Carson Slater

6376 Homework 10 Solutions

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

Preliminary Citations

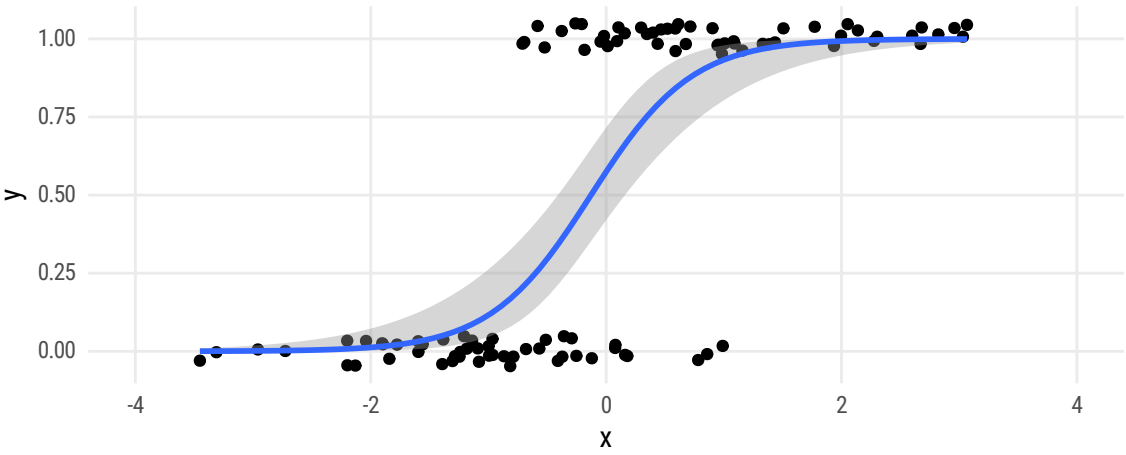
For question 2(d), I consulted ChatGPT 3.5 to check the linearity of the inner product. For questions 2(e) and 3(d), our cohort worked through it on a whiteboard and came to some group consensuses.

Question 1

```
set.seed(2)
logistic <- binomial()$linkinv
n <- 50
df <- tibble(x = c(rnorm(n, -1), rnorm(n, 1)),
             p = logistic(0 + 2*x),
             y = rbinom(2*n, 1, p)
             )
```

```
df |> ggplot(aes(x = x, y = y)) +
  geom_jitter(height = 0.05) +
  geom_smooth(method = "glm",
              method.args = list(
              family = binomial(link = "logit"))
              ) +
  xlim(c(-4,4))
```

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



Question 2

(a)

The elements of $\mathbf{b}$ represent coefficients of expansion in the standard basis $\mathbb{C}^m$. When each value is a complex number, it serves as a coefficient that tells how much of each corresponding standard basis vector contributes to the vector $\mathbf{b}$'s representation.

(b)

Each element in $\mathbf{x}$ is the coefficient indicating how much the column vectors of $\mathbf{A}$ must be scaled to attain the vector $\mathbf{b}$.

(c)

In $\mathbf{b} = \mathbf{A}\mathbf{x}$, $\mathbf{b}$ represents the results of applying a linear transformation represented by $\mathbf{A}$ on the vector $\mathbf{x}$. Assuming $\mathbf{A}^{-1}$ exists, we can solve for the vector $\mathbf{x}$ by right multiplying $\mathbf{A}^{-1}$ on both sides. So each element of the vector $\mathbf{b}$ represents the scalar complex multiple of the corresponding row of $\mathbf{A}^{-1}$ to the elements of $\mathbf{x}$.

