Domača naloga 2

Bayesova statistika

Alen Kahteran

5. 12. 2020

Implementacija algoritma Metropolis-Hastings

Najprej je potrebno pripraviti naše znane podatke

```
# samples for likelihood
x <- c(1.91, 1.94, 1.68, 1.75, 1.81, 1.83, 1.91, 1.95, 1.77, 1.98,
       1.81, 1.75, 1.89, 1.89, 1.83, 1.89, 1.99, 1.65, 1.82, 1.65,
       1.73, 1.73, 1.88, 1.81, 1.84, 1.83, 1.84, 1.72, 1.91, 1.63)

# sigma for likelihood
sig_like <- 0.1

# sigma for prior
sig_prior <- 0.2

# mu_0 for prior
mu_0 <- 1.78

# number of iterations for metropolis hastings
iter_num <- 40000
```

Ravno tako pripravimo naše funkcije za izračun verjetja pri določenem θ in znanih podatkih. Dodana je možnost za uporabo logaritma.

```
# defining likelihood function
verjetje <- function(theta, x, log_=FALSE) {
    if (log_) {
        # just here for testing
        # return(sum(log((1/(sqrt(2*pi)*sig_like))*exp(-(x-theta)^2/(2*sig_like^2)))))
        return(sum(dnorm(x, theta, sig_like, log=log_)))
    } else {
        # just here for testing
        # return(prod((1/(sqrt(2*pi)*sig_like))*exp(-(x-theta)^2/(2*sig_like^2))))
        return(prod(dnorm(x, theta, sig_like)))
    }
}

# defining prior function
apriorna <- function(theta, log_=FALSE){
    return(dnorm(theta, mu_0, sig_prior, log=log_))
}
```

Nato združimo vse skupaj v Metropolis-Hastings algoritem.

```r
# defining metropolis-hastings algorithm in a function
metro_hasti <- function(n_iter, theta_init, sig_prop, log_=FALSE){

    # vector for saving
    posterior <- rep(0, n_iter)
    # set first value to theta_init
    posterior[1] <- theta_init

    # loop n_iter - 1 times
    for(i in 2:n_iter){
        # change current_theta
        current_theta <- posterior[i - 1]

        # get new_theta (where proposal distribution is normal)
        new_theta <- current_theta + rnorm(1, 0, sig_prop)

        # calculate prior and likelihood for new theta
        prior_new <- apriorna(new_theta, log=log_)
        like_new <- verjetje(new_theta, x, log=log_)

        # calculate prior and likelihood for current theta
        prior_curr <- apriorna(current_theta, log=log_)
        like_curr <- verjetje(current_theta, x, log=log_)

        # calculate alpha
        if (log_) {
            A <- exp((prior_new + like_new) - (prior_curr + like_curr))
        } else {
            A <- (prior_new * like_new) / (prior_curr * like_curr)
        }

        # accept/reject new_theta based on alpha
        if(runif(1) < A){
            posterior[i] <- new_theta # "accept" move with probability min(1, A)
        } else {
            posterior[i] <- current_theta # otherwise "reject" move.
        }
    }
    # return a sample of posterior distribution
    return(posterior)
}
```

