Introduction to Bayesian statistics with R
2. Markov chains Monte Carlo (MCMC)

Olivier Gimenez

last updated: 2025-03-10

Get posteriors with Markov chains Monte Carlo (MCMC) methods

Back to the Bayes' theorem

- Bayes inference is easy! Well, not so easy in real-life applications.

Back to the Bayes' theorem

- Bayes inference is easy! Well, not so easy in real-life applications.
- The issue is in $\Pr(\theta \mid \text{data}) = \dfrac{\Pr(\text{data} \mid \theta)\ \Pr(\theta)}{\Pr(\text{data})}$

Back to the Bayes' theorem

- Bayes inference is easy! Well, not so easy in real-life applications.
- The issue is in $\Pr(\theta \mid \text{data}) = \dfrac{\Pr(\text{data} \mid \theta)\,\Pr(\theta)}{\Pr(\text{data})}$
- $\Pr(\text{data}) = \int L(\text{data} \mid \theta)\,\Pr(\theta) d\theta$ is a N-dimensional integral if $\theta = \theta_1, \ldots, \theta_N$

Back to the Bayes' theorem

- Bayes inference is easy! Well, not so easy in real-life applications.
- The issue is in $\Pr(\theta \mid \text{data}) = \dfrac{\Pr(\text{data} \mid \theta)\ \Pr(\theta)}{\Pr(\text{data})}$
- $\Pr(\text{data}) = \int L(\text{data} \mid \theta)\, \Pr(\theta) d\theta$ is a N-dimensional integral if $\theta = \theta_1, \ldots, \theta_N$
- Difficult, if not impossible to calculate!

Brute force approach via numerical integration

- Deer data

```
y <- 19 # nb of success
n <- 57 # nb of attempts
```

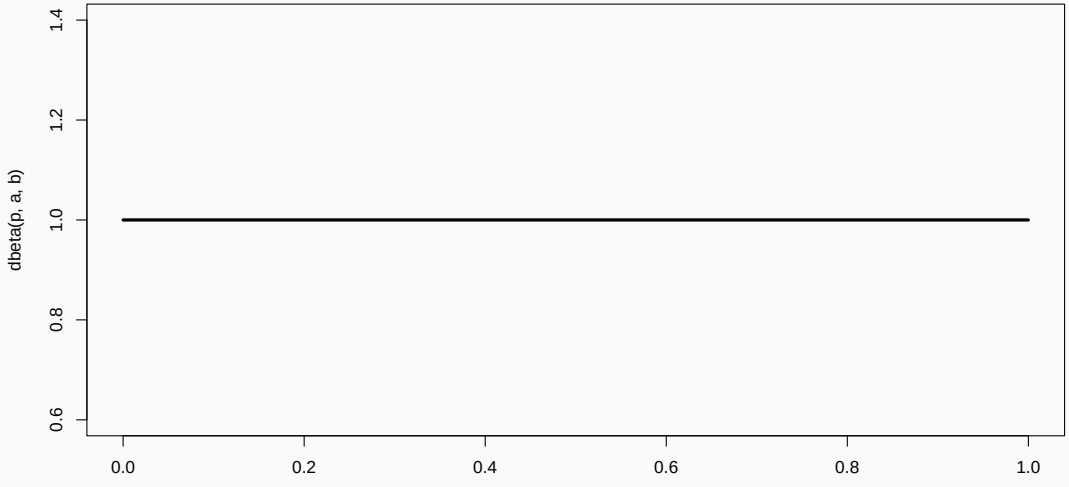
- Likelihood Binomial$(57, \theta)$
- Prior Beta$(a = 1, b = 1)$

Beta prior

```
a <- 1; b <- 1; p <- seq(0,1,.002)
```

Beta prior

```
plot(p, dbeta(p,a,b), type='l', lwd=3)
```



Apply Bayes theorem

- Likelihood times the prior: $\Pr(\text{data} \mid \theta)\ \Pr(\theta)$

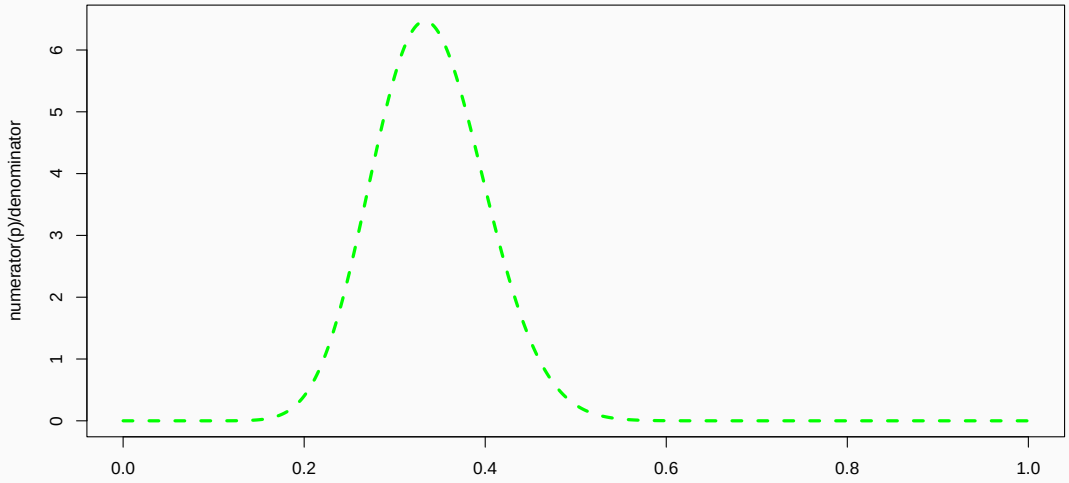
```
numerator <- function(p) dbinom(y,n,p)*dbeta(p,a,b)
```

- Averaged likelihood: $\Pr(\text{data}) = \int L(\theta \mid \text{data})\ \Pr(\theta) d\theta$

```
denominator <- integrate(numerator,0,1)$value
```

