Homework 4

Emmie Jenkins and Bradley Thompson

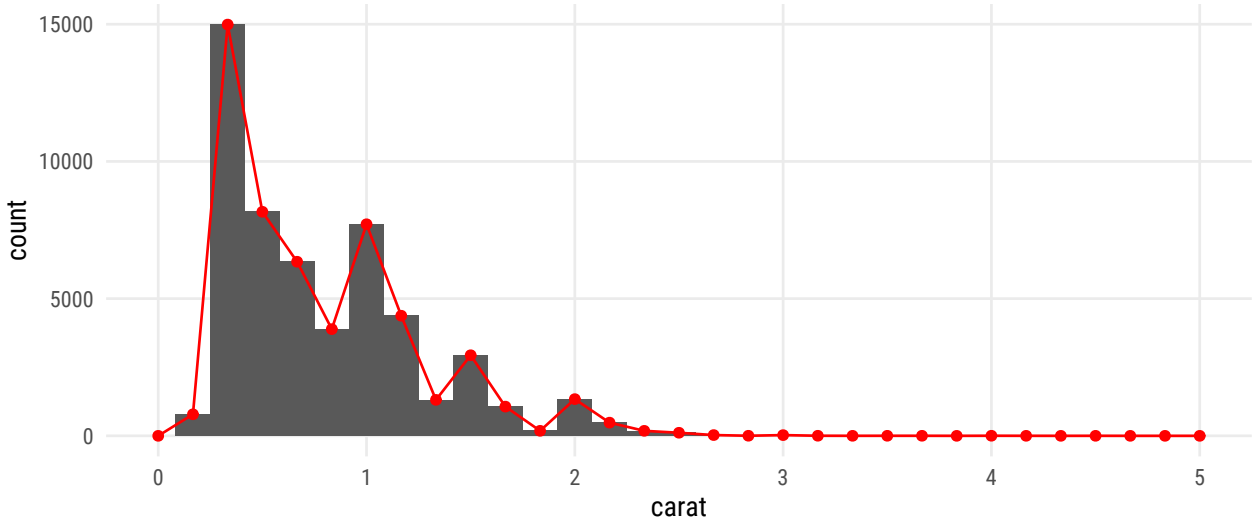
10/25/2020

a)

```
library("tidyverse")
library("extrafont")
library("viridis")
library("MASS")
library("patchwork")

#theme_set(theme_minimal())
theme_set(theme_minimal(base_family = "Roboto Condensed"))
#theme_update(panel.grid.minor = element_blank())

ggplot(diamonds) +
  geom_histogram(aes(carat), binwidth = 1/6) +
  geom_freqpoly(aes(carat), color="red", binwidth=1/6) +
  geom_point(stat="bin", aes(carat), binwidth=1/6, color="red")+
  scale_y_continuous(minor_breaks = NULL)+
  scale_x_continuous(minor_breaks = NULL,
                     limits = c(0,5))
```



b)

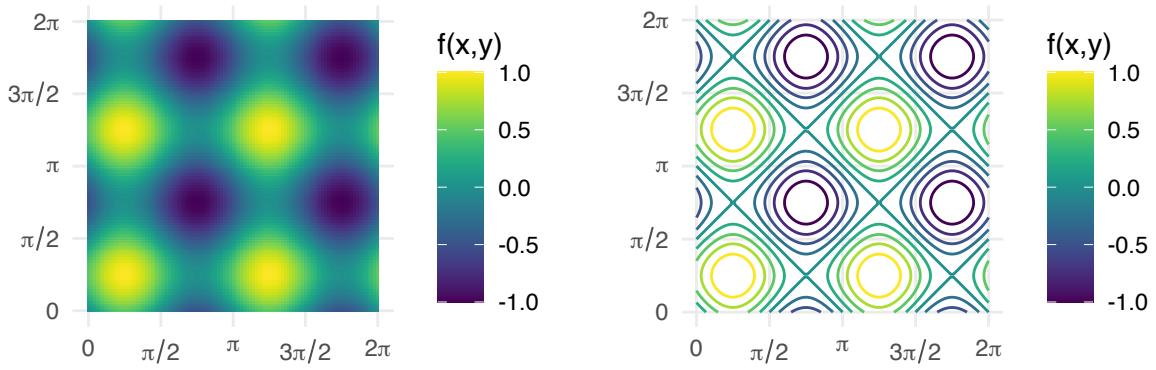
```
n <- 101
df <- expand.grid(
  "x" = seq(0, 2*pi, length.out = n),
  "y" = seq(0, 2*pi, length.out = n)
)
```

```
df <- df %>%
  mutate("fuzz" = (sin(2*x) + sin(2*y)) / 2)

a <- ggplot(df, aes(x, y, fill = fuzz))+
  geom_tile()+
  theme_minimal()+
  scale_fill_viridis(limits = c(-1,1), breaks = c(-1, -0.5, 0, 0.5, 1))+
  scale_x_continuous(name = "",
                     breaks = seq(from = 0, to = 2*pi, by = pi/2),
                     minor_breaks = NULL,
    labels = expression(0, pi/2, pi, 3*pi/2, 2*pi))+
   scale_y_continuous(name = "",
                     breaks = seq(from = 0, to = 2*pi, by = pi/2),
                     minor_breaks = NULL,
    labels = expression(0, pi/2, pi, 3*pi/2, 2*pi))+
  labs(fill = "f(x,y)")+
  coord_equal()

b <- ggplot(df, aes(x, y, z = fuzz))+
  geom_contour(aes(color = stat(nlevel)))+
  theme_minimal()+
  scale_color_viridis(limits = c(-1,1), breaks = c(-1, -0.5, 0, 0.5, 1))+
  scale_x_continuous(name = "",
                     breaks = seq(from = 0, to = 2*pi, by = pi/2),
                     minor_breaks = NULL,
    labels = expression(0, pi/2, pi, 3*pi/2, 2*pi))+
   scale_y_continuous(name = "",
                     breaks = seq(from = 0, to = 2*pi, by = pi/2),
                     minor_breaks = NULL,
    labels = expression(0, pi/2, pi, 3*pi/2, 2*pi))+
  labs(color = "f(x,y)")+
  coord_equal()

a + b
```