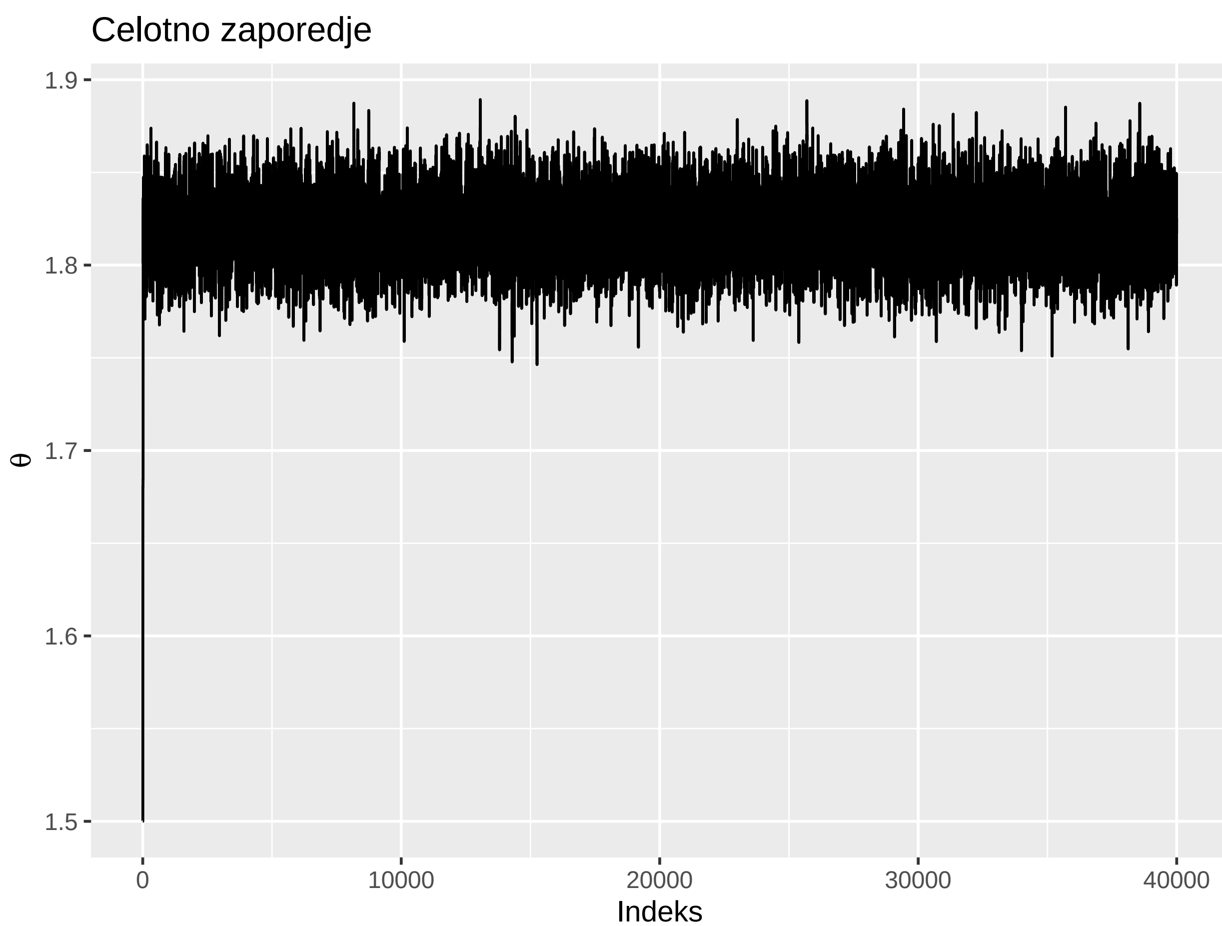
Preizkus algoritma

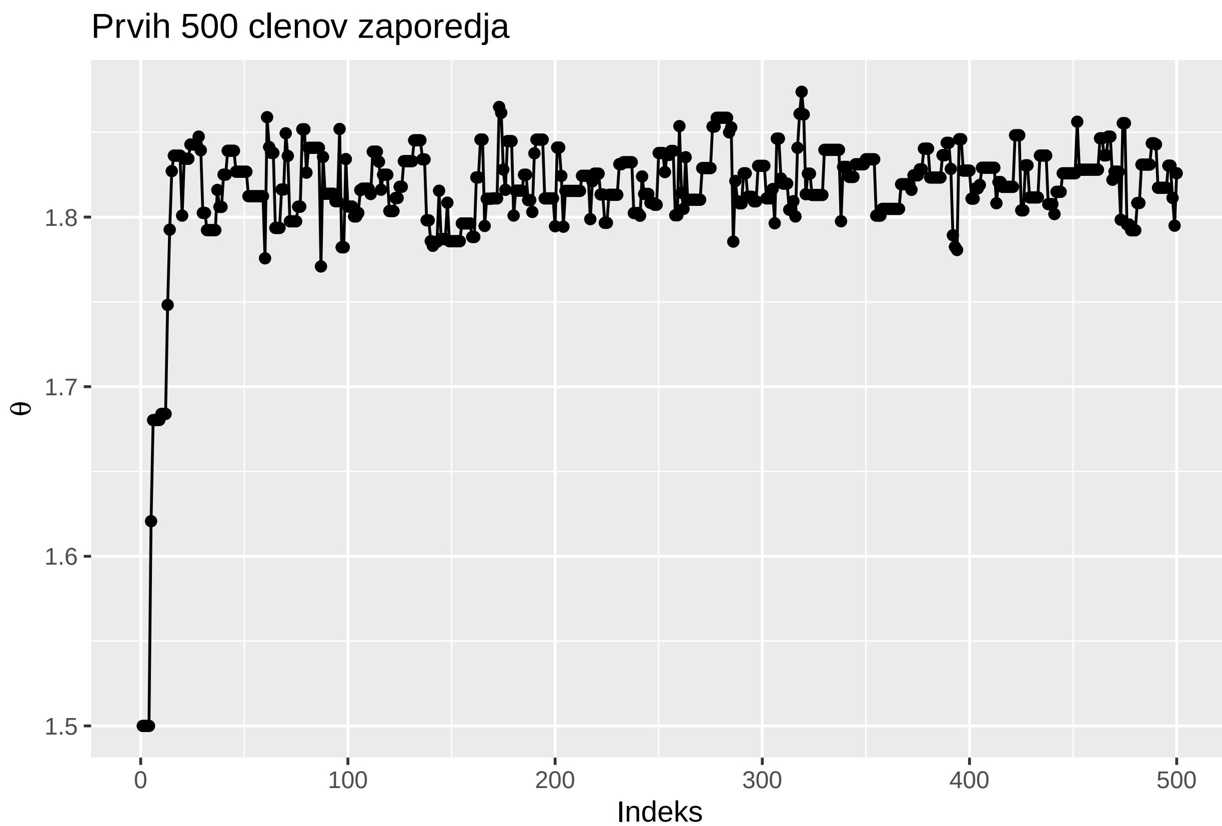
Najprej si poglejmo, kako algoritem deluje na naših podatkih iz vaj, z naslednjimi vrednostmi

$$
\begin{aligned}
\sigma^2 &= 0.1^2\,, \\
\mu_0 &= 1.78\,, \\
\sigma_0^2 &= 0.2^2\,, \\
n &= 40000\,, \\
\theta_z &= 1.5\,, \\
\sigma_p^2 &= 0.05^2\,,
\end{aligned}
$$