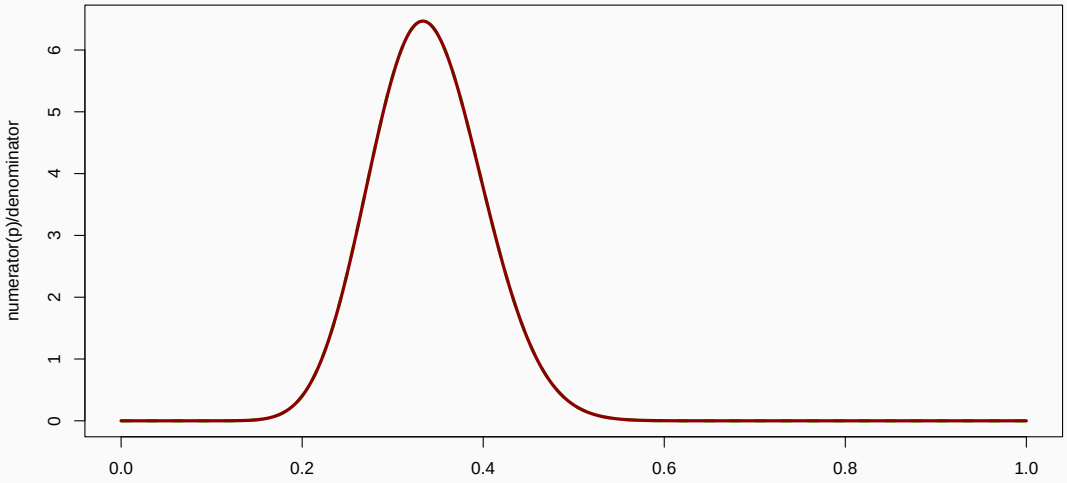
Posterior inference via numerical integration

```
plot(p, numerator(p)/denominator,type="l", lwd=3, col="green", lty=2)
```



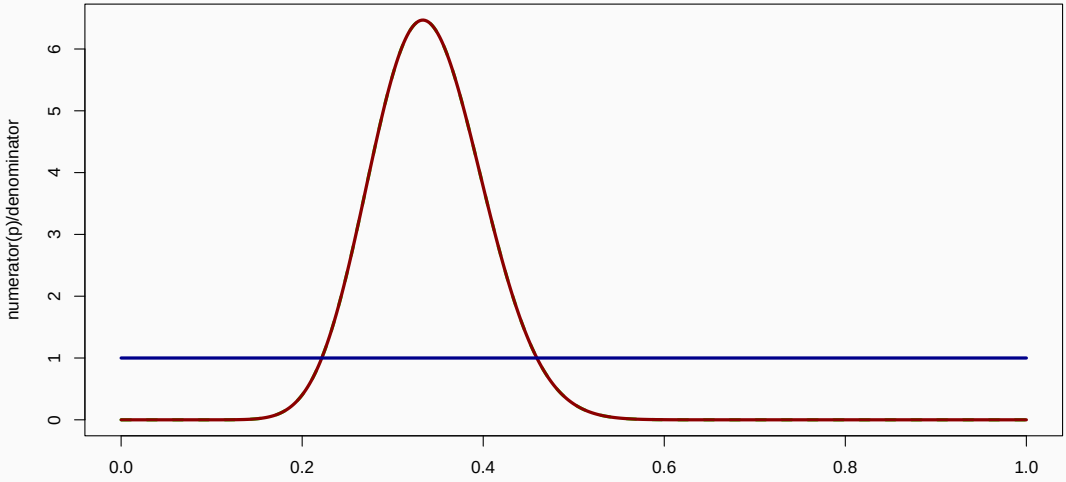
Superimpose explicit posterior distribution (Beta formula)

```
lines(p, dbeta(p,y+a,n-y+b), col='darkred', lwd=3)
```



And the prior

```
lines(p, dbeta(p,a,b), col='darkblue', lwd=3)
```



What if multiple parameters, like in a simple linear regression?

- Example of a linear regression with parameters α, β and σ to be estimated.

What if multiple parameters, like in a simple linear regression?

- Example of a linear regression with parameters α, β and σ to be estimated.
- Bayes' theorem says:

$$P(\alpha, \beta, \sigma \mid \text{data}) = \frac{P(\text{data} \mid \alpha, \beta, \sigma)\, P(\alpha, \beta, \sigma)}{\iiint P(\text{data} \mid \alpha, \beta, \sigma)\, P(\alpha, \beta, \sigma)\, d\alpha\, d\beta\, d\sigma}$$

What if multiple parameters, like in a simple linear regression?

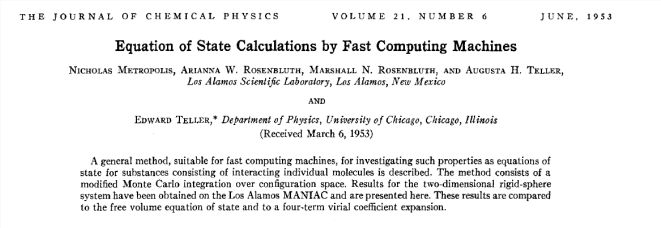
- Example of a linear regression with parameters α, β and σ to be estimated.
- Bayes' theorem says:

$$P(\alpha, \beta, \sigma \mid \text{data}) = \frac{P(\text{data} \mid \alpha, \beta, \sigma)\, P(\alpha, \beta, \sigma)}{\iiint P(\text{data} \mid \alpha, \beta, \sigma)\, P(\alpha, \beta, \sigma)\, d\alpha\, d\beta\, d\sigma}$$

- Do we really wish to calculate a 3D integral?

Bayesian computation

- In the early 1990s, statisticians rediscovered work from the 1950's in physics.



