Carson Slater

6376 Homework 9 Solutions

```
source(here::here("helpers.R"))
```

Preliminary Citations

For the $\ell(r)$ function in number 6, Thomas was the one who shared the appropriate a value to render the most appropriate brightness for the plot.

Question 1

I have read the selected pages of the paper "***ggdensity***: *Improved Bivariate Density Visualization in R*."

For creating a 95% HDR for an IID random sample $x_1, \ldots, x_n$ from a continuous distribution, `ggdensity` computes the PDF over a mesh that spans a region larger than the support of the data, discretizes this PDF, and creates a data frame with the value of each respective observation, the value of the PDF at that observation, and the probability corresponding to that discretized region of PDF, p_i (area of rectangle for each x_i). `ggdensity` then orders the observations by p_i in descending order and takes the cumulative sum of the first k observations such that $\sum_{i=1}^{k} p_{(i)} = a_{(k)} \leq 0.95$. The HDR is then the set of all values $a_{(i)}$ such that $a_{(i)} \geq a_{(k)}$.

Question 2

I have read the first three pages of Trefethen and Bau[1].

Question 3

```
svd_crossprod <- function(A) {

  m <- nrow(A)
  n <- ncol(A)

  if (n > m) A <- t(A)
```

[1] Trefethen, Lloyd Nicholas, and David Bau. *Numerical Linear Algebra*. United States: Society for Industrial and Applied Mathematics, 1997.

```
  eigen_decomp <- mat_mat_prod(t(A), A) |> eigen()

  sigma <- eigen_decomp$values |>
    positize() |>
    sqrt()

  U <- mat_mat_prod(A, eigen_decomp$vectors) |>
    sweep(2, sqrt(eigen_decomp$values), FUN = "/")

  if (n > m) {
    list(
      "d" = sigma,
      "u" = eigen_decomp$vectors,
      "v" = U
    )
  } else {
    list(
      "d" = sigma,
      "u" = U,
      "v" = eigen_decomp$vectors
    )
  }
}
```

```
(A <- matrix(c(1:11,14), nrow = 4))
```

```
     [,1] [,2] [,3]
[1,]    1    5    9
[2,]    2    6   10
[3,]    3    7   11
[4,]    4    8   14
```

```
A |> svd_crossprod()
```

```
$d
[1] 26.4534110  1.3903935  0.5327779

$u
          [,1]       [,2]       [,3]
```

```
[1,] 0.3887329 -0.8016185  0.1990673
[2,] 0.4470412 -0.2622195 -0.2544179
[3,] 0.5053495  0.2771796 -0.7079030
[4,] 0.6274255  0.4602392  0.6281059

$v
          [,1]        [,2]       [,3]
[1,] 0.2006760  0.96838646  0.1481782
[2,] 0.4983388  0.02931657 -0.8664865
[3,] 0.8434379 -0.24772606  0.4767015
```

```
A |> svd_crossprod() |> with(u %*% diag(d) %*% t(v))
```

```
     [,1] [,2] [,3]
[1,]    1    5    9
[2,]    2    6   10
[3,]    3    7   11
[4,]    4    8   14
```

```
A |> t() |> svd_crossprod() |> with(u %*% diag(d) %*% t(v))
```

```
     [,1] [,2] [,3] [,4]
[1,]    1    2    3    4
[2,]    5    6    7    8
[3,]    9   10   11   14
```

```
all.equal( svd(A), svd_crossprod(A) )
```

```
[1] "Component \"u\": Mean relative difference: 0.7082003"
[2] "Component \"v\": Mean relative difference: 0.720899"
```

