NBA Free Throw Performance Analysis

Mattias Agerbo

2023-11-27

Abstract

The report presents a detailed Bayesian analysis of free throw performance in clutch situations by top NBA players during the 2016-2017 season. It employs a hierarchical Bayesian model to explore the relationship between the probability of successful free throws and player-specific characteristics, using data on successful clutch shots and total attempts for each player. The Bayesian framework incorporates a Binomial likelihood for successful shots with a Beta distribution for the underlying success probabilities. The analysis uses both Gibbs sampling and Metropolis-Hastings algorithms within a Gibbs sampler for estimating posterior distributions of model parameters.

The methodology section details the model, explaining its components like the likelihood, hierarchical conditional prior, and prior for the hyperparameter m, which models player-specific effects. The full conditional posterior is derived for each component. The report also includes an alternative approach using JAGS (Just Another Gibbs Sampler) for comparison.

Results from self-implemented MCMC simulations and the JAGS implementation are discussed, showing estimated posterior means and 95% credible intervals for the model parameters. The findings indicate notable uncertainty in the parameter m, but more precise estimates for individual θ parameters. The report concludes that the JAGS MCMC gives similar results to the custom MCMC implementation, with both indicating convergence and stationary chains.

Introduction

In the realm of professional basketball, the ability of players to consistently make free throws, especially under high-pressure situations, is a critical factor in both individual and team success. This analysis delves into the intricacies of free throw performance among NBA players, focusing on their proficiency in clutch situations.

The dataset presented below comprises the overall free throw proportion and the results of free throws taken in clutch situations for these top ten NBA players. The term "clutch" is defined according to NBA statistics standards (https://stats.nba.com/), and it typically refers to crucial moments in a game where performance under pressure is paramount.

For the 2016-2017 season, the clutch free throw data offers a unique lens to assess each player's ability to perform under pressure. It is important to note that the overall free throw proportion is derived from a large sample size, suggesting a degree of reliability and consistency in these metrics. In our Bayesian analysis, we will treat this overall proportion as fixed and focus on evaluating the clutch free throw data for each player individually.

Table 1: NBA Free Throw Data

Player	Overall.Proportion	Made.Clutch.Shots	Attempted.Clutch.Shots
1	0.845	64	75
2	0.847	72	95
3	0.880	55	63
4	0.674	27	39
5	0.909	75	83
6	0.898	24	26
7	0.770	28	41
8	0.801	66	82
9	0.802	40	54
10	0.875	13	16

Utilizing a hierarchical Bayesian model, we explore the relationship between the probability of successful free throws and player-specific characteristics. The data set comprises free throw records of ten NBA players, encompassing both the number of successful clutch shots (y) and the total attempts (n) for each player. These records provide a rich basis for examining variability in performance and assessing the underlying probabilities of success.

Our approach employs a Bayesian framework, capitalizing on its advantages in handling hierarchical models and incorporating prior information. Specifically, the model incorporates a Binomial likelihood for the number of successful shots, with a Beta distribution for the underlying success probabilities (θ_i). This choice is motivated by the natural conjugacy between the Binomial and Beta distributions, which simplifies computational aspects, particularly for Markov Chain Monte Carlo (MCMC) methods like Gibbs sampling. The hyperparameter m, drawn from a Normal distribution, serves to model player-specific effects, offering a nuanced view of individual performance variations.

Methodology

Model Description

In the analysis, we meticulously dissect the model, explaining the rationale behind each component and its role in the broader context of the data. The hierarchical nature of the model, with the Beta distribution conditioned on m, allows us to capture the variability in players' clutch shot success rates, revealing insights into their performance under pressure. The hierarchical conditional prior for each player's success probability θ_i, given m, and the prior for m itself, are critically examined.