THE JOURNAL OF CHEMICAL PHYSICS VOLUME 21, NUMBER 6 JUNE, 1953

Equation of State Calculations by Fast Computing Machines

NICHOLAS METROPOLIS, ARIANNA W. ROSENBLUTH, MARSHALL N. ROSENBLUTH, AND AUGUSTA H. TELLER,
Los Alamos Scientific Laboratory, Los Alamos, New Mexico

AND

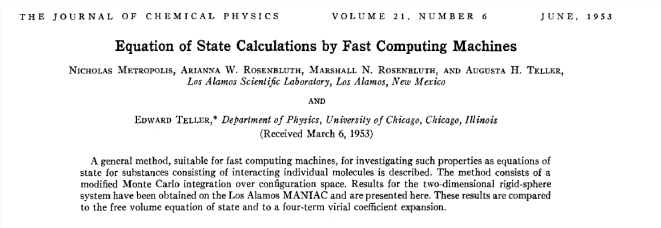
EDWARD TELLER,* *Department of Physics, University of Chicago, Chicago, Illinois*

(Received March 6, 1953)

A general method, suitable for fast computing machines, for investigating such properties as equations of state for substances consisting of interacting individual molecules is described. The method consists of a modified Monte Carlo integration over configuration space. Results for the two-dimensional rigid-sphere system have been obtained on the Los Alamos MANIAC and are presented here. These results are compared to the free volume equation of state and to a four-term virial coefficient expansion.

Bayesian computation

- In the early 1990s, statisticians rediscovered work from the 1950's in physics.



THE JOURNAL OF CHEMICAL PHYSICS VOLUME 21, NUMBER 6 JUNE, 1953

Equation of State Calculations by Fast Computing Machines

NICHOLAS METROPOLIS, ARIANNA W. ROSENBLUTH, MARSHALL N. ROSENBLUTH, AND AUGUSTA H. TELLER,
Los Alamos Scientific Laboratory, Los Alamos, New Mexico

AND

EDWARD TELLER,* *Department of Physics, University of Chicago, Chicago, Illinois*

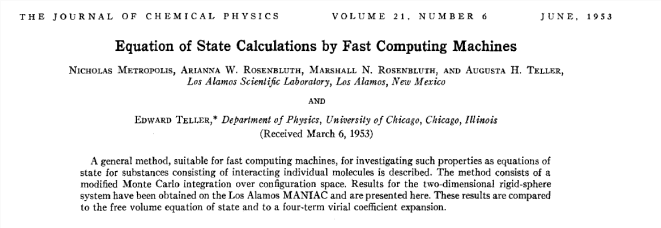
(Received March 6, 1953)

A general method, suitable for fast computing machines, for investigating such properties as equations of state for substances consisting of interacting individual molecules is described. The method consists of a modified Monte Carlo integration over configuration space. Results for the two-dimensional rigid-sphere system have been obtained on the Los Alamos MANIAC and are presented here. These results are compared to the free volume equation of state and to a four-term virial coefficient expansion.

- Use stochastic simulation to draw samples from posterior distributions.

Bayesian computation

- In the early 1990s, statisticians rediscovered work from the 1950's in physics.



THE JOURNAL OF CHEMICAL PHYSICS VOLUME 21, NUMBER 6 JUNE, 1953

Equation of State Calculations by Fast Computing Machines

NICHOLAS METROPOLIS, ARIANNA W. ROSENBLUTH, MARSHALL N. ROSENBLUTH, AND AUGUSTA H. TELLER,
Los Alamos Scientific Laboratory, Los Alamos, New Mexico

AND

EDWARD TELLER,* *Department of Physics, University of Chicago, Chicago, Illinois*

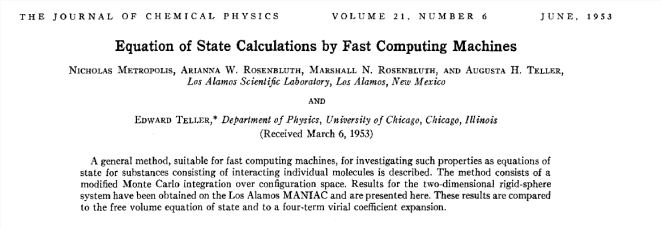
(Received March 6, 1953)

A general method, suitable for fast computing machines, for investigating such properties as equations of state for substances consisting of interacting individual molecules is described. The method consists of a modified Monte Carlo integration over configuration space. Results for the two-dimensional rigid-sphere system have been obtained on the Los Alamos MANIAC and are presented here. These results are compared to the free volume equation of state and to a four-term virial coefficient expansion.

- Use stochastic simulation to draw samples from posterior distributions.
- Avoid explicit calculation of integrals in Bayes formula.

Bayesian computation

- In the early 1990s, statisticians rediscovered work from the 1950's in physics.



THE JOURNAL OF CHEMICAL PHYSICS VOLUME 21, NUMBER 6 JUNE, 1953

Equation of State Calculations by Fast Computing Machines

NICHOLAS METROPOLIS, ARIANNA W. ROSENBLUTH, MARSHALL N. ROSENBLUTH, AND AUGUSTA H. TELLER,
Los Alamos Scientific Laboratory, Los Alamos, New Mexico

AND

EDWARD TELLER,* *Department of Physics, University of Chicago, Chicago, Illinois*

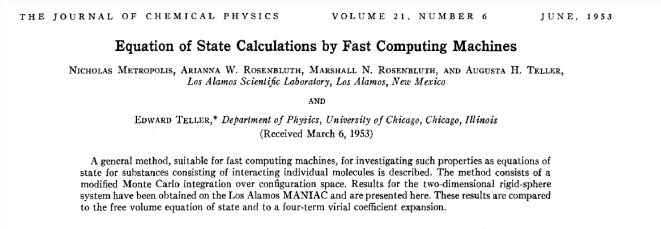
(Received March 6, 1953)

A general method, suitable for fast computing machines, for investigating such properties as equations of state for substances consisting of interacting individual molecules is described. The method consists of a modified Monte Carlo integration over configuration space. Results for the two-dimensional rigid-sphere system have been obtained on the Los Alamos MANIAC and are presented here. These results are compared to the free volume equation of state and to a four-term virial coefficient expansion.

