Computational Statistics - STAT 575 - HW #3

Alex Towell (atowell@siue.edu)

Problem 1

Consider the following integration

$$\int_{-\infty}^{\infty} e^{-x^2} dx.$$

Part (a)

$\boxed{\text{Evaluate the integral in closed form using } \pi.}$

The integrand is the kernel of a normal density, and so we evaluate the integral by transforming it into a problem recognizable as an expectation $\mathrm{E}_X[k(\pi)]$ where $X \sim N(0, \sigma^2)$, i.e.,

$$\sqrt{2\pi\sigma^2} \int_{-\infty}^{\infty} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}x^2} dx.$$

By basic pattern matching, we see that integrand $\exp(-x^2)$ is the kernel of $N(0, \sigma^2 = 0.5)$. Substituting σ^2 with 0.5, we obtain

$$\sqrt{\pi} \int_{-\infty}^{\infty} \frac{1}{\sqrt{\pi}} e^{-x^2} dx$$

which is equivalent to $\sqrt{\pi}\,\mathrm{E}_X(1) = \sqrt{\pi}$.

Part (b)

$\boxed{\begin{array}{l}\text{Estimate the above integral using Riemman's Rule. Give a estimate} \\ \text{of } \pi\text{. Does it at least provide a couple digits worth of accuracy?}\end{array}}$

The right-handed Riemann sum is given by

$$h \sum_{i=1}^{n} f(a + hi) \approx \int_a^b f(x) dx$$

where $h = (b - a)/n$, which we straightforwardly implement with:

```r
right_riemann_sum <- function(f, a, b, n) {
    h <- (b - a)/n
    h * sum(f(a + (1:n) * h))
}
```

We use this numerical integrator to approximate the integration problem whose solution is $\sqrt{\pi}$ and then take its square to estimate π, i.e.,

$$\hat{\pi}_{\text{riemann}} = \text{right\_riemann\_sum}(g, -r, r, n)^2.$$

where $g(x) = e^{-x^2}$, $(-r, r)$, $r > 0$, is the domain to numerically integrate over, and n is the number of blocks in the partition. We implement this estimator with the following code:

```r
riemann_pi <- Vectorize(function(n, r) {
    right_riemann_sum(function(x) {
        exp(-x^2)
    }, -r, r, n)^2
})
```

We provide an estimate of π with:

```r
riemann_pi(10, 4)
```

```
## [1] 3.141595
```

The estimator provides 5 decimals of accuracy based on Riemman's rule with $n = 10$ over $(-4, 4)$. This provides unusually good accuracy for the right-handed Riemann rule due to the fact that the integrand is symmetric about the origin and thus the overestimate of the integral over $(-4, 0)$ is canceled by the underestimate over $(0, 4)$.

There are more efficient and more accurate ways to estimate π, but in theory, if we ignore numerical errors on the computer,

$$\lim_{n \to \infty, r \to \infty} \text{riemann\_pi}(n, r) = \pi.$$

One may have the insight that, since e^{-x^2} is symmetric about the origin, we can just sum over $[0, r]$ instead, which would obtain $\sqrt{\pi}/2$. Let us try:

```r
4 * right_riemann_sum(function(x) {
    exp(-x^2)
}, 0, 4, 10)^2
```

```
## [1] 1.88363
```

This is a very poor estimate of π. Let us try with $n = 100000$:

```r
4 * right_riemann_sum(function(x) {
    exp(-x^2)
}, 0, 4, 1e+05)^2
```

```
## [1] 3.141451
```

Remarkably, it still performs relatively poorly compared to $\hat{\pi}_{\text{riemann}}$.

Part (c)

Redo part (b) to estimate π using Gauss-Hermite quadrature. You may use the fastGHQuad function in R.

```
library(fastGHQuad)
hermite_pi <- Vectorize(function(n) {
    aghQuad(function(x) {
        exp(-x^2)
    }, 0, 1.1, gaussHermiteData(n))^2
})
pi.hat.herm <- hermite_pi(10)
pi.hat.herm
```

```
## [1] 3.14025
```

We see that the estimator $\hat{\pi}_{\text{hermite}} = 3.14025$ with $n = 10$ is a relatively poor estimator compared to $\hat{\pi}_{\text{riemann}}$. Let us find

$$\epsilon^*_{\text{riemann}} = \min_{n,r}(\text{riemann\_pi}(n, r))$$

and

$$\epsilon^*_{\text{hermite}} = \min_{n}(\text{hermite\_pi}(n)).$$

Here is the code that finds the argument that minimizes these estimators, along with their respective errors:

```
arg.min <- function(xs, f) {
    y.min <- Inf
    x.min <- NULL
    for (x in xs) {
        y <- f(x)
        if (y < y.min) {
            x.min <- x
            y.min <- y
        }
    }
    list(x.min = x.min, y.min = y.min)
}
```

Now, we try it:

```
hermite.min <- arg.min(10:100, function(n) {
    abs(hermite_pi(n) - pi)
})
riemann4.min <- arg.min(10:100, function(n) {
    abs(riemann_pi(n, 4) - pi)
})
riemann5.min <- arg.min(10:100, function(n) {
    abs(riemann_pi(n, 5) - pi)
})
riemann6.min <- arg.min(10:100, function(n) {
    abs(riemann_pi(n, 6) - pi)
})
print(hermite.min)
```

```
## $x.min
## [1] 45
##
## $y.min
## [1] 4.440892e-16
```

```
print(riemann4.min)
```

```
## $x.min
## [1] 100
##
## $y.min
## [1] 1.002528e-07
```

```
print(riemann5.min)
```

```
## $x.min
## [1] 100
##
## $y.min
## [1] 1.046851e-11
```

```
print(riemann6.min)
```

```
## $x.min
## [1] 24
##
## $y.min
## [1] 4.440892e-16
```

