Bayesian Statistics: Laboratory 2

Vincenzo Gioia

DEAMS

University of Trieste

vincenzo.gioia@units.it

Building D, room 2.13

Office hour: Friday, 15 - 17

18/04/2024

Outline

Section 1

Accept-Reject algorithm

Accept-Reject algorithm

Accept-Reject (A-R) algorithm:

- Moving beyond the direct method (*inverse transform method*, see e.g. Section 2.1.2 in Robert and Casella, 2010) for generating values from a distribution f
- Helpful for Monte-Carlo integration

$$E_f[h(\theta)] = \int_\Theta h(\theta) f(\theta) d\theta$$

where $h(\cdot)$ is a parameter function (in Bayesian inference $f(\theta)$ translates into the posterior distribution $\pi(\theta|y)$)

Accept-Reject algorithm

A-R algorithm:

- target distribution $f(\theta)$
- candidate/instrumental density $g(\theta)$:
  - $f(\theta)$ and $g(\theta)$ have compatible supports
  - There exists a constant c: $f(\theta)/g(\theta) \leq c$, $\forall\theta$

A-R algorithm

1. Generate $\theta^* \sim g$ and, independently, $U \sim \mathcal{U}_{[0,1]}$;
2. Accept $\theta = \theta^*$ if $U \leq \frac{f(\theta^*)}{cg(\theta^*)}$, otherwise reject θ^* (and U) and go back to step 1

A-R algorithm

- The probability of acceptance (efficiency of the algorithm) is equal to $1/c$ (higher this probability, the fewer wasted simulations from g)
- Generating from $U \sim \mathcal{U}_{[0,1]}$ and multiplying for c is equivalent to generate from $U \sim \mathcal{U}_{[0,c]}$
- The *A-R* method is not an MCMC method (simulation of independent draws)
- For implementation purposes: Fix the number of attempts (we consider such a strategy) or the number of accepted values

A-R algorithm

- The algorithm works also if you know $f(\theta)$ and $g(\theta)$ up to multiplicative constants, that is consider $f(\theta) = K_f \tilde{f}(\theta)$ and $g(\theta) = K_g \tilde{g}(\theta)$ and you know only $\tilde{f}(\theta)$ and $\tilde{g}(\theta)$. In such a case the requirement is $\tilde{f}(\theta)/\tilde{g}(\theta) \leq \tilde{c} = cK_g/K_f$
- However, $1/\tilde{c}$ is not a probability of acceptance because missing constants are absorbed into $\tilde{c}$

A-R algorithm

1. Generate $\theta^* \sim g$ and, independently, $U \sim \mathcal{U}_{[0,1]}$;
2. Accept $\theta = \theta^*$ if $U \leq \frac{\tilde{f}(\theta^*)}{\tilde{c}\tilde{g}(\theta^*)}$, otherwise reject θ^* (and U) and go back to step 1

A-R algorithm in practice

Write down *Accept–Reject* algorithm, consisting in

- Generate a sample from the target f using g as candidate
- Draw the density function on top of the histogram
- Compute the acceptance probability

by considering

- Target: (standard) Gaussian

$$f(\theta) = \exp(-\theta^2/2)/\sqrt{2\pi} \qquad \theta \in \mathbb{R}$$

- Candidate: (standard) Cauchy

$$g(\theta) = [\pi(1+\theta^2)]^{-1} \qquad \theta \in \mathbb{R}$$

A-R: Gaussian - Cauchy

Find the optimal c, that is $\sup_\theta f(\theta)/g(\theta) = c$

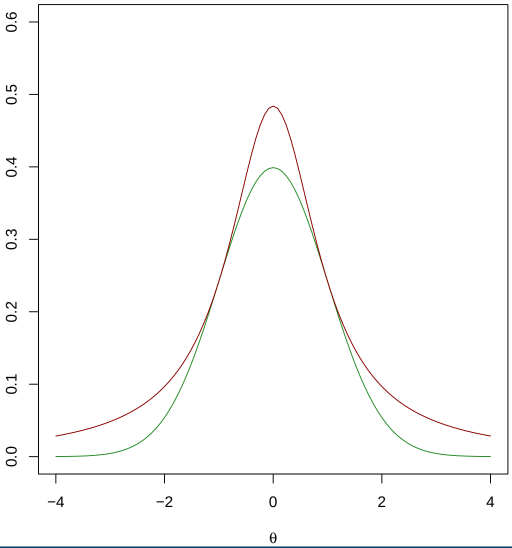
```r
c <- optimize(f = function(x){
  dnorm(x) /dcauchy(x)},
              maximum = T, interval = c(-6, 6))$objective
c
```

```
## [1] 1.520347
```

Plot $f(\theta)$ and $cg(\theta)$

```r
plot(NULL, NULL, xlim = c(-4, 4), ylim = c(0, 0.5),
     xlab = expression(theta), ylab = "")
curve(dnorm(x), col = "forestgreen", add = TRUE)
curve(c * dcauchy(x), col = "red4", add = TRUE)
```

A-R: Gaussian - Cauchy



A-R: Gaussian - Cauchy

Simulating draws from the target distribution via A-R algorithms.