- Use stochastic simulation to draw samples from posterior distributions.
- Avoid explicit calculation of integrals in Bayes formula.
- Instead, approximate posterior to arbitrary degree of precision by drawing large sample.

Bayesian computation

- In the early 1990s, statisticians rediscovered work from the 1950's in physics.



THE JOURNAL OF CHEMICAL PHYSICS VOLUME 21, NUMBER 6 JUNE, 1953

Equation of State Calculations by Fast Computing Machines

NICHOLAS METROPOLIS, ARIANNA W. ROSENBLUTH, MARSHALL N. ROSENBLUTH, AND AUGUSTA H. TELLER,
Los Alamos Scientific Laboratory, Los Alamos, New Mexico

AND

EDWARD TELLER,* *Department of Physics, University of Chicago, Chicago, Illinois*

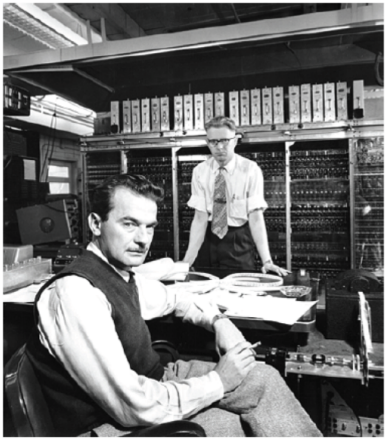
(Received March 6, 1953)

A general method, suitable for fast computing machines, for investigating such properties as equations of state for substances consisting of interacting individual molecules is described. The method consists of a modified Monte Carlo integration over configuration space. Results for the two-dimensional rigid-sphere system have been obtained on the Los Alamos MANIAC and are presented here. These results are compared to the free volume equation of state and to a four-term virial coefficient expansion.

- Use stochastic simulation to draw samples from posterior distributions.
- Avoid explicit calculation of integrals in Bayes formula.
- Instead, approximate posterior to arbitrary degree of precision by drawing large sample.
- Markov chain Monte Carlo = MCMC; boost to Bayesian statistics!

MANIAC

MANIAC:
Mathematical Analyzer, Numerical Integrator, and Computer



MANIAC:
1000 pounds
5 kilobytes of memory
70k multiplications/sec

Your laptop:
4–7 pounds
2–8 million kilobytes
Billions of multiplications/sec

Why are MCMC methods so useful?

- MCMC: stochastic algorithm to produce sequence of dependent random numbers (from Markov chain).

Why are MCMC methods so useful?

- MCMC: stochastic algorithm to produce sequence of dependent random numbers (from Markov chain).
- Converge to equilibrium (aka stationary) distribution.

Why are MCMC methods so useful?

- MCMC: stochastic algorithm to produce sequence of dependent random numbers (from Markov chain).
- Converge to equilibrium (aka stationary) distribution.
- Equilibrium distribution is the desired posterior distribution!

Why are MCMC methods so useful?

- MCMC: stochastic algorithm to produce sequence of dependent random numbers (from Markov chain).
- Converge to equilibrium (aka stationary) distribution.
- Equilibrium distribution is the desired posterior distribution!
- Several ways of constructing these chains: e.g., Metropolis-Hastings, Gibbs sampler, Metropolis-within-Gibbs.

Why are MCMC methods so useful?

- MCMC: stochastic algorithm to produce sequence of dependent random numbers (from Markov chain).
- Converge to equilibrium (aka stationary) distribution.
- Equilibrium distribution is the desired posterior distribution!
- Several ways of constructing these chains: e.g., Metropolis-Hastings, Gibbs sampler, Metropolis-within-Gibbs.
- How to implement them in practice?!

The Metropolis algorithm

- Let's go back to the deer example and survival estimation.

The Metropolis algorithm

- Let's go back to the deer example and survival estimation.
- We illustrate sampling from the posterior distribution of winter survival.

The Metropolis algorithm

- Let's go back to the deer example and survival estimation.
- We illustrate sampling from the posterior distribution of winter survival.
- We write functions in R for the likelihood, the prior and the posterior.

```r
# deer data, 19 "success" out of 57 "attempts"
survived <- 19
released <- 57

# log-likelihood function
loglikelihood <- function(x, p){
  dbinom(x = x, size = released, prob = p, log = TRUE)
}

# prior density
logprior <- function(p){
  dunif(x = p, min = 0, max = 1, log = TRUE)
}
```

```r
# posterior density function (log scale)
posterior <- function(x, p){
  loglikelihood(x, p) + logprior(p) # - log(Pr(data))
}
```

To simulate from this posterior distribution, we use the **Metropolis algorithm**:

To simulate from this posterior distribution, we use the **Metropolis algorithm**:

1. We start at any possible value of the parameter to be estimated.

To simulate from this posterior distribution, we use the **Metropolis algorithm**:

1. We start at any possible value of the parameter to be estimated.
2. To decide where to visit next, we propose to move away from the current value of the parameter. We add to this current value some random value from say a normal distribution with some variance. We call this the **candidate** location.

To simulate from this posterior distribution, we use the **Metropolis algorithm**:

1. We start at any possible value of the parameter to be estimated.
2. To decide where to visit next, we propose to move away from the current value of the parameter. We add to this current value some random value from say a normal distribution with some variance. We call this the **candidate** location.
3. We compute the ratio of the probabilities at the candidate and current locations $R = posterior(candidate)/posterior(current)$. This is where the magic of MCMC happens, in that $\Pr(data)$ (the denominator of the Bayes theorem) cancels out when we compute R.

To simulate from this posterior distribution, we use the **Metropolis algorithm**:

1. We start at any possible value of the parameter to be estimated.
2. To decide where to visit next, we propose to move away from the current value of the parameter. We add to this current value some random value from say a normal distribution with some variance. We call this the **candidate** location.
3. We compute the ratio of the probabilities at the candidate and current locations $R = posterior(candidate)/posterior(current)$. This is where the magic of MCMC happens, in that $\Pr(data)$ (the denominator of the Bayes theorem) cancels out when we compute R.
4. We spin a continuous spinner that lands anywhere from 0 to 1 — call the random spin X. If X is smaller than R, we move to the candidate location, otherwise we remain at the current location.