(d)

Considering $\mathbf{b}$ as a linear expansion of the column vectors $\mathbf{a}_1, \ldots, \mathbf{a}_n$, we have,

$$\mathbf{b} = \mathbf{a}_1 x_1 + \mathbf{a}_2 x_2 + \ldots + \mathbf{a}_n x_n,$$

Taking the inner product of both sides with $\mathbf{a}_j$ yields

$$\langle \mathbf{a}_j, \mathbf{b} \rangle = \langle \mathbf{a}_j,\ \mathbf{a}_1 x_1 + \mathbf{a}_2 x_2 + ... + \mathbf{a}_n x_n \rangle.$$

By the linearity of the inner product, we have

$$\langle \mathbf{a}_j, \mathbf{b} \rangle = x_1 \langle \mathbf{a}_j,\ \mathbf{a}_1 \rangle + x_2 \langle \mathbf{a}_j,\ \mathbf{a}_2 \rangle + ... + x_n \langle \mathbf{a}_j,\ \mathbf{a}_n \rangle.$$

If the column vectors $\mathbf{a}_1, \dots, \mathbf{a}_n$ are orthogonal, then $x_i \langle \mathbf{a}_j,\ \mathbf{a}_i \rangle = 0$ if $i \neq j$. When $i = j$, then $\langle \mathbf{a}_j,\ \mathbf{a}_i \rangle$ is the inner product of $\mathbf{a}_j$ with itself, which is 1 (due to the inner product being the squared distance of an orthogonal vector). This would reduce the inner product of $\mathbf{a}_j$ and $\mathbf{b}$ to

$$\langle \mathbf{a}_j, \mathbf{b} \rangle = x_j \langle \mathbf{a}_j, \mathbf{a}_j \rangle.$$

This yields the result where x_j is now trivial,

$$x_j = \langle \mathbf{a}_j, \mathbf{b} \rangle.$$

(e)

Suppose there exists matrix $\mathbf{A} = \mathbf{A}^{-1}$, where

$$\mathbf{A} = \begin{bmatrix} a & b \\ c & d \end{bmatrix}.$$

Then $\mathbf{AA} = \mathbf{I}_2$, so

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} a & b \\ c & d \end{bmatrix} = \begin{bmatrix} a^2 + bc & ab + bd \\ ac + cd & bc + d^2 \end{bmatrix} = \mathbf{I}_2.$$

This implies that

$$\begin{bmatrix} a^2 + bc & b(a + d) \\ c(a + d) & bc + d^2 \end{bmatrix} = \mathbf{I}_2.$$

Here we can see that if $(a + d) = 0$, then b or c does not necessarily need to equal zero for the equality to hold. So then $\mathbf{A}$ does not necessarily equal $\pm\mathbf{I}_2$.

Question 3

(a)

```
Fn <- function(n, scale = FALSE) {
  omega <- roots(n)$z
  mat <- matrix(rep(0, times = n^2), nrow = n)
  for (i in 1:(n)) mat[, i] <- omega^(n - i + 1)
  if (scale == TRUE) mat/sqrt(n)
  else mat
}
```

```
Fn(4)
```

```
     [,1]  [,2]  [,3]  [,4]
[1,] 1+0i  1+0i  1+0i  1+0i
[2,] 1+0i  0-1i -1+0i  0+1i
[3,] 1+0i -1+0i  1+0i -1+0i
[4,] 1+0i  0+1i -1+0i  0-1i
```

(b)

```
dft <- function(x, scale = FALSE) {
  stopifnot(is.atomic(x))
  x %*% Fn(length(x), scale = scale) |> as.vector()
}
```

```
x <- 1:4
x |> dft()
```

```
[1] 10+0i -2+2i -2+0i -2-2i
```

```
x |> fft()
```

```
[1] 10+0i -2+2i -2+0i -2-2i
```

