From random numbers to MCMC

Bayesian statistics 3 – Markov Chain Monte Carlo

Frédéric Barraquand (CNRS, IMB)

02/11/2021

Some things that we learnt the last time

- You can fit easily the linear model in JAGS/BUGS
- Similar results to frequentist statistics for large samples
- Yet very convenient representation of uncertainty
- JAGS uses the Gibbs sampler
- The Gibbs sampler is one instance of Markov Chain Monte Carlo (MCMC)

Today: What is Markov Chain Monte Carlo? Two aspects

- Markov Chain
- Monte Carlo

The Monte Carlo method

Using random numbers allows to

- compute complex integrals $\int_a^b h(x)dx$ where we know what $h(x)$ looks like mathematically but not its integral (you can do thus numerically too, but not when x is of dimension 10 or 100).
  $\mathbb{E}(f(X)) = f(x)p(x)dx$. According to the law of large numbers, $\mathbb{E}(f(X)) \approx \frac{1}{n} f(x_i)$ for n large.
- sample from a complex probability density function $f(x)$ which we know up to a constant.

Why do we want this in a Bayesian stats context?

- Let's say we want $\mathbb{P}(a < \theta < b|y)$ or the mean parameter value $\mathbb{E}(\theta) = \int_{\mathbb{R}} \theta p(\theta|y) d\theta$. We're fine with computing $\int_a^b p(\theta|y) d\theta$ or the expected mean for many posteriors $p(\theta|y)$ over a grid for a *single* parameter θ

Why do we want this in a Bayesian stats context?

- Let's say we want $\mathbb{P}(a < \theta < b|y)$ or the mean parameter value $\mathbb{E}(\theta) = \int_{\mathbb{R}} \theta p(\theta|y) d\theta$. We're fine with computing $\int_a^b p(\theta|y) d\theta$ or the expected mean for many posteriors $p(\theta|y)$ over a grid for a *single* parameter θ but not $\int \int \int ... \int_{[a_1,b_1]\times[a_2,b_2]\times...\times[a_d,b_d]} p(\theta_1, \theta_2, ..., \theta_d|y) d\theta_1 d\theta_2 ... d\theta_d$

(imagine the really horrible grid in d dimensions).

Why do we want this in a Bayesian stats context?

- Let's say we want $\mathbb{P}(a < \theta < b|y)$ or the mean parameter value $\mathbb{E}(\theta) = \int_{\mathbb{R}} \theta p(\theta|y) d\theta$. We're fine with computing $\int_a^b p(\theta|y) d\theta$ or the expected mean for many posteriors $p(\theta|y)$ over a grid for a *single* parameter θ but not $\int \int \int ... \int_{[a_1,b_1]\times[a_2,b_2]\times...\times[a_d,b_d]} p(\theta_1, \theta_2, ..., \theta_d|y) d\theta_1 d\theta_2 ... d\theta_d$

(imagine the really horrible grid in d dimensions).

- Any *marginal posterior distribution* will likely be like this. $p(\theta_1|y) = \int \int ... \int_{\mathbb{R}^{d-1}} p(\theta_1, \theta_2, ..., \theta_d|y) d\theta_2 ... d\theta_d$. For an ANOVA with 1 mean and 5 effects this is already a quadruple integral. Nasty.

Why do we want this in a Bayesian stats context?

- Let's say we want $\mathbb{P}(a < \theta < b|y)$ or the mean parameter value $\mathbb{E}(\theta) = \int_{\mathbb{R}} \theta p(\theta|y)d\theta$. We're fine with computing $\int_a^b p(\theta|y)d\theta$ or the expected mean for many posteriors $p(\theta|y)$ over a grid for a *single* parameter θ but not $\int \int \int ... \int_{[a_1,b_1]\times[a_2,b_2]\times...\times[a_d,b_d]} p(\theta_1, \theta_2, ..., \theta_d|y)d\theta_1 d\theta_2 ... d\theta_d$

(imagine the really horrible grid in d dimensions).

- Any *marginal posterior distribution* will likely be like this. $p(\theta_1|y) = \int \int ... \int_{\mathbb{R}^{d-1}} p(\theta_1, \theta_2, ..., \theta_d|y)d\theta_2 ... d\theta_d$. For an ANOVA with 1 mean and 5 effects this is already a quadruple integral. Nasty.

- Even with a single parameter θ, remember the annoying constant $\mathbb{P}(\text{data}) = m(y) = \int l(y|\theta)\pi(\theta)d\theta$ that pollutes the equation $p(\theta|y) = \frac{l(y|\theta)\pi(\theta)}{m(y)}$? In many cases we only know $p(\theta|y)$ up to a proportionality constant.

A very short history of Monte Carlo integration

Stanislas "Stan" Ulam and John "Johnny" von Neumann were working in Los Alamos. Stan Ulam, a Polish mathematician, came up with the idea of using random numbers to evaluate complex integrals involved in chain reactions. Nicholas Metropolis suggested the code name *Monte Carlo*, given the connection between probability theory and games of chance (and that according to Metropolis, Ulam had an uncle that went playing there).

A very short history of Monte Carlo integration

Stanislas “Stan” Ulam and John “Johnny” von Neumann were working in Los Alamos. Stan Ulam, a Polish mathematician, came up with the idea of using random numbers to evaluate complex integrals involved in chain reactions. Nicholas Metropolis suggested the code name *Monte Carlo*, given the connection between probability theory and games of chance (and that according to Metropolis, Ulam had an uncle that went playing there).

A bit more history: Ulam stays in the US with von Neumann in 1935 (at Princeton). He was 26. In 1939 he flees to the US. In 1943 he joins the Manhattan project. Gets involved in issues that requires "pure pragmatism and the necessity to get a heuristic survey of the problem by simple-minded brute force, rather than by massive numerical work".

A very short history of Monte Carlo integration

Stanislas “Stan” Ulam and John “Johnny” von Neumann were working in Los Alamos. Stan Ulam, a Polish mathematician, came up with the idea of using random numbers to evaluate complex integrals involved in chain reactions. Nicholas Metropolis suggested the code name *Monte Carlo*, given the connection between probability theory and games of chance (and that according to Metropolis, Ulam had an uncle that went playing there).

A bit more history: Ulam stays in the US with von Neumann in 1935 (at Princeton). He was 26. In 1939 he flees to the US. In 1943 he joins the Manhattan project. Gets involved in issues that requires "pure pragmatism and the necessity to get a heuristic survey of the problem by simple-minded brute force, rather than by massive numerical work".

In 1945 moves to UCLA as prof. Back to Los Alamos in 1946 to work on nuclear fusion, after some time in the hospital. In 1946-47 develops with von Neumann a statistical approach to the problem of neutron diffusion in fissionable material (Eckard 1987). Monte Carlo integration concept published in 1949 with Metropolis.

A simpler example: computing π

Here we use random points to sample an area. Same problem as computing an integral.