To simulate from this posterior distribution, we use the **Metropolis algorithm**:

1. We start at any possible value of the parameter to be estimated.
2. To decide where to visit next, we propose to move away from the current value of the parameter. We add to this current value some random value from say a normal distribution with some variance. We call this the **candidate** location.
3. We compute the ratio of the probabilities at the candidate and current locations $R = posterior(candidate)/posterior(current)$. This is where the magic of MCMC happens, in that $\Pr(data)$ (the denominator of the Bayes theorem) cancels out when we compute R.
4. We spin a continuous spinner that lands anywhere from 0 to 1 — call the random spin X. If X is smaller than R, we move to the candidate location, otherwise we remain at the current location.
5. We repeat 2-4 a number of times called **steps** (many steps).

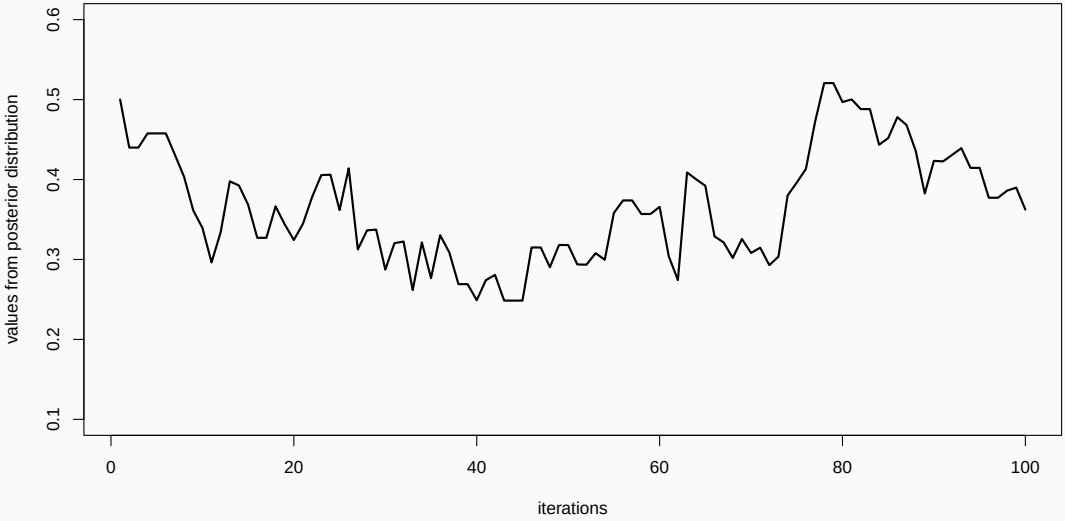
```r
# propose candidate value
move <- function(x, away = .2){
  logitx <- log(x / (1 - x))
  logit_candidate <- logitx + rnorm(1, 0, away)
  candidate <- plogis(logit_candidate)
  return(candidate)
}

# set up the scene
steps <- 100
theta.post <- rep(NA, steps)
set.seed(1234)
```

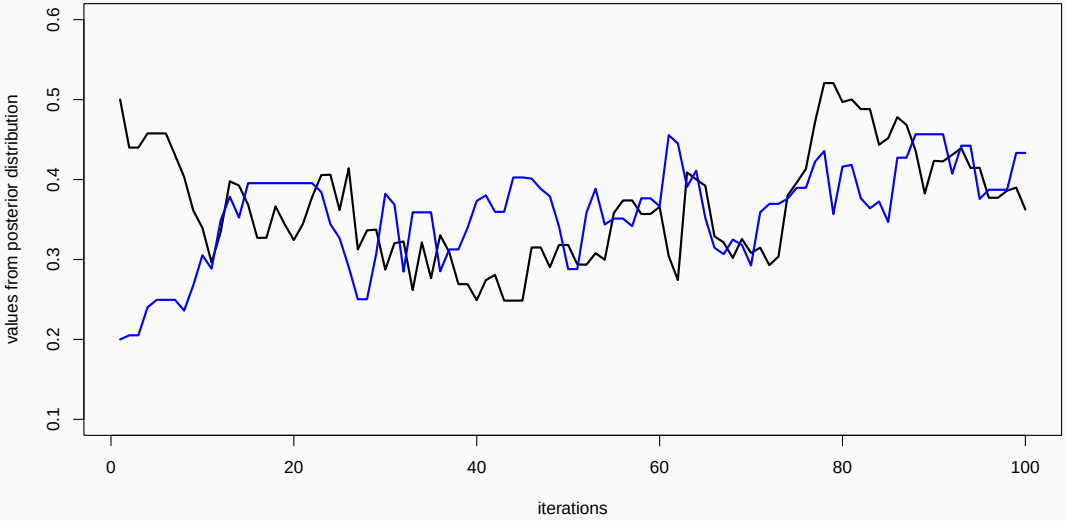
```r
# pick starting value (step 1)
inits <- 0.5
theta.post[1] <- inits
```

