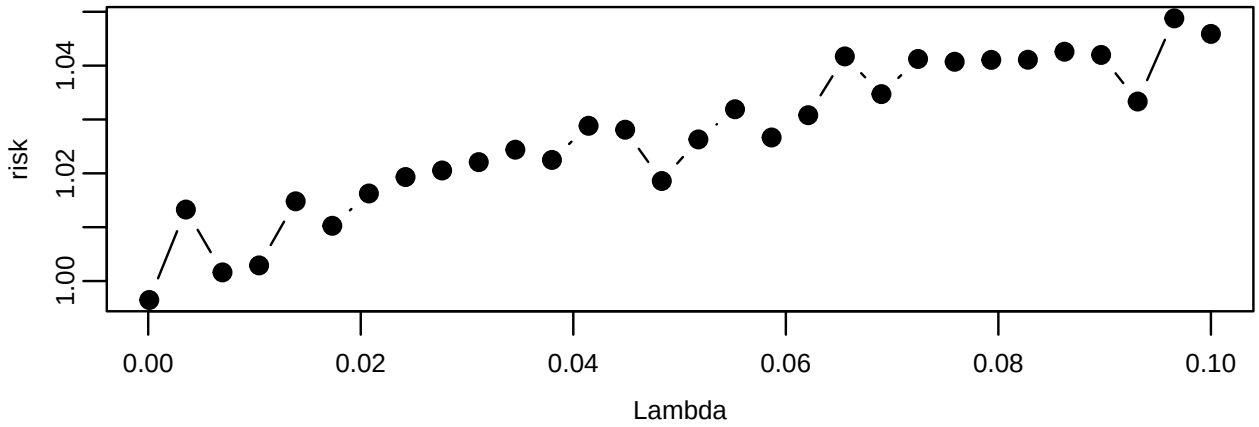


Figure 15: Line plot of cross validation ridge regression risk for tau = 0.9 over different lambda values.

Table 6: QRR output for the quadratic and interaction model

| tau | 0.1 | 0.5 | 0.9 |
|---|---|---|---|
| beta_0 | 0.1266 | 0.2239 | 0.0844 |
| beta_1 | 0.9106 | 1.5118 | 0.3981 |
| beta_2 | 0.3945 | 0.7062 | 0.2782 |
| beta_3 | 0.0087 | 0.0064 | 0.0187 |
| beta_4 | 0.6374 | 0.6771 | 0.8283 |
| beta_5 | -0.3042 | -0.3940 | -0.2194 |
| lambda | 0.0001 | 0.0035 | 0.0001 |
| empirical_loss | 0.6727 | 1.5015 | 0.8321 |
| empirical_coverage | 0.1316 | 0.4934 | 0.8750 |

**Part d.**

Using the results from Tables 4, 5 and 6:

Comparing the $\hat{\beta}$ magnitudes, we can see when we introduce the quadratic elements, the $\hat{\beta}_4, \hat{\beta}_5$ coefficients are non-negligible, but the $\hat{\beta}_3$ coefficients for all quantiles with and without ridge regression seem very close to zero. What this likely means is `lstat`$^2$ has little to no quadratic relationship with `medv`. Once we add in ridge regression, we don't see a clear impact as the $\lambda^*$ values found were close to zero, resulting in little change in the levels of magnitude of the coefficients. However, this may not be the full picture, as we will explore.

We can now move to test loss, where we see a reduction in loss when comparing the linear model with the quadratic for all $\tau$ values. This reduction in test loss tells us it is more than likely that there is some quadratic or interaction relationship between the response and observations. The ridge regression shows significant decrease in loss for $\tau = 0.5$, and a very slight increase for the tails. Perhaps the $\lambda^*_{0.5}$ was significant enough to punish some overfitting, as the quadratic coefficients did decrease in magnitude (though all the others increased). Typically, we would expect this increased performance in the median and decrease in the tails to be something of an overly smoothed function, but we don't quite have enough information to say this is the case here.

On the surface there isn't strong evidence for overfitting. The covariate-data ratio is very high, which discourages overfitting and when we implemented two anti-overfitting methods (ridge regression and cross validation), we saw only little improvement when compared to the regular quadratic and interaction model. While we can't insepct the fitted function shapes due to dimensionality, we can observe the rate of quantile crossing (will be discussed in more detail in the next part) in Table 7, where we can observe some of the smoothing effect of the ridge regression. In the linear model there is no quantile crossing, but with the quadratic and interaction terms there becomes enough variance for the quantile models to cross. And most interestingly, this is no longer the case when the ridge regression and cross validation is implemented. Perhaps a sign of overfitting in the end.

Finally, we can consider empirical coverage. Perhaps another sign that the original quadratic-interaction model may have been overfitted, is its performance in this category. It is the worst performer out of the three models in coverage for the median, whilst being the best performer in tail coverage. The reason an overfitted model may perform this way is in the middle of the data points there is simply more data to overfit on. In the tails, it may be that the few extreme values are actually better modelled by this higher variance model. This is backed up by the fact that the original quadratic-interaction model has the lowest empirical loss for $\tau = \{0.1, 0.9\}$.

Concluding the comparisons, we can infer that the original linear model provided a smooth, but biased fit for our covariates, shown by the high level of empirical loss. The quadratic-interaction model improved on this, but was prone to overfitting, particularly in the denser areas of data. The ridge regression and cross validation did very little to change the modelling of extreme values, with the small penalty of $\lambda^* = 0.1$ applied, but

did discourage larger quadratic coefficient values in the center of the distribution which resulted in a large reduction in overall empirical loss and great improvement in coverage for the median.

