STAT 812: Computational Statistics

Importance Sampling

Longhai Li

September 2018

Contents

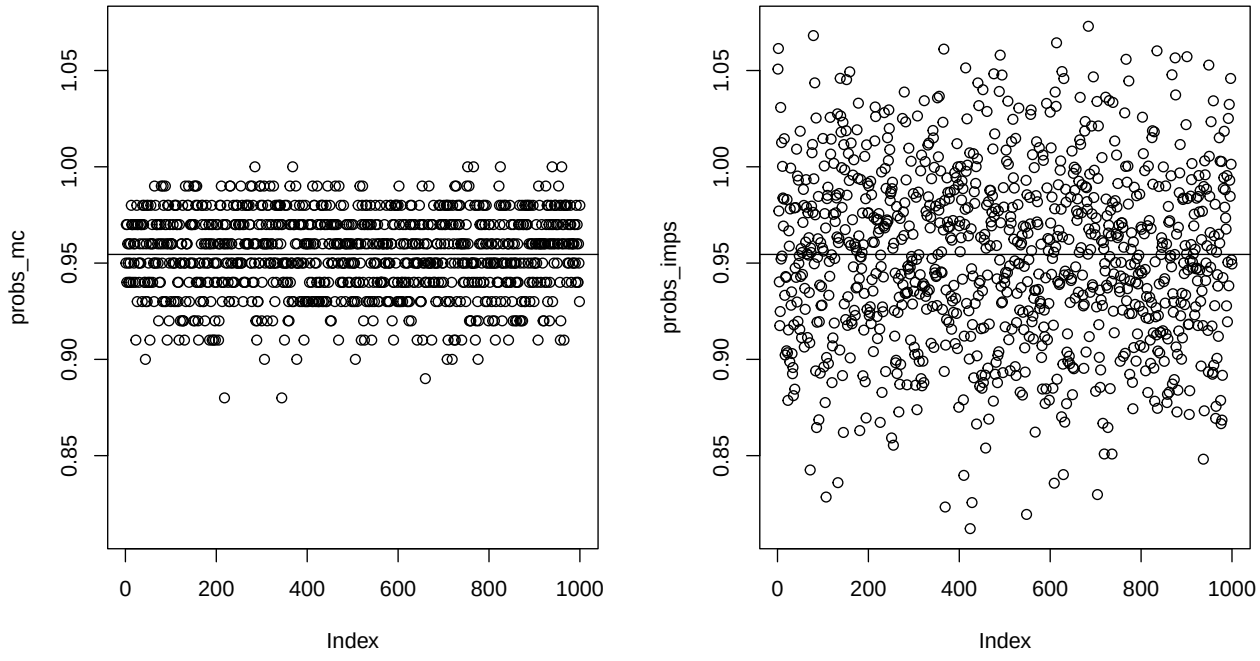
1 Estimate Normal Probability P(a < X < b) Using Importance Sampling

```
## a function computing the sum of numbers represented with logarithm
## lx     --- a vector of numbers, which are the log of another vector x.
## the log of sum of x is returned
log_sum_exp <- function(lx)
{   mlx <- max(lx)
    mlx + log(sum(exp(lx-mlx)))
}

## estimating the probability P(X in A) for X ~ N(0,1)
## by sampling from N(0,1)
est_normprob_mc <- function(A,iters_mc)
{
     X <- rnorm(iters_mc)
     mean((X > A[1]) * (X<A[2]))
}

## estimating the probability P(X in A) for X ~ N(0,1)
## by sampling from Unif(A[1],A[2])
est_normprob_imps <- function(A, iters_mc)
```

```r
{
     X <- runif(iters_mc,A[1],A[2])
     mean(dnorm(X))*(A[2]-A[1])
}

## estimating the probability P(X in A) for X ~ N(0,1)
## by sampling from Unif(A[1],A[2])
est_log_normprob_imps <- function(A, iters_mc)
{
     X <- runif(iters_mc,A[1],A[2])
     log_sum_exp(dnorm(X,log = TRUE))-log(iters_mc)+ log(A[2]-A[1])
}
```

1.1 Test 1

```r
A <- c(-2,2)
tp <- pnorm (A[2]) - pnorm (A[1])

probs_mc <- replicate(1000,est_normprob_mc(A,100))
probs_imps <- replicate(1000,est_normprob_imps(A,100))
par(mfrow = c(1,2))
ylim <- range (probs_mc, probs_imps)
plot(probs_mc, ylim = ylim); abline (h=tp)
plot(probs_imps, ylim = ylim); abline (h=tp)
```




```r
mean((probs_mc-tp)^2)
```