(c)

```
iFn <- function(n, scale = FALSE) {
  apply(Fn(n, scale = scale), MARGIN = 2, FUN = \(col) col^(-1))/n
}
```

```
iFn(4)
```

```
        [,1]        [,2]     [,3]        [,4]
[1,] 0.25+0i  0.25+0.00i  0.25+0i  0.25+0.00i
[2,] 0.25+0i  0.00+0.25i -0.25+0i  0.00-0.25i
[3,] 0.25+0i -0.25+0.00i  0.25+0i -0.25+0.00i
[4,] 0.25+0i  0.00-0.25i -0.25+0i  0.00+0.25i
```

```
(iFn(4) %*% Fn(4)) |> zapsmall()
```

```
     [,1] [,2] [,3] [,4]
[1,] 1+0i 0+0i 0+0i 0+0i
[2,] 0+0i 1+0i 0+0i 0+0i
[3,] 0+0i 0+0i 1+0i 0+0i
[4,] 0+0i 0+0i 0+0i 1+0i
```

(d)

The elements of the DFT $\hat{x}$ are the coefficients of expansion of x in the (*not inverse*) unity basis $\mathbf{F}_n^{-1}$.

(e)

```
idft <- function(xhat, scale = FALSE) {
  stopifnot(is.atomic(x))
  xhat %*% iFn(length(xhat), scale = scale) |> as.vector()
}
```

```
x |> dft() |> idft()
```

```
[1] 1+0i 2+0i 3+0i 4+0i
```

```
x |> fft() |> fft(inverse = TRUE) / length(x)
```

```
[1] 1+0i 2+0i 3+0i 4+0i
```

(f)

```
threshold <- function(x, thresh) {
   for (i in seq_along(x)) if (Mod(x[i]) < thresh) x[i] <- 0
   x
}

denoise <- function(x, thresh = 10) {
```

```
  dft(x) |>
    threshold(thresh = thresh) |>
    idft() |>
    Re() |>
    as.vector()
}
```

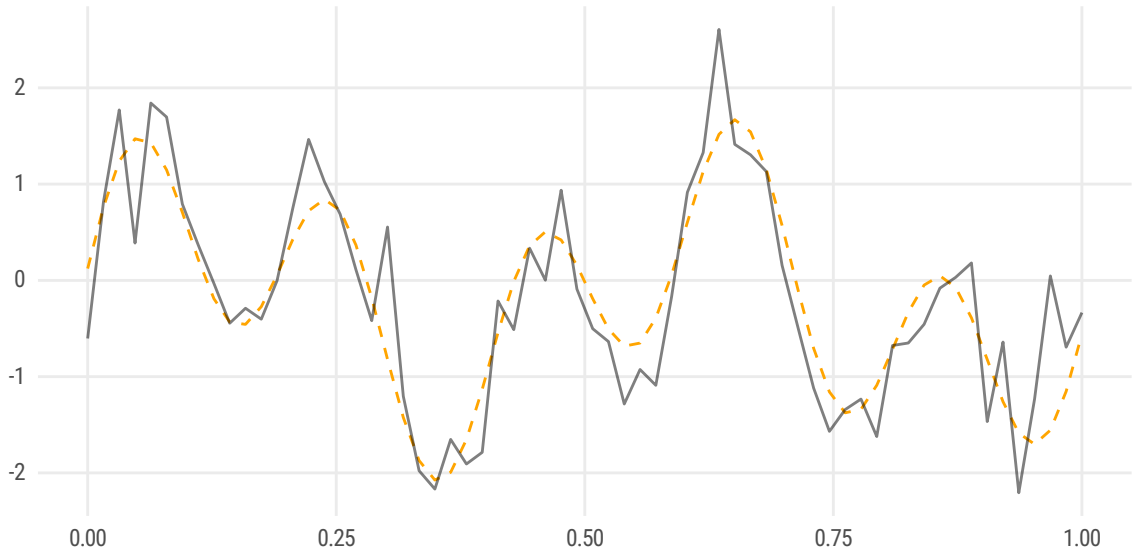
```
set.seed(1234)
# set simulation parameters
n <- 2^6
L <- 1
f <- function(x) sin(2 * (2*pi/L)*x) + sin(5 * (2*pi/L)*x)
si <- .50

# make simulation dataset
df <- tibble(
  "x" = seq(0, L, length.out = n),
  "y" = f(x) + rnorm(n, sd = si)
)

# denoise column
df <- df |> mutate("denoisey" = denoise(y))

# denoised plot
p_denoise_sig <- ggplot(df) +
  geom_line(aes(x, denoisey),
            color = "orange",
            linetype = 2) +
  geom_line(aes(x, y), alpha = .50)

p_denoise_sig & theme(axis.title = element_blank())
```