Table 7: Rate of inter-quartile crossing between the linear, quadratic-interaction and quadratic-interaction with ridge regression and cross validation

| | Q01_05 | Q05_09 | Q01_09 |
|---|---|---|---|
| linear | 0.0 | 0 | 0 |
| quad | 0.1 | 0 | 0 |
| quad_ridge | 0.0 | 0 | 0 |

**Part e.**

No, quantile prediction models we have fitted will not satisfy this relationship by default. Each quantile $\tau$ is fitted independently of the other, the only time we could find a model which will always satisfy this no matter the data size would be a model where we can perfectly explain all variance of the response variables, i.e. no noise and perfect prediction.

There are existing methods for working around this issue, with a popular R package for this being `quantreg`; the methodology behind it described in Muggeo et al. (2013) forms a lot of the basis for the the proposed methodology. We can adjust the risk function similarly to how we did for ridge regression, but this time with a penalty for every crossing, scaled by the severity of the infraction. We define the following risk functions:

$$\widehat{R}^{\text{cross+}}_{\lambda,\tau}(\beta) = \frac{1}{n_{\text{train}}} \sum_{i=1}^{n_{\text{train}}} \ell_\tau\big(Y_i, f_\tau(X_i;\beta_\tau)\big) + \lambda \sum_{i=1}^{n_{\text{train}}} \mathbf{1}\Big\{ f_\tau(X_i;\beta_\tau) < f_{\tau^-}(X_i;\beta_{\tau^-}) \Big\} \Big[ f_{\tau^-}(X_i;\beta_{\tau^-}) - f_\tau(X_i;\beta_\tau) \Big]$$

$$\widehat{R}^{\text{cross-}}_{\lambda,\tau}(\beta) = \frac{1}{n_{\text{train}}} \sum_{i=1}^{n_{\text{train}}} \ell_\tau\big(Y_i, f_\tau(X_i;\beta_\tau)\big) + \lambda \sum_{i=1}^{n_{\text{train}}} \mathbf{1}\Big\{ f_\tau(X_i;\beta_\tau) > f_{\tau^+}(X_i;\beta_{\tau^+}) \Big\} \Big[ f_\tau(X_i;\beta_\tau) - f_{\tau^+}(X_i;\beta_{\tau^+}) \Big]$$

The portion to the left chosen here is the previously used pre-ridge regression risk. In practice, we can replace this with whatever we like, it is the right side of the addition that is the novelty.

Given a pre-defined general risk $\widehat{R}_\tau(\beta)$, the pre-defined cross risks $\widehat{R}^{\text{cross-}}_{\lambda,\tau}(\beta), \widehat{R}^{\text{cross+}}_{\lambda,\tau}(\beta)$, assuming you have all the information needed for quantile regression, given a penalty term $\lambda$ and given you have a finite number of ordered quantiles of interest $\tau = \{\tau_1, \ldots, \tau_\delta = 0.5, \ldots, \tau_\nu\}$, where $\delta = (\nu+1)/2$:

- Minimise $\widehat{R}_{\tau_\delta}(\beta)$.
- For every $\tau_i \in \{\tau_{\delta+1}, \ldots, \tau_\nu\}$, iterating through the lowest to the highest, minimise $\widehat{R}^{\text{cross+}}_{\lambda,\tau_i,\tau_{i-1}}(\beta)$. Continue until $\tau_\nu$ is reached.
- For every $\tau_i \in \{\tau_1, \ldots, \tau_{\delta-1}\}$, iterating through highest to the lowest, minimise $\widehat{R}^{\text{cross-}}_{\lambda,\tau_i,\tau_{i+1}}(\beta)$. Continue until $\tau_1$ is reached.
- Return optimised $\beta_\tau$ values.

This framework has assumed that we both are interested in the median and that our quantiles of interest are symmetric around the median. If not we could replace the starting value of 0.5 with the quantile corresponding to the "median" index of the number of quantiles.

Note that as $\lambda \to \infty$, then this becomes a strict constraint, i.e. for $\tau^+ > \tau$, $\beta_{\tau^+} > \beta_\tau$ will strictly hold.

Note also we could simply remove the indicator's coefficient in the summation and keep just the indicator function if we cared solely about the number of crossings and not their severity.

With these constraints comes a smoother function, but of course this is a bias-variance trade-off, this decrease in variance will come with an increase in empirical loss, after all we are no longer simply optimising the loss, we are partitioning our admissible $\beta$ values. If the model is flexible enough and bias is not drastically increased, then coverage will often improve, but this is not an optimiser of coverage, rather an optimiser of smoothness, and as $\lambda \to \infty$ a minimiser of variance. To summarise, just because they're ordered doesn't mean they're perfectly calibrated.

**Additional notes for Q2.**

Probably a large place for practical improvement would be adjusting the optimisation algorithms to allow for different initial values of $\beta$ for each $\tau$ value, I just didn't build my functions like that as I thought we had a simple enough data setup.

The code is very messy and follows no discernible pattern as it was done over different days with too little thought for modularity. And apologies for a very messy set of notation on the last bit.

# 3 Supplementary Material

```
#(1) (a)
gelman.diag(mcmc.list(chains))
#> Potential scale reduction factors:
#>
#>      Point est. Upper C.I.
#> [1,]       1.09       1.14
mean(acc)
#> [1] 0.3548155

#(1) (b)
integrate(function(x) g(x), -99, 2)$value
#> [1] 2.408544e-173
```

