Homework 2 / 02-13-2022

Renjie Wei UNI:rw2844

```
library(tidyverse)
```

Problem 1

Develop two Monte Carlo methods for the estimation of $\theta = \int_0^1 e^{x^2} dx$ and implement in **R** .

Answer:

Method 1: simple Monte Carlo integration

$$\theta = \int_0^1 e^{x^2} dx = \int_0^1 e^{x^2} \cdot 1dx = E\{g(U)\}, U \sim U(0,1)$$

So we can get the estimation of θ by estimating the $E\{g(U)\}$

$$E\{g(U)\} \approx \frac{1}{n}\sum_{i=1}^{n} g(U_i) = \hat{\theta}$$

```
set.seed(2022)
N = 1e5
u <- runif(N)
theta_hat_1 = mean(exp(u^2))
```

From simple Monte Carlo integration, the estimated $\hat{\theta}_1$ = 1.4632808

Method 2: Importance Sampling

Suppose X is a r.v. with density $f(x)$, such that $f(x) > 0$ on the set $x : g(x) > 0$. Let Y be r.v. $g(X)/f(X)$. Then we can transform the integration to the following form:

$$\theta = \int_0^1 e^{x^2} dx = \int_0^1 \frac{e^{x^2}}{f(x)} \cdot f(x)dx = E[Y]$$

Then we can estimate the integration by estimating $E[Y]$:

$$E[Y] = \frac{1}{m}\sum_{i=1}^{m} Y_i = \frac{1}{m}\sum_{i=1}^{m} \frac{e^{X_i^2}}{f(X_i)}$$

Where $X_i \overset{i.i.d}{\sim} f(x)$, and in this case, I chose $f(x) = \frac{e^{-x}}{1-e^{-1}}, 0 < x < 1$, and using inverse CDF to sample from this distribution.

```
set.seed(2022)
# invCDF of fx
fx <- -log(1-u*(1-exp(-1)))
fg = exp(fx^2)/(exp(-fx)/(1-exp(-1)));
theta_hat_2= mean(fg)
```

From Importance Sampling, the estimated $\hat{\theta}_2$ *= 1.4630146*

Problem 2

Show that in estimating $\theta = E\{\sqrt{1-U^2}\}$ *it is better to use* U^2 *rather than* U *as the control variate, where* $U \sim U(0,1)$*. To do this, use simulation to approximate the necessary covariances. In addition, implement your algorithms in* **R**.

Answer:

Since $E[f_1(x)] = E[U^2] = \int_0^1 x^2 dx = \frac{1}{3}, E[f_2(x)] = E[U] = 0.5$, so we can estimate θ by $E[g(x)] = E\{\sqrt{1-U^2}\} = E\{\sqrt{1-U^2} + c_1 \times (U - 0.5)\} = E\{\sqrt{1-U^2} + c_2 \times (U^2 - \frac{1}{3})\}$. And we choose

$$c_i = -\frac{Cov(g(X), f_i(X))}{Var(f_i(X))}, X \sim U(0,1)$$

To get c_i, we need $Cov(g(U), f_i(U))$ and $Var(f_i(U))$. In this case, these can be estimated from a preliminary Monte Carlo experiment.

```
set.seed(2022)
gfun <- function(x) sqrt(1-x^2)
# f_1_x
ffun_1 <- function(x) x^2
# f_2_x
ffun_2 <- function(x) x
# generating X from Unif(0,1)
n_2 = 1e6
u_2 = runif(n_2)

gx = gfun(u_2)
fx_1 = ffun_1(u_2)
fx_2 = ffun_2(u_2)
# Calculate cov and var
cov_gf1 = cov(gx,fx_1)
cov_gf2 = cov(gx,fx_2)
var_f1 = var(fx_1)
var_f2 = var(fx_2)
# compute the constant c_i's
c1 = -cov_gf1/var_f1
c2 = -cov_gf2/var_f2
# get the estimates of theta
T_c1 = gx+c1*(fx_1 - 1/3)
T_c2 = gx+c2*(fx_2 - 1/2)
# compare the estimated theta and their variance
theta_c1 = mean(T_c1)
```

```
theta_c2 = mean(T_c2)
var_reduce_1 = cov_gf1^2/var_f1
var_reduce_2 = cov_gf2^2/var_f2
imp_pct = (var(T_c2)-var(T_c1))/var(T_c2)
```