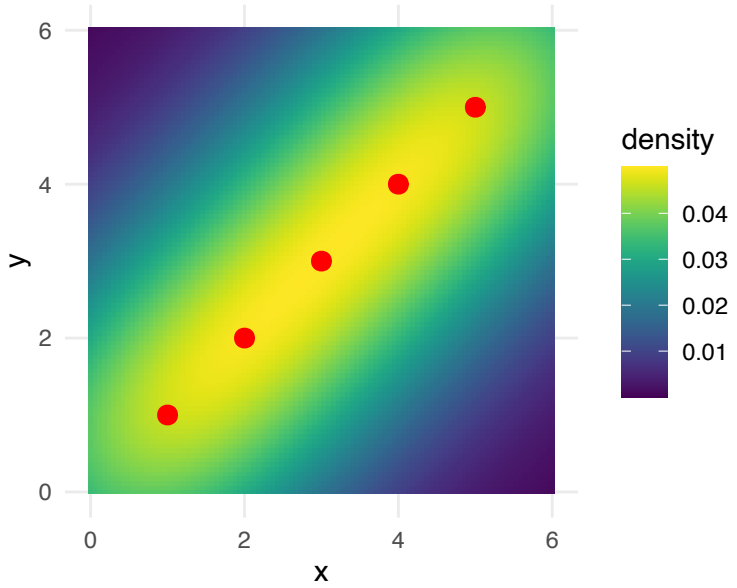
c)

```
d_df <- data.frame(x = 0:6, y = 0:6)

df <- data.frame(x = 1:5, y = 1:5)

dens_df <- data.frame(1:5, 1:5, kde2d(0:6, 0:6))
```

```r
ggplot()+
  stat_density2d(data = d_df,
    aes(x, y, fill = stat(ndensity)/20),
    geom = "raster",
    contour = FALSE
  )+
  scale_fill_viridis(limits = c(0.00001,.049999))+
  theme_minimal()+
  geom_point(data = df, aes(x,y), color = "red", size = 3)+
  coord_equal()+
  scale_x_continuous(
                     breaks = c(0,2,4,6),
                     minor_breaks = NULL,
    labels = c("0", "2", "4", "6"))+
   scale_y_continuous(
                     breaks = c(0,2,4,6),
                     minor_breaks = NULL,
    labels = c("0", "2", "4", "6"))+
  labs(fill="density")
```



d) What is the difference between cut_interval(),cut_number(), and cut_width()?

- cut_interval takes the given vector and cuts it into the given n number of parts, assuring that the range is the same for each of the intervals.
- cut_number takes a given vector and cuts it into n number of equal parts, although the range may not be the same for each of the groups.
- cut_width cuts the given vector into groups with equal range, but instead of being cut into n number of parts, it is cut into parts by a given width