# 4 Code appendix

```
knitr::opts_chunk$set(
  collapse = TRUE,
  comment = "#>"
)
include_solutions <- TRUE
require(rmarkdown)
require(knitr)
require(kableExtra)
require(coda)
require(MASS)
set.seed(06049216) # College id
logf <- function(x) {
  a1 <- -3*(x-2)
  a2 <- -(x-30)^2
  a3 <- -(x-20)^2/0.01
  m <- max(a1,a2,a3)
  m + log(exp(a1-m) + exp(a2-m) + exp(a3-m))
}
par(mar=c(4, 3.5, 2, 2), cex.lab=0.7, cex.axis=0.7, mgp=c(1.5, 0.5, 0))
f <- function(x) { ifelse(x<2,0,exp(-3*(x-2)) + exp(-(x-30)^2) + exp(-(x-20)^2/0.01)) }
```

```r
x <- seq(0,40,0.001)
plot(x,f(x),type="l")
par(mar=c(4, 3.5, 2, 2), cex.lab=0.7, cex.axis=0.7, mgp=c(1.5, 0.5, 0))
set.seed(06049216) # College id
do.the.rwmh <- function(num.steps, sigma, X0) {
  X <- X0
  samples <- rep(0.0, num.steps)
  for (i in 1:num.steps) {
    samples[i] <- X
    Y <- X + sigma*rnorm(1)
    if (Y < 2) {
      logalpha <- -Inf
    } else {
      logalpha <- logf(Y) - logf(X)
    }
    if (log(runif(1)) <= logalpha) X <- Y
  }
  samples
}
sigmas <- seq(0.1,10,length.out=50)
Rhat <- rep(NA,length(sigmas))
acc <- rep(NA,length(sigmas))

for (j in seq_along(sigmas)) {
  s <- sigmas[j]
  chains <- list()
  a <- 0
  for (i in 1:30) {
    samples <- do.the.rwmh(10000, s, runif(1,2,40))
    a <- a + mean(diff(samples)!=0)
    chains[[i]] <- mcmc(samples[2001:10000])
  }
  Rhat[j] <- gelman.diag(mcmc.list(chains))$psrf[1]
  acc[j] <- a/30
}


plot(sigmas, Rhat, type="l", col="blue", ylab="Rhat", xlab="sigma")

j <- which(Rhat < 1.05)[1]
points(sigmas[j], Rhat[j], pch=16, col="purple")

par(new=TRUE)

plot(sigmas, acc, type="l", col="red", axes=FALSE, xlab="", ylab="")
axis(side=4)
mtext("Acceptance rate", side=4, line=2.5, cex=0.7)

abline(h=0.44, col="darkgreen", lty=2)

i <- which.min(abs(acc - 0.44))
points(sigmas[i], acc[i], pch=16, col="darkgreen")
sigma_1 <- sigmas[i]
```

```r
sigma_2 <- sigmas[j]


par(mar=c(4, 3.5, 2, 2), cex.lab=0.7, cex.axis=0.7, mgp=c(1.5, 0.5, 0))
sigmas <- seq(1,50,length.out=50)
Rhat <- rep(NA,length(sigmas))
ESS <- rep(NA,length(sigmas))

for (j in seq_along(sigmas)) {
  s <- sigmas[j]
  chains <- list()
  for (i in 1:30) {
    samples <- do.the.rwmh(10000, s, runif(1,2,40))
    chains[[i]] <- mcmc(samples[2001:10000])
  }
  Rhat[j] <- gelman.diag(mcmc.list(chains))$psrf[1]
  ESS[j] <- mean(effectiveSize(mcmc.list(chains)))
}

par(mar=c(4, 3.5, 2, 2), cex.lab=0.7, cex.axis=0.7, mgp=c(1.5, 0.5, 0))

plot(sigmas, Rhat, type="l", col="blue", ylab="Rhat", xlab="sigma")



par(new=TRUE)

plot(sigmas, ESS, type="l", col="red", axes=FALSE, xlab="", ylab="")
axis(side=4)
mtext("ESS", side=4, line=2.5, cex=0.7)
i <- which.max(ESS[sigmas > 5])
sigma_opt <- sigmas[sigmas > 5][i]
ess_opt <- ESS[sigmas > 5][i]

points(sigma_opt, ess_opt, pch=16, col="darkgreen")

par(mar=c(4, 3.5, 2, 2), cex.lab=0.7, cex.axis=0.7, mgp=c(1.5, 0.5, 0))
do.rwmh.mix <- function(num.steps, sigma1, sigma2, p, X0) {
  X <- X0
  samples <- rep(0.0, num.steps)
  for (i in 1:num.steps) {
    samples[i] <- X
    if (runif(1) < p) s <- sigma1 else s <- sigma2
    Y <- X + s*rnorm(1)
    if (Y < 2) logalpha <- -Inf else logalpha <- logf(Y) - logf(X)
    if (log(runif(1)) <= logalpha) X <- Y
  }
  samples
}

set.seed(06049216)
chains <- list()
acc <- numeric(30)
```

```r
for (i in 1:30) {
  samps <- do.rwmh.mix(10000, sigma_1, sigma_2, 0.5, runif(1,2,40))
  acc[i] <- mean(diff(samps) != 0)
  chains[[i]] <- mcmc(samps[2001:10000])
}



chain1 <- chains[[1]]
chain2 <- chains[[2]]
chain3 <- chains[[3]]

idx <- seq(1, length(chain1), by = 5)
chain1_t <- chain1[idx]
chain2_t <- chain2[idx]
chain3_t <- chain3[idx]

plot(chain1_t, pch = '.', col = 'green', xlab = 'Iteration', ylab = 'x')
points(chain2_t, pch = '.', col = 'red')
points(chain3_t, pch = '.', col = 'blue')

par(mar=c(4, 3.5, 2, 2), cex.lab=0.7, cex.axis=0.7, mgp=c(1.5, 0.5, 0))
set.seed(06049216) # College id
f <- function(x) { ifelse(x<2,0,exp(-3*(x-2)) + exp(-(x-30)^2) + exp(-(x-20)^2/0.01)) }

x <- seq(0,40,0.001)
plot(x,f(x),type="l")
logf <- function(x) {
  a1 <- -3*(x-2)
  a2 <- -(x-30)^2
  a3 <- -(x-20)^2/0.01
  m <- max(a1,a2,a3)
  m + log(exp(a1-m) + exp(a2-m) + exp(a3-m))
}
#find k REFPLOT
rect(2,0,4.5,max(f(x)),col=rgb(1,0,0,0.2),border=NA)
rect(4.5,0,19.5,max(f(x)),col=rgb(1,1,0,0.4),border=NA)
rect(19.5,0,20.5,max(f(x)),col=rgb(1,0,0,0.2),border=NA)
rect(20.5,0,27,max(f(x)),col=rgb(1,1,0,0.4),border=NA)
rect(27,0,33,max(f(x)),col=rgb(1,0,0,0.2),border=NA)
rect(33,0,40,max(f(x)),col=rgb(1,1,0,0.4),border=NA)

g1 <- integrate(function(x) f(x), 2, 4.5)$value
g1_2 <- integrate(function(x) f(x), 4.5, 19.5)$value
g2 <- integrate(function(x) f(x), 19.5, 20.5)$value
g2_3 <- integrate(function(x) f(x), 20.5, 27)$value
