An Introduction to Estimating Monte Carlo Standard Errors with R Package `mcmcse`

Dootika Vats

August 29, 2018

Contents

1 Introduction

The R package `mcmcse` provides estimates of Monte Carlo standard errors for Markov chain Monte Carlo (MCMC). This package is useful when estimating means and quantiles of functions of the MCMC output. In addition to MCMC output, the package can be used for time series and other correlated processes.

The package is predominantly useful after MCMC output has been obtained by the user. In addition to estimating the Monte Carlo standard errors, the package also provides univariate and multivariate estimates of effective sample size and tools to determine whether enough Monte Carlo samples have been obtained. There are also some graphical tools to ascertain the behavior of the Monte Carlo estimates.

2 An MCMC Example

To illustrate the use of our package, we present the following simple multivariate AR(1) process. The process is defined for $t = 1, 2, 3, \ldots$ as,

$$y_t = w + Ay_{t-1} + \epsilon_t,$$

where w is a constant vector in $\mathbb{R}^p$, $y_t \in \mathbb{R}^p$, A is a $p \times p$ matrix and $\epsilon_t \sim N_p(0, C)$. In our example, we let A and C be diagonal matrices. The invariant distrbution for this process is $F = N_p(0, V)$ where V is a function of A and C.

The function `mAr.sim` in package `mAr` draws samples from the above model. We let $p = 3$.

```
library(mAr)
p <- 3
A <- diag(c(.1, .5, .8))
C <- diag(rep(2, 3))

# Monte Carlo sample size is N
N <- 1e5
set.seed(100)
chain <- mAr.sim(w = rep(2,p), A = A, C = C, N = N)
```

For using the `mcmcse` package the rows of the MCMC output should store each iteration of the algorithm. Thus the output should have n rows and p columns. We will denote each row i of the output as $(y_i^{(1)}, y_i^{(2)}, y_i^{(3)})$.

```
#Rows has observations (samples) and each comlumn is a component.
head(chain)
```

```
          Y1       Y2        Y3
1 -0.3101768 3.967189  8.421232
2  5.2384324 6.391334 10.407224
3  2.5886888 6.497850 11.156644
4  3.8339966 4.992644 12.447253
5 -0.2396186 4.900606 13.565273
6  2.2108764 5.144180 13.927434
```

This vignette will discuss estimating two sets of features of interest of F.

- $\mathrm{E}_F y$: For estimating $\mu = \mathrm{E}_F y$, the estimator is the Monte Carlo sample mean

$$\mu_n = \frac{1}{n}\sum_{t=1}^{n} y_t.$$

  In `R`, μ_n is obtained using the usual `colMeans` function. If $p = 1$, then use `mean` instead oc `colMeans`.

```
colMeans(chain)
```

```
      Y1        Y2        Y3
2.225577  4.012947 10.007805
```

- $\mathrm{E}_F(y^{(1)2} + y^{(2)2} + y^{(3)2})$: When interested in estimating the sum of the second moments of each component of y, define the function $g : \mathbb{R}^3 \to \mathbb{R}$ as $g((x_1, x_2, x_3)) = x_1^2 + x_2^2 + x_3^2$. This is defined in `R` by creating a function that takes a vector argument.

```
g <- function(x)
{
        return(sum(x^2))
}
```

  The Monte Carlo estimator for g is

$$\mu_{g,n} = \frac{1}{n}\sum_{t=1}^{n} g(y_t),$$

```r
# Apply the function g to each row
gofy <- apply(chain, 1, g)

# Monte Carlo estimate
mean(gofy)

[1] 131.4661
```

Thus, to obtain Monte Carlo estimates from MCMC output, the base package is sufficient (generally). However, Monte Carlo estimates must be reported with Monte Carlo standard error. That is, if the following central limit theorems hold

$$\sqrt{n}(\mu_n - \mu) \xrightarrow{d} N_p(0, \Sigma)\,, \tag{1}$$

and

$$\sqrt{n}(\mu_{g,n} - \mu) \xrightarrow{d} N_p(0, \Sigma_g)\,, \tag{2}$$

then estimates of Σ and Σ_g must be reported. Since the samples obtained are correlated, these quantities require more sophisticated tools than usual sample estimators. (Note that a Markov chain CLT is not always guaranteed to hold. In fact, it depends on the rate of convergence of the Markov chain. Most of the functions in this package assume that a Markov chain CLT holds. Such an assumption is also made when using many of the convergence diagnostics).

