Advanced Bayesian Methods: Theory and Applications in R

Markov Chain Monte Carlo Simulations

Nikolaus Umlauf

https://nikum.org/abm.html

Markov Chain Monte Carlo Simulations

Numerically assessing the posterior:

- The ultimate outcome of a Bayesian data analysis is the posterior, reflecting posterior beliefs about the parameter of interest.
- This is often reduced to point estimates, credible intervals, etc.
- Unfortunately, in most models of reasonable complexity, the posterior is not analytically accessible.
- In particular, the normalizing constant

$$p(\mathbf{y}) = \int p(\mathbf{y}|\boldsymbol{\theta})p(\boldsymbol{\theta})d\boldsymbol{\theta}$$

  is unknown and for models of at least moderate complexity it can also not easily be numerically determined.

Markov Chain Monte Carlo Simulations

- If we could obtain random samples $\boldsymbol{\theta}^{[t]}$, $t = 1, \ldots, T$ from the posterior, we could empirically estimate any quantity of interest at any desired level of precision:
  - Posterior expectations can be determined based on the law of large numbers via
$$\frac{1}{T}\sum_{t=1}^{T} g(\boldsymbol{\theta}^{[t]}) \to \mathrm{E}(g(\boldsymbol{\theta})|\mathbf{y}).$$
  - Similar statements exist for empirical quantiles.
  - Even the complete posterior could be estimated based on histograms or kernel density estimates.

- Markov chain Monte Carlo simulations are a way of simulating from the unknown and numerically untractable posterior!

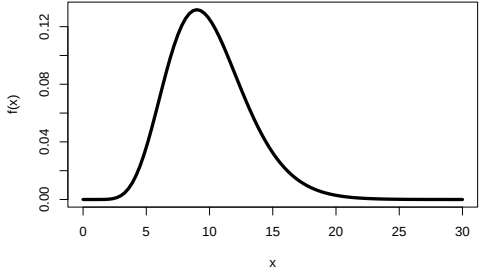
Markov Chain Monte Carlo Simulations

- Hence, if we can get the computer to take a random sample from $p(\boldsymbol{\theta}|\mathbf{y})$, we may approximate $\mathrm{E}(g(\boldsymbol{\theta})|\mathbf{y})$ by simply averaging the function of interest evaluated at the random sample.
- In practice, this means that we simply have to draw enough samples from the posterior distribution to get a sufficiently good approximation.

Markov Chain Monte Carlo Simulations

Example:
Suppose we would like to compute the integral of the Gamma distribution with `shape = 10` and `rate = 1` (the default) within the interval $[0, 30]$.

```
R> par(mar = c(4.1, 4.1, 0.5, 0.1))
R> f <- function(x) dgamma(x, shape = 10)
R> curve(f, 0, 30, lwd = 4)
```



Markov Chain Monte Carlo Simulations

Integral can be computed using the cumulative distribution function (CDF) of the Gamma distribution and gives

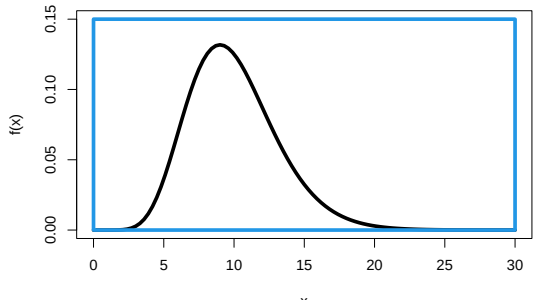
```
R> pgamma(30, shape = 10) - pgamma(0, shape = 10)
[1] 0.9999929
```

and results not too surprisingly nearly 1.

Markov Chain Monte Carlo Simulations

To compute the integral using simulation we need to define a rectangle that encloses the density function, e.g., with

```
R> par(mar = c(4.1, 4.1, 0.5, 0.1))
R> curve(f, 0, 30, ylim = c(0, 0.15), lwd = 4)
R> rect(0, 0, 30, 0.15, border = 4, lwd = 4)
```

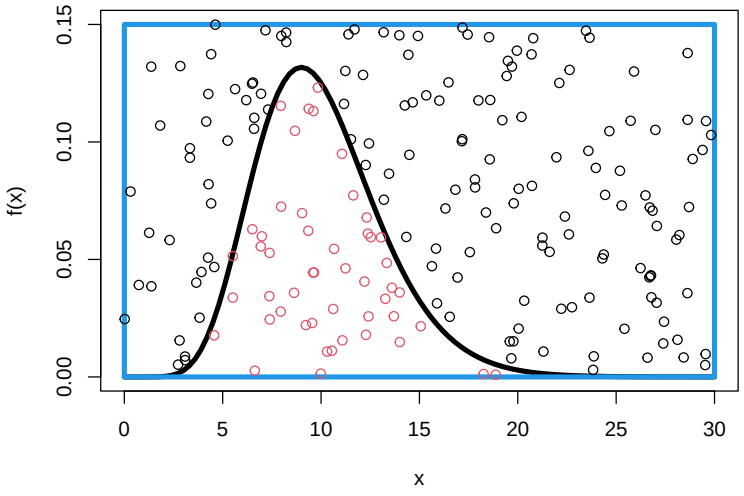


Markov Chain Monte Carlo Simulations

Now, we can easily draw random points within this rectangle using the `runif()` function. Let's sample 200 points for the moment and color each point that lies below the density curve in red.

```
R> set.seed(123)
R> par(mar = c(4.1, 4.1, 0.5, 0.1))
R> curve(f, 0, 30, ylim = c(0, 0.15), lwd = 4)
R> rect(0, 0, 30, 0.15, border = 4, lwd = 4)
R> n <- 200
R> p <- data.frame("x" = runif(n, 0, 30), "y" = runif(n, 0, 0.15))
R> fx <- f(p$x)
R> col <- rep(1, n)
R> col[p$y <= fx] <- 2
R> points(p, col = col)
```

Markov Chain Monte Carlo Simulations



Markov Chain Monte Carlo Simulations

In order to approximate the integral by simulation, we only need the ratio of points that lie below the density function and scale it with the area of the rectangle $30 \cdot 0.15 = 4.5$

```
R> mean(p$y <= fx) * (30 * 0.15)
[1] 1.1025
```

Ok, this is a very rough approximation since the integral should be very close to 1. If we use numerical integration as implemented in the `integrate()` function we get

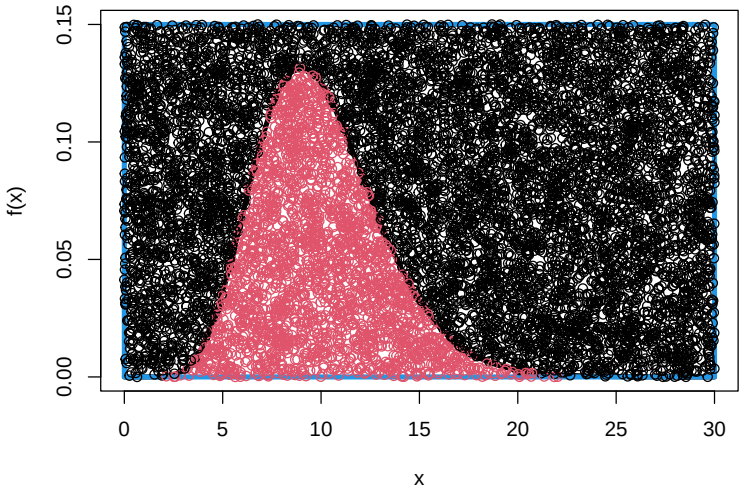
```
R> integrate(f, 0, 30)
0.9999929 with absolute error < 2.1e-06
```

Markov Chain Monte Carlo Simulations

However, when sampling more data points, e.g., 10000 we get pretty close

```
R> par(mar = c(4.1, 4.1, 0.5, 0.1))
R> curve(f, 0, 30, ylim = c(0, 0.15), lwd = 4)
R> rect(0, 0, 30, 0.15, border = 4, lwd = 4)
R> n <- 10000
R> p <- data.frame("x" = runif(n, 0, 30), "y" = runif(n, 0, 0.15))
R> fx <- f(p$x)
R> col <- rep(1, n)
R> col[p$y <= fx] <- 2
R> points(p, col = col)
R> ## Compute the integral.
R> mean(p$y <= fx) * (30 * 0.15)
[1] 1.0287
```