g3 <- integrate(function(x) f(x), 27, 33)$value
g3_i <- integrate(function(x) f(x), 33, 100)$value
total <- g1 + g1_2 + g2 + g2_3 + g3 + g3_i
k<- 1/total

gg <- g1 + g2 + g3
g1_w <- g1 / gg
```

```r
g2_w <- g2 / gg
g3_w<- g3 / gg
set.seed(06049216) # College id


g <- function(x) {
  d1 <- ifelse(x>2,dexp(x-2, rate = 3),0 )
  d2 <- dnorm(x, mean = 20, sd = sqrt(0.01))
  d3 <- dnorm(x, mean = 30, sd = 1)
  g1_w * d1 + g2_w * d2 + g3_w * d3
}



gsample <- function(){
  i <- sample(1:3, 1, prob = c(g1_w,g2_w,g3_w))
  if (i == 1) return(rexp(1,3) + 2)
  if (i == 2) return(rnorm(1,20,0.1) )
  rnorm(1,30,1)
}
mh_indep <- function(n, x0) {
  x <- numeric(n)
  x[1] <- x0
  for (t in 2:n) {
    y <- gsample()
    numerr <- f(y) * g(x[t - 1])
    den <- f(x[t - 1]) * g(y)
    alpha <- min(1, numerr / den)
    if (runif(1) < alpha) {
      x[t] <- y
    } else {
      x[t] <- x[t - 1]
    }
  }
  x
}
sam <- numeric(30 * 8000)
pos <- 1

for (i in 1:30) {
  x0 <- runif(1, 2, 40)
  ch <- mh_indep(10000, x0)
  sam[pos:(pos + 7999)] <- ch[2001:10000]
  pos <- pos + 8000
}

p1 <- mean(sam >= 2 & sam <= 4.5)
p2 <- mean(sam >= 19.5 & sam <= 20.5)
p3 <- mean(sam >= 27 & sam <= 33)

tab <- data.frame(
  Proportion = c(p1, p2, p3),
  w_i = c(g1_w,g2_w,g3_w)
```

```
)

tabl <- kable(tab, digits = 4, col.names = c("Proportion", "$w_i$"),
                  caption = "(\\#tab:tabl) Proportion of samples in each mode in
              240,000 samples of Algorithm b and proportional area
of each mode", format = "latex", escape = F)
kable_styling(tabl, latex_options = "HOLD_position")
par(mar=c(4, 3.5, 2, 2), cex.lab=0.7, cex.axis=0.7, mgp=c(1.5, 0.5, 0))
d <- density(sam)
plot(d$x, d$y, type = "l", lwd = 2, xlab = "x", ylab = "Density")
par(mar=c(4, 3.5, 2, 2), cex.lab=0.7, cex.axis=0.7, mgp=c(1.5, 0.5, 0))
samsplit <- split(sam, rep(1:30, each = 8000))
samsplit_mcmc <- mcmc.list(lapply(samsplit, mcmc))
gelman.plot(mcmc.list(samsplit_mcmc))
par(mar=c(4, 3.5, 2, 2), cex.lab=0.7, cex.axis=0.7, mgp=c(1.5, 0.5, 0))
chain1 <- sam[1:8000]
chain2 <- sam[(8001):(16000)]
chain3 <- sam[(16001):(24000)]

idx <- seq(1, 8000, by = 5)

chain1_t <- chain1[idx]
chain2_t <- chain2[idx]
chain3_t <- chain3[idx]


plot(chain1_t,
     pch = '.',
     col = 'green',
     xlab = 'Iteration',
     ylab = 'x')

points(chain2_t,
       pch = '.',
       col = 'red')

points(chain3_t,
       pch = '.',
       col = 'blue')

par(mar=c(4, 3.5, 2, 2), cex.lab=0.7, cex.axis=0.7, mgp=c(1.5, 0.5, 0))
temps <- c(1, 2, 5, 10, 20, 40 ,70)
xs <- seq(0, 45, length = 1000)

fT <- function(x, T) f(x)^(1/T)

plot(xs, fT(xs, temps[1]), type = "l", lwd = 2,
     ylim = c(0, max(sapply(temps, function(T) fT(xs, T)))),
     xlab = "x", ylab = "f_T(x)")

cols <- rainbow(length(temps))
for (i in seq_along(temps)) {
  lines(xs, fT(xs, temps[i]), lwd = 2, col = cols[i])
```

```
}

legend("topright", legend = paste0("T=", temps), col = cols, lwd = 2)
par(mar=c(4, 3.5, 2, 2), cex.lab=0.7, cex.axis=0.7, mgp=c(1.5, 0.5, 0))
pt_run <- function(n_iter = 10000,
                   temps = c(1, 2, 5, 10, 20, 40, 70),
                   x0 = runif(length(temps), 2, 40)) {
  n_ch <- length(temps)
  x <- matrix(0, nrow = n_iter, ncol = n_ch)
  x[1, ] <- x0
  for (t in 2:n_iter) {

    for (j in 1:n_ch) {
      y <- gsample()
      top <- f(y)^(1/temps[j]) * g(x[t - 1, j])
      botm <- f(x[t - 1, j])^(1/temps[j]) * g(y)
      alpha <- min(1, top / botm)
      if (runif(1) < alpha) {
        x[t, j] <- y
      } else {
        x[t, j] <- x[t - 1, j]
      }
    }

    for (j in 1:n_ch) {

      if (j == 1) k <- 2
      else if (j == n_ch) k <- n_ch - 1 else k <- sample(c(j - 1, j + 1), 1)

      xtj <- x[t, j]
      xtk <- x[t, k]

      top <- f(xtk)^(1/temps[j] - 1/temps[k])
      botm <- f(xtj)^(1/temps[j] - 1/temps[k])
      a_swap <- min(1, top/botm)

      if (runif(1) < a_swap) {
        x_tmp <- x[t, j]
        x[t, j] <- x[t, k]
        x[t, k] <- x_tmp
      }
    }
  }

  x[, 1]
}

pt_multiple_runs <- function(n_runs = 30,
                             n_iter = 10000,
                             temps = c(1, 2, 5, 10, 20, 40, 70)) {
  out <- vector("list", n_runs)
  for (r in 1:n_runs) {
    x0 <- runif(length(temps), 2, 40)
```

```
    xr <- pt_run(n_iter = n_iter, temps = temps, x0 = x0)
    out[[r]] <- xr[(0.2*n_iter + 1):n_iter]
  }
  unlist(out)
}

res <- pt_multiple_runs(
  n_runs = 30,
  n_iter = 10000,
  temps = c(1, 2, 5, 10, 20, 40, 70)
)

chain1r <- res[1:8000]
chain2r <- res[(8001):(16000)]
chain3r <- res[(16001):(24000)]

idx <- seq(1, 8000, by = 5)

chain1r_t <- chain1r[idx]
chain2r_t <- chain2r[idx]
chain3r_t <- chain3r[idx]


plot(chain1r_t,
     pch = '.',
     col = 'green',
     xlab = 'Iteration',