Question 4

(a)

I have reviewed the `read_rds()` and `write_rds()` functions.

(b)

```
# reading in the data
samps <- read_rds(here::here("Homework_10", "stanfit.rds"))

print(samps)
```

```
Inference for Stan model: 805f8805505f9e89d620dd65a9908e2e.
8 chains, each with iter=375; warmup=125; thin=1;
post-warmup draws per chain=250, total post-warmup draws=2000.

       mean se_mean   sd  2.5%   25%   50%  75% 97.5% n_eff Rhat
x     -0.14    0.19 0.77 -1.14 -0.92 -0.17 0.57  1.13    16 1.53
y      0.29    0.07 0.74 -0.93 -0.37  0.28 1.07  1.24   119 1.06
num    0.00    0.00 0.02 -0.04  0.00  0.00 0.01  0.06  1094 1.00
denom  1.63    0.12 1.75  0.01  0.24  0.90 2.87  5.44   230 1.03
ng     0.00    0.00 0.01 -0.02 -0.01  0.00 0.01  0.02  1302 1.00
lp__   3.13    0.03 0.79  0.87  2.94  3.42 3.63  3.69   678 1.01
```

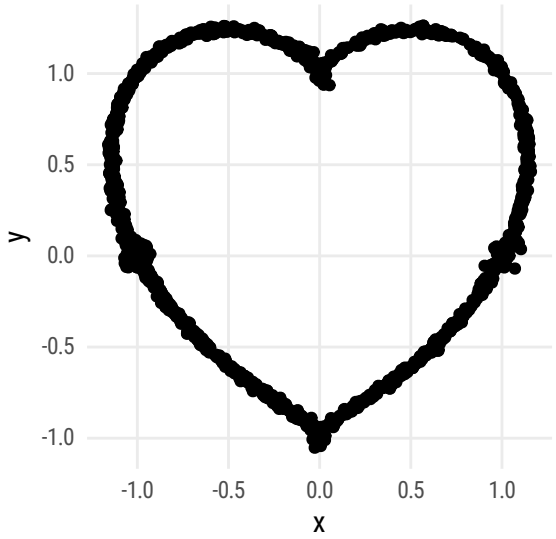
```
Samples were drawn using NUTS(diag_e) at Wed Mar 27 21:05:53 2019.
For each parameter, n_eff is a crude measure of effective sample size,
and Rhat is the potential scale reduction factor on split chains (at
convergence, Rhat=1).
```

(c)

```
samps_df <- samps |>
  array_branch(1) |>
  imap_dfr(.f = ~as_tibble(.x) |>
             mutate(chain = row_number())) |>
  mutate("iteration" = rep(1:250, each = 8) |> as.factor(),
         "chain" = as.factor(chain))
```