3 Estimating Monte Carlo Standard Error

In this package, the functions `mcse`, `mcse.mat`, `mcse.multi`, and `mcse.initseq` estimate the Monte Carlo standard error of μ_n (or $\mu_{g,n}$).

- `mcse`: consistent estimates of $\sqrt{\Sigma/n}$ (standard error) when Σ is 1×1.
- `mcse.mat`: consistent estimates of the square root of the diagonals of Σ/n.
- `mcse.multi`: consistent estimates of Σ.
- `mcse.initseq`: asymptotically conservative estimates of Σ using initial sequence estimators.

Using the `mcmcse` package we can estimate Σ in (1) with the `mcse.multi` and `mcse.initseq` function.

```
library(mcmcse)
mcerror_bm <- mcse.multi(x = chain, method =  "bm",
        size = "sqroot", g = NULL, level = .95, large = FALSE)
mcerror_bart <- mcse.multi(x = chain, method =  "bartlett",
        size = "cuberoot", g = NULL, level = .95, large = FALSE)
mcerror_tuk <- mcse.multi(x = chain, method =  "tukey",
        size = "sqroot", g = NULL, level = .95, large = FALSE)
mcerror_is <- mcse.initseq(x = chain, g = NULL,
                           level = .95, adjust = FALSE)
mcerror_isadj <- mcse.initseq(x = chain, g = NULL,
                              level = .95, adjust = TRUE)
```

- `x` takes the $n \times p$ MCMC data. `x` can take only numeric entries in the form of a matrix or data frame. The rows of `x` are the iterations of the MCMC.
- `method = ''bm''`, `''bartlett''`, `''tukey''` calculates the estimate using the batch means method and spectral variance methods with the modified-Bartlett and Tukey-Hanning windows.
- `size` is the batch size for the `bm` method and the truncation point for `tukey` and `bartlett` methods. `size = ''sqroot''` sets the size as $\lfloor \sqrt{n} \rfloor$ and `size = ''cuberoot''` sets it at $\lfloor n^{1/3} \rfloor$. An integer value of `size` less than n is also valid.

  For reference on `bm` (batch means estimators) see Jones et al. (2006) and Vats et al. (2017a).

  For reference on `bartlett` and `tukey` (spectral variance estimators)see Flegal et al. (2010) and Vats et al. (2017b).
- `g` is a function that is applied to each row of `x` and represents the features of interest of the process. Since here we are interested in only means, `g` is `NULL`. `g` will be explained in later examples.
- `level` is the confidence level of the resulting confidence region. This is required to calculate the volume of the confidence region.
- `large` is a logical argument. If `large` is `TRUE` the volume of the confidence region is the large sample volume obtained using χ^2 critical values. By default, volume is calculated using F distribution critical values.

- `adjust` is a logical argument only used for the `mcse.initseq` function. If `adjust` is `TRUE`, the eigenvalues of the initial sequence estimator are increased slightly.

  For reference on `mcse.initseq` (initial sequence estimators) see Dai and Jones (2017).