Markov Chain Monte Carlo Simulations

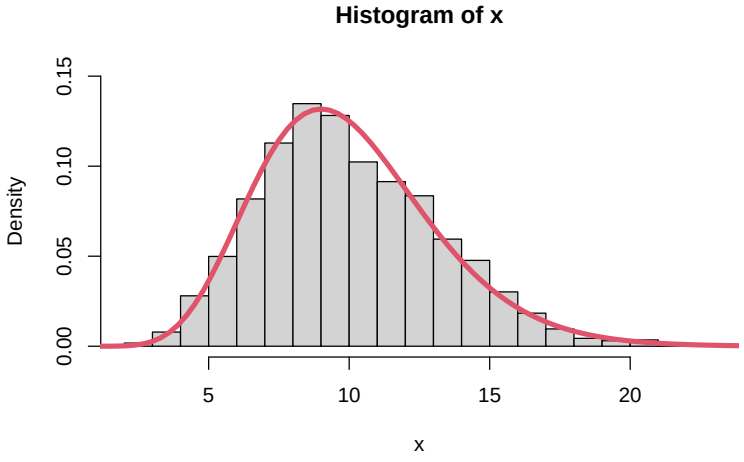


Markov Chain Monte Carlo Simulations

More interestingly for us is the fact that we indeed sampled random numbers from the Gamma distribution with ‘shape = 10‘ within the interval $[0, 30]$. To see this, we just need to drop all points that are above the density function and only save the position of the x-coordinate

```
R> ## Extract the samples saving only
R> ## the x-coordinate of the points.
R> x <- p[p$y <= fx, ]$x
R> ## Draw a histogram.
R> par(mar = c(4.1, 4.1, 4.1, 0.1))
R> hist(x, freq = FALSE, ylim = c(0, 0.15), breaks = "Scott")
R> ## Add the true density curve.
R> curve(f, 0, 30, add = TRUE, col = 2, lwd = 4)
```

Markov Chain Monte Carlo Simulations



Markov Chain Monte Carlo Simulations

If these are true samples from the Gamma distribution, it is easy to compute for example the mean of this distribution

```
R> mean(x)
[1] 10.02421
```

So this is very close to the true mean of 10. Again, let's compare this using numerical integration

```
R> integrate(function(x) f(x) * x, -Inf, Inf)
10 with absolute error < 9.2e-07
```

Of course we could have drawn the Gamma distributed random numbers directly via the inversion method, but that is not the point, for more complex Bayesian regression models we need a general method to draw random numbers from general distributions!

Rejection Sampling

Rejection Sampling

- The Monte Carlo sampling method for drawing random numbers from general distributions has limitations. E.g., specifying interval limits to construct the bounding rectangle can be challenging, and exact generation of random numbers may not always be feasible.
- Rejection sampling addresses this problem by drawing points from a simpler, known distribution $f_Y(x)$ from which random numbers can be easily generated (e.g., using the inversion method). This approach avoids the need for a bounding rectangle.
- The key idea is that the known distribution $f_Y(x)$ must enclose the target distribution $f_X(x)$. Specifically, there must exist a constant ν with $0 < \nu < 1$ such that:

$$\nu f_X(x) \leq f_Y(x) \quad \text{for all } x \in \text{supp}(X).$$

Rejection Sampling

- This condition ensures that $f_Y(x)$ is an envelope distribution. We then accept a sample x from $f_Y(x)$ as a sample from $f_X(x)$ with acceptance probability:

$$\alpha(x) = \nu \cdot \frac{f_X(x)}{f_Y(x)}.$$

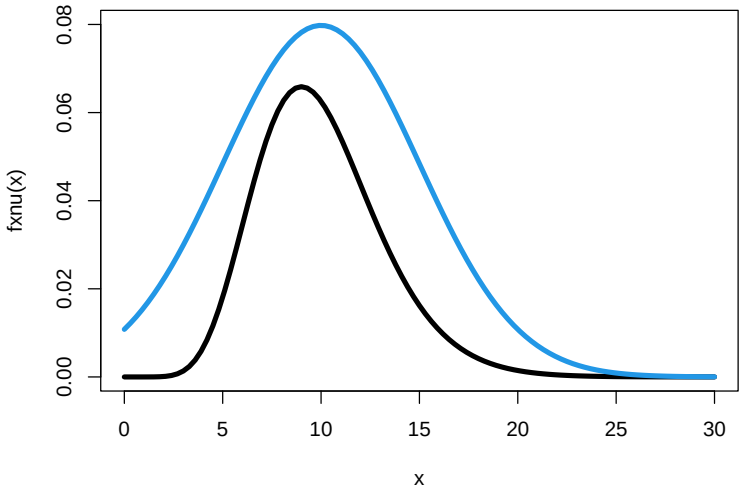
- Here, ν is the probability of accepting a sample, and a larger ν generally improves the efficiency of the sampling process. This method is conceptually similar to using Monte Carlo integration where we check if the sampled height (y-coordinate) is under the density curve we want to sample from.

Rejection Sampling

Example:
To implement the rejection sampler we use the normal distribution with $\mu = 10$ and $\sigma^2 = 25$ as an envelope distribution and set $\nu = 0.5$.

```
R> ## The Gamma distribution we want to draw from.
R> f <- function(x) dgamma(x, shape = 10)
R> ## The nu-scaled version.
R> fxnu <- function(x, nu = 0.5) nu * f(x)
R> par(mar = c(4.1, 4.1, 0.5, 0.1))
R> curve(fxnu, 0, 30, lwd = 4, ylim = c(0, 0.08))
R> ## The envelope distribution.
R> f_Y <- function(x) dnorm(x, mean = 10, sd = 5)
R> ## Add to plot.
R> curve(f_Y, 0, 30, col = 4, lwd = 4, add = TRUE)
```

Rejection Sampling



Rejection Sampling

The algorithm can be implemented with

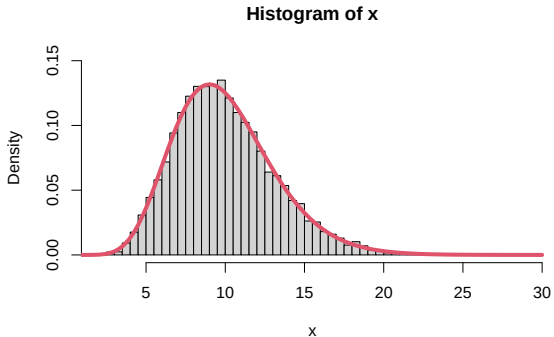
```
R> ## The rejection sampler function.
R> rsamp <- function(n, f_X, f_Y, q_Y, nu)
+  {
+    x <- rep(NA, length = n)
+    i <- 0
+    while(i < n) {
+      y <- q_Y(runif(1))
+      alpha <- nu * f_X(y) / f_Y(y)
+      u <- runif(1)
+      if(u <= alpha) {
+        i <- i + 1
+        x[i] <- y
+      }
+    }
+    return(x)
+  }
```

Rejection Sampling

```
R> ## Quantile function of the envelope distribution to
R> ## draw samples using the inversion method.
R> q_Y <- function(p) qnorm(p, mean = 10, sd = 5)
R> ## Draw 10000 samples.
R> set.seed(123)
R> x <- rsamp(10000, f_X = f, f_Y = f_Y, q_Y = q_Y, nu = 0.5)
```

Rejection Sampling

```
R> ## And plot the histogram.
R> hist(x, freq = FALSE, breaks = "Scott", ylim = c(0, 0.15))
R> curve(f, 0, 30, lwd = 4, col = 2, add = TRUE)
```



Rejection Sampling

The rejection sampler apparently worked very well, compared to the Monte Carlo method we can actually generate random numbers for arbitrary distributions without having to determine interval limits in advance.