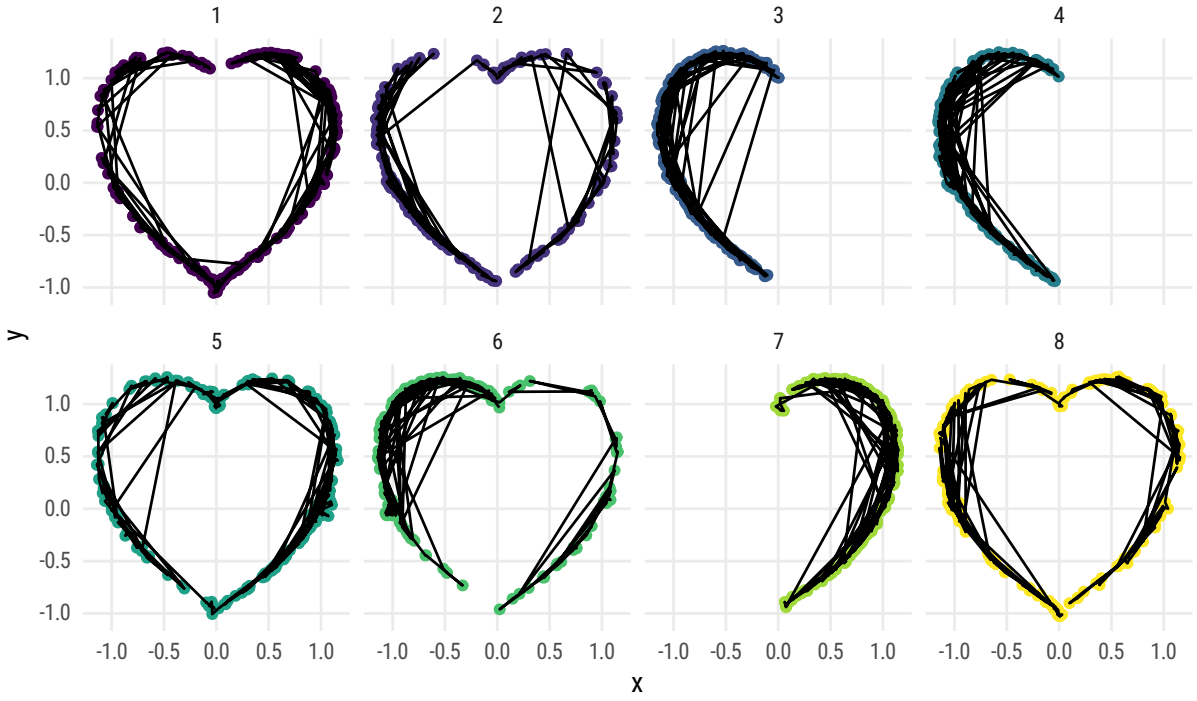
(d)

```
samps_df |> ggplot(aes(x, y)) +
  geom_point() +
  coord_equal()
```



(e)