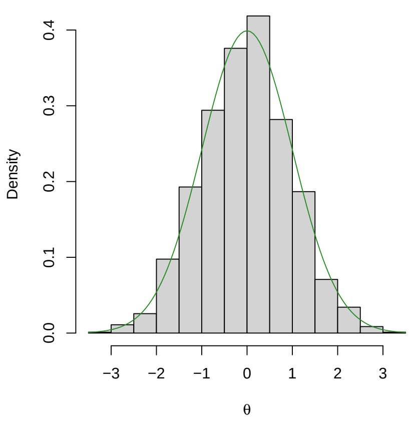
```r
# Fix the number of simulations
Nsim <- 2500
## Set a seed
set.seed(123)
## Generate Nsim values from a U(0,c)
u <- runif(Nsim, max = c)
## Generate Nsim values from a standard Cauchy
th_star <- rcauchy(Nsim)
## Condition
cond <- u <= dnorm(th_star) / dcauchy(th_star)
## Then accept the values satisfying the condition
th <- th_star[cond]
```

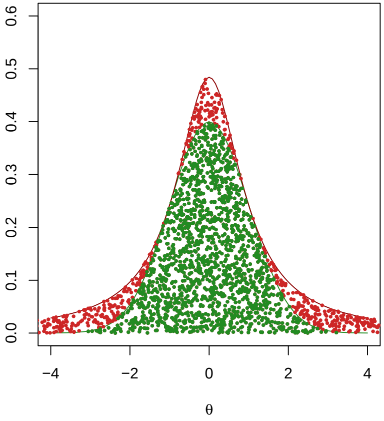
A-R: Gaussian - Cauchy

Histogram of θ (with overlapping density of $\mathcal{N}(0,1)$) and plot of the pairs $(\theta^*,\ Ug(\theta^*))$ with accepted in green and rejected in red

```r
hist(th, prob = TRUE, main = "", xlab = expression(theta))
curve(dnorm(x), col = "red", add = TRUE)

plot(NULL, NULL, xlim = c(-4, 4), ylim = c(0, 0.6),
     xlab = expression(theta), ylab = "")
curve(dnorm(x), col = "forestgreen", add = TRUE)
curve(c * dcauchy(x), col = "red4", add = TRUE)
points(th_star, u * dcauchy(th_star),
       pch = 19, cex = 0.4, col = "firebrick3")
points(th, u[cond] * dcauchy(th),
       pch = 19, cex = 0.4, col = "forestgreen")
```

A-R: Gaussian-Cauchy





A-R: Gaussian-Cauchy

Acceptance probability and comparison with the empirical counterpart

```
##acceptance probability
1/c
```

```
## [1] 0.6577446
```

```
## proportion of accepted draws
sum(cond)/Nsim
```

```
## [1] 0.6556
```

A-R: Gamma - Gamma

Another toy example of the A-R algorithm by considering as target density: $\mathrm{Gamma}(\alpha_1 = 4.3, \beta_1 = 6.2)$

$$f(\theta) = \frac{\beta_1^{\alpha_1}}{\Gamma(\alpha_1)} \theta^{\alpha_1 - 1} e^{-\beta_1 \theta} \qquad \theta \in \mathbb{R}^+$$

Select an appropriate candidate density between

- $\mathrm{Gamma}(\alpha_2 = 4, \beta_2 = 7)$
- $\mathrm{Gamma}(\alpha_2 = 4, \beta_2 = 6)$

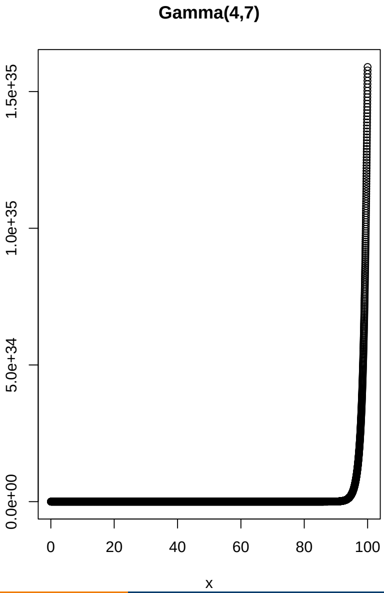
$$g(\theta) = \frac{\beta_2^{\alpha_2}}{\Gamma(\alpha_2)} \theta^{\alpha_2 - 1} e^{-\beta_2 \theta} \qquad \theta \in \mathbb{R}^+$$