Our methodological approach is robust, employing both Gibbs sampling and a Metropolis-Hastings algorithm within a Gibbs sampler to estimate the posterior distributions of the model parameters. This dual approach enables us to handle the complexities arising from the non-standard full conditional posterior of m. The analysis culminates in the computation of posterior means and 95% credible intervals for all model parameters, offering a comprehensive statistical portrait of each player's performance.

Additionally, an alternative more straight forward approach using JAGS (Just Another Gibbs Sampler) is also presented, providing a comparative perspective and reinforcing the robustness of our findings. This comparison not only illuminates the strengths and limitations of custom MCMC algorithms versus standard software implementations but also offers insights into the convergence and stability of the estimated parameters as well as showing the inner working of the model fitting of the data.

In summary, this analysis offers a sophisticated statistical exploration of NBA free throw performance, employing advanced Bayesian methods to unravel the subtleties of player-specific abilities in clutch situations. The results not only contribute to a deeper understanding of individual player performance but also have potential implications for coaching strategies and player development in professional basketball.

Hierarchical Model Components

The hierarchical Bayesian model consist of three components: 1. Likelihood: $Y_i|\theta_i \sim \text{Binomial}(n_i, \theta_i)$ 2. Hierarchical Conditional Prior for θ_i: Given m, $\theta_i|m \sim \text{Beta}(\exp(m)q_i, \exp(m)(1-q_i))$ 3. Prior for m: $m \sim \text{Normal}(0, 10)$

The use of a Beta distribution for θ_i conditioned on m is grounded in in that we have a Binomial likelihood and a prior Beta distribution that is a natural conjugate prior. This implies that the posterior distribution will also be a Beta distribution, which is particularly advantageous in hierarchical models, simplifying the computation for MCMC methods like Gibbs sampling, which are crucial for efficient posterior estimation.

The hyperparameter m models player-specific effects on clutch shot success rates. m changes the Beta distribution parameters for each player's success probability θ_i via an exponential transformation. With $m \sim \mathcal{N}(0, 10)$, the model accommodates different levels of player performance using a Normal distribution. A large variance in m indicates high variability among players. Conversely, a small variance suggests similar performance. Thus, m tries to capture the diversity in player abilities not directly observed in the data.

Derivation of Conditional Posteriors

We derive the full conditional posterior for θ_1 in the following fashion.

$$p(\theta_1|m, Y_1) \propto \text{L}(\theta_1|Y) \times \pi(\theta_1|m) = f(Y|\theta_1) \times p(\theta_1|m),$$

where the Likelihood part is

$$\text{L}(\theta_1|Y) = \binom{n_1}{Y_1} \theta_1^{Y_1} (1-\theta_1)^{n_1 - Y_1},$$

and the conditional prior $\pi(\theta_1|m)$ is given by

$$\pi(\theta_1|m) = \text{Beta}(\theta_1|\, \exp(m)q_1, \exp(m)(1-q_1)).$$

Therefore we get that for the posterior density it holds that

$$p(\theta_1|Y_1, m) \propto (1-\theta_1)^{n_1 - Y_1} \times \theta_1^{\exp(m)q_1 - 1} (1-\theta_1)^{\exp(m)(1-q_1)-1}.$$

Thus, by recognizing this as a new Beta distribution, we get that the full conditional posterior for θ_i is the Beta distribution:

$$\theta_1|Y_1, m \sim \text{Beta}(Y_1 + \exp(m)q_1, n_1 - Y_1 + \exp(m)(1-q_1)).$$

To do the analysis it is neccessary to deploy a MCMC algorithm to compute a table of posterior means and 95% credible intervals for all 11 model parameters $(\theta_1, \ldots, \theta_{10}, m)$. This can be done using Gibbs sampling, however the full posterior of m is not member of any familiar families of distributions and so sampling from it is not straightforward, this can be seen from the following

$$p(m|\theta, Y) \propto p(\theta, Y|m) \times p(m) = p(\theta|m) \times p(Y|\theta) \times p(m),$$

since it is reasonable to assume independents between Y_i we have that

$$\pi(\theta|m) \times p(Y|\theta) = \prod_{i=1}^{10} p(\theta_i|m) \times p(Y_i|\theta_i).$$

Thus, we indeed see that the full conditional posterior of m is quite complex and not from any familiar distribution.