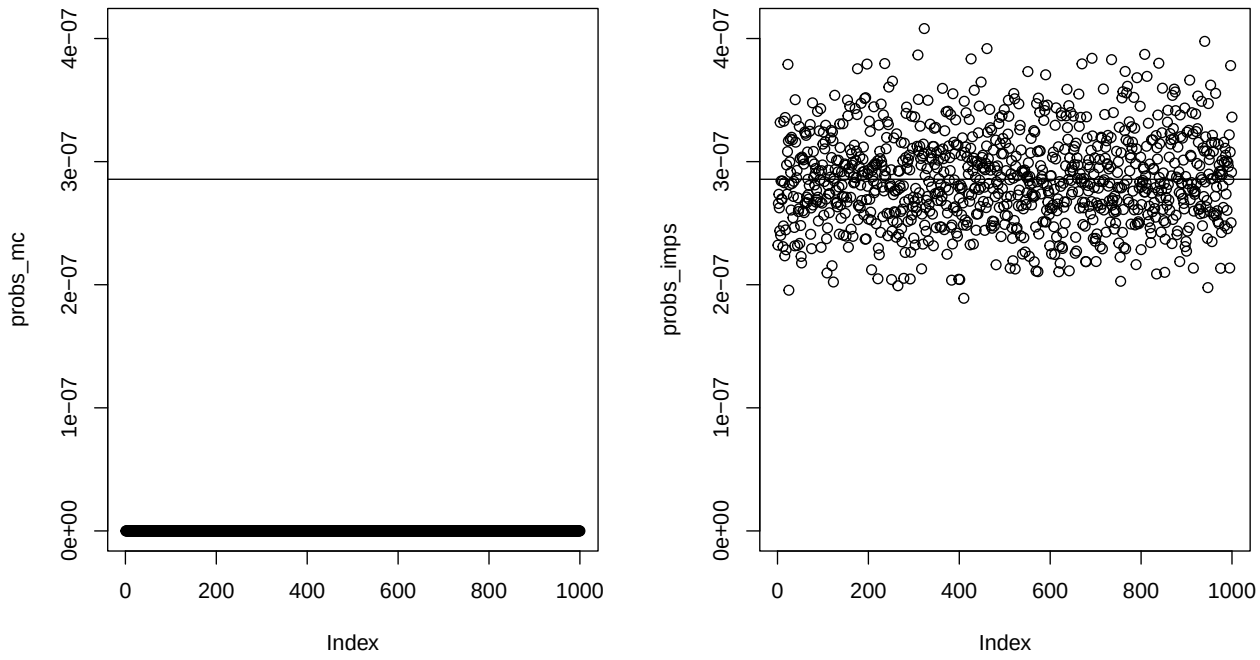
```
## [1] 0.00043699
```

```r
mean((probs_imps-tp)^2)
```

```
## [1] 0.002079687
```

1.2 Test 2

```
A <- c(5,6)
tp <- pnorm (A[2]) - pnorm (A[1])

probs_mc <- replicate(1000,est_normprob_mc(A,100))
probs_imps <- replicate(1000,est_normprob_imps(A,100))
par(mfrow = c(1,2))
ylim <- range (probs_mc, probs_imps)
plot(probs_mc, ylim = ylim); abline (h=tp)
plot(probs_imps, ylim = ylim); abline (h=tp)
```




```
mean((probs_mc-tp)^2)
```

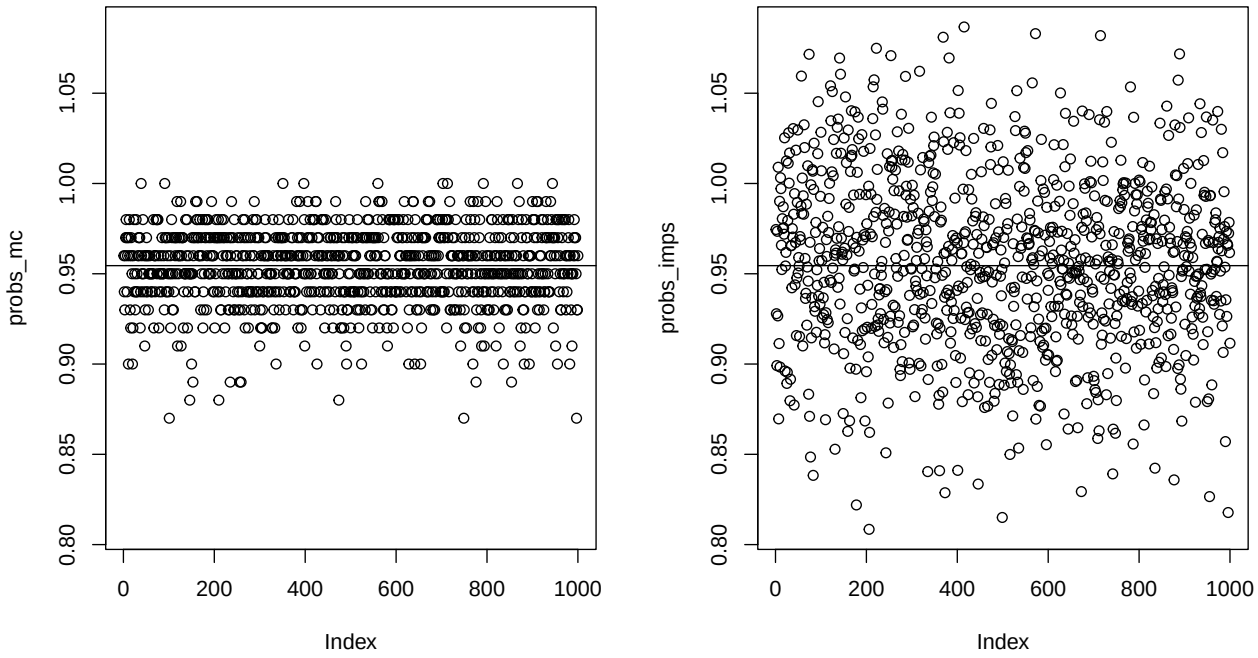
```
## [1] 8.160448e-14
```

```
mean((probs_imps-tp)^2)
```

```
## [1] 1.378715e-15
```