`mcse.multi` and `mcse.initseq` return a list with multiple components. `cov` stores the estimate of Σ obtained using the method chosen, `vol` returns the volume to the pth root of the resulting confidence region, `est` stores the estimate of g applied on the Markov chain and `nsim` stores the arguments used to calculate Σ. `mcse.multi` also returns `size` which indicates the size of batches/truncation, `method` used, and whether a `large` sample volume is returned. `mcse.initseq` also returns `cov.adj`, `vol.adj`, and whether an adjusted estimator was used (`adjust`).

```
mcerror_bm$cov

            [,1]        [,2]       [,3]
[1,]  2.54156743 -0.03432219  0.1910580
[2,] -0.03432219  7.21141694  0.8609699
[3,]  0.19105799  0.86096994 48.1650871

mcerror_bart$cov

            [,1]        [,2]        [,3]
[1,]  2.46757319 -0.09722749  0.07579761
[2,] -0.09722749  7.80304805 -0.08045297
[3,]  0.07579761 -0.08045297 45.63237887

mcerror_tuk$cov

           [,1]       [,2]       [,3]
[1,]  2.5776854 -0.2770822  0.3590341
[2,] -0.2770822  7.5494259  0.2492997
[3,]  0.3590341  0.2492997 48.3199332

mcerror_is$cov

            [,1]        [,2]        [,3]
[1,]  2.5029696 -0.10902926  0.11007175
[2,] -0.1090293  8.18913933  0.02202062
[3,]  0.1100717  0.02202062 50.96199582
```

```
mcerror_isadj$cov.adj

             [,1]         [,2]         [,3]
[1,]  2.58127094 -0.060426547  0.083477237
[2,] -0.06042655  8.242248961 -0.002410464
[3,]  0.08347724 -0.002410464 51.034334815

rbind(mcerror_bm$est, mcerror_bart$est, mcerror_tuk$est,
      mcerror_is$est, mcerror_isadj$est)

           Y1       Y2       Y3
[1,] 2.225577 4.012947 10.00781
[2,] 2.225577 4.012947 10.00781
[3,] 2.225577 4.012947 10.00781
[4,] 2.225577 4.012947 10.00781
[5,] 2.225577 4.012947 10.00781

c(mcerror_bm$vol, mcerror_bart$vol, mcerror_tuk$vol,
  mcerror_is$vol, mcerror_isadj$vol)

[1] 0.04450036 0.04409869 0.04456552 0.04538659 0.04538659
```

Note: the Monte Carlo estimates of μ are not affected by the choice of the method.

Note: the initial sequence estimators of Σ have larger volume than the consistent estimators. This is expected since initial sequence estimators are intentionally conservative.

Note: for consistent estimation, the batch means estimators are significantly faster to calculate than the spectral variance estimators. The user is advised to use the default `method = ''bm''` for large input matrices.

Note: `cov` returns an estimate of Σ and not Σ/n.

If the diagonals of Σ are σ_{ii}^2, the function `mcse` and `mcse.mat` returns $\sigma_{ii}/\sqrt{n}$. `mcse` does it for one component and `mcse.mat` does it for all diagonals.

```
mcse(x = chain[,1], method = "bm", g = NULL)
```

```
$est
[1] 2.225577

$se
[1] 0.005041396

mcse.mat(x = chain, method = "bm", g = NULL)

         est          se
Y1  2.225577 0.005041396
Y2  4.012947 0.008492006
Y3 10.007805 0.021946546
```

In order to estimate $\mu_{n,g}$ and Σ_g as in (2), we use the R function g we had defined before. Recall that g should be a function that takes vector inputs.

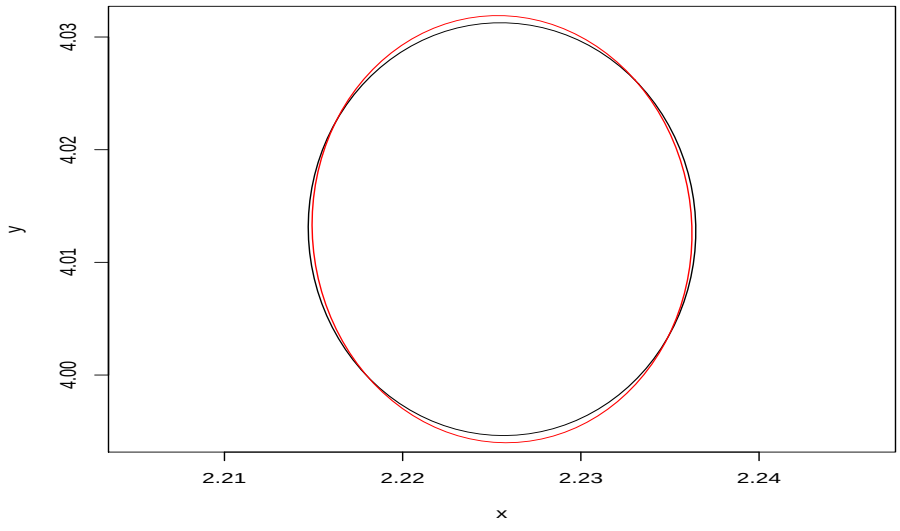
```
g

function(x)
{
return(sum(x^2))
}
<bytecode: 0x7f87ac743488>

mcerror_g_bm <- mcse.multi(x = chain, g = g)
mcerror_g_is <- mcse.initseq(x = chain, g = g)

mcerror_g_bm$cov

         [,1]
[1,] 20258.01

# Initial Sequence error is larger than batch means, as expected.
mcerror_g_is$cov

         [,1]
[1,] 21293.28

# Returned value is asymptotic variance.
# So we calculate the standard error here.
sqrt(mcerror_g_bm$cov/N)
```

```
          [,1]
[1,] 0.450089

sqrt(mcerror_g_is$cov/N)

           [,1]
[1,] 0.4614465
```