Question 4

```
# creates a square matrix
pad_matrix <- function(A){
  stopifnot(is.matrix(A))
```

```
  m <- nrow(A)
  n <- ncol(A)

  if (n == m) return(A)

  if (m > n) {
    for (i in seq_along(diff)) {
      mat <- diag(0, m)
      mat[1:m, 1:n] <- A
    }
  } else {
    for (i in seq_along(diff)) {
      mat <- diag(0, n)
      mat[1:m, 1:n] <- A
    }
  }
  if (nrow(mat) != ncol(mat)) stop("pad_matrix() failed to produce a square matrix")
  mat
}

# normalize columns function from assignment
normalize_columns <- function(A) A |> apply(2, normalize)

# svd_H function
svd_H <- function(A) {
  m <- nrow(A)
  n <- ncol(A)

  pad_A <- pad_matrix(A)

  n_pad <- ncol(pad_A)

  H <- diag(0, 2*n_pad)
  H[(2*n_pad - n_pad + 1):(2*n_pad), 1:n_pad] <- pad_A
  H[1:n_pad, (2*n_pad - n_pad + 1):(2*n_pad)] <- t(pad_A)

  eigen_decomp_H <- eigen(H)

  sigma <- eigen_decomp_H$values[1:ifelse(m > n, n, m)]

  if (m > n) {
```

```
    V <- eigen_decomp_H$vectors[1:n, 1:n] |>
      normalize_columns()
  } else {
    V <- eigen_decomp_H$vectors[1:n, 1:m] |>
      normalize_columns()
  }

  U <- mat_mat_prod(A, V) |>
    sweep(2, sqrt(sigma), FUN = "/") |>
    normalize_columns()

  list(
    "d" = sigma,
    "u" = U,
    "v" = V
  )
}
```

```
A |> svd_H()
```

```
$d
[1] 26.4534110  1.3903935  0.5327779

$u
           [,1]       [,2]       [,3]
[1,] -0.3887329 -0.8016185  0.1990673
[2,] -0.4470412 -0.2622195 -0.2544179
[3,] -0.5053495  0.2771796 -0.7079030
[4,] -0.6274255  0.4602392  0.6281059

$v
           [,1]        [,2]       [,3]
[1,] -0.2006760  0.96838646  0.1481782
[2,] -0.4983388  0.02931657 -0.8664865
[3,] -0.8434379 -0.24772606  0.4767015
```

```
A |> svd_H() |> with(u %*% diag(d) %*% t(v))
```

```
     [,1] [,2] [,3]
[1,]    1    5    9
```

```
[2,]    2    6   10
[3,]    3    7   11
[4,]    4    8   14
```

```
A |> t() |> svd_H() |> with(u %*% diag(d) %*% t(v))
```

```
     [,1] [,2] [,3] [,4]
[1,]    1    2    3    4
[2,]    5    6    7    8
[3,]    9   10   11   14
```

```
all.equal(svd(A), svd_H(A))
```

```
[1] TRUE
```

Question 5

(a)

```
# roots function
roots <- function(n) {
  stopifnot(is.numeric(n))

  # if input is not an integer,
  # it is coerced into an integer
  n <- as.integer(n)

  k <- 0:(n-1)
  x <- sapply(k, \(k) cos(k*(2*pi)/n))
  y <- sapply(k, \(k) sin(k*(2*pi)/n))
  z <- sapply(k, \(k) exp(k*2*pi*1i/n))

  data.frame(x, y, z, k, "n" = rep(n, length.out = n)) |>
    zapsmall()
}
```

```
# testing roots()
roots(2)
```

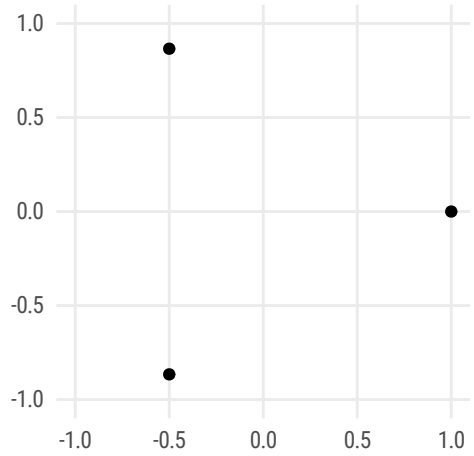
```
   x y     z k n
1  1 0  1+0i 0 2
2 -1 0 -1+0i 1 2
```

```
roots(5)
```

```
          x         y                   z k n
1  1.000000  0.000000  1.000000+0.000000i 0 5
2  0.309017  0.951057  0.309017+0.951057i 1 5
3 -0.809017  0.587785 -0.809017+0.587785i 2 5
4 -0.809017 -0.587785 -0.809017-0.587785i 3 5
5  0.309017 -0.951057  0.309017-0.951057i 4 5
```