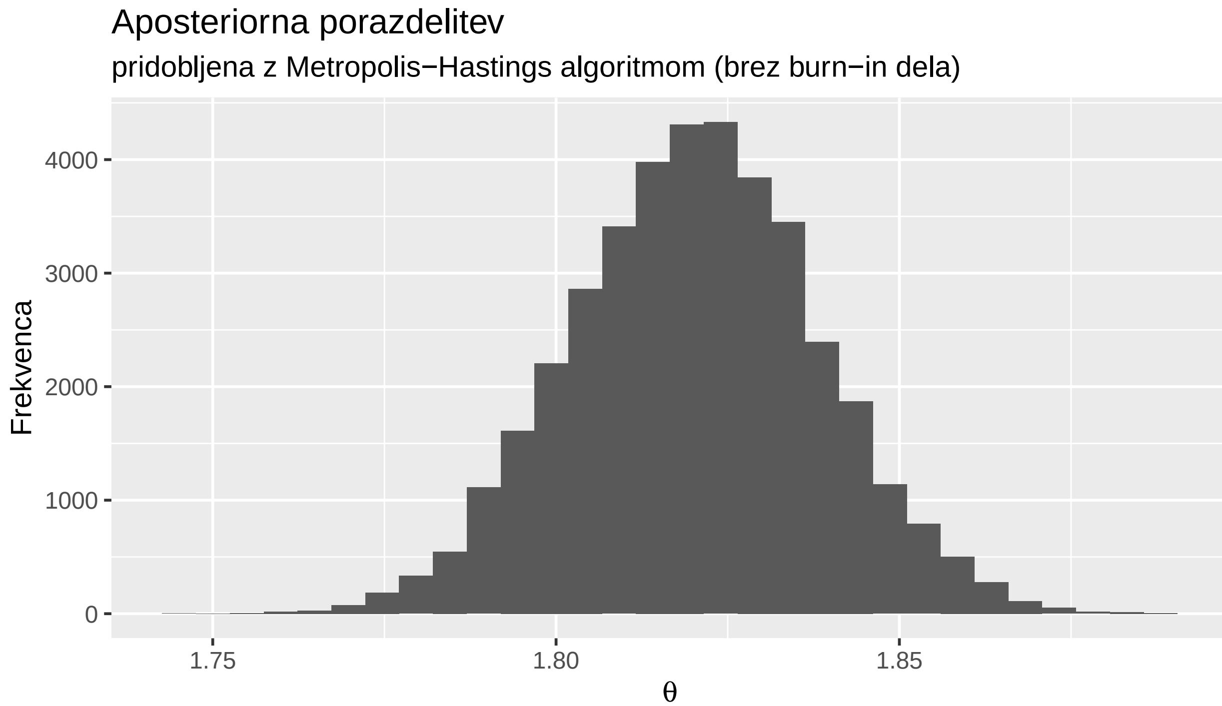
kjer θ_z predstavlja začetni (predlagalni) θ in σ_p^2 predstavlja varianco predlagalne porazdelitve. $\theta_z = 1.5$ je mogoče "nerealna" vrednost, vendar je nastavljena tako, da je opazen *burn-in*.

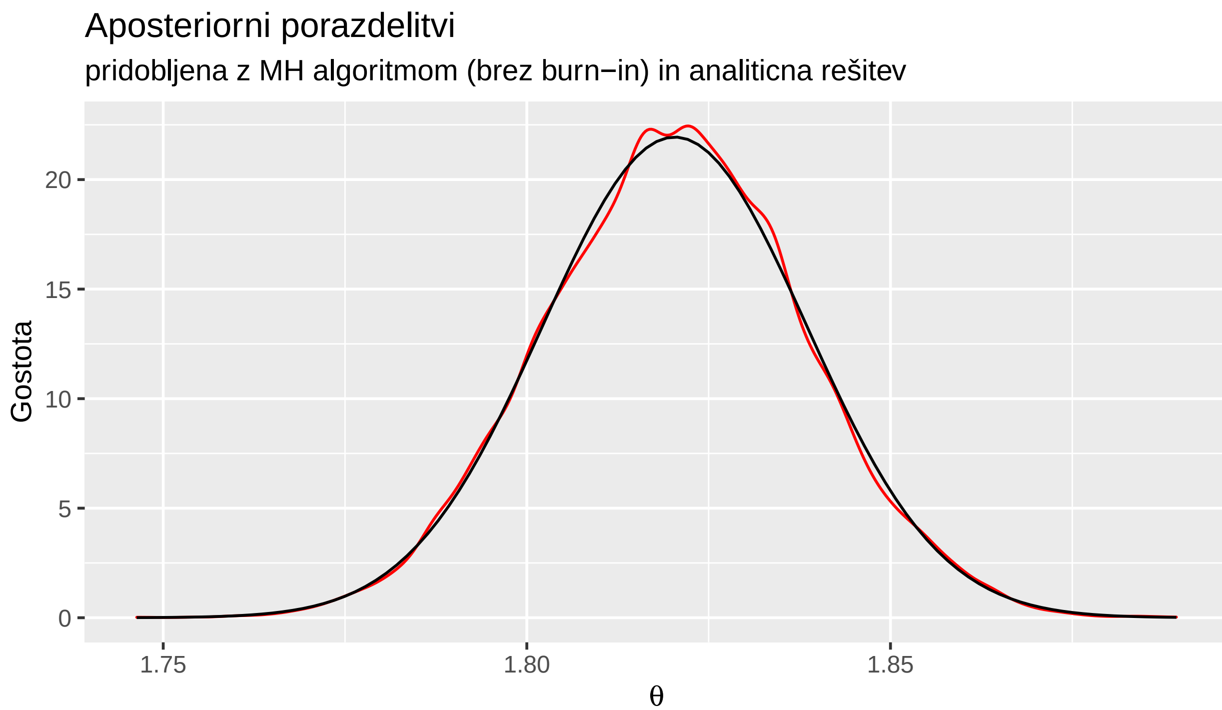
```
# setting seed for reproducibility
set.seed(9)

theta_start <- 1.5
sigma_prop <- 0.05

# get metropolis-hastings posterior chain for initial theta = 1.5, and proposal density
# standard deviation equal to 0.05 or variance equal to 0.05^2.
chain_df <- tibble(theta = metro_hasti(iter_num, theta_start, sigma_prop, log_=FALSE))
```