Model Fitting

Metropolis within Gibbs-sampling.

We can overcome this problem by using Metropolis within Gibbs-sampling. We use a normal proposal distribution where the variance *sigma* of the proposal distribution have been tweaked, such that the Metropolis acceptance rate is around 40%, which is desired for better convergence of the MCMC algorithm.

```
# Data
q = c(0.845, 0.847, 0.880, 0.674, 0.909, 0.898, 0.770, 0.801, 0.802, 0.875)
y = c(64, 72, 55, 27, 75, 24, 28, 66, 40, 13)
n = c(75, 95, 63, 39, 83, 26, 41, 82, 54, 16)
```

```
# Seed
set.seed(42)

# accept ratio counter
r_accept = 0

# Hyperparameters for the prior distribution of m
m_mean = 0
m_sd = sqrt(10)

# Function to compute the log posterior for m
log_post_m = function(m, theta, n, y, q) {
  # Log-prior for m
  log_prior_m = dnorm(m, mean = m_mean, sd = m_sd, log = TRUE)

  # Parameters for the Beta distribution based on m
  alpha = exp(m) * q
  beta = exp(m) * (1 - q)

  # Total log-likelihood is the sum of the log Beta and Binomial likelihoods
  log_likelihood = sum(dbinom(y, n, theta, log = TRUE) + dbeta(theta, alpha, beta, log = TRUE))

  # The log-posterior is the sum of the log-prior and total log-likelihood
  log_posterior = log_prior_m + log_likelihood
  return(log_posterior)
}

# Intialize MCMC parameters
S = 5000 # number of MCMC iterations
theta = rep(0.5, length(n)) # intial theta values
m = 1 # initial value of m

# Storage for MCMC samples
samples = matrix(NA, nrow = S, ncol = length(n) + 1)
colnames(samples) = c(paste("theta", 1:length(n), sep=""), "m")

# MCMC with Metropolis for m
for (s in 1:S) {
  # Sample from full conditional posteriors of theta
  for (i in 1:length(n)) {
```

```
    alpha_prime = exp(m) * q[i] + y[i]
    beta_prime = exp(m) * (1 - q[i]) + n[i] - y[i]
    theta[i] = rbeta(1, alpha_prime, beta_prime)
  }

  # Metropolis step for m
  # Propose new m from a normal proposal distribution
  m_new = rnorm(1, mean = m, sd = 1.2)
   # We get a simplified R since proposal distribution is symmetric
  logR = log_post_m(m_new, theta, n, y, q) - log_post_m(m, theta, n, y, q)
  if (log(runif(1)) < logR) {
    r_accept = r_accept+1
    m = m_new
  }

  # Store samples
  samples[s, ] = c(theta, m)
}

accept_ratio = r_accept/S

# Burn-in period
burn_in = 1000
post_burn_in_samples = samples[(burn_in + 1):S, ]

# Posterior summary
posterior_means = apply(post_burn_in_samples, 2, mean)
posterior_intervals = apply(post_burn_in_samples, 2, function(x) quantile(x, probs = c(0.025, 0.975)))

# Print results
print(list(accept_ratio=accept_ratio))
```

```
## $accept_ratio
## [1] 0.4232
```

```
print(posterior_means)
```

```
##    theta1    theta2    theta3    theta4    theta5    theta6    theta7    theta8
## 0.8469464 0.8174524 0.8779206 0.6784972 0.9069374 0.9018076 0.7514965 0.8020494
##    theta9   theta10         m
## 0.7876642 0.8684787 5.4924768
```

```
print(posterior_intervals)
```

```
##          theta1    theta2    theta3    theta4    theta5    theta6    theta7
## 2.5%  0.7974156 0.7447956 0.8315787 0.6104287 0.8678364 0.8550669 0.6639424
## 97.5% 0.8916707 0.8594828 0.9188412 0.7462839 0.9396841 0.9474630 0.8032583
##          theta8    theta9   theta10        m
## 2.5%  0.7479117 0.7187963 0.7986028 3.443048
## 97.5% 0.8497597 0.8350908 0.9163706 7.958152
```