Rejection Sampling - Linear Model

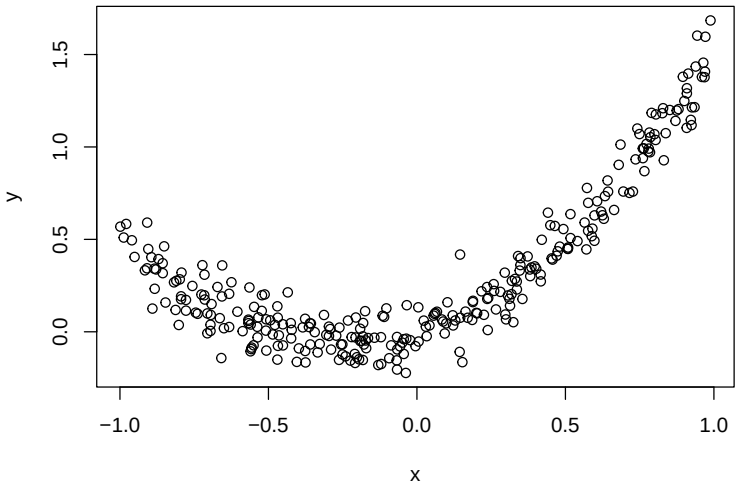
We simulate a nonlinear relationship with

$$y = 0.5x + x^2 + \varepsilon$$

and $\varepsilon \sim N(0, 0.1^2)$.

```
R> library("MASS")  ## For mvrnorm() function.
R> set.seed(123)
R> n <- 300
R> x <- runif(n, -1, 1)
R> y <- 0 + 0.5 * x + x^2 + rnorm(n, sd = 0.1)
R> par(mar = c(4.1, 4.1, 0.5, 0.1))
R> plot(x, y)
```

Rejection Sampling - Linear Model



Rejection Sampling - Linear Model

Assign a vague normal prior on the regression coefficients β. Moreover, we work with the log-densities which is numerically much more stable. The log-posterior can be set up with

```
R> log_likelihood <- function(theta, X, y) {
+    sum(dnorm(y, mean = X %*% theta, sd = 0.1, log = TRUE))
+  }
R> log_prior <- function(theta) {
+    sum(dnorm(theta, mean = 0, sd = 1000, log = TRUE))
+  }
R> log_posterior <- function(theta, X, y) {
+    log_likelihood(theta, X, y) + log_prior(theta)
+  }
R> log_envelope <- function(theta) {
+    -0.5 * theta %*% Sigma %*% theta
+  }
```

Rejection Sampling - Linear Model

Now, we set up the design matrix **X** using orthogonal polynomials and specify the scaling parameter ν of the rejection sampler. The algorithm runs until 100 samples are generated.

```
R> ## Design matrix.
R> X <- cbind(1, poly(x, 3))
R> k <- ncol(X)
R> ## Rejection sampler scaling parameter.
R> nu <- 10000
R> ## Number of samples.
R> nsamps <- 100
R> ## Store the accepted samples.
R> theta_samples <- matrix(NA, nsamps, k)
R> ## Multivariate envelope distribution: Mean = 0, Covariance = Diagonal.
R> Sigma <- diag(10, k)
R> ## Initialize counters.
R> i <- 0; j <- 0
```

Rejection Sampling - Linear Model

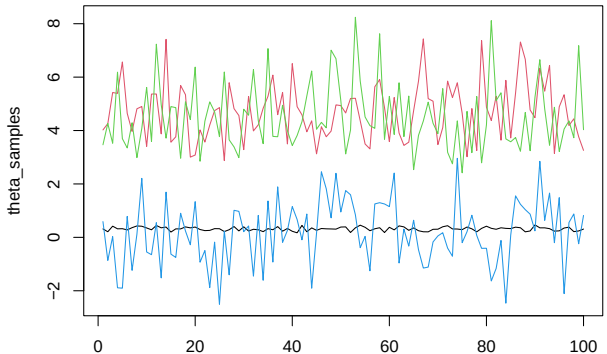
```
R> ## Rejection sampler.
R> while(i < nsamps) {
+    ## Step 1: Propose theta from the multivariate normal envelope distribution.
+    theta_proposal <- mvrnorm(1, mu = rep(0, k), Sigma = Sigma)
+
+    ## Step 2: Compute log acceptance ratio.
+    log_acceptance_ratio <- log_posterior(theta_proposal, X, y) -
+                            log_envelope(theta_proposal) + log(nu)
+
+    ## Step 3: Decide whether to accept.
+    u <- runif(1)
+    if(u <= exp(log_acceptance_ratio)) {
+      i <- i + 1
+      theta_samples[i, ] <- theta_proposal  ## Save the accepted sample.
+    }
+
+    j <- j + 1
+  }
```

Rejection Sampling - Linear Model

```
R> ## Remove NA rows (unsuccessful samples).
R> theta_samples <- na.omit(theta_samples)
R> ## Final number of saved samples.
R> print(dim(theta_samples))
[1] 100   4
R> print(j)
[1] 209088
```

Rejection Sampling - Linear Model

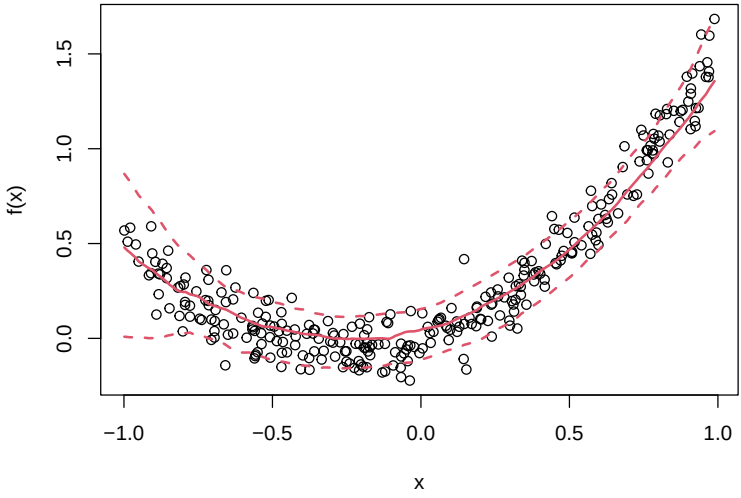
```
R> par(mar = c(4.1, 4.1, 0.5, 0.1))
R> matplot(theta_samples, type = "l", lty = 1, col = 1:ncol(theta_samples))
```



Rejection Sampling - Linear Model

```
R> ## Evaluate the regression curve for each sample.
R> fit <- apply(theta_samples,
+    MARGIN = 1,
+    function(beta) {
+      drop(X %*% beta)
+  })
R> ## Compute quantiles from samples.
R> fit <- t(apply(fit, 1, quantile, probs = c(0.1, 0.5, 0.9), na.rm = TRUE))
R> ## Plot the estimate.
R> par(mar = c(4.1, 4.1, 0.1, 0.1))
R> plot(x, y, xlab = "x", ylab = "f(x)")
R> i <- order(x)
R> matplot(x[i], fit[i, ], type = "l",
+    lty = c(2, 1, 2), col = 2,
+    lwd = 2, add = TRUE)
```

Rejection Sampling - Linear Model



Rejection Sampling - Linear Model

- Seems work quite well, however, we only obtained 100 samples although we used 209088 iterations of the algorithm. Hence, the acceptance rate is extremely low at 0.05%.
- Also, the standard deviation of the envelop density is crucial, if it is too low or too high the sampler will always reject.
- This behaviour already shows a little bit the limits of this method, especially if we remember that we want to estimate much more complex models with much more parameters. Rejection sampling quickly reaches its limits here, so we need more efficient methods to draw random numbers from the posterior density. The reason is that the probability of rejection increases exponentially as a function of the number of parameters.

Markov Chains

Markov Chain: A sequence of random variables $\left\{\theta^{[t]}\right\}$ where each state $\theta^{[t]}$ depends only on the previous state $\theta^{[t-1]}$.

- The chain evolves according to a **transition kernel** $K(\cdot, \cdot)$.
- Formally, the distribution of $\theta^{[t+1]}$ given the past states depends only on $\theta^{[t]}$:

$$\theta^{[t+1]} | \theta^{[0]}, \theta^{[1]}, \ldots, \theta^{[t]} \sim K(\theta^{[t]}, \theta^{[t+1]}).$$

Transition Kernel vs. Proposal Density

Proposal Density $Q(\theta^*|\theta^{[t]})$:

- Defines how to propose a new state θ^* given the current state $\theta^{[t]}$.
- Example: In a random walk, $\theta^* = \theta^{[t]} + \varepsilon_t$ with $\varepsilon_t \sim N(0, \sigma^2)$.

Transition Kernel $K(\theta^{[t]}, \theta^{[t+1]})$:

- Describes the probability of transitioning from $\theta^{[t]}$ to $\theta^{[t+1]}$.
- Combines the proposal density and the acceptance probability.

$$K(\theta^{[t]}, \theta^{[t+1]}) = Q(\theta^*|\theta^{[t]}) \cdot \alpha(\theta^*|\theta^{[t]}).$$

Metropolis-Hastings Algorithm

To sample from the posterior distribution $p(\boldsymbol{\theta}|\mathbf{y})$, use the **Metropolis-Hastings (MH) algorithm**:

- **Proposal Step**: Propose a new state $\boldsymbol{\theta}^*$ using a proposal density $Q(\boldsymbol{\theta}^*|\boldsymbol{\theta}^{[t]})$.
- **Acceptance Step**: Accept or reject $\boldsymbol{\theta}^*$ based on the acceptance probability:

$$\alpha(\boldsymbol{\theta}^*|\boldsymbol{\theta}^{[t]}) = \min\left\{\frac{p(\boldsymbol{\theta}^*|\mathbf{y}) \cdot Q(\boldsymbol{\theta}^{[t]}|\boldsymbol{\theta}^*)}{p(\boldsymbol{\theta}^{[t]}|\mathbf{y}) \cdot Q(\boldsymbol{\theta}^*|\boldsymbol{\theta}^{[t]})}, 1\right\}$$

- If accepted, set $\boldsymbol{\theta}^{[t+1]} = \boldsymbol{\theta}^*$; otherwise, $\boldsymbol{\theta}^{[t+1]} = \boldsymbol{\theta}^{[t]}$.

