Monte Carlo Methods 2

Hun Lee

(1-1) Monte Carlo integration using Importance sampling

$$\int_0^1 g(x)dx = \int_0^1 \frac{g(x)}{p(x)}p(x)dx$$

Using pdf of U(0,1) = 1,

$\int_0^1 e^{x^2}dx = \int_0^1 \frac{e^{x^2}}{1}1dx \approx 1.463$ and we approximate the integral using $E[g(x)] \approx \frac{1}{n}\sum_{i=1}^{n} g(x)$ (by LLN)

```
set.seed(1)
LLN_integ <- function(n){
   u <- runif(n)
   y = sum(exp(u^2))/n #by LLN
   return(cat("Estimated integral value:", y))
}

LLN_integ(1000)
```

```
## Estimated integral value: 1.463324
```

(1-2) Monte carlo integration using the method of control variate

We approximate the integral with $\int_0^1 g(x) = \int_0^1 m(x) + g(x) - m(x)dx = \int_0^1 m(x)dx + \int_0^1 g(x) - m(x)dx$

where $g(x) = e^{x^2}$, $m(x) = x^2 + 1$, And we approximate the integral using $\frac{1}{n}\sum_{i=1}^{n}\big(g(x) - m(x)\big)$

```
set.seed(1)
ctrl_variate_integ <- function(n){
   gfun <- function(x){exp(x^2)}
   mfun <- function(x){x^2 + 1}
   u <- runif(n)
   y <- sum(mfun(u))/n + sum(gfun(u) - mfun(u))/n
   return(cat("Estimated integral value:", y))
}

ctrl_variate_integ(1000)
```

```
## Estimated integral value: 1.463324
```

(2) Monte Carlo estimation using two control variates

In estimating $E[\sqrt{1-U^2}] = \frac{\pi}{4} \approx 0.785$, let's confirm U^2 is a better control variate than U, where $U \sim U(0,1)$.

$$E\left[\sqrt{1-U^2}\right] = E\left[\sqrt{1-U^2} + c^*(m(x) - \mu_{m(x)})\right]$$

$$\text{where } c^* = -\frac{Cov(\sqrt{1-U^2}, m(x))}{var(\sqrt{1-U^2})} \text{ and m(x) is control variate, } U^2 \text{ or } U$$

$$\text{Hence, we can approximate the integral } \int_0^1 \sqrt{1-U^2}dx \approx \frac{1}{n}\sum_{i=1}^{n}\left[\sqrt{1-U^2} + c^*(m(x) - \mu_{m(x)})\right]$$

$$\text{And, } Var\left[\sqrt{1-U^2} + c^*(m(x) - \mu_{m(x)})\right] = Var(\sqrt{1-U^2}) + (c^*)^2 Var(m(x)) + 2c^* Cov\left(\sqrt{1-U^2}, m(x)\right)$$

Using U^2 vs U as control variate

```
ctrl_vari_integ_u_squared <- function(n){
   set.seed(4); u <- runif(n)
   gfunc <- function(x){sqrt(1 - u^2)}; mfunc_1 <- function(x){u^2};
   covariance <- cov(gfunc(u), mfunc_1(u))
   c <- -cov(gfunc(u), mfunc_1(u))/var(mfunc_1(u)); mu_1 <- 1/3
   estimate1 <- mean(gfunc(u) + c*(mfunc_1(u) - mu_1)); variance <- var(gfunc(u) + c*(mfunc_1(u) - mu_1
   return(tibble(estimate1, variance, covariance))
}

ctrl_vari_integ_u <- function(n){
   set.seed(4); u <- runif(n)
   gfunc <- function(x){sqrt(1 - u^2)}; mfunc_2 <- function(x){u}
   covariance <- cov(gfunc(u), mfunc_2(u))
   c <- -cov(gfunc(u), mfunc_2(u))/var(mfunc_2(u)); mu_2 <- 1/2
   estimate2 <- mean(gfunc(u) + c*(mfunc_2(u) - mu_2)); variance <- var(gfunc(u) + c*(mfunc_2(u) - mu_2
   return(tibble(estimate2, variance, covariance))
}

ctrl_vari_integ_u_squared(1000)
```

```
## # A tibble: 1 x 3
##   estimate1 variance covariance
##       <dbl>    <dbl>      <dbl>
## 1     0.785   0.0014    -0.0638
```

```
ctrl_vari_integ_u(1000)
```

```
## # A tibble: 1 x 3
##   estimate2 variance covariance
##       <dbl>    <dbl>      <dbl>
## 1     0.785  0.00695    -0.0586
```