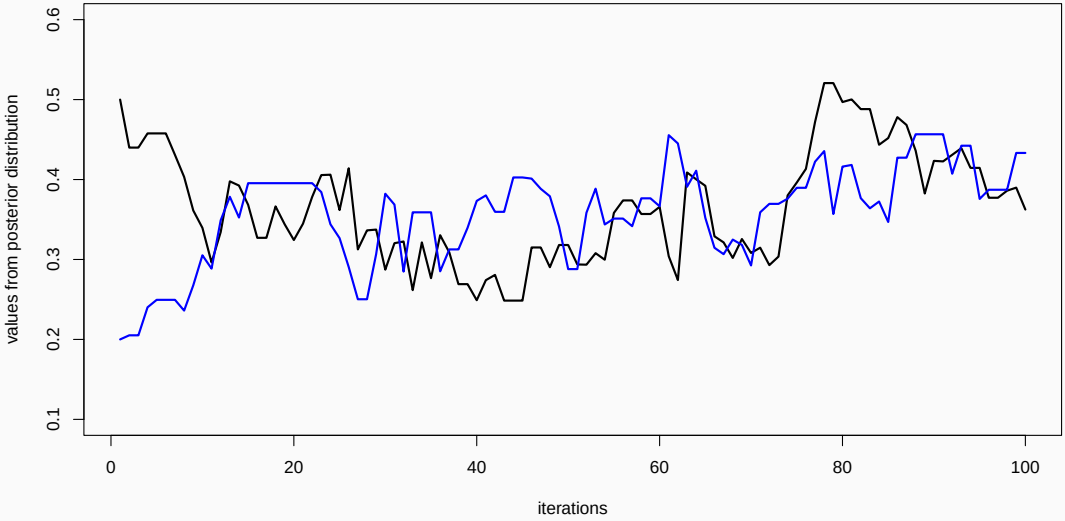
```r
for (t in 2:steps){ # repeat steps 2-4 (step 5)

  # propose candidate value for prob of success (step 2)
  theta_star <- move(theta.post[t-1])

  # calculate ratio R (step 3)
  pstar <- posterior(survived, p = theta_star)
  pprev <- posterior(survived, p = theta.post[t-1])
  logR <- pstar - pprev
  R <- exp(logR)

  # decide to accept candidate value or to keep current value (step 4)
  accept <- rbinom(1, 1, prob = min(R, 1))
  theta.post[t] <- ifelse(accept == 1, theta_star, theta.post[t-1])
}
```

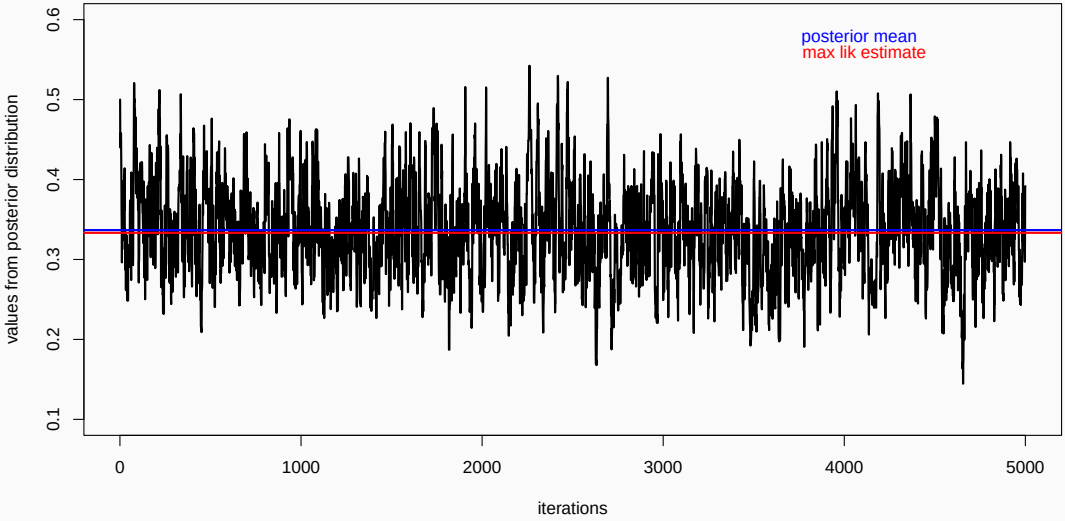
Starting at the value 0.5 and running the algorithm for 100 iterations.

```
head(theta.post)
#> [1] 0.5000000 0.4399381 0.4399381 0.4577124 0.4577124 0.4577124
tail(theta.post)
#> [1] 0.4145878 0.3772087 0.3772087 0.3860516 0.3898536 0.3624450
```


values from posterior distribution
0.1
0.2
0.3
0.4
0.5
0.6
0
20
40
60
80
100
iterations


values from posterior distribution
0.1
0.2
0.3
0.4
0.5
0.6
0
20
40
60
80
100
iterations


values from posterior distribution
0.1
0.2
0.3
0.4
0.5
0.6
0
20
40
60
80
100
iterations


posterior mean
max lik estimate
values from posterior distribution
0.1
0.2
0.3
0.4
0.5
0.6
0
1000
2000
3000
4000
5000
iterations

Animating the Metropolis algorithm - 1D example

https://gist.github.com/oliviergimenez/5ee33af9c8d947b72a39ed1764040bf3

Animating the Metropolis algorithm - 2D example

https://mbjoseph.github.io/posts/2018-12-25-animating-the-metropolis-algorithm/

The Markov-chain Monte Carlo Interactive Gallery

https://chi-feng.github.io/mcmc-demo/

Assessing convergence

Principle

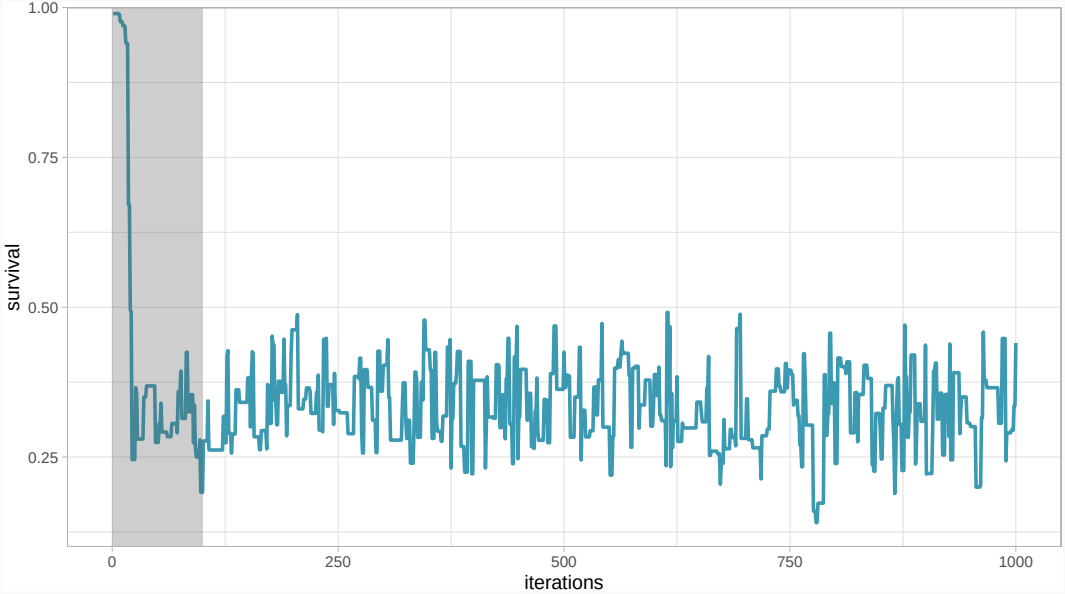
When implementing MCMC, we need to determine how long it takes for our Markov chain to converge to the target distribution, and the number of iterations we need after achieving convergence to get reasonable Monte Carlo estimates of numerical summaries (posterior means and credible intervals).