4 Confidence Regions

Using the function `confRegion` in the package, the user can create joint confidence regions for two parameters. The input for this function is the output list from the `mcse.multi` or `mcse.initseq` function. The function uses the attributes cov, `est`, and `nsim` from the output list. If the `mcse.initseq` is input and `adjust = TRUE` had been used, then `cov.adj` is used instead of `cov`. mcse.multi also uses the attribute `size`.

```
plot(confRegion(mcerror_bm, which = c(1,2), level = .90), type = 'l', asp = 1)
lines(confRegion(mcerror_bart, which = c(1,2), level = .90), col = "red")
```



- `which` should be a vector of size 2 that indicates the two components for which the confidence ellipse is to be constructed.

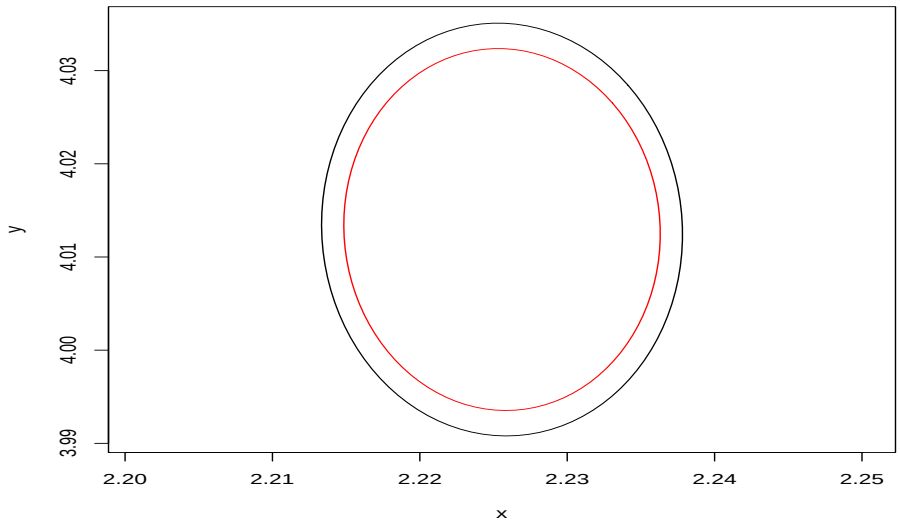
- `level` is the confidence level of the confidence region. The default is .95

NOTE: The argument `confRegion` calls on the function `ellipse` in package `ellipse` to draw the ellipse.

NOTE: Since the confidence region is created for two parameters only, the size of the ellipse is determined by setting $p = 2$ irrespective of the original dimension of the problem.

To determine the effect of the confidence level, we draw two regions with difference confidence levels. We use `mcse.initseq` this time.

```
plot(confRegion(mcerror_is, which = c(1,2), level = .95), type = 'l', asp = 1)
lines(confRegion(mcerror_is, which = c(1,2), level = .90), col = "red")
```



5 Effective Sample Size

Reporting $p \times p$ covariance matrix estimates is impractical and uninterpretable. The motivation of estimating Monte Carlo standard error is to ensure that said error is small. This is essentially the idea behind estimating effective sample size and ensuring that the estimated effective sample size is larger than a prespecified lower bound.