When using U^2 and U as control variate, the estimate values are almost the same. However, U^2 is a better choice because its covariance with given function g(x) is smaller than that of U with g(x) and hence gives lower variance of the estimate.

(3) Monte Carlo estimation using Importance sampling

Our goal is to obtain a Monte Carlo estimate of

$$\int_1^\infty \frac{x^2}{\sqrt{2\pi}} e^{\frac{-x^2}{2}} dx \approx 0.4$$

We use

$$e^{-\frac{x^2}{2}} dx = \int_1^\infty g(x)dx = \int_1^\infty \frac{g(x)}{p(x)} p(x)dx$$

where

$$g(x) = \frac{x^2}{\sqrt{2\pi}} e^{\frac{-x^2}{2}} I\{x \geq 1\} \quad \text{and} \quad p(x) = \frac{1}{\sqrt{4\pi}} e^{-\frac{(x-3)^2}{4}} \quad \text{which is pdf of N(3,2)}$$

We estimate integral with

$$\frac{1}{n} \sum_{i=1}^{n} \frac{g(X_i)}{p(X_i)}$$

Importance sampling and evaluate its variance

```
set.seed(16)
MC_importance <- function(n){
   u <- rnorm(n)
   gfunc <- function(x){x^2/sqrt(2*pi)*exp(-x^2/2) * (u >= 1) + 0 * (u < 1)}
   # Given nominal distribution function g(x)

   pfunc <- function(x){1/(sqrt(2*pi)*sqrt(4))*exp(-(x - 3)^2/(4))}
   # Importance distribution function p(x): N(3,2) pdf

   estimate <- sum(gfunc(u)/pfunc(u))/n

   variance <- var((gfunc(u)/pfunc(u)))/n

   return(tibble(estimate, variance))
 }

MC_importance(1000)
```

```
## # A tibble: 1 x 2
##   estimate variance
##      <dbl>    <dbl>
## 1    0.404 0.000967
```

The estimate value, 0.404, is close to its true value, 0.4. The variance of the estimate is about 0.000967 The standard deviation of the estimate is 0.0311 which is reasonable considering the estimate value is 0.404 and its true value is 0.4.

(4) Bisection algorithm to find the local minimum value of f(x)

$f(x) = -e^{-x}sin(x)$ and $f^{'}(x) = e^{-x}(\sin(x) - \cos(x))$ and $f^{'}(\frac{\pi}{4}) = 0$

Answer we are looking for: $\frac{\pi}{4} \approx 0.7853982$

```
bisection_optimization <- function(a, b, tol = 1e-10){
   f <- function(x){-exp(-x)*sin(x)}; f_prime <- function(x){exp(-x)*(sin(x) - cos(x))}
   f_prime(0) #starting value
   i <- 0; cur <- (a + b)/2; res <- c(i, cur, f_prime(cur))
   while (abs(f_prime(cur)) > tol) {
      i <- i + 1
      if (f_prime(a) * f_prime(cur) > 0)
         a <- cur
      else
         b <- cur; cur <- (a + b)/2; res <- rbind(res, c(i, cur, f_prime(cur)))
      colnames(res) <- c("Number of trial", "Approximated estimate", "Score function value")
   }
   return(res)}
bisection_optimization(0, 1.5) %>% kable() %>%
   kable_styling(font_size = 6, latex_options = "HOLD_position")
```