Simplification for Symmetric Proposals

When using symmetric proposals where $Q(\boldsymbol{\theta}^*|\boldsymbol{\theta}^{[t]}) = Q(\boldsymbol{\theta}^{[t]}|\boldsymbol{\theta}^*)$:

- The acceptance probability simplifies to:

$$\alpha(\boldsymbol{\theta}^*|\boldsymbol{\theta}^{[t]}) = \min\left\{\frac{p(\boldsymbol{\theta}^*|\mathbf{y})}{p(\boldsymbol{\theta}^{[t]}|\mathbf{y})}, 1\right\}$$

- This simplifies computations because the proposal density terms cancel out.

Random Walk Kernel

A simple random walk kernel is defined as:

$$\theta^{[t+1]} = \theta^{[t]} + \varepsilon_t$$

where $\varepsilon_t \sim N(0, \sigma^2)$.

- This kernel suggests moving from $\theta^{[t]}$ by adding a normally distributed error term.
- The choice of σ^2 affects the step size and exploration.

Challenges in MCMC

- All MCMC methods dealt with so far have the disadvantage that one has to carefully turn several set screws in order to be able to cope with more complicated estimation problems.
- A particularly attractive method, and I will say right away that it is not suitable for all possible problems, is called **slice sampling** (Neal 2003).

Concept of Slice Sampling

- Suppose you want to sample from an arbitrary density an univariate variable $X \sim p(x)$ and further assume that it is impossible to sample from $p(x)$ directly.
- The idea is to introduce an auxiliary variable U that allows us to explore the joint density $(X, U) \sim p(x, u)$ as our target density.
- By simple rules we may write:

$$p(x, u) = p(x)p(u \mid x).$$

- This leads to an auxiliary sampling strategy where we can first sample U and X afterwards.

Slice Sampling Algorithm

The general sampling scheme with posterior density $p(\boldsymbol{\theta} \mid \mathbf{y})$ is actually quite simple:

1. Sample $h \sim \mathcal{U}(0, p(\theta_j^{[t]} \mid \cdot))$.
2. Sample $\theta_j^{[t+1]} \sim \mathcal{U}(S)$ from the horizontal slice

$$S = \{\theta_j \mid h < p(\theta_j \mid \cdot)\}.$$

Detailed Steps

- Given the current position $\theta_j^{[t]}$, evaluate the (log-) posterior to obtain the height of the function $p(\theta_j^{[t]} \mid \cdot)$.
- Sample uniformly between 0 and $p(\theta_j^{[t]} \mid \cdot)$ to obtain a vertical position h.
- From this sampled height, search for the edges of the target function. This step is computationally more involved.
- If we have the edges, sample a new position $\theta_j^{[t+1]}$ uniformly on this slice.

Implementation

In R:

```
R> ## Function to find edges.
R> ## Argument x is the current position, target is the target function,
R> ## height is the height where we need to find the edges, step is
R> ## the steplength we go left and right of x to find the edges,
R> ## eps is the relative error allowed for finding an edge.
R> find_edges <- function(x, target, height,
+    step = 0.5, eps = 0.0001)
+  {
+    ## Initial left and right edge.
+    left <- right <- x
+
+    ## Step size for both, left and right.
+    step <- rep(step, length.out = 2)
+
+
+
```

Implementation

```
+    ## Search the left side.
+    repeat {
+      if((hs <- target(left)) <= height) {
+        err <- abs((hs - height)/height)
+        if(err <= eps) {
+          break
+        } else {
+          left <- left + step[1]
+          step[1] <- 0.95 * step[1]
+        }
+      }
+      left <- left - step[1]
+    }
+
+
+
+
+
```

Implementation

```
+    ## Search the right side.
+    repeat {
+      if((hs <- target(right)) <= height) {
+        err <- abs((hs - height)/height)
+        if(err <= eps) {
+          break
+        } else {
+          right <- right - step[2]
+          step[2] <- 0.95 * step[2]
+        }
+      }
+      right <- right + step[2]
+    }
+
+    c(left, right)
+  }
```

Implementation

Test on complicated mixture distribution.

```
R> f <- function(x) {
+    0.4 * dnorm(x, mean = -1, sd = 1) + 0.6 * dgamma(x, shape = 6, rate = 1)
+  }
```

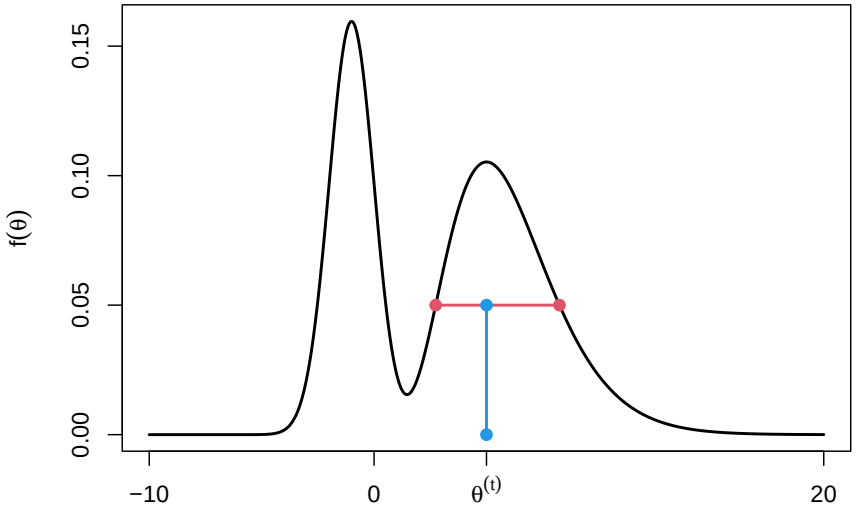
Find the edges.

```
R> edges <- find_edges(5, target = f, height = 0.05)
R> print(edges)
[1] 2.739709 8.246376
```

Implementation

```
R> par(mar = c(4.1, 4.1, 0.1, 0.1))
R> curve(f, -10, 20, lwd = 2, axes = FALSE, n = 500,
+    xlab = "", ylab = expression(f(theta)))
R> points(edges[1], f(edges[1]), col = 2, pch = 16, cex = 1.2)
R> points(edges[2], f(edges[2]), col = 2, pch = 16, cex = 1.2)
R> lines(c(edges), f(edges), col = 2, lwd = 2)
R> lines(c(5, 5), c(0, 0.05), col = 4, lwd = 2)
R> points(c(5, 5), c(0, 0.05), col = 4, pch = 16, cex = 1.2)
R> axis(1, at = 5, label = expression(theta^(t)))
R> axis(2)
R> axis(1, at = c(-10, 0, 20))
R> box()
```

Implementation



Gibbs Sampling

- Often, random numbers can be directly drawn from the conditional densities $p(\theta_1|\cdot), p(\theta_2\,|\,\cdot), \ldots, p(\theta_S|\cdot)$, where $p(\theta_s|\cdot)$ denotes the conditional density of θ_s given all other blocks $\theta_1, \ldots, \theta_{s-1}, \theta_{s+1}, \ldots, \theta_S$ and the data $\mathbf{y}$.
- These densities are called **full conditionals**.
- **Gibbs sampling** consists in successively drawing random numbers from the full conditionals in each iteration and accepting them (with probability one) as the next state of the Markov chain.
- After a burn-in phase the sampled random numbers can be considered as realizations from the marginal posteriors $p(\theta_1|\mathbf{y}), p(\theta_2|\mathbf{y}), \ldots, p(\theta_S|\mathbf{y})$.
- In the following, we set up a Gibbs sampler for the normal linear regression model.