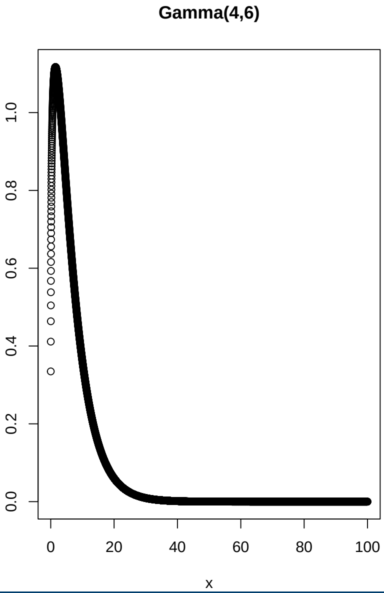
A-R: Gamma - Gamma

A sensible choice is the candidate $\mathrm{Gamma}(\alpha_2 = 4, \beta_2 = 6)$. Indeed, take a look to the following plots: clearly there in not a value of c for bounding the ratio f/g in $\mathrm{Gamma}(\alpha_2 = 4, \beta_2 = 7)$

```
x <- seq(0.01, 100, by =  0.01)
plot(x, dgamma(x, 4.3, 6.2) / dgamma(x,4,7),
     ylab = "", main = "Gamma(4,7)")
plot(x, dgamma(x, 4.3, 6.2) / dgamma(x,4,6),
     ylab = "", main = "Gamma(4,6)")
```

A-R: Gamma - Gamma





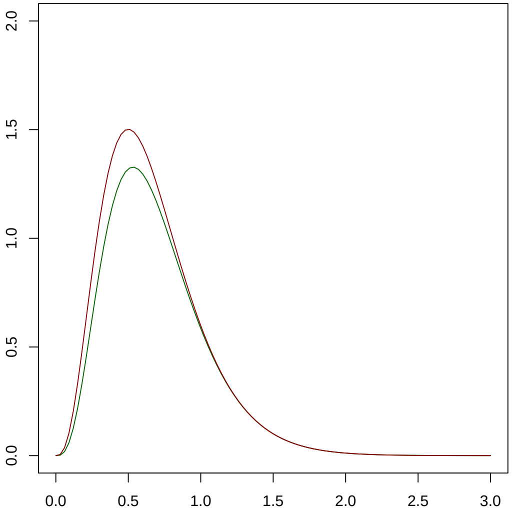
A-R: Gamma - Gamma

```r
# Parameters of the target density
a1 <- 4.3; b1 <- 6.2
# Parameters of the candidate density
a2 <- 4; b2 <- 6

c <- optimize(f = function(x){
  dgamma(x, a1, b1) / dgamma(x, a2, b2) },
              maximum = T, interval = c(0, 10))$objective
c
```

```
## [1] 1.117285
```

```r
plot(NULL, NULL, xlim = c(0,3), ylim = c(0,2),
     xlab = "", ylab = "")
curve(dgamma(x, a1, b1), col = "darkgreen", add = TRUE)
curve(c * dgamma(x, a2, b2), col = "red4", add = TRUE)
```

A-R: Gamma - Gamma



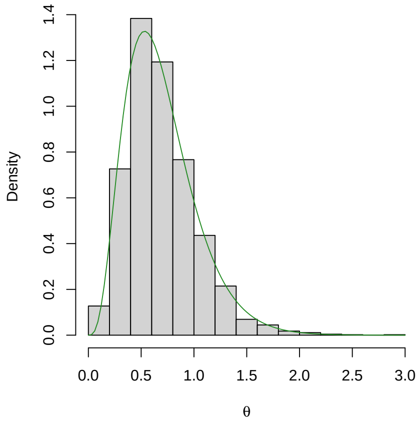
A-R: Gamma - Gamma

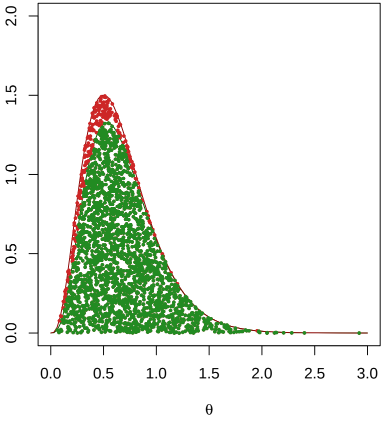
```
# Fix the number of simulations
Nsim <- 2500
## Set a seed
set.seed(123)
## Generate Nsim values from a U(0,c)
u <- runif(Nsim, max = c)
## Generate Nsim values from a standard Cauchy
th_star <- rgamma(Nsim, a2 ,b2)
## Condition
cond <- u <= dgamma(th_star, a1, b1) / dgamma(th_star, a2, b2
## Then accept the values satisfying the condition
th <- th_star[cond]
```

A-R: Gamma - Gamma

```
hist(th, prob=TRUE, main = "", xlab=expression(theta))
curve(dgamma(x, a1, b1), col = "forestgreen", add=TRUE)

plot(NULL, NULL,
     xlim=c(0,3), ylim = c(0,2),
     xlab=expression(theta), ylab="")
curve(dgamma(x, a1, b1), col = "darkgreen", add = TRUE)
curve(c * dgamma(x, a2, b2), col = "red4", add = TRUE)
points(th_star,u*dgamma(th_star,a2,b2),pch=19,cex=.4, col="fi
points(th, u[cond] * dgamma(th, a2, b2),
       pch = 19, cex = 0.4, col = "forestgreen")
```

A-R: Gamma - Gamma





A-R: Gamma - Gamma

Acceptance probability and comparison with the empirical counterpart

```
##acceptance probability
1/c
```

```
## [1] 0.8950264
```

```
## proportion of accepted draws
sum(cond)/Nsim
```

```
## [1] 0.8948
```

A-R: Gamma - Gamma

Exercise: What if we consider as candidate density

- $\mathcal{E}(1)$ (Exponential with rate equal to 1)
- $\mathrm{Gamma}(4, \tilde{\beta})$ where $\tilde{\beta} = 4\beta_1/4.3$

Compare the results with the previous ones?