	Number of trial	Approximated estimate	Score function value
res	0	0.7500000	-0.0236420
	1	1.1250000	0.1529409
	2	0.9375000	0.0839117
	3	0.8437500	0.0354721
	4	0.7968750	0.0073156
	5	0.7734375	-0.0078048
	6	0.7851562	-0.0001560
	7	0.7910156	0.0036018
	8	0.7880859	0.0017284
	9	0.7866211	0.0007876
	10	0.7858887	0.0003161
	11	0.7855225	0.0000801
	12	0.7853394	-0.0000379
	13	0.7854309	0.0000211
	14	0.7853851	-0.0000084
	15	0.7854080	0.0000064
	16	0.7853966	-0.0000010
	17	0.7854023	0.0000027
	18	0.7853994	0.0000008
	19	0.7853980	-0.0000001
	20	0.7853987	0.0000004
	21	0.7853984	0.0000001
	22	0.7853982	0.0000000
	23	0.7853981	0.0000000
	24	0.7853981	0.0000000
	25	0.7853982	0.0000000
	26	0.7853982	0.0000000
	27	0.7853982	0.0000000
	28	0.7853982	0.0000000
	29	0.7853982	0.0000000
	30	0.7853982	0.0000000
	31	0.7853982	0.0000000

With bisection algorithm on the closed interval [0, 1.5], we reach our goal of having estimate being close to 0.7853982 and score function being close to 0 around at 22 trials of bisection algorithm. This method is not the most efficient approach, but the result seems successful considering the method gives us the estimate very close to the true value within 31 trials.

(5) Modified Newton-Raphson algorithm with step-halving and re-direction steps

The Poisson distribution, written as

$$P(Y = y) = \frac{\lambda^y e^{-\lambda}}{y!}$$

for $\lambda > 0$, is often used to model "count'' data — e.g., the number of events in a given time period.

A Poisson regression model states that

$$Y_i \sim \text{Poisson}(\lambda_i),$$

where

$$\log \lambda_i = \alpha + \beta x_i$$

$$L(\alpha, \beta; y_i) = \prod_{i=1}^{n} = \frac{e^{-\lambda}\lambda^{y_i}}{y_i!} = \prod_{i=1}^{n} \frac{e^{-e^{\alpha+\beta x_i}}(e^{\alpha+\beta x_i})^{y_i}}{y_i!}$$

$$l(\alpha, \beta; y_i) = -\sum_{i=1}^{n} e^{\alpha+\beta x_i} + \sum_{i=1}^{n} y_i(\alpha + \beta x_i) - \sum_{i=1}^{n} \log(y_i!)$$

$$\frac{\partial l}{\partial \alpha} = -\sum_{i=1}^{n} e^{\alpha+\beta x_i} + \sum_{i=1}^{n} y_i$$

$$\frac{\partial l}{\partial \beta} = -\sum_{i=1}^{n} x_i e^{\alpha+\beta x_i} + \sum_{i=1}^{n} y_i x_i$$

$$\nabla l(\alpha, \beta) = \begin{pmatrix} \sum_{i=1}^{n} y_i - \sum_{i=1}^{n} e^{\alpha+\beta x_i} \\ \sum_{i=1}^{n} x_i(y_i - e^{\alpha+\beta x_i}) \end{pmatrix}$$

$$\nabla^2 l(\alpha, \beta) = \begin{pmatrix} -\sum_{i=1}^{n} e^{\alpha+\beta x_i} & -\sum_{i=1}^{n} x_i e^{\alpha+\beta x_i} \\ -\sum_{i=1}^{n} x_i e^{\alpha+\beta x_i} & -\sum_{i=1}^{n} x_i^2 e^{\alpha+\beta x_i} \end{pmatrix}$$

Generating random sample

```
  #sample size
n <- 500
  #regression coefficients
alpha <- 2
beta1 <- 1

set.seed(1)
x <- runif(n)

lambda <- exp(alpha + beta1 * x)

set.seed(1)
y <- rpois(n = n, lambda = lambda)
  #data set
data <- tibble(Y = y, X = x)
```