Gibbs Sampling

Linear model: For $i = 1, \ldots, n$:

$$\begin{aligned} y_i &= f(x_{i1}, \ldots, x_{ik}) + \varepsilon_i \\ &= \beta_0 + \beta_1 x_{i1} + \ldots + \beta_k x_{ik} + \varepsilon_i \\ &= \beta_0 + \sum_{j=1}^{k} \beta_j x_{ij} + \varepsilon_i \\ &= \mathbf{x}_i^\top \boldsymbol{\beta} + \varepsilon_i. \\ &= \eta_i + \varepsilon_i. \end{aligned}$$

In matrix notation:

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon} = \boldsymbol{\eta} + \boldsymbol{\varepsilon},$$

and $\varepsilon \sim N(\mathbf{0}, \sigma^2 \mathbf{I})$

Gibbs Sampling

- The likelihood of the model is again

$$p(\mathbf{y}|\mu,\sigma^2) = \frac{1}{(2\pi\sigma^2)^{\frac{n}{2}}}\exp\left(-\frac{1}{2\sigma^2}(\mathbf{y}-\mathbf{X}\boldsymbol{\beta})^\top(\mathbf{y}-\mathbf{X}\boldsymbol{\beta})\right)$$

- The priors are given as follows

$$p(\boldsymbol{\beta},\sigma^2) = p(\boldsymbol{\beta}|\sigma^2)p(\sigma^2),$$

where

$$\boldsymbol{\beta}|\sigma^2 \sim N(\mathbf{m},\sigma^2\mathbf{M}),\ \ \sigma^2 \sim IG(a,b),$$

Gibbs Sampling

$$
\begin{aligned}
p(\boldsymbol{\beta}, \sigma^2) &= \frac{1}{(2\pi)^{\frac{p}{2}} (\sigma^2)^{\frac{p}{2}} |\mathbf{M}|^{\frac{1}{2}}} \exp\left(-\frac{1}{2\sigma^2} (\boldsymbol{\beta} - \mathbf{m})^\top \mathbf{M}^{-1} (\boldsymbol{\beta} - \mathbf{m}) \right) \\
&\quad \frac{b^a}{\Gamma(a)} (\sigma^2)^{-(a+1)} \exp\left(-b/\sigma^2 \right).
\end{aligned}
$$

Gibbs Sampling

- Combining the prior with the likelihood, and dropping all terms that are multiplicatively unrelated to our parameters of interest yields the posterior kernel

$$
\begin{aligned}
p(\boldsymbol{\beta}|\cdot) &\propto \exp\left(-\frac{1}{2\sigma^2}(\mathbf{y}-\mathbf{X}\boldsymbol{\beta})^\top(\mathbf{y}-\mathbf{X}\boldsymbol{\beta})\right) \\
&\quad \exp\left(-\frac{1}{2\sigma^2}(\boldsymbol{\beta}-\mathbf{m})^\top\mathbf{M}^{-1}(\boldsymbol{\beta}-\mathbf{m})\right) \\
&\propto \exp\left(-\frac{1}{2\sigma^2}\left((\mathbf{y}-\mathbf{X}\boldsymbol{\beta})^\top(\mathbf{y}-\mathbf{X}\boldsymbol{\beta}) + \right.\right. \\
&\quad \left.\left.(\boldsymbol{\beta}-\mathbf{m})^\top\mathbf{M}^{-1}(\boldsymbol{\beta}-\mathbf{m})\right)\right).
\end{aligned}
$$

Gibbs Sampling

- $$\mathbf{y}^\top\mathbf{y} - \mathbf{y}^\top\mathbf{X}\boldsymbol{\beta} - \boldsymbol{\beta}^\top\mathbf{X}^\top\mathbf{y} + \boldsymbol{\beta}^\top\mathbf{X}^\top\mathbf{X}\boldsymbol{\beta} + \boldsymbol{\beta}^\top\mathbf{M}^{-1}\boldsymbol{\beta} - \boldsymbol{\beta}^\top\mathbf{M}^{-1}\mathbf{m} - \mathbf{m}^\top\mathbf{M}^{-1}\boldsymbol{\beta} + \mathbf{m}^\top\mathbf{M}^{-1}\mathbf{m},$$

since $\mathbf{y}^\top\mathbf{X}\boldsymbol{\beta} = \boldsymbol{\beta}^\top\mathbf{X}^\top\mathbf{y}$ and $\boldsymbol{\beta}^\top\mathbf{M}^{-1}\mathbf{m} = \mathbf{m}^\top\mathbf{M}^{-1}\boldsymbol{\beta}$ (both have dimension 1×1), we get

$$\begin{aligned} & -\frac{1}{2\sigma^2}\left(-2\boldsymbol{\beta}^\top\mathbf{X}^\top\mathbf{y} + \boldsymbol{\beta}^\top\mathbf{X}^\top\mathbf{X}\boldsymbol{\beta} + \boldsymbol{\beta}^\top\mathbf{M}^{-1}\boldsymbol{\beta} - 2\boldsymbol{\beta}^\top\mathbf{M}^{-1}\mathbf{m}\right) \\ = \; & \frac{1}{\sigma^2}\boldsymbol{\beta}^\top\mathbf{X}^\top\mathbf{y} - \frac{1}{2\sigma^2}\boldsymbol{\beta}^\top\mathbf{X}^\top\mathbf{X}\boldsymbol{\beta} - \frac{1}{2\sigma^2}\boldsymbol{\beta}^\top\mathbf{M}^{-1}\boldsymbol{\beta} + \frac{1}{\sigma^2}\boldsymbol{\beta}^\top\mathbf{M}^{-1}\mathbf{m} \\ = \; & \boldsymbol{\beta}^\top\left(\frac{1}{\sigma^2}\mathbf{X}^\top\mathbf{y} + \frac{1}{\sigma^2}\mathbf{M}^{-1}\mathbf{m}\right) - \frac{1}{2}\boldsymbol{\beta}^\top\left(\frac{1}{\sigma^2}\mathbf{X}^\top\mathbf{X} + \frac{1}{\sigma^2}\mathbf{M}^{-1}\right)\boldsymbol{\beta}. \end{aligned}$$

Gibbs Sampling

- The pdf of the multivariate normal distribution is

$$p(\mathbf{x}) = (2\pi)^{-\frac{1}{2}} |\mathbf{\Sigma}| \exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^\top \mathbf{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right).$$

- Similar to the above, ignoring all terms not depending on $\mathbf{x}$ we get

$$\begin{aligned} p(\mathbf{x}) &\propto \exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^\top \mathbf{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right) \\ &= \exp\left(-\frac{1}{2}\mathbf{x}^\top \mathbf{\Sigma}^{-1}\mathbf{x} + \mathbf{x}^\top \mathbf{\Sigma}^{-1}\boldsymbol{\mu} - \frac{1}{2}\boldsymbol{\mu}^\top \mathbf{\Sigma}^{-1}\boldsymbol{\mu}\right) \\ &\propto \exp\left(-\frac{1}{2}\mathbf{x}^\top \mathbf{\Sigma}^{-1}\mathbf{x} + \mathbf{x}^\top \mathbf{\Sigma}^{-1}\boldsymbol{\mu}\right). \end{aligned}$$

Gibbs Sampling

- Hence, the full conditional $p(\boldsymbol{\beta}|\cdot)$ is multivariate normal with covariance matrix

$$\boldsymbol{\Sigma}_{\boldsymbol{\beta}} = \left(\frac{1}{\sigma^2} \mathbf{X}^\top \mathbf{X} + \frac{1}{\sigma^2} \mathbf{M}^{-1} \right)^{-1}$$

  and mean

$$\boldsymbol{\mu}_{\boldsymbol{\beta}} = \boldsymbol{\Sigma}_{\boldsymbol{\beta}} \left(\frac{1}{\sigma^2} \mathbf{X}^\top \mathbf{y} + \frac{1}{\sigma^2} \mathbf{M}^{-1} \mathbf{m} \right).$$

- In similar fashion we derive the full conditional for σ^2

$$\begin{aligned} p(\sigma^2|\cdot) \quad \propto \quad & \frac{1}{(\sigma^2)^{\frac{n}{2}}} \exp\left(-\frac{1}{2\sigma^2} (\mathbf{y} - \mathbf{X}\boldsymbol{\beta})^\top (\mathbf{y} - \mathbf{X}\boldsymbol{\beta}) \right) \\ & \frac{1}{(\sigma^2)^{\frac{p}{2}}} \exp\left(-\frac{1}{2\sigma^2} (\boldsymbol{\beta} - \mathbf{m})^\top \mathbf{M}^{-1} (\boldsymbol{\beta} - \mathbf{m}) \right) \frac{1}{(\sigma^2)^{a+1}} \exp(-b/\sigma^2). \end{aligned}$$