     ylab = 'x')

points(chain2r_t,
       pch = '.',
       col = 'red')

points(chain3r_t,
       pch = '.',
       col = 'blue')


par(mar=c(4, 3.5, 2, 2), cex.lab=0.7, cex.axis=0.7, mgp=c(1.5, 0.5, 0))
dd <- density(res)
plot(dd$x, dd$y, type = "l", lwd = 2, xlab = "x", ylab = "Empirical sample density")
set.seed(06049216) # College id


start <- Sys.time()
alga <- numeric(30 * 800)
pos <- 1
for (i in 1:30) { result <- do.the.rwmh(1000, 28, runif(1, 2, 40))
alga[pos:(pos + 799)] <- result[201:1000]
pos <- pos + 800}
end <- Sys.time()
timea <- as.numeric(end - start)
```

```r
start <- Sys.time()
algb <- numeric(30 * 800)
pos <- 1

for (i in 1:30) {
  x0 <- runif(1, 2, 40)
  ch <- mh_indep(1000, x0)
  algb[pos:(pos + 799)] <- ch[201:1000]
  pos <- pos + 800
}
end <- Sys.time()
timeb <- as.numeric(end - start)



start <- Sys.time()
algc <- pt_multiple_runs(
  n_runs = 30,
  n_iter = 1000,
  temps = c(1, 2, 5, 10, 20, 40, 70)
)
end <- Sys.time()
timec <- as.numeric(end - start)



essa <- effectiveSize(mcmc(alga))
essb <- effectiveSize(mcmc(algb))
essc <- effectiveSize(mcmc(algc))

mata <- matrix(alga, nrow = 800, ncol =30 , byrow = FALSE)

chains_lista <- lapply(1:30, function(j) mcmc(mata[, j]))


mcmc_chainsa <- mcmc.list(chains_lista)

gelmana <- as.numeric(unlist(gelman.diag(mcmc_chainsa)[1]))
gelmanaUpper <- as.numeric(unlist(gelman.diag(mcmc_chainsa)[2]))


matb <- matrix(algb, nrow = 800, ncol = 30, byrow = FALSE)

chains_listb <- lapply(1:30, function(j) mcmc(matb[, j]))

mcmc_chainsb <- mcmc.list(chains_listb)

gelmanb <- as.numeric(unlist(gelman.diag(mcmc_chainsb)[1]))
gelmanbUpper <- as.numeric(unlist(gelman.diag(mcmc_chainsb)[2]))


matc <- matrix(algc, nrow = 800, ncol =30, byrow = FALSE)
```

```r
chains_listc <- lapply(1:30, function(j) mcmc(matc[, j]))

mcmc_chainsc <- mcmc.list(chains_listc)

gelmanc <- as.numeric(unlist(gelman.diag(mcmc_chainsc)[1]))
gelmancUpper <- as.numeric(unlist(gelman.diag(mcmc_chainsc)[2]))

tab <- data.frame(
  Algorithm= c("Gelman-Rubin statistic","Gelman-Rubin upper CI",
    "ESS",
    "Time (seconds)",
    "ESS per second"
  ),
  a = c(sprintf("%.2f", gelmana),
    sprintf("%.2f", gelmanaUpper),
    sprintf("%.0f", essa),
    sprintf("%.2f", timea),
    sprintf("%.0f", essa / timea)),
  b = c(sprintf("%.2f", gelmanb),
    sprintf("%.2f", gelmanbUpper),
    sprintf("%.0f", essb),
    sprintf("%.2f", timeb),
    sprintf("%.0f", essb / timeb)),
  c = c(sprintf("%.2f", gelmanc),
    sprintf("%.2f", gelmancUpper),
    sprintf("%.0f", essc),
    sprintf("%.2f", timec),
    sprintf("%.0f", essc / timec))
)

tabl_alg <- kable(
  tab,
  col.names = c("Algorithm", "a", "b", "c"),
  caption = "Convergence and efficiency summaries for Algorithms a, b, and c.
  30 chains of length 800",
  format ="latex",
  booktabs= TRUE,
  align = c('l', 'r', 'r', 'r'),
  escape = FALSE)

kable_styling(tabl_alg, latex_options = "HOLD_position")

f_norm <- function(x) { ifelse(x<2,0,k*(exp(-3*(x-2)) +
exp(-(x-30)^2) + exp(-(x-20)^2/0.01))) }
segment_1 <- integrate(function(x) f_norm(x) * x, 2,   4.5)$value
segment_2 <- integrate(function(x) f_norm(x) * x, 4.5, 19.5)$value
segment_3 <- integrate(function(x) f_norm(x) * x, 19.5, 20.5)$value
segment_4 <- integrate(function(x) f_norm(x) * x, 20.5, 27)$value
segment_5 <- integrate(function(x) f_norm(x) * x, 27,  33)$value
segment_6 <- integrate(function(x) f_norm(x) * x, 33,  100)$value
tot_l <- segment_1 + segment_2 + segment_3 + segment_4 + segment_5 + segment_6
# tot_l # this is E(X)
par(mar=c(4, 3.5, 2, 2), cex.lab=0.7, cex.axis=0.7, mgp=c(1.5, 0.5, 0), mfrow =c(3,1))
```

```r
set.seed(06049216) #College id

longa <- do.the.rwmh(35000, 28, runif(1, 2, 40))
longb <- mh_indep(35000, runif(1, 2, 40))
longc <- pt_run(35000)


batch_means_ci <- function(x) {
  n <- length(x)
  b <- floor(n^(1/3))
  m <- floor(n / b)
  y <- x[1:(m * b)]
  batchmeans <- colMeans(matrix(y, nrow = b, ncol = m))
  tval <- qt(0.975, df = m - 1)
  c( mean(x) - qt(0.975, df = m - 1) *
  (sd(batchmeans)/sqrt(m)), mean(x), mean(x) + qt(0.975, df = m - 1) *
  (sd(batchmeans)/sqrt(m)))
}

n_dy <-  round(seq(5,35000, length.out = 200))
dynamic_a_ci <- sapply(n_dy, function(k) batch_means_ci(longa[1:k]))
dynamic_b_ci <- sapply(n_dy, function(k) batch_means_ci(longb[1:k]))
dynamic_c_ci <- sapply(n_dy, function(k) batch_means_ci(longc[1:k]))

plot(n_dy, dynamic_a_ci[2,], type = "l", lwd = 2,
     xlab = "n",
     ylab = "I_a",
     ylim = c(22,28))

lines(n_dy, dynamic_a_ci[1,], lty = 2, col = "red")
lines(n_dy, dynamic_a_ci[3,], lty = 2, col = "red")
abline(h = tot_l, lwd = 1.5, col = rgb(0.5, 0.5, 0.5, 0.6))

plot(n_dy, dynamic_b_ci[2,], type = "l", lwd = 2,
     xlab = "n",
