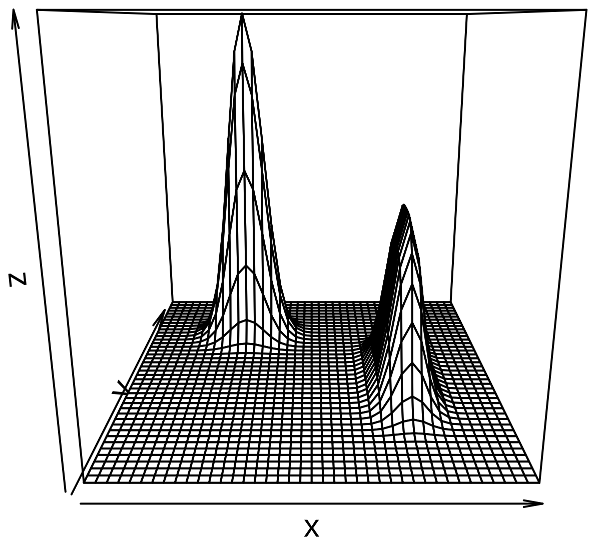
GRADUATE STUDENT STAT 840 A1

Vsevolod Ladtchenko 20895137

Problem 9

(a)

```r
f = function(x,y)
{
  exp1 = -90*(x-0.5)^2 -45*(y+0.1)^4
  exp2 = -45*(x+0.4)^2 -60*(y-0.5)^2
  result = 0.5*exp(exp1) + exp(exp2)
  return(result)
}

plot_3d = function(f)
{
  x = seq(-1,1,0.05)
  y = seq(-1,1,0.05)
  z = outer(x, y, f)
  persp(x, y, z)
}

riemann_integral = function(f)
{
  # integrate using Riemann sums
  dx = 0.002 # each interval length is (1 - -1)/1000 = 0.002
  dy = dx
  x = seq(-0.998,1,dx) # grid of 1000 x 1000 points
  y = seq(-0.998,1,dy)
  length(x)

  integral = 0
  for (i in x)
  {
    for (j in y)
    {
      z = f(i,j)
      integral = integral + z*dy*dx
    }
  }
  return(integral)
}

plot_3d(f)
```



```
riemann_integral(f)
```

```
## [1] 0.125844
```

(b)

The MC method seems to approach the correct answer of part (a). But its efficiency is questionable: in part (a) we evaluated f 1 million times, since we looped over each of 1000 elements of x and each of 1000 elements of y. If we run the MC simulation with n = 1,000,000 then the CI is (.12590, .12632) so we don't even have 3 significant digits. This is not effective here because a lot of the points are near 0 as seen in the graph, and most draws from the uniform distribution will miss the areas of concentrated mass, thus leading to high variability of the estimate.

```
get_SE = function(x) # return variance of sample mean
{
  n = length(x)
  mu = sum(x) / n
  s_sq = sum((x - mu)^2) / ((n-1)*n) # notes p 30
  return(sqrt(s_sq))
}

get_CI = function(xbar, se, conf=.95)
{
  a = 1 - conf
  L = xbar + qnorm(a/2)*se
  U = xbar + qnorm(1 - a/2)*se
  return(c(L,U))
}

run_mc = function(n)
{
  x = runif(n,-1,1)
  y = runif(n,-1,1)

  deltas = rep(0,n)

  integral = 0
  for (i in 1:n)
  {
    z = f(x[i], y[i])
    deltas[i] = z
    # include a factor of 4 because the pdf of uniform(-1,1) is 1/2 and
    # for two axes will be (1/2)*(1/2) = 1/4
    # similar to question 8.
    integral = integral + 4*z/n
  }
  integral
  se = get_SE(deltas)
  ci = get_CI(integral, se)
  return(round(c(integral=integral, se=se, ci=ci), 5))
}
run_mc(1000)
```

```
## integral       se      ci1      ci2
##  0.12805  0.00342  0.12134  0.13477
```

```
run_mc(5000)
```

```
## integral       se      ci1      ci2
##  0.12776  0.00158  0.12467  0.13086
```

```
run_mc(10000)
```

```
## integral       se      ci1      ci2
##  0.11911  0.00104  0.11708  0.12114
```

```
run_mc(1000000)
```

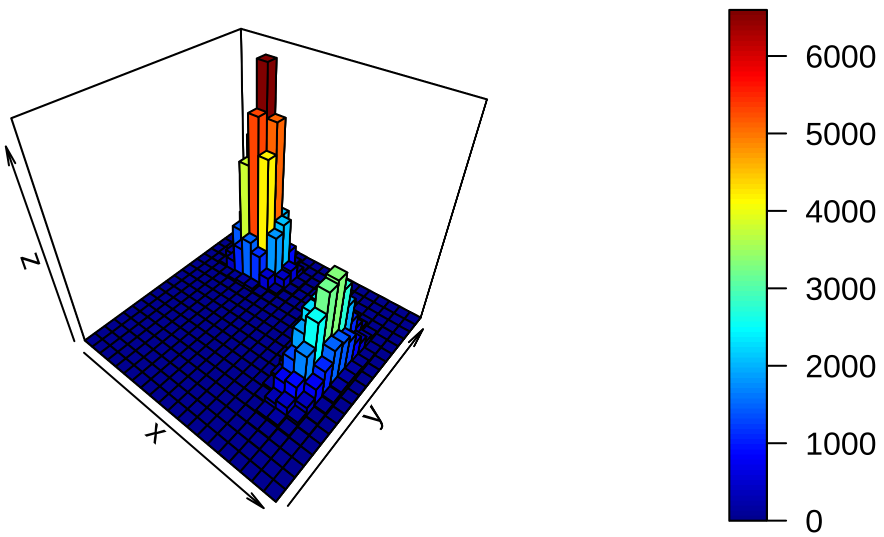
```
## integral       se      ci1      ci2
##  0.12523  0.00011  0.12502  0.12544
```