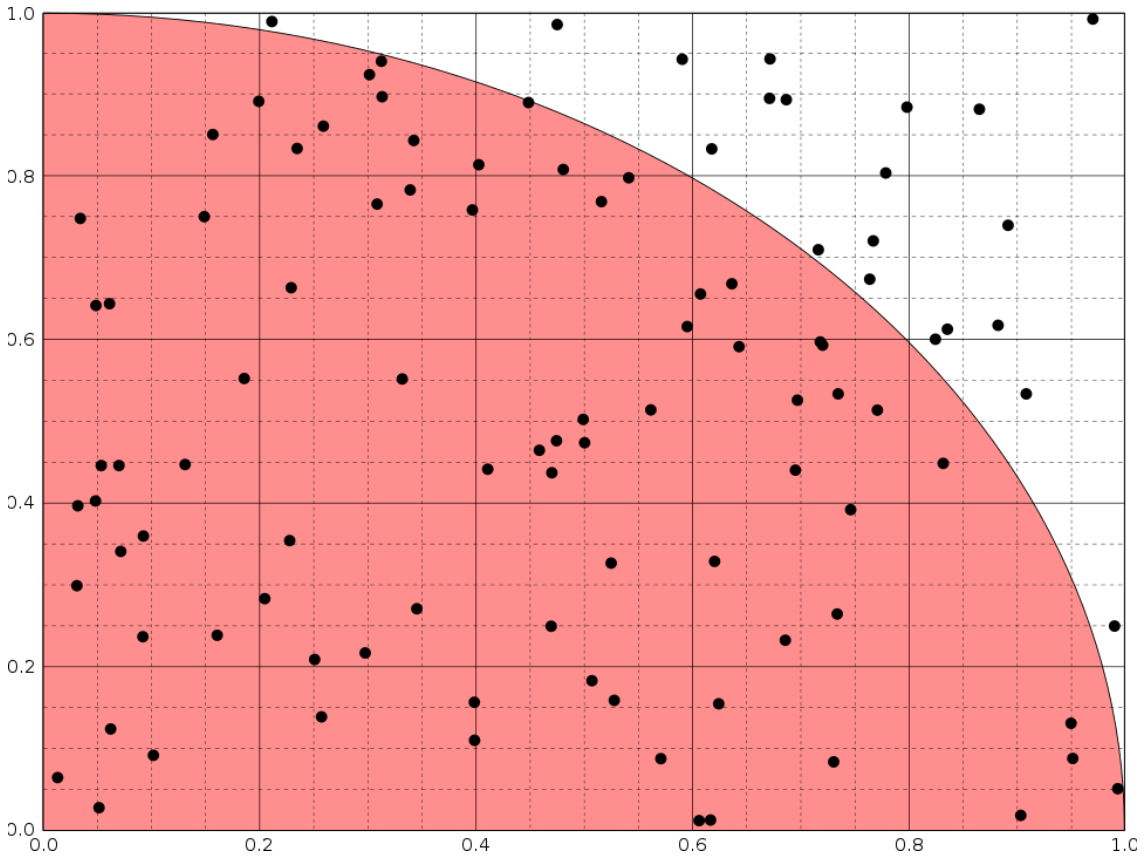


Figure 1: Autiwa - CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=6515107

Computing π: your turn

$A = \pi R^2$ for the area of the circle.

We compute numerically p the probability of landing a randomly thrown dart in the circle.

p is the ratio between the area of the circle to that of the square ($(2R)^2 = 4R^2$).

$p = \frac{\pi_{\text{est}}}{4}$. From this, find π

Computing π: solution

```
sample_p = function(n,R,L){
  x=runif(n,-L,L)
  y=runif(n,-L,L)
  d2 = x^2+y^2
  return(sum(d2<R^2)/n)
}
4*sample_p(100,1,1)
```

```
## [1] 3.08
```

```
4*sample_p(100,1,1)
```

```
## [1] 3
```

```
4*sample_p(1000,1,1)
```

```
## [1] 3.132
```

```
4*sample_p(1000,1,1)
```

```
## [1] 3.168
```

How do we get (nonuniform) random numbers?

Uniform random numbers given by God.

How do we get (nonuniform) random numbers?

Uniform random numbers given by God.

- Just joking - I just don't have the time to cover it. Very useful topic for anybody doing computational science with random numbers. Check out `set.seed()` for the random generators used in R.

How do we get (nonuniform) random numbers?

Uniform random numbers given by God.

- Just joking - I just don't have the time to cover it. Very useful topic for anybody doing computational science with random numbers. Check out `set.seed()` for the random generators used in R.
- An important technique when we know the cumulative density function $F(x) = \mathbb{P}(X < x)$ is **inverse transform sampling**

How do we get (nonuniform) random numbers?

Uniform random numbers given by God.

- Just joking - I just don't have the time to cover it. Very useful topic for anybody doing computational science with random numbers. Check out `set.seed()` for the random generators used in R.
- An important technique when we know the cumulative density function $F(x) = \mathbb{P}(X < x)$ is **inverse transform sampling**
- For all the rest, one ref

Non-uniform random variate generation by Luc Devroye

Inverse transform sampling: theory

We have random variable X with pdf $f(x)$ and cdf $\mathbb{P}(X < x) = F(x)$. Let's assume transform T is such that $T(U) = X$ where U has uniform distribution in $[0, 1]$.

Inverse transform sampling: theory

We have random variable X with pdf $f(x)$ and cdf $\mathbb{P}(X < x) = F(x)$. Let's assume transform T is such that $T(U) = X$ where U has uniform distribution in $[0, 1]$.

Then $F(x) = \mathbb{P}(T(U) < x) = \mathbb{P}(U < T^{-1}(x)) = T^{-1}(x)$.

Inverse transform sampling: theory

We have random variable X with pdf $f(x)$ and cdf $\mathbb{P}(X < x) = F(x)$. Let's assume transform T is such that $T(U) = X$ where U has uniform distribution in $[0, 1]$.

Then $F(x) = \mathbb{P}(T(U) < x) = \mathbb{P}(U < T^{-1}(x)) = T^{-1}(x)$.

The last bit *is* confusing, that's because the $\mathbb{P}(U < u) = u$ for a uniform.

Inverse transform sampling: theory

We have random variable X with pdf $f(x)$ and cdf $\mathbb{P}(X < x) = F(x)$. Let's assume transform T is such that $T(U) = X$ where U has uniform distribution in $[0, 1]$.

Then $F(x) = \mathbb{P}(T(U) < x) = \mathbb{P}(U < T^{-1}(x)) = T^{-1}(x)$.

The last bit *is* confusing, that's because the $\mathbb{P}(U < u) = u$ for a uniform.

Thus $T = F^{-1}$ and $F^{-1}(U)$ has the distribution of X.

Inverse transform sampling: theory

We have random variable X with pdf $f(x)$ and cdf $\mathbb{P}(X < x) = F(x)$. Let's assume transform T is such that $T(U) = X$ where U has uniform distribution in $[0, 1]$.

Then $F(x) = \mathbb{P}(T(U) < x) = \mathbb{P}(U < T^{-1}(x)) = T^{-1}(x)$.

The last bit *is* confusing, that's because the $\mathbb{P}(U < u) = u$ for a uniform.

Thus $T = F^{-1}$ and $F^{-1}(U)$ has the distribution of X.

There are smart variants of this even when you don't know analytically the cumulative distribution functions. See e.g. the Box-Muller algorithm for generating Gaussian random variables

Inverse transform sampling: exponential distribution

Exponential distribution with parameter λ.