We see that, likely due to numerical errors in the default implementation of numbers in R, $\hat{\pi}_{\text{hermite}}$ obtains its best estimate at $n = 45$ and $\hat{\pi}_{\text{riemann}}$ obtains its best estimate at $n = 24$

and $r = 6$. Moreover,

$$\epsilon^*_{\text{hermite}} = \epsilon^*_{\text{riemann}} = 4.440892 \times 10^{-16}.$$

Interesting.

Problem 2

Use Monte Carlo simulation to evaluate the confidence (coverage) level of 95% CI for regression slope in the model

$$y_i = 3x_i + \epsilon_i, \epsilon_i \sim N(0, 1).$$

In each Monte Carlo sample, first generate a vector of x (you may pick x from any distribution, say a normal or a uniform). Then generate ϵ from $N(0, 1)$ and then y according to the regression formula.
Use lm() to fit the regression model, and confint() to get the 95% confidence interval for the slope parameter. Run the MC iterations for 10000 times and get the proportion of CI that covers the true slope $\beta_1 = 3$. Verify the proportion is close to 0.95.

```
N <- 10000
covers <- 0
n <- 1000
for (i in 1:N) {
    x <- runif(n, -100, 100)
    e <- rnorm(n)
    y <- 3 * x + e
    fit <- lm(y ~ x)
    ci <- confint(fit)
    if (ci[2] <= 3 && 3 <= ci[4])
        covers <- covers + 1
}

prop <- covers/N
prop
```

```
## [1] 0.9511
```

As we can see, the proportion of confidence intervals that cover the true parameter value is approximately 95%.

Problem 3

Let $Y \sim \text{Bernoulli}(0.7)$ and the conditional distribution of X given Y is $X|Y \sim N(\mu_Y, 1)$, where $\mu_0 = -2$ and $\mu_1 = 2$.

Part (a)

Derive the marginal pdf of X.

First, we compute the joint pdf of X and Y, which is just

$$f_{X,Y}(x,y) = f_Y(y) f_{X|Y}(x|y)$$

which is

$$f_{X,Y}(x,y) = (.7)^y (.3)^{1-y} \left(I(y=0)\phi(x+2) + I(y=1)\phi(x-2) \right).$$

The marginal pdf f_X is computed by summing over Y,

$$f_X(x) = f_{X,Y}(x,0) + f_{X,Y}(x,1)$$

which is just

$$f_X(x) = 0.3\phi(x+2) + 0.7\phi(x-2).$$

Clearly, this is a simple Gaussian mixture model.

Part (b)

Use iterated expectation and variance to find $\mathrm{E}(X)$ and $\mathrm{Var}(X)$ exactly.

Using iterated expectation, we obtain

$$\begin{aligned}
\mathrm{E}(X) &= \mathrm{E}_Y(\mathrm{E}(X|Y)) && (1)\\
&= \mathrm{E}(X|y=0) f_Y(0) + \mathrm{E}(X|y=1) f_Y(1) && (2)\\
&= (-2)(0.3) + (2)0.7 && (3)\\
&= 0.8. && (4)
\end{aligned}$$

Using iterated variance, we obtain

$$\begin{aligned}
\mathrm{Var}(X) &= \mathrm{E}_Y(\mathrm{Var}(X|Y)) + \mathrm{Var}_Y(\mathrm{E}(X|Y)) && (5)\\
&= \mathrm{E}_Y(1) + \mathrm{Var}(\mu_Y) && (6)\\
&= 1 + \mathrm{E}_Y(\mu_Y^2) - \mathrm{E}_Y^2(\mu_Y) && (7)\\
&= 1 + \mu_0^2(1-p) + \mu_1^2 p - \mu^2 && (8)\\
&= 1 + 4(0.3) + 4(0.7) - 0.8^2 && (9)\\
&= 4.36 && (10)\\
& && (11)
\end{aligned}$$

Part (c)

$\boxed{\text{Obtain a Monte Carlo sample of size } m = 10000. \text{ Use this sample to compute (i) E}(X), \text{(ii) Var}(X), \text{(iii) 90-th percentile of } X.}$

First, we perform a MC simulation to obtain a sample from $\{X_m\}$:

```r
m <- 10000
p <- 0.7
mu_0 <- -2
mu_1 <- 2
xs <- vector(length = m)
ys <- rbinom(m, 1, p)
for (i in 1:m) {
    if (ys[i] == 0)
        xs[i] <- rnorm(1, mu_0) else xs[i] <- rnorm(1, mu_1)
}
```

Now, we just apply the necessary statistics to the sample to estimate the parameters.

Part (i)bb

The parameter μ is estimated to be:

```r
mean(xs)
```

```
## [1] 0.8008948
```

Part (ii)

The parameter σ^2 is estimated to be:

```r
var(xs)
```

```
## [1] 4.421046
```

Part (iii)

The 90% percentile is estimated to be:

```r
quantile(xs, c(0.9))
```

```
##      90%
## 3.090646
```