1.3 Test 3

```
A <- c(-2,2)
tp <- pnorm (A[2]) - pnorm (A[1])

probs_mc <- replicate(1000,est_normprob_mc(A,100))
probs_imps <- replicate(1000,est_normprob_imps(A,100))
par(mfrow = c(1,2))
ylim <- range (probs_mc, probs_imps)
plot(probs_mc, ylim = ylim); abline (h=tp)
plot(probs_imps, ylim = ylim); abline (h=tp)
```




```
mean((probs_mc-tp)^2)
```

```
## [1] 0.0004562895
```

```
mean((probs_imps-tp)^2)
```

```
## [1] 0.002244745
```

1.4 Test 4: An example of computing a underflow probability

```
A <- c(50,100)
#A method that cannot compute so small probability
tp1 <- pnorm (A[2]) - pnorm (A[1]); log(tp1)
```

```
## [1] -Inf
```

```
#The reason is that pnorm(A[1]) is so close to 1, or pnorm(-A[1]) underflow
pnorm (A[1])
```

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

```
log(pnorm(A[1]))
```

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

```
#Instead, we can use this way to compute the log probability
log_minus_exp <- function (la, lb) la + log (1-exp(lb-la))
la <- pnorm(-A[1], log=TRUE); la
```

```
## [1] -1254.831
```

```
lb <- pnorm(-A[2], log=TRUE); lb
```

```
## [1] -5005.524
```

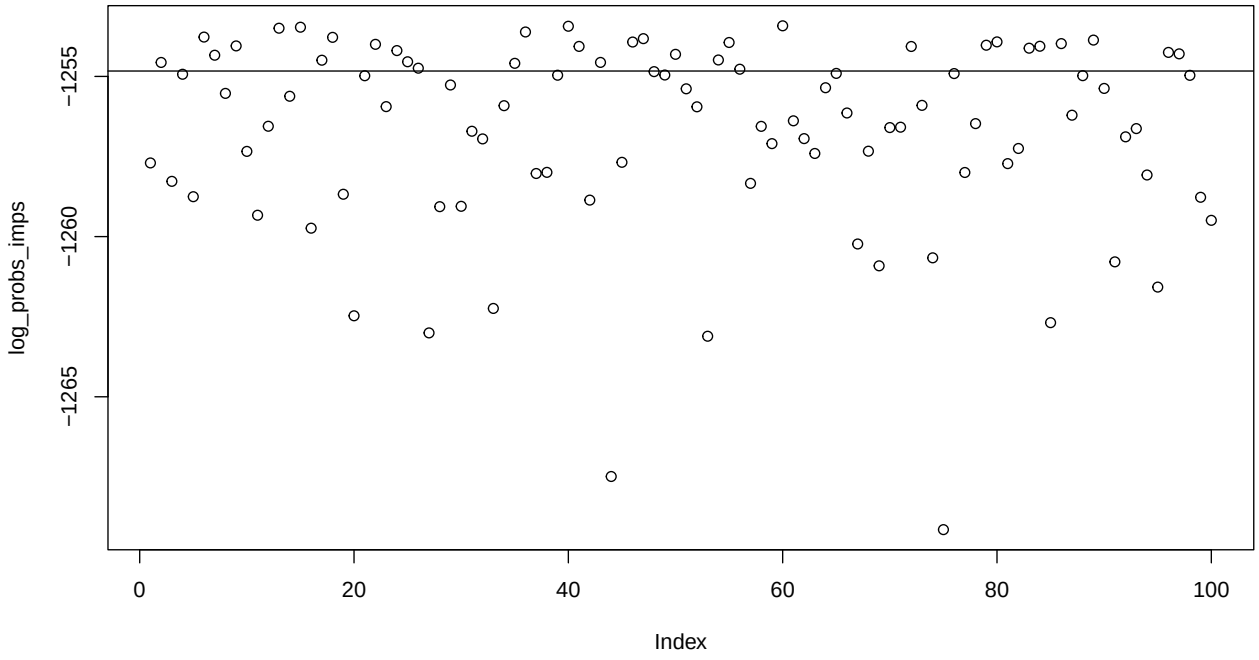
```
log_tp2 <- log_minus_exp(la, lb); log_tp2
```

```
## [1] -1254.831
```

```
#We still use the same importance sampling procedure, but taking care of the underflow problem in dnorm
log_probs_imps <- replicate(100,est_log_normprob_imps(A,1000)); log_probs_imps
```

```
##   [1] -1257.701 -1254.564 -1258.275 -1254.933 -1258.755 -1253.772 -1254.338
##   [8] -1255.530 -1254.048 -1257.340 -1259.333 -1256.555 -1253.495 -1255.619
##  [15] -1253.464 -1259.737 -1254.494 -1253.780 -1258.678 -1262.472 -1254.979
##  [22] -1253.998 -1255.943 -1254.194 -1254.543 -1254.743 -1263.006 -1259.066
##  [29] -1255.268 -1259.054 -1256.712 -1256.951 -1262.242 -1255.913 -1254.591
##  [36] -1253.613 -1258.030 -1257.995 -1254.963 -1253.434 -1254.063 -1258.861
##  [43] -1254.563 -1267.488 -1257.681 -1253.927 -1253.821 -1254.851 -1254.956
##  [50] -1254.310 -1255.387 -1255.949 -1263.110 -1254.487 -1253.941 -1254.775
##  [57] -1258.338 -1256.559 -1257.098 -1253.420 -1256.386 -1256.942 -1257.405
##  [64] -1255.353 -1254.906 -1256.141 -1260.230 -1257.334 -1260.913 -1256.597
##  [71] -1256.585 -1254.066 -1255.903 -1260.663 -1269.147 -1254.911 -1257.999
##  [78] -1256.473 -1254.026 -1253.922 -1257.722 -1257.249 -1254.119 -1254.058
##  [85] -1262.687 -1253.976 -1256.211 -1254.977 -1253.866 -1255.375 -1260.788
##  [92] -1256.886 -1256.629 -1258.077 -1261.576 -1254.250 -1254.296 -1254.967
##  [99] -1258.773 -1259.492
```

```
plot (log_probs_imps); abline (h = log_tp2)
```



