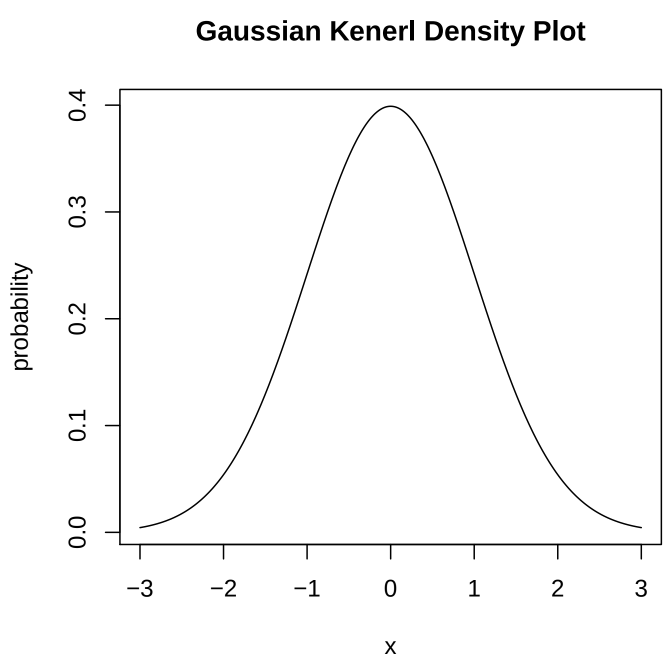
PH240C Assignment 4

Xiangyu Hu

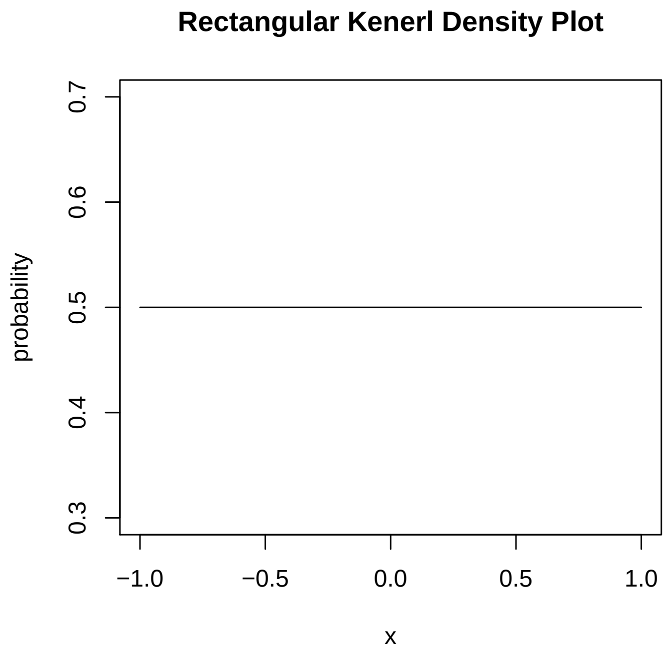
November 20, 2014

Problem 1. R implementation

```
# kernel density estimator
kde = function(x, data, k, h){
  k.para = (x-data)/h
  k.results = sapply(k.para, k)
  estimator = mean(k.results/h)
  return(estimator)
}

gaussian = dnorm
rectangular = function(x){
  0.5*as.numeric(abs(x)<=1)
}

# plot kernel densities
x1 = seq(-3,3,length.out=500)
x2 = seq(-1,1,length.out=100)
plot(gaussian(x1)~x1, type="l", main="Gaussian Kenerl Density Plot", xlab="x", ylab="probability")
```




```
plot(rectangular(x2)~x2, type="l", main="Rectangular Kenerl Density Plot", xlab="x", ylab="probability")
```



Problem 2. Simulation

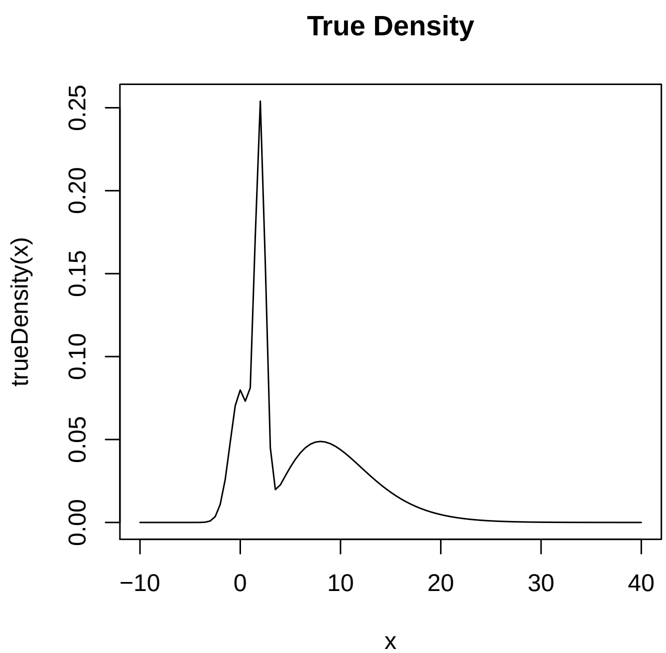
Let $X_1 \sim N(0,1), X_2 \sim N(2, 0.25), X_3 \sim X^2(10), X \sim P$, which P is defined in the problem.