(c)

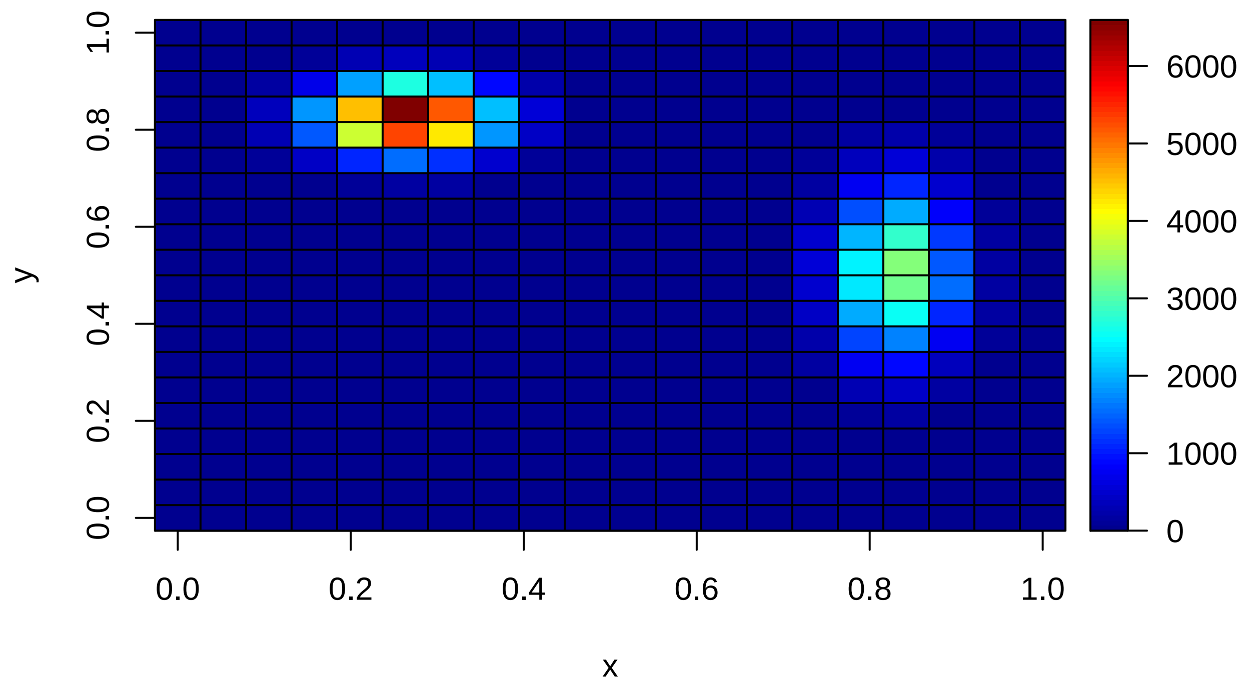
My first answer was wrong for 2 reasons: first, I did not sample the bivariate normal correctly. On piazza it was explained that I need to generate a Bernoulli 0.46 RV and when it is 1, sample from the first MVnorm, and if 0, sample from the other MVnorm. Second, I did not include the normalization constant of g. I have not figured it out analytically. But running the Riemann integration from part (a) yields around 0.1128183 (just replace f with g). When the IS estimate is multiplied by this constant, the result seems reasonable.

```
library(plot3D)
library(MASS)

g = function(x,y)
{
```

```r
  exp1 = -90*(x-0.5)^2 -10*(y+0.1)^2
  exp2 = -45*(x+0.4)^2 -60*(y-0.5)^2
  result = 0.5*exp(exp1) + exp(exp2)
  return(result)
}

plot_g = function(x, y)
{
  ##  Create cuts:
  x_c <- cut(x, 20)
  y_c <- cut(y, 20)
  ##  Calculate joint counts at cut levels:
  z <- table(x_c, y_c)
  hist3D(z=z, border="black")
  image2D(z=z, border="black")
}

run_is = function(n)
{
  deltas = rep(0,n)
  for (i in 1:n)
  {
    deltas[i] = f(x[i], y[i]) / g(x[i], y[i])
  }
  integral = mean(deltas) * 0.1128183 # the integral of g using part (a)
  se = get_SE(deltas)
  ci = get_CI(integral, se)
  return(round(c(integral=integral, se=se, ci=ci), 5))
}

n = 100000
bern = rbinom(n,1,0.46)
mv1 = mvrnorm(n=n,mu=c(0.5,-0.1),Sigma=matrix(c(1/180, 0, 0, 1/20), ncol=2))
mv2 = mvrnorm(n=n,mu=c(-0.4,0.5),Sigma=matrix(c(1/90, 0, 0, 1/120), ncol=2))
xx = mv1*bern + mv2*(1-bern)
x <- xx[,1]
y <- xx[,2]

plot_g(x, y)
```



z
x
y
6000
5000
4000
3000
2000
1000
0



```
run_is(1000)
```

```
## integral        se       ci1       ci2
##  0.12628  0.00785  0.11089  0.14168
```

```
run_is(5000)
```

```
## integral        se       ci1       ci2
##  0.12549  0.00329  0.11905  0.13193
```

```
run_is(10000)
```

```
## integral        se       ci1       ci2
##  0.12562  0.00229  0.12114  0.13011
```