2 Estimate E(a(X)) of Truncated Normal Using Importance Sampling

2.1 Estimating Function

```
## compute E(a) with importance sampling
est_tnorm_imps <- function(a, A, iters_mc)
{
    X <- runif(iters_mc,A[1],A[2])
    W <- dnorm (X)
    ahat <- sum (a(X) * W) / sum (W)
```

```
    attr(ahat, "effective sample size") <- 1/sum((W/sum(W))^2)
    ahat
}


## a generic function for approximating 1-D integral with midpoint rule
## the logarithms of the function values are passed in
## the log of the integral result is returned
## log_f  --- a function computing the logarithm of the integrant function
## range  --- the range of integral varaible, a vector of two elements
## n      --- the number of points at which the integrant is evaluated
## ...    --- other parameters needed by log_f
log_int_mid <- function(log_f, range, n,...)
{   if(range[1] >= range[2])
    stop("Wrong ranges")
    h <- (range[2]-range[1]) / n
    v_log_f <- sapply(range[1] + (1:n - 0.5) * h, log_f,...)
    log_sum_exp(v_log_f) + log(h)
}

## compute E(a) with midpoint rule
est_tnorm_mid <- function (a, A, iters_mc)
{
    log_f <- function (x) dnorm (x, log = T) + log (a(x))
    exp(log_int_mid (log_f, A, iters_mc) ) / (pnorm (A[2]) - pnorm (A[1]))
}

library (ars)

## a direct rejection sampling for truncated normal
sample_tnorm_drs <- function (n, lb = -Inf, ub = Inf)
{
    x <- rep (0, n)
    for (i in 1:n)
    {
        rej <- TRUE
        while (rej)
        {
            x[i] <- rnorm (1)
            if (x[i] >= lb & x[i] <= ub) rej <- FALSE
        }
    }
    x
}

## sample from truncated normal using ars package
sample_tnorm_ars <- function (n, lb, ub)
{
    logf <- function (x) dnorm (x, log = TRUE) ## define log density
    fprima <- function (x) -x ## define derivative of log density

    ars (10000, f = logf, fprima = fprima,
         x = c(lb, (lb + ub )/2, ub), # starting points
```

```
        lb = TRUE, ub = TRUE, xlb = lb, xub = ub) # boundary of log density
}
```

2.2 Test 1

```
## define the function a
a <- function (x) x^2
interval <- c(1,2)
A <- est_tnorm_mid (a, interval, 100000) ## midpoint rule

## estimate E(a) with rejection sampling
system.time(
    {
        rn_tnorm_ars <- sample_tnorm_ars (1000, interval[1],interval[2]) # draw samples from tnorm
        mean (a (rn_tnorm_ars))
    }
)
```

```
##    user  system elapsed
##   0.091   0.022   0.115
```

```
system.time(
est_tnorm_imps (a, interval, 1000000) ## importance sampling
)
```

```
##    user  system elapsed
##   0.062   0.008   0.071
```