- probability density function $f(x) = \lambda \exp(-\lambda x)$
- cumulative distribution function $F(x) = 1 - \exp(-\lambda x)$

$T = F^{-1}$? ...

Then simulate it in R.

Inverse transform sampling: exponential (solution)

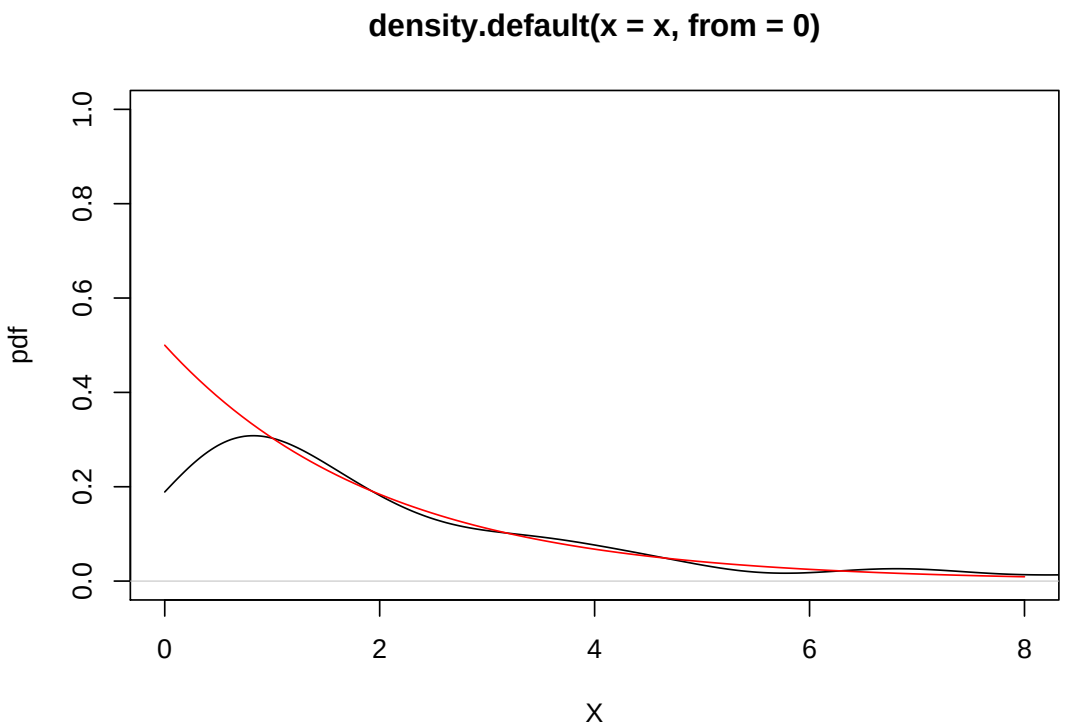
Exponential distribution with parameter λ.

- probability density function $f(x) = \lambda \exp(-\lambda x)$
- cumulative distribution function $F(x) = 1 - \exp(-\lambda x)$

$T = F^{-1}$? $T(u) = -\frac{1}{\lambda} \ln(1-u)$

Simulating the exponential distribution

```
u=runif(100,0,1)
lambda = 0.5
x = (-1/lambda)*log(1-u)
plot(density(x,from=0),xlim=c(0,8),ylim=c(0,1),xlab="X",ylab="pdf")
curve(dexp(x,rate=0.5),col="red",add=TRUE)
```



Ideas to improve this

Acceptance-rejection sampling: theory

Needs an upper bound

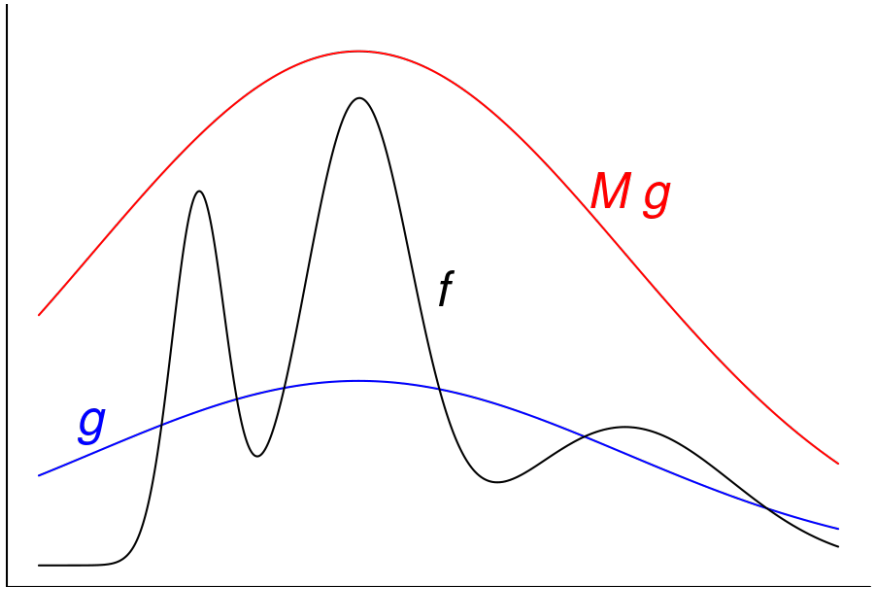


Figure 2: Rejection method, from
https://kingaa.github.io/short-course/pfilter/monteCarlo.html

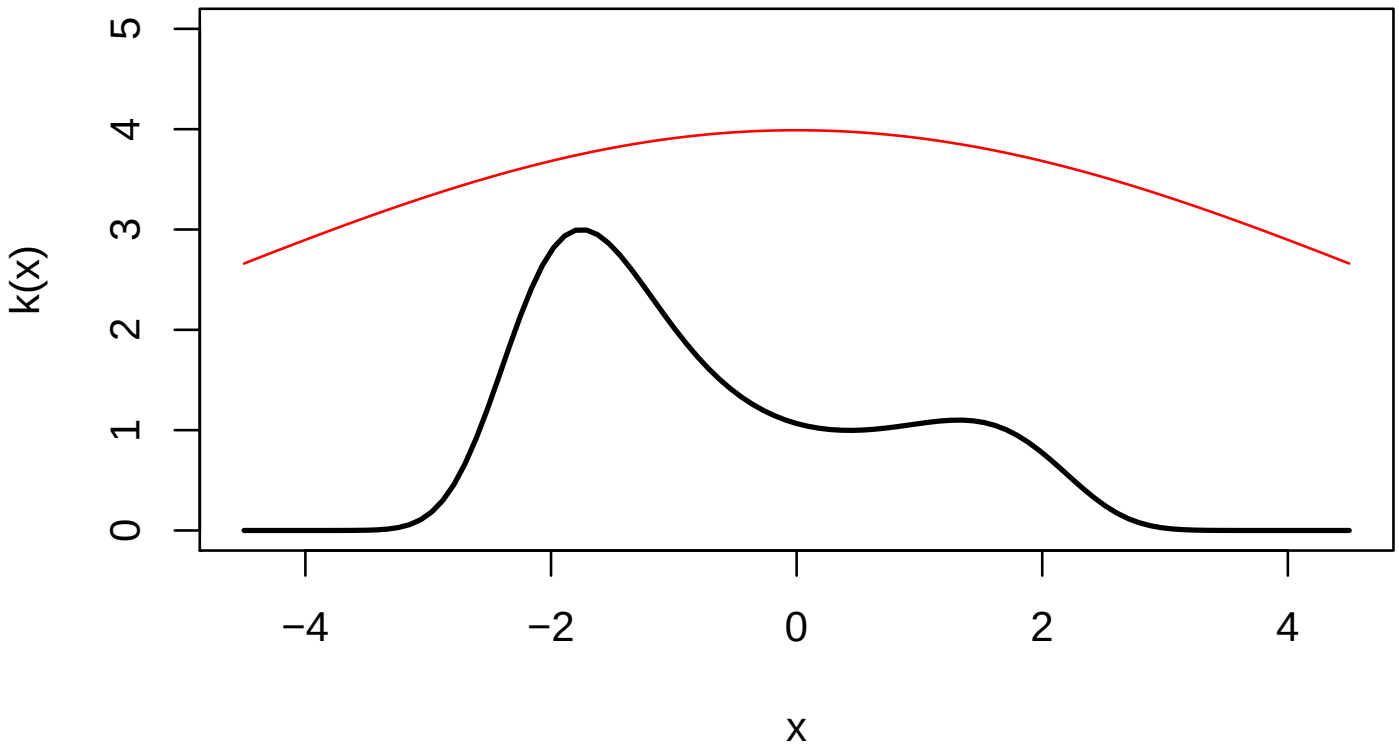
- Draw $X_i \sim g$ and $U_i \sim \mathcal{U}(0,1)$
- if $u_i < \frac{f(x_i)}{Mg(x_i)}$ accept and set $y_i = x_i$, else reject and redraw