Gibbs Sampling

- $$p(\sigma^2|\cdot) \propto \frac{1}{(\sigma^2)^{a+\frac{n}{2}+\frac{p}{2}+1}} \exp\left(-\frac{1}{\sigma^2}\left(b+\frac{1}{2}(\mathbf{y}-\mathbf{X}\boldsymbol{\beta})^\top(\mathbf{y}-\mathbf{X}\boldsymbol{\beta})+\frac{1}{2}(\boldsymbol{\beta}-\mathbf{m})^\top\mathbf{M}^{-1}(\boldsymbol{\beta}-\mathbf{m})\right)\right)$$

- Now we get new paramaters a' and b' with

$$a' = a + \frac{n}{2} + \frac{p}{2}$$

and

$$b' = b + \frac{1}{2}(\mathbf{y}-\mathbf{X}\boldsymbol{\beta})^\top(\mathbf{y}-\mathbf{X}\boldsymbol{\beta}) + \frac{1}{2}(\boldsymbol{\beta}-\mathbf{m})^\top\mathbf{M}^{-1}(\boldsymbol{\beta}-\mathbf{m})$$

Gibbs Sampling

- Now, similar to the MH-sampler we generate starting values for $\boldsymbol{\beta}$ and σ^2.
- Then, we consecutively sample for $t = 1, \ldots, T$ from the full conditionals

$$\boldsymbol{\beta}^{(t)}|\cdot \sim N\left(\boldsymbol{\mu}_{\boldsymbol{\beta}}^{(t-1)}, \boldsymbol{\Sigma}_{\boldsymbol{\beta}}^{(t-1)}\right)$$

and

$$\sigma^{2(t)}|\cdot \sim IG\left(a'^{(t-1)}, b'^{(t-1)}\right).$$

Locally Quadratic Approximation

- Asymptotic theory suggests that posteriors tend to be normal for large samples.
- The moments of the asymptotic normal can be estimated as the mode and the negative curvature at the mode of the log-posterior.
- Idea: Apply the same idea to the log-full conditionals and propose from the resulting local quadratic approximation.
- Advantages:
  - Automatically adapts to the shape of the full conditionals.
  - Deviations from the asymptotic normal will be corrected for in the acceptance step.
- In the context of regression models, this is also referred to as **iteratively weighted least squares** (IWLS) updates since they resemble IWLS optimisation for finding maximum likelihood estimates.

Locally Quadratic Approximation

- When $\boldsymbol{\Sigma}_{jk}$ is fixed, Markov chains may have poor mixing in regions of low posterior density.
- A solution is to use approximate full conditionals based on a Taylor series expansion of the log-posterior.
- The proposal density $Q(\boldsymbol{\beta}_{jk}^{\star}|\boldsymbol{\beta}_{jk}^{(t)})$ is normal:

$$\left(\boldsymbol{\Sigma}_{jk}^{(t)}\right)^{-1} = -\mathbf{H}_{kk}(\boldsymbol{\beta}_{jk}^{(t)})$$

$$\boldsymbol{\mu}_{jk}^{(t)} = \boldsymbol{\beta}_{jk}^{(t)} - \left[\mathbf{J}_{kk}(\boldsymbol{\beta}_{jk}^{(t)}) + \mathbf{G}_{jk}(\boldsymbol{\tau}_{jk})\right]^{-1} \mathbf{s}(\boldsymbol{\beta}_{jk}^{(t)}).$$

- This is similar to Newton or Fisher scoring iterations towards the posterior mode.

Locally Quadratic Approximation

- For models with basis functions $f_{jk}(\cdot)$ (upcoming), the precision and mean of the proposal distribution are:

$$\left(\boldsymbol{\Sigma}_{jk}^{(t)}\right)^{-1} = \mathbf{X}_{jk}^{\top}\mathbf{W}_{kk}\mathbf{X}_{jk} + \mathbf{G}_{jk}(\boldsymbol{\tau}_{jk})$$

$$\boldsymbol{\mu}_{jk}^{(t)} = \boldsymbol{\Sigma}_{jk}^{(t)}\mathbf{X}_{jk}^{\top}\mathbf{W}_{kk}\left(\mathbf{z}_k - \boldsymbol{\eta}_{k,-j}^{(t)}\right)$$

  where $\mathbf{W}_{kk} = -\operatorname{diag}\left(\frac{\partial^2 \ell(\boldsymbol{\beta};\mathbf{y},\mathbf{X})}{\partial \boldsymbol{\eta}_k \partial \boldsymbol{\eta}_k^{\top}}\right)$.

- The computation of $\boldsymbol{\mu}_{jk}^{(t)}$ is equivalent to one Newton step or Fisher scoring iteration.
- If computing weights is expensive, update $\mathbf{W}_{kk}$ after sampling all parameters of $\boldsymbol{\theta}_k$.

Proposal Density for Sampling

- The proposal density $Q(\boldsymbol{\beta}_{jk}^{\star} | \, \boldsymbol{\beta}_{jk}^{(t)})$ is a normal distribution:

$$Q(\boldsymbol{\beta}_{jk}^{\star} | \, \boldsymbol{\beta}_{jk}^{(t)}) = \mathcal{N}(\boldsymbol{\mu}_{jk}^{(t)}, \boldsymbol{\Sigma}_{jk}^{(t)}).$$

- The precision matrix is given by:

$$\left(\boldsymbol{\Sigma}_{jk}^{(t)}\right)^{-1} = -\mathbf{H}_{kk}\left(\boldsymbol{\beta}_{jk}^{(t)}\right).$$

- The mean of the proposal density is:

$$\begin{aligned}
\boldsymbol{\mu}_{jk}^{(t)} &= \boldsymbol{\Sigma}_{jk}^{(t)}\left[s\left(\boldsymbol{\beta}_{jk}^{(t)}\right) - \mathbf{H}_{kk}\left(\boldsymbol{\beta}_{jk}^{(t)}\right)\boldsymbol{\beta}_{jk}^{(t)}\right] \\
&= \boldsymbol{\beta}_{jk}^{(t)} - \left[\mathbf{H}_{kk}\left(\boldsymbol{\beta}_{jk}^{(t)}\right)\right]^{-1}\mathbf{s}\left(\boldsymbol{\beta}_{jk}^{(t)}\right) \\
&= \boldsymbol{\beta}_{jk}^{(t)} - \left[\mathbf{J}_{kk}\left(\boldsymbol{\beta}_{jk}^{(t)}\right) + \mathbf{G}_{jk}(\boldsymbol{\tau}_{jk})\right]^{-1}\mathbf{s}\left(\boldsymbol{\beta}_{jk}^{(t)}\right).
\end{aligned}$$

Proposal Density for Sampling

- With basis function representation $f_{jk}(\cdot)$, the precision matrix is:

$$\left(\boldsymbol{\Sigma}_{jk}^{(t)}\right)^{-1} = \mathbf{X}_{jk}^{\top}\mathbf{W}_{kk}\mathbf{X}_{jk} + \mathbf{G}_{jk}(\boldsymbol{\tau}_{jk}),$$

where $\mathbf{W}_{kk} = -\mathrm{diag}\left(\frac{\partial^2 \ell(\boldsymbol{\beta};\mathbf{y},\mathbf{X})}{\partial \boldsymbol{\eta}_k \partial \boldsymbol{\eta}_k^{\top}}\right)$.

Proposal Density for Sampling

- The mean is computed as:

$$
\begin{aligned}
\boldsymbol{\mu}_{jk}^{(t)} &= \boldsymbol{\Sigma}_{jk}^{(t)} \left[\mathbf{X}_{jk}^{\top} \mathbf{u}_k^{(t)} - \mathbf{G}_{jk}(\boldsymbol{\tau}_{jk}) \boldsymbol{\beta}_{jk}^{(t)} + (\mathbf{X}_{jk}^{\top} \mathbf{W}_{kk} \mathbf{X}_{jk} + \mathbf{G}_{jk}(\boldsymbol{\tau}_{jk})) \boldsymbol{\beta}_{jk}^{(t)} \right] \\
&= \boldsymbol{\Sigma}_{jk}^{(t)} \left[\mathbf{X}_{jk}^{\top} \mathbf{u}_k^{(t)} + \mathbf{X}_{jk}^{\top} \mathbf{W}_{kk} \mathbf{X}_{jk} \boldsymbol{\beta}_{jk}^{(t)} \right] \\
&= \boldsymbol{\Sigma}_{jk}^{(t)} \left[\mathbf{X}_{jk}^{\top} \mathbf{u}_k^{(t)} + \mathbf{X}_{jk}^{\top} \mathbf{W}_{kk} \left(\boldsymbol{\eta}_k^{(t)} - \boldsymbol{\eta}_{k,-j}^{(t)} \right) \right] \\
&= \boldsymbol{\Sigma}_{jk}^{(t)} \mathbf{X}_{jk}^{\top} \left[\mathbf{u}_k^{(t)} + \mathbf{W}_{kk} \left(\boldsymbol{\eta}_k^{(t)} - \boldsymbol{\eta}_{k,-j}^{(t)} \right) \right] \\
&= \boldsymbol{\Sigma}_{jk}^{(t)} \mathbf{X}_{jk}^{\top} \mathbf{W}_{kk} \left[\boldsymbol{\eta}_k^{(t)} + \mathbf{W}_{kk}^{-1} \mathbf{u}_k^{(t)} - \boldsymbol{\eta}_{k,-j}^{(t)} \right] \\
&= \boldsymbol{\Sigma}_{jk}^{(t)} \mathbf{X}_{jk}^{\top} \mathbf{W}_{kk} \left[\mathbf{z}_k - \boldsymbol{\eta}_{k,-j}^{(t)} \right],
\end{aligned}
$$

where $\mathbf{z}_k = \boldsymbol{\eta}_k^{(t)} + \mathbf{W}_{kk}^{-1} \mathbf{u}_k^{(t)}$.