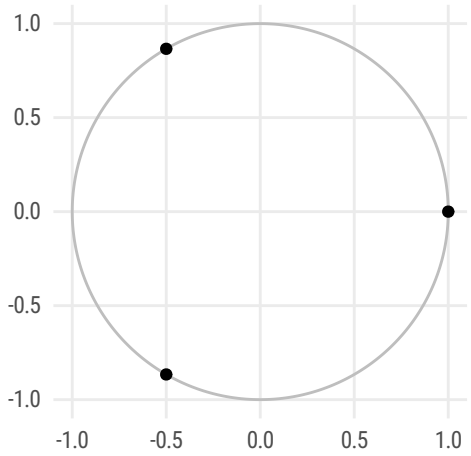
(b)

```
roots(3) |>
  ggplot(aes(x, y)) +
  geom_point() +
  coord_equal(xlim = c(-1,1), ylim = c(-1,1)) +
  theme(axis.title = element_blank())
```



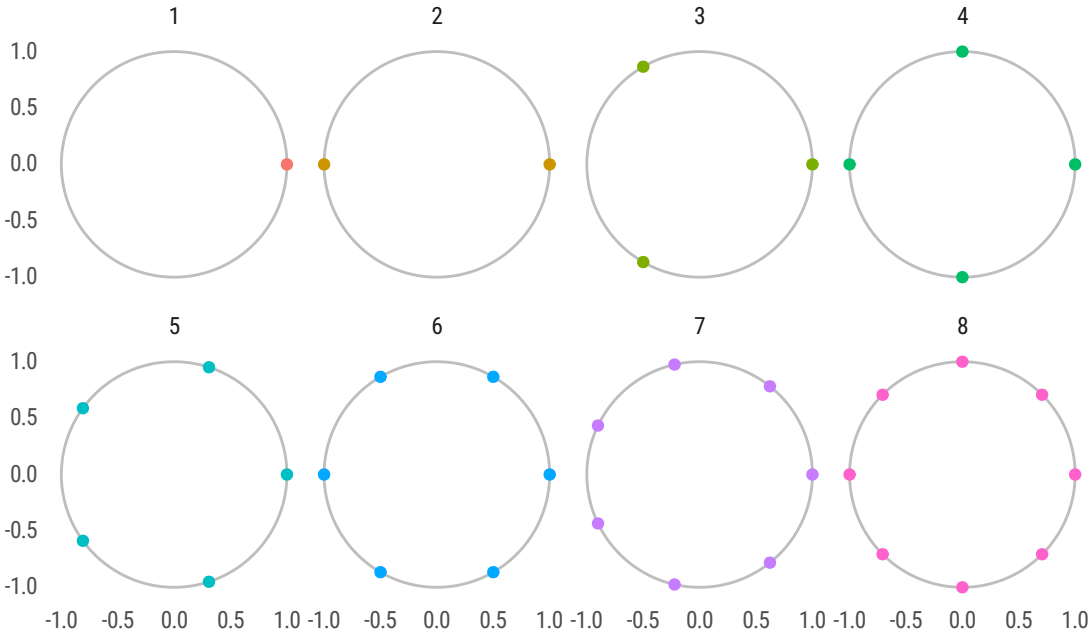
(c)

```
circle <- roots(1000) |> select(x, y)

roots(3) |>
  ggplot(aes(x, y)) +
  geom_path(data = circle, aes(x, y), color = "grey") +
  geom_point() +
  coord_equal(xlim = c(-1,1), ylim = c(-1,1)) +
  theme(axis.title = element_blank())
```