Eventually $(y_1, ...y_n) \sim f$ iid. Kinda magic, huh?

Acceptance-rejection sampling: practice

Let's take some not so nice function and something on top

```
k <- function(x, a=.4, b=.08){exp( a * (x - a)^2 - b * x^4)}
curve(k(x), from = -4.5, to = 4.5, lwd=2, col="black",ylim=c(0,5))
curve(50*dnorm(x,0,5),from=-4.5,to=4.5,col="red",add=TRUE)
```

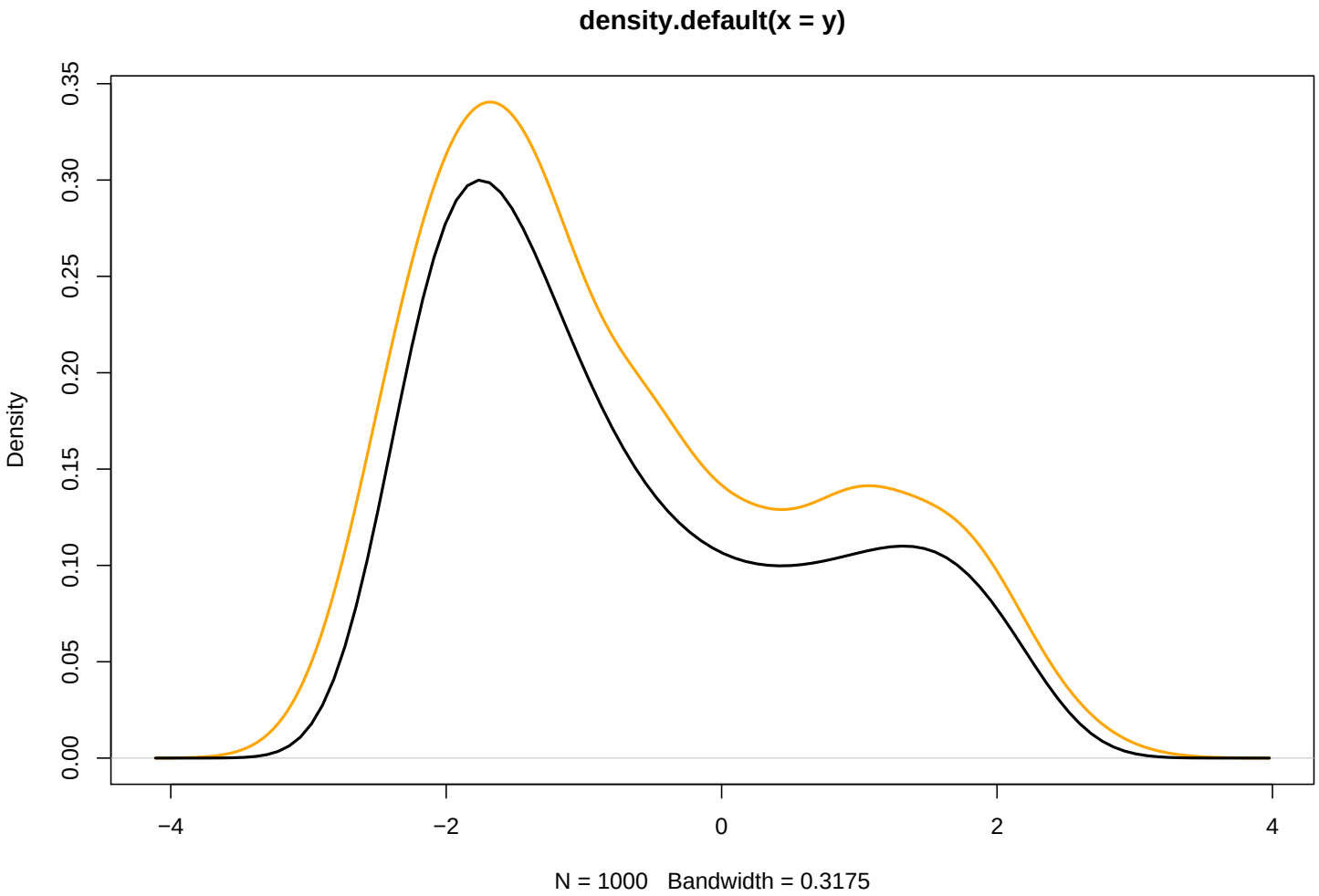


Acceptance-rejection sampling: R code

```
counter = 0
n=1000
y = rep(0,n)
while (counter<n){
    x=rnorm(1,0,5)
    u=runif(1)
    if(u<(k(x)/(50*dnorm(x,0,5)))){
      y[counter] = x
      counter=counter+1
      }
}
```

Acceptance-rejection sampling: results

```
plot(density(y),lwd=2,col="orange")
curve(k(x)/10, lwd=2, col="black",add=TRUE)
```



Now we know how to do Monte-Carlo

We want $\mathbb{E}(Y)$ where Y follows density $k(y)$? Compute $\frac{1}{n}\sum_{i=1}^{n} y_i$.

```
mean(y)
```

```
## [1] -0.6765236
```

```
sd(y)
```

```
## [1] 1.404464
```

etc.

Now to Markov Chain Monte Carlo. $\rightarrow$ Markov Chain?

Markov chain

A little bit of formalism:

- A **Markov chain** is a special kind of *stochastic process*.