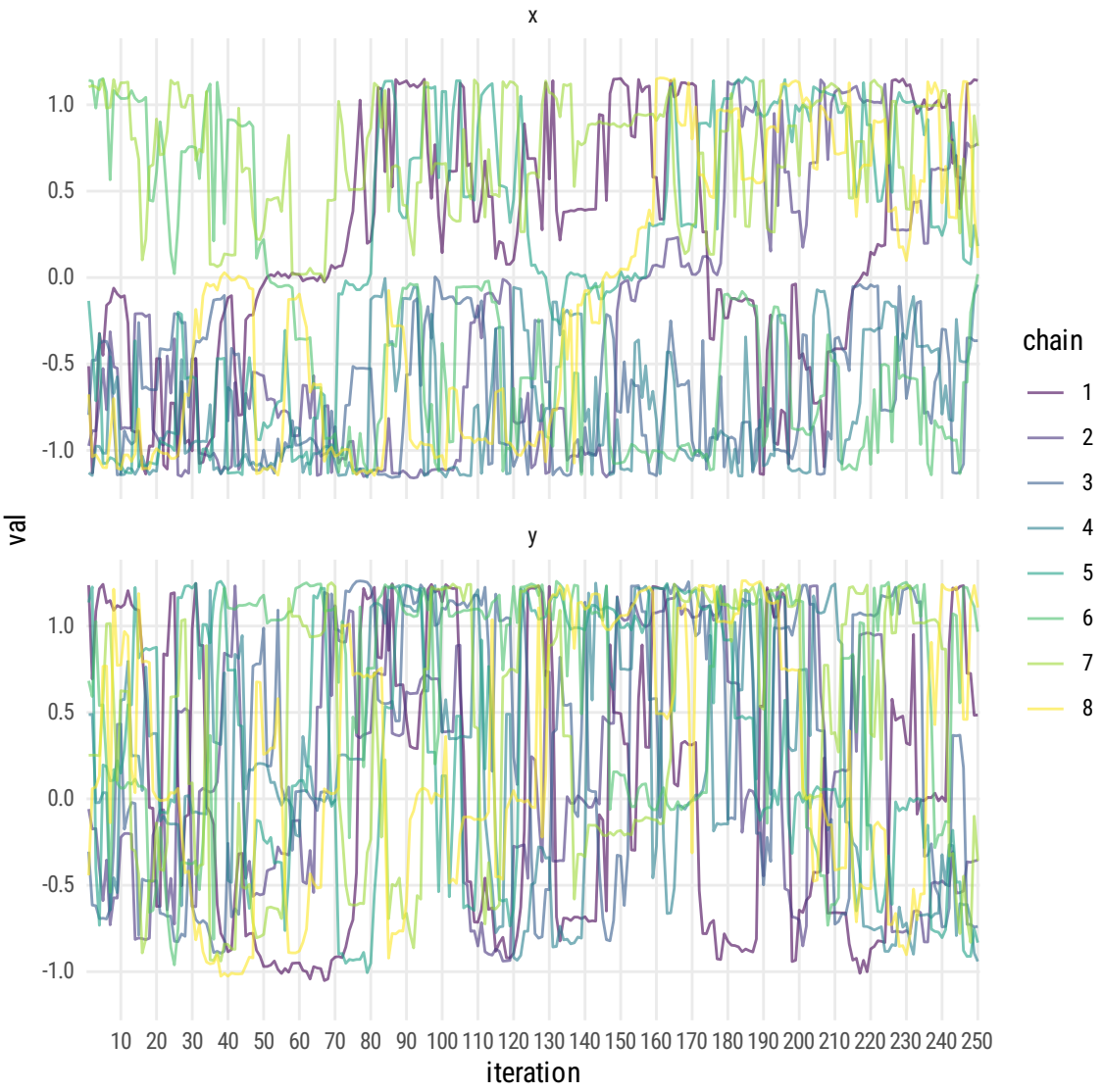
```
samps_df |>
  ggplot(aes(x, y)) +
    geom_point(aes(col = chain)) +
    geom_path() +
    facet_wrap(chain ~ ., nrow = 2) +
    scale_color_viridis_d() +
    coord_equal() +
    theme(legend.position = "none")
```



(f)

```
samps_df |> select(x, y, iteration, chain) |>
  tidyr::pivot_longer(cols = x:y, names_to = "var", values_to = "val") |>
  ggplot() +
    geom_line(aes(iteration, val,
                  group = chain,
```

```
                color = chain),
                alpha = 0.6) +
  facet_wrap(var ~ ., nrow = 2) +
  scale_color_viridis_d() +
  scale_x_discrete(labels = seq(0, 251, 10),
                   breaks = seq(0, 251, 10))
```



(g)

```
chains <- paste0("Chain ", seq(from = 1, to = 8))

samps_df |>
  group_by(chain) |>
  group_map(\(df, key) list("Mean X" = mean(df$x) |> round(3),
                            "Mean Y" = mean(df$y) |> round(3))) |>
  sapply(as.data.frame) |> knitr::kable(col.names = chains)
```

	Chain 1	Chain 2	Chain 3	Chain 4	Chain 5	Chain 6	Chain 7	Chain 8
Mean.X	0.195	-0.281	-0.678	-0.758	0.182	-0.35	0.707	-0.134
Mean.Y	0.014	0.153	0.364	0.161	0.325	0.673	0.36	0.256

```
  # technically cheating but I really liked the output in the rendering
```