Prvih 500 clenov zaporedja
1.8
1.7
1.6
1.5
θ
0
100
200
300
400
500
Indeks


Brez prvih 500 clenov zaporedja, ki predstavljajo burn−in
1.85
1.80
1.75
θ
0
10000
20000
30000
40000
Indeks





Poglejmo si še našo oceno parametra θ in pripadajoč interval zaupanja za naš algoritem, ter za analitično rešitev. Zaradi simetričnosti porazdelitve lahko izberemo povprečje za oceno parametra θ.

	Povprečje	2.5%	97.5%
Metropolis-Hastings	1.82025	1.78538	1.85599
Analitično	1.82033	1.78469	1.85597

Nesmiselna začetna vrednost

Glede na podan problem, ko nas zanima višina študentov moškega spola, sem si izbral nesmiselno začetno vrednost $\theta_z = 0$, saj je nemogoče da bi bila dejanska višina enaka 0. Standardno deviacijo oz. varianco sem pustil enako kot pri prejšnjem poglavju, $\sigma_p = 0.05$ oz. $\sigma_p^2 = 0.05^2$. Pri zagonu naslednje funkcije

```
tmp <- metro_hasti(iter_num, 0, sig_prop, log_=FALSE)
```

se pojavi naslednja napaka

```
Error in if (runif(1) < A) { : missing value where TRUE/FALSE needed
```

Takoj vidimo da je problem v primerjavi `runif(1)` in `A` (α). Ker v `runif(1)` ne more biti težava, mora biti težava v izračunu `A` (α). Način kako izračunamo `A` (α) je naslednji

```
A <- (prior_new * like_new) / (prior_curr * like_curr)
```

Hitro vidimo težavo v primeru, ko je ali `prior_curr` ali `like_curr` enak 0, saj v tem primeru delimo z 0. Poglejmo si kaj nam vrnejo funkciji `apriorna()` in `verjetje()` za $\theta = 0$, saj jih uporabimo za izračun `prior_curr` in `like_curr`.

```
# calculate prior and likelihood for initial theta = 0
apriorna(0, log=FALSE)
```

```
## [1] 0.000000000000000001257903
```

```
verjetje(0, x, log=FALSE)
```

```
## [1] 0
```

Vidimo da je verjetje enako 0, in posledično takoj v prvi iteraciji, delimo z 0, kar predstavlja težavo. Če pa to spravimo na raven logaritma tj. da kjer se stvari množijo jih seštejemo, in kjer se delijo jih odštejemo, dobimo pa naslednje rezultate

```
# calculate prior and likelihood for initial theta = 0, where we transform to log
apriorna(0, log=TRUE)
```