Markov chain

A little bit of formalism:

- A **Markov chain** is a special kind of *stochastic process*.
- A *stochastic process* is a family of random variables X_t where t is an index that allows to model dependency between the variables (often the time)

Markov chain

A little bit of formalism:

- A **Markov chain** is a special kind of *stochastic process*.
- A *stochastic process* is a family of random variables X_t where t is an index that allows to model dependency between the variables (often the time)
- When time is discrete, we can say "chains"

Markov chain

A little bit of formalism:

- A **Markov chain** is a special kind of *stochastic process*.
- A *stochastic process* is a family of random variables X_t where t is an index that allows to model dependency between the variables (often the time)
- When time is discrete, we can say "chains"
- E.g. weather. $\mathbb{P}$(rain tomorrow) depends on whether it has rained today.

Markov chain

A little bit of formalism:

- A **Markov chain** is a special kind of *stochastic process*.
- A *stochastic process* is a family of random variables X_t where t is an index that allows to model dependency between the variables (often the time)
- When time is discrete, we can say "chains"
- E.g. weather. $\mathbb{P}$(rain tomorrow) depends on whether it has rained today.
- Remember, a *random variable* is in fact a function $X : \Omega \to \mathbb{R}$, associating one or several events ω (e.g. $\omega =$ "it rains") to numbers (e.g., $x = 1$).

Markov chain

A little bit of formalism:

- A **Markov chain** is a special kind of *stochastic process*.
- A *stochastic process* is a family of random variables X_t where t is an index that allows to model dependency between the variables (often the time)
- When time is discrete, we can say "chains"
- E.g. weather. $\mathbb{P}$(rain tomorrow) depends on whether it has rained today.
- Remember, a *random variable* is in fact a function $X : \Omega \to \mathbb{R}$, associating one or several events ω (e.g. $\omega =$ "it rains") to numbers (e.g., $x = 1$).
- Thus a stochastic process $X_t(\omega)$ or $X(t, \omega)$ returns numbers as a function of two things: an event and an index (of time or position).

A very simple Markov chain – weather prediction

Every day, either it rains or the sun shines. This is the true state $\{1, 2\}$ I'm interested in. We assume that weather obeys the **Markov property** so that $\mathbb{P}(S_t|S_{t-1}, S_{t-2}, S_{t-3}, ...) = \mathbb{P}(S_t|S_{t-1})$.

A very simple Markov chain – weather prediction

Every day, either it rains or the sun shines. This is the true state $\{1, 2\}$ I'm interested in. We assume that weather obeys the **Markov property** so that $\mathbb{P}(S_t|S_{t-1}, S_{t-2}, S_{t-3}, ...) = \mathbb{P}(S_t|S_{t-1})$.

Of course, IRL it doesn't. But let's assume there are four possible transitions for tomorrow's weather $1 \to 1$, $1 \to 2$, $2 \to 1$, $2 \to 2$ and these only depend on the weather of the day.

A very simple Markov chain – weather prediction

Every day, either it rains or the sun shines. This is the true state $\{1, 2\}$ I'm interested in. We assume that weather obeys the **Markov property** so that $\mathbb{P}(S_t|S_{t-1}, S_{t-2}, S_{t-3}, ...) = \mathbb{P}(S_t|S_{t-1})$.

Of course, IRL it doesn't. But let's assume there are four possible transitions for tomorrow's weather $1 \to 1$, $1 \to 2$, $2 \to 1$, $2 \to 2$ and these only depend on the weather of the day.

Markov himself used his eponymous chains to model transitions between letters in Pushkin's "Eugene Onegin".

What are the model parameters?

These are

$\mathbb{P}(\text{rain at } t | \text{sun at } t-1) = 1 - \mathbb{P}(\text{sun at } t | \text{sun at } t-1)$ which is formally $\mathbb{P}(S_t = 1 | S_{t-1} = 2)$ // rain $= 1$, sun $= 2$.

What are the model parameters?

These are

$\mathbb{P}(\text{rain at } t | \text{sun at } t-1) = 1 - \mathbb{P}(\text{sun at } t | \text{sun at } t-1)$ which is formally $\mathbb{P}(S_t = 1 | S_{t-1} = 2)$ // rain $= 1$, sun $= 2$.

We also consider

$\mathbb{P}(\text{sun at } t | \text{rain at } t-1) = 1 - \mathbb{P}(\text{rain at } t | \text{rain at } t-1)$ which is formally $\mathbb{P}(S_t = 2 | S_{t-1} = 1)$.

What are the model parameters?

These are

$\mathbb{P}(\text{rain at } t | \text{sun at } t-1) = 1 - \mathbb{P}(\text{sun at } t | \text{sun at } t-1)$ which is formally $\mathbb{P}(S_t = 1 | S_{t-1} = 2)$ // rain $= 1$, sun $= 2$.

We also consider

$\mathbb{P}(\text{sun at } t | \text{rain at } t-1) = 1 - \mathbb{P}(\text{rain at } t | \text{rain at } t-1)$ which is formally $\mathbb{P}(S_t = 2 | S_{t-1} = 1)$.

We assume these do not change over time, so truly there are two transition probabilities:

$\mathbb{P}(\text{rain} \to \text{sun})$ and $\mathbb{P}(\text{sun} \to \text{rain})$.

This makes a *transition matrix*

$$\Gamma = \begin{pmatrix} 0.8 & 0.2 \\ 0.3 & 0.7 \end{pmatrix}$$

A little bit of general theory

We stack all this in a matrix of **transition probabilities** $\Gamma = (\gamma_{ij})$ with $\gamma_{ij} = \mathbb{P}(S_t = j | S_{t-1} = i)$[1] so that the rows sum to 1.

[1] Many people use P but we stick with Γ to match @mcclintock2020uncovering

A little bit of general theory

We stack all this in a matrix of **transition probabilities** $\Gamma = (\gamma_{ij})$ with $\gamma_{ij} = \mathbb{P}(S_t = j | S_{t-1} = i)$[1] so that the rows sum to 1.

In general, let's say I want to compute the probability of the states at time t from those at times $t-1$. We use the Law of Total Probability to obtain

[1] Many people use P but we stick with Γ to match @mcclintock2020uncovering

A little bit of general theory

We stack all this in a matrix of **transition probabilities** $\Gamma = (\gamma_{ij})$ with $\gamma_{ij} = \mathbb{P}(S_t = j | S_{t-1} = i)$[1] so that the rows sum to 1.

In general, let's say I want to compute the probability of the states at time t from those at times $t-1$. We use the Law of Total Probability to obtain

$p_t[j] = \mathbb{P}(S_t = j) = \sum_i \mathbb{P}(S_t = j | S_{t-1} = i)\mathbb{P}(S_{t-1} = i)$. Thus, switching to matrix notation, $p_t = p_{t-1}\Gamma$.

[1] Many people use P but we stick with Γ to match @mcclintock2020uncovering

A little bit of theory – cont'd

So we have the update equation $p_t = p_{t-1}\Gamma$. It's for an **homogeneous** Markov chain as Γ does not depend of time (the laws of weather do not change over time – well, except for seasonality, climate change,...).

A little bit of theory – cont'd

So we have the update equation $p_t = p_{t-1}\Gamma$. It's for an **homogeneous** Markov chain as Γ does not depend of time (the laws of weather do not change over time – well, except for seasonality, climate change,. . .).