Since we set $f_1(U) = U^2, f_2(U) = U$, and from Monte Carlo simulation, we got $Cov(g(U), f_1(U)) = Cov(\sqrt{1-U^2}, U^2)$ = -0.065,$Cov(g(U), f_2(U)) = Cov(\sqrt{1-U^2}, U)$ = -0.059. The $Var(g(U))$ is reduced by $\frac{[Cov(g(U),f_1(U))]^2}{Var(f_1(U))} = 0.048$ and $\frac{[Cov(g(U),f_2(U))]^2}{Var(f_2(U))} = 0.042$, so we can see that $f_1(U) = U^2$ is a better control variate than $f_2(U) = U$, and the percent of improvement in variance reduction is 78.41 %. The estimated θ_{ci} is $\theta_{c1} = 0.785$ and $\theta_{c2} = 0.786$.

Problem 3

Obtain a Monte Carlo estimate of

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

by importance sampling and evaluate its variance. Write a **R** *function to implement your procedure.*

Answer:

Let $g(x) = \frac{x^2}{\sqrt{2\pi}} e^{-\frac{x^2}{2}} dx, x > 1$ I use the pdf of $Exp(1)$, that is \$f(x) =e^{-x},x>1 \$ as my importance function. The integration can be written as $\theta = \int_1^\infty \frac{x^2}{\sqrt{2\pi}} e^{-\frac{x^2}{2}} dx = \int_1^\infty \frac{g(x)}{f(x)} f(x) dx = E[\frac{g(x)}{f(x)}]$, and we can estimate this integration by estimating $E[\frac{g(x)}{f(x)}]$.

```
set.seed(2022)

g <- function(x) {
  x ^ 2 / sqrt(2*pi) * exp(-x^2/2) * (x > 1)
}
f_imp <- function(x) {
     exp(-x)
}

r_imp <- function(N) {
    rexp(N)
}
# function of importance sampling, given g(x), f(x), and sampling function from f(x)
imp_sampling <- function(gfun, ffun, rffun, N = 1e7){
    xs <- rffun(N)
    T_x <- gfun(xs)/ffun(xs)
    return(c(mean(T_x),var(T_x)))
}
imp_result <- imp_sampling(g,f_imp,r_imp,1e7)
```

The result shows that the estimated $\hat{\theta} = 0.4006$, and the $Var(\hat{\theta}) = 0.3432$. The true value of θ can be derived by the following code:

```r
integrate(g, 1, Inf)
```

```
## 0.400626 with absolute error < 5.7e-07
```

We can see that our Monte Carlo estimator is very close to the true parameter.

Problem 4:

Design an optimization algorithm to find the minimum of the continuously differential function

$$f(x) = -e^{-x}\sin(x)$$

on the closed interval $[0, 1.5]$. *Write out your algorithm and implement it into* ***R***.

Answer:

I'd like to use Newton's method as the optimization algorithm. The Newton algorithm involves doing this iteratively:

$$\theta_i = \theta_{i-1} - \frac{f(\theta_{i-1})}{f'(\theta_{i-1})}$$

Until $|f(\theta_i)|$ *is sufficiently close to zero. Applying Newton's method to this problem, we got the local minimun when* $f'(x) = 0$

$$f'(x) = e^{-x}\sin(x) - e^{-x}\cos(x) = e^{-x}(\sin(x) - \cos(x))$$
$$f''(x) = 2e^{-x}\cos(x)$$

so each time we update the x_i *by the following equation:*

$$x_i = x_{i-1} - \frac{e^{-x}(\sin(x) - \cos(x))}{2e^{-x}\cos(x)}$$

```r
#R codes:
ffunc <- function(a){
    return(-exp(a)*sin(a))
}
d_func <- function(a){
    return(exp(-a)*(sin(a)-cos(a)))
}
d2_func <- function(a){
    return(2*exp(-a)*cos(a))
}
i = 0
tol = 1e-10
cur <- start <- 0 # start from 0
resnewton <- c(i, cur, d_func(cur))
while(abs(d_func(cur)) > tol){
    i <- i+1
    renew <- cur - d_func(cur)/d2_func(cur)
    cur <- renew
    resnewton <- rbind(resnewton, c(i, cur, d_func(cur)))
}
min_x  = resnewton[nrow(resnewton), 2]
```

From the Newton's method, we got the minimum of $f(x) = -e^{-x}\sin(x)$ *in* $[0, 1.5]$. *When x = 0.7853982, the minimum of* $f(x) = -e^{-x}\sin(x)$ *in* $[0, 1.5]$ *is -1.5508832.*

Problem 5:

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

$$Y_i \sim Poisson(\lambda_i),$$

where

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

for some explanatory variable x_i. The question is how to estimate α and β given a set of independent data $(x_1, Y_1), (x_2, Y_2), \ldots, (x_n, Y_n)$.