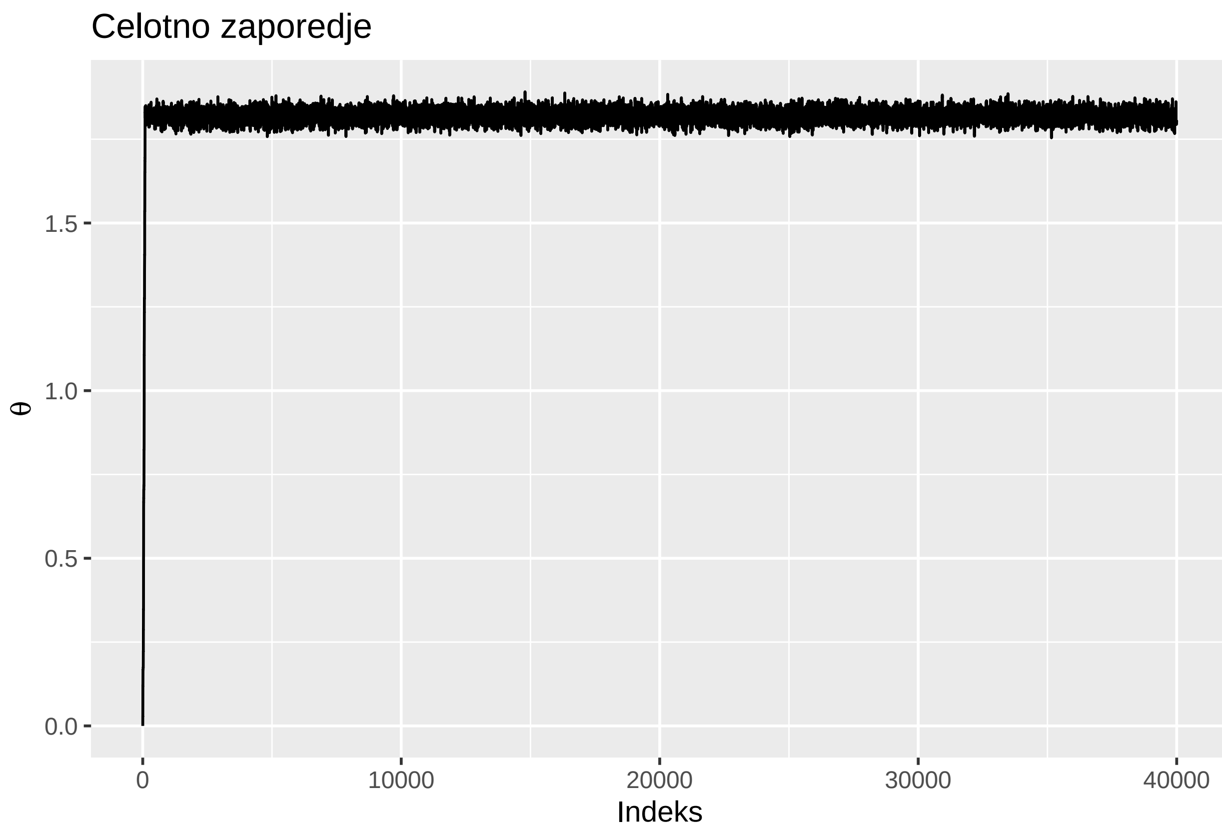
```
## [1] -38.9145
```

```
verjetje(0, x, log=TRUE)
```

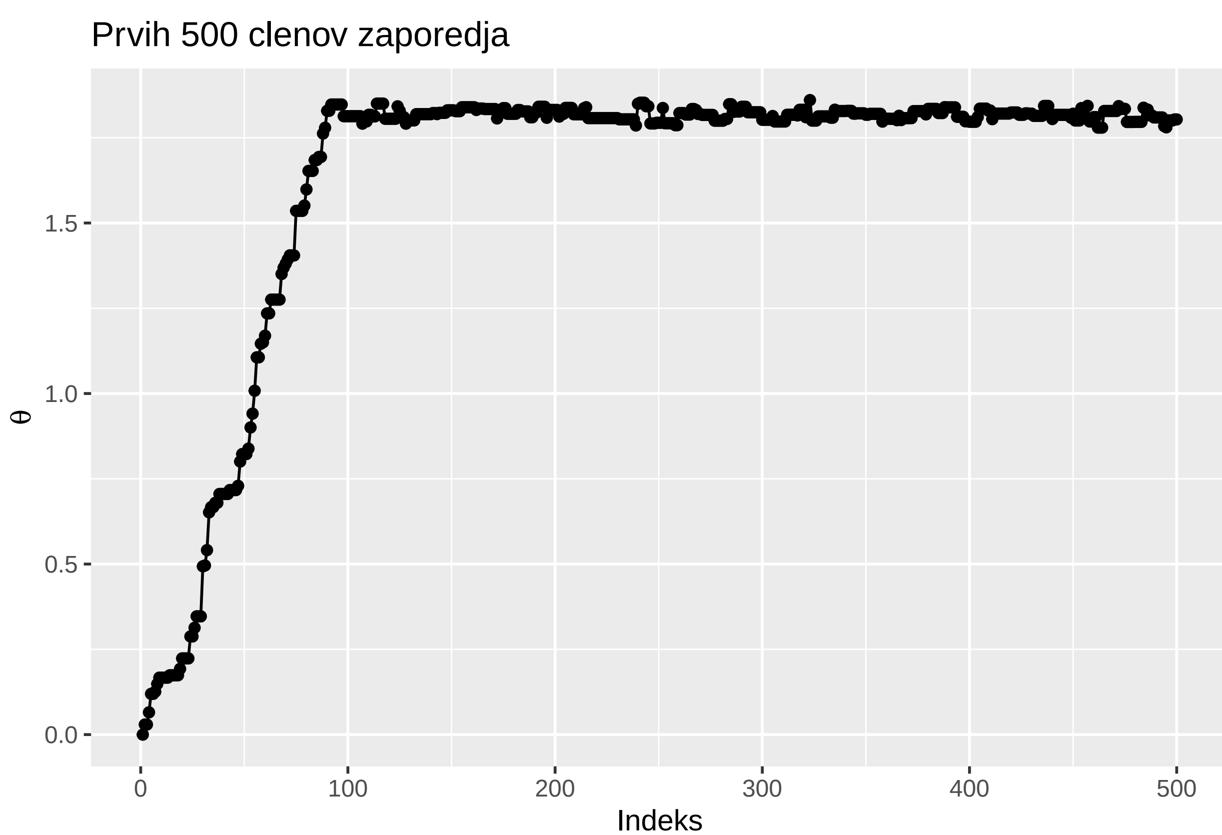
```
## [1] -4944.821
```

Se pravi logaritem nam pravzaprav reši težavo računalniške natančnosti. Poglejmo si sedaj naše zaporedje ko je začetna vrednost $\theta_z = 0$ in $\sigma_p = 0.05$, kjer uporabimo naš algoritem z logaritmom.