Stationary distribution: $\pi = \pi\Gamma$ so that $\pi(I - \Gamma) = 0$. The left eigenvectors of Γ with an eigenvalue of 1 are the stationary probabilities.

A little bit of theory – cont'd

So we have the update equation $p_t = p_{t-1}\Gamma$. It's for an **homogeneous** Markov chain as Γ does not depend of time (the laws of weather do not change over time – well, except for seasonality, climate change,. . .).

Stationary distribution: $\pi = \pi\Gamma$ so that $\pi(I - \Gamma) = 0$. The left eigenvectors of Γ with an eigenvalue of 1 are the stationary probabilities.

True only when a *single stationary distribution exists*, but for discrete-time Markov chains we know the conditions for such existence. The chain must be:

- irreducible (all states reachable from anywhere)
- aperiodic
- recurrent (all states are visited infinitely many times)

A little bit of theory – cont'd

So we have the update equation $p_t = p_{t-1}\Gamma$. It's for an **homogeneous** Markov chain as Γ does not depend of time (the laws of weather do not change over time – well, except for seasonality, climate change,. . .).

Stationary distribution: $\pi = \pi\Gamma$ so that $\pi(I - \Gamma) = 0$. The left eigenvectors of Γ with an eigenvalue of 1 are the stationary probabilities.

True only when a *single stationary distribution exists*, but for discrete-time Markov chains we know the conditions for such existence. The chain must be:

- irreducible (all states reachable from anywhere)
- aperiodic
- recurrent (all states are visited infinitely many times)

All true for many reasonable examples.

Let's do another example: Doudou the hamster

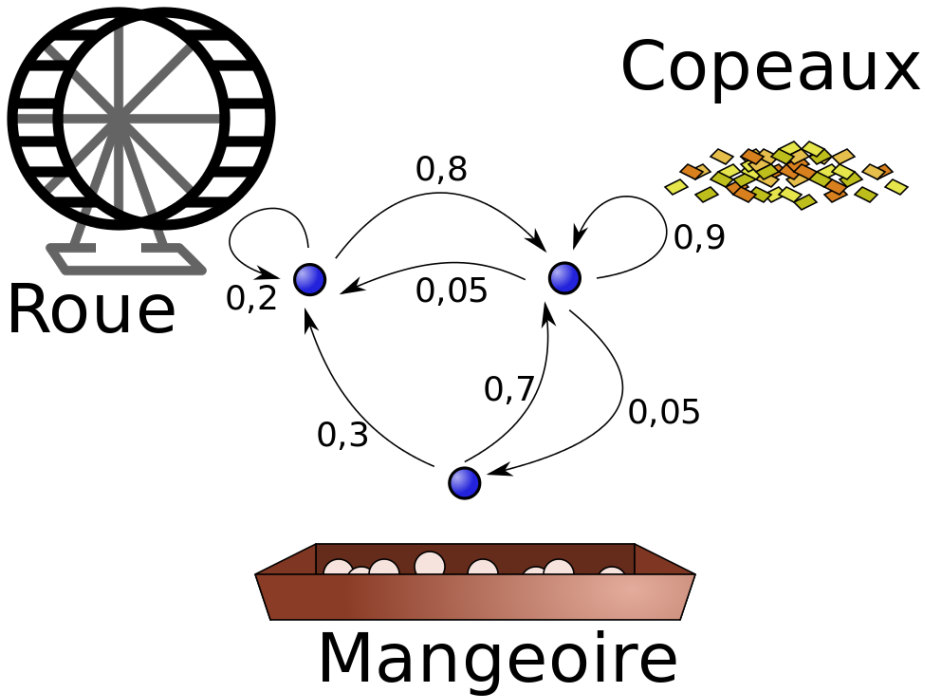


Figure 3: Each arrow is a transition probability

Three states = Sleep, Eat, Work out.

Doudou the hamster

- Is the Markov chain irreducible ? recurrent ? aperiodic ? ...
- Suppose Doudou is now asleep. What about in 2 min ? in 10 min ? ...
- Suppose now that Doudou is working out. What about in 10 min ? ...

Doudou the hamster - solutions

- Is the Markov chain irreducible ? recurrent ? aperiodic ?

Yes.

- Suppose Doudou is now asleep. What about in 2 min ? in 10 min ?

$$p_0 = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} . p_0\Gamma^2 = \begin{pmatrix} 0.885 \\ 0.045 \\ 0.070 \end{pmatrix}, p_2\Gamma^8 = \begin{pmatrix} 0.884 \\ 0.044 \\ 0.072 \end{pmatrix}$$

- Suppose now that Doudou is working out. What about in 10 min ?

$$p_0 = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} . p_0\Gamma^1 0 = \begin{pmatrix} 0.884 \\ 0.044 \\ 0.072 \end{pmatrix}$$

There is convergence!

MCMC = Markov Chain + Monte Carlo

Requires two types of convergence to evaluate say, the mean of parameter θ_1, that is, $\mathbb{E}(\theta_1) = \int \theta_1 p(\theta_1|y) d\theta_1$.

MCMC = Markov Chain + Monte Carlo

Requires two types of convergence to evaluate say, the mean of parameter θ_1, that is, $\mathbb{E}(\theta_1) = \int \theta_1 p(\theta_1|y) d\theta_1$.

We need Monte Carlo convergence, $\frac{1}{n}\sum_{i=1}^{n} \theta_1^{(i)} \longrightarrow \mathbb{E}(\theta_1)$

MCMC = Markov Chain + Monte Carlo

Requires two types of convergence to evaluate say, the mean of parameter θ_1, that is, $\mathbb{E}(\theta_1) = \int \theta_1 p(\theta_1|y) d\theta_1$.

We need Monte Carlo convergence, $\frac{1}{n}\sum_{i=1}^{n} \theta_1^{(i)} \longrightarrow \mathbb{E}(\theta_1)$

We also need Markov chain convergence, so that $p(\theta_1|y)$ is the equilibrium distribution of $\theta_1^{(i)}$.

MCMC = Markov Chain + Monte Carlo

Requires two types of convergence to evaluate say, the mean of parameter θ_1, that is, $\mathbb{E}(\theta_1) = \int \theta_1 p(\theta_1|y) d\theta_1$.

We need Monte Carlo convergence, $\frac{1}{n}\sum_{i=1}^{n} \theta_1^{(i)} \longrightarrow \mathbb{E}(\theta_1)$

We also need Markov chain convergence, so that $p(\theta_1|y)$ is the equilibrium distribution of $\theta_1^{(i)}$.

The latter is typically trickier to evaluate.

MCMC = Markov Chain + Monte Carlo

Requires two types of convergence to evaluate say, the mean of parameter θ_1, that is, $\mathbb{E}(\theta_1) = \int \theta_1 p(\theta_1|y) d\theta_1$.

We need Monte Carlo convergence, $\frac{1}{n}\sum_{i=1}^{n} \theta_1^{(i)} \longrightarrow \mathbb{E}(\theta_1)$

We also need Markov chain convergence, so that $p(\theta_1|y)$ is the equilibrium distribution of $\theta_1^{(i)}$.

The latter is typically trickier to evaluate.

What about other parameters, you would say? I gloss over those details. We'll see this later with the Gibbs sampler.