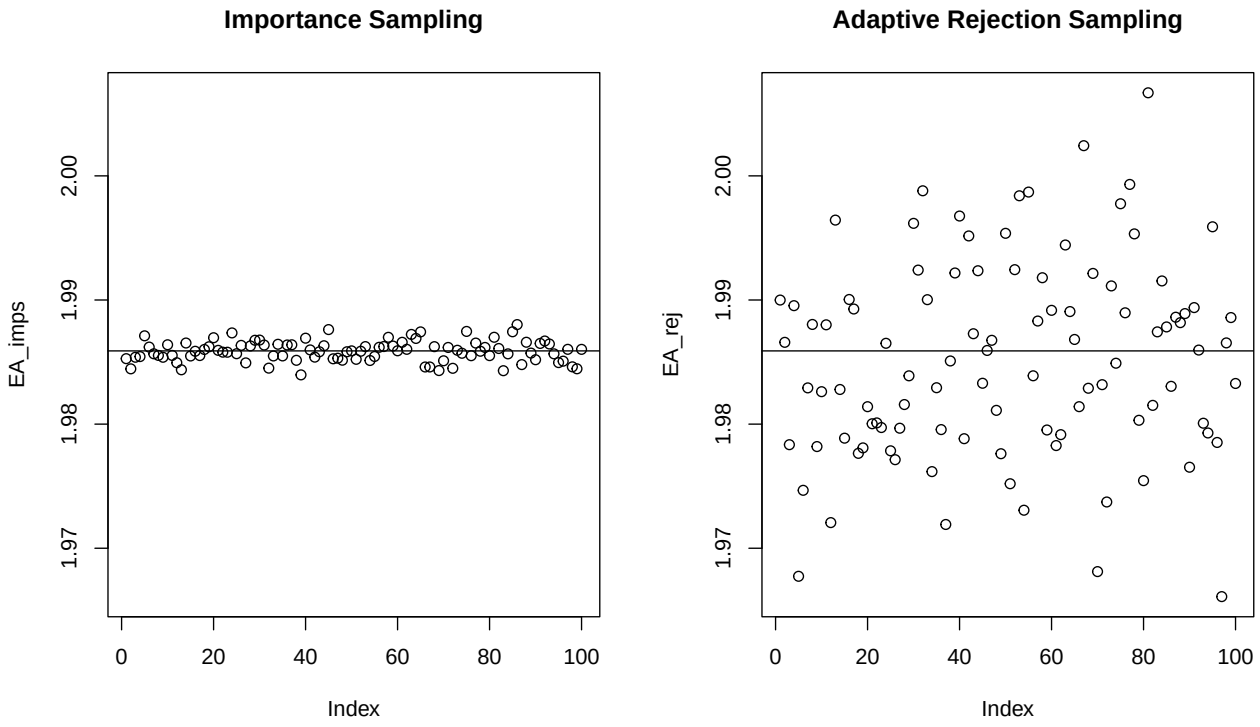
```
## simulation comparison of importance sampling and rejection sampling
times.imps <- system.time(
EA_imps <- replicate (100, est_tnorm_imps (a, interval, 1000000))
)

times.imps
```

```
##    user  system elapsed
##   5.915   0.179   6.199
```

```
times.rej <- system.time (
EA_rej <- replicate (100,
    {   rn_tnorm_ars <- sample_tnorm_ars (1000, interval[1],interval[2])
        mean (a (rn_tnorm_ars))
    }
    )
)
times.rej
```

```
##    user  system elapsed
##   7.958   0.697   8.712
```

```
par (mfrow = c(1,2))
ylim <- range (EA_imps, EA_rej)
plot (EA_imps, ylim = ylim, main = "Importance Sampling")
abline (h = A)
plot (EA_rej, ylim = ylim, main = "Adaptive Rejection Sampling")
abline (h = A)
```




```
mean ((EA_rej-A)^2)
```

```
## [1] 6.261365e-05
```

```
mean ((EA_imps-A)^2)
```

```
## [1] 6.894636e-07
```

2.3 Test 2

```
## define the function a
a <- function (x) x^2
interval <- c(-1,1)
A <- est_tnorm_mid (a, interval, 100000) ## midpoint rule

## estimate E(a) with rejection sampling
system.time(
    {
        rn_tnorm_ars <- sample_tnorm_ars (1000, interval[1],interval[2]) # draw samples from tnorm
        mean (a (rn_tnorm_ars))
    }
)
```

```
##    user  system elapsed
##   0.076   0.006   0.083
```

```
system.time(
est_tnorm_imps (a, interval, 1000000) ## importance sampling
)
```

```
##    user  system elapsed
##   0.059   0.007   0.067
```