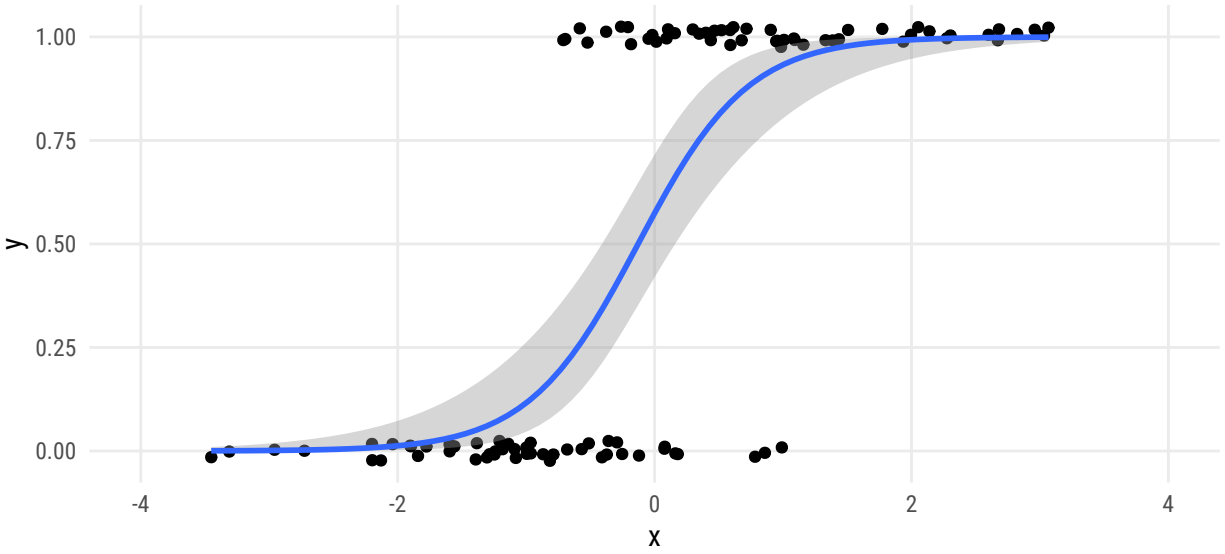
e)

```r
set.seed(2)
logistic <- function(x) 1 / (1 + exp(-x))
n <- 50
df <- tibble(
```

```
  x = c(rnorm(n, -1),rnorm(n, 1)),
  p = logistic(0 + 2*x),
  y = rbinom(2*n, 1, p)
  )

ggplot(df, aes(x, y)) +
  geom_jitter(height=.025) +
  stat_smooth(method = "glm", method.args=list(family = "binomial"))+
  scale_y_continuous(minor_breaks = NULL)+
  scale_x_continuous(minor_breaks = NULL, limits = c(-4,4))
```

```
## `geom_smooth()` using formula 'y ~ x'
```



2pre)

a)

```
make_positive <- function(x) sign(x[1]) * x

emax <- function(A, tol = sqrt(.Machine$double.eps)) {
  y <- c(1, rep(0, ncol(A)-1))
  new_norm <- 1
  old_norm <- 0
  while(abs(new_norm - old_norm) > tol){
    old_norm <- new_norm
    y <- A %*% y
    new_norm <- norm(y)
    y <- normalize(y)
    y <- make_positive(y)
  }

  list("value" = t(y) %*% A %*% y, "vector" = y)
}

#testing function
(A <- matrix(c(1:4, 6:10), nrow = 3, byrow = TRUE))
```

```
##      [,1] [,2] [,3]
```

```
## [1,]    1    2    3
## [2,]    4    6    7
## [3,]    8    9   10
```

```
eigen(A)
```

```
## eigen() decomposition
## $values
## [1] 18.0321458 -1.3251013  0.2929554
##
## $vectors
##            [,1]       [,2]       [,3]
## [1,] -0.2071014 -0.6238002  0.3599805
## [2,] -0.5424913 -0.3344309 -0.8132015
## [3,] -0.8141328  0.7064201  0.4572935
```

```
emax(A)
```

```
## $value
##          [,1]
## [1,] 18.03215
##
## $vector
##           [,1]
## [1,] 0.2071014
## [2,] 0.5424913
## [3,] 0.8141328
```

b)

```
make_first_row_positive <- function(A) apply(A, 2, make_positive)

eigen_qr <- function(A, tol = sqrt(.Machine$double.eps)){
  # initialize A_i and U_i
  A_i <- A
  U_i <- diag(ncol(A))

  while (TRUE) {
    # do the QR decomposition
    QR <- qr(A_i)
    Q_i <- qr.Q(QR)
    R_i <- qr.R(QR)

    # reorganize and update U
    U_i_1 <- U_i
    A_i <- R_i %*% Q_i
    U_i <- make_first_row_positive(U_i %*% Q_i)

    if (norm(U_i - U_i_1) <= tol) break
  }

  list("values" = diag(A_i), "vectors" = U_i)
}

#testing function
(A <- matrix(c(1:4, 6:10), nrow = 3, byrow = TRUE))
```

```
##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]    4    6    7
## [3,]    8    9   10
```

```
A <- crossprod(A)
eigen(A)
```

```
## eigen() decomposition
## $values
## [1] 357.91047441   2.02181118   0.06771441
##
## $vectors
##            [,1]       [,2]       [,3]
## [1,] -0.4715411  0.8359628  0.2807404
## [2,] -0.5813187 -0.0552796 -0.8117960
## [3,] -0.6631120 -0.5459948  0.5120274
```

```
eigen_qr(A)
```

```
## $values
## [1] 357.91047441   2.02181118   0.06771441
##
## $vectors
##           [,1]        [,2]       [,3]
## [1,] 0.4882518  0.82699823  0.2787188
## [2,] 0.5799121 -0.06878806 -0.8117698
## [3,] 0.6521597 -0.55798040  0.5131721
```

c)

```
det_qr <- function(A){
  x <- eigen_qr(A)
  y <- x$values
  z <- 1
  for(i in 1:length(y)){
    z <- z * y[i]
  }
  z
}

#testing
set.seed(2)
A <- matrix(rpois(64, lambda = 5), nrow = 8)
A <- crossprod(A)
det(A)
```

```
## [1] 220047556
```

```
det_qr(A)
```

```
## [1] 220047556
```