Back to Los Alamos

Metropolis (et al.) algorithm from 1953. Equation of State Calculations by Fast Computing Machines. Largely coded by Adrianna Rosenbluth and Augusta Teller, and conceptualized by Marshall Rosenbluth and Ed Teller. As the story goes, Metropolis mostly allowed to use the up-to-date computer MANIAC (Mathematical Analyzer, Numerical Integrator, and Computer).

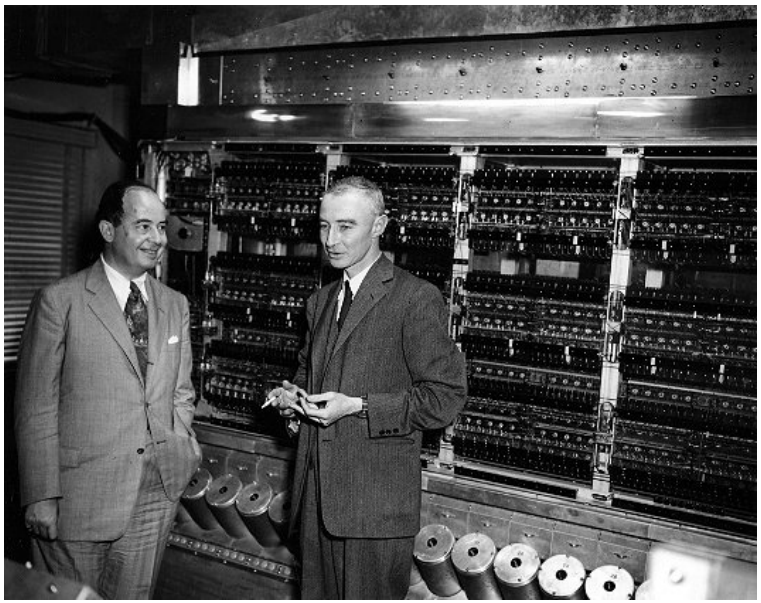


Figure 4: J von Neumann and R Oppenheimer

(incidentally, all of the authors were involved in making the H-bomb)

The Metropolis algorithm

(formulation based on Gelman et al. Bayesian Data Analysis 2020)

1. Decide on $\theta^{(0)}$ the initial value, preferably random.
2. For $k = 0, 1, 2, ...$
   a. Draw proposal parameter θ^* using proposal distribution $g(\theta^*|\theta^{(k)})$. In the Metropolis case (not Metropolis-Hastings), we require symmetry in g so that $g(\theta_a|\theta_b) = g(\theta_b|\theta_a)$.
   b. Calculate the ratio $r = \frac{p(\theta^*|y)}{p(\theta^{(k)}|y)}$
   c. Set $\theta^{(k+1)} = \theta^*$ with probability $\min(1, r)$, with $\theta^{(k+1)} = \theta^{(k)}$ if the draw fails. Bernoulli sampling with param $\min(1, r)$.

Provided that you know $p(\theta|y)$ up to a proportionality constant you can always compute r.

Metropolis on a real example

Binomial model with Beta prior (remember lecture 1, $\mathbb{P}$(white falcon)).
Now an example from my colleague Olivier Gimenez, binomial model for deer survival.

```
survived = 19
released = 47

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

Metropolis on a real example II

For Metropolis

- we assume that the proposal step for θ^* is done by any symmetric distribution
- we code the model using a function that returns a proposal given the previous value

[I'm copying Olivier's example here since it is a nice way to to implement such algorithms with modular functions.]

Now, your turn!

Metropolis on a real example III

```
# propose candidate value
move <- function(x, away = .2){
  logitx <- log(x / (1 - x))
  logit_candidate <- logitx + rnorm(1, 0, away)
  candidate <- plogis(logit_candidate) # or exp(y)/(1+exp(y))
  return(candidate)
}
# but wait is this a symmetric proposal kernel? not sure.

# propose candidate value
move <- function(x, epsilon = .05){
  v <- rnorm(1,0,epsilon)
  candidate <- x + v
  return(candidate)
}
# can go over the edges but is clearly symmetric
```

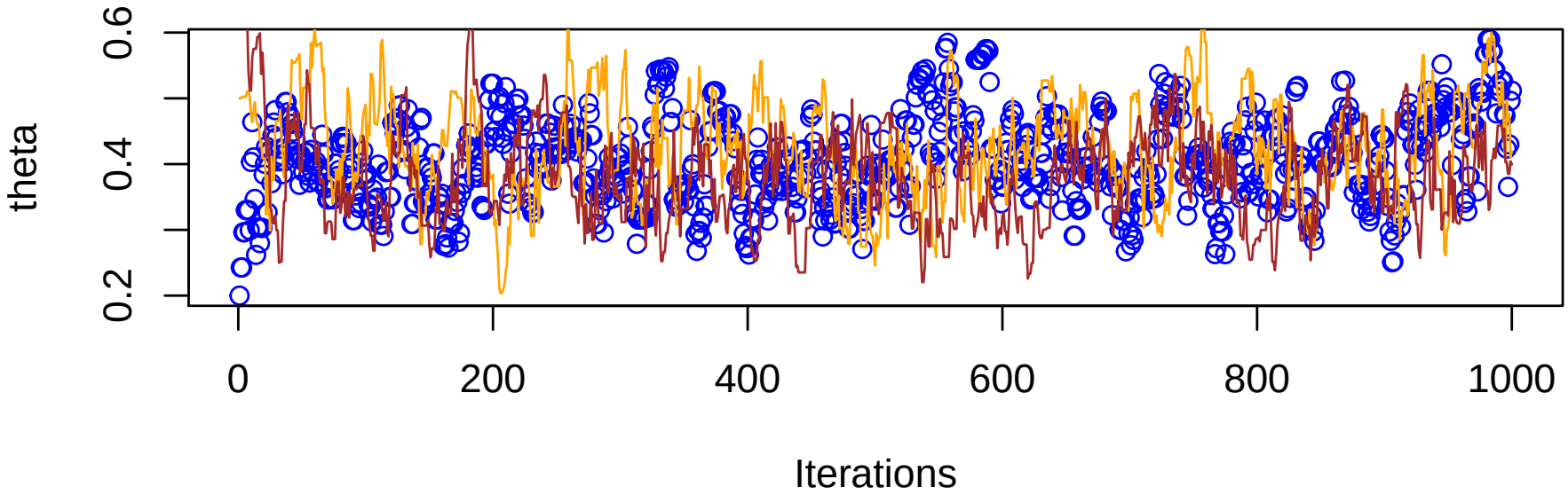
Metropolis on a real example IV

```r
metropolis <- function(steps = 100, theta0 = 0.5){

  theta <- rep(NA, steps) # pre-alloc memory
  theta[1] <- theta0   # start

  for (k in 2:steps){
    # propose candidate value for prob of success
    theta_star <- move(theta[k-1])
    # calculate ratio R
    pstar <- logposterior(survived, p = theta_star)
    pprev <- logposterior(survived, p = theta[k-1])
    logR <- pstar - pprev
    R <- exp(logR)
    # decide to accept candidate value or to keep current value
    accept <- rbinom(1, 1, prob = min(R, 1))
    theta[k] <- ifelse(accept == 1, theta_star, theta[k-1])
  }
  theta
}
```

Metropolis on a real example V