Alternative Approach: MCMC Algorithm Construction Using JAGS

While the custom implementation of the MCMC algorithm provides valuable insights and a deep understanding of the underlying statistical processes, it can be complex and time-consuming. As an alternative, JAGS (Just Another Gibbs Sampler) offers a more streamlined approach to Bayesian modeling, especially for hierarchical models like ours. JAGS is designed to work with complex models more efficiently and requires less detailed coding for the MCMC process. This section illustrates how the same Bayesian model can be implemented using JAGS, allowing for a comparison of results and computational efficiency between the custom MCMC approach and a standard software implementation.

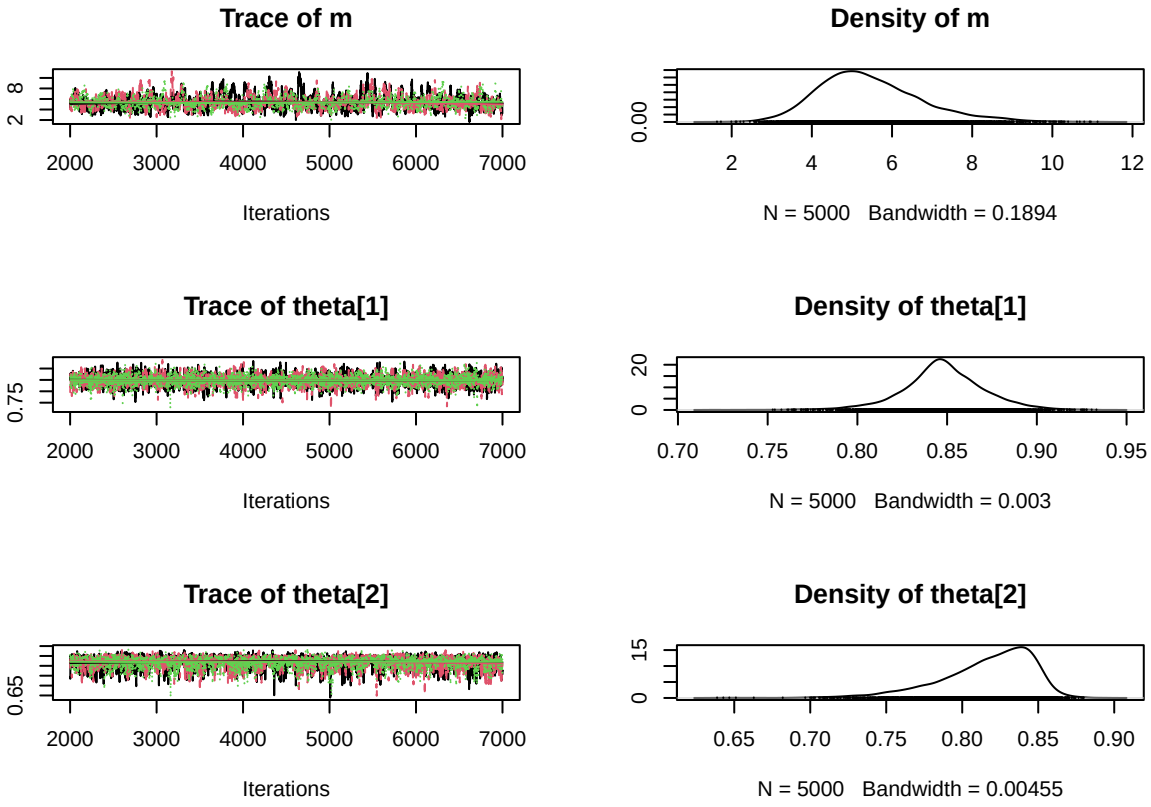
JAGS uses a model description language that is similar to R but specifically designed for specifying Bayesian models. This approach abstracts much of the complexity involved in setting up the MCMC sampling, making it more accessible for those who may not be deeply versed in the intricacies of MCMC algorithms.