Before sampling the Markov chain, the user is advised to used the function `minESS` to ascertain what is the minimum effective sample size needed for stable analysis. See Vats et al. (2017a) for theoretical details.

```
# For mu
minESS(p = 3, alpha = .05, eps = .05)

minESS
  8123

#For mu_g
minESS(p = 1, alpha = .05, eps = .05)

minESS
  6146
```

- `p` is the dimension of the estimation problem.
- `alpha` is the confidence level
- `eps` is the tolerance level. Default is .05. Reasonable levels are anywhere from .01 to .05. The smaller the tolerance, the larger the minimum effective samples. `eps` represents a tolerance level relative to the variability in the target distribution. It is akin to the idea of margin-of-error.

`minESS` is independent of the Markov chain or process, and is only a function of the p, α, and ϵ. The user should find `minESS` and then sample their process until the required minimum samples are achieved.
Alternatively, we often don't have to luxury of obtaining a lot of samples, and reaching a minimum effective sample size os not possible. In such a scneario, it is useful to know the ϵ tolerance level the number of estimated effective samples correspond to. So if we can only obtain 1000 effective samples,

```
# For mu
minESS(p = 3, alpha = .05, ess = 1000)

  Epsilon
0.1425016
```

```
#For mu_g
minESS(p = 1, alpha = .05, ess = 1000)

 Epsilon
0.123959
```

Thus, if you obtained a sample with estimates effective sample size equaling 1000 for estimating μ_g and $\mu_{n,g}$, then the precision level of your estimate is $\epsilon = .143$ and $\epsilon = .124$, respectively. `multiESS` and `ess` are two functions that calculate the effective sample size of a correlated sample. `ess` calculations are based on Gong and Flegal (2016) and is component-wise, and `multiESS` utilizes the multivariate nature of the problem.

```
ess(chain)

      Y1       Y2       Y3
79119.49 36626.45 11628.72
```

Since `ess` produces a different estimate for each component, conservative practice dictates choosing the smallest of the values. `multiESS` returns one estimate of the effective sample size based on the whole sample. The function calls `mcse.multi` function to obtain a batch means estimate of Σ. The user can provide another estimate of Σ using the `covmat` argument.

```
multiESS(chain)

[1] 32327.07

# Using spectral variance estimators
multiESS(chain, covmat = mcerror_bart$cov)

[1] 32356.83

# Using initial sequence estimators
# Since this is a conservative estimator, ess will be smaller
multiESS(chain, covmat = mcerror_is$cov)

[1] 30544.9
```

Since the effective sample size is less than the minimum effective samples, we should simulate more. Looking at the ratio of the Monte Carlo samples size of 10^4 and multiESS, we might need around $28,000$ Monte Carlo samples.

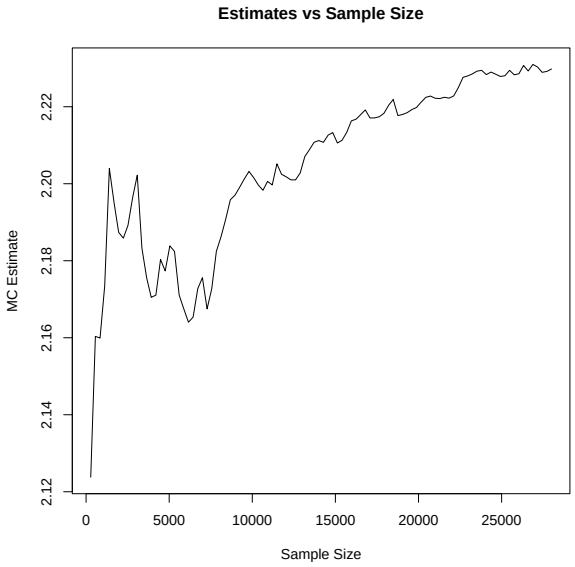
```
set.seed(100)
chain <- mAr.sim(w = rep(2,p), A = A, C = C, N = 28000)

# larger than 8123
multiESS(chain)

[1] 8832.498

# larger than 8123
multiESS(chain, covmat = mcerror_bart$cov)

[1] 9085.629

# larger than 8123
multiESS(chain, covmat = mcerror_is$cov)

[1] 8576.849
```