Generic MCMC

Input: $\mathbf{y}$, $\mathbf{X}$, α, $\boldsymbol{\beta}^{(t)}$, $\boldsymbol{\tau}^{(t)}$.

Set: Sampling method, e.g., derivative-based MCMC, Slice Sampling, etc.

Sample $\boldsymbol{\beta}_{jk}^{\star} \leftarrow Q_{jk}(\boldsymbol{\beta}_{jk}^{\star} \mid \boldsymbol{\beta}_{jk}^{(t)})$.

Compute acceptance probability $\alpha\left(\boldsymbol{\beta}_{jk}^{\star} \mid \boldsymbol{\beta}_{jk}^{(t)}\right)$.

if uniform draw $U(0,1) \leq \alpha\left(\boldsymbol{\beta}_{jk}^{\star} \mid \boldsymbol{\beta}_{jk}^{(t)}\right)$ **then**

$\boldsymbol{\beta}_{jk}^{(t+1)} \leftarrow \boldsymbol{\beta}_{jk}^{\star}$

else

$\boldsymbol{\beta}_{jk}^{(t+1)} \leftarrow \boldsymbol{\beta}_{jk}^{(t)}$.

end if

Generate $\boldsymbol{\tau}_{jk}^{(t+1)}$ analogously.

Output: Next state $\boldsymbol{\beta}_{jk}^{(t+1)}$, $\boldsymbol{\tau}_{jk}^{(t+1)}$.

Advantages of MCMC

- Access to the complete posterior distribution without requiring asymptotic considerations.
- Divide and conquer approach based on updating blocks of parameters separately allows handling very complex models having hundreds or thousands of parameters.
- Modular representation of hierarchically formulated statistical models where certain parts of the model can be replaced without affecting the other model components
- From the samples of the model parameters, we can determine not only inferences about these parameters themselves, but also inference for complex functionals of these parameters.

Introduction to MCMC Engines

- **MCMC engines** are software tools that automate the sampling process from complex posterior distributions in Bayesian analysis.
- **JAGS** (Just Another Gibbs Sampler) and **Stan** are two of the most popular MCMC engines.
- These tools simplify the implementation of MCMC methods, allowing researchers to focus on model building.

What is JAGS?

- **JAGS (Just Another Gibbs Sampler)** is an open-source program for performing Bayesian analysis using MCMC.
- JAGS is designed to work with the BUGS (Bayesian inference Using Gibbs Sampling) language, commonly used in Bayesian models.
- JAGS allows for hierarchical models, multivariate models, and complex likelihoods.
- It uses adaptive MCMC algorithms to sample from posterior distributions.

Key Features of JAGS

- JAGS is particularly well-suited for hierarchical models.
- Supports a wide range of distributions for likelihoods and priors.
- It is portable and cross-platform.
- Can be used in conjunction with R via the `rjags` package.
- JAGS focuses on Gibbs sampling, but supports other MCMC methods such as Metropolis-Hastings and slice sampling.

Example: JAGS Code for Simple Linear Regression

```
model {
   for (i in 1:N) {
       y[i] ~ dnorm(mu[i], tau)
      mu[i] <- beta0 + beta1 * x[i]
  }
  beta0 ~ dnorm(0, 0.0001)
  beta1 ~ dnorm(0, 0.0001)
  tau ~ dgamma(0.001, 0.001)
  sigma <- 1 / sqrt(tau)
}
```

- JAGS model specification for simple linear regression.
- Priors are assigned to the coefficients and variance.

What is Stan?

- **Stan** is a state-of-the-art platform for statistical modeling and high-performance computation using MCMC.
- Stan employs the **Hamiltonian Monte Carlo (HMC)** algorithm, which improves sampling efficiency in high-dimensional spaces.
- It also supports variational inference and other optimization methods.
- Stan uses a more expressive probabilistic programming language compared to BUGS.

Key Features of Stan

- Stan is known for its use of Hamiltonian Monte Carlo (HMC) and its variant, the No-U-Turn Sampler (NUTS), which result in faster convergence and better exploration of posterior distributions.
- Offers efficient handling of large models and high-dimensional parameter spaces.
- Stan integrates well with R, Python, and other programming environments.
- It provides a flexible and highly expressive modeling language.

Example: Stan Code for Simple Linear Regression

```
data {
  int<lower=0> N;
  vector[N] x;
  vector[N] y;
}
parameters {
  real alpha;
  real beta;
  real<lower=0> sigma;
}
model {
  y ~ normal(alpha + beta * x, sigma);
}
```

- Stan model specification for simple linear regression.
- Data, parameters, and the model structure are clearly separated.

JAGS vs. Stan

Feature	JAGS	Stan
Algorithms	Gibbs Sampling, Metropolis-Hastings	Hamiltonian Monte Carlo, NUTS
Language	BUGS	Custom Stan Language
Efficiency	Slower in high dimensions	Faster in high dimensions
Complexity	Easier for beginners	More complex, requires understanding of HMC
Inference	MCMC	MCMC, Variational Inference

When to Use JAGS or Stan

- **Use JAGS** if:
  - You are working with simpler models, particularly hierarchical or multivariate models.
  - You prefer a BUGS-like syntax or want a more straightforward learning curve.
- **Use Stan** if:
  - You are working with high-dimensional or complex models where efficient sampling is necessary.
  - You want to leverage the power of Hamiltonian Monte Carlo or variational inference.

JAGS Example

```
R> library("rjags")

R> ## Simulate data.
R> set.seed(123)
R> n <- 300
R> x <- runif(n, -3, 3)
R> y <- 1.2 + sin(x) + rnorm(n, sd = 0.2)

R> ## Create the design matrix for a cubic polynomial.
R> X_poly <- poly(x, 5)

R> ## Prepare data for JAGS.
R> data_jags <- list(
+    y = y,
+    X = as.matrix(X_poly),  ## Design matrix (poly of x).
+    n = n,
+    p = 5                   ## Number of predictors (degree 5 polynomial).
+  )
```

JAGS Example

```
R> ## JAGS model.
R> model_string <- "
+  model {
+    for (i in 1:n) {
+      y[i] ~ dnorm(mu[i], tau)
+      mu[i] <- beta0 + inprod(X[i,], beta[1:p])
+    }
+
+    # Priors for regression coefficients.
+    beta0 ~ dnorm(0, 0.0001)
+    for (j in 1:p) {
+      beta[j] ~ dnorm(0, 0.0001)
+    }
+
+    ## Prior for precision (inverse of variance).
+    tau ~ dgamma(0.001, 0.001)
+    sigma <- 1 / sqrt(tau)  ## Standard deviation.
```

JAGS Example

```
+  }
+  "

R> ## Write the model to a temporary file.
R> writeLines(model_string, con = "model_jags.txt")

R> ## Initial values.
R> inits <- function() {
+    list(beta0 = 0, beta = rnorm(5), tau = 1)
+  }

R> ## Parameters to monitor.
R> params <- c("beta0", "beta", "sigma")

R> ## Set up the JAGS model.
R> model <- jags.model("model_jags.txt", data = data_jags,
+    inits = inits, n.chains = 2)
```

JAGS Example

```
Compiling model graph
   Resolving undeclared variables
   Allocating nodes
Graph information:
   Observed stochastic nodes: 300
   Unobserved stochastic nodes: 7
   Total graph size: 2716

Initializing model

R> ## Burn-in.
R> update(model, 1000)

R> ## Sample from the posterior distribution.
R> samples <- coda.samples(model, variable.names = params, n.iter = 1000)
```