$$
\begin{aligned}
E(X) &= \int_a^b x f_x dx \\
&= \int_a^b x(0.2f_{x_1} + 0.3f_{x_2} + 0.5f_{x_3})dx \\
&= 0.2\int_a^b x f_{x_1} dx + 0.3\int_a^b x f_{x_2} dx + 0.5\int_a^b x f_{x_3} dx \\
&= 0.2E(X_1) + 0.3E(X_2) + 0.5E(X_3) \\
&= 0.2*0 + 0.3*2 + 0.5*10 \\
&= 5.6
\end{aligned}
$$

$$
\begin{aligned}
Var(X) &= E(X^2) - (E(X))^2 \\
&= \int_a^b x^2 f_x dx - \mu^2 \\
&= \int_a^b x^2(0.2f_{x_1} + 0.3f_{x_2} + 0.5f_{x_3})dx - 5.6^2 \\
&= 0.2\int_a^b x^2 f_{x_1} dx + 0.3\int_a^b x^2 f_{x_2} dx + 0.5\int_a^b x^2 f_{x_3} dx - 5.6^2 \\
&= 0.2\int_a^b x^2 f_{x_1} dx + 0.3\int_a^b x^2 f_{x_2} dx + 0.5\int_a^b x^2 f_{x_3} dx \\
&- 0.2\mu_1^2 - 0.3\mu_2^2 - 0.5\mu_3^2 + 0.2\mu_1^2 + 0.3\mu_2^2 + 0.5\mu_3^2 - 5.6^2 \\
&= 0.2Var(X_1) + 0.3Var(X_2) + 0.5Var(X_3) + 0.2\mu_1^2 + 0.3\mu_2^2 + 0.5\mu_3^2 - 5.6^2 \\
&= 0.2*1 + 0.3*0.25 + 0.5*20 + 0.2*0 + 0.3*4 + 0.5*100 - 5.6^2 \\
&= 30.115
\end{aligned}
$$

```
# plot the density function
trueDensity = function(x){
  return (0.2*dnorm(x,0,sd=1) + 0.3*dnorm(x, 2, sd=0.5) + 0.5*dchisq(x,10))
```

```
}

curve(trueDensity, from = -10, to = 40, main="True Density")
```

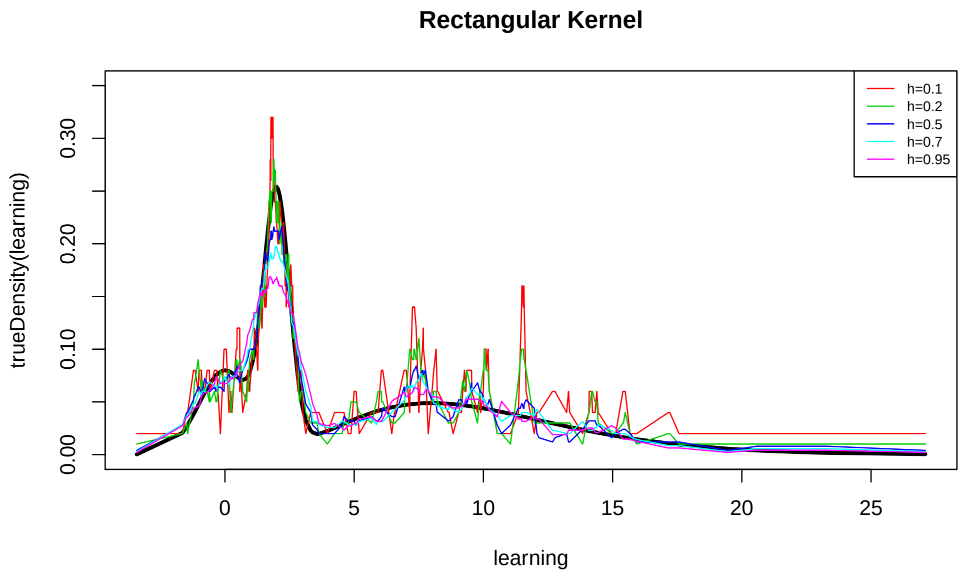



```
# simulate a learning set from the mixture distribution
set.seed(240)
randU = runif(250)
simu = numeric()
for (i in 1:length(randU)){
  if (randU[i] <= 0.2){
    simu[i] = rnorm(1)
  }
  else if (randU[i] <=0.5){
    simu[i] = rnorm(1, 2, sd = 0.5)
  }
  else{
    simu[i] = rchisq(1,10)
  }
}

learning = sort(simu)
```