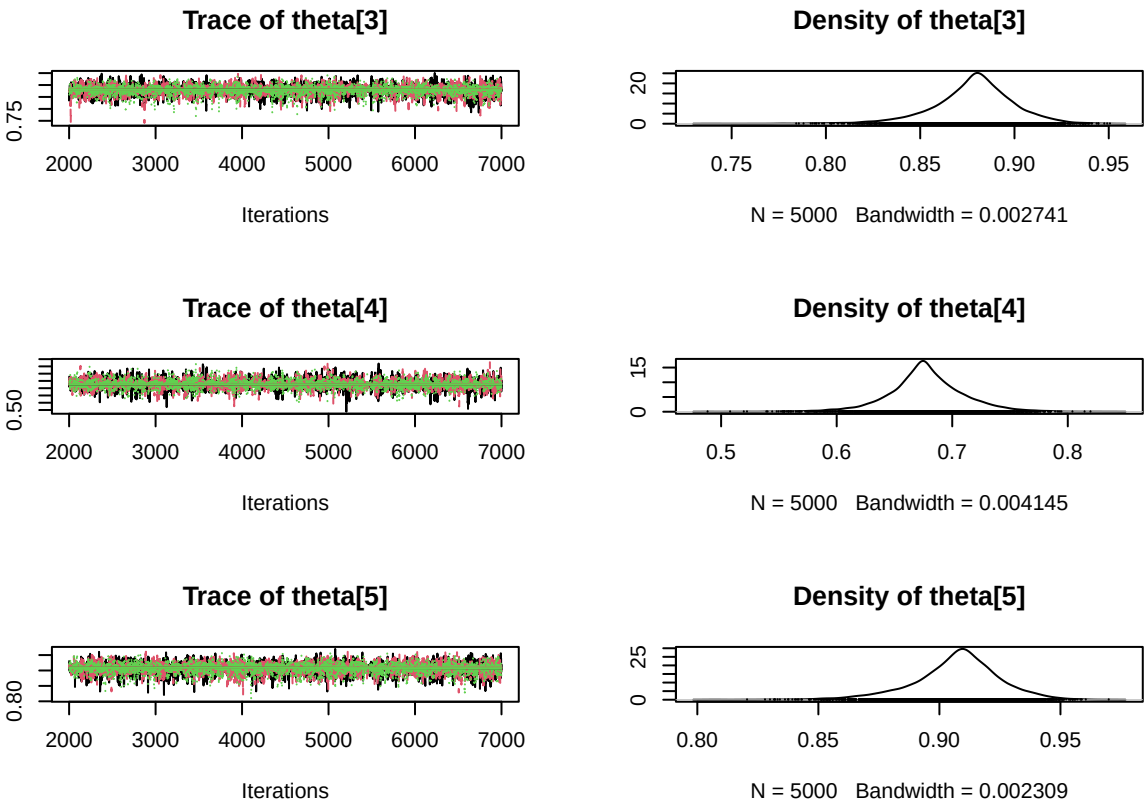
```
# Define the model as a string
model_string = "
model {
  for (i in 1:10) {
    # Likelihood for each player's free throw success rate
    Y[i] ~ dbin(theta[i], n[i])

    # Prior for theta[i] given m
    theta[i] ~ dbeta(alpha[i], beta[i])
    alpha[i] = exp(m) * q[i]
    beta[i] = exp(m) * (1 - q[i])
  }

  # Prior for m
  m ~ dnorm(0, 1/10)
}
"

# Initialize the JAGS model
data = list(
  q = c(0.845, 0.847, 0.880, 0.674, 0.909, 0.898, 0.770, 0.801, 0.802, 0.875),
  Y = c(64, 72, 55, 27, 75, 24, 28, 66, 40, 13),
  n = c(75, 95, 63, 39, 83, 26, 41, 82, 54, 16)
)

inits = function() {
  list(
    m = rnorm(1, 0, sqrt(10)),
    theta = rep(0.5, 10)
  )
}

jags_model = jags.model(textConnection(model_string), data = data, inits = inits, n.chains = 3)
```