     ylab = "I_b",
     ylim = c(22,28))

lines(n_dy, dynamic_b_ci[1,], lty = 2, col = "red")
lines(n_dy, dynamic_b_ci[3,], lty = 2, col = "red")
abline(h = tot_l, lwd = 1.5, col = rgb(0.5, 0.5, 0.5, 0.6))

plot(n_dy, dynamic_c_ci[2,], type = "l", lwd = 2,
     xlab = "n",
     ylab = "I_c",
  ylim = c(22,28))

lines(n_dy, dynamic_c_ci[1,], lty = 2, col ="red")
lines(n_dy, dynamic_c_ci[3,], lty = 2, col = "red")
abline(h = tot_l, lwd = 1.5, col = rgb(0.5, 0.5, 0.5, 0.6))


I_a_lower <- dynamic_a_ci[1,200]
```

```r
I_a <- dynamic_a_ci[2,200]
I_a_upper <- dynamic_a_ci[3,200]

I_b_lower <- dynamic_b_ci[1,200]
I_b <- dynamic_b_ci[2,200]
I_b_upper <- dynamic_b_ci[3,200]

I_c_lower <- dynamic_c_ci[1,200]
I_c <- dynamic_c_ci[2,200]
I_c_upper <- dynamic_c_ci[3,200]

tab_simple <- data.frame(
  Algorithm = c("a", "b", "c"),
  Lower.bound = c(
    sprintf("%.3f", I_a_lower),
    sprintf("%.3f", I_b_lower),
    sprintf("%.3f", I_c_lower)),
  MC.estimate = c(
    sprintf("%.3f", I_a),
    sprintf("%.3f", I_b),
    sprintf("%.3f", I_c)),
  Upper.bound = c(
    sprintf("%.3f", I_a_upper),
    sprintf("%.3f", I_b_upper),
    sprintf("%.3f", I_c_upper)),
  Width = c(
    sprintf("%.3f", I_a_upper - I_a_lower),
    sprintf("%.3f", I_b_upper - I_b_lower),
    sprintf("%.3f", I_c_upper - I_c_lower))
)

tabl_simple <- kable(
  tab_simple,
  col.names = c("Algorithm", "Lower bound", "MC estimate", "Upper bound", "Width"),
  caption = "MC estimates and batch means C.I.s for Algorithms a,
  b, c. True value is 25.18.",
  format = "latex",
  booktabs = TRUE,
  align = c("l", "r", "r", "r", "r"),
  escape = FALSE
)

kable_styling(tabl_simple, latex_options = "HOLD_position")


set.seed(06049216)

dat <- Boston[, c("medv", "lstat", "rm")]
train <- sample(1:nrow(dat), 0.7 * nrow(dat))
dat_train <- dat[train, ]
dat_test  <- dat[-train, ]

loss <- function(y, q, tau){
```

```r
  outpt <- numeric(length(y))
  outpt[y > q]  <-  tau * (y - q)[y > q]
  outpt[y <= q] <- (tau - 1) * (y - q)[y <= q]
  outpt
}

pred <- function(beta, data){
  beta[1] +
    beta[2] * data$lstat +
    beta[3] * data$rm
}

risk <- function(beta, data, tau){
  q <- pred(beta, data)
  mean(loss(data$medv, q, tau))
}

tau <- c(0.1, 0.5, 0.9)

fit <- lapply(tau, function(t)
  optim(rep(0, 3),
        fn   = risk,
        data = dat_train,
        tau  = t,
        method = "BFGS")
)



betas  <- sapply(fit, function(obj) obj$par)
losses <- sapply(1:3, function(i)
  mean(loss(dat_test$medv, pred(betas[, i], dat_test), tau[i]))
)

coverage <- sapply(1:3, function(i)
  mean(dat_test$medv <= pred(betas[, i], dat_test))
)

results <- rbind(
  beta_0 = betas[1, ],
  beta_1 = betas[2, ],
  beta_2 = betas[3, ],
  empirical_loss = losses,
  empirical_coverage = coverage
)

colnames(results) <- c("0.1", "0.5", "0.9")
results <- data.frame(tau = rownames(results), results,
                      row.names = NULL, check.names = FALSE)

tabl_parta <- kable(results, digits = 4, booktabs = TRUE,
                    caption = "QR output for the linear model")
```

```r
kable_styling(tabl_parta, latex_options = "HOLD_position")

dat_train$lstat2 <- dat_train$lstat^2
dat_train$rm2 <- dat_train$rm^2
dat_train$lstat_rm <- dat_train$lstat * dat_train$rm

dat_test$lstat2 <- dat_test$lstat^2
dat_test$rm2 <- dat_test$rm^2
dat_test$lstat_rm <- dat_test$lstat * dat_test$rm

pred1 <- function(beta, data){
  beta[1] +
    beta[2] * data$lstat +
    beta[3] * data$rm +
    beta[4] * data$lstat2 +
    beta[5] * data$rm2 +
    beta[6] * data$lstat_rm
}

risk1 <- function(beta, data, tau){
  q <- pred1(beta, data)
  mean(loss(data$medv, q, tau))
}



fit1 <- lapply(tau, function(t)
  optim(rep(0, 6),
        fn   = risk1,
        data = dat_train,
        tau  = t,
        method = "BFGS")
)


betas1 <- sapply(fit1, function(obj) obj$par)


losses1 <- sapply(1:3, function(i)
  mean(loss(dat_test$medv,
            pred1(betas1[, i], dat_test),
            tau[i]))
)


coverage1 <- sapply(1:3, function(i)
  mean(dat_test$medv <= pred1(betas1[, i], dat_test))
)


results1 <- rbind(
  beta_0 = betas1[1, ],
  beta_1 = betas1[2, ],
```

```
  beta_2= betas1[3, ],
  beta_3 = betas1[4, ],
  beta_4 = betas1[5, ],
  beta_5 = betas1[6, ],
  empirical_loss = losses1,
  empirical_coverage = coverage1
)

colnames(results1) <- c("0.1", "0.5", "0.9")
results1 <- data.frame(tau = rownames(results1), results1,
                       row.names = NULL,
                       check.names = FALSE)

tabl_partb <- kable(results1,
                    digits = 4,
                    booktabs = TRUE,
                    caption = "QR output for the quadratic and interaction model")

kable_styling(tabl_partb, latex_options = "HOLD_position")