A-R algorithm

- Drawback: generates "useless" simulations from the proposal g when rejecting, even those necessary to validate the output as being generated from the target f
- The method of importance sampling can be used to bypass this problem. Indeed the sampling importance resampling provides an alternative way to simulate from complex distributions (see Section 3.3.2 in Robert and Casella, 2010)

Section 2

Markov chain

Markov chain

A Markov chain $\{X^{(t)}\}$ is a sequence of dependent random variables

$$X^{(0)}, X^{(1)}, \dots, X^{(t)}, \dots$$

such that the probability distribution of $X^{(t)}$, given the past variables, depends only on $X^{(t-1)}$.

This means that the conditional probability distribution of $X^{(t)}$ given all the previous values is given by

$$p(X^{(t)}|X^{(0)}, X^{(1)}, \dots, X^{(t-1)}) \equiv p(X^{(t)}|X^{(t-1)})$$

or $X^{(t)}|X^{(0)}, X^{(1)}, \dots, X^{(t-1)} \sim K(X^{(t-1)}, X^{(t)})$

where $K(\cdot)$ is the transition kernel.

Markov chain: A discrete state and time example

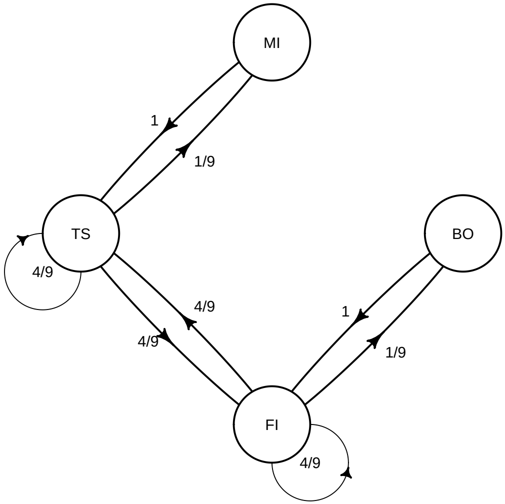
Consider a discrete state space with 4 possible states (cities), the transition kernel is a 4×4 matrix of transition probabilities P, where in the j-th row, $j = 1, \ldots, 4$ we can read the probabilities to reach any another state (k = 1, . . . , 4) starting from a state (j) in one step (e.g. $P_{21} = P(X^{(t+1)} = 1 | X^{(t)} = 2)$).

Imagine that the following transition probability describes the probabilities to move between 4 cities (MI, TS, FI, BO)

$$P = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1/9 & 4/9 & 4/9 & 0 \\ 0 & 4/9 & 4/9 & 1/9 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

For instance the probability to move from Trieste to Milan is 0.11 and the probability to move from Milan to Trieste is 1; etc.

Markov chain: A discrete state and time example



Markov chain: A discrete state and time example

```r
Tmat <- matrix(c(0, 1, 0, 0,
                 1/9, 4/9, 4/9, 0,
                 0, 4/9, 4/9, 1/9,
                 0, 0, 1, 0),
               nrow = 4, ncol = 4, byrow = TRUE)
colnames(Tmat) <- rownames(Tmat) <- c('MI', 'TS', 'FI', 'BO')
Tmat
```

```
##           MI        TS        FI        BO
## MI 0.0000000 1.0000000 0.0000000 0.0000000
## TS 0.1111111 0.4444444 0.4444444 0.0000000
## FI 0.0000000 0.4444444 0.4444444 0.1111111
## BO 0.0000000 0.0000000 1.0000000 0.0000000
```

Markov chain: A discrete state and time example

Now we define an initial (row) vector of probabilities $\pi^{(0)}$. Consider $\pi^{(0)} = (0, 1, 0, 0)$, that means to fix the initial position in Trieste

Then compute the probabilities to move to the 4 cities at time 1, that is

$$\pi^{(1)} = \pi^{(0)} P$$

```
pi0 <- c(0,1,0,0)
pi1 <- pi0 %*% Tmat; pi1
```

```
##             MI        TS        FI BO
## [1,] 0.1111111 0.4444444 0.4444444  0
```

and at time 2:

```
pi2 <- pi1 %*% Tmat; pi2
```

```
##              MI        TS        FI         BO
## [1,] 0.04938272 0.5061728 0.3950617 0.04938272
```

Markov chain: A discrete state and time example

Extend the process to compute the vector of probabilities $\pi^{(t)}$.

```
pt <- function(t, pi0, Tmat) {
  pit <- pi0
  for (i in 1 : t){
    pit <- pit %*% Tmat
    }
  return(pit)
  }
```