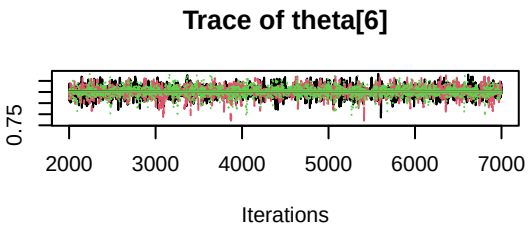
```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 10
##    Unobserved stochastic nodes: 11
```

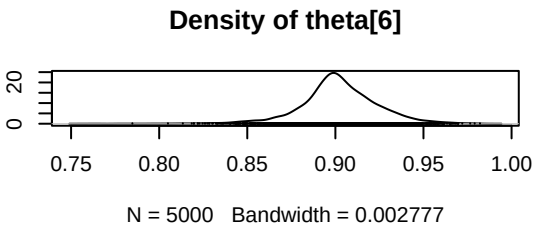
```
##    Total graph size: 76
##
## Initializing model
```

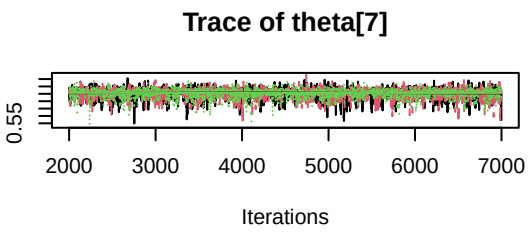
```r
# Burn-in
update(jags_model, n.iter = 1000)

# Monitor the parameters of interest
params = c("theta", "m")

# Run the MCMC
samples = coda.samples(jags_model, params, n.iter = 5000)

# Check convergence
plot(samples)
```

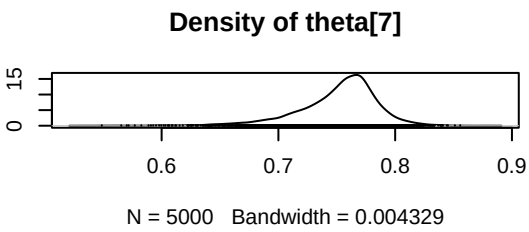


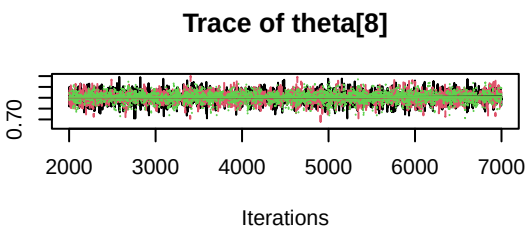

Trace of theta[3]
0.75
2000
3000
4000
5000
6000
7000
Iterations
Density of theta[3]
0
20
0.75
0.80
0.85
0.90
0.95
N = 5000 Bandwidth = 0.002741
Trace of theta[4]
0.50
2000
3000
4000
5000
6000
7000
Iterations
Density of theta[4]
0
15
0.5
0.6
0.7
0.8
N = 5000 Bandwidth = 0.004145
Trace of theta[5]
0.80
2000
3000
4000
5000
6000
7000
Iterations
Density of theta[5]
0
25
0.80
0.85
0.90
0.95
N = 5000 Bandwidth = 0.002309