#largely using the WEEK10 lecture1 skeleton
riskridge <- function(beta, data, tau, lambda){
  q <- pred1(beta, data)
  mean(loss(data$medv, q, tau)) + lambda * sum(beta[2:6]^2) / 2
}
best_lambda <- function(t){
n_train <- length(dat_train$medv)
kfold <- 10
fold <- sample(rep(1:kfold, length.out = n_train))

pred_cols <- c("lstat", "rm", "lstat2", "rm2", "lstat_rm")
lambda <- seq(0.0001,0.1,length.out = 30)
cv_risk <- numeric(length(lambda))

for (l in seq_along(lambda)) {
  risk <- numeric(kfold)
  for (j in 1:kfold) {
    tr <- fold != j
    va <- fold == j

    df_tr <- data.frame(medv = dat_train$medv[tr], as.matrix(dat_train[tr, pred_cols]))
    df_va <- data.frame(medv = dat_train$medv[va], as.matrix(dat_train[va, pred_cols]))
    fit <- optim(rep(0, 6),
            fn   = riskridge,
            data = df_tr,
            tau  = t,
            lambda = lambda[l],
            method = "BFGS")


    risk[j] <-  riskridge(fit$par,df_va,t,lambda[l])
  }
  cv_risk[l] <- mean(risk)
```

```
}


plot(lambda, cv_risk, type = "b", pch = 19,
     xlab = "Lambda", ylab ="risk")

best_lam_idx <- which.min(cv_risk)
best_lam <- lambda[best_lam_idx]
return(best_lam)
}
par(mar=c(4, 3.5, 2, 2), cex.lab=0.7, cex.axis=0.7, mgp=c(1.5, 0.5, 0))
t <- 0.1
ptone_lambda <- best_lambda(t)
ptone_ridge_beta <- optim(rep(0, 6),
            fn   = riskridge,
            data = dat_train,
            tau  = t,
            lambda = ptone_lambda,
            method = "BFGS")
par(mar=c(4, 3.5, 2, 2), cex.lab=0.7, cex.axis=0.7, mgp=c(1.5, 0.5, 0))
t <- 0.5
ptfive_lambda <- best_lambda(t)
ptfive_ridge_beta <- optim(rep(0, 6),
            fn   = riskridge,
            data = dat_train,
            tau  = t,
            lambda = ptfive_lambda,
            method = "BFGS")
par(mar=c(4, 3.5, 2, 2), cex.lab=0.7, cex.axis=0.7, mgp=c(1.5, 0.5, 0))
t <- 0.9
ptnine_lambda <- best_lambda(t)
ptnine_ridge_beta <- optim(rep(0, 6),
            fn   = riskridge,
            data = dat_train,
            tau  = t,
            lambda = ptnine_lambda,
            method = "BFGS")

tau <- c(0.1,0.5,0.9)
betas_matrix <- rbind(
  ptone_ridge_beta$par,
  ptfive_ridge_beta$par,
  ptnine_ridge_beta$par
)

losses2 <- sapply(1:3, function(i)
  mean(loss(dat_test$medv,
            pred1(betas_matrix[i,], dat_test),
            tau[i]))
)
coverage2 <- sapply(1:3, function(i)
  mean(dat_test$medv <= pred1(betas_matrix[i,], dat_test))
)
```

```r
lambdas <- c(ptone_lambda, ptfive_lambda, ptnine_lambda)

rownames(betas_matrix) <- c("0.1", "0.5", "0.9")
colnames(betas_matrix) <- paste0("beta_", 0:5)


results2 <- rbind(
  t(betas_matrix),
  lambda = lambdas,
  empirical_loss= losses2,
  empirical_coverage = coverage2
)

results2 <- data.frame(
  tau = rownames(results2),
  results2,
  row.names = NULL,
  check.names = FALSE
)

tabl_partc <- kable(
  results2,
  digits = 4,
  booktabs = TRUE,
  caption = "QRR output for the quadratic and interaction model"
)

kable_styling(tabl_partc, latex_options = "HOLD_position")
lstat_seq <- seq(min(dat$lstat), max(dat$lstat), length.out = 80)
rm_seq <- seq(min(dat$rm), max(dat$rm), length.out = 80)

grid <- expand.grid(lstat = lstat_seq, rm = rm_seq)
grid_2 <- grid
grid_2$lstat2 <- grid$lstat^2
grid_2$rm2 <- grid$rm^2
grid_2$lstat_rm <- grid$lstat * grid$rm

q01_1 <- pred(betas[,1], grid)
q05_1 <- pred(betas[,2], grid)
q09_1 <- pred(betas[,3], grid)
cross_01_05_1 <- round(mean(q01_1 > q05_1), 2)
cross_05_09_1 <- round(mean(q05_1 > q09_1), 2)
cross_01_09_1 <- round(mean(q09_1 < q01_1), 2)

q01_2 <- pred1(betas1[,1], grid_2)
q05_2 <- pred1(betas1[,2], grid_2)
q09_2 <- pred1(betas1[,3], grid_2)
cross_01_05_2 <- round(mean(q01_2 > q05_2), 2)
cross_05_09_2 <- round(mean(q05_2 > q09_2), 2)
cross_01_09_2 <- round(mean(q09_2 < q01_2), 2)

q01_3 <- pred1(betas_matrix[1,], grid_2)
q05_3 <- pred1(betas_matrix[1,], grid_2)
```

```r
q09_3 <- pred1(betas_matrix[1,], grid_2)
cross_01_05_3 <- round(mean(q01_3 > q05_3), 2)
cross_05_09_3 <- round(mean(q05_3 > q09_3), 2)
cross_01_09_3 <- round(mean(q09_3 < q01_3), 2)

summry <- rbind(
  linear = c(cross_01_05_1, cross_05_09_1, cross_01_09_1),
  quad = c(cross_01_05_2, cross_05_09_2, cross_01_09_2),
  quad_ridge = c(cross_01_05_3, cross_05_09_3, cross_01_09_3)
)
summry <- data.frame(
  Q01_05 = summry[,1],
  Q05_09 = summry[,2],
  Q01_09 = summry[,3]
)

tabl_partd <- kable(
  summry,
  booktabs = TRUE,
  caption = "Rate of inter-quartile crossing between the linear,
  quadratic-interaction and quadratic-interaction
  with ridge regression and cross validation"
)

kable_styling(tabl_partd, latex_options = "HOLD_position")

#(1) (a)
gelman.diag(mcmc.list(chains))
mean(acc)

#(1) (b)
integrate(function(x) g(x), -99, 2)$value
```