Markov chain: A discrete state and time example

Then, we can see the vector probabilities until time 10 (in a matrix form)

```
pi.1_10 <-sapply(1 : 10, function(.x) pt(.x, pi0, Tmat))
t(pi.1_10)
```

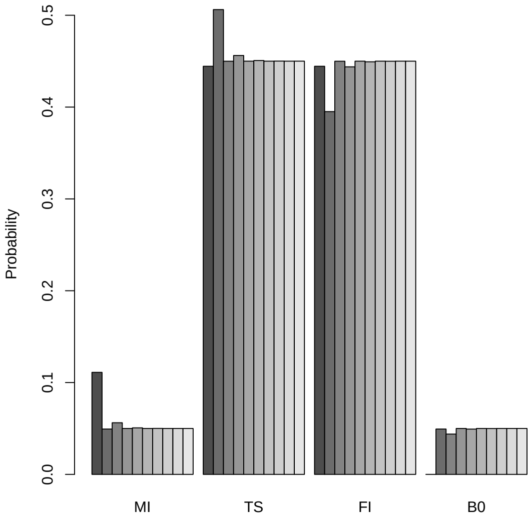
```
##             [,1]      [,2]      [,3]       [,4]
##  [1,] 0.11111111 0.4444444 0.4444444 0.00000000
##  [2,] 0.04938272 0.5061728 0.3950617 0.04938272
##  [3,] 0.05624143 0.4499314 0.4499314 0.04389575
##  [4,] 0.04999238 0.4561805 0.4438348 0.04999238
##  [5,] 0.05068672 0.4499992 0.4499992 0.04931498
##  [6,] 0.04999991 0.4506860 0.4493142 0.04999991
##  [7,] 0.05007622 0.4500000 0.4500000 0.04992380
##  [8,] 0.05000000 0.4500762 0.4499238 0.05000000
##  [9,] 0.05000847 0.4500000 0.4500000 0.04999153
## [10,] 0.05000000 0.4500085 0.4499915 0.05000000
```

Markov chain: A discrete state and time example

Then, we can visualise the evolution in time of the vector of probabilities

```r
rownames(pi.1_10) <- c('MI', 'TS', 'FI', 'BO')
barplot(t(pi.1_10), beside = T, ylab = "Probability")
```

Markov chain: A discrete state and time example



Markov chain: A discrete state and time example

By analysing the vector of probabilities at 50 and 100 steps, it emerges clearly that we reached a stationary point

```
pi50 <- pt(50, pi0, Tmat)
pi50
```

```
##          MI   TS   FI   BO
## [1,] 0.05 0.45 0.45 0.05
```

```
pi100 <- pt(100, pi0, Tmat)
pi100
```

```
##          MI   TS   FI   BO
## [1,] 0.05 0.45 0.45 0.05
```

Markov chain: A discrete state and time example

The stationary distribution to which our Markov Chain converges can be also obtained by leveraging on the eigendecomposition. Indeed the stationary distribution is

$$\pi P = \pi$$

By eigendecomposing P and taking the left eigenvectors, let's say v, corresponding to the eigenvalue $\lambda = 1$, suitably normalized, we obtain the stationary distribution

$$\pi = \frac{v}{\sum_{i=1}^{4} v_i}$$

- Left eigenvector of A: v s.t. $vA = \lambda v$
- Note: The left eigenvectors can be obtained as the transposes of the right eigenvectors (as below)

Markov chain: A discrete state and time example

```
eigendec <- eigen(t(Tmat))
eigendec$vectors
```

```
##             [,1]       [,2]       [,3] [,4]
## [1,] -0.07808688 -0.2236068  0.2236068  0.5
## [2,] -0.70278193  0.6708204  0.6708204 -0.5
## [3,] -0.70278193 -0.6708204 -0.6708204 -0.5
## [4,] -0.07808688  0.2236068 -0.2236068  0.5
```

```
eigendec$value
```

```
## [1]  1.0000000 -0.3333333  0.3333333 -0.1111111
```

```
p.distr <- eigendec$vectors[, 1] / sum(eigendec$vectors[, 1])
p.distr
```

```
## [1] 0.05 0.45 0.45 0.05
```

Markov chain

- Underlying idea: We aim to simulate from a stationary distribution and using a proper kernel (the transition matrix in the example) we obtain simulations from the stationary distribution
- The MCMC methods consider dependent draws from a Markov chain whose limiting distribution is the target distribution, that is the posterior distribution
- Note that simulations will not be independent!