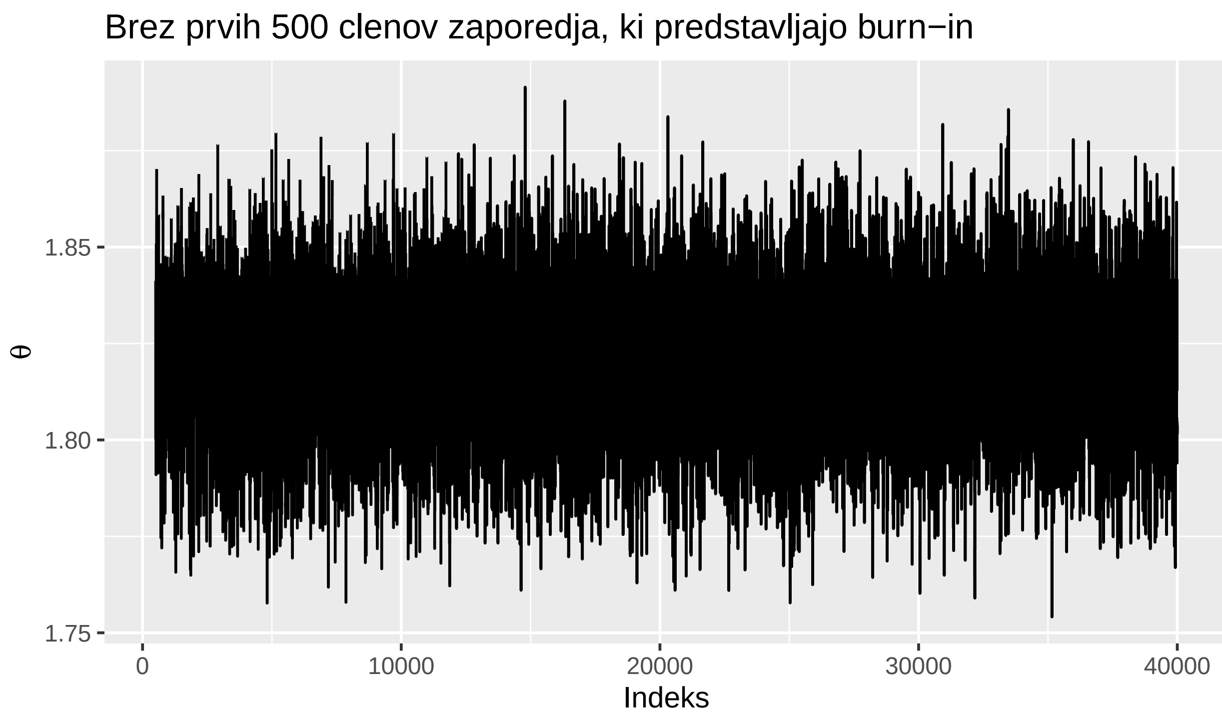
```
chain_df_log <- tibble(theta = metro_hasti(iter_num, 0, sigma_prop, log_=TRUE))
```



Celotno zaporedje
θ
1.5
1.0
0.5
0.0
0
10000
20000
30000
40000
Indeks



Prvih 500 clenov zaporedja
θ
1.5
1.0
0.5
0.0
0
100
200
300
400
500
Indeks



Primeri spreminjanja variance

Začetno vrednost bomo nastavili kot v začetnih primerih na $\theta_z = 1.5$, spreminjali bomo varianco. Odločil sem se za naslednje variance