# References


Jarner, Søren Fiig, and Ernst Hansen. 2000. "Geometric Ergodicity of Metropolis Algorithms." *Stochastic Processes and Their Applications* 85 (2): 341–61. https://doi.org/https://doi.org/10.1016/S0304-4149(99)00082-4.

Jones, Galin L. 2004. "On the Markov Chain Central Limit Theorem." *Probability Surveys* 1: 299–320. https://doi.org/10.1214/154957804100000051.

Kohavi, Ron. 1995. "A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection." In *Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 2*, 1137–43. IJCAI'95. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Liu, Ying, Dootika Vats, and James M. Flegal. 2019. "Batch Size Selection for Variance Estimators in MCMC." https://arxiv.org/abs/1804.05975.

Mengersen, K. L., and R. L. Tweedie. 1996. "Rates of Convergence of the Hastings and Metropolis Algorithms." *The Annals of Statistics* 24 (1): 101–21. https://doi.org/10.1214/aos/1033066201.

Muggeo, Vito M. R., Mariangela Sciandra, Agostino Tomasello, and Sebastiano Calvo. 2013. "Estimating Growth Charts via Nonparametric Quantile Regression: A Practical Framework with Application in Ecology." *Environmental and Ecological Statistics* 20 (4): 519–31. https://doi.org/10.1007/s10651-012-0232-1.

Rosenthal, Jeffrey S. 2011. "Optimal Proposal Distributions and Adaptive MCMC." In *Handbook of Markov Chain Monte Carlo*, edited by S. Brooks, A. Gelman, G. L. Jones, and X.-L. Meng. Chapman & Hall/CRC.