Section 3

Metropolis-Hastings

Metropolis-Hastings

- Underlying idea: Given a target density $f(\theta)$, we build a transition kernel K with stationary distribution f and then generate a Markov chain using this kernel so that the limiting distribution is f

- Given the target density f we need to choose an instrumental/proposal distribution g easy to simulate from, whose support contains the support of f

- The **basic MH algorithm** consider generating θ^* from a conditional proposal distribution $g(\theta^*|\theta^{(s-1)})$

  - Instead, the *independent MH algorithm* $g(\theta^*|\theta^{(s-1)}) = g(\theta^*)$

- g can be almost arbitrary in that the only theoretical requirements are that the ratio $f(\theta^*)/g(\theta^*|\theta^{(s-1)})$ is known up to a constant independent of $\theta^{(s-1)}$

Metropolis-Hastings

Independent Metropolis-Hastings algorithm

- Initialize $\theta^{(0)}$ on a value of the support of the target distribution (f)
- At s iteration ($s = 1, \dots, S$)
  - generate a candidate θ^* from the proposal distribution $g(\theta^*)$ [Note that in the usual MH algorithm you have $g(\theta^*|\theta^{(s-1)})$].
  - Compute the acceptance probability as

$$\rho(\theta^{(s-1)}, \theta^*) = \min\left(1, \frac{f(\theta^*)g(\theta^{(s-1)})}{f(\theta^{(s-1)})g(\theta^*)}\right)$$

  - Generate $U \sim U(0,1)$: if $U < \rho(\theta^{(s-1)}, \theta^*)$, then $\theta^{(s)} = \theta^*$, otherwise $\theta^{(s)} = \theta^{(s-1)}$.

Note that the proposal distribution at time s does not depend on the value of the chain at time $s-1$ and that the ratio of proposal distributions in the acceptance ratio does not simplify.

Independent Metropolis-Hastings algorithm

Straightforward generalization of the AR method for simulating draws from the target density. However:

- The AR sample is iid, while the MS sample is not
- When you reject, the AR method discard that value, while the MH retain the value generated at the previous step
- The AR acceptance step requires the calculation of the upper bound c, not required by the MH

Metropolis-Hastings: t-student(4) - N(0,1)

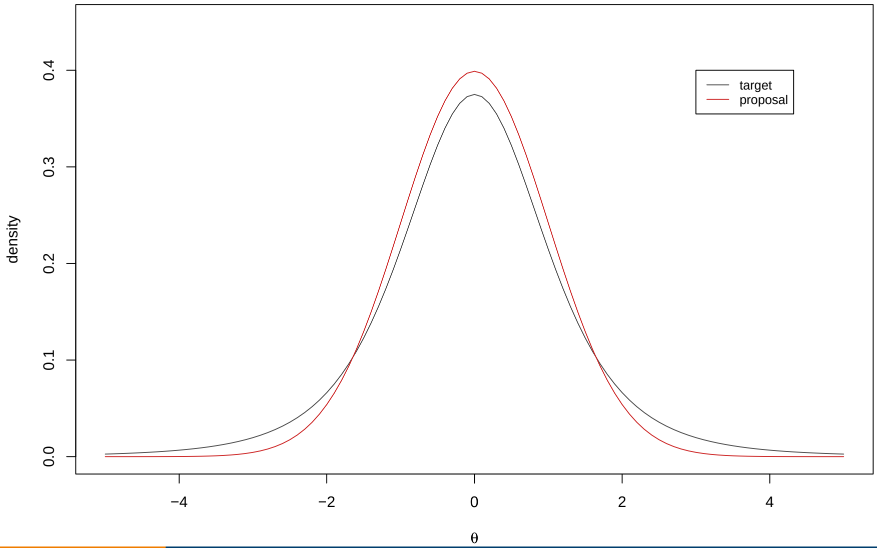
Consider to compute the mean of a Student's t distribution with 4 degrees of freedom ($\nu = 4$) using a Metropolis–Hastings algorithm with candidate density $\mathcal{N}(0,1)$. Then, monitor the convergence across iterations and compute the acceptance rate.

Recall that Student's t density with ν degrees of freedom is given by

$$f(\theta|\nu) = \frac{\Gamma\left(\frac{\nu+1}{2}\right)}{\Gamma\left(\frac{\nu}{2}\right)} \frac{1}{\sqrt{\nu\pi}} (1 + \theta^2/\nu)^{-(\nu+1)/2}$$

```
plot(NULL, NULL, xlab = expression(theta), ylab = "density",
     ylim = c(0, .45), xlim = c(-5, 5))
curve(dt(x, 4), col = "gray30", add = TRUE)
curve(dnorm(x),  col = "firebrick3", add = TRUE)
legend(3, .4, lty = c(1, 1), col = c("gray30", "firebrick3"),
       legend = c("target", "proposal"), cex = .8)
```

Metropolis-Hastings: t-student(4) - N(0,1)



Metropolis-Hastings: t-student(4) - N(0,1)

```r
Nsim <- 10^4
df_f <- 4
set.seed(123)
th <- rep(0, Nsim); th[1] <- rnorm(1)
acc.rate <- 0
for(s in 2 : Nsim){
    th_star <- rnorm(1)
    num <- dt(th_star, df_f) * dnorm(th[s - 1])
    den <- dt(th[s - 1], df_f) * dnorm(th_star)
    rho <- min(1, num / den)
    if ( runif(1) < rho ) {
        th[s] <-  th_star
        acc.rate <- acc.rate + 1
    } else {
        th[s] = th[s - 1]
    }
}
```

Metropolis-Hastings: t-student(4) - N(0,1)