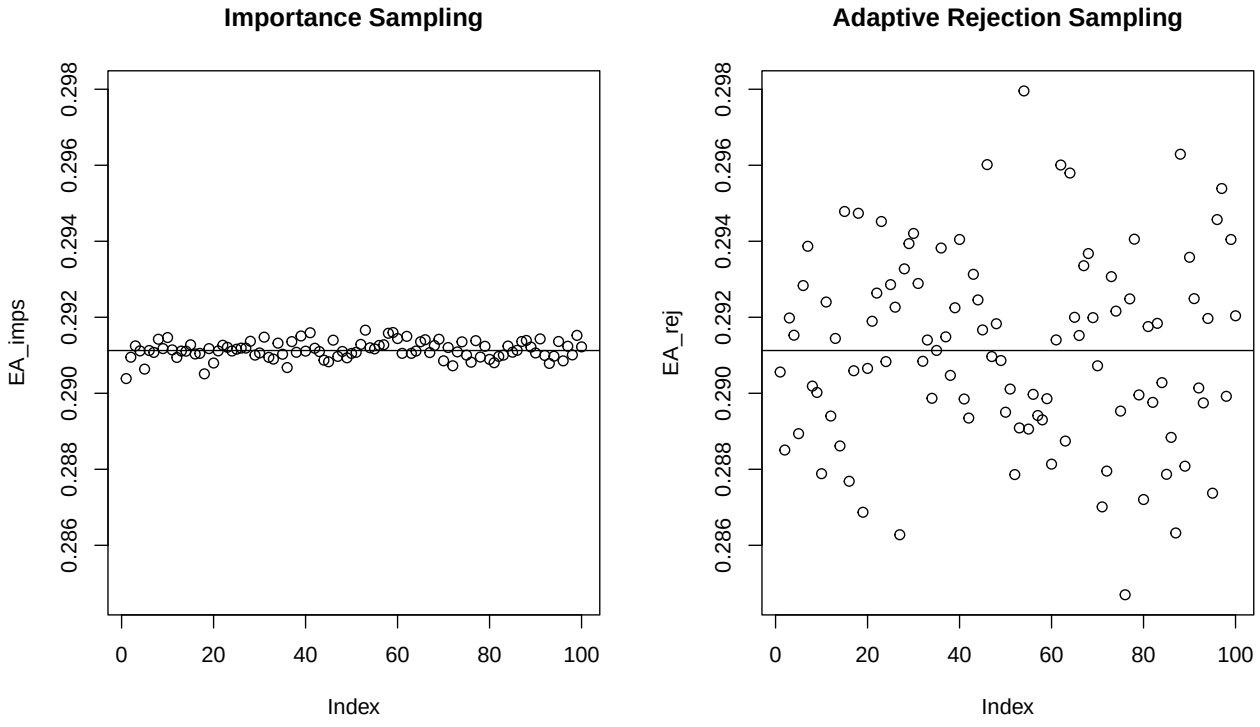
```
## simulation comparison of importance sampling and rejection sampling
times.imps <- system.time(
EA_imps <- replicate (100, est_tnorm_imps (a, interval, 1000000))
)

times.imps
```

```
##    user  system elapsed
##   5.849   0.175   6.146
```

```
times.rej <- system.time (
EA_rej <- replicate (100,
    {   rn_tnorm_ars <- sample_tnorm_ars (1000, interval[1],interval[2])
        mean (a (rn_tnorm_ars))
    }
    )
)
times.rej
```

```
##    user  system elapsed
##   8.155   0.619   8.845
```

```
par (mfrow = c(1,2))
ylim <- range (EA_imps, EA_rej)
plot (EA_imps, ylim = ylim, main = "Importance Sampling")
abline (h = A)
plot (EA_rej, ylim = ylim, main = "Adaptive Rejection Sampling")
abline (h = A)
```




```
mean ((EA_rej-A)^2)
```

```
## [1] 6.258295e-06
```

```r
mean ((EA_imps-A)^2)
```

```
## [1] 5.611324e-08
```