Function for log likelihood, gradient, and Hessian

```
logisticstuff <- function(dat, betavec) {
   u <- betavec[1] + betavec[2] * dat$x

   exp_u <- exp(u)

   loglik <- -sum(exp_u) + sum(dat$y * u) - sum(log(factorial(dat$y)))

   grad <- c(sum(dat$y - exp_u), sum(dat$x*(dat$y - exp_u)))

   Hess <- -matrix(c(sum(exp_u),
                  rep(sum(dat$x*exp_u),2),
                  sum((dat$x)^2 * exp_u)), ncol = 2)

   return(list(loglik = loglik, grad = grad, Hess = Hess))
}
```

Function for (basic) Newton Raphson algorithm

```
NewtonRaphson <- function(dat, func, start, tol=1e-10, maxiter = 200) {
   i <- 0
   cur <- start
   stuff <- func(dat, cur)
   asc_dir_check <- -t(stuff$grad) %*% solve(stuff$Hess) %*% stuff$grad
   res <- c(0, stuff$loglik, cur, asc_dir_check)
   prevloglik <- -Inf # To make sure it iterates

   while (i < maxiter && abs(stuff$loglik - prevloglik) > tol) {
      i <- i + 1
   prevloglik <- stuff$loglik
   prev <- cur
   cur <- prev - solve(stuff$Hess) %*% stuff$grad
   stuff <- func(dat, cur) # log likelihood, gradient, Hessian
   asc_dir_check <- -t(stuff$grad) %*% solve(stuff$Hess) %*% stuff$grad
   res <- rbind(res, c(i, stuff$loglik, cur, asc_dir_check))
   colnames(res) <- c("Number of trial", "Log likelihood", "Alpha", "Beta", "asc_dir_check")
   }
   return(res)
}

coef <- c(0,0) # Randomly assigned coefficients (starting point)

ans <- NewtonRaphson(list(x = x, y = y), logisticstuff, coef)
```

Table 1: Basic Newton Raphson result

	Number of trial	Log likelihood	Alpha	Beta	asc_dir_check
res	0	-1.157406e+04	0.0000000	0.000000	7.198287e+04
	1	-2.186238e+09	5.2221650	12.581001	2.186293e+09
	2	-8.042386e+08	4.2223234	12.580833	8.042865e+08
	3	-2.958332e+08	3.2227539	12.580376	2.958749e+08
	4	-1.088052e+08	2.2239240	12.579133	1.088407e+08
	5	-4.000540e+07	1.2271038	12.575757	4.003470e+07
	6	-1.469929e+07	0.2357413	12.566587	1.472241e+07
	7	-5.393583e+06	-0.7408244	12.541705	5.410632e+06
	8	-1.974029e+06	-1.6774539	12.474415	1.985259e+06
	9	-7.196776e+05	-2.5076133	12.294036	7.257500e+05
	10	-2.613716e+05	-3.0633352	11.821944	2.638348e+05
	11	-9.480762e+04	-2.9751662	10.664054	9.639291e+04
	12	-3.380200e+04	-1.7404639	8.278097	3.621234e+04
	13	-1.099843e+04	0.2295282	5.091003	1.234540e+04
	14	-3.385355e+03	1.3119337	2.917549	3.114104e+03
	15	-1.548582e+03	1.7889518	1.639605	3.798096e+02
	16	-1.341807e+03	1.9560050	1.099951	8.539803e+00
	17	-1.337469e+03	1.9802069	1.012171	5.109800e-03
	18	-1.337466e+03	1.9807749	1.010018	0.000000e+00
	19	-1.337466e+03	1.9807752	1.010016	0.000000e+00
	20	-1.337466e+03	1.9807752	1.010016	0.000000e+00

```
## True Alpha: 2, True Beta: 1
```