Then, the mean of a *t*-student distribution with $\nu = 4$ by using the MH algorithm is simply given by

```
mean(th)
```

```
## [1] -0.03735362
```

and the acceptance rate is given

```
acc.rate/Nsim
```

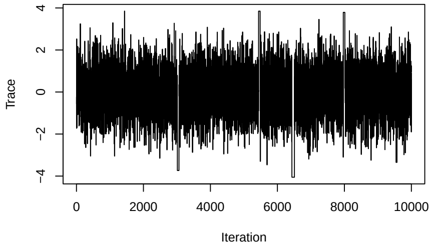
```
## [1] 0.9099
```

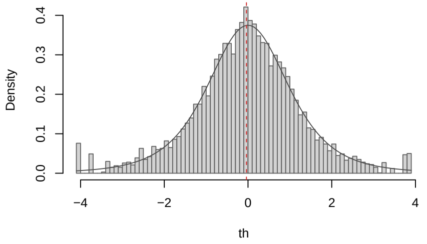
Below, you can see some graphical diagnostics

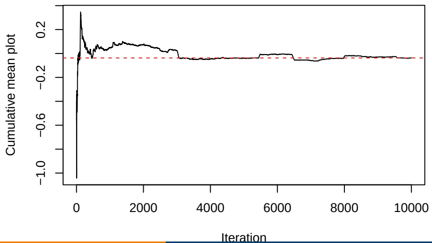
Metropolis-Hastings: t-student(4) - N(0,1)

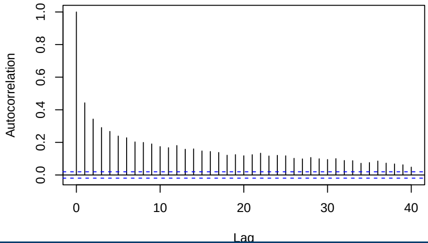
```r
##traceplot
plot(th, type = "l", ylab = "Trace", xlab = "Iteration")

##histogram
hist(th, breaks = 100, border = "gray40",freq = F, main = "")
curve(dt(x, df_f), col = "gray30", add = TRUE)
abline(v = mean(th), col = "firebrick3", lty = 2)

##cumulative mean
plot(cumsum(th)/ (1 : Nsim), type = "l",
     ylab = "Cumulative mean plot", xlab = "Iteration")
abline(h = mean(th), col = "firebrick3", lty = 2)

##autocorrelation
acf(th, main = "", ylab = "Autocorrelation")
```

Metropolis-Hastings: t-student(4) - N(0,1)







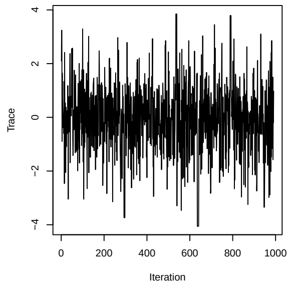


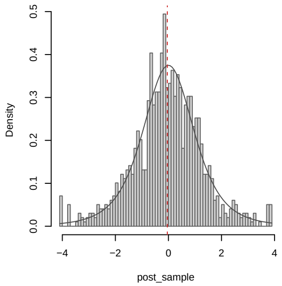
Metropolis-Hastings: t-student(4) - N(0,1)

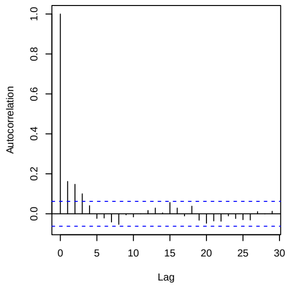
- Burn - in: discard an inital set of samples
- Thinning: break the dependence between draws

```r
##traceplot, histogram and autocorrelation
## after burn-in and thinning
post_sample <- th[seq(100, Nsim, by = 10)]

plot(post_sample, type = "l",
     ylab = "Trace", xlab = "Iteration")
hist(post_sample, breaks = 100, border = "gray40",
     freq = F, main = "")
curve(dt(x, df_f), col = "gray30", add = TRUE)
abline(v = mean(post_sample), col = "firebrick3", lty = 2)
acf(post_sample, main = "", ylab = "Autocorrelation")
```

Metropolis-Hastings: t-student(4) - N(0,1)







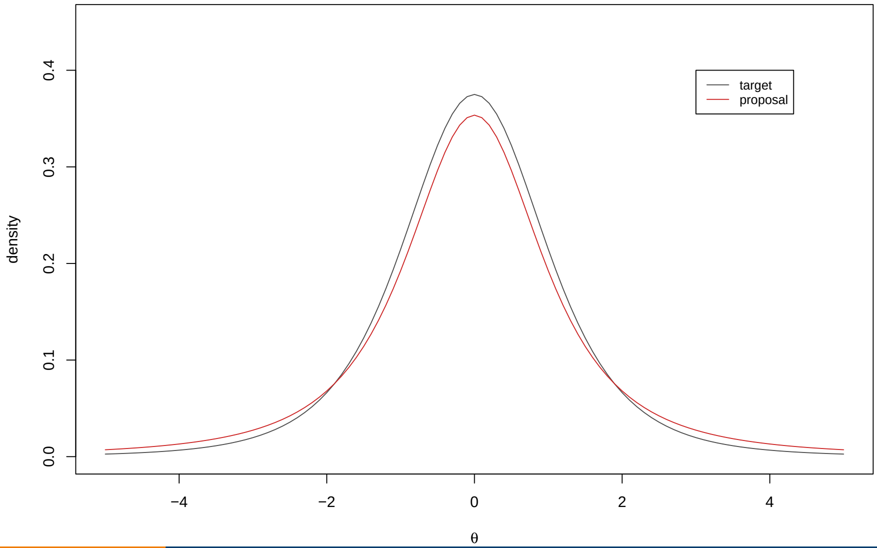
Metropolis-Hastings: t-student(4) - t-student(2)