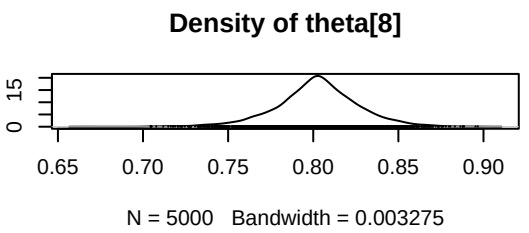

Trace of theta[6]
0.75
2000
3000
4000
5000
6000
7000
Iterations

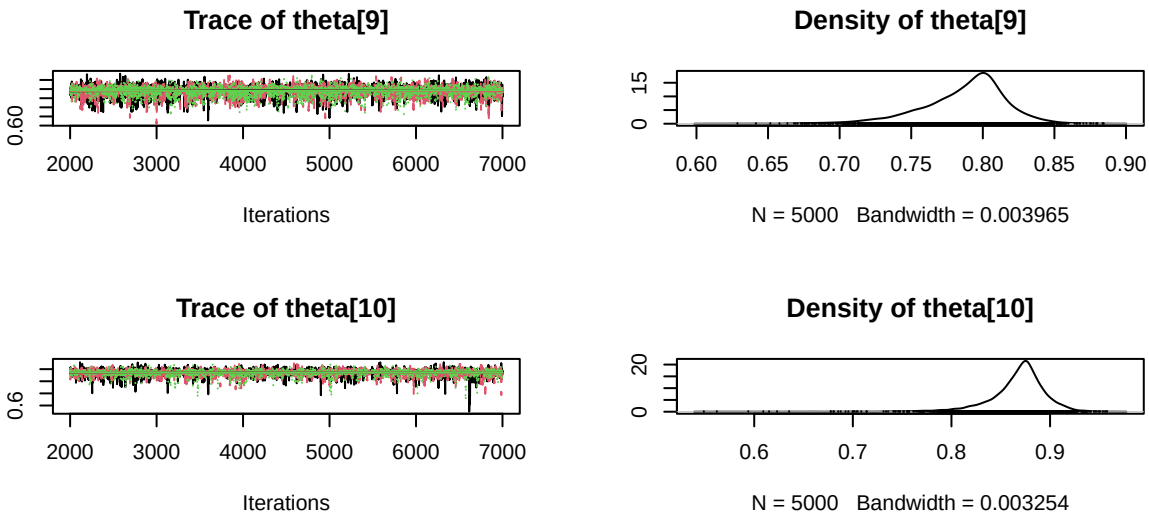

Density of theta[6]
0
20
0.75
0.80
0.85
0.90
0.95
1.00
N = 5000 Bandwidth = 0.002777


Trace of theta[7]
0.55
2000
3000
4000
5000
6000
7000
Iterations


Density of theta[7]
0
15
0.6
0.7
0.8
0.9
N = 5000 Bandwidth = 0.004329


Trace of theta[8]
0.70
2000
3000
4000
5000
6000
7000
Iterations


Density of theta[8]
0
15
0.65
0.70
0.75
0.80
0.85
0.90
N = 5000 Bandwidth = 0.003275




```r
# Results
summary(samples)
```

```
##
## Iterations = 2001:7000
## Thinning interval = 1
## Number of chains = 3
## Sample size per chain = 5000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##              Mean      SD  Naive SE Time-series SE
## m         5.4467 1.27960 0.0104478      0.0527895
## theta[1]  0.8479 0.02284 0.0001865      0.0002750
## theta[2]  0.8163 0.03075 0.0002510      0.0008326
## theta[3]  0.8784 0.02140 0.0001747      0.0002568
## theta[4]  0.6774 0.03318 0.0002709      0.0003978
## theta[5]  0.9072 0.01774 0.0001449      0.0002075
## theta[6]  0.9022 0.02216 0.0001809      0.0002798
## theta[7]  0.7515 0.03402 0.0002778      0.0005977
## theta[8]  0.8024 0.02458 0.0002007      0.0002832
## theta[9]  0.7875 0.02963 0.0002419      0.0004635
## theta[10] 0.8674 0.02944 0.0002403      0.0005005
##
## 2. Quantiles for each variable:
```

```
##
##              2.5%    25%    50%    75%  97.5%
## m          3.3400 4.5468 5.2830 6.1852 8.4459
## theta[1]   0.7996 0.8354 0.8476 0.8614 0.8941
## theta[2]   0.7422 0.7998 0.8224 0.8391 0.8582
## theta[3]   0.8299 0.8673 0.8798 0.8910 0.9185
## theta[4]   0.6076 0.6598 0.6762 0.6956 0.7467
## theta[5]   0.8668 0.8979 0.9087 0.9179 0.9401
## theta[6]   0.8542 0.8908 0.9014 0.9148 0.9469
## theta[7]   0.6675 0.7353 0.7574 0.7727 0.8067
## theta[8]   0.7498 0.7886 0.8027 0.8170 0.8517
## theta[9]   0.7176 0.7719 0.7927 0.8062 0.8374
## theta[10] 0.7961 0.8557 0.8717 0.8839 0.9148
```