Function for Modified Newton Raphson algorithm

```
NewtonRaphson <- function(dat, func, start, tol=1e-10, maxiter = 200) {
   i <- 0
   cur <- start
   stuff <- func(dat, cur)
   asc_dir_check <- -t(stuff$grad) %*% solve(stuff$Hess) %*% stuff$grad
   res <- c(0, stuff$loglik, cur, asc_dir_check)
   prevloglik <- -Inf # To make sure it iterates
   lambda <- 1

   while (i < maxiter && abs(stuff$loglik - prevloglik) > tol) {
   i <- i + 1

   prev <- cur

   #checking if direction is ascent. If not, transform Hessian into negative definite.
   if (asc_dir_check < 0) {
      stuff$Hess = stuff$Hess - (max(stuff$Hess) + 50)
      cur <- prev - lambda * (solve(stuff$Hess) %*% stuff$grad)
      prev <- cur
      stuff <- func(dat, cur)
      prevloglik <- stuff$loglik
   }

   else {
      cur <- prev - lambda * (solve(stuff$Hess) %*% stuff$grad)
```

```
      prev <- cur
      stuff <- func(dat, cur)
      prevloglik <- stuff$loglik
   }

   cur2 <- prev - lambda * (solve(stuff$Hess) %*% stuff$grad)
   stuff2 <- func(dat, cur2) # log likelihood, gradient, Hessian
   asc_dir_check <- -t(stuff2$grad) %*% solve(stuff2$Hess) %*% stuff2$grad

   #condition check before step halving process
   if (stuff2$loglik > prevloglik) {
      cur = cur2
      stuff = stuff2
      asc_dir_check <- -t(stuff$grad) %*% solve(stuff$Hess) %*% stuff$grad
   }

   #step halving process
   else {
      repeat {
      lambda = lambda/2
      cur = prev - lambda * (solve(stuff$Hess) %*% stuff$grad)
      stuff = func(dat, cur)
      if (stuff$loglik > prevloglik) {
      cur = cur
      stuff = stuff
      asc_dir_check <- -t(stuff$grad) %*% solve(stuff$Hess) %*% stuff$grad
      }
      break}
      }

   res <- rbind(res, c(i, stuff$loglik, cur, asc_dir_check))
   colnames(res) <- c("Number of trial", "Log likelihood", "Alpha", "Beta", "asc_dir_check")
   }
   return(res)
}

ans <- NewtonRaphson(list(x = x, y = y), logisticstuff, c(0, 0))
```

Table 2: Modified Newton Raphson result

	Number of trial	Log likelihood	Alpha	Beta	asc_dir_check
res	0	-1.157406e+04	0.0000000	0.000000	7.198287e+04
	1	-8.042386e+08	4.2223234	12.580833	8.042865e+08
	2	-1.088052e+08	2.2239240	12.579133	1.088407e+08
	3	-1.469929e+07	0.2357413	12.566587	1.472241e+07
	4	-1.974029e+06	-1.6774539	12.474415	1.985259e+06
	5	-2.613716e+05	-3.0633352	11.821944	2.638348e+05
	6	-3.380200e+04	-1.7404639	8.278097	3.621234e+04
	7	-3.385355e+03	1.3119337	2.917549	3.114104e+03
	8	-1.341807e+03	1.9560050	1.099951	8.539803e+00
	9	-1.337466e+03	1.9807749	1.010018	0.000000e+00
	10	-1.337466e+03	1.9807752	1.010016	0.000000e+00

```
## True Alpha: 2, True Beta: 1
```

Hessian matrix from Poisson regression model is negative definite and the log-likelihood function of the Poisson regression model is globally concave. Hence, as we can observed from table 1 and table 2, it is to be observed that the direction is always an ascent direction. Note that Asc_dir_check variable is computed by using such formula: $d^{'}\nabla f(\theta_{i-1}) = -(\nabla f(\theta_{i-1}))^{'}[\nabla^2 f(\theta_{i-1})]^{-1}\nabla f(\theta_{i-1})$ and we are checking if Asc_dir_check is positive to confirm that its direction is ascent. With step halving application, we observe that the Modified Newton Raphson algorithm reduces about twice the number of trials compared to the basic Newton Raphson algorithm. Given that the true α is 2 and the true β is 1, both the basic and the Modified Newton Raphson algorithm successfully find alpha and beta values that are very close to the true values.