```
steps <- 1000
chain1 <- metropolis(steps = steps, theta0 = 0.2)
chain2 <- metropolis(steps = steps, theta0 = 0.5)
chain3 <- metropolis(steps = steps, theta0 = 0.7)
mean(c(chain1,chain2,chain3))
```

```
## [1] 0.4038762
```



One further exercise

Compare the theoretical Beta$(\alpha + y, \beta + n - y)$ posterior with the Metropolis estimates of the posterior distribution.

some ggplot & co. magic to make it appear even better

Further developments

Metropolis-Hasting (1970). Acceptance-rejection algorithm with proposal transition probabilities not necessarily symmetric.

Gibbs sampler, Geman brothers 1984. Popularized by Gelfand and Smith (1990). See Gelman's book for more historical details. Useful for high-dimensional θ.

Further developments

Metropolis-Hasting (1970). Acceptance-rejection algorithm with proposal transition probabilities not necessarily symmetric.

Gibbs sampler, Geman brothers 1984. Popularized by Gelfand and Smith (1990). See Gelman's book for more historical details. Useful for high-dimensional θ.

JAGS/BUGS is based on the Gibbs sampler, which is a particular multicomponent version of the Metropolis-Hastings algorithm, where *the proposal distributions are based on conditional distributions of the model* (as opposed to some random proposal).

Further developments

Metropolis-Hasting (1970). Acceptance-rejection algorithm with proposal transition probabilities not necessarily symmetric.

Gibbs sampler, Geman brothers 1984. Popularized by Gelfand and Smith (1990). See Gelman's book for more historical details. Useful for high-dimensional θ.

JAGS/BUGS is based on the Gibbs sampler, which is a particular multicomponent version of the Metropolis-Hastings algorithm, where *the proposal distributions are based on conditional distributions of the model* (as opposed to some random proposal).

A mathematical consequence of this is that the Metropolis ratio r is 1, and one always accept each move.

Gibbs sampling algorithm

$$
\begin{aligned}
\theta_1^{(k+1)} &\sim p(\theta_1^{(k)}|\theta_2^{(k)}, \theta_3^{(k)}, ..., \theta_d^{(k)}, y) \\
\theta_2^{(k+1)} &\sim p(\theta_2^{(k)}|\theta_1^{(k+1)}, \theta_3^{(k)}, ..., \theta_d^{(k)}, y) \\
\theta_3^{(k+1)} &\sim p(\theta_3^{(k)}|\theta_1^{(k+1)}, \theta_2^{(k+1)}, \theta_4^{(k)} ..., \theta_d^{(k)}, y) \\
&\dots \\
\theta_d^{(k+1)} &\sim p(\theta_d^{(k)}|\theta_1^{(k+1)}, \theta_2^{(k+1)}, ..., \theta_{d-1}^{(k+1)}, y)
\end{aligned}
$$

Based on the sampling of the so-called *full conditionals* or *conditional posteriors*

- the proposal distribution $p(\theta_i|\theta_{-i}, y)$ is **directly** taken from the model
- this works well when we have a way to specify analytically the posteriors up to a constant
- reduces a d-dimensional sampling problem to d 1-dimensional problems
- if each step cannot be sampled directly, we can use *Metropolis-within-Gibbs* or in the case of JAGS, the slice sampler.

Examples of MCMC sampling by the various algorithms

We will use https://chi-feng.github.io/mcmc-demo/

Alternatives to JAGS

- Stan, based on Hamiltonian Monte-Carlo and specifically NUTS. Large and dynamic community of developers. Language closer to C than R, although can be called easily from R.
- Nimble, which implements a variety of samplers. Close to BUGS and JAGS.
- BayesianTools, a R package calling various samplers
- INLA for Latent Gaussian Models
- Greta, which uses Google TensorFlow and can be used on massive datasets.

And these are just the ones I know about...

Why does the Metropolis algorithm work??

Let's start from the end. Assume we have a Markov chain with stationary probability density $p(\theta)$ and transition densities $p(\theta_b|\theta_a)$. We have the detailed balance condition $p(\theta_b|\theta_a)p(\theta_a) = p(\theta_a|\theta_b)p(\theta_b)$. This means the chain is *reversible*; you get the same joint probability if time flows forward or backward.

Why does the Metropolis algorithm work??

Let's start from the end. Assume we have a Markov chain with stationary probability density $p(\theta)$ and transition densities $p(\theta_b|\theta_a)$. We have the detailed balance condition $p(\theta_b|\theta_a)p(\theta_a) = p(\theta_a|\theta_b)p(\theta_b)$. This means the chain is *reversible*; you get the same joint probability if time flows forward or backward.

This can be rewritten $\frac{p(\theta_b|\theta_a)}{p(\theta_a|\theta_b)} = \frac{p(\theta_b)}{p(\theta_a)}$

Why does the Metropolis algorithm work??

Let's start from the end. Assume we have a Markov chain with stationary probability density $p(\theta)$ and transition densities $p(\theta_b|\theta_a)$. We have the detailed balance condition $p(\theta_b|\theta_a)p(\theta_a) = p(\theta_a|\theta_b)p(\theta_b)$. This means the chain is *reversible*; you get the same joint probability if time flows forward or backward.

This can be rewritten $\frac{p(\theta_b|\theta_a)}{p(\theta_a|\theta_b)} = \frac{p(\theta_b)}{p(\theta_a)}$

Now we separate the transition in two steps, proposal and acceptance

$$p(\theta_b|\theta_a) = g(\theta_b|\theta_a)a(\theta_b, \theta_a)$$

and conversely we have

$$p(\theta_a|\theta_b) = g(\theta_a|\theta_b)a(\theta_a, \theta_b)$$

Why does the Metropolis algorithm work??

Let's start from the end. Assume we have a Markov chain with stationary probability density $p(\theta)$ and transition densities $p(\theta_b|\theta_a)$. We have the detailed balance condition $p(\theta_b|\theta_a)p(\theta_a) = p(\theta_a|\theta_b)p(\theta_b)$. This means the chain is *reversible*; you get the same joint probability if time flows forward or backward.

This can be rewritten $\frac{p(\theta_b|\theta_a)}{p(\theta_a|\theta_b)} = \frac{p(\theta_b)}{p(\theta_a)}$

Now we separate the transition in two steps, proposal and acceptance

$$p(\theta_b|\theta_a) = g(\theta_b|\theta_a)a(\theta_b, \theta_a)$$

and conversely we have

$$p(\theta_a|\theta_b) = g(\theta_a|\theta_b)a(\theta_a, \theta_b)$$

Thus $\frac{p(\theta_b|\theta_a)}{p(\theta_a|\theta_b)} = \frac{a(\theta_b,\theta_a)}{a(\theta_a,\theta_b)} = \frac{p(\theta_b)}{p(\theta_a)}$. We need to find an acceptance probability $a(.,.)$ that verifies this. $a(\theta_b, \theta_a) = \frac{p(\theta_b)}{p(\theta_a)}$ and 1 in the other direction. See Gelman et al.'s book for another take at this, p. 279 section 11.2 (free pdf).

For the more math-savvy: measure-theoretic derivation for the MH algo by Gundersen