(d)

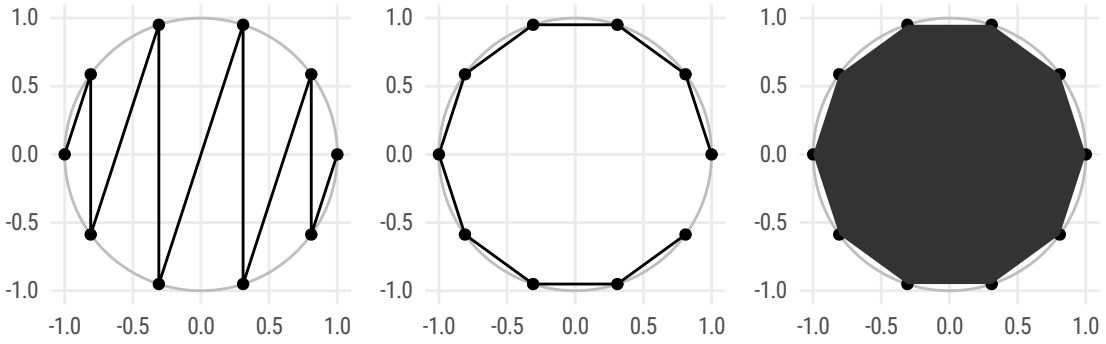
```
n <- 1:8
roots_graphic <- purrr::map_dfr(.x = n, .f = roots)

ggplot() +
  geom_path(data = circle, aes(x, y), color = "grey") +
  geom_point(data = roots_graphic, aes(x, y, color = as.factor(n))) +
  facet_wrap(. ~ n, nrow = 2) +
  coord_equal(xlim = c(-1,1), ylim = c(-1,1)) +
  theme(legend.position = "none",
        axis.title = element_blank(),
        panel.grid.major = element_blank(),
        panel.grid.minor = element_blank())
```



(e)

```
r10 <- roots(10)

p1 <- r10 |>
  ggplot(aes(x, y)) +
  geom_path(data = circle, aes(x, y), color = "grey") +
  geom_point() +
  geom_line() +
  coord_equal(xlim = c(-1,1), ylim = c(-1,1)) +
  theme(axis.title = element_blank())

p2 <- r10 |>
  ggplot(aes(x, y)) +
  geom_path(data = circle, aes(x, y), color = "grey") +
  geom_point() +
  geom_path() +
  coord_equal(xlim = c(-1,1), ylim = c(-1,1)) +
  theme(axis.title = element_blank())

p3 <- r10 |>
  ggplot(aes(x, y)) +
```

```
  geom_path(data = circle, aes(x, y), color = "grey") +
  geom_point() +
  geom_polygon() +
  coord_equal(xlim = c(-1,1), ylim = c(-1,1)) +
  theme(axis.title = element_blank())

p1 + p2 + p3
```