3 Computing Log Marginalized Likelihood for Normal Models with Importance Sampling

3.1 Functions

```r
## a function computing the sum of numbers represented with logarithm
## lx     --- a vector of numbers, which are the log of another vector x.
## the log of sum of x is returned
log_sum_exp <- function(lx)
{   mlx <- max(lx)
    mlx + log(sum(exp(lx-mlx)))
}
## computing the log probability density function of multivariate normal
## x     --- a vector, the p.d.f at x will be computed
## mu   --- the mean vector of multivariate normal distribution
## A     --- the inverse covariance matrix of multivariate normal distribution
log_pdf_mnormal <- function(x, mu, A)
{   0.5 * ( -length(mu)*log(2*pi) + sum(log(svd(A)$d)) - t(x-mu) %*% A %*% (x-mu) )

}
## the function for computing log likelihood of normal data
log_lik <- function(x,mu,w)
{   sum(dnorm(x,mu,exp(w),log=TRUE))
}

## the function for computing log prior
log_prior <- function(mu,w, mu_0,sigma_mu,w_0,sigma_w)
{   dnorm(mu,mu_0,sigma_mu,log=TRUE) + dnorm(w,w_0,sigma_w,log=TRUE)
}

## the function for computing the negative log of likelihood * prior
neg_log_post <- function(x, theta, mu_0,sigma_mu,w_0,sigma_w)
{   - log_lik(x,theta[1], theta[2]) -
    log_prior(theta[1],theta[2],mu_0,sigma_mu,w_0,sigma_w)
}


## computing the log marginal likelihood using importance sampling with
## the posterior distribution approximated by the Gaussian distribution at
## its mode
log_mar_gaussian_imps <- function(x,mu_0,sigma_mu,w_0,sigma_w,iters_mc)
{   result_min <- nlm(f=neg_log_post,p=c(mean(x),log(sqrt(var(x)))),
                      hessian=TRUE,
                      x=x,mu_0=mu_0,sigma_mu=sigma_mu,w_0=w_0,sigma_w=sigma_w)
    hessian <- result_min$hessian
    mu <- result_min$estimate

    ## finding the multiplier for sampling from multivariate normal
```

```
    Sigma <- t( chol(solve(hessian)) )
    ## draw samples from N(mu, Sigma %*% Sigma')
    thetas <- Sigma %*% matrix(rnorm(2*iters_mc),2,iters_mc) + mu

    ## values of log approximate p.d.f. at samples
    log_pdf_mnormal_thetas <- apply(thetas,2,log_pdf_mnormal,mu=mu,A=hessian)
    ## values of log true p.d.f. at samples
    log_post_thetas <- - apply(thetas,2,neg_log_post,x=x, mu_0=mu_0,
                               sigma_mu=sigma_mu,w_0=w_0,sigma_w=sigma_w)

    ## averaging the weights, returning its log
    log_sum_exp(log_post_thetas-log_pdf_mnormal_thetas) - log(iters_mc)
}


## we use Monte Carlo method to debug the above function
log_mar_gaussian_mc <- function(x,mu_0,sigma_mu,w_0,sigma_w,iters_mc)
{
    ## draw samples from the priors
    mus <- rnorm(iters_mc,mu_0,sigma_mu)
    ws <- rnorm(iters_mc,w_0,sigma_w)
    one_log_lik <- function(i)
    {  log_lik(x,mus[i],ws[i])
    }
    v_log_lik <- sapply(1:iters_mc,one_log_lik)
    log_sum_exp(v_log_lik) - log(iters_mc)
}
## the generic function for finding laplace approximation of integral of 'f'
## neg_log_f    --- the negative log of the intergrand function
## p0           --- initial value in searching mode
## ...          --- other arguments needed by neg_log_f
bayes_inference_lap <- function(neg_log_f,p0,...)
{   ## looking for the mode and hessian of the log likehood function
    result_min <- nlm(f=neg_log_f,p=p0, hessian=TRUE,...)
    hessian <- result_min$hessian
    neg_log_like_mode <- result_min$minimum


    estimates <- result_min$estimate ## posterior mode
    SIGMA <- solve(result_min$hessian) ## covariance matrix of posterior mode
    sds <- sqrt (diag(SIGMA)) ## standard errors of each estimate
    log_mar_lik <- ## log marginalized likelihood
        - neg_log_like_mode + 0.5 * ( sum(log(2*pi) - log(svd(hessian)$d) ))

    list (estimates = estimates, sds = sds, SIGMA = SIGMA, log_mar_lik = log_mar_lik)
}


## approximating the log of integral of likelihood * prior
bayes_inference_lap_gaussian <- function(x,mu_0,sigma_mu,w_0,sigma_w)
{   bayes_inference_lap(
                neg_log_post,p0=c(mean(x),log(sqrt(var(x)))),
                x=x,mu_0=mu_0,sigma_mu=sigma_mu,w_0=w_0,sigma_w=sigma_w
```

```
    )
}
```

3.2 Testing

```
## debugging the program
x <- rnorm(50)
log_mar_gaussian_imps(x,0,1,0,5,100)
```

```
## [1] -73.35034
```

```
log_mar_gaussian_mc(x,0,1,0,5,10000)
```

```
## [1] -73.5719
```

```
bayes_inference_lap_gaussian(x,0,1,0,5)
```

```
## $estimates
## [1]  0.02905778 -0.06963118
##
## $sds
## [1] 0.13077623 0.09999276
##
## $SIGMA
##                [,1]          [,2]
## [1,]  1.710242e-02 -8.963959e-06
## [2,] -8.963959e-06  9.998553e-03
##
## $log_mar_lik
## [1] -73.41088
```

```
x <- rnorm(10) # another debug
log_mar_gaussian_imps(x,0,1,0,5,100)
```

```
## [1] -18.96407
```

```
log_mar_gaussian_mc(x,0,1,0,5,10000)
```

```
## [1] -19.01825
```