So no matter which estimator we choose for the Monte Carlo standard error, 28000 Monte Carlo samples are sufficient to have $\epsilon = .05$ relative tolerance. **NOTE:** Ideally, we do not want to get more samples using the last iteration of the previous Markov chain. However, `mAr.sim` does not allow user specified starting values, so to demonstrate the use of `minESS` and `multiESS`, we get a new sample altogether. When making `R` packages that simulate a Markov chain, it is often very useful to allow user specific starting values for this reason.

6 Graphical Diagnostics

The function `estvssamp` plots the Monte Carlo estimates versus the sample size for a component of the MCMC output. This plot indicates whether the Monte Carlo estimate has stabilized.

```
estvssamp(chain[,1])
```

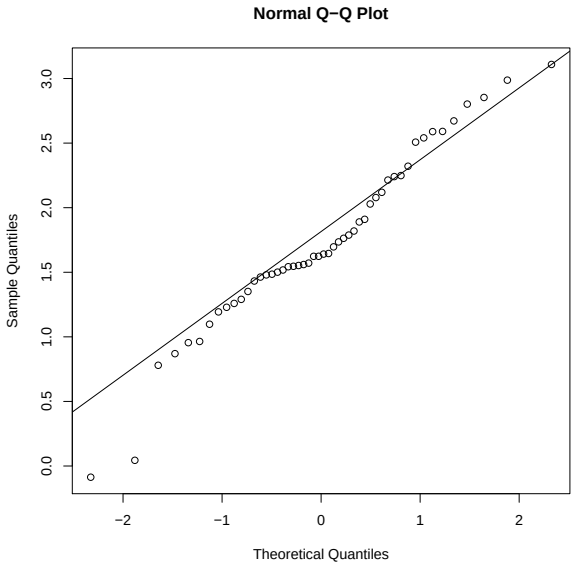


Additionally, if p is not too small, due to the central limit theorem in (1) and an estimate of Σ using the `mcse.multi` function, a QQ plot of the standardized estimates gives an idea of whether asymptopia has been achieved. We generate a new Markov chain with $p = 50$.

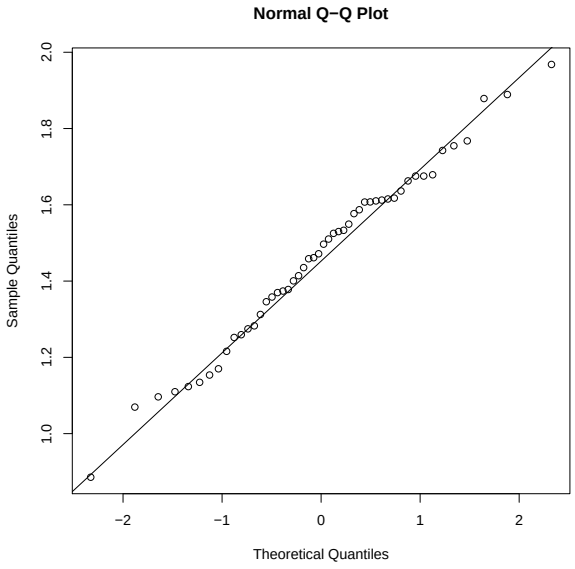
```
p <- 50
A <- diag(seq(.1, .9, length = p))
C <- diag(rep(2, p))

set.seed(100)
chain <- mAr.sim(w = rep(2,p), A = A, C = C, N = 10000)
```

For this new Markov chain, we find an estimate of Σ to use for the `qqTest` function.

```
mcerror_bm <- mcse.multi(chain, method = "bm")
mcerror_isadj <- mcse.initseq(chain, adjust = TRUE)
qqTest(mcerror_bm)
```




```
qqTest(mcerror_isadj)
```



Thus, we see here that the chain has not quite reached asymptopia.