Burn-in

Burn-in

- In practice, we discard observations from the start of the Markov chain and just use observations from the chain once it has converged. The initial observations that we discard are usually referred to as the *burn-in*.
- The simplest method to determine the length of the burn-in period is to look at trace plots. Going back to our example, let's have a look to a trace plot of a chain that starts at value 0.99.

Burn-in



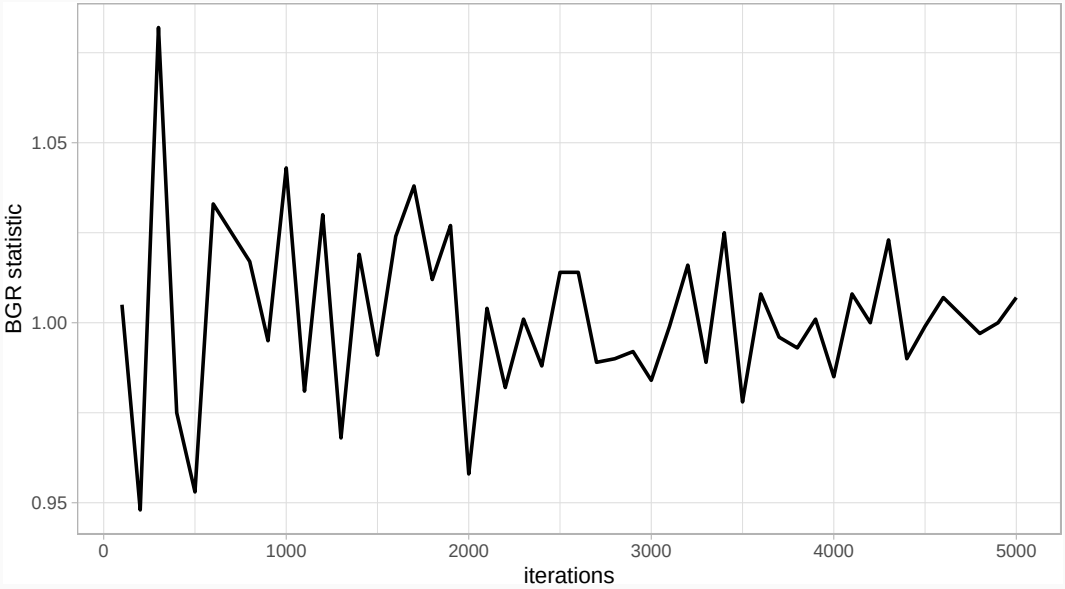
Burn-in

- Inspecting the trace plot for a single run of the Markov chain is useful.
- However, we usually run the Markov chain several times, starting from different over-dispersed points, to check that all runs achieve the same stationary distribution.
- This approach is formalised by using the Brooks-Gelman-Rubin (BGR) statistic $\hat{R}$ which measures the ratio of the total variability combining multiple chains (between-chain plus within-chain) to the within-chain variability.
- The BGR statistic asks whether there is a chain effect, and is very much alike the F test in an analysis of variance.
- Values below 1.1 indicate likely convergence.

Burn-in

Back to our example, we run two Markov chains with starting values 0.2 and 0.8 using 100 up to 5000 iterations, and calculate the BGR statistic using half the number of iterations as the length of the burn-in.