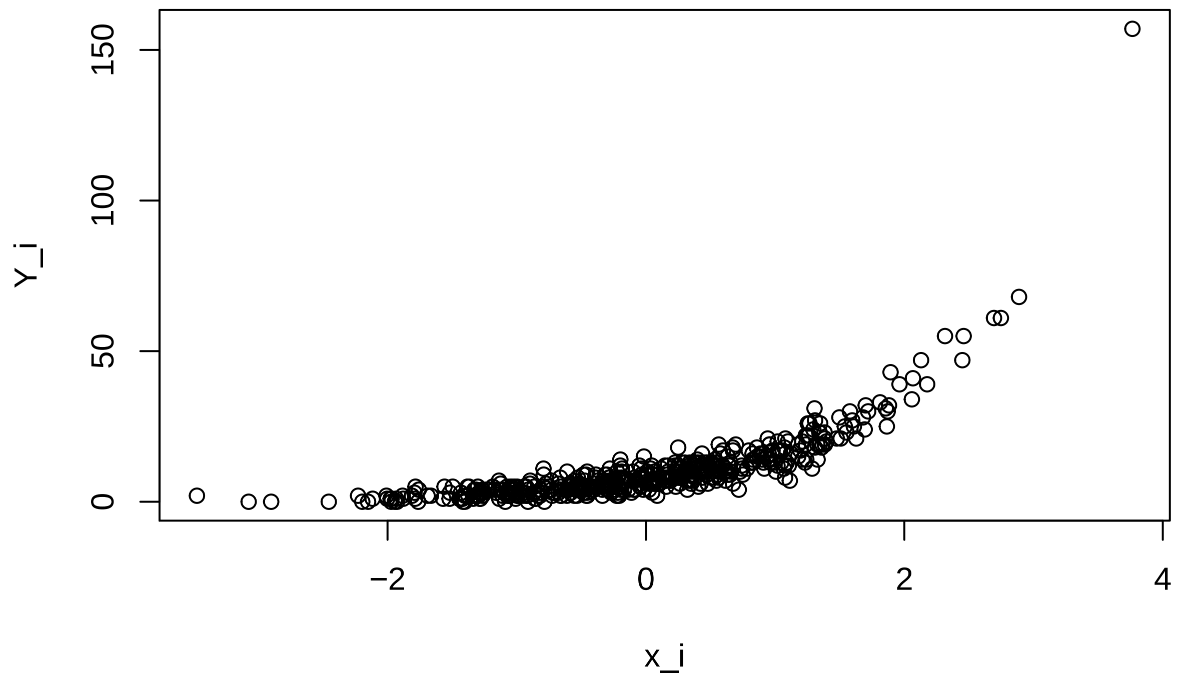
1. *Generate a random sample (x_i, Y_i) with $n = 500$ from the Possion regression model above. You can choose the true parameters (α, β) and the distribution of X.*
2. *Write out the likelihood of your simulated data, and its Gradient and Hessian functions.*
3. *Develop a modify Newton-Raphson algorithm that allows the step-halving and re-direction steps to ensure ascent directions and monotone-increasing properties.*
4. *Write down your algorithm and implement it in R to estimate α and β from your simulated data.*

Answer:

1. *Generate a random sample (x_i, Y_i) with $n = 500$ from the Poisson regression model above.*

Suppose the true *parameters are $\alpha = 2, \beta = 0.8$. And $x_i \sim N(0, 1)$. The generated data are displayed in the following plot.*

```
set.seed(2022)
n <- 500
# generating x_i and Y_i
alpha <- 2
beta <- 0.8
x_i <- rnorm(n)
lambda_i <- exp(alpha + beta*x_i)
Y_i <- rpois(n, lambda_i)
samples <- tibble(
    x = x_i,
    y = Y_i
)
plot(x_i,Y_i,type = "p")
```



2. *Write out the likelihood of your simulated data, and its Gradient and Hessian functions.*