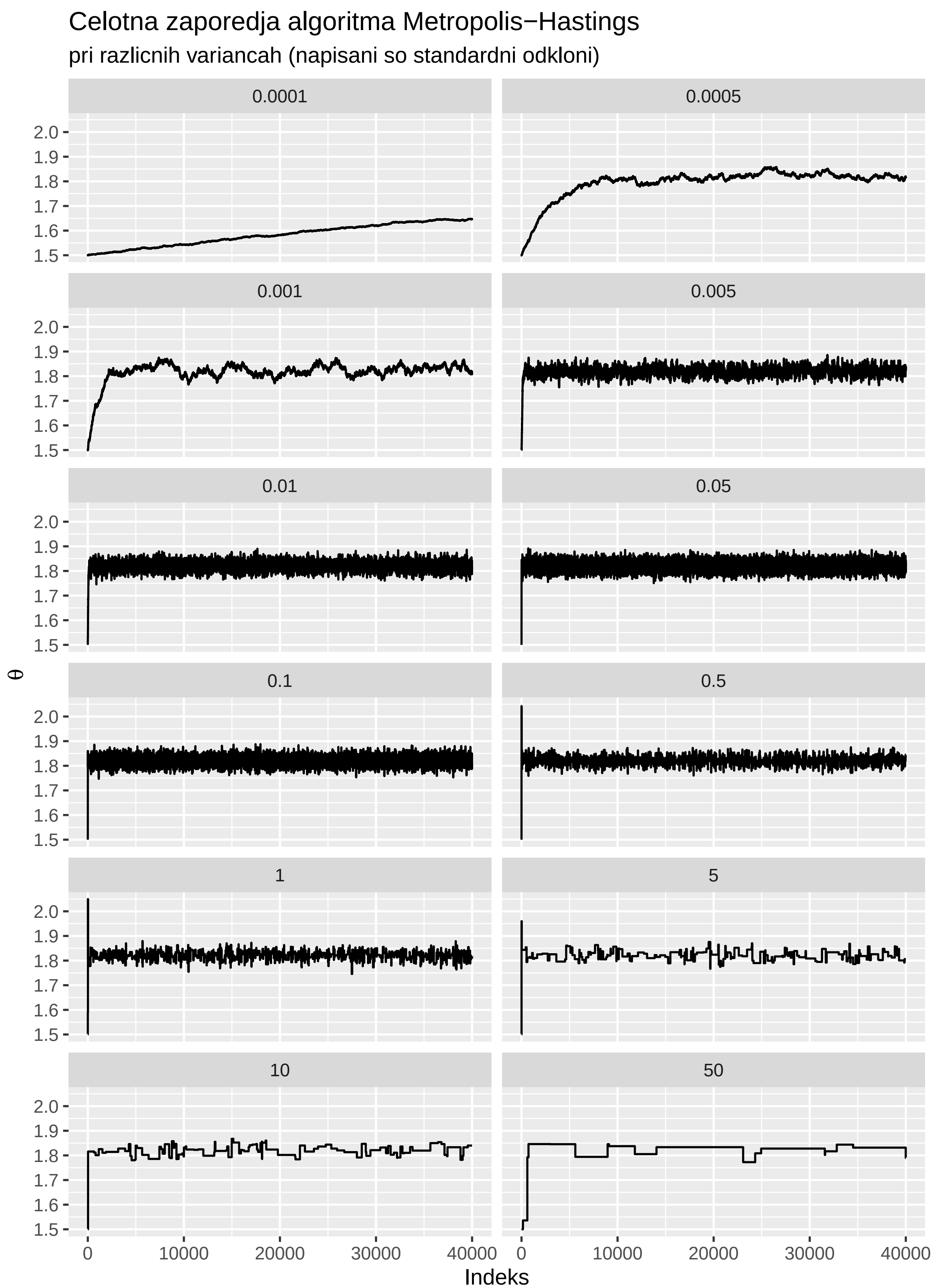
$$\sigma_p^2 \in \{0.0001^2, 0.0005^2, 0.001^2, 0.005^2, 0.01^2, 0.05^2, 0.1^2, 0.5^2, 1^2, 5^2, 10^2, 50^2\}$$