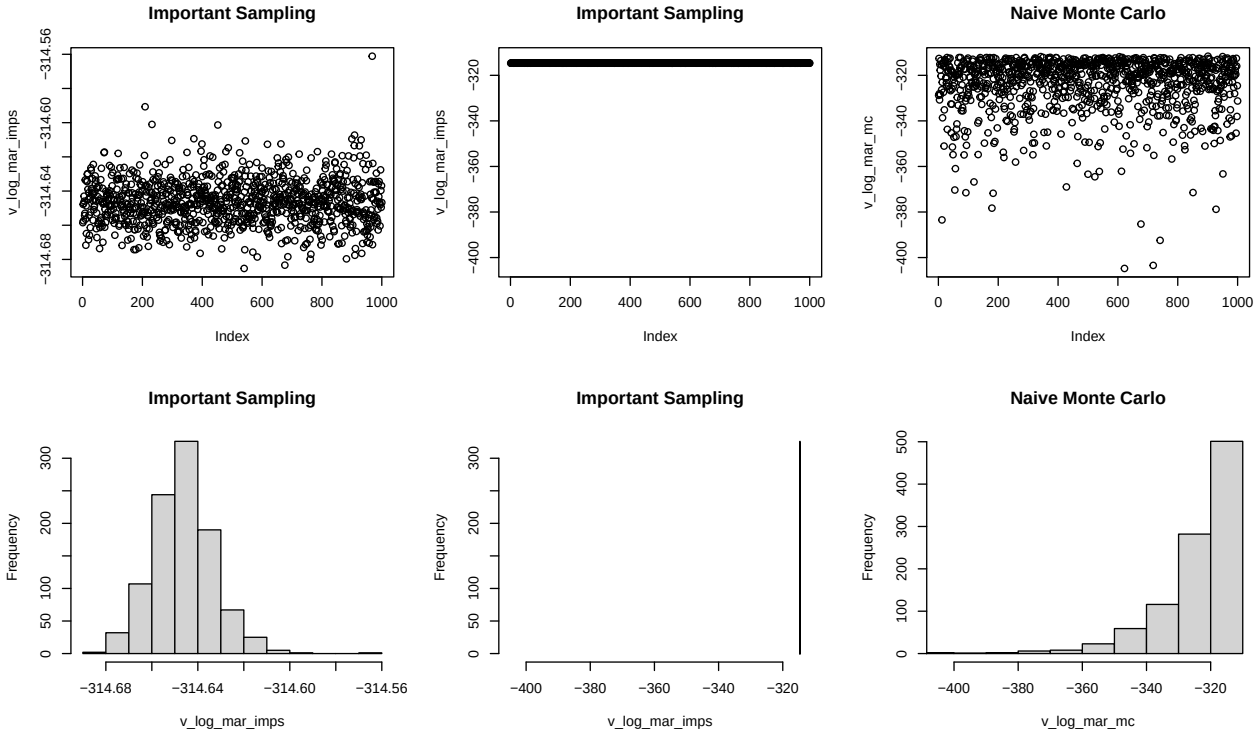
```
bayes_inference_lap_gaussian(x,0,1,0,5)
```

```
## $estimates
## [1] -0.07190741  0.05481965
##
## $sds
## [1] 0.3168537 0.2233930
##
## $SIGMA
##               [,1]         [,2]
## [1,] 0.1003962818 0.0007248575
## [2,] 0.0007248575 0.0499044160
##
## $log_mar_lik
## [1] -18.99896
```

```r
## comparing importance sampling with Gaussian approximation with naive monte carlo
x <- rnorm(200)
bayes_inference_lap_gaussian(x,0,1,0,5)
```

```
## $estimates
## [1] 0.007028495 0.118606615
##
## $sds
## [1] 0.07936393 0.05000188
##
## $SIGMA
##              [,1]         [,2]
## [1,] 6.298633e-03 2.461681e-08
## [2,] 2.461681e-08 2.500188e-03
##
## $log_mar_lik
## [1] -314.6507
```

```r
v_log_mar_imps <- replicate(1000, log_mar_gaussian_imps(x,0,1,0,5,100))
v_log_mar_mc <- replicate(1000, log_mar_gaussian_mc(x,0,1,0,5,100))


par(mfcol=c(2,3))
xlim <- c(min(c(v_log_mar_imps,v_log_mar_mc)),max(c(v_log_mar_imps,v_log_mar_mc)))

plot(v_log_mar_imps, main="Important Sampling")
hist(v_log_mar_imps,main="Important Sampling")

plot(v_log_mar_imps, ylim = xlim, main="Important Sampling")
hist(v_log_mar_imps,main="Important Sampling",xlim=xlim)

plot(v_log_mar_mc,main="Naive Monte Carlo",ylim=xlim)
hist(v_log_mar_mc,main="Naive Monte Carlo",xlim=xlim)
```




```
## comparing variance of importance sampling and naive sampling
var (v_log_mar_imps)
```

```
## [1] 0.0001784331
```

```
var (v_log_mar_mc)
```

```
## [1] 152.9014
```

Comparing Monte Carlo variance of Importance sampling and Laplace Approximation

```
mean (v_log_mar_imps)
```

```
## [1] -314.6463
```

```
bayes_inference_lap_gaussian(x,0,1,0,5)
```

```
## $estimates
## [1] 0.007028495 0.118606615
##
## $sds
## [1] 0.07936393 0.05000188
##
## $SIGMA
##               [,1]         [,2]
## [1,] 6.298633e-03 2.461681e-08
## [2,] 2.461681e-08 2.500188e-03
##
## $log_mar_lik
## [1] -314.6507
```

We see that Laplace approximation is really good for this problem, but it may not be good for other problems in which the posterior cannot be approximated by Gaussian.