Homework 2

Anna Ma (ym2813)

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: importance sampling. Use the uniform distribution as the importance distribution:

```
set.seed(8160)
g1 = function(x) {exp(x^2)}

N = 10000
u1 = runif(N)
theta11 = sum(g1(u1))/N

theta11
```

```
## [1] 1.458144
```

Method 2: Find a function who has similar graph with e^{x^2}, for example, $f(x) = 1+x^4$, use it as the importance distribution for the estimation

```
x12 = 1+u1^4
fg12 = g1(x12)/(1+x12^4)
theta12 = sum(fg12)/N
theta12
```

```
## [1] 1.489974
```

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:

```
# Define functions
g2 = function(x) {sqrt(1 - x^2)}
f1 = function(x) {x}
f2 = function(x) {x^2}
```

```r
set.seed(8160)
# generate random number U ~ U(0,1)
u = runif(10000)
```

```r
ga = g2(u)
theta = mean(ga)
va_crude = var(ga)

# Use U as the control variate
f1a = f1(u)
beta1 = lm(ga~f1a)$coef[2]
hha1 = beta1*1 + (ga - beta1*f1a)

c(va_crude, var(hha1))
```

```
## [1] 0.049950286 0.007566322
```

```r
## Using U^2 as the control variate
f2a = f2(u)
beta2 = lm(ga~f2a)$coef[2]
hha2 = beta2*1/3 + (ga - beta2*f2a)

c(va_crude, var(hha2))
```

```
## [1] 0.049950286 0.001656127
```

```r
#efficiency improvement
eff1 = (va_crude - var(hha1))/va_crude
eff2 = (va_crude - var(hha2))/va_crude
```

U^2 has a variance of 0.0016561, which is smaller than both U's variance 0.0075663 and the crude estimator's variance of 0.0499503. Using U^2 also improves the efficiency by 0.9668445, which is more than the improvement of 0.8485229 brought by U. Therefore, U^2 is a better control variate.

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 the standard normal distribution to be the importance function such that

$$P(x) = e^{-x}, 0 < x < \infty$$

and

$$g(x) = \frac{x^2}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}$$

```r
# g(x) nominal distribution
g3 = function(x) {((x^2/sqrt(2*pi))*exp(-x^2/2))}
```

```r
set.seed(8160)
N = 10000
# Let the standard normal distribution to be the importance function

# integral from 0 to inf
x = rexp(N,1)
fg = g3(x)/exp(-x)
theta_hat1 = sum(fg)/N

# integral from 0 to 1

x1 = runif(N)
fg1 = g3(x1)
theta_hat2 = sum(fg1)/N

theta_hat = theta_hat1 - theta_hat2
var = var(fg) + var(fg1) - 2*cov(fg, fg1)

theta_hat
```

```
## [1] 0.4014728
```

```r
var
```

```
## [1] 0.2956245
```

The estimate is 0.4014728, and the variance is 0.2956245

Problem 4:

Design an optmization algorithm to find the minimum of the continuously differentiable function

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

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

Answer

```r
f = function(x){
  return(-exp(-x)*sin(x))
}

find_min = function(f,w,a,b,tol){
  x1 = (1 - w) * (b - a) + a
  x2 = x1 + w * (b - a) * (1 - w)
  while (abs(x1 - x2) > tol) {
    if( f(x1) < f(x2) ) {
      a = a
      b = x2
    }
    else {
      a = x1
      b = b
    }
```

```r
      x1 = (1 - w) * (b - a) + a
      x2 = x1 + w * (b - a) * (1 - w)
    result = f(a)
  }
  return(result)
}
```

In case when $f = -e^{-x}\sin(x)$, w is set to be the golden ratio such that w = 0.618, interval [a,b] = [0,1.5], and tolerance is 1e-10,

```r
find_min(f,0.618,0,1.5,1e-10)
```

```
## [1] -0.3223969
```

```r
f(0.7854043)
```

```
## [1] -0.3223969
```

verify the result:

```r
optimize(f, interval = c(0,1.5))
```

```
## $minimum
## [1] 0.7854043
##
## $objective
## [1] -0.3223969
```

Therefore, we find that the minimum of the function $f = -e^{-x}\sin(x)$ is -0.3223969 when x = 0.7854043

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 Possion regression model.*

Let the true parameters to be such that $\alpha = 1 and, \beta = 0.3$, and let X follow a normal distribution

```
gen_data = function(n,a,b){
  x = rnorm(n)
  lambda = exp(a + b * x)
  y = rpois(n,lambda)
  dat = list(x = x,y = y)
  return(dat)
}
```

2. *Likelihood of the simulated data, and its Gradient and Hessian functions.*

$L(\lambda; y_i) = \prod_{j=1}^{n} exp(-\lambda)\frac{1}{y_j!}\lambda^{y_j}$

log likelihood function $l(\lambda; y_i) = -n\lambda - \sum_{i=j}^{n} ln(y_j!) + ln(\lambda)\sum_{j=1}^{n} y_j$

```
# define function to get loglikelihood function, gradience and Hessian matrix
poissonstuff = function(dat, betavec) {
  alpha = betavec[1]
  beta = betavec[2]
  log_lambda = alpha + beta * dat$x
  lambda = exp(log_lambda)
  # Log-likelihood at betavec
  loglik = sum(dat$y * log_lambda - lambda - log(factorial(dat$y)))
   # gradient at betavec
  grad = c(sum(dat$y * dat$x - dat$x * lambda),
            sum(dat$y - lambda))
   # Hessian at betavec
  Hess = matrix(c(sum((-1)*lambda),rep(sum((-1)*dat$x * lambda),2),sum(((-1)*dat$x^2 * lambda))), ncol
  return(list(loglik = loglik, grad = grad, Hess = Hess))
}
```

3. *Newton Raphson algorithm*

```
NewtonRaphson = function(dat, func, start, tol=1e-10, maxiter = 500) {
  i = 0
  cur = start
  stuff = func(dat, cur)
  l = 1
  res = c(0, stuff$loglik, cur)
  prevloglik = -Inf
    while (i < maxiter && abs(stuff$loglik - prevloglik) > tol) {
      i=i+1
      prevloglik = stuff$loglik
      prev_stuff = stuff
      Hess  = stuff$Hess
      grad = stuff$grad
      prev = cur
      if( t(grad) %*% Hess %*% grad > 0 ){
      Hess = Hess - 3*max(diag(Hess)) * diag(nrow(Hess))
      }
      else{Hess = Hess}
      cur = prev - l * solve(Hess) %*% grad
```

```r
      stuff = func(dat, cur)
      while(stuff$loglik <= prevloglik) {
        l = l / 2
        cur = prev - l * solve(prev_stuff$Hess) %*% prev_stuff$grad
        stuff = func(dat, cur)
        }
        res = rbind(res, c(i, stuff$loglik, cur))
    }
    colnames(res) = c("iter","likelihood","alpha","beta")
    return(res)
}
```

4. Estimate the parameters

```r
set.seed(8160)
#When true parameter is alpha = 2, beta = 1
data1 = gen_data(500,2,1)
#start point at (1,0.5)
NewtonRaphson(data1,poissonstuff,start = c(1,0.5))
```

```
##      iter likelihood    alpha     beta
## res     0  -5227.336 1.000000 0.500000
##         1  -2026.514 2.121716 1.314058
##         2  -1876.219 1.612082 1.508935
##         3  -1876.219 1.612082 1.508935
```