Since $Y_i \sim Pois(\lambda_i)$. *So the likelihood function of the data is:*

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

Maximizing the likelihood is equivalent to maximizing the log-likelihood:

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

So the gradient of this function is:

$$\nabla \ell(\alpha,\beta) = \begin{bmatrix} \frac{\partial \ell}{\partial \alpha} \\ \frac{\partial \ell}{\partial \beta} \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{n} \left(y_i - e^{\alpha+\beta x_i}\right) \\ \sum_{i=1}^{n} x_i(y_i - e^{\alpha+\beta x_i}) \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{n} (y_i - \lambda_i) \\ \sum_{i=1}^{n} x_i(y_i - \lambda_i) \end{bmatrix}$$

The Hessian is given by:

$$\nabla^2 \ell(\alpha,\beta) = -\begin{bmatrix} \sum_{i=1}^{n} e^{\alpha+\beta x_i}, \sum_{i=1}^{n} x_i e^{\alpha+\beta x_i} \\ \sum_{i=1}^{n} x_i e^{\alpha+\beta x_i}, \sum_{i=1}^{n} x_i^2 e^{\alpha+\beta x_i} \end{bmatrix} = -\begin{bmatrix} \sum_{i=1}^{n} \lambda_i, \sum_{i=1}^{n} x_i \lambda \\ \sum_{i=1}^{n} x_i \lambda_i, \sum_{i=1}^{n} x_i^2 \lambda_i \end{bmatrix}$$

3. *Develop a modify Newton-Raphson algorithm that allows the step-halving and re-direction steps*

To allow the step-halving to ensure monotone-increasing properties, we modified the Newton-Raphson at each step with a check-function:

$$\theta_i(\gamma) = \theta_{i-1} - \gamma[\nabla^2\ell(\theta_{i-1})]^{-1}\nabla\ell(\theta_{i-1})$$

if $f(\theta_i(1)) > f(\theta_{i-1})$, then set $\theta_i = \theta_{i-1}$. Otherwise, search for a value $\gamma \in (0,1)$ for which $f(\theta_i(\gamma)) > f(\theta_{i-1})$ and set $\theta_i = \theta_i(\gamma)$. In step-halving, we search $\gamma \in \{1/2, 1/4, 1/8, \dots\}$.

To allow re-direction steps and ensure we get ascent direction in each step, we have to make sure the Hessian matrix is negative definite. *We can ensure that by changing the Newton-Raphson algorithm direction$\mathbf{d} = -[\nabla^2\ell(\theta_{\mathbf{i-1}})]^{-1}\nabla\ell(\theta_{\mathbf{i-1}})$ to $\mathbf{d}' = -[\nabla^2\ell(\theta_{\mathbf{i-1}}) - \eta\mathbf{I}]^{-1}\nabla\ell(\theta_{\mathbf{i-1}})$, where η is large enough to make sure the modified Hessian is negative definite, that is, forced all eigenvalue of modified Hessian to less than zero. A simple choice of η is the largest positive eigenvalue of Hessian then plus one.*

4. *Write down your algorithm and implement it in R to estimate α and β from your simulated data.*