Burn-in

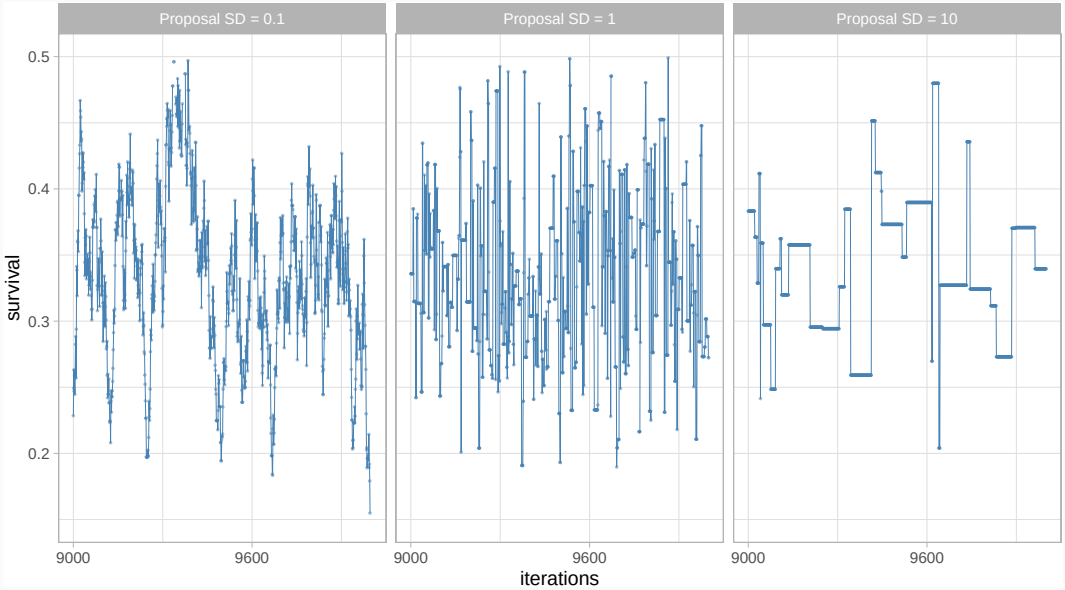

BGR statistic
1.05
1.00
0.95
0
1000
2000
3000
4000
5000
iterations

Chain length

Chain length

- How long of a chain is needed to produce reliable parameter estimates?
- To answer this question, you need to keep in mind that successive steps in a Markov chain are not independent – this is usually referred to as *autocorrelation*.
- Ideally, we would like to keep autocorrelation as low as possible.
- Here again, trace plots are useful to diagnose issues with autocorrelation.
- Let's get back to our survival example.

Chain length



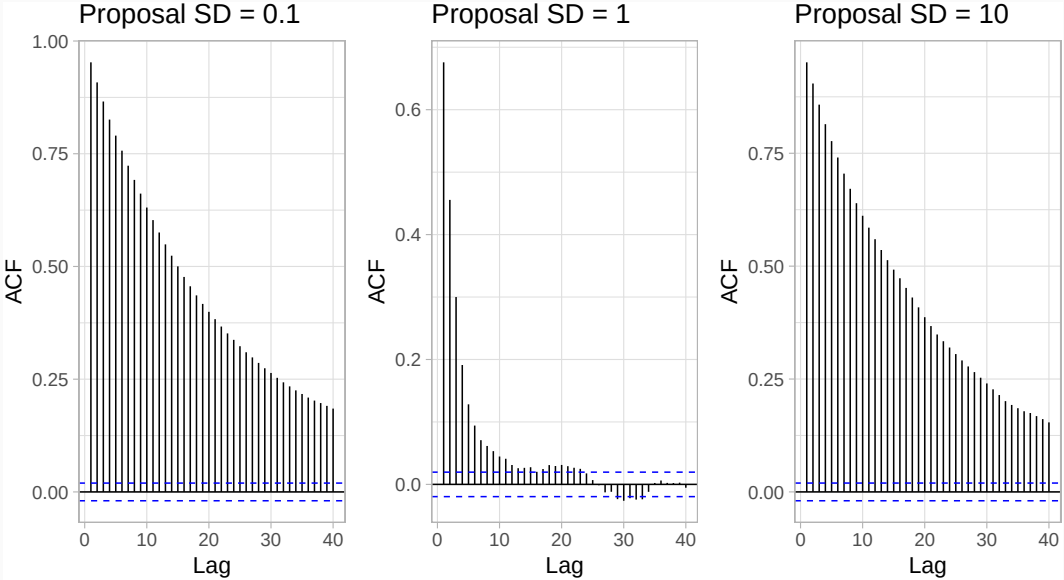
Chain length

- The movement around the parameter space is referred to as *mixing*.
- Mixing is bad when the chain makes small and big moves, and good otherwise.

Chain length

- In addition to trace plots, autocorrelation function (ACF) plots are a convenient way of displaying the strength of autocorrelation in a given sample values.
- ACF plots provide the autocorrelation between successively sampled values separated by an increasing number of iterations, or *lag*.
- We obtain the autocorrelation function plots for different values of the standard deviation of the proposal distribution with the R `forecast::ggAcf()` function.

Chain length



Chain length

- Autocorrelation is not necessarily a big issue. Strongly correlated observations just require large sample sizes and therefore longer simulations. But how many iterations exactly?
- The effective sample size (`n.eff`) measures chain length while taking into account chain autocorrelation.
- You should check the `n.eff` of every parameter of interest, and of any interesting parameter combinations.
- In general, we need n.eff ≥ 100 independent steps to get reasonable Monte Carlo estimates of model parameters.
- In the animal survival example, `n.eff` can be calculated with the R `coda::effectiveSize()` function.

Chain length

```
#> # A tibble: 3 x 2
#>   `Proposal SD` n.eff
#>           <dbl> <dbl>
#> 1           0.1   224
#> 2           1    1934
#> 3          10     230
```

What if you have issues of convergence?

What if you have issues of convergence?

- When mixing is bad and effective sample size is small, you may just need to increase burn-in and/or sample more.
- Using more informative priors might also make Markov chains converge faster by helping your MCMC sampler (e.g. the Metropolis algorithm) navigating more efficiently the parameter space.
- In the same spirit, picking better initial values for starting the chain does not harm. For doing that, a strategy consists in using estimates from a simpler model for which your MCMC chains do converge.

What if you have issues of convergence?

- If convergence issues persist, often there is a problem with your model.
- A bug in the code? A typo somewhere? A mistake in your maths?
- As often when coding is involved, the issue can be identified by removing complexities, and start with a simpler model until you find what the problem is.

What if you have issues of convergence?

- A general advice is to see your model as a data generating tool in the first place, simulate data from it using some realistic values for the parameters, and try to recover these parameter values by fitting the model to the simulated data.
- Simulating from a model will help you understanding how it works, what it does not do, and the data you need to get reasonable parameter estimates.

Summary

Summary

- With the Bayes' theorem, you update your beliefs (prior) with new data (likelihood) to get posterior beliefs (posterior): posterior $\propto$ likelihood $\times$ prior.
- The idea of Markov chain Monte Carlo (MCMC) is to simulate values from a Markov chain which has a stationary distribution equal to the posterior distribution you're after.
- In practice, you run a Markov chain multiple times starting from over-dispersed initial values.
- You discard iterations in an initial burn-in phase and achieve convergence when all chains reach the same regime.
- From there, you run the chains long enough and proceed with calculating Monte Carlo estimates of numerical summaries (e.g. posterior means and credible intervals) for parameters.