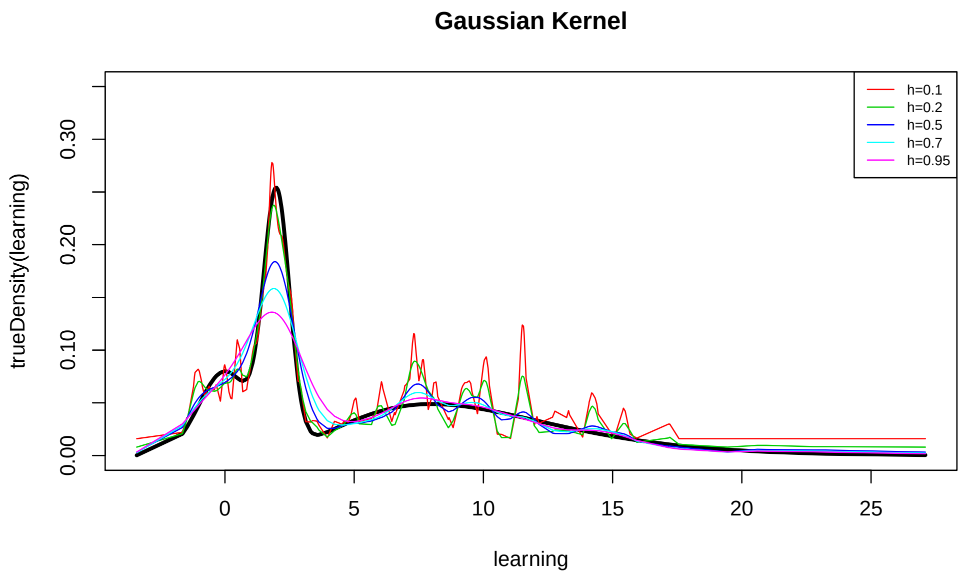
Problem 3. Density Estimation

```
h = c(0.1,0.2,0.5,0.7,0.95)

# rectangular kernel
K.est.R = matrix(NA, nrow=length(learning), ncol = length(h))
for (i in 1:length(h)){
  K.est.R[,i] = sapply(learning, kde, data=learning, k = rectangular, h=h[i])
}

# Gaussian kernel
K.est.G = matrix(NA, nrow=length(learning), ncol = length(h))
for (i in 1:length(h)){
```

```
  K.est.G[,i] = sapply(learning, kde, data=learning, k = gaussian, h=h[i])
}

plot(trueDensity(learning)~learning, type="l", lwd=3, ylim=c(0.0, 0.35), main="Rectangular Kernel")
for (i in 1:5){
  lines(K.est.R[,i] ~ learning, col=i+1)
}
legend("topright", c("h=0.1","h=0.2", "h=0.5", "h=0.7", "h=0.95"), col=2:6, lty=1,cex=0.7)
```




```
plot(trueDensity(learning)~learning, type="l", lwd=3, ylim=c(0.0, 0.35), main="Gaussian Kernel")
for (i in 1:5){
  lines(K.est.G[,i] ~ learning, col=i+1)
}
legend("topright", c("h=0.1","h=0.2", "h=0.5", "h=0.7", "h=0.95"), col=2:6, lty=1,cex=0.7)
```



Comment: The kernel density estimates become smoother when the bandwidth(h) is bigger. For rectangular kernel, I think the optimal bandwidth is somewhere between 0.2 and 0.5. On the plot, these two bandwidths correspond to the green and blue lines. The green line($h = 0.2$) does a better job of estimation at the peak area($x \in (0, 5)$). The blue line($h = 0.5$) underestimates the density at the peak area. However, in the flatter area($x \in (5, 15)$), the blue line does a better job of estimation. The green line has bigger variability. For gaussian kernel, we can see similar results. Overall, I would conclude that the optimal bandwidths is between 0.2 and 0.5. We can use cross-validation to find the optimal one.

Problem 4. Cross-Validation

```r
# Reparameterization of kernel density estimator

kde.sqr = function(x, data, k, h){
  k.para = (x-data)/h
  k.results = sapply(k.para, k)
  estimator = mean(k.results/h)
  sqr = estimator*estimator
  return(sqr)
}

# least squares cross validation proposed by Rudemo (1982) and Bowman (1984)
# leave-one out cross validation kernel density estimate
leave1out = function(h,k,data){
  n = length(data)
  k.est1out = rep(NA,n)
  for (i in 1:n){
    k.est1out[i] = kde(data[i], data[-i],k,h)
  }
  return(k.est1out)
}


# least squares cross validation
CV = function(h, data, k){
  # first term
  firstTerm = integrate(Vectorize(kde.sqr, vectorize.args = "x"), -Inf, Inf, data=learning,k=gaussian, h=h)

  k.est1out = leave1out(h, gaussian,learning)
  n = length(data)
  cv.result = firstTerm$value - (2/n)*sum(k.est1out)

  return(cv.result)
}

# optimize over the CV function to find the minimizer
optimize(CV, interval = c(-20,40), data=learning, k=gaussian)

## $minimum
## [1] 0.3616
##
## $objective
## [1] -0.08277
```

The optimal bandwidth from cross-validation using ISE is 0.36159, which is between 0.2 and 0.5 as we conclude in the previous part.