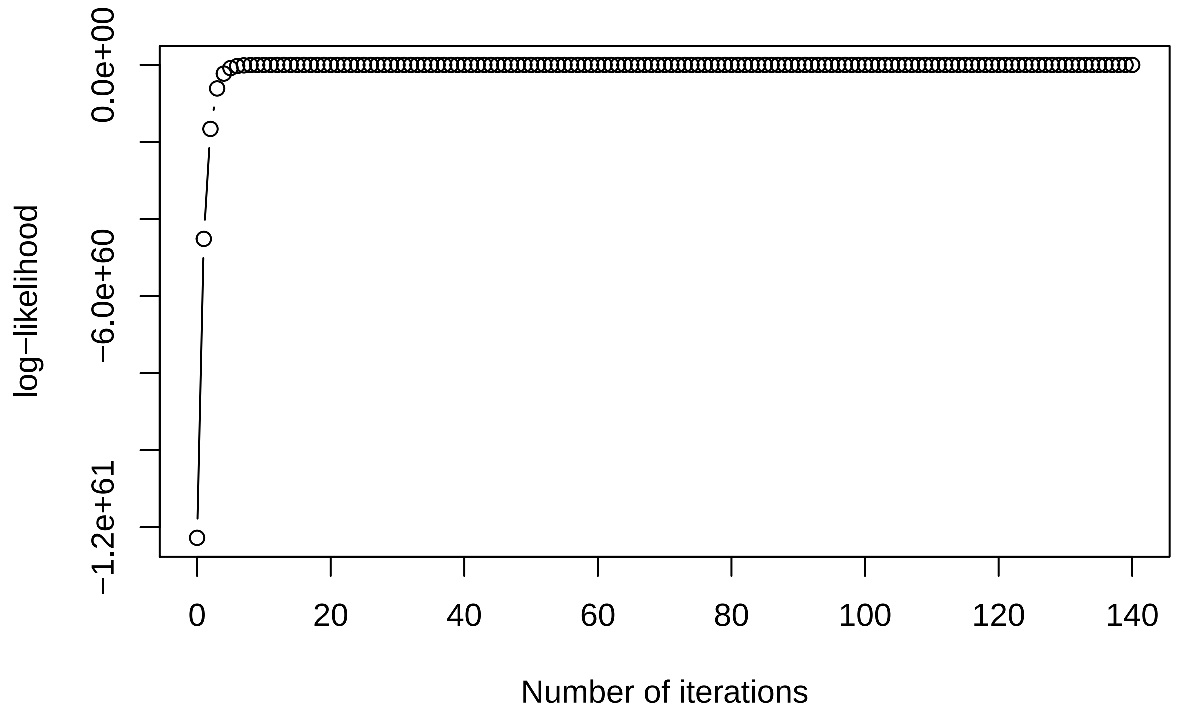
The code of the modified Newton-Raphson algorithm is in the following trunk.

```r
# function giving gradient and Hessian
poissonstuff <- function(dat, betavec){
    u <- betavec[1] + betavec[2]*dat$x
    lambda <- exp(u)
    # loglik at betavec
    loglik <- sum(dat$y * u - exp(u) - log(factorial(dat$y)))
    # get the gradient at betavec
    grad <- c(sum(dat$y - lambda), sum(dat$x*(dat$y - lambda)))
    # hessian matrix at betavec
    Hess <- -matrix(
        c(sum(lambda), sum(dat$x*lambda), sum(dat$x*lambda), sum((dat$x)^2*lambda)), ncol = 2
    )
    return(list(loglik = loglik, grad = grad, Hess = Hess))
}

ModifiedNewtonRaphson <- function(dat, func, start, tol = 1e-10, maxiter = 1000){
    i <- 0
    cur <- start
    stuff <- func(dat, cur)
    res <- (c(0, stuff$loglik, cur))
    prevloglik <- -Inf
    while (i < maxiter && abs(stuff$loglik - prevloglik) > tol) {
        i <- i + 1
        prevloglik <- stuff$loglik
        prev <- cur
        Hess <- stuff$Hess
        e_hess <- eigen(Hess)$values
        # modified algorithm 1: re-direction
        # check if the hess is negative definite
        if (sum(e_hess) > 0){
            eta <- max(e_hess) + 1
            Hess <- Hess - eta*diag(x = 1, dim(Hess))
        }
        # modified algorithm 2: step-halving
        cur <- prev - solve(Hess) %*% stuff$grad
        j <- 0
        while (func(dat, cur)$loglik < func(dat, prev)$loglik) {
            j <- j + 1
            Gamma <- 0.5^(j)
            cur <- prev - Gamma * solve(Hess) %*% stuff$grad
        }
        # modification procedure ends
        stuff <- func(dat, cur)
        res <- rbind(res, c(i, stuff$loglik, cur))
    }
    return(res)
}
```

I choose a extreme combination of $\{\alpha, \beta\}$ at the start value to test the robustness of my algorithm.

```
ans <- ModifiedNewtonRaphson(samples, poissonstuff, c(-47,-54))
ans <- data.frame(ans)
colnames(ans) <- c("iterations", "loglik", "alpha", "beta")
plot(ans$iterations, ans$loglik, type = "b", xlab = "Number of iterations", ylab = "log-likelihood")
```



```
alpha.hat <- ans[nrow(ans),]$alpha
beta.hat <- ans[nrow(ans),]$beta
```

We can show the trace of the log-likelihood of the optimization procedure.

There are total 140 iterations, and the estimated parameters are $\hat{\alpha}$ = 1.9966463 and $\hat{\beta}$ = 0.7897409. Remembering the true values are $\alpha = 2, \beta = 0.8$, which means that our algorithm has a high accuracy in estimating the parameters of the model.