Similarly, we compute the mean of a Student's t distribution with $\nu = 4$ using a Metropolis–Hastings algorithm with candidate density Student's t with $\nu = 2$

As above, we can monitor the convergence across iterations and compute the acceptance rate.

```r
plot(NULL, NULL, xlab = expression(theta), ylab = "density",
     ylim = c(0, .45), xlim = c(-5, 5))
curve(dt(x, 4), type = "l", col = "gray30")
curve(dt(x, 2), type = "l", col = "firebrick3")
legend(3, .4, lty = c(1, 1), col = c("gray30", "firebrick3"),
       legend = c("target", "proposal"), cex = .8)
```

Metropolis-Hastings: t-student(4) - t-student(2)



Metropolis-Hastings: t-student(4) - t-student(2)

```r
set.seed(123)
df_g <- 2 # degrees of freedom proposal
th <- rep(0, Nsim); th[1] <- rnorm(1)
acc.rate <- 0
for(s in 2 : Nsim){
    th_star <- rt(1, df_g)
    num <- dt(th_star, df_f) * dt(th[s - 1], df_g)
    den <- dt(th[s - 1], df_f) * dt(th_star, df_g)
    rho <- min(1,  num / den)
    if(runif(1) < rho){
       th[s] <- th_star
       acc.rate <- acc.rate + 1
    } else {
        th[s] <- th[s - 1]
    }
}
```

Metropolis-Hastings: t-student(4) - t-student(2)

Then, the mean of a t-student distribution with $\nu = 4$ by using the MH algorithm is simply given by

```
mean(th)
```

```
## [1] -0.003172407
```

and the acceptance rate is given

```
acc.rate/Nsim
```

```
## [1] 0.9161
```

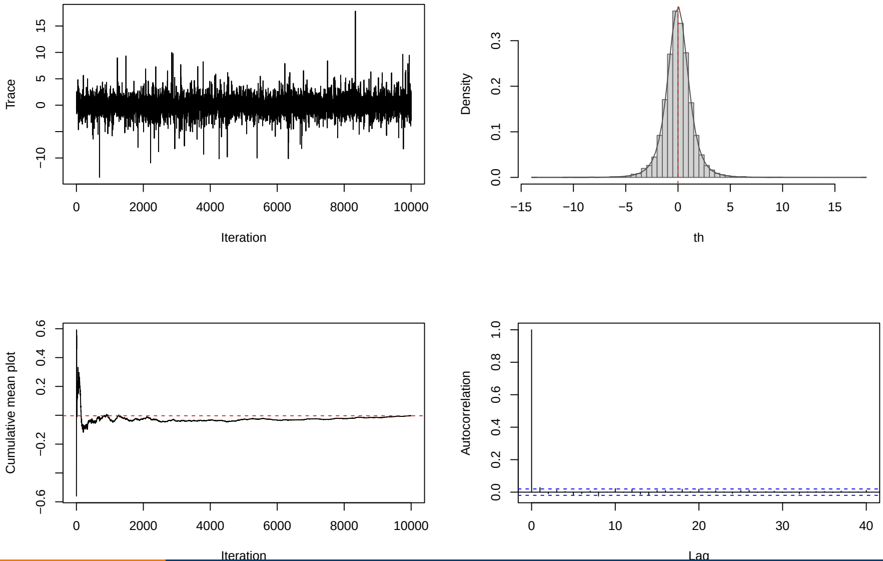
The latter is quite similar (a bit higher) to the acceptance rate by adopting the standard normal as candidate distribution

Below, you can see some graphical diagnostics

Metropolis-Hastings: t-student(4) - t-student(2)

```
##traceplot
plot(th, type = "l", ylab = "Trace", xlab = "Iteration")

##histogram
hist(th, breaks = 100, border = "gray40",freq = F, main = "")
curve(dt(x, df_f),  col = "gray30", add = TRUE)
abline(v = mean(th), col = "firebrick3", lty = 2)

##cumulative mean
plot(cumsum(th)/ (1 : Nsim), type = "l",
     ylab = "Cumulative mean plot", xlab = "Iteration")
abline(h = mean(th), col = "firebrick3", lty = 2)

##autocorrelation
acf(th, main = "", ylab = "Autocorrelation")
```

Metropolis-Hastings: t-student(4) - t-student(2)


Trace
Iteration
Density
th
Cumulative mean plot
Iteration
Autocorrelation
Lag