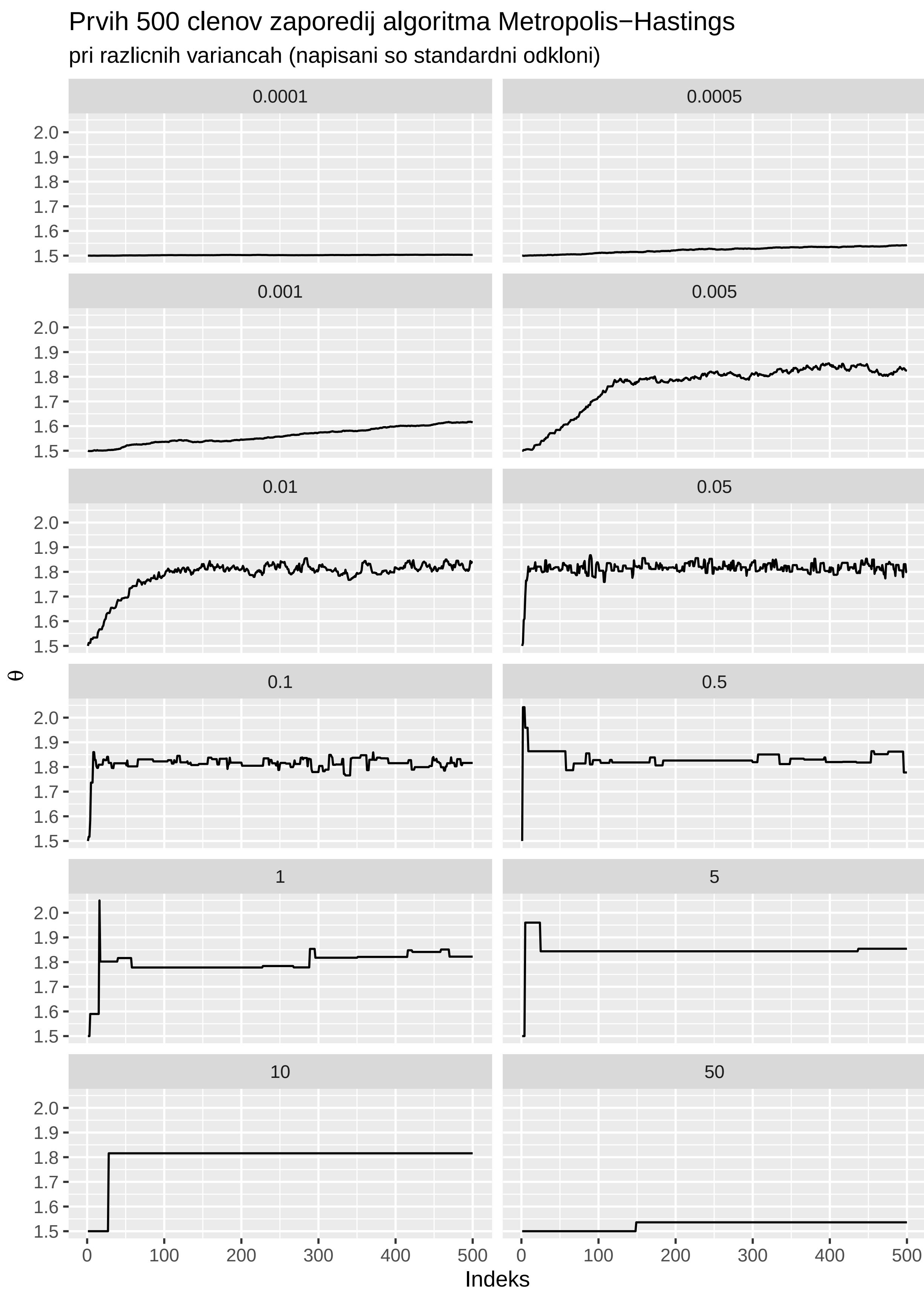
Če pogledamo slike celotnih zaporedij pri manjših variancah, vidimo da potrebujejo zelo dolgo časa, da konvergirajo. V primeru ko je ta enaka 0.0001^2, pravzaprav niti ne dosežemo konvergence. Pri ostalih pa vidimo da je "gostota" okrog "točke konvergence" različna. Pri ostalih dveh manjših variancah (0.0005^2 in 0.001^2), je videti da zaporedje še precej oscilira okrog te točke (ki je v našem primeru θ, ki ga iščemo). "Gostote" pri 0.005^2, 0.01^2, 0.05^2 in 0.1^2 izgledajo dokaj podobne, in je težko kaj sklepati iz teh slik. Pri 0.5^2 in 1^2 je videti da se ta "gostota" zopet manjša okrog naše točke konvergence. Pri 5^2, 10^2 in 50^2 se pa že opazi, da velikokrat ostanemo na istem θ. To je tudi smiselno, saj ko je varianca ogromna, imamo veliko več možnosti da izberemo θ, ki je "slabši" od trenutnega. Posledično se zgodi da zelo malokrat sprejmemo nov θ in ostanemo na mestu prejšnjega, ker je verjetnost, da bi se zgodilo da je θ ravno na območju kjer je ta "boljši", manjša.

Te stvari je tudi dobro videti na slikah kjer imamo samo 500 iteracij. Vidimo kot že omenjeno da pri 0.0001^2, 0.0005^2 in 0.001^2 vse skupaj zelo počasi konvergira in je težko kaj razbrati iz teh slik. Od 0.5^2 dalje vidimo podobno kot na prejšnjih slikah, da veliko časa stojimo na enem θ. Ostale bi pa ocenil kot dobre, kar je pa sicer odvisno od namena ki ga želimo doseči.

Po eni strani je cilj čim hitrejša konvergenca, po drugi pa da večkrat zamenjamo θ, da čimbolje opišemo aposteriorno porazdelitev. Konvergenca je hitrejša večja kot je varianca (do neke točke). Porazdelitev pa bolje opišemo če je varianca manjša. Se pravi je vse odvisno od danega problema.

Na podlagi tega, bi lahko mogoče ustvarili način, ki adaptivno manjša varianco predlagalne porazdelitve. Predpostavljam da obstaja več različnih načinov kako to doseči. Npr. na podlagi št. iteracij, na podlagi hitrosti konvergence, istočasni zagon algoritma pri več različnih začetnih vrednostih, ipd.. Sicer je tu mogoče potrebno paziti na ohranitev lastnosti markovskih verig (da je naslednje stanje neodvisno od tega kje smo bili v prejšnjih, kar bi bilo potrebno preveriti za prva dva primera).


Celotna zaporedja algoritma Metropolis−Hastings
pri razlicnih variancah (napisani so standardni odkloni)
0.0001
0.0005
0.001
0.005
0.01
0.05
0.1
0.5
1
5
10
50
2.0
1.9
1.8
1.7
1.6
1.5
θ
0
10000
20000
30000
40000
Indeks


Prvih 500 clenov zaporedij algoritma Metropolis−Hastings
pri razlicnih variancah (napisani so standardni odkloni)
0.0001
0.0005
0.001
0.005
0.01
0.05
0.1
0.5
1
5
10
50
θ
Indeks
2.0
1.9
1.8
1.7
1.6
1.5
0
100
200
300
400
500