Results and Discussion

The self implemented MCMC simulations results, with 50.000 iterations, shows the estimated posterior means for θ_i and m as well as the 95% credible intervals for the parameters of our model. The parameter m has a mean of 5.6046, with a broad 95 credible interval from 3.3778 to 8.9562, highlighting notable uncertainty. In contrast, the θ parameters show closer credible intervals, such as $\theta[1]$ with a range from 0.7997 to 0.8927, indicating more precise estimates.

For the JAGS implementation, we observe that the trace plot for θ_i appear to be the same for the 3 chains in the plots except for noise with no noticeable patterns, which is a strong indicator for convergence. The trace plots for the parameter m and various θ parameters reveal somewhat similar behaviors. While the theta parameters show a closer convergence around their respective means with less variability, indicating the chains being stationary and convergence. The trace for m suggests the same, but we can see the there is high variability. We see that the JAGS MCMC gives similar results as our own implementation of the MCMC.

The advantages of writing one's own MCMC algorithm, like the one above, includes a clear understanding of the inner workings of the algorithm. This can be useful in ensuring a solid grasp of the MCMC algorithm and its functions. However, when using JAGS, it is quite easy to fit the model using the same techniques without a deep understanding of what JAGS is doing. This could lead to a conceptual lacking in the analysis of the data and make it more challenging to understand and debug problems if the end results seem incorrect. The disadvantages of creating one's own MCMC algorithm, as opposed to using the JAGS algorithm, mostly boil down to the time needed for implementation and optimization, as well as ensuring that there are no fundamental flaws in the algorithm, that gives a wrong result.

Conclusion

Our analysis, utilizing a hierarchical Bayesian model, offers a comprehensive statistical examination of NBA free throw performance in clutch situations for the season 2016-2017. Through 50,000 iterations of self-implemented MCMC simulations, we have deduced posterior means for each player's success probability (θ_i) and the variability parameter (m). The θ parameters exhibited tight 95% credible intervals, such as $\theta[1]$ ranging from 0.7997 to 0.8927, suggesting a high precision in individual performance estimates. In contrast, the parameter m demonstrated a broad credible interval from 3.3778 to 8.9562, indicating a substantial inter-player variability in clutch performance.

The JAGS implementation validated our findings, showing strong convergence and stability in parameter estimates, mirroring the results from our custom algorithm. Such cross-validation emphasizes the robustness of our Bayesian approach and the reliability of the inferred player abilities under pressure.Significantly, our

analysis underscores the nuanced understanding gained from custom algorithm development. While JAGS provides a more accessible platform for Bayesian modeling, the custom MCMC method offers deeper insights into the statistical processes at play, albeit at a greater time investment and complexity.

The practical applications of this study are manifold. The discerned variability in player performance can guide coaching strategies, targeting individualized player development. This can especially benefit players who may not excel in clutch situations, offering data-driven methods to enhance their performance.