JAGS Example

```
R> print(head(samples))
[[1]]
Markov Chain Monte Carlo (MCMC) output:
Start = 1001
End = 1007
Thinning interval = 1
      beta[1]      beta[2]   beta[3]    beta[4]   beta[5]    beta0     sigma
[1,] 10.35490 -0.272916888 -6.694296  0.3230170 0.6115224 1.182533 0.1877252
[2,] 10.63288  0.205489998 -6.529839  0.1904117 0.5589678 1.153706 0.2051813
[3,] 10.71078 -0.224102077 -6.443891 -0.2078798 0.4833106 1.177085 0.2115810
[4,] 10.42182 -0.476634728 -6.546285  0.5148199 0.2397256 1.166886 0.1926633
[5,] 10.42216 -0.232314525 -6.625866 -0.2566665 0.6904027 1.194251 0.1966496
[6,] 10.62362 -0.222925444 -6.471449  0.1779714 0.4465330 1.160563 0.2060578
[7,] 10.15890 -0.006989872 -6.624863 -0.2234371 1.0190441 1.168644 0.2080756

[[2]]
Markov Chain Monte Carlo (MCMC) output:
```

JAGS Example

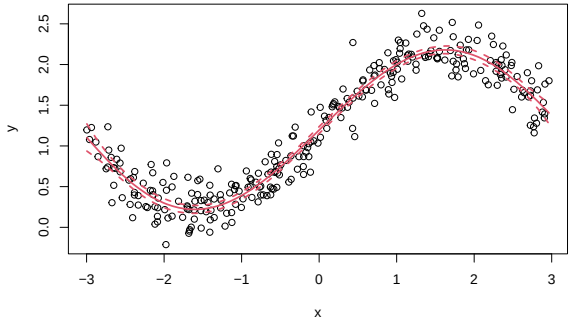
```
Start = 1001
End = 1007
Thinning interval = 1
       beta[1]     beta[2]   beta[3]      beta[4]   beta[5]    beta0     sigma
[1,] 10.516473 -0.07847717 -6.314681 -0.267959188 0.6945177 1.181283 0.1943936
[2,] 10.424636 -0.34317226 -6.329643 -0.025303217 0.7981460 1.188000 0.1916866
[3,] 10.500319 -0.21433109 -6.560139  0.050396654 0.7574165 1.155332 0.2052327
[4,] 10.428978 -0.48661691 -6.796730 -0.489256219 1.0974617 1.167023 0.1956970
[5,] 10.259558  0.02301548 -6.111242  0.004325772 0.6059063 1.174159 0.1860764
[6,] 10.687256  0.06717566 -6.747883  0.153628345 0.5067582 1.184894 0.2082724
[7,]  9.981962  0.06216834 -6.968661 -0.004328742 0.3941233 1.180452 0.2143863

attr(,"class")
[1] "mcmc.list"
```

JAGS Example

```
R> ## Compute fitted values.
R> fit <- lapply(samples, function(x) {
+    apply(x[, 1:6], 1, function(beta) drop(X_poly %*% beta[1:5]) + beta[6])
+  })
R> fit <- do.call("cbind", fit)
R> fit <- t(apply(fit, 1, quantile, probs = c(0.025, 0.5, 0.975)))
```

JAGS Example

```
R> par(mar = c(4, 4, 1, 1))
R> plot(x, y)
R> i <- order(x)
R> matplot(x[i], fit[i, ], type = "l",
+    lty = c(2, 1, 2), col = 2, lwd = 2, add = TRUE)
```



R package *bamlss*

- The *bamlss* package in R is designed for fitting Bayesian additive models for location, scale, and shape (BAMLSS).
- It allows users to specify complex models using flexible regression techniques.
- The package supports many response distributions and includes efficient MCMC samplers.
- Some common applications: non-linear regression, spatial models, and time series.

Key Features

- Supports **additive models** with a wide variety of link functions.
- Handles **location**, **scale**, and **shape** models, extending beyond standard GLMs.
- Inference is performed via **MCMC** methods.
- **Customizable priors** and model structure.
- **Built-in samplers** for efficient Bayesian estimation.
- Supports flexible **smooth terms** such as splines.

Example Continued

```
R> library("bamlss")

R> ## Bayesian linear model using IWLS proposals.
R> b <- bamlss(y ~ poly(x, 5))
AICc 160.5543 logPost -127.872 logLik -73.0854 edf 7.0000 eps 0.5933 iteration   1
AICc -18.6278 logPost -38.2813 logLik  16.5057 edf 7.0000 eps 0.1750 iteration   2
AICc -99.0825 logPost   1.9461 logLik  56.7330 edf 7.0000 eps 0.1013 iteration   3
AICc -112.008 logPost   8.4090 logLik  63.1959 edf 7.0000 eps 0.0416 iteration   4
AICc -112.294 logPost   8.5523 logLik  63.3392 edf 7.0000 eps 0.0066 iteration   5
AICc -112.295 logPost   8.5524 logLik  63.3393 edf 7.0000 eps 0.0001 iteration   6
AICc -112.295 logPost   8.5524 logLik  63.3393 edf 7.0000 eps 0.0000 iteration   7
AICc -112.295 logPost   8.5524 logLik  63.3393 edf 7.0000 eps 0.0000 iteration   7
elapsed time:  0.02sec
Starting the sampler...

|                    |   0%  0.71sec
|*                   |   5%  0.68sec  0.04sec
```

Example Continued

```
|**                  |  10%  0.58sec  0.06sec
|***                 |  15%  0.56sec  0.10sec
|****                |  20%  0.52sec  0.13sec
|*****               |  25%  0.51sec  0.17sec
|******              |  30%  0.47sec  0.20sec
|*******             |  35%  0.44sec  0.24sec
|********            |  40%  0.41sec  0.27sec
|*********           |  45%  0.38sec  0.31sec
|**********          |  50%  0.35sec  0.35sec
|***********         |  55%  0.32sec  0.39sec
|************        |  60%  0.28sec  0.42sec
|*************       |  65%  0.25sec  0.46sec
|**************      |  70%  0.21sec  0.49sec
|***************     |  75%  0.20sec  0.60sec
|****************    |  80%  0.16sec  0.64sec
|*****************   |  85%  0.12sec  0.68sec
|******************  |  90%  0.08sec  0.71sec
```

Example Continued

```
|****************** |  95%  0.04sec  0.75sec
|*******************| 100%  0.00sec  0.79sec

R> summary(b)
Call:
bamlss(formula = y ~ poly(x, 5))
---
Family: gaussian
Link function: mu = identity, sigma = log
*---
Formula mu:
---
y ~ poly(x, 5)
-
Parametric coefficients:
               Mean     2.5%      50%    97.5% parameters
(Intercept)  1.17247  1.15107  1.17215  1.19451      1.173
```

Example Continued

```
poly(x, 5)1 10.49787 10.09569 10.49774 10.89576     10.495
poly(x, 5)2 -0.20047 -0.56834 -0.19501  0.17026     -0.193
poly(x, 5)3 -6.50922 -6.87121 -6.51020 -6.12913     -6.511
poly(x, 5)4  0.04406 -0.36683  0.03905  0.43782      0.047
poly(x, 5)5  0.67587  0.27029  0.67182  1.07189      0.673
-
Acceptance probability:
      Mean 2.5% 50% 97.5%
alpha    1    1   1     1
---
Formula sigma:
---
sigma ~ 1
-
Parametric coefficients:
              Mean   2.5%    50%  97.5% parameters
(Intercept) -1.620 -1.692 -1.621 -1.537      -1.63
-
```

Example Continued

```
Acceptance probability:
        Mean   2.5%    50% 97.5%
alpha 0.9774 0.8160 1.0000     1
---
Sampler summary:
-
DIC = -112.6508 logLik = 59.8167 pd = 6.9826
runtime = 0.79
---
Optimizer summary:
-
AICc = -112.295 edf = 7 logLik = 63.3393
logPost = 8.5524 nobs = 300 runtime = 0.021
```

Predict method, as for `lm()` or `glm()`.

```
R> fit2 <- predict(b, model = "mu", FUN = c95)
```

Example Continued

```
R> par(mar = c(4, 4, 1, 1))
R> plot(x, y)
R> matplot(x[i], fit[i, ], type = "l", lty = c(2, 1, 2), col = 2, lwd = 2, add = TRUE)
R> matplot(x[i], fit2[i, ], type = "l", lty = c(2, 1, 2), col = 4, lwd = 2, add = TRUE)
```