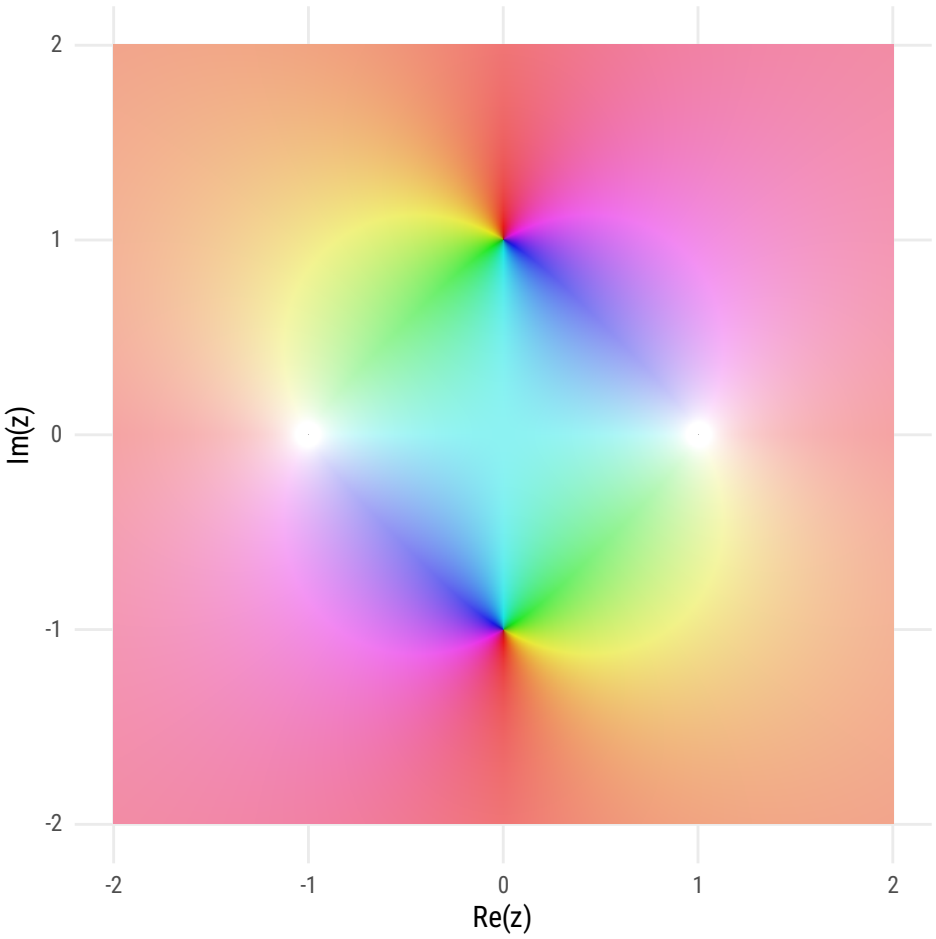
Here, we see that `geom_line()` (left) connects the points as the `x` aesthetic increases, whereas `geom_path()` (middle) connects the points in the order as they appear in the data frame being passed into `ggplot()`. `geom_polygon()` takes it one step further than `geom_path()` and connects the first and last points in the data frame passed to `ggplot()`, connects them, and fills in the shape created by the path geom.

Question 6

```
# functions
modulus <- function(z) Re(sqrt(z * Conj(z)))
f <- function(z) (z^2 + 1)/(z^2 - 1)
rad2deg <- function(rad, rotate = 0) ((rad + rotate) * 360/(2*pi)) %% 360

# definitions
# boundaries
L <- -2
U <- 2
# making mesh
domain <- seq(L, U, length.out = 1001)

mesh <- expand.grid(domain, domain) |>
```

```
  mutate("x" = Var1,
         "y" = Var2) |>
  select(x, y)

# color mapping components
z <- complex(real = mesh$x, imaginary = mesh$y)

fz <- z |> f()

mod_fz <- fz |> modulus()

# I ended using the second one from the Wikipedia page
# and I added a constant to shift the color, as well as
# a scale of 100 (a = 0.4 too)

a <- 0.4
ell1 <- (2/pi)*atan(mod_fz) + 65
ell2 <- (mod_fz^a/(mod_fz^a + 1))/10 + 65
ell3 <- 100*mod_fz^a/(mod_fz^a + 1) + 25 # used this one

# used saturation of 80
my_colors <- Arg(fz) |>
  rad2deg() |>
  cbind(80, ell3) |> farver::encode_colour(from = "hsl")

df <- cbind(mesh, my_colors) |>
  as.data.frame()
```

```
# plot replica attempt
df |>
  ggplot(aes(x, y)) +
  geom_raster(aes(fill = my_colors)) +
  labs(x = "Re(z)", y = "Im(z)") +
  scale_fill_identity() +
  coord_equal(xlim = c(-2,2), ylim = c(-2,2))
```


2
1
0
-1
-2
Im(z)
-